# Message Handling Unit Architecture

## version 3.0

| | |
|---|---|
| Document number | ARM-AES-0072 |
| Document quality | eac |
| Document version | A.a |
| Document confidentiality | Non-confidential |

# Message Handling Unit Architecture

## Release information

| Date | Version | Changes |
|---|---|---|
| 2024/Mar/11 | A.a | • Initial release of MHU v3.0 architecture. |

## Non-Confidential Proprietary Notice

# Contents

# Message Handling Unit Architecture

## Preface

## Part A    Message Handling Unit Architecture Overview

## Part B    Message Handling Unit v3.0

## Part C   Programmers Model

# Part D  Appendix

# Glossary

# Preface

This preface introduces the Message Handling Unit Architecture Specification. It contains the following sections:

- About this specification.
- Conventions.
- Additional reading.
- Feedback.

# About this specification

This specification describes the Message Handling Unit(MHU) architecture version 3.0.

Throughout this specification, references to *the MHU* or *a MHU* refer to a device that implements the MHUv3.0 architecture.

## Intended audience

This architecture is for users who want to design, implement or program the MHU in a range of Arm-compliant implementations of the MHUv3. It does not assume familiarity with previous versions of the MHU.

The architecture assumes that that users have some experience of Arm products, and are familiar with the terminology that describes the Arm A-profile architecture.

# Conventions

## Typographical conventions

The typographical conventions are:

*italic*

> Introduces special terminology, and denotes citations.

**bold**

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

`monospace`

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

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

SMALL CAPITALS

> Used for some common terms such as IMPLEMENTATION DEFINED.

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

Red text

> Indicates an open issue.

Blue text

> Indicates a link. This can be

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

## Numbers

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

## Pseudocode descriptions

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

## Assembler syntax descriptions

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

# Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

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

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

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

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

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

## Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

## Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: $L_{iiiii}$

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

## Content item classes

### Declaration

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

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

A Declaration does not describe behavior.

A Declaration is rendered with the label *D*.

### Rule

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

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

### Information

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

An Information statement is rendered with the label *I*.

### Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

### Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

### Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

# Additional reading

This section lists publications by Arm and by third parties.

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

[1] *AMBA® Low Power Interface Specification*. (ARM IHI 0068) Arm.

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

[3] *Arm® CoreSight Architecture Specification v3*. (ARM IHI 0029) Arm.

# Feedback

Arm welcomes feedback on its documentation.

## Feedback on this book

If you have comments on the content of this book, create a ticket at (https://support.developer.arm.com). Give:

- The title (Message Handling Unit Architecture).
- The number (ARM-AES-0072 A.a).
- The section name to which your comments refer.
- The page number(s) to which your comments apply.
- The rule identifier(s) to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

---

**Note**

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

---

# Part A
## Message Handling Unit Architecture Overview

# Chapter A1
# Introduction to Message Handling Unit Architecture

$I_{SDPFS}$  In an SoC there can be multiple systems, each with their own processing elements, peripherals and memory. Each system executes an independent software stack and those software stacks want to communicate with one or more of the other software stacks. The following systems are just some of the many systems that might be found in a typical SoC:

- Application Processor – Executes the RichOS or RTOS and performs the primary task of the SoC

- Communication Processor – A modem or WiFi system, which provides communication to an external network

- System Control Processor – System control functionality, such as power and clock control

- Security Processor – A secure system which provides security functionality to the rest of the SoC. This is typically implemented as a separate system to reduce the possibility of a malicious agent gaining access to secrets.

For the SoC to function as required, the different software entities operating on each of these systems need to communicate with one another. There are different software based communication methods, for example:

- System calls, such as SVC or SMC.

- Events.

- Interrupts.

Each type of communication has advantages and disadvantages and is suitable in different situations.

The usage of system calls and events for communication is considered outside the scope of this architecture. For the remainder of this architecture only interrupt-based communication is considered.

## A1.1  Interrupt Based Communication

$I_{NTKVN}$      At the lowest level, interrupt communication depends on a Sender passing an indication of an event to a Receiver. This event is generated by a software operation, that is, an instruction or memory operation. This event is often passed using a physical signal called an interrupt to the Receiver. Alongside the interrupt, additional information can be included either in-band or out-band. The interrupt and additional information form a transfer, sent by the Sender to the Receiver. A number of transfers then form a complete message, with the number of messages dependent on the message protocol.

$I_{GKWTZ}$      A system that uses interrupts includes an Interrupt Controller. It is common for software to be able to generate an interrupt, often referred to as a Inter-Processor Interrupt (IPI), which can be used to support Inter-Process Communication (IPC). SGIs can be generated in many ways including writing a value to a defined memory location.

# A1.2 Usage

$I_{DRFGP}$      The expected usage of a Message Handling Unit (MHU) is to facilitate IPC between pieces of software which do not share a common Interrupt Controller. The MHU provides a standard interface to software, enabling the Sender of a message to use a generic driver to send a message without having specific knowledge of the Receiver's Interrupt Controller.

$I_{MMJNN}$      This architecture is written with the following assumptions:

- Messages sent across the Message Handling Unit use one of the transport protocols defined in Transport Protocol.
- Sending and receiving agents either have a pre-defined communication format or use an IMPLEMENTATION DEFINED software scheme to negotiate the communication format to use.

## A1.3  Architecture Structure

$I_{BRRXV}$     The MHUv3 architecture provides extensions in  Table A1.1.

**Table A1.1: Summary of the Extension in the Message Handling Unit v3.0 architecture**

| Extension | Description |
| --- | --- |
| *Doorbell Extension* (DBE) | Adds support for Doorbell events to be sent. |
| *FIFO Extension* (FE) | Adds support for variable size in-band payload to be sent using the MHU. |
| *Fast Channel Extension* (FCE) | Adds support for a low overhead Channel designed to be used to provide the last value written. |
| *TrustZone Extension* (TZE) | Adds support for Arm® TrustZone® to the MHU. |
| *Realm Management Extension* (RME) | Adds support for the Realm Management Extension (RME) of the Armv9-A architecture |

$I_{DNLGQ}$     There can be rules defining dependencies between two or more extensions. Examples of these dependencies are:

- Requiring that when extension A is implemented extension B must be implemented.
- Requiring that when extension A is implemented extension B must not be implemented.

---

**Note**

In this architecture, the term RME is used to refer to the Realm Management Extension defined by the MHUv3.0 architecture.

---

# Part B
## Message Handling Unit v3.0

# Chapter B1
# Message Handling Unit v3.0 Overview

$R_{\text{NWMZD}}$     The MHUv3.0 architecture defines the following extensions:

- Doorbell Extension (DBE)
- FIFO Extension (FE)
- Fast Channel Extension (FCE)
- TrustZone Extension (TZE)
- Realm Management Extension (RME)

$R_{\text{ZLLHG}}$     It is IMPLEMENTATION DEFINED which extension an implementation of the MHU implements, but an implementation must implement one of the following extensions:

- DBE
- FE
- FCE

$R_{\text{DYPPK}}$     An implementation of the MHUv3.0 sets the fields of the PBX_AIDR, MBX_AIDR, SSC_AIDR and RSC_AIDR registers to the following values:

- ARCH_REV_MAJOR field is set to `0x2`.
- ARCH_REV_MINOR field is set to `0x0`.

$S_{\text{XSGPW}}$     The extensions implemented are identified to software by the following registers:

- PBX_FEAT_SPT0 and PBX_FEAT_SPT1.
- MBX_FEAT_SPT0 and MBX_FEAT_SPT1.
- SSC_FEAT_SPT0 and SSC_FEAT_SPT1.
- RSC_FEAT_SPT0 and RSC_FEAT_SPT1.

$I_{\text{WQMGP}}$     MHUv3.0 is not backward compatible with previous versions of the MHU architecture.

# B1.1  Structure

$I_{NTMSN}$    An implementation of MHUv3 has the structure as shown in  Figure B1.1
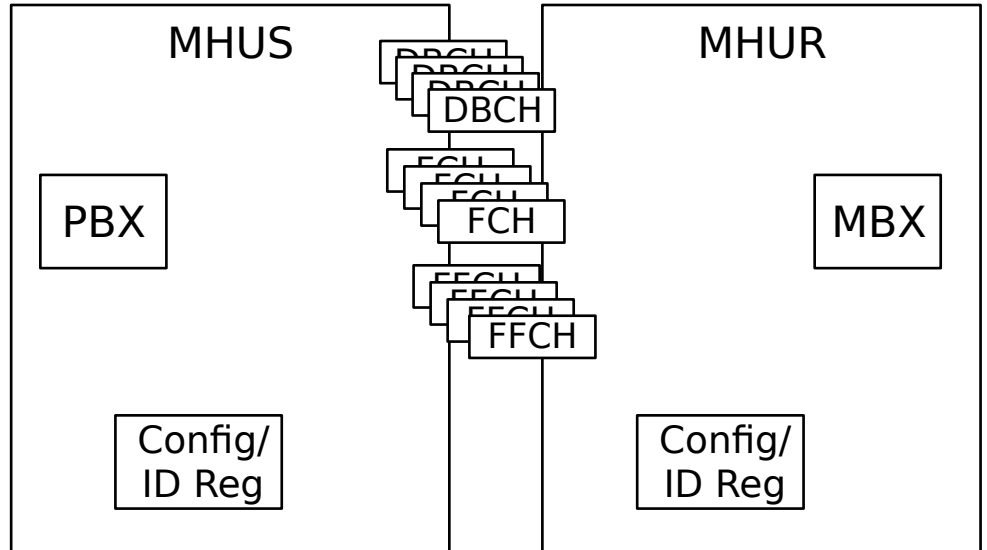


**Figure B1.1: MHU Structure**

- MHUS – MHU Sender.
- MHUR – MHU Receiver.
- DBCH – Doorbell Channel.
- FFCH – FIFO Channel.
- FCH – Fast Channel.
- PBX – Postbox.
- MBX – Mailbox.
- Config/ID Reg – Configuration and Identification registers.

$R_{FDRJH}$    An instance of the MHUv3.0 includes:

- One MHUS.
- One MHUR.
- One PBX.
- One MBX.
- At least one channel.

# Chapter B2
# MHU Sender and Receiver

$I_{YRQTP}$  An implementation of MHUv3.0 can be implemented as a monolithic or distributed component.

$I_{JWQHX}$  The choice of how the MHU is implemented is typically decided by the requirement for the MHU to cross one or more of the following types of domains:

- Clock
- Reset
- Power
- Voltage

$R_{NVQXF}$  The MHUS is used by the agent sending the Transfer.

$R_{ZQWHB}$  The MHUS is used by the agent receiving the Transfer.

$I_{RVGLR}$  In this architecture the agents sending and receiving the Transfer are referred to as the *Sender* and *Receiver*.

$R_{GWCXP}$  When the MHU is implemented as a monolithic component, the MHUS and MHUR are considered as a single entity and enter and exit the Operational state at the same time. For more information on Operational and Non-operational states refer to  Chapter B4 *Power Control*.

$R_{PTFSJ}$  When the MHU is implemented as distributed component, the MHUS and MHUR are two different components connected by an interconnect.

$R_{QMHGN}$  When the MHU is implemented as a distributed component, the MHUS and MHUR are independent entities and are allowed to enter and exit Operational and Non-operational states independently of one another. For information on Operational and Non-operational states refer to  Chapter B4 *Power Control*.

$R_{ZCVWV}$  There must be one MHUS and one MHUR connected together to enable unidirectional communication regardless of whether the MHU is implemented as a monolithic or distributed component.

$I_{QWFQP}$  To separate instances of MHU are required to enable full duplex communication.

$R_{SFBNQ}$      The interconnect used to connect the MHUS and MHUR is IMPLEMENTATION DEFINED, and is allowed to be shared between multiple MHUS/MHUR pairs, as long as one MHUS/MHUR pair is not able to block another.

$R_{XHMZQ}$      The MHUS and MHUR implement the same extensions, except for the following exceptions:

- TZE
- RME

$R_{CNFRV}$      The MHUS has the following interfaces:

- Configuration interface
- Interrupt interfaces
- MHU interface for connection to the MHUR, if the implementation is distributed

$R_{CDBBL}$      The MHUR has the following interfaces:

- Configuration interface
- Interrupt interface
- MHU interface for connection to the MHUS, if the implementation is distributed

# Chapter B3
# **Postbox and Mailbox**

$I_{GGGPQ}$ The MHUS and MHUR contain one Postbox (PBX) and one Mailbox (MBX) respectively.

$R_{DPVVK}$ There is one PBX that is part of the MHUS and is used by the Sender to:

- Configure the behavior of the MHU.
- Send Transfers to the Receiver.
- Receive acknowledgement of a Transfer from the Receiver.

$R_{YGMFZ}$ There is one MBX that is part of the MHUR and is used by the Receiver to:

- Configure the behavior of the MHU.
- Receive Transfers from the Sender.
- Acknowledge Transfers sent by the Sender.

$I_{KHVTF}$ The PBX and MBX contain all the registers used to send Transfers and configure the behavior of the channels allocated to the PBX/MBX.

$R_{WDZXK}$ A PBX and MBX are allocated a single 64KB frame for all their registers.

$R_{KHDJV}$ A PBX and MBX contains a IMPLEMENTATION DEFINED number of:

- *Doorbell Channel* (DBCH), when DBE is implemented.
- *FIFO Channel* (FFCH), when FE is implemented.
- *Fast Channel* (FCH), when FCE is implemented.

For information on DBCH, FFCH and FCH refer to  Chapter B6 *Doorbell Extension*,  Chapter B8 *FIFO Extension* and  Chapter B7 *Fast Channel Extension* respectively.

# Chapter B4
# Power Control

## B4.1 Overview

$R_{\text{VFNDK}}$  The MHU can be implemented as either monolithic or distributed depending on the configuration of the reset, power and voltage domains. Depending on the implementation the following configuration of reset, power and voltages domains are allowed under the MHUv3.0 architecture:

- Monolithic

    - MHUS and MHUR share the same reset, power and voltage domains.

- Distributed

    - MHUS and MHUR can have independent or shared reset, power and voltage domains.

$I_{\text{KPHYX}}$  For the purposes of this architecture the clock is ignored as it is presumed the clock domain is always available or available when required by the MHU. Therefore it is considered to be transparent to the operation of the MHU.

$I_{\text{DXBBL}}$  Whether the implementation of the MHU is monolithic or distributed determines the features which are required to be implemented.

$I_{\text{LBRJT}}$  When the MHU implementation is monolithic, the MHUS and MHUR are considered as a single entity and are both in the Operational or Non-operational state at the same time.

$I_{\text{JVCSG}}$  When the MHU implementation is distributed, the MHUS and MHUR are considered separate. It is possible for any of the following to occur:

- MHUS to be in a Non-operational state when the MHUR is in an Operational state.
- MHUR to be in a Non-operational state when the MHUS is in an Operational state.

$D_{\text{FGTMK}}$      An Operational state is where all the following are true:

- the power is supplied
- the reset is de-asserted
- the MHUS or MHUR are informed, using an IMPLEMENTATION DEFINED mechanism, that it is in an Operational state

$D_{\text{LYFRJ}}$      A Non-operational state is where any of the following are true:

- the power is not supplied
- the reset is asserted
- the MHUS or MHUR are informed, using an IMPLEMENTATION DEFINED mechanism, that it is in a Non-operational state

$U_{\text{JWKHL}}$      An example of an interface which allows the MHUS or MHUR to know whether it is in an Operational or Non-operational state is an instance of an AMBA® P-Channel or Q-Channel interface protocol as defined in [1].

$R_{\text{PBDXY}}$      Before the Sender can send Transfers it must ensure the MHUS is in and ensure the MHUS remains in an Operational state throughout the Transfer.

$R_{\text{TCQHK}}$      Before the Receiver can access the MHUR to receive information about a Transfer it must make sure the MHUR is in and remains in an Operational state whilst it is doing so.

$I_{\text{RKLRS}}$      The method by which the Sender and Receiver get the MHUS and MHUR into an Operational state is IMPLEMENTATION DEFINED.

## B4.2  Auto Op

$I_{ZWMZV}$   Auto Op reduces the software overhead for both the Sender and Receiver when sending Transfers by not requiring the Sender to request the Receiver enter an Operational state before sending the Transfer. Instead the Sender sends the Transfer and the MHUS requests that the MHUR enters an Operational state. The Sender and Receiver can work on a Transfer per Transfer basis or can establish longer communication sessions which cover many Transfers sent over an IMPLEMENTATION DEFINED time period.

$I_{QBCSW}$   There are two level of the Auto Op protocol:

- Auto Op(Min)
- Auto Op(Full)

$R_{GXDVN}$   The Sender and Receiver can determine which version of the Auto Op protocol is implemented by reading the value of the <x>_FEAT_SPT1.AUTO_OP field. Where <x> is one of the following block prefixes: PBX, MBX, SSC, RSC.

### B4.2.1  Auto Op(Min)

$R_{ZFFRG}$   Auto Op(Min) is only allowed to be implemented when the MHU is implemented as a monolithic component.

$I_{YQVJH}$   Auto Op(Min) provides no guarantees that any access to the registers of the MHUS or MHUR will complete or that any Transfer previously sent by Sender will be received by the Receiver.

$R_{NWSYP}$   To guarantee that any accesses to the MHUS and MHUR registers complete, or to guarantee that any Transfers sent by the Sender are received by the Receiver, software must perform a software defined sequence to keep the MHU in an Operational state. For example, making a request to a System Control Processor to enter the MHU into an Operational state.

$I_{MSYCW}$   As the MHU is implemented as a monolithic component keeping either the MHUS or MHUR in an Operational state also means that the MHUR or MHUS respectively will be kept in an Operational state.

$R_{DRHYN}$   When Auto Op(Min) is implemented there is no requirement for the all channels to be idle before entering into the Non-operational state. For information on entry to the Non-operational state and when a channel is considered idle refer to  B4.3 *Entry to a Non-Operational State*.

$S_{WNZBZ}$   When Auto Op(Min) is implemented it is the responsibility of the Sender and Receiver to make sure all Transfers and interrupts have been handled before allowing or requesting the MHUS or MHUR to enter into the Non-operational state.

$R_{YWTTL}$   If the Sender of a Transfer requires to know that the Receiver has received a Transfer, the Sender must:

- Request that the Receiver acknowledges Transfers, using the method defined in the transport protocol being used
- Keep the MHUS in an Operational state until it has received acknowledgements using a software IMPLEMENTATION DEFINED method

$S_{VRGXC}$   If requested, the Receiver of a Transfer must acknowledge the Transfer.

$S_{TJVMN}$   When Auto Op(Min) is implemented the Sender and Receiver must perform a software IMPLEMENTATION DEFINED sequence, to ensure the MHUS and MHUR enter and remain in an Operational state through the sending of the Transfer(s). This must be performed before the Sender and Receiver perform the steps defined by the transport protocol used to send the Transfer.

After the transport protocol is complete the Sender and Receiver can perform a software IMPLEMENTATION DEFINED sequence, to enable the MHUR and MHUR to enter a Non-operational state, if there are no further Transfers to send, otherwise they continue to send Transfer using the steps defined by the transport protocol to be used.

## B4.2.2 Auto Op(Full)

$R_{QRHPW}$     Auto Op(Full) is used when the MHU is implemented as a distributed component.

$I_{TXDWB}$     It is allowed for a monolithic implementation of the MHU to implement Auto Op(Full).

$I_{SMSKS}$     Auto Op(Full) provides the following features:

- An automatic request for MHUR to enter an Operational state when a new Transfer is started.
- A method for the Sender or Receiver to request the MHUS or MHUR respectively remains in an Operational state.
- A method for the Sender or Receiver to ignore outstanding Transfers.

$R_{XWLJD}$     When the Sender attempts to send a Transfer the MHUS is required to request the MHUR enters an Operational state. The method by which the MHUS achieves this is IMPLEMENTATION DEFINED.

$R_{NZDHL}$     The MHUS is required to keep the MHUR in an Operational state until it has completed the actions required to inform the MHUR of a pending Transfer.

$R_{VYSHZ}$     Once the MHUS has performed the actions required to inform the MHUR of a pending Transfer, it is the responsibility of the MHUR to keep itself in an Operational state, if it is configured to consider the state of the channels before entering the Non-operational state, until the Receiver acknowledges the Transfer.

$R_{QGZCP}$     The Sender and Receiver can use the PBX_CTRL.CH_OP_MSK and MBX_CTRL.CH_OP_MSK fields respectively to control whether the MHUS and MHUR consider the state of channels when requested to enter a Non-operational state.

$R_{PNLFC}$     The Sender and Receiver can use the PBX_PWR_CFG.OP_REQ and MBX_PWR_CFG.OP_REQ fields respectively to request that either MHUS or MHUR remains in an Operational state.

$R_{JYVVY}$     The method by which the MHUS and MHUR indicate the need to be in an Operational state is is IMPLEMENTATION DEFINED.

$U_{LYGPM}$     An example of an interface which allows the MHUS or MHUR to indicate it requires to remain in an Operational state is an instance of an AMBA® P-Channel or Q-Channel interface protocol as defined in [1].

$R_{FWTVL}$     If the Sender of a Transfer requires to know that the Receiver has received a Transfer, the Sender must:

- Request that the Receiver acknowledges Transfers, using the method defined in the transport protocol being used.
- Keep the MHUS in an Operational state until it has received acknowledgements by setting the PBX_CTRL.OP_REQ field to `0b1`.

$S_{TDFYV}$     If requested, the Receiver of a Transfer must acknowledge the Transfer.

$I_{TGGMV}$     The method by which the Receiver acknowledges a Transfer depends on the transport protocol being used.

$S_{YVPCV}$     The recommended software sequence for the Sender to send a Transfer when Auto Op(Full) is implemented is:

1. When the Sender wants to send a Transfer it first set the PBX_CTRL.OP_REQ field to `0b1` to make sure the MHUS remains in an Operational state.
2. Perform the steps defined by the selected transport protocol to send the Transfer.
3. If the Sender has more Transfers to send, it repeats step 2.
4. Set the PBX_CTRL.OP_REQ to `0b0` to remove the requirement for the MHUS to remain in the Operational state.

It is allowed for the Sender to send another Transfer, if required, using the same or different channels before completing all steps of the transport protocol, as long as all the following are all true:

- All steps in the transport protocol are performed before step 4.
- When sending the new Transfer using the same DBCH the Transfer uses a flag bit which currently doesn't have a Transfer outstanding.

$S_{QMNYB}$  The recommended software sequence for the Receiver to use when Auto Op(Full) protocol on receiving an interrupt from the MHUR is:

1. If the Sender and Receiver are setting up a long term communication session the Receiver sets the MBX_CTRL.OP_REQ fields to 0b1

2. Process the interrupt from the MHU using the steps defined by the transport protocol being used to send the Transfer

   This processing can include one or more of the following:

   - Reading out-band memory locations.
   - Acknowledge the Transfer.

3. Wait for the next Transfer in the communication session to arrive before repeating step 2.

4. When the communication session is no longer required the Receiver sets the MBX_CTRL.OP_REQ field to 0b0.

It is allowed for the Receiver to receiver another Transfer, if required, using the same or different channels before completing all steps of the transport protocol, as long as all steps in the transport protocol are performed before step 4.

$S_{NNPTS}$  When the Sender no longer wants to send Transfers to the Receiver and it has received all the expected Transfer Acknowledgements it was expecting, it sets the PBX_CTRL.OP_REQ field to 0b0, if it is not already set to 0b0.

$S_{WGGSY}$  The MHUS and MHUR can only enter the Non-operational state if all the DBCHs, FFCHs and FCs are idle. If the Sender or Receiver wants to enter the MHUS or MHUR into a Non-operational state immediately without waiting for any outstanding or new Transfers to complete, the Sender or Receiver can set PBX_CTRL.CH_OP_MSK or MBX_CTRL.CH_OP_MSK to 0b1. This causes all channels to be considered idle for the purpose of entry into the Non-operational state. Setting either PBX_CTRL.CH_OP_MSK or MBX_CTRL.CH_OP_MSK to 0b1 can lead to lost of Transfer data and Transfer Acknowledgements.

Arm recommends that the:

- Sender or Receiver informs the other that they want to enter the MHUS or MHUR into a Non-operational state before they set the PBX_CTRL.CH_OP_MSK or MBX_CTRL.CH_OP_MSK field to 0b1.
- Sender on receipt of this request stops sending new Transfers.
- Receiver on receipt of this request is prepared that accesses to retrieve data from the FIFO may result in no data being returned.

$S_{BKSHQ}$  There are times when the MHUR enters a Non-operational state, whereby it may be slower to exit from the Non-operational state than normal. For example, the MHUR will take longer to exit a Non-operational state where the power is removed compare to the time taken for the MHUR to exit a Non-operational state where the power was not removed and only the reset was asserted.

In these cases it is desirable for the Sender to be aware so that it can perform a software IMPLEMENTATION DEFINED sequence to wake the MHUR. This allows the Sender to perform other actions whilst the MHUR enters the Operational state,

It is the responsibility of the Sender and Receiver to negotiate this using a software IMPLEMENTATION DEFINED method before the MHUR enters the Non-operational state.

$S_{YDQHS}$  There are times when the MHUR enters a Non-operational state, whereby it is unable to exit from without actions being performed by another entity in the system. An example of this is when the MHUR enters a Non-operational state and the voltage supply used by the power domain is turned off. For the MHUR to enter an Operational state the voltage supply must be turned on before then powering the power domain back on. The process of enabling and disabling voltage supplies typically requires software interventions, for example by a System Control Processor.

In these cases it is required that the Sender is aware, so that it can perform a software IMPLEMENTATION DEFINED sequence to wake the MHUR. The software IMPLEMENTATION DEFINED sequence may be limited to removing the deadlock possibility, for example sending a request to a System Control Processor to enable the voltage supply for the power domain, and not actually entering the MHUR into an Operational state.

# B4.3 Entry to a Non-Operational State

$I_{BWJVC}$ The MHUS and MHUR can enter a Non-operational state in either a controlled or uncontrolled manner.

$I_{QQQVK}$ Depending on whether it is the MHUS or MHUR and whether the entry is controlled or uncontrolled the entry into a Non-operational state is different. The following sections defines the rules for both controlled and uncontrolled entry into a Non-operational state for both the MHUS and MHUR.

## B4.3.1 Controlled entry into Non-operational state

$R_{NNYLN}$ A controlled entry into the Non-operational state for either the MHUS or MHUR is defined as when the MHUS or MHUR is requested to enter the Non-operational state and the MHUS or MHUR accept the request before then entering the Non-operational state.

$R_{YQDSM}$ It is valid for the MHUS or MHUR to deny entry into a Non-operational state and to then remain in the Operational state.

$I_{YQBDG}$ After a denial to a request to enter the Non-operational state in a controlled manner, it is valid for additional requests, to be made without any changes in conditions. It is also valid for an uncontrolled entry into the Non-operational state to occur after a denial of a controlled entry.

$I_{VQHCF}$ The request to enter a Non-operational state comes from an IMPLEMENTATION DEFINED hardware entity which might have been triggered by software running on the Sender, Receiver or another agent. The method by which the IMPLEMENTATION DEFINED hardware entity makes the request to the MHUS or MHUR is IMPLEMENTATION DEFINED.

$U_{VFSJN}$ An instance of an AMBA® P-Channel or Q-Channel interface protocol, as defined in [1], is an example of an interface which allows for the IMPLEMENTATION DEFINED hardware entity, referred to as a controller by [1], to make requests to the MHUS or MHUR to enter or exit a Non-operational state.

$R_{QQVMS}$ For the MHUS to accept entry into the Non-operational state, the following must all be true:

- No outstanding accesses from the MHUS or Sender to the MHUR reset domain.
- No outstanding accesses from the MHUR or Receiver to the MHUS reset domain.
- No outstanding accesses to the MHUS by the Sender.
- When all DBCHs and FFCHs are idle.
- The value of PFFCW<n>_CTRL.FF is equal to PFFCW<n>_ST.FF for all implemented FFCHs.
- PBX_CTRL.OP_REQ is set to `0b0`, if Auto Op(Full) is implemented.

$I_{WPDZS}$ Arm strongly recommends that before the MHUS enters the Non-operational state in a controlled manner that the Sender completes all Transfers, including receiving any acknowledgements associated with those Transfers.

$R_{QYXDM}$ For the MHUR to accept entry into the Non-operational state the following must all be true:

- No outstanding accesses from the MHUS or Sender to the MHUR reset domain.
- No outstanding accesses from the MHUR or Receiver to the MHUS reset domain.
- No outstanding accesses to the MHUR from the Receiver.
- All DBCHs, FFCHs and FCs are idle.
- The value of MFFCW<n>_CTRL.FF is equal to MFFCW<n>_ST.FF for all implemented FFCHs.
- MBX_CTRL.OP_REQ is set to `0b0`, if Auto Op(Full) is implemented.

$I_{SZRND}$ Arm strongly recommends that when the MHUR enters the Non-operational state in a controller manner and the exit time is considered to be a long duration or exit from the Non-operational state requires interaction with a hardware or software entity external to the MHU that the Sender is informed of this. This allows the Sender to take different action when it requires the MHUR to enter an Operational state.

$R_{TDCNZ}$ An access to the MHUS or MHUR is considered outstanding until all the required effects of the access are complete. This includes any side effects from the access. If the registers is one of the registers which supports early write

responses, the access is considered outstanding, even though the write response may been returned, until the required steps have been completed.

Examples of side effects an access can have are:

- Update to fields within other registers.

  For example, a write to the MFFCW<n>_FIFO_POP field, when the MFFCW<n>_CTRL.RA_EN field is set to `0b0`, cause a number of bytes to be popped from the FIFO. This leads to an update of the following fields:

  - PFFCW<n>_ST.FFS
  - PFFCW<n>_PAY.FFS
  - PFFCW<n>_ACK_CNT.ACN_CNT and PFFCW<n>_ACK_CNT.ACK_CNT_OVRFLW fields, if any of popped bytes generated a FIFO Pop Ack event
  - MFFCW<n>_ST.FFL
  - MFFCW<n>_FLG.FFL

- Update of the status of any interrupts.

  For example, a read of the MFCW<n>_PAY register causes the Channel Transfer interrupt for the FCH to be updated.

- Update of any IMPLEMENTATION DEFINED internal state of the MHU.

$R_{KTPMT}$   When Auto Op(Min) is implemented a DBCH is always considered idle.

$R_{DHSLS}$   When Auto Op(Full) is implemented, for the MHUS a DBCH is considered idle when either of the following are true:

- All bits in the PDBCW<n>_ST register are `0b0`, and PBX_CTRL.CH_OP_MSK is set to `0b0`.
- PBX_CTRL.CH_OP_MSK is set to `0b1`.

$R_{KMQWB}$   When Auto Op(Full) is implemented, for the MHUR a DBCH is considered idle when either of the following are true:

- All bits in the MDBCW<n>_ST register are `0b0`, and MBX_CTRL.CH_OP_MSK is set to `0b0`.
- MBX_CTRL.CH_OP_MSK is set to `0b1`.

$R_{VBYYM}$   When Auto Op(Min) is implemented a FFCH is always considered idle.

$R_{CMHCV}$   When Auto Op(Full) is implemented, for the MHUS a FFCH is considered idle when either of the following are true:

- FIFO is empty and PBX_CTRL.CH_OP_MSK is set to `0b0`.
- PBX_CTRL.CH_OP_MSK is set to `0b1`.

$R_{CBTZN}$   When Auto Op(Full) is implemented, for the MHUR a FFCH is considered idle when either of the following are true:

- FIFO is empty and MBX_CTRL.CH_OP_MSK is set to `0b0`.
- MBX_CTRL.CH_OP_MSK is set to `0b1`.

$R_{THYRL}$   When Auto Op(Min) is implemented a FCH is always considered idle.

$R_{VCZVF}$   When Auto Op(Full) is implemented a FCH is considered idle when either of the following are true:

- MBX_CTRL.CH_OP_MSK is set to `0b0` and there is no outstanding Transfer
- MBX_CTRL.CH_OP_MSK is set to `0b1`.

$R_{PNBDG}$   A Transfer is considered outstanding for an FCH from the point where the write to the PFCW<n>_PAY register occurs until the read of the MFCW<n>_PAY register, allowing for any delay in crossing a clock, reset and power boundary between the MHUS and MHUR.

$I_{ZMNQY}$     The value of the MBX_FCH_CTRL.INT_EN field has no effect on whether a Transfer in an FCH is considered outstanding or not. The MBX_FCH_CTRL.INT_EN field only controls whether an interrupt is generated.

$I_{RBSVR}$     For more information on Auto Op(Min) and Auto Op(Full) refer to  B4.2 *Auto Op*.

## B4.3.2  Uncontrolled entry into a Non-operational state

$R_{GSRNY}$     An uncontrolled entry into the Non-operational state for either the MHUS or MHUR, is defined as when the MHUS or MHUR enter the Non-operational state, without having accepted a request to enter the Non-operational state.

$I_{YWFYT}$     An uncontrolled entry into the Non-operational state can occur at any point, including after a previous attempt to enter the Non-operational state in a controlled manner that was denied by either the MHUS or MHUR.

$I_{RKPKH}$     Uncontrolled resets are not expected to be generated under normal operating conditions and are expected to be generated as a last response to a system deadlock or security impact. For example, a watchdog event.

$R_{FHLRF}$     If the MHUS or MHUR enter the Non-operational state in an uncontrolled manner it is UNPREDICTABLE whether:

  - Any Transfers that were outstanding at the time are lost or not.
  - If FE is implemented:

    - The value of PFFCW<n>_ST.FFS and MFFCW<n>_ST.FFL accurately reflects the number of valid bytes in the FIFO.
    - Any bytes are written to the FIFO on a write of the MFFCW<n>_PAY register.
    - Any bytes are read from the FIFO on a read of the MFFCW<n>_PAY register.
    - Any bytes are popped from the FIFO on a read of MFFCW<n>_PAY, when MFFCW<n>_CTRL.RA_EN is set to `0b1,` or on a write to the MFFCW<n>_FIFO_POP register.

$S_{RDKMF}$     Software must be prepared that if an uncontrolled entry into the Non-operational state occurs for either the MHUS or MHUR that:

  - Any Transfers which have not been acknowledged by the Receiver might have been lost.
  - Sending of any future Transfers might not be possible.

$S_{QKBGJ}$     Arm recommends that software perform an IMPLEMENTATION DEFINED sequence to recover the MHU to a known state before sending any new Transfers. Part of the IMPLEMENTATION DEFINED sequence might include flushing any FFCHs which are implemented, using the FIFO flush mechanism.

$R_{WCRCG}$     It is UNPREDICTABLE whether the MHUS considers the MHUR to be in an Operational or Non-operational state, if the MHUR entered the Non-operational state in an uncontrolled manner. This means it is valid for the MHUS to treat the MHUR as if it was in an Operational state even though it has entered an Non-operational state.

$R_{VSCXQ}$     It is UNPREDICTABLE whether the MHUR considers the MHUS to be in an Operational or Non-operational state, if the MHUS entered the Non-operational state in an uncontrolled manner. This means it is valid for the MHUR to treat the MHUS as if it was in an Operational state even though it has entered an Non-operational state.

$I_{XKCGC}$     For the remainder of this architecture when talking about either the MHUS or MHUR being in a Non-operational state, implies that the MHUS or MHUR entered the Non-operational state in a controlled manner.

# Chapter B5
# Data Endianness

$I_{\text{JYRLY}}$      It is possible that the Sender and Receiver use different data endianness when sending Transfers between each other.

$R_{\text{FBSSF}}$      The registers of the MHU are little-endian device.

$S_{\text{JTRFV}}$      It is the responsibility of the Sender and Receiver to perform any byte swapping operations when passing data in-band using the MHU. This means any Sender or Receiver that uses an endianness other than little-endian will need to perform a sequence to arrange the bits and bytes of the payload to be little-endian, when accessing any registers of the MHU.

$S_{\text{TDDYX}}$      If the Sender and Receiver used different endianness, software must agree on, the endianness of any out-band payload which is sent along side the Transfer. Failure to do so will lead to corruption or lost of Transfer data.

$S_{\text{PXTMM}}$      The method the Sender and Receiver agree on a data endianness to use for out-band payload is software defined.

# Chapter B6
# Doorbell Extension

$I_{RVWPV}$      The Doorbell Extension (DBE) defines a type of channel called a Doorbell Channel (DBCH).

$I_{MNMQF}$      A DBCH enables a single bit Transfer to be sent from the Sender to Receiver. The Transfer indicates that an event has occurred.

$R_{HBNFC}$      When DBE is implemented, the number of DBCHs implemented is IMPLEMENTATION DEFINED between 1 and 128.

$R_{KSKHN}$      When DBE is not implemented, all registers related to DBCHs are reserved and it is considered that the MHU has 0 DBCH.

$R_{ZJLJR}$      DBCH are numbered starting from 0 in ascending order.

$R_{KGFDB}$      Each DBCH contains 32 individual fields, referred to as flags, each of which can be used independently.

$R_{PWNZN}$      The meaning of each flag in a DBCH is defined by software.

$R_{QBZLZ}$      It is possible for the Sender to send multiple Transfers at once using a single DBCH, so long as each Transfer uses a different flag in the DBCH.

$R_{RDWWB}$      The Receiver of the Transfer is required to acknowledge receipt of the Transfer, by clearing the flag, before it can receive a new Transfer using the same flag.

$S_{JCTVP}$      Sending two Transfers using the same flag of the same DBCH without waiting for the first Transfer to be acknowledged by the Receiver can lead lead to the two Transfers being merged into a single Transfer as observed by the Receiver.

$R_{TYVKT}$      A DBCH has the following resources:

- A Doorbell Channel Window.
- Storage for 32 flags.

- Channel Transfer event.
- Channel Transfer Acknowledge event.

# B6.1  Doorbell Channel Window

$R_{YYDGW}$     The Doorbell Channel Window is 8 32-bit words.

$R_{HYMRJ}$     Table B6.1 shows the registers in the DBCW when accessed via the PBX:

**Table B6.1: DBCW when accessed from PBX**

| Offset | Name | Access |
|--------|------|--------|
| 0x00 | PDBCW<n>_ST | RO |
| 0x04 | Reserved | |
| 0x08 | Reserved | |
| 0x0C | PDBCW<n>_SET | WO/RAZ |
| 0x10 | PDBCW<n>_INT_ST | RO |
| 0x14 | PDBCW<n>_INT_CLR | WO/RAZ |
| 0x18 | PDBCW<n>_INT_EN | RW |
| 0x1C | PDBCW<n>_CTRL | RW |

$R_{JYYHF}$     Table B6.2 shows the registers in the DBCW when accessed via the MBX:

**Table B6.2: DBCW when accessed from MBX**

| Offset | Name | Access |
|--------|------|--------|
| 0x00 | MDBCW<n>_ST | RO |
| 0x04 | MDBCW<n>_ST_MSK | RO |
| 0x08 | MDBCW<n>_CLR | WO/RAZ |
| 0x0C | Reserved | |
| 0x10 | MDBCW<n>_MSK_ST | RO |
| 0x14 | MDBCW<n>_MSK_SET | WO/RAZ |
| 0x18 | MDBCW<n>_MSK_CLR | WO/RAZ |
| 0x1C | MDBCW<n>_CTRL | RW |

$R_{FFHVZ}$     The PDBCW<n>_SET and MDBCW<n>_CLR register update the value of the {PDBCW/MDBCW}<n>_ST registers as follows:

- Any field set to 0b0 by a write to either register is ignored.
- Any field set to 0b1 in the PDBCW<n>_SET register sets the associated field in the {PDBCW/MDBCW}<n>_ST registers to 0b1.
- Any field set to 0b1 in the MDBCW<n>_CLR register clears the associated field in the {PDBCW/MDBCW}<n>_ST registers to 0b0.

$R_{CYWTR}$     The setting of a bit in the {PDBCW/MDBCW}<n>_ST takes precedence over clearing.

$S_{SPCCB}$      Arm strongly recommends that the Sender never writes to the PDBCW<n>_SET register to set a field which is already set in the {PDBCW/MDBCW}<n>_ST register.

$R_{CRNNS}$      The MDBCW<n>_ST_MSK register is the result of a bitwise AND between the MDBCW<n>_ST and the inverted value of the MDBCW<n>_MSK_ST register.

$R_{CHMVJ}$      The MDBCW<n>_MSK_SET and MDBCW<n>_MSK_CLR registers update the value of the MDBCW<n>_MSK_ST as follows:

- Any field set to 0b0 by a write to either register is ignored.
- Any field set to 0b1 in the MDBCW<n>_MSK_SET register sets the associated field in the MDBCW<n>_MSK_ST register to 0b1.
- Any field set to 0b1 in the MDBCW<n>_MSK_CLR register clears the associated field in the MDBCW<n>_MSK_ST register to 0b0.

$R_{ZSRMX}$      The setting of a bit in the MDBCW<n>_MSK_ST register takes precedence over clearing.

## B6.2 Events

$I_{DCMYB}$    Each DBCH has the following events:

- 32 Channel Transfer.
- Channel Transfer Acknowledge.

### B6.2.1 Channel Transfer event

$R_{BTBCW}$    There is a Channel Transfer event per each flag in the DBCH.

$R_{WMYTF}$    Channel Transfer event <x> is generated when a write to the PDBCW<n>_SET.FLG<x> with a value of `0b1` occurs.

### B6.2.2 Channel Transfer Acknowledge event

$R_{LRFNQ}$    A Channel Transfer Acknowledge event is generated when the Receiver writes to the MDBCW<n>_CLR register, independent of whether any bits in the {PDBCW/MDBCW}<n>_ST register are updated.

# Chapter B7
# Fast Channel Extension

$I_{BVHJC}$      The Fast Channel Extension (FCE) defines a type of channel called a Fast Channel (FCH).

$I_{YMZJW}$      A FCH is intended for lower overhead communication between Sender and Receiver at the expense of determinism. A FCH allows the Sender to update the channel value at any time, regardless of whether the previous value has been seen by the Receiver. When the Receiver reads the channel's contents it gets the last value written to the channel.

$I_{FNZMK}$      A FCH is considered lossy in nature, and means that the Sender has no architected way of knowing if or when the Receiver will act on the Transfer.

$I_{ZPZSN}$      FCHs are expected to behave as RAM which generates interrupts when writes occur to the locations within the RAM.

$R_{YJXZZ}$      The reset value of a FCH is UNKNOWN.

$R_{FMNWT}$      When FCE is implemented, the number of FCH that an implementation of the MHU can support is IMPLEMENTATION DEFINED between:

- 1-1024, when the Fast Channel word-size is 32-bits.
- 1-512, when the Fast Channel word-size is 64-bits.

$R_{XFZZB}$      When FCE is not implemented, the number of FCH is 0 and all registers related to FCHs are Reserved.

$R_{CKLPQ}$      FCH are numbered from 0 in ascending order.

$S_{YWKNX}$      FCHs must only be used for sending Transfers where all the following apply:

- If the Sender writes two values to the same FCH it does not matter if the Receiver sees both values as two separate Transfer or sees only the last value written.
- Sender does not require any indication that the Receiver has received the Transfer.

$R_{XVPTY}$        A FCH has the following resources:

- A Fast Channel Window.
- Fast Channel Storage.
- A Channel Transfer event.

$R_{CZCHK}$        Alongside the resource each FCH has, there is a set of shared Fast Channel Control registers which are shared between all FCH implemented in the Mailbox.

## B7.1  Fast Channel Groups

$I_{BWJJT}$    FCHs are controlled in groups called Fast Channel Groups (FCG).

$R_{RBQDQ}$    There can be between 1 and 32 FCGs in a Postbox or Mailbox.

$R_{LHHTS}$    A FCG can contain between 1 and 32 FCHs.

$R_{BCTPB}$    The number of FCGs is less than or equal to the number of FCHs implemented in the MHU.

$R_{QZQBP}$    The number of FCH in each FCG is defined as:

NUM_FCH_PER_FCG = NUM_FCH / NUM_FCG

Where:

- NUM_FCG is the number of FCGs in the PBX or MBX.
- NUM_FCH is the number of FCHs in the PBX or MBX.
- NUM_FCH_PER_FCG is the number of FCHs per FCG.

$S_{XKBTS}$    Software can discover:

- The number of FCHs in the Postbox or Mailbox using the value of <x>_FCH_CFG0.NUM_FCH field.
- The number of FCGs in the Postbox or Mailbox using the value of <x>_FCH_CFG0.NUM_FCG field.
- The number of FCHs per FCG using the value of <x>_FCH_CFG0.NUM_FCH_PER_FCG field.

Where <x> is either PBX for the Postbox or MBX for the Mailbox.

$R_{DZVYV}$    The FCH which are part of FCG <n> can be determined as follows:

- Lowest FCH in the Group is = n * NUM_FCH_PER_FCG.
- Highest FCH in the Group is = ((n + 1) * NUM_FCH_PER_FCG) - 1.

Where <n> is the FCG number.

## B7.2  Fast Channel Storage

$I_{XYVYR}$        The size of the storage of the FCH is set by the Fast Channel word-size.

$R_{CMPSG}$        The Fast Channel word-size can be either 32-bit or 64-bit in size.

$R_{QDTWK}$        All FCHs have the same Fast Channel word-size.

$S_{RHCHH}$        Software can discover the Fast Channel word-size by reading the value of the <x>_FCH_CFG.FCH_WS field.

          Where <x> is either PBX for the Postbox or MBX for the Mailbox.

## B7.3  Fast Channel Window

R$_{KBWZW}$     The Fast Channel Window (FCW) allows the Sender and Receiver to access the registers of the FCH.

R$_{JZZPV}$     When the Fast Channel word-size is 32-bit the FCW occupies 4bytes aligned on a 4 byte boundary.

R$_{BLKRB}$     When the Fast Channel word-size is 64-bit the FCW occupies 8bytes aligned on an 8 byte boundary.

R$_{SCKMT}$      Table B7.1 shows the registers in the FCW when accessed via the PBX:

**Table B7.1: FCW when accessed from PBX**

| Offset | Name | Access |
|---|---|---|
| 0x00 | PFCW<n>_PAY | RW |

R$_{NYDRX}$      Table B7.2 shows the registers in the FCW when accessed via the MBX:

**Table B7.2: FCW when accessed from MBX**

| Offset | Name | Access |
|---|---|---|
| 0x00 | MFCW<n>_PAY | RW |

---

**Note**

Both the PFCW<n>_PAY and MFCW<n>_PAY registers are either a 32-bit or 64-bit version depending on the Fast Channel word-size. In this architecture, the term PFCW<n>_PAY and MFCW<n>_PAY refers to either version of the register, whilst PFCW<n>_PAY32/MFCW<n>_PAY32 and PFCW<n>_PAY64/MFCW<n>_PAY64 refer to the 32-bit and 64-bit versions of the registers respectively.

---

I$_{ZDJCN}$     The PFCW<n>_PAY and MFCW<n>_PAY registers are windows to the storage of the FCH and therefore a write to the FCH through either the PFCW<n>_PAY or MFCW<n>_PAY register is visible by a read to either register.

S$_{MCTKS}$     The Sender and Receiver must access the FCW atomically aligned to the Fast Channel word-size, otherwise, partial or corrupted data can be read from or stored to the FCH(s).

S$_{NGTYH}$     If software accesses the FCW with a size that is not equal to the Fast Channel word-size, but is a supported transaction size of the MHU, the MHU treats this as a legal access and will perform the operation, including generating the Channel Transfer event. For information on the supported transaction sizes of the MHU refer to C1.1 *Register Access*,

R$_{KBSFG}$     There is no guarantee of ordering between read and write accesses between either the PFCW<n>_PAY and MFCW<n>_PAY register.

S$_{PCPVL}$     To avoid data loss or consumption of UNKNOWN data, the Sender and Receiver must:

- Only use a FCH to send data from the Sender to the Receiver as defined by one of the transport protocols defined in  Chapter D2 *Transport Protocols*.

- Never allow both Sender and Receiver to write to the FCH at the same time, the method by which the Sender and Receiver coordinate this is software IMPLEMENTATION DEFINED.

- If the FCH is not to be initialized by either the Sender or Receiver to a known value, the:

- Sender must write all bytes with a value when writing to a FCH.
- Receiver must only read the FCH after the write has occurred for that FCH. The FCH Transfer or Fast Channel Group Transfer interrupts can be used to know when the write has occurred.

- If the FCH is to be initialized to a known value by the Receiver, then:

  - The Receiver must inform the Sender it is going to initialize the FCH using a software IMPLEMENTATION DEFINED method.
  - The Sender must not perform any writes to the FCH which is to be initialized.
  - The Receiver must inform the Sender when the FCH has been initialized using a software IMPLEMENTATION DEFINED method.
  - The Sender must check that it can observe the initialized value before sending any Transfers using the FCH.

- If the FCH is to be initialized to a known value by the Sender, then:

  - The Sender must inform the Receiver it is going to initialize the FCH using a software IMPLEMENTATION DEFINED method.
  - The Sender must inform the Receiver when the FCH has been initialized using a software IMPLEMENTATION DEFINED method.
  - The Receiver must clear any Channel Transfer event for the FCH which is being initialized, ignoring the value in FCH.

## B7.4 Fast Channel Control registers

$I_{BXFDT}$      FCHs are controlled exclusively by the Receiver using the Fast Channel Control registers implemented in the Mailbox Control page.

$I_{YNPDC}$      There are no control registers for FCHs in the Postbox.

$R_{ZSKVF}$      Table B7.3 shows the registers used to control the FCHs and their location within the Mailbox Control page of the Mailbox.

**Table B7.3: Fast Channel control registers located in the MBX_CTRL page when accessed via MBX**

| Offset | Name | Access |
| --- | --- | --- |
| `0x140` | MBX_FCH_CTRL | RW |
| `0x144` | MBX_FCH_GRP_INT_EN | RW |
| `0x470` | MBX_FCH_GRP_INT_ST | RO |
| `0x480 + (4 * n)` | MBX_FCG<n>_INT_ST | RO |

$R_{XDBJN}$      The MBX_FCH_GRP_INT_EN, MBX_FCH_GRP<n>_INT_ST and MBX_FCH_GRP_INT_ST registers are only implemented if Fast Channel Group Transfer Interrupts are implemented, otherwise they are RES0.

Refer to  Chapter B11 *Interrupts* for more information on Fast Channel Group Transfer Interrupts.

## B7.5  Read-Acknowledge

$I_{ZSYWN}$   FCHs support Read-Acknowledge, which reduces the software overhead for the Receiver by allowing it to perform all the following with a single read access to the MFCW<n>_PAY register:

- Read the value held in the MFCW<n>_PAY register.
- Acknowledge the Transfer.
- Clear the Channel Transfer interrupt, if it was asserted, for the FCH.

$S_{GGJXT}$   Unlike other channel types, FCHs do not generate a Channel Transfer Acknowledge event back to the Sender, but the Transfer must be acknowledged by the Receiver to enable future Fast Channel Transfer interrupts to be generated by the channel. For more information on the Fast Channel Transfer interrupt refer to  Chapter B11 *Interrupts*

# B7.6 Events

$I_{LTTPJ}$        An FCH only has a Channel Transfer event.

$R_{RXQHS}$        The Channel Transfer event is generated for a FCH when a write of any size to the PFCW<n>_PAY register occurs, independent of the value written.

# Chapter B8
# FIFO Extension

$I_{GDGVG}$  The FIFO Extension (FE) defines a Channel type called a FIFO Channel (FFCH).

$I_{GXDRW}$  A FFCH allows a Sender to send:

- Multiple Transfer to the Receiver without having to wait for a previous Transfer to be acknowledged by the Receiver, as long as the FIFO has room for the Transfer.
- Transfers which require the Receiver to provide acknowledgement.
- Transfers which have in-band payload.

In all cases, the data is guaranteed to be observed by the Receiver in the same order which the Sender sent it, reducing the overhead in sending multiple or long Transfers.

$I_{RFCBB}$  FFCH allow larger payloads to be sent, however the intent is for the Transfers to have sizes which are 10s of bytes long. An example usage for a FFCH is to send a command packet which contains a command header of a few bytes, followed by an address pointer to an out-band payload which the command relates to. The sending of very long Transfers can have a performance impact on the Sender or Receiver over using shared memory.

$R_{QDFMY}$  When FE is implemented, the number of FFCH an implementation of the MHU can support is IMPLEMENTATION DEFINED between 1 and 64.

$R_{ZPHHM}$  When FE is not implemented, the number of FFCH is 0 and all registers related to FE are Reserved.

$R_{FPTGY}$  FFCH are numbered starting from the 0 in ascending order.

$R_{JPVGT}$  A FFCH has the following resources:

- A FIFO.
- A Flag History Buffer.
- A FIFO Channel Window (FFCW).
- Channel Transfer event.

- Channel Transfer Acknowledge event and acknowledge counter.
- Channel Tidemark events.
- FIFO Pop Ack event.
- FIFO Flush events.

$R_{PPLXC}$   The depth of the FIFO must satisfy both of the following conditions:

- Be between 1 and 1024 bytes.
- A multiple of the largest push operation which the MHU implementation supports.

  For example, if the largest push operation allowed is four bytes the depth of the FIFO must be a multiple of four.

$R_{HMFXJ}$   All FIFOs in an implementation of the MHU are the same depth.

$R_{HRYRD}$   Bytes within the FIFO are either invalid or valid.

$R_{VQQLV}$   Bytes are invalid until the Sender pushes data onto the FIFO, which is stored in the bytes, making these bytes valid. The bytes remain valid until the Receiver pops the data, stored in the bytes, from the FIFO. When data is popped from the FIFO the bytes which store the popped data become invalid.

# B8.1 Data Flags

$I_{\text{YFSZR}}$      Each byte stored in the FIFO has three different flags stored alongside the value of the byte. Theses flags are referred to as Data flags and are called:

- Start of Transfer (SOT) - Indicates the start of a new Transfer in a stream of bytes.
- End of Transfer (EOT) - Indicates the last byte of a Transfer in a stream of bytes.
- Acknowledge (ACK) - Request by the Sender for an indication when the byte has been popped from the FIFO. This is considered to be when the Receiver acknowledges the Transfer.

$R_{\text{FFPRG}}$      The SOT and EOT flags can either be:

- Managed by the Sender.
- Managed jointly by the Sender and MHU.
- Managed by the MHU.

$I_{\text{DTBCJ}}$      The different ways of managing the SOT and EOT flags is referred to as the Transfer Delineation mode. For more information refer to  *B8.1.1 Transfer Delineation Mode*.

$R_{\text{HWMNV}}$      The ACK flag is always managed by the Sender.

$R_{\text{WKKYX}}$      The value of the data flags, which are to be used for the next byte(s) pushed onto the FIFO, can be read and set via the PFFCW<n>_FLG register.

$R_{\text{LJYWB}}$      The SOT and EOT data flags can be read by the Receiver, for the last bytes read from the FIFO, using the MFFCW<n>_FLG register.

$I_{\text{GNHJG}}$      The ACK flag is not exposed to the Receiver and is instead used by the MHUR to generate the Channel Transfer event. For more information refer to  *B8.5 Events*.

$S_{\text{QQNKK}}$      It is the responsibility of software to:

- Select the Transfer Delineation mode to be used before sending a Transfer.
- Set the ACK flag as required when pushing the last byte of the Transfer onto the FIFO.
- If using the software or partial flag Transfer Delineation mode, to set the SOT and EOT flags for each byte as required, before pushing the byte onto the FIFO, based on the bytes position within the Transfer.

Failure to set the SOT and EOT flags correctly can lead to data corruption or loss when the Receiver reads data from the FIFO.

Failure to set the EOT fields will lead to no Channel Transfer event being generated when the last byte is pushed onto the FIFO, which can lead to the Receiver not being informed about a pending Transfer when using the FIFO Transport protocol.

## B8.1.1 Transfer Delineation Mode

$I_{\text{XVVQK}}$      The SOT and EOT flags are used to mark the boundaries of a Transfer in a stream of bytes.

$I_{\text{SRCLS}}$      The MHU supports three modes of using the SOT and EOT flags called Transfer Delineation modes. Each mode of Transfer Delineation has a different intended usage listed below:

- Software flag
- Partial flag
- Auto flag

$R_{\text{NJNYP}}$      The PFFCW<n>_CTRL.TDM field controls which Transfer Delineation mode is used.

$I_{\text{VBFVJ}}$      Depending on the transport protocol being used to send the Transfer, refer to  *D2.2 FIFO* or  *D2.3 Streaming FIFO* for information on:

- Which Transfer Delineation mode to use.

- How to use each Transfer Delineation mode.

### B8.1.1.1  Software flag

$R_{PMHZG}$   When software flag Transfer Delineation mode is used the flags are managed by the Sender and the values of the PFFCW<n>_FLG.{SOT/EOT} fields will not be updated by the MHU.

$R_{HJSVC}$   Software flag Transfer Delineation mode is selected by setting PFFCW<n>_CTRL.TDM to `0b00`.

### B8.1.1.2  Partial Flag

$R_{SZXNH}$   When partial flag Transfer Delineation mode is used the flags are managed jointly by the Sender and the MHU, with the values of the PFFCW<n>_FLG.{SOT,EOT} fields being updated by both.

$R_{YMBCQ}$   Partial flag Transfer Delineation mode is selected by setting the PFFCW<n>_CTRL.TDM is set to `0b01`.

$R_{PKXTW}$   When partial flag Transfer Delineation mode is selected the SOT and EOT flags behave as follows:

- When a write occurs to PFFCW<n>_CTRL.TDM which sets it to `0b01`, the PFFCW<n>_FLG.{SOT,EOT} fields are set to `0b1` and `0b0` respectively.
- The PFFCW<n>_FLG.{SOT/EOT} fields can only be written by the Sender if at least one of the fields will be set to `0b1`, otherwise the write will not update the SOT and EOT fields but will update other fields in the register.
- When one or more bytes are pushed onto the FIFO the values of the SOT and EOT fields change as shown in Table B8.1.

**Table B8.1: SOT and EOT flag values after a push of one or more bytes to the FIFO when using partial Transfer Delineation mode**

| Value of data flags of before push | | Value of data flags for after push | |
|---|---|---|---|
| SOT | EOT | SOT | EOT |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |

$I_{VZDVP}$   The PFFCW<n>_FLG.ACK field behavior is not affected by using the partial flag Transfer Delineation mode. The value of ACK field is always the value last written by the Sender or its reset value.

$S_{NQJSJ}$   When the Sender wants to update the value of the PFFCW<n>_FLG.ACK field the Sender must set the SOT and EOT fields to `0b0`. This will cause only the ACK field to be updated.

Failure to do this can lead to data loss or corruption when the Receiver reads the data from the FIFO as the SOT or EOT fields may not follow a valid sequence.

### B8.1.1.3  Auto Flag

$R_{WWQWV}$   When auto flag Transfer Delineation mode is used the flags are managed by the MHU, with the values of the PFFCW<n>_FLG.{SOT,EOT} fields are updated only by the MHU.

$R_{ZLYXR}$   Auto flag Transfer Delineation mode is selected by setting the PFFCW<n>_CTRL.TDM is set to `0b10`.

$R_{\text{ZPJFJ}}$    When auto flag Transfer Delineation mode is selected the SOT and EOT flags behave as follows:

- When a writes occurs to the PFFCW<n>_CTRL.TDM field which sets it to `0b10`:

  - The PFFCW<n>_FLG.{SOT,EOT} fields become read-only.
  - The PFFCW<n>_FLG.{SOT,EOT} fields are set to `0b1` and `0b0` respectively.

- When one or more bytes are pushed onto the FIFO the value of the SOT and EOT fields toggle.

$I_{\text{VWRKS}}$    The PFFCW<n>_FLG.ACK field behavior is not affected by using the auto flag Transfer Delineation mode. The value of ACK field is always the value last written by the Sender or its reset value.

## B8.1.2  Data Flag Storage

$R_{\text{DBBMR}}$    It is IMPLEMENTATION DEFINED how the data flags are stored alongside the Transfer data but the rules in this section must be followed.

$R_{\text{GQXLH}}$    The relationship between data flags and the bytes they are associated with is determined at the point when the write to the PFFCW<n>_PAY register occurs. It must never be possible for the association between data flags and byte to be altered.

$R_{\text{ZYGSL}}$    Data flags and the associated Transfer data byte must be read, written and popped from the FIFO atomically.

## B8.2 FIFO Behavior

$R_{MYCDY}$    It is IMPLEMENTATION DEFINED how the FIFO is implemented.

### B8.2.1 Reset

$I_{RPVZT}$    It is IMPLEMENTATION DEFINED whether the FIFO is part of the MHUS or MHUR or split between the two. This includes any logic which is used to control the locations within the FIFO where new data is pushed to or data is read or popped from, for example read and write pointers.

$R_{VVTNW}$    When either the MHUS or MHUR is in a Non-operational state, all bytes in the FIFO become invalid and contain an UNKNOWN value.

> **Note**
>
> $R_{VVTNW}$ only applies for controlled entry into the Non-operational state. For an uncontrolled entry of the MHUS or MHUR into a Non-operational state $R_{FHLRF}$ takes precedence.

$R_{TGLBZ}$    When the MHUS is in a Non-operational state, no bytes can be pushed onto the FIFO.

$S_{QXJGK}$    The Sender must be capable of handling the loss of Transfers due to the MHUR entering a Non-operational state in a controlled manner. For example, the Receiver sets the MBX_CTRL.CH_OP_MSK field to `0b1`, when there are outstanding Transfers in the FIFO. Arm recommends that the Receiver indicates to the Sender when it wants to enter a Non-operational state before it sets the MBX_CTRL.CH_OP_MSK field to `0b1` and starts to enter the MHUR into a Non-operational state.

$S_{FTWHV}$    The Receiver must be capable of handling spurious interrupts where the contents of the FIFO are lost due to the Sender entering a Non-operational state whilst the FIFO contains one or more valid bytes. The Receiver can use the MFFCW<n>_FLG.VFLG<m> fields to know whether a previous read from the FIFO returned any bytes with valid data.

$R_{VLXGB}$    Attempts to push bytes onto the FIFO when the MHUR is in a Non-operational state always generate a request to enter the MHUR into the Operational state. It is allowed for data to be pushed onto the FIFO, if it is possible whilst the MHUR is in a Non-operational state.

$R_{VPVBG}$    When the MHUR is in a Non-operational state no bytes can be read or popped from the FIFO.

### B8.2.2 Pushing data onto the FIFO

$R_{ZTYGZ}$    Bytes are pushed onto the FIFO when the Sender writes to the PFFCW<n>_PAY register.

$R_{LVCVN}$    When one or more bytes are pushed onto the FIFO, the value of the flags in the PFFCW<n>_FLG register, at the time of the write to the PFFCW<n>_PAY register, is recorded along with the bytes and stored in the FIFO.

$R_{DNVMP}$    The SOT flag is always associated with the first byte pushed onto the FIFO.

$R_{KNQYQ}$    The EOT and ACK flags are always associated with the last byte pushed onto the FIFO.

$R_{LBCSG}$    The number of bytes pushed onto FIFO is equal to the size of the write access to the PFFCW<n>_PAY register.

$R_{ZHYJT}$    On a write to the PFFCW<n>_PAY register, the following occurs:

- If the number of bytes to be pushed onto the FIFO is less than or equal to the space in the FIFO:
  - All bytes are pushed onto the FIFO.
  - PFFCW<n>_ST.PPE field is set to `0b0`.

- Update to the PFFCW<n>_FLG.{SOT,EOT} fields based on the Transfer Delineation Mode selected by the PFFCW<n>_CTRL.TDM field.

- If the number of bytes to be pushed onto the FIFO is greater than the space in the FIFO:

  - No bytes are pushed onto the FIFO.
  - PFFCW<n>_ST.PPE field is set to 0b1.
  - No update to the PFFCW<n>_FLG.{SOT,EOT} fields are made.

$S_{NBZZY}$  Arm recommends that software either:

- Never attempts to push more bytes onto the FIFO than the value of the PFFCW<n>_ST.FFS field indicates are free.
- Checks the PFFCW<n>_ST.PPE field after each write to the PFFCW<n>_PAY register and if it is set to 0b1 repeat the write.

Otherwise writes to the FIFO without sufficient space could result in data loss.

$R_{YQLGF}$  The order in which bytes are pushed onto the FIFO depends on the setting of the PFFCW<n>_CTRL.MSBF field when the write occurs.

$I_{LWCLD}$  For more information on how the PFFCW<n>_CTRL.MSBF field affect the order in which bytes are pushed onto the FIFO refer to B8.2.6 *Ordering of bytes when push, read or popping multiple bytes*.

## B8.2.3  Reading data from the FIFO

$R_{GCZCC}$  The Receiver reads the contents of the FIFO using the MFFCW<n>_PAY register.

$R_{RHCQK}$  When one or more bytes are read from the FIFO by the Receiver reading the MFFCW<n>_PAY register, the flags associated with those bytes read are stored in the Flag History Buffer (FHB).

$I_{CSZMC}$  The entries of the FHB are used in subsequent reads of the MFFCW<n>_FLG register.

$R_{XVYRH}$  A byte and the associated flags must be read from the FIFO atomically.

---

**Note**

It is possible that whilst reading data from the FIFO any of the following occurs:

- FIFO Flush.
- MHUS or MHUR enter a Non-operational state.

All of the above conditions cause all bytes in the FIFO to become invalid. Depending on the implementation of the FIFO, it may be possible that one or more of the bytes and associated flags being atomically read from the FIFO are still to be read. These bytes are now invalid and are treated as such in the returned read data and in the associated entries in the FHB. $R_{Qxscs}$ describes the behavior for reading bytes which are invalid from the FIFO.

---

$I_{VTDGV}$  When reading multiple bytes from the FIFO it is possible that the bytes belong to one or more Transfers. This can make processing Transfers more complicated as the bytes from a later Transfer need to be buffered, when Read-Acknowledge is enabled, as the act of reading the bytes causes them to be popped from the FIFO. To reduce the software complexity, FFCHs support automatically buffering future Transfer data, referred to a Future Transfer Auto Buffering.

$R_{DBKWN}$  Future Transfer Auto Buffering is enabled using the MFFCW<n>_CTRL.FTAB field.

$R_{MLQJF}$  Future Transfer Auto Buffering is only enabled for the FIFO, when Read-Acknowledge is also enabled for the FFCH.

$R_{CFJRX}$  When MFFCW<n>_CTRL.FTAB is set to 0b0 or MFFCW<n>_CTRL.RA_EN is set to 0b0, no buffering of Transfers occurs.

---

$R_{SSPSL}$     When MFFCW<n>_CTRL.FTAB is set to `0b1` and MFFCW<n>_CTRL.RA_EN is `0b1,` buffering of future of Transfers occurs.

$R_{XPVVK}$     The number of bytes read from the FIFO when the Receiver performs a read access to the MFFCW<n>_PAY register depends on:

- The value of the MFFCW<n>_CTRL.RA_EN field.
- The value of the MFFCW<n>_CTRL.FTAB field.
- The size of the read access.
- Number of valid bytes in the FIFO.

Table B8.2 shows the number of bytes read from the FIFO when the Receiver performs a read access to the MFFCW<n>_PAY register.

**Table B8.2: Number of bytes read when the Receiver performs a read access to the MFFCW<n>_PAY register**

| RA_EN | FTAB | Read Access Size (Bytes) | Bytes Read |
|---|---|---|---|
| 0 | X | 1 | min(1,FFL) |
|  |  | 2 | min(2,FFL) |
|  |  | 4 | min(4,FFL) |
|  |  | 8 | min(8,FFL) |
| 1 | 0 | 1 | min(1,FFL) |
|  |  | 2 | min(2,FFL) |
|  |  | 4 | min(4,FFL) |
|  |  | 8 | min(8,FFL) |
| 1 | 1 | 1 | min(1,FFL,FBE) |
|  |  | 2 | min(2,FFL,FBE) |
|  |  | 4 | min(4,FFL,FBE) |
|  |  | 8 | min(8,FFL,FBE) |

$I_{JQDFS}$     In Table B8.2:

- min() is a function which takes two or more arguments and returns the argument with the smallest value.
- FFL is the current number of valid bytes in the FIFO.
- FBE is the byte number of the first byte in the FIFO which is associated with an EOT flag, starting from 1 for the first byte read.

$I_{FPYXF}$     A byte is said to be valid in the FIFO if it has been pushed onto the FIFO by the Sender and not yet popped from the FIFO by the Receiver. Refer to $R_{VQQLV}$ for more information on when bytes are valid or invalid.

$R_{QXSCS}$     If a read of the MFFCW<n>_PAY register results in less bytes being read from the FIFO than requested by the read, the following occurs:

- Byte offsets of the read data, which does not contain data for a byte read from the FIFO, are set to an UNKNOWN value.
- Entries in the FHB, for a byte which has its value set to an UNKNOWN value, are set to be invalid.

For more information on FHB entries refer to B8.2.4 *Flag History Buffer*.

$I_{JFBHD}$    This can be due to either:

- The FIFO contain less valid bytes than the requested number of bytes.
- The number of bytes remaining in the Transfer were less than the size of the read, as indicated by the last byte of the Transfer having the EOT flag set to `0b1`, and the MFFCW<n>_CTRL.RA_EN and MFFCW<n>_CTRL.FTAB fields were both set to `0b1`.

$I_{JGWBQ}$    The byte offsets within the read data returned from the MFFCW<n>_PAY register depend on the setting of the MFFCW<n>_CTRL.MSBF field. For more information refer  B8.4.1 *Accesses to Payload and Flag registers*.

## B8.2.4  Flag History Buffer

$I_{GQBNG}$    The Flag History Buffer (FHB), holds the values of flags for the bytes read from the FIFO by the last read operation of the MFFCW<n>_PAY register.

$R_{RFGHK}$    At reset all entries in the FHB are invalid.

$R_{SDCQK}$    The FHB is considered part of the MHUR reset domain.

$R_{HDDTJ}$    The FHB buffers up to N entries, where N is the maximum size of a read access supported to the MFFCW<n>_PAY register.

$R_{JNFBJ}$    FHB entries are numbered 0 to N-1.

$I_{WQCPC}$    In this architecture FHB entries are referred to as FHB[x] where x is the FHB entry number in the range 0 to N-1, for example FHB[0] refers to the first FHB entry.

$R_{MXWTV}$    Each entry contains the following:

- Valid byte indicator.
- SOT field.
- EOT field.

$R_{TJFTS}$    When an entry is valid the SOT and EOT must contain the value of the Data Flag for the byte it is associated with.

$R_{YWMSM}$    When an entry is invalid it is UNKNOWN what value the SOT and EOT fields take in the entry.

$R_{YTJQG}$    The FHB contents are updated each time a read of the MFFCW<n>_PAY register occurs, replacing any previous data that was in the buffer.

$R_{DFLXG}$    The entries within the FHB are populated in the order in which the bytes are read from the FIFO, with the first byte read being stored in FHB[0].

$R_{FNNMX}$    When a read access to the MFFCW<n>_PAY register returns less bytes than there are entries in the FHB any remaining entries in the FHB are set to be invalid.

$I_{KVPRM}$    This can occur due to one of the following reasons:

- Size of the last read access to the MFFCW<n>_PAY register being smaller than maximum supported size read.
- The FIFO contained less valid bytes than the requested by the last read of the MFFCW<n>_PAY register.
- The number of bytes remaining in the Transfer were less than the size of the read of the MFFCW<n>_PAY register, as indicated by the last byte of the Transfer having the EOT flag set to `0b1`, and the MFFCW<n>_CTRL.RA_EN and MFFCW<n>_CTRL.FTAB fields both being set to `0b1`.

## B8.2.5  Popping bytes from the FIFO

$R_{SYTTK}$    Bytes are popped from the FIFO when the FIFO contains at least one byte and the Receiver performs one of the following actions:

- Reads from the MFFCW<n>_PAY register and MFFCW<n>_CTRL.RA_EN is set to `0b1`.

• Writes to the MFFCW<n>_FIFO_POP.POP field and MFFCW<n>_CTRL.RA_EN is set to `0b0`.

$R_{VLSBF}$ The number of bytes popped from the FIFO when the Receiver performs an action which would cause bytes to be popped from the FIFO is the smallest value of the following:

• The number of bytes requested by the Receiver.
• The number of bytes in the FIFO.
• Whether a previous byte popped from the same request had the EOT field set, when the MFFCW<n>_CTRL.RA_EN and MFFCW<n>_CTRL.FTAB fields are both set to `0b1`.

$R_{KQYYB}$ Table B8.3 shows the number of bytes popped from the FIFO when the Receiver performs a read of the MFFCW<n>_PAY register, based on:

• Value of MFFCW<n>_CTRL.RA_EN field.
• Value of MFFCW<n>_CTRL.FTAB field.
• Size of the read access to the MFFCW<n>_PAY register.

**Table B8.3: Number of bytes popped from the FIFO when the Receiver performs a read of the MFFCW<n>_PAY register**

| RA_EN | FTAB | Access Size (Bytes) | Bytes Popped from FIFO |
|---|---|---|---|
| 0 | X | X | 0 |
| 1 | 0 | 1 | min(1,FFL) |
| | | 2 | min(2,FFL) |
| | | 4 | min(4,FFL) |
| | | 8 | min(8,FFL) |
| 1 | 1 | 1 | min(1,FFL,FBE) |
| | | 2 | min(2,FFL,FBE) |
| | | 4 | min(4,FFL,FBE) |
| | | 8 | min(8,FFL,FBE) |

$I_{CDKDQ}$ In Table B8.3:

• min() is a function which takes two or more arguments and returns the argument with the smallest value.
• FFL is the current number of valid bytes in the FIFO.
• FBE is the byte offset, with respect to the head of the FIFO, of the first byte which is associated with an EOT flag.

$R_{PHJLP}$  Table B8.4 shows the number of bytes popped from the FIFO when the Receiver performs a write to the MFFCW<n>_FIFO_POP.POP field, based on:

- Value of MFFCW<n>_CTRL.RA_EN field.
- Value of MBX_FFCH_CFG0.M64BA_SPT field.
- Value written to the MFFCW<n>_FIFO_POP.POP field

**Table B8.4: Number of bytes popped from the FIFO when the Receiver performs a write of the MFFCW<n>_FIFO_POP.POP field**

| RA_EN | Pop Field | M64BA_SPT | M32BA_SPT | M16BA_SPT | M8BA_SPT | Bytes Popped from FIFO |
|---|---|---|---|---|---|---|
| 1 | X | X | X | X | X | 0 |
| 0 | 0 | X | X | X | 0 | IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• Treated as if the Pop value was another supported value. |
| | | X | X | X | 1 | min(1,FFL) |
| | 1 | X | X | 0 | X | IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• Treated as if the Pop value was another supported value. |
| | | X | X | 1 | X | min(2,FFL) |
| | 2 | X | X | X | X | IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• Treated as if the Pop value was another supported value. |
| | 3 | 1 | 0 | X | X | IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• Treated as if the Pop value was another supported value. |
| | | X | 1 | X | X | min(4,FFL) |
| | 4-6 | X | X | X | X | IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• Treated as if the Pop value was another supported value. |

| RA_EN | Pop Field | M64BA_SPT | M32BA_SPT | M16BA_SPT | M8BA_SPT | Bytes Popped from FIFO |
|---|---|---|---|---|---|---|
| | 7 | 0 | 1 | X | X | IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• Treated as if the Pop value was another supported value. |
| | | 1 | X | X | X | min(8,FFL) |

$I_{JLKPS}$  In  Table B8.4:

- min() is a function which takes two or more arguments and returns the argument with the smallest value.
- FFL is the current number of valid bytes in the FIFO.
- FBE is the byte offset, with respect to the head of the FIFO, of the first byte which is associated with an EOT flag.

$I_{CZQVY}$  Some combinations of settings in  Table B8.4 are not show or are illegal as the architecture requires that either M32BA_SPT or M64BA_SPT or both are set to 0b1.

$R_{GVJHV}$  For POP field values in  Table B8.4 which are marked as IMPLEMENTATION DEFINED and the implementation selects to treat the POP field value as another supported value it is allowed to pop more or less bytes.

$S_{HVVKD}$  Software must only write values to the MFFCW<n>_FIFO_POP.POP field which are supported as indicated by the MBX_FFCH_CFG0.{M8BA_SPT/M16BA_SPT/M32BA_SPT/M64BA_SPT} fields, otherwise it can lead to corruption or lost of Transfer data.

## B8.2.6  Ordering of bytes when push, read or popping multiple bytes

$I_{FKYTL}$  It is possible for the Sender to push multiple bytes onto the FIFO and the Receiver to read or pop multiple bytes from the FIFO using a single access. The data is stored to and read or popped from the FIFO one byte at a time.

$R_{GFMXZ}$  The order in which multiple bytes are stored in the FIFO is determined by the value of the PFFCW<n>_CTRL.MSBF field, at the time the push occurs.

$R_{QWNCR}$  The order in which the bytes are read or popped from the FIFO is determined by the value of the MFFCW<n>_CTRL.MSBF field at the time the read or pop occurs.

$R_{HNHCT}$  Depending on the value of the PFFCW<n>_CTRL.MSBF field bytes are pushed onto the FIFO:

- 0b0 - Least-Significant Byte (LSB) first.
- 0b1 - Most-Significant Byte (MSB) first.

$R_{QPRYY}$  Depending on the value of the MFFCW<n>_CTRL.MSBF field the first byte read from the FIFO:

- 0b0 - is considered the LSB.
- 0b1 - is considered the MSB.

$I_{LXMKH}$  The value of MFFCW<n>_CTRL.MSBF has no effect on the order in which flags are stored in the FHB. For more information on the order in which flags are stored in the FHB refer to  B8.2.4 *Flag History Buffer*.

$R_{WDWXS}$  FFCHs are considered little endian where the LSB is in the lowest offset within the access and the MSB is in the highest offset within the access.

$S_{SQRDD}$  The Sender and Receiver must set the {PFFCW/MFFCW}<n>_CTRL.MSBF fields to the same value for the same FFCH otherwise the order of the bytes in the Transfer will be observed differently by the Receiver to the order the Sender sent them in.

## B8.3  FIFO Flush

$I_{PQCGK}$    Each FFCH has a mechanism to return it to a known state, know as FIFO flush. The FIFO flush can be triggered by either the Sender or Receiver.

$S_{DJZQX}$    The FIFO flush is not intended to be used during normal operation but as a method to restore the FIFO back to a known state when unexpected events occur. For example, either the MHUS or MHUR enter a Non-operational state in an uncontrolled manner.

$R_{WJXWB}$    Each FFCH has two independent FIFO flush mechanisms, one in the MHUS and one in the MHUR, which operate independently of one another. It is valid for both mechanism to be trigger at the same time.

$R_{LWVXX}$    Each FFCH's flush mechanisms are independent of one another, meaning multiple FFCHs can be flushed at once.

$R_{ZQDZN}$    The FIFO flush mechanism in the MHUS is controlled is PFFCW<n>_CTRL.FF field and the status is shown in the PFFCW<n>_ST.FF field.

$R_{QHMJV}$    The FIFO flush mechanism in the MHUR is controlled is MFFCW<n>_CTRL.FF field and the status is shown in the MFFCW<n>_ST.FF field.

$I_{NZFTD}$    For the remainder of sections:

- <x>FFCW<n>_CTRL.FF is used when the statement could apply to either the PFFCW<n>_CTRL.FF or MFFCW<n>_CTRL.FF fields.
- <x>FFCW<n>_ST.FF is used when the statement could apply to either the PFFCW<n>_ST.FF or MFFCW<n>_ST.FF fields.

However, the remainder of this section should be read from the prospective of single flush mechanism. For example, if a flush is in progress by the flush mechanism of the MHUS, PFFCW<n>_CTRL.FF is set to 0b1, and a write of 0b0 to the MFFCW<n>_CTRL.FF occurs this has no effect on the flush in progress by the flush mechanism of the MHUS. The same applies the other way round.

$R_{GPKVB}$    When a flush is triggered, by software writing 0b1 to <x>FFCW<n>_CTRL.FF field, the following steps takes places:

- All bytes within the FIFO becomes invalid.
- The location to read or pop the next byte is set to same location as the location to push the next byte to.
- Any IMPLEMENTATION DEFINED state which software is unable to return to the state it was in immediately after reset, is returned back to its reset state.

The order in which the above steps take place is IMPLEMENTATION DEFINED.

Once all the steps for the flush have taken place the <x>FFCW<n>_ST.FF field is set to 0b1.

$I_{LVSXC}$    Other state of the FFCH is also affected by the side effects of the flush. The PFFCW<n>_ST.FFS and MFFCW<n>_ST.FFL fields report the number of invalid and valid bytes in the FIFO respectively, their values will change to reflect all bytes being made invalid.

$R_{NNNCQ}$    Even though the FIFO flush mechanism makes all bytes in the FIFO become invalid it does not generate a Sender or Receiver FIFO Low Tide event.

$S_{FYLPK}$    If software is using the Sender or Receiver FIFO Low Tide events to know when the FIFO has less than a certain value of valid bytes, the Sender or Receiver FIFO Flush events should be used to know when a flush of the FIFO has occurred and the the number of valid bytes is less than low tide value.

$R_{DSYPS}$    When software sets the <x>FFCW<n>_CTRL.FF field to 0b0 and the <x>FFCW<n>_ST.FF field is set to 0b1, the <x>FFCW<n>_ST.FF field is set to 0b0 at the same time.

$R_{TCCLM}$    If software sets the <x>FFCW<n>_CTRL.FF field to 0b0 when that FIFO flush mechanism is performing a flush and the <x>FFCW<n>_ST.FF field is set to 0b0 it is IMPLEMENTATION DEFINED whether:

- The FIFO flush mechanism completes the current FIFO flush or aborts. If the FIFO flush completes the <x>FFCW<n>_ST.FF field is set to 0b0 when it completes.

           • Generates the Sender or Receiver FIFO flush event.

$R_{CRDXR}$    If, due to the <x>FFCW<n>_CTRL.FF field being set to `0b0`, an in-progress FIFO flush mechanism is aborted it must leave the FIFO in a state whereby it is possible to send new Transfers between the Sender and Receiver.

$R_{YQVWP}$    It is CONSTRAINED UNPREDICTABLE whether a new FIFO flush is generated if the <x>FFCW<n>_CTRL.FF field is set to `0b1` when <x>FFCW<n>_ST.FF is not `0b0`.

$R_{MGMGV}$    The time taken for all the steps of the FIFO flush to take place is IMPLEMENTATION DEFINED, but it must be complete within a bounded period of time.

$R_{VMKYT}$    It is CONSTRAINED UNPREDICTABLE whether a push operation which starts or is outstanding whilst a FIFO flush is in progress is treated as completing before or after the FIFO flush request.

$R_{RRGLX}$    It is CONSTRAINED UNPREDICTABLE whether a read or pop operation which starts or is outstanding whilst a FIFO flush is in progress is treated as completing before or after the FIFO flush request.

$S_{ZSSSP}$    It is software responsibility to sequence flushing of the FIFO with respect to any outstanding push, read or pop operation.

$R_{BXGXC}$    A FIFO flush request triggered in the MHUS does not cause the MHUR to enter an Operational state, if the MHUR is in a Non-operational state. If, to complete all steps required for a FIFO flush, the MHUR is required to enter the Operational state the following applies:

        • The steps are skipped or delayed until the MHUR returns to the Operational state for another reason.
        • When the MHUR returns to the Operational state the steps are performed, if required.
        • The location to read or pop the next byte is set to one of the following:

                • The location of the first valid byte in the FIFO, if one or more bytes have been pushed onto the FIFO after the flush completed.
                • The same as the location to push the next byte to, if no bytes have been pushed onto the FIFO after the flush completed.

$R_{HKJMV}$    A FIFO flush request trigger in the MHUR does not cause the MHUS to enter an Operational state, if the MHUS is in a Non-operational state. If, to complete all steps required for a FIFO flush, the MHUS is required to enter the Operational state the following applies:

        • The steps are skipped or delayed until the MHUS returns to the Operational state for another reason.
        • When the MHUS returns to the Operational state the steps are performed, if required.

$I_{MYRWD}$    When referring to bytes being pushed onto the FIFO after the flush completed, this includes bytes pushed onto the FIFO due to push operations which started during or were outstanding when the FIFO flush event and were treated as if they occurred after the FIFO flush.

$S_{XXPRP}$    The <x>FFCW<n>_CTRL.FF and <x>FFCW<n>_CTRL.FF fields form a four-phase handshake between the Sender or Receiver and the MHUS and MHUR respectively.

     Software must follow this sequence otherwise it is UNPREDICTABLE whether the the flush completes all steps required.

    1. Set <x>FFCW<n>_CTRL.FF to `0b1`.
    2. Poll <x>FFCW<n>_ST;.FF until it reads as `0b1`.
    3. Set <x>FFCW<n>_CTRL.FF to `0b0`.
    4. Poll <x>FFCW<n>_ST.FF until it reads as `0b0`.

$S_{QTDMT}$    Arm recommends software never sets the <x>FFCW<n>_CTRL.FF field to `0b1` when <x>FFCW<n>_ST.FF is not `0b0`.

$S_{WHZXJ}$    It is the responsibility of the Sender or Receiver or both to:

        • Clear any outstanding interrupts.
        • Return any previously programmed fields to their reset values, if required.

## B8.4 FIFO Channel Window

R$_{\text{NHHSG}}$    The Sender and Receiver use the FIFO Channel Window (FFCW) to access the registers of the FFCH.

R$_{\text{VVNNQ}}$    The FFCW occupies 16 consecutive words.

R$_{\text{CJBQX}}$    Table B8.5 shows the registers in the FFCW when accessed through the PBX:

**Table B8.5: FFCW when accessed from PBX**

| Offset | Name | Access |
|---|---|---|
| `0x00` | PFFCW<n>_PAY | RW |
| `0x08` | PFFCW<n>_FLG | RW |
| `0x0C` | Reserved | |
| `0x10` | PFFCW<n>_INT_ST | RO |
| `0x14` | PFFCW<n>_INT_CLR | WO/RAZ |
| `0x18` | PFFCW<n>_INT_EN | RW |
| `0x1C` | Reserved | |
| `0x20` | PFFCW<n>_CTRL | RW |
| `0x24` | PFFCW<n>_ST | RO |
| `0x28` | PFFCW<n>_ACK_CNT | RO |
| `0x2C` | PFFCW<n>_TIDE | RW |
| `0x30` | Reserved | |
| `0x34` | Reserved | |
| `0x38` | Reserved | |
| `0x3C` | Reserved | |

R$_{\text{VPTCQ}}$    Table B8.6 shows the registers in the FFCW when accessed through the MBX:

**Table B8.6: FFCW when accessed from MBX**

| Offset | Name | Access |
|---|---|---|
| `0x00` | MFFCW<n>_PAY | RO |
| `0x08` | MFFCW<n>_FLG | RO |
| `0x10` | MFFCW<n>_INT_ST | RO |
| `0x14` | MFFCW<n>_INT_CLR | WO/RAZ |
| `0x18` | MFFCW<n>_INT_EN | RW |
| `0x1C` | Reserved | |
| `0x20` | MFFCW<n>_CTRL | RW |
| `0x24` | MFFCW<n>_ST | RO |

| Offset | Name | Access |
|--------|------|--------|
| `0x28` | MFFCW&lt;n&gt;_FIFO_POP | WO |
| `0x2C` | MFFCW&lt;n&gt;_TIDE | RW |
| `0x30` | Reserved | |
| `0x34` | Reserved | |
| `0x38` | Reserved | |
| `0x3C` | Reserved | |

### B8.4.1 Accesses to Payload and Flag registers

$I_{BCNFY}$ This section provides information on accesses to the PFFCW&lt;n&gt;_PAY, MFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_FLG registers.

$I_{PKKSX}$ The PFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_PAY registers are windows which allow the Sender and Receiver to interact with the contents of the FIFO as follows:

- A write to the PFFCW&lt;n&gt;_PAY register pushes more data onto the FIFO.
- A read of the MFFCW&lt;n&gt;_PAY register reads and optionally pops data from the FIFO.
- A read of the MFFCW&lt;n&gt;_FLG register provides the values of the flags stored in the FHB for the last read of the MFFCW&lt;n&gt;_PAY register.

$R_{ZGDML}$ The PFFCW&lt;n&gt;_PAY, MFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_FLG registers are 64-bits.

$I_{HYCNN}$ The size of the access to the: PFFCW&lt;n&gt;_PAY, MFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_FLG registers effect:

- PFFCW&lt;n&gt;_PAY register determines the amount of data pushed onto the FIFO.
- MFFCW&lt;n&gt;_PAY register determines the amount of data read or popped from the FIFO.
- MFFCW&lt;n&gt;_PAY register determines the number of flags which are return.

An implementation of the MHU can support 8-, 16-, 32- or 64-bit accesses to the PFFCW&lt;n&gt;_PAY, MFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_FLG registers. For more information on the supported accesses sizes to the PFFCW&lt;n&gt;_PAY, MFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_FLG registers refer to  C1.1 *Register Access*.

---

**Note**

In this architecture the terms:

- PFFCW&lt;n&gt;_PAY{8/16/32/64} are used to refer to 8-, 16-, 32-, 64-bit accesses to the PFFCW&lt;n&gt;_PAY register respectively.
- MFFCW&lt;n&gt;_PAY{8/16/32/64} are used to refer to 8-, 16-, 32-, 64-bit accesses to the MFFCW&lt;n&gt;_PAY register respectively.
- MFFCW&lt;n&gt;_FLG{8/16/32/64} are used to refer to 8-, 16-, 32-, 64-bit accesses to the MFFCW&lt;n&gt;_FLG register respectively.

---

$S_{BGGWD}$ Software can use the:

- PBX_FFCH_CFG0.{P8BA_SPT/P16BA_SPT/P32BA_SPT/P64BA_SPT} fields, to determine the supported access sizes to the PFFCW&lt;n&gt;_PAY register.
- MBX_FFCH_CFG0.{M8BA_SPT/M16BA_SPT/M32BA_SPT/M64BA_SPT} fields, to determine the supported access sizes of the MFFCW&lt;n&gt;_PAY and MFFCW&lt;n&gt;_FLG registers.

$S_{WYQBC}$    When accessing the PFFCW<n>_PAY, MFFCW<n>_PAY and MFFCW<n>_FLG registers software must:

- Only access the registers with a supported access size.
- Have the access aligned to a boundary equal to the size of the access.

Otherwise it can lead to data lost or corruption.

$S_{XXXQB}$    Software must never generate an access to the PFFCW<n>_PAY/MFFCW<n>_PAY registers which has a start address and size which would cause locations allocated outside the PFFCW<n>_PAY, MFFCW<n>_PAY or MFFCW<n>_FLG registers to be accessed, otherwise it is can lead to lost of data.

### B8.4.1.1 PFFCW<n>_PAY register

$I_{KXDWQ}$    This sections covers the behavior when the the Sender accesses the PFFCW<n>_PAY register.

$R_{PPVGB}$    There is no requirement for the read or write accesses to always target any specific offset(s) in the PFFCW<n>_PAY register, for accesses with a smaller size than PFFCW<n>_PAY.

$I_{VQRRM}$    Writes to the PFFCW<n>_PAY register behave as defined in B8.2.2 *Pushing data onto the FIFO*.

$R_{GLTWK}$    Reads of the PFFCW<n>_PAY register return the value of the PFFCW<n>_ST.{PPE,FFS} fields.

---

**Note**

The value of the PFFCW<n>_ST.FFS field returned when performing a read of the PFFCW<n>_PAY register can be truncated to fit within the size of the read to the PFFCW<n>_PAY register.

---

$R_{MTGRD}$    Table B8.7 shows the arrangement of the PFFCW<n>_ST.{PPE,FFS} fields in the returned data for a read of the PFFCW<n>_PAY register depending the size of the read access.

**Table B8.7: Returned read data for a read of PFFCW<n>_PAY register depending on size of the read access**

| Size of read of PFFCW<n>_PAY register | Returned re ad data bit offset | Field of PFFCW<n>_ST register | Note |
|---|---|---|---|
| 8-bit | 7 | PPE | |
| | 6:0 | FFS | If value of FFS field is greater than 127 the value in the returned read data is set to 127 |
| 16-bit | 15 | PPE | |
| | 10:0 | FFS | |
| 32-bit | 31 | PPE | |
| | 10:0 | FFS | |
| 64-bit | 63 | PPE | |
| | 10:0 | FFS | |

$R_{QKPFW}$    All bit offsets not listed in Table B8.7 are reserved and treated as RES0.

### B8.4.1.2 MFFCW<n>_PAY register

$I_{QBWLR}$  The behavior of reads to the MFFCW<n>_PAY register behave as defined in B8.2.3 *Reading data from the FIFO*.

$R_{HCKPR}$  There is no requirement for the read accesses to always target any specific offset(s) in the MFFCW<n>_PAY register, for accesses with a smaller size than MFFCW<n>_PAY register.

### B8.4.1.3  MFFCW<n>_FLG register

$I_{FRCPT}$  This section covers reads of the MFFCW<n>_FLG register.

$R_{KKNXY}$  Read accesses to the MFFCW<n>_FLG register returns:

- The SOT and EOT flags for the bytes last read from the FIFO and stored in the FHB, by the last read access to the MFFCW<n>_PAY register.
- A valid flag field per each set of flag returned.
- The value of the MFFCW<n>_ST.FFL field.

---

**Note**

The value of the MFFCW<n>_ST.FFL field returned when performing a read of the MFFCW<n>_FLG register can be truncated to fit within the size of the read to the MFFCW<n>_FLG register.

---

$R_{ZQBCD}$  When a read of the MFFCW<n>_FLG register occurs, the entries in the FHB are used to generate the read response as follows:

- When MFFCW_CTRL.MSBF is `0b0`:

    - FHB[0] is associated with FLG0 and VFLG0 fields.
    - FHB[x] is associated with FLGx and VFLGx fields.

- When MFFCW_CTRL.MSBF is `0b1`:

    - FHB[0] is associated with FLGx and VFLGx fields.
    - FHB[x] is associated with FLG0 and VFLG0 fields.

Where x is equal to the size of the read access to the MFFCW<n>_FLG registers minus 1.

$S_{JFXLH}$  Software must:

- Access the MFFCW<n>_FLG register with the same size of access as it did to read the data from the MFFCW<n>_PAY register.
- Use the same value of MFFCW<n>_CTRL.MSBF to access the MFFCW<n>_PAY and MFFCW<n>_FLG register.

Otherwise it can lead to loss or corruption of information.

$S_{KTTFC}$  Software must use the values of MFFCW<n>_FLG.{VFLG,FLG}<m> fields, to know whether:

- The byte previously read from the MFFCW<n>_PAY register associated with the flag is valid.
- The byte was considered the start of a new Transfer, if it is valid.
- The byte was considered the end of a Transfer, if it is valid.

Invalid bytes and flags must not be used by software, otherwise, Transfer data as read by the Receiver, will include bytes which were not pushed onto the FIFO by the Sender.

$R_{LWJQC}$  Table B8.8 shows the format of the returned read data, depending on the size of the read access, for a read of the MFFCW<n>_FLG register.

**Table B8.8: Returned read data for a read of MFFCW<n>_FLG register depending on size of the read access**

| Size of read of MFFCW<n>_FLG register | Returned read data bit offset | Description | Note |
|---|---|---|---|
| 8-bit | 7:4 | MFFCW<n>_ST.FFL | If the value of FFL field is greater than 15 the value in the returned r ead data is set to 15. |
| | 2 | Valid Flag 0 | |
| | 1:0 | Flag 0 | |
| 16-bit | 15:8 | MFFCW<n>_ST.FFL | If the value of FFL field is greater than 255 the value in the returned read data is set to 255. |
| | 7 | Valid Flag 1 | |
| | 6:4 | Flag 1 | |
| | 2 | Valid Flag 0 | |
| | 3:0 | Flag 0 | |
| 32-bit | 31:21 | MFFCW<n>_ST.FFL | |
| | 14 | Valid Flag 3 | |
| | 13:12 | Flag 3 | |
| | 10 | Valid Flag 2 | |
| | 9:8 | Flag 2 | |
| | 6 | Valid Flag 1 | |
| | 5:4 | Flag 1 | |
| | 2 | Valid Flag 0 | |
| | 1:0 | Flag 0 | |
| 64-bit | 63:53 | MFFCW<n>_ST.FFL | |
| | 30 | Valid Flag 7 | |
| | 29:28 | Flag 7 | |
| | 26 | Valid Flag 6 | |
| | 25:24 | Flag 6 | |
| | 22 | Valid Flag 5 | |
| | 21:20 | Flag 5 | |
| | 18 | Valid Flag 4 | |
| | 17:16 | Flag 4 | |
| | 14 | Valid Flag 3 | |
| | 13:12 | Flag 3 | |
| | 10 | Valid Flag 2 | |
| | 9:8 | Flag 2 | |
| | 6 | Valid Flag 1 | |

| Size of read of MFFCW<n>_FLG register | Returned read data bit offset | Description | Note |
|---|---|---|---|
| | 5:4 | Flag 1 | |
| | 2 | Valid Flag 0 | |
| | 1:0 | Flag 0 | |

**Note**

The association between Flag and the FHB entries depends on the value of the MFFCW<n>_CTRL.MSBF field at the time of the read of the MFFCW<n>_FLG register.

$R_{MYWXN}$  All bit offsets not listed in  Table B8.8 are reserved and treated as RES0.

## B8.4.2  FIFO Status

$I_{KKHHC}$  The PFFCW<n>_ST and MFFCW<n>_ST registers provided the Sender and Receiver respectively with information on the FIFO.

### B8.4.2.1  PFFCW<n>_ST register

$R_{VGRKS}$  The PFFCW<n>_ST register provides the following fields:

- FFS field - Number of free bytes in the FIFO.
- PPE field - Whether the last attempt to push data onto the FIFO was successful or not.

$R_{NMGFD}$  The PFFCW<n>_ST.FFS field indicates the number of invalid bytes in the FIFO. The behavior of the FFS field is as follows:

- When a FIFO flush occurs is set to the depth of the FIFO.
- Decremented by one for every byte which is pushed onto the FIFO.
- Incremented by one for every byte which is popped from the FIFO.

$R_{WGXZJ}$  The behavior of the PFFCW<n>_ST.PPE field is as follows:

- Reset to 0b0.
- Set to 0b0 when a write to the PFFCW<n>_PAY register successfully pushes all bytes onto the FIFO.
- Set to 0b1 when a write to the PFFCW<n>_PAY register does not successfully push all bytes onto the FIFO.

$I_{CYBWG}$  The conditions that can cause one or more bytes to not be pushed onto the FIFO are defined in  B8.2.2 *Pushing data onto the FIFO* and  C1.1 *Register Access*.

$R_{YNZJP}$  The PPE field is not affected by an access which fails the security checks if TZE or RME are implemented.

### B8.4.2.2  MFFCW<n>_ST register

$R_{QMLML}$  The MFFCW<n>_ST.FFL field indicates the number of valid bytes in the FIFO. The behavior of the FFL field is as follows:

- FIFO flush occurs is set to 0.
- Incremented by one for every byte is pushed onto the FIFO.
- Decremented by one from every byte which is popped from the FIFO.

### B8.4.3 Transfer Acknowledge Tracking

$I_{NSLSC}$    As it is possible for the Sender to send multiple Transfer without waiting for a previous Transfer to be acknowledged, it is also possible for the Receiver to acknowledge multiple Transfer in the time taken for the Sender to detect this. To allow the Sender to keep track of how many Transfers have been acknowledged the Sender can use the PFFCW<n>_ACK_CNT register which counts the number of Transfer, which had the ACK flag set to `0b1` for the last byte of the Transfer, which have been popped from the FIFO by the Receiver.

$R_{VYFBW}$    For every FIFO Pop Ack event generated PFFCW<n>_ACK_CNT.ACK_CNT field is incremented by 1.

$I_{HRQTQ}$    The FIFO Pop Ack event indicates when the last byte of the Transfer has been popped from the FIFO and the byte had the ACK flag set. For more information on the FIFO Pop Ack event refer to B8.5 *Events*

$R_{QXLLR}$    The max value of the PFFCW<n>_ACK_CNT.ACK_CNT field is $2^{clog2((FFCH\_DEPTH/MTS)+1)}$-1, where MTS is the minimum size of a Transfer in bytes which the Sender can send. The minimum size of a Transfer depends on the value of the P{8/16/32/64}BA_SPT and the allowed values are shown in Table B8.9

**Table B8.9: Formula used to calculate minimum Transfer size in bytes**

| P8BA_SPT | P16BA_SPT | P32BA_SPT | P64BA_SPT | Minimum Transfer Size (Bytes) |
|----------|-----------|-----------|-----------|-------------------------------|
| 1 | X | X | X | 1 |
| 0 | 1 | X | X | 2 |
| 0 | 0 | 1 | X | 4 |
| 0 | 0 | 0 | 1 | 8 |

$R_{HWFNN}$    If when the PFFCW<n>_ACK_CNT.ACK_CNT field is incremented and the value overflows the PFFCW<n>_ACK_CNT.ACK_CNT_OVRFLW field is set to `0b1`.

$R_{JGWTP}$    It is UNPREDICTABLE whether the value of the PFFCW<n>_ACK_CNT.ACK_CNT is incremented, if between the generation of the a FIFO Pop Ack event and the event being counted, the MHUR enters a Non-operational state.

$R_{MDTHX}$    On a read of the PFFCW<n>_ACK_CNT register, the values of the ACK_CNT and ACK_CNT_OVRFLW fields are set to `0x0` and `0b0` respectively if there is no needed to update the ACK_CNT due to FIFO Pop Ack events. If there is a need to update the counter due to FIFO Pop Ack events the ACK_CNT field is set to the number of FIFO Pop Ack events and the ACK_CNT_OVRFLW field is set to `0b0`.

## B8.5 Events

$I_{YVJVC}$    Each FFCH is associated with the following events:

- Channel Transfer event
- FIFO Pop Ack event
- Channel Transfer Acknowledge event
- Sender FIFO Low Tide event
- Sender FIFO High Tide event
- Receiver FIFO Low Tide event
- Receiver FIFO High Tide event
- Sender FIFO flush event
- Receiver FIFO flush event

### B8.5.1 Channel Transfer event

$R_{BRRKX}$    A Channel Transfer event is generated when one or more bytes are pushed onto the FIFO and the PFFCW<n>_FLG.EOT field was set to `0b1` for one of those bytes.

### B8.5.2 FIFO Pop Ack event

$R_{KJLVT}$    When one or more bytes are popped from the FIFO for each byte popped which has the ACK and EOT fields both set to `0b1` a FIFO Pop Ack event is generated.

### B8.5.3 Channel Transfer Acknowledge event

$R_{MJDDH}$    A Channel Transfer Acknowledge event is generated when the PFFCW<n>_ACK_CNT.ACK_CNT field was zero and then becomes non-zero.

### B8.5.4 Sender FIFO Low and High Tide events

$R_{KGWZM}$    A Sender FIFO Low Tide event is generated when one or more bytes are popped from the FIFO and both the following are true:

- The number of valid bytes in the FIFO before the pop operation was greater than the value of the PFFCW<n>_TIDE.LOW field.
- The number of valid bytes in the FIFO after the pop operation is less than or equal to the value of the PFFCW<n>_TIDE.LOW field.

$R_{XGHGJ}$    A Sender FIFO High Tide event is generated when one or more bytes are pushed onto the FIFO and both the following are true:

- The number of valid bytes in the FIFO before the push operation was less than or equal the value of the PFFCW<n>_TIDE.HIGH field.
- The number of valid bytes in the FIFO after the push operation is greater than the value of the PFFCW<n>_TIDE.HIGH field.

$I_{LXYBS}$    The value of PFFCW<n>_TIDE.{HIGH/LOW} used in the calculations includes any offsets of the field which are RES0 due to the configuration of the MHU.

$I_{HZKYC}$  The intended uses of the Sender FIFO Low and High Tide events are:

- FIFO Low Tide event indicates to the Sender that there is N or more bytes free in the FIFO. It can use this to know when it can push more bytes onto the FIFO, for example it is possible to push all bytes of a new Transfer onto the FIFO.
- FIFO High Tide event indicates to the Sender that there is less than N bytes free in the FIFO. It can use this to know when it should stop pushing more bytes onto the FIFO, for example it is not possible to push a all bytes of a new Transfer onto the FIFO.

Where N is equal to the FIFO Depth minus the value of the PFFCW<n>_TIDE.{LOW/HIGH} field respectively.

## B8.5.5 Receiver FIFO Low and High Tide events

$R_{XHDQD}$  A Receiver FIFO Low Tide event is generated when one or more bytes are popped from the FIFO and both the following are true:

- The number of valid bytes in the FIFO before the pop operation was greater than the value of the MFFCW<n>_TIDE.LOW field.
- The number of valid bytes in the FIFO after the pop operation is less than or equal to the value of the MFFCW<n>_TIDE.LOW field.

$R_{LYXWB}$  A Receiver FIFO High Tide event is generated when one or more bytes are pushed onto the FIFO and both the following are true:

- The number of valid bytes in the FIFO before the push operation was less than or equal the value of the MFFCW<n>_TIDE.HIGH field.
- The number of valid bytes in the FIFO after the push operation is greater than the value of the MFFCW<n>_TIDE.HIGH field.

$I_{NWJFL}$  The value of MFFCW<n>_TIDE.{HIGH/LOW} used in the calculations includes any offsets of the field which are RES0 due to the configuration of the MHU.

$I_{TYDNB}$  The intended uses of the Receiver FIFO Low and High Tide events are:

- FIFO Low Tide event indicates to the Receiver that there is N or less bytes used in the FIFO. It can use this to know when it can stop processing data in the FIFO, for example there is not a complete Transfer in the FIFO.
- FIFO High Tide event indicates to the Receiver that there is less than N bytes free in the FIFO. It can use this to know when it should start processing data in the FIFO, for example there is a complete Transfer in the FIFO.

Where N is equal to the value of the MFFCW<n>_TIDE.{LOW/HIGH} field respectively.

## B8.5.6 Sender FIFO flush event

$R_{ZGNXL}$  A Sender FIFO flush event is generated, by the MHUS, when all the following are true:

- A FIFO flush, performed by the mechanisms in the MHUR completes, and the MFFCW<n>_ST.FF field is set to `0b1`.
- MHUS is in an Operational state.

## B8.5.7 Receiver FIFO flush event

$R_{WMMSD}$  A Receiver FIFO flush event is generated, by the MHUR, when all the following are true:

- A FIFO flush, performed by the mechanisms in the MHUS completes, and the PFFCW<n>_ST.FF field is set to `0b1`.
- MHUR is in an Operational state.

# Chapter B9
# TrustZone and Realm Management Extension

## B9.1 Overview

$I_{XMGNY}$  TrustZone (TZE) and Realm Management (RME) extensions enable an MHU to be integrated into a system with an Arm processor that supports multiple Security states and for the MHU to restrict which Security states can access the MHU resources.

$I_{NSNHG}$  The Realm Management Extension defined by the Arm A-profile architecture v9 is referred to as FEAT_RME.

$I_{TSNFQ}$  The following sections cover the rules of TZE and RME.

$R_{YWLRF}$  The MHUS and MHUR can have different support for the TZE and RME extensions.

$I_{HBSFJ}$  Arm recommends that TZE and RME are implemented to match the features of the systems into which the MHUS or MHUR are to be integrated. For example, an Application system communicating with a System Control Processor (SCP) in full-duplex mode, where the Application system includes an Armv9-A processor with support for FEAT_RME and the System Control Processor includes an Armv8-M processor without support for Arm® TrustZone® for Armv8-M. Table B9.1 shows the configuration of TZE and RME for both MHUS and MHUR of the two MHUs.

**Table B9.1: Example configuration of a pair of MHUs, used in full duplex mode, between an Application and System Control Processor**

| Communication Link | MHUS/MHUR | TZE | RME |
|---|---|---|---|
| Application -> SCP | MHUS | Yes | Yes |
| Application -> SCP | MHUR | No | No |

| Communication Link | MHUS/MHUR | TZE | RME |
|---|---|---|---|
| SCP -> Application | MHUS | No | No |
| SCP -> Application | MHUR | Yes | Yes |

Where -> indicates the direction in which Transfers are sent.

$I_{QCFNF}$ Both TZE and RME define the following concepts:

- All accesses to the registers of the MHUS and MHUR have a security property.

  The security property indicates the security of the access.

- Security Group.

  There is a Security Group per each security world which is implemented. Resources of the MHU are then allocated to a Security Group, by software.

## B9.2 Security Assignment Agent

$D_{SSQNS}$      When TZE or RME is implemented, introduces a software agent called the Security Assignment Agent.

$I_{HPZXL}$      There is a Security Assignment Agent for both the MHUS and MHUR, depending on whether TZE or RME are implemented for MHUS or MHUR.

$I_{JQTBK}$      The task of the Security Assignment Agent is to assign the resources of the MHUS or MHUR to the different Security Groups.

$R_{JKGJD}$      The Security Assignment Agent can only allocate the Postbox or Mailbox, including all channels in the MHU, to a specific Security Group.

$I_{RWMJZ}$      Arm recommends that the assignment of Postbox and Mailbox to a Security Group is not changed dynamically.

$S_{KTLGX}$      Software is responsible for handling the security implications if it re-assigns the Postbox or Mailbox to a different Security Group. This includes completing the Transfers for any outstanding Transfer and preventing any leakage of data between security worlds.

## B9.3 TZE

R$_{\text{YPHCW}}$    The rules in this section only apply when TZE is implemented by the MHUS or MHUR.

R$_{\text{NRMQT}}$    When TZE is implemented, there are two Security Groups defined:

- Secure
- Non-secure

R$_{\text{QYLVY}}$    When TZE is implemented, the Postbox and Mailbox of the MHU can be assigned to either the Secure Security Group or the Non-secure Security Group.

R$_{\text{YLVRN}}$    At reset, the Postbox and Mailbox are assigned to the Secure Security Group.

R$_{\text{WTHVC}}$    When TZE is implemented for the MHUS, the MHUS includes a Sender Security Control (SSC) block.

R$_{\text{JNPTV}}$    When TZE is implemented for the MHUR, the MHUR includes a Receiver Security Control (RSC) block.

I$_{\text{ZWKMN}}$    The SSC and RSC blocks provide registers to configure the Security Group which the Postbox or Mailbox belongs to.

R$_{\text{FZMMY}}$    The SSC and RSC blocks are only accessible by Secure accesses.

R$_{\text{SZGSW}}$    Any Non-secure access to either the SSC or RSC blocks is treated as an illegal access. For more information on the behavior of illegal accesses refer to  C1.1.5 *Illegal Accesses*.

R$_{\text{KJHJD}}$     Table B9.3 shows whether an access is allowed to access the registers of the Postbox or Mailbox depending on the security of the access and the Security Group which the Postbox or Mailbox is assigned to.

**Table B9.2: Whether an access is allowed to access the registers the Postbox or Mailbox when TZE is implemented and not RME**

| Security of access | Postbox/Mailbox Security Group | Access Allowed |
|---|---|---|
| Secure | Any | No |
| Non-secure | Secure | No |
|  | Non-secure | Yes |

Any access which is not allowed is treated as an illegal access. For more information on the behavior of illegal accesses refer to  C1.1.5 *Illegal Accesses*.

# B9.4 RME

R<sub>DSCBJ</sub>    When RME is implemented, TZE is also implemented.

R<sub>CPSSP</sub>    The rules in this section only apply when RME is implemented.

I<sub>RZSDD</sub>    RME extends the capabilities of TZE to support the four security worlds defined by Realm Management Extension (RME) of the Armv9-A architecture.

R<sub>HMLSB</sub>    RME adds two additional Security Groups:

- Root Security Group.
- Realm Security Group.

R<sub>TRVBR</sub>    At reset, the Postbox and Mailbox are assigned to the Root Security Group.

R<sub>XCPJW</sub>    The SSC and RSC block are enhanced to enable selection between the four Security Groups.

R<sub>RMHPN</sub>    The registers of the SSC and RSC blocks are accessible only by the Root Security Group.

R<sub>HZQTH</sub>    Any access to the registers of either the SSC or RSC blocks from the Secure, Non-secure or Realm security worlds are treated as an illegal access. For more information on the behavior of illegal accesses refer to  C1.1.5 *Illegal Accesses*.

R<sub>BZTSK</sub>    Table B9.3 shows whether an access is allowed to access the registers of the Postbox or Mailbox depending on the security of the access and the Security Group which the Postbox or Mailbox is assigned to.

**Table B9.3: Whether an access is allowed to access the registers the Postbox or Mailbox when RME is implemented**

| Security of access | Postbox/Mailbox Security Group | Access Allowed |
| --- | --- | --- |
| Root | Any | Yes |
| Realm | Root | No |
|  | Realm | Yes |
|  | Secure | No |
|  | Non-secure | Yes |
| Secure | Root | No |
|  | Realm | No |
|  | Secure | Yes |
|  | Non-secure | Yes |
| Non-secure | Root | No |
|  | Realm | No |
|  | Secure | No |
|  | Non-secure | Yes |

Any access which is not allowed is treated as an illegal access. For more information on the behavior of illegal accesses refer to  C1.1.5 *Illegal Accesses*.

R<sub>CVZBG</sub>    When the MHUS or MHUR implement RME, it is IMPLEMENTATION DEFINED whether the MHUS or MHUR implements a tie-off signal called **LEGACY_TZ_EN**.

$R_{YJPBH}$     The sampled value of the:

- **LEGACY_TZ_EN** for the MHUS only effects the behavior of the MHUS.
- **LEGACY_TZ_EN** for the MHUR only effects the behavior of the MHUR.

$R_{TLKPF}$     The value of **LEGACY_TZ_EN** signal for the MHUS, where present, is sampled at de-assertion of the reset of the MHUS.

$R_{TRBKT}$     The value of **LEGACY_TZ_EN** signal for the MHUR, where present, is sampled at de-assertion of the reset of the MHUR.

$I_{KLSHD}$     It is valid for the sampled **LEGACY_TZ_EN** value to be different between different between de-assertions of the reset of the MHU.

$R_{SXXLF}$     When the sampled value of **LEGACY_TZ_EN** tie-off is `0b1`, RME is disabled and the MHUS or MHUR behaves as if RME was not implemented.

This includes:

- Modifying the behavior of the registers and fields to behave as if RME was not implemented.

  All <x>_FEAT_SPT0.RME_SPT fields must read as `0x0`, where <x> is one of the following register block prefixes:

  - PBX
  - MBX
  - SSC
  - RSC

- Any security access check only checks whether the access is considered to be secure or non-secure.

  - Any access which is Root is considered to be Secure.
  - Any access which is Realm is considered to be Non-secure.

$R_{GTCLM}$     When the sampled value of **LEGACY_TZ_EN** tie-off is `0b1`, it does not remove support for TZE.

$I_{PPLSD}$     As RME requires TZE to be implemented when RME is disabled, via the sampled value of the **LEGACY_TZ_EN** being `0b1`, the MHUS or MHUR behaves as if it only implemented TZE.

$R_{ZTRJX}$     It is IMPLEMENTATION DEFINED whether fields which are defined as part of RME are writeable or not when **LEGACY_TZ_EN** sampled value is `0b1`. If the field is writeable the value of the field has no effect on the operation of the MHU and only be used to return a value when the field is read.

# Chapter B10
# Reliability, Availability and Serviceability Extensions

## B10.1 Overview

$I_{JTDZS}$   Reliability, Availability and Serviceability Extensions (RASE) defines a framework for how RAS features can be included in a MHU implementation. Unlike other extensions whereby the architecture defines a set of features, RASE is an IMPLEMENTATION DEFINED features due to the highly implementation specific nature of the type of errors which can be detected. The architecture mandates a set of requirements which must be followed.

$I_{DSFTF}$   Arm recommends that the implementor implements RAS features as defined in B10.7 *Recommend implementation of RAS using Arm RAS extensions*.

$R_{KTBFC}$   When RASE is implemented, the Sender RAS (SRAS) and Receiver RAS (RRAS) blocks are implemented.

$X_{CBSRW}$   The RAS register block is implemented as a separate 64KB block to allow the software agent which is acting on the RAS events to be independent of the Sender or Receiver using the Postbox or Mailbox.

$S_{NWRZG}$   Software can discover whether RASE is implemented by reading the value of the RASE_SPT field in the following registers:

- PBX_FEAT_SPT0.
- MBX_FEAT_SPT0.
- SSC_FEAT_SPT0.
- RSC_FEAT_SPT0.

$R_{CPFDQ}$   The contents of the SRAS and RRAS blocks are IMPLEMENTATION DEFINED.

## B10.2 Reporting Requirements

$R_{TRPDL}$     When an error is detected it must be reported. The mechanism by which it is reported and the information that is logged about the error is IMPLEMENTATION DEFINED.

$R_{GBFDV}$     It is IMPLEMENTATION DEFINED whether an attempt is made to correct the error. Refer to  B10.3 *Implications of error detection* for more information on the actions allowed to be taken to correct an error.

$I_{FDJGW}$     The MHU is formed of two components, MHUS and MHUR which is used by two different software entities the Sender and Receiver. The Sender uses the MHUS to send Transfers and the Receiver uses the MHUR to receive the Transfers. To transfer a Transfer from the Sender to the Receiver, it requires logic in both the MHUS and MHUR to function correctly. Certain errors in one component can lead to incorrect operation that may only be detected by the other component or the software entity using the other component.

For example, corruption of the data inside the FIFO could occur and would be detected when the Receiver reads data from the FIFO. The Receiver will be informed of the data corruption and would considered the Transfer to be invalid. There are cases where the Sender would also want to be informed of the corrupted data so that it could re-send the Transfer.

$R_{DQRFH}$     Errors that are detected and can only effect the Sender or Receiver are only reported by the RAS logic in the MHUS or MHUR respectively. Examples of errors which only effect the Sender or Receiver are:

- Error of the Acknowledge Counter logic of the FIFO only affects the Sender and is only needed to be reported by the MHUS RAS logic.
- Error of the Sender or Receiver Tidemark logic only effects the Sender or Receiver respectively and is only needed to be reported by the MHUS RAS logic for errors with the Sender Tidemark logic or the MHUR RAS logic for errors with the Receiver Tidemark logic.
- Error of interrupt generation logic only effects the Sender or Receiver, and is only needed to be reported by the MHUS RAS logic for errors with interrupt generation logic of the MHUS or the MHUR RAS logic for errors with the MHUR interrupt generation logic.
- Corruption of any payload data which has been corrected is only required to be reported in the MHUS RAS if the correction is performed on writing the data or MHUR if the correction is performed on reading the data.

$R_{DYFSP}$     Errors that are detected and can effect both the Sender or Receiver, even if they will only effect the one of them, are reported by the RAS logic of both MHUS and MHUR. Examples of errors which can effect both the Sender and Receiver:

- Corruption of any payload data which has not been corrected.
- Corruption of the FIFO read or write logic.

$I_{QCPXR}$     Detected errors can be specific to a specific part of the MHU. For example, an error could be related to a specific channel or type of channels. This is referred to as the impact zone of the error.

$R_{VFBDR}$     The impact zone of an error detected in the MHU, is one of the following:

- Global - An error which effects either the whole MHU or when the MHU is unable to be identify or report which specific channel or types of channels of which are impacted.
- Channel - An error which effects a single channel of the MHU.
- Channel type - An error which effects all channels of a specific type.

$R_{RBLLF}$     Impact zone for an error which apply to multiple channels and are all of the same type are reported as either effecting all channels of that type or global.

$R_{HTKHB}$     Impact zone for an error which effects multiple channels of different types is reported as global.

$R_{LJWKG}$     Impact zone for an error which does not effect a specific channel or all channels of a specific type is reported as global.

$R_{VCSWD}$     An MHU implementation is only required to support the global impact zone.

$R_{WPMQX}$     It is IMPLEMENTATION DEFINED which impact zone is reported for a specific error.

$R_{KXLSV}$    It is not required that different types of errors are reported with the same impact zone or that the same type of error but for a different piece of logic be reported with the same impact zone. However, the same type of error with the same bit of logic must always be reported with the same impact zone.

For example, in an MHU which supports 2 DBCH and 1 FFCH a double bit error detected in the PDBCW0_ST or PDBCW1_ST must both be reported with the same impact zone, whilst a double bit error detected in the FIFO of the FFCH can be reported with a different impact zone.

## B10.3 Implications of error detection

R<sub>MBNJW</sub>    When an error is detected, it is IMPLEMENTATION DEFINED whether the MHU attempts to correct it. For example, the MHU may implement ECC for the FIFO and be able to recover from a single bit error without impacting the functionality of the MHU.

I<sub>WMVRK</sub>    Errors where the MHU is able to correct the error are referred to as corrected errors, whilst errors where the MHU is unable to correct the error is referred to as an uncorrected error.

I<sub>TLXVT</sub>    Deferred errors are errors which are detected but are not correct at that point in time. Deferred errors might later be consumed and at that point become either corrected or uncorrected errors.

R<sub>NGTZK</sub>    It is IMPLEMENTATION DEFINED whether the MHU takes any actions to recover from an uncorrected error or not, including taking actions which can cause the lost of Transfers which are outstanding at the point when the error was detected.

R<sub>XBJMW</sub>    If a MHU implementation does takes steps to recover from an uncorrectable error that causes the lost of Transfer data, the following applies:

- The error must be reported to both the Sender and Receiver using the MHUS and MHUR RAS logic, before any steps to correct the error are started.
- The Sender or Receiver must be informed that data has been lost using an IMPLEMENTATION DEFINED method.
- The lost of any Transfer data must never be wider than the reported impact zone of the error. For example, if the error is reported as being specific to FFCH0, then only Transfer data in FFCH0 is allowed to be lost. If Transfer data from another channel is lost then the error must be reported as either:
  - Impact all channels of that type.
  - Impact the entire MHU.

S<sub>RZHJT</sub>    Software must be prepared to perform an IMPLEMENTATION DEFINED sequence to recover from an uncorrectable error. Examples of these steps could be:

- Completing any outstanding Transfers with the channel which is effected, but discarding the Transfer data. After this resending any Transfers which have been discarded.
- Completing any outstanding Transfers with all channels that are effected, but discarding the Transfer data. After this resending any Transfers which have been discarded.
- Performing a reset of either of or both the MHUS or MHUR, before resending any Transfers outstanding when the error was detected or sending new Transfers.

S<sub>NXHVT</sub>    It is valid for the Sender or Receiver to be responsible for acting on RAS events reported via the SRAS/RRAS blocks or for another piece of software, RAS agent, to be responsible for this. If the RAS agent is present, it is the responsibility of the RAS agent to either:

- Implement a method by which the Sender or Receiver can discover that there has been a lost of data due to a RAS event.
- To perform an IMPLEMENTATION DEFINED sequence to handle the possible lost of data.

## B10.4 Error record requirements

$I_{GZZKY}$    An error record records information about an error which has been detected.

$R_{MHYHG}$    It is IMPLEMENTATION DEFINED whether on detection of an error an error record is generated. If no error record is generated then the behavior for all detected errors must be as follows:

- All errors are considered global errors.
- All errors are considered to have causes lost of Transfer data.

$R_{ZGQRP}$    If an error recorded is generated, the format of any error record is IMPLEMENTATION DEFINED but must include the following information:

- The impact zone of the error if impact zones other than global are supported.
- Whether the MHU has taken corrective action which has lead to the lost of Transfer data.

$R_{WJCXW}$    Whether an error recorded is generated or not for the error must be reported back to the Sender/Receiver.

## B10.5 RAS and Security

$R_{BNLSF}$   The rules in this section only apply:

- To the Sender RAS block when the MHUS implements TZE or RME.
- The the Receiver RAS block when the MHUR implements TZE or RME.

### B10.5.1 Error reporting

$I_{VGNVB}$   When TZE or RME is implemented in the MHUS or MHUR, the MHU enforces isolation between the different Security Groups. When RASE is also implemented this is extended to the reporting of RAS related errors. There is a strong industry requirement to allow RAS related events to be handled by the Kernel or Hypervisor running in the Non-secure world. To avoid breaking the security boundaries RAS error records must be sanitized to avoid leakage of information.

$I_{TVNFH}$   A RAS error is reported when an error is detected with a hardware resource which is related with a:

- Channel.
- Postbox or Mailbox.
- Sender or Receiver Security Control.

When the error is reported, information related to the error can be reported alongside the indication of the error in an error record. Information reported in an error record can be consider confidential information, depending on the Security Group of the hardware entity which generated the error and the security state of the memory access accessing the error record.

$D_{TQTPF}$   In this section, confidential information is defined as being information which would not be accessible to the software agent handling the RAS error if it attempted to access the information via the normal method. This comprises:

- Any data values which are part of a Transfer, include any contents held within the FIFO of a FFCH.
- Any data values of any control or status registers which is part of the Postbox or Mailbox.
- Any data values of any control or status registers which is part of the Sender or Receiver Security Control.

The following is not consider confidential information:

- Address offset of the register location which generated the error.
- Identifiers the channel number or type of channel.
- Information used to ascertain the severity of an error.

$R_{SJCMX}$   RAS error are associated with a Security Group.

$R_{BTGJN}$   The Security Group which an error is associated with depends on whether the error relates to a hardware resource associated with a:

- Channel.

  As channels are part of a Postbox and Mailbox, which could be assigned to different Security Groups, the Security Group of the error is determined as follows:

  - For an error reported in an error record in the Sender RAS block, the Security Group of the Postbox is used.
  - For an error reported in an error record in the Receiver RAS block, the Security Group of the Mailbox is used.

  If the error is reported in both the Sender and Receiver RAS blocks, this is consider two separate errors which are both treated separately and both have their own independent security.

- Register of a Postbox or Mailbox.

  The Security Group the Postbox or Mailbox is associated with.

• Register of the Sender or Receiver Security Control.

The Root Security Group, when RME is implemented, otherwise it is the Secure Security Group.

$R_{YCLRQ}$  There can be a time delay between when an error occurs until it is generated. In this time period the Security Group of the hardware resource which generated or caused the error may change. The Security Group associated with an error is taken to be the Security Group of the hardware resource which detects the error at the point it detects the error.

$S_{JJXZJ}$  Software is responsible for handling the security implications, when changing the Security Group of a resource within the MHU.

$R_{KHFVV}$  Confidential information, in an error record, must only be accessed by a security state which is of the same security state or higher. Table B10.1 shows the allowed security state of a memory access to the error record to be able to view confidential information of an error in the error record, depending on the security state associated with the error.

**Table B10.1: Confidential information access security**

| Error Security State | Allowed security state of access to the er ror record |
| --- | --- |
| Non-secure | Any |
| Secure | Secure or Root |
| Realm | Root or Realm |
| Root | Root |

$U_{GMPGB}$  A number of implementation options are possible to meet the requirements for reporting errors when TZE or RME are implemented:

• RAS error records contain no confidential information. For example, only containing information about the offset or Channel which detected the error.
• RAS error records filter the information based on the security of the error and the security of the access to the error record.
• RAS error records which contain confidential information are only accessible by:

  • Root, if RME is implemented.
  • Secure, if RME is not implemented.

**Note**

Arm strongly recommends against making error records only accessible by a Root or Secure access.

## B10.5.2 Error detection and reporting information

$R_{LSQJF}$  Whether error detection logic is enabled and what information is reported in an error record may be control via IMPLEMENTATION DEFINED registers. If these registers are implemented the following rules apply:

• When RME is implemented, controls must default to accessible only to Root accesses.
• When RME is not implemented, controls must default to accessible only to Secure accesses.

It is allowed for the Root or Secure state to assign these controls to another security state if it so wishes.

### B10.5.3 Error Injection

$R_{TTMGQ}$  If the implementation supports generating an error, as defined by the Common Fault Injection Model Extension in [2], on an access to a register of the MHU the error is only generated if the access passes the security checks defined in  Chapter B9 *TrustZone and Realm Management Extension*.

$R_{WMBCQ}$  An access which fails its security checks is considered an illegal access as defined in  Chapter B9 *TrustZone and Realm Management Extension*.

## B10.6  Interrupts

$R_{LFCKP}$     It is IMPLEMENTATION DEFINED whether one or more interrupts are generated when an error is detected. There can be different interrupts based on the type or severity of error detected and any corrective action taken.

$I_{XYLKZ}$     These interrupts are referred to a RAS interrupts in this architecture.

$R_{YCGXP}$     Any RAS interrupts must be a separate interrupt to any of the interrupts defined in  Chapter B11 *Interrupts*. It is allowed for an IMPLEMENTATION DEFINED interrupt to combined one or more RAS interrupts with one or more of the interrupts defined in  Chapter B11 *Interrupts*. This rule applies whether the combined interrupt is internal or external interrupt.

# B10.7 Recommend implementation of RAS using Arm RAS extensions

$I_{NRTXT}$    Arm recommends that an implementation of the MHU which implements RASE, follows the rules in this section.

$R_{RGNSY}$    The registers in the SRAS/RRAS block follow the format described in [2] for RAS registers.

$R_{LSCHM}$    Unless stated here the the rules of [2] defines the behavior of the RAS logic.

$I_{MTZRR}$    Arm recommends that FEAT_RASSA_ACR is implemented for the registers in the SRAS/RRAS block when either TZE or RME are implemented.

$X_{GDPKF}$    This allows for control of which security world can access the SRAS/RRAS blocks.

$R_{HHQDJ}$    When following all rules in this section the value of the RASE_SPT field is set to `0b0011`. Otherwise, it must be set to `0b0010` if RASE is implemented.

$I_{TVVJG}$    The ERR<n>MISC0[31:0] are used to convey information about the error. All of bits in ERR<n>MISC0 are as defined in [2].

$R_{CDFQG}$    ERR<n>MISC0[31:17] and ERR<n>MISC0[15] is RES0.

$R_{CPJZK}$    ERR<n>MISC0[16] indicates whether the error may have caused lost of Transfer data from the channels within the impact zone.

$R_{BCVNN}$    ERR<n>MISC0[14:13] is the IZ, Impact Zone, field and has the following values:

- `0b00` - Global.
- `0b01` - Channel specific.
- `0b10` - Channel Type.

$R_{SZCKK}$    ERR<n>MISC0[12:0] when IZ is `0b00` are RES0.

$R_{KBXVC}$    ERR<n>MISC0[12:0] when IZ is `0b01` are as follows:

- [12:10] - CT, Channel Type and has the following values:

    - `0b000` - DBCH.
    - `0b001` - FCH.
    - `0b010` - FFCH.

- [9:0] - CN, Channel Number.

$R_{JSHYC}$    ERR<n>MISC0[12:0] when IZ is `0b10` are as follows:

- [12:10] - CT, Channel Type and has the following values:

    - `0b000` - DBCH.
    - `0b001` - FCH.
    - `0b010` - FFCH.

- [9:0] - RES0.

$R_{NDMNJ}$    If the Common Fault Injection Model Extension defined in [2] is implemented, the value of the ERR<n>MISC0[31:0] bits are set as follows:

- If ERR<n>PFGF.{NA,MV} are set to any value other than `0b00` the value of ERR<n>MISCO[31:0] reads as `0x0000\_0000`, indicating the error is reported with a impact zone of global.
- If ERR<n>PFGF.{NA,MV} are set to `0b0` and `0b0` respectively the value in the ERR<n>MISC0[31:0] is set as follows, when an access to a register in the MHU occurs:

    - If the access is to a location, even a Reserved location, of an implemented Channel Window the error is recorded as an error with an impact zone of only the channel which was accessed. For example, an access to the PDBCW0_ST would be recorded as an error for the DBCH0.

- If the access is to a Reserved location outside an implemented Channel Window in PDBCW_page, PFFCW_page, PFCW_page of PDCW_page and the page is implemented, or any location in an unimplemented Channel Window, the error is recorded as having an impact zone effecting all channels of that type. For example, an access to MFFCW4_PAY when only a single FFCH is implemented is recorded with an impact zone effecting all FFCHs.
- The behavior of accesses to registers in the <x>_IMPL_DEF_pages is IMPLEMENTATION DEFINED. Arm recommends these are report as Global unless the register accessed controls features of a specific channel or channel type.
- If the access is to any other location the error is recorded with an impact zone of global.

$R_{MKSWQ}$   When reporting injected errors, if the impact zone is not supported by the implementation the impact zone is recorded as follows:

- Error should be reported with an impact zone of a specific channel is reported as being specific to a channel type or Global.
- Error should be reported with an impact zone of a channel type is reported as being Global.

# Chapter B11
# **Interrupts**

$I_{NGCPH}$    The MHU generates a number of events some of which are used to generate interrupts. These interrupts are then indicated to the Interrupt Controllers of the Sender and Receiver using a wired interrupt.

*Non-confidential*

# B11.1  Interrupt Types

$I_{XBJGB}$     The MHU supports the following interrupt types:

- Maskable interrupt
- Enable interrupt
- Shared enable interrupt
- Combined interrupt
- Combined status interrupt
- Combined enabled interrupt

$I_{DRGWB}$     Each type of interrupt defines the following:

- How events are used to determine whether to assert the interrupt or not.
- How software interacts with the interrupt, for example whether the interrupt has an enable and status field.

## B11.1.1  Maskable interrupt

$I_{RCXDJ}$     A maskable interrupt is an interrupt that has a mask which is used to select which events generate (unmasked) and do not generate (masked) an interrupt.

$R_{JRVTB}$     A maskable interrupt has the following registers:

- Mask status - Provides the status of the mask.
- Mask set - Allows setting of bits in the mask status register to `0b1`.
- Mask clear - Allows settings of bits in the mask status register to `0b0`.
- Status - Raw status.
- Status masked - Status with the mask applied.
- Set - Allows settings of bits in the status register to `0b1`.
- Clear - Allows settings of bits in the status register to `0b0`.

$R_{DPDMZ}$     A maskable interrupt is asserted when any bit of the status masked register is `0b1`.

$R_{RPKZT}$     The mask is set and cleared by writing `0b1` to the associated field of the mask set and clear register respectively.

$R_{GWYHJ}$     Setting of a field in the mask takes precedence over clearing the field.

$R_{CNQJS}$     Field <x> in the status register is set to:

- `0b1`, when a write to field <x> in the set register causes the event to be triggered.
- `0b0`, when a write to field <x> in the clear register causes any previously triggered events for that event to be cleared.

$R_{MMLJR}$     Setting of a field <x> in the status register takes precedence over clearing the field.

$R_{SZZVV}$     The status masked is generated by ANDing together the status and the inverted mask value.

$R_{PDJDQ}$     The interrupt is cleared by making all fields in the status masked register be `0b0`, by either:

- Setting all fields in the mask status register to `0b1`.
- Setting all fields in the status register to `0b0`.

$R_{KNWDD}$    Figure B11.1 shows the logic used to generated a maskable interrupt.
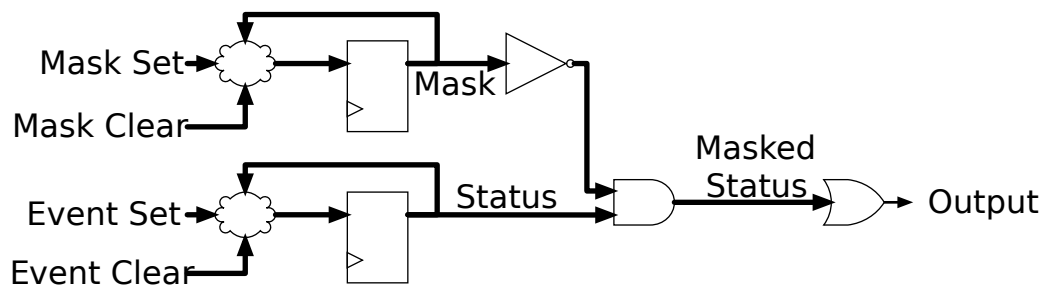


**Figure B11.1: Maskable interrupt logic**

---

**Note**

In Figure B11.1 the thick lines indicate buses and all logic on these buses is replicated for each bit except for the final output OR gate.

---

## B11.1.2  Enable interrupt

$I_{FLQDC}$    An enable interrupt is an interrupt that has an enable which selects whether the event(s) generate an interrupt or not.

$R_{JZWFR}$    An enable interrupt has the following fields:

- Interrupt status - Indicates whether the event has occurred.
- Interrupt enable - Controls whether the event should generate an interrupt.
- Interrupt clear - Allows software to clear the status bit.

$R_{GGJBD}$    The interrupt status field is:

- Set to $0b1$ when the event occurs and the enable bit is set to $0b1$.
- Set to $0b0$ when $0b1$ is written to the interrupt clear field.

$R_{ZHVKX}$    Set takes precedence over clear for the interrupt status field.

$R_{PDRRP}$    An enable interrupt is asserted when the status field is $0b1$.

$R_{BCZQZ}$    Figure B11.2 shows the logic used to generated an enable interrupt.
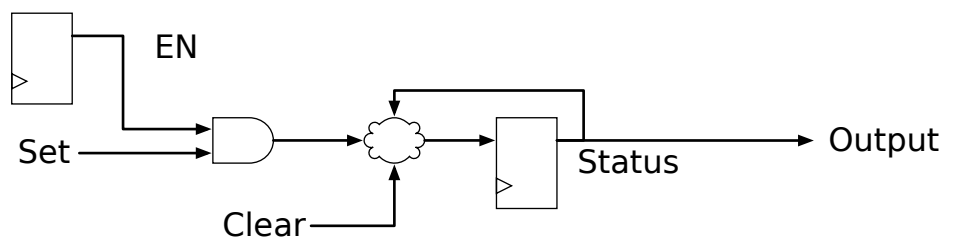


**Figure B11.2: Enable interrupt logic**

---

### B11.1.3  Shared enable interrupt

$I_{CTHFW}$      Shared enable interrupt are similar to enable interrupts except they:

- Have a shared interrupt enable field which is used to enable two or more shared enable interrupts.
- They have no software visible interrupt status field.
- They have no interrupt clear field.

$R_{LMGDF}$      A shared enable interrupt is asserted when the event occurs and the interrupt enable is set to `0b1`.

$R_{RXJSS}$      There is no dedicated clear register.

$R_{MHRPX}$      The interrupt is cleared by performing an action which acknowledges the interrupt.

$R_{RWRXR}$      Set takes precedence over clear for a shared enable interrupt.

$R_{DTZGB}$      There is no software visible status register.

$X_{TMMWJ}$      As there is an interrupt per shared enable interrupt there is no need for a dedicated status registers as when a shared enable interrupt output is asserted that is the indication of the source of the interrupt.

$R_{GCHGK}$      Figure B11.3 shows the logic used to generate a shared enable interrupt
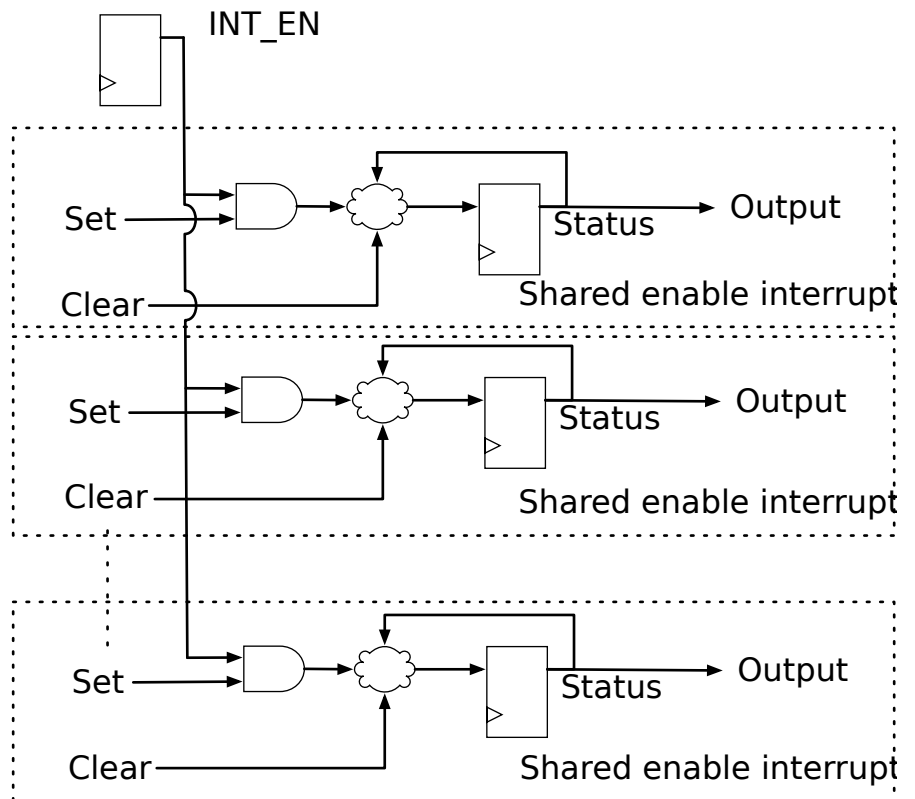


**Figure B11.3: Shared enable interrupt logic**

$I_{VTYJN}$      In Figure B11.3 more than one shared enable interrupt interrupt is shown to demonstrate that a single enable register is used to enable all shared enable interrupts.

## B11.1.4  Combined interrupt

$I_{MFQTF}$   A combined interrupt is a combination of several interrupts in the MHU to reduce the number of interrupts from the MHU.

$R_{WQWPZ}$   A combined interrupt has no dedicated status, enable or clear fields.

$R_{BGHBH}$   The interrupt is asserted when any of the interrupts combined by the interrupt are asserted.

$S_{TDDPT}$   Software uses the underlying fields of the interrupts which are combined as follows:

- Enable field to select whether the interrupt is enabled and contributes to the combined interrupt.
- Status field to identify which interrupt caused the combined interrupt to be asserted.
- Clear field to clear the interrupt.

$R_{XYKRT}$   Figure B11.4 shows the logic used to generate a combined interrupt.
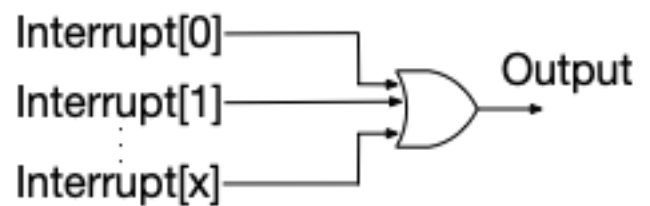
**Figure B11.4: Combined interrupt**

## B11.1.5  Combined status interrupt

$I_{SBVYK}$   A combined status interrupt is a combination of several interrupt in the MHU to reduce the number of interrupts from the MHU with a dedicated status field for each interrupt it combines.

$R_{QRKKR}$   A combined status interrupt has a dedicated status field for each interrupt it combines.

$R_{KSHKB}$   The status field is set to `0b1` when the interrupt it is associated with is asserted, otherwise it is set to `0b0`.

$R_{BQBNW}$   The combined status interrupt is asserted when any of the status fields are set to `0b1`.

$S_{VLJJG}$   Software clears a combined status interrupt by clearing all the interrupts which are combined together.

$R_{FRBYY}$   Figure B11.5 shows the logic used to generate a combined status interrupt.
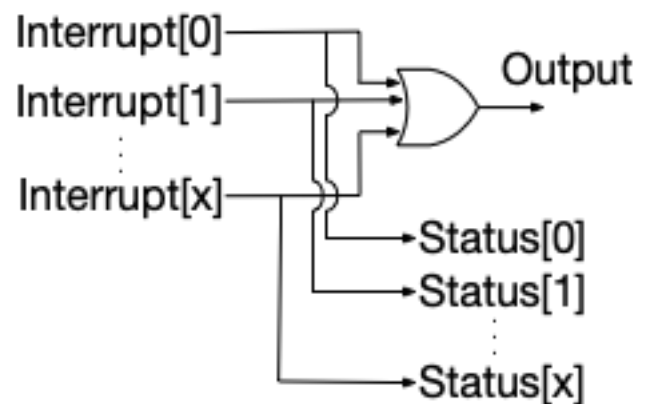
**Figure B11.5: Combined status interrupt**

## B11.1.6  Combined enable interrupt

$I_{KGVBP}$      Combined enable interrupts are similar to combined interrupts but have a dedicated enable and a status field per each interrupt which it combines.

$R_{YGZSY}$      There is an enable and status field for each interrupt which is combined together to form the combined enable interrupt.

$R_{HRKBT}$      A field in the status register is set to `0b1` when both the enable field is set to `0b1` and the interrupt the field is associated with is asserted, otherwise the field is set to `0b0`.

$R_{XTGTF}$      A combined enable interrupt is asserted when one or more of the status fields are `0b1`.

$S_{SLQSZ}$      Software uses the enable and status fields as follows:

- Enable

  Select which interrupt it receives via the combined enable interrupt.

- Status

  Identify which of the combined interrupts caused the combined enable interrupt to be asserted.

$S_{TFZMD}$      To clear a combined enable interrupt, software must do one of the following:

- Clear all interrupts which are combined by the combined enable interrupt.
- Set the enable for that interrupt to `0b0`.

$R_{QXLSZ}$      Figure B11.6 shows the logic used to generate a combined enable interrupt.
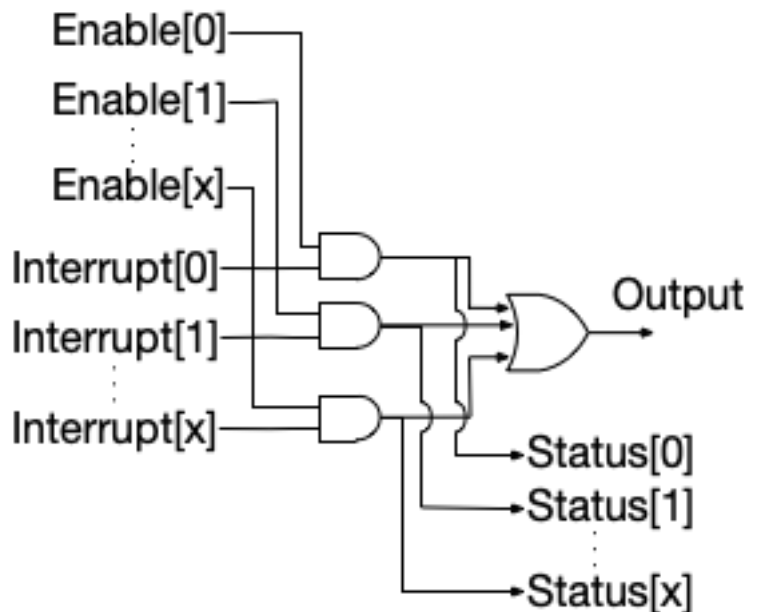


**Figure B11.6: Combined enable interrupt**

# B11.2 MHU Interrupts

$I_{RWFMY}$     This section covers the interrupts of the MHU.

$R_{DXFTW}$     All interrupts of the MHU are level-sensitive wired interrupts.

$R_{LGXZT}$     Interrupts of the MHU can be either:

- Internal

  Internal interrupts do not have an interrupt output and are only exposed to software via another interrupt which has an interrupt output.

- External

  External interrupts must have an interrupt output for software to receive the interrupt directly. External interrupts can also be used internally to generate other interrupts.

$I_{HYXJD}$     The MHU has the following interrupts:

- Channel Transfer
- Channel Transfer Acknowledge
- Fast Channel Transfer
- Fast Channel Group Transfer
- Sender FIFO Low Tidemark
- Sender FIFO High Tidemark
- Receiver FIFO Low Tidemark
- Receiver FIFO High Tidemark
- Sender FFCH Combined
- Receiver FFCH Combined
- Postbox Combined
- Mailbox Combined

The following sections provide details on each interrupt.

$I_{PTFZB}$     Chapter D1 *Postbox and Mailbox Interrupt Logic* shows the structure of both the Postbox and Mailbox interrupt generation logic showing all interrupts.

## B11.2.1 Channel Transfer

$I_{CZTZL}$     The Channel Transfer interrupt indicates when there is a new Transfer in DBCH or FFCH.

$R_{SMGPR}$     There is a Channel Transfer interrupt per each DBCH and FFCH implemented in the MHU.

$R_{QRCFQ}$     It is IMPLEMENTATION DEFINED whether the Channel Transfer interrupt is implemented as an internal or external interrupt.

$R_{NWSSD}$     If the Channel Transfer interrupt is implemented as an external interrupt the interrupt output is present on the MHUR.

$I_{MJRZC}$     The Channel Transfer interrupt behavior is dependent on the type of channel it is associated with.

### B11.2.1.1 DBCH

$R_{CVQXG}$     The Channel Transfer interrupt for a DBCH is a maskable interrupt and uses the Channel Transfer events of the DBCH as the event inputs.

$R_{XVGGL}$     The Channel Transfer interrupt for a DBCH has the following registers:

- MDBCW<n>_MSK_ST is the mask register.
- MDBCW<n>_MSK_SET is the mask set register.

- MDBCW<n>_MSK_CLR is the mask clear register.
- MDBCW<n>_ST is the status register.
- MDBCW<n>_ST_MSK is the status masked register.
- PDBCW<n>_SET is the set register.
- MDBCW<n>_CLR is the clear register.

### B11.2.1.2 FFCH

$R_{RHJGW}$ The Channel Transfer interrupt for an FFCH is an enable interrupt and uses the Channel Transfer event of the FFCH as the event input.

$R_{ZMFHH}$ The Channel Transfer interrupt for a FFCH uses the following fields:

- MFFCW<n>_INT_ST.TFR - interrupt status.
- MFFCW<n>_INT_EN.TFR - interrupt enable.
- MFFCW<n>_INT_CLR.TFR - interrupt clear.

## B11.2.2 Channel Transfer Acknowledge

$I_{TZMQG}$ The Channel Transfer Acknowledge interrupt indicates to the Sender when a Transfer in a DBCH or FFCH has been acknowledge by the Receiver.

$R_{QXNVJ}$ There is a Channel Transfer Acknowledge interrupt per each DBCH and FFCH implemented in the MHU.

$I_{CGJHF}$ FC do not have a Channel Transfer Acknowledge as it is considered a lossy channel and a Transfer does not require the Receiver indicates acknowledgement back to the Sender.

$R_{GQYPS}$ It is IMPLEMENTATION DEFINED whether the Channel Transfer Acknowledge interrupt is implemented as an internal or external interrupt.

$R_{CBKXM}$ If the Channel Transfer Acknowledge interrupt is implemented as an external interrupt the interrupt output is present on the MHUS.

$R_{CVGTW}$ The Channel Transfer Acknowledge interrupt is an enable interrupt and uses the Channel Transfer Acknowledge event of the DBCH or FFCH.

$R_{ZCTCL}$ The Channel Transfer Acknowledge interrupt for the DBCH uses the following fields:

- PDBCW<n>_INT_ST.TFR_ACK - interrupt status.
- PDBCW<n>_INT_EN.TFR_ACK - interrupt enable.
- PDBCW<n>_INT_CLR.TFR_ACK - interrupt clear.

$R_{GDHGF}$ The Channel Transfer Acknowledge interrupt for the FFCH uses the following fields:

- PFFCW<n>_INT_ST.TFR_ACK - interrupt status.
- PFFCW<n>_INT_EN.TFR_ACK - interrupt enable.
- PFFCW<n>_INT_CLR.TFR_ACK - interrupt clear.

## B11.2.3 Fast Channel Transfer

$I_{TQPWT}$ The Fast Channel Transfer interrupt indicates when there is a new Transfer in a FC.

$R_{HZDJR}$ There is a Fast Channel Transfer interrupt per each FC implemented in the MHU.

$R_{HLKGD}$ If Fast Channel Group Transfer interrupts are implemented, it is IMPLEMENTATION DEFINED whether the Fast Channel Transfer interrupt is implemented as an internal or external interrupt.

$R_{QFVST}$ If Fast Channel Group Transfer interrupts are not implemented, the Fast Channel Transfer interrupt must be implemented as an external interrupt.

$R_{KKRKV}$    If the Fast Channel Transfer interrupt is implemented as an external interrupt the interrupt output is present on the MHUR.

$R_{JWBYZ}$    The Fast Channel Transfer interrupt is a shared enable interrupt and uses the Channel Transfer event of the FC as the event input.

$R_{VZYFJ}$    The MBX_FCH_CTRL.INT_EN is the interrupt enable field.

$R_{XNVBJ}$    The interrupt is cleared when the Receiver acknowledges the Transfer by reading the MFCW<n>_PAY register.

## B11.2.4  Fast Channel Group Transfer

$I_{NSRJY}$    Fast Channel Group Transfer interrupt indicates when one or more of the FCHs in a FCG, have asserted the Fast Channel Transfer interrupt.

$R_{QKDVG}$    It is IMPLEMENTATION DEFINED whether Fast Channel Group Transfer interrupts are implemented and whether it is implemented as an internal or external interrupt.

$R_{CJWKW}$    If the Fast Channel Group Transfer interrupts are implemented as an external interrupt the interrupt output is present on the MHUR.

$S_{SSGYY}$    Software can discover whether the Fast Channel Group Transfer interrupts are implemented using the MBX_FCH_CFG0.FCGI_SPT field.

$R_{JFNNV}$    When Fast Channel Group Transfer interrupts are implemented the following all applies:

 • MBX_FCH_CFG0.FCGI_SPT is set to `0b1`.
 • MBX_FCH_GRP<n>_INT_ST registers are implemented.

$R_{MSFBJ}$    When Fast Channel Group Transfer is implemented there is one Fast Channel Group Transfer interrupt per each FCG implemented in the MHU.

$R_{QCTMJ}$    The Fast Channel Group Transfer interrupt is a combined status interrupt and combines the Fast Channel Transfer interrupts of each Fast Channel in the FCG.

$R_{DGLDS}$    MBX_FCH_GRP<n>_INT_ST.FCH_INT_ST<m> is the status field for each Fast Channel Transfer interrupt combined by the Fast Channel Group Transfer interrupt, where:

 • <n> is the FCG number.
 • <m> is the FCH number within the FCG.

 Both <n> and <m> are numbered starting from 0 in ascending order.

## B11.2.5  Sender FIFO Low Tidemark

$I_{XFLTT}$    The Sender FIFO Low Tidemark interrupt indicates when the number of bytes used in the FIFO becomes less than or equal the low tidemark value set by the Sender.

$R_{XVNSY}$    There is a Sender FIFO Low Tidemark interrupt per each FFCH implemented in the MHU.

$R_{YFRNZ}$    It is IMPLEMENTATION DEFINED whether the Sender FIFO Low Tidemark interrupt is implemented as an internal or external interrupt.

$R_{QXJRQ}$    If the Sender FIFO Low Tidemark interrupt is implemented as an external interrupt the interrupt output is present on the MHUS.

$R_{MZJRM}$    The Sender FIFO Low Tidemark interrupt is an enable interrupt and uses the Sender FIFO Low Tidemark event of the FFCH as the event input.

$R_{NMPJL}$   The Sender FIFO Low Tidemark interrupt for the FFCH uses the following fields:

- PFFCW<n>_INT_ST.FLT - interrupt status.
- PFFCW<n>_INT_EN.FLT - interrupt enable.
- PFFCW<n>_INT_CLR.FLT - interrupt clear.

## B11.2.6  Sender FIFO High Tidemark

$I_{KPKXQ}$   The Sender FIFO High Tidemark interrupt indicates when the number of valid bytes in the FIFO becomes greater than the high tidemark value set by the Sender.

$R_{HZTJG}$   There is a Sender FIFO High Tidemark interrupt per each FFCH implemented in the MHU.

$R_{VSSGL}$   It is IMPLEMENTATION DEFINED whether the Sender FIFO High Tidemark interrupt is implemented as an internal or external interrupt.

$R_{PJPLZ}$   If the Sender FIFO High Tidemark interrupt is implemented as an external interrupt the interrupt output is present on the MHUS.

$R_{LXCPZ}$   The Sender FIFO High Tidemark interrupt is an enable interrupt and uses the Sender FIFO High Tidemark event of the FFCH as the event input.

$R_{KHTRL}$   The Sender FIFO High Tidemark interrupt for the FFCH uses the following fields:

- PFFCW<n>_INT_ST.FHT - interrupt status.
- PFFCW<n>_INT_EN.FHT - interrupt enable.
- PFFCW<n>_INT_CLR.FHT - interrupt clear.

## B11.2.7  Sender FIFO Flush

$I_{BNRBC}$   The Sender FIFO flush interrupt indicates when a FIFO flush, requested by the Receiver, has completed.

$R_{JDQBC}$   There is a Sender FIFO flush interrupt per each FFCH implemented in the MHU.

$R_{GVYRG}$   It is IMPLEMENTATION DEFINED whether the Sender FIFO flush interrupt is implemented as an internal or external interrupt.

$R_{MWHXF}$   If the Sender FIFO flush interrupt is implemented as an external interrupt the interrupt output is present on the MHUS.

$R_{LRYBR}$   The Sender FIFO flush interrupt is an enable interrupt and uses the Sender FIFO flush event of the FFCH as the event input.

$R_{ZBPCD}$   The Sender FIFO flush interrupt for the FFCH uses the following fields:

- PFFCW<n>_INT_ST.FF - interrupt status.
- PFFCW<n>_INT_EN.FF - interrupt enable.
- PFFCW<n>_INT_CLR.FF - interrupt clear.

## B11.2.8  Receiver FIFO Low Tidemark

$I_{VHPGY}$   The Receiver FIFO Low Tidemark interrupt indicates when the number of bytes used in the FIFO becomes less than or equal the low tidemark value set by the Receiver.

$R_{XHLSF}$   There is a Receiver FIFO Low Tidemark interrupt per each FFCH implemented in the MHU.

$R_{LVKLR}$   It is IMPLEMENTATION DEFINED whether the Receiver FIFO Low Tidemark interrupt is implemented as an internal or external interrupt.

$R_{CVTHH}$    If the Receiver FIFO Low Tidemark interrupt is implemented as an external interrupt the interrupt output is present on the MHUR.

$R_{FLGDH}$    The Receiver FIFO Low Tidemark interrupt is an enable interrupt and uses the Receiver FIFO Low Tidemark event of the FFCH as the event input.

$R_{YLZSV}$    The Receiver FIFO Low Tidemark interrupt for the FFCH uses the following fields:

- MFFCW<n>_INT_ST.FLT - interrupt status.
- MFFCW<n>_INT_EN.FLT - interrupt enable.
- MFFCW<n>_INT_CLR.FLT - interrupt clear.

### B11.2.9  Receiver FIFO High Tidemark

$I_{FDZLT}$    The Receiver FIFO High Tidemark interrupt indicates when the number of valid bytes in the FIFO becomes greater than the high tidemark value set by the Receiver.

$R_{DVMSQ}$    There is a Receiver FIFO High Tidemark interrupt per each FFCH implemented in the MHU.

$R_{GKRGN}$    It is IMPLEMENTATION DEFINED whether the Receiver FIFO High Tidemark interrupt is implemented as an internal or external interrupt.

$R_{FHQRP}$    If the Receiver FIFO High Tidemark interrupt is implemented as an external interrupt the interrupt output is present on the MHUR.

$R_{MZNZT}$    The Receiver FIFO High Tidemark interrupt is an enable interrupt and uses the Receiver FIFO High Tidemark event of the FFCH as the event input.

$R_{JNPBK}$    The Receiver FIFO High Tidemark interrupt for the FFCH uses the following fields:

- MFFCW<n>_INT_ST.FHT - interrupt status.
- MFFCW<n>_INT_EN.FHT - interrupt enable.
- MFFCW<n>_INT_CLR.FHT - interrupt clear.

### B11.2.10  Receiver FIFO Flush

$I_{VNDJD}$    The Receiver FIFO flush interrupt indicates when a FIFO flush, requested by the Sender, has completed.

$R_{XYRSG}$    There is a Receiver FIFO flush interrupt per each FFCH implemented in the MHU.

$R_{RLZTH}$    It is IMPLEMENTATION DEFINED whether the Receiver FIFO flush interrupt is implemented as an internal or external interrupt.

$R_{LCWBG}$    If the Receiver FIFO flush interrupt is implemented as an external interrupt the interrupt output is present on the MHUR.

$R_{HHLJG}$    The Receiver FIFO flush interrupt is an enable interrupt and uses the Receiver FIFO flush event of the FFCH as the event input.

$R_{HDHBS}$    The Receiver FIFO flush interrupt for the FFCH uses the following fields:

- MFFCW<n>_INT_ST.FF - interrupt status.
- MFFCW<n>_INT_EN.FF - interrupt enable.
- MFFCW<n>_INT_CLR.FF - interrupt clear.

### B11.2.11  Sender FFCH Combined

$I_{JFTBW}$    The Sender FFCH Combined interrupt is provided to reduce the number of interrupt wires required for the Sender to receive all possible events from a FFCH.

---

$R_{NXBTC}$  There is a Sender FFCH Combined interrupt per each FFCH implemented in the MHU.

$R_{YCCTJ}$  It is IMPLEMENTATION DEFINED whether the Sender FFCH Combined interrupt is implemented as an internal or external interrupt.

$R_{DBXHB}$  If the Sender FFCH Combined interrupt is implemented as an external interrupt the interrupt output is present on the MHUS.

$R_{FWHYF}$  The Sender FFCH Combined interrupt is a combined interrupt and combines the following interrupts of the FFCH:

- Channel Transfer Acknowledge interrupt
- Sender FIFO Low Tidemark interrupt
- Sender FIFO High Tidemark interrupt
- Sender FIFO Flush interrupt

### B11.2.12  Receiver FFCH Combined

$I_{MCSGZ}$  The Receiver FFCH Combined interrupt is provided to reduce the number of interrupt wires required for the Receiver to receive all possible events from a FFCH.

$R_{XBCKQ}$  There is a Receiver FFCH Combined interrupt per each FFCH implemented in the MHU.

$R_{TDWYH}$  It is IMPLEMENTATION DEFINED whether the Receiver FFCH Combined interrupt is implemented as an internal or external interrupt.

$R_{BSGYD}$  If the Receiver FFCH Combined interrupt is implemented as an external interrupt the interrupt output is present on the MHUS.

$R_{HWCFF}$  The Receiver FFCH Combined interrupt is a combined interrupt and combines the following interrupts of the FFCH:

- Channel Transfer
- Receiver FIFO Low Tidemark interrupt
- Receiver FIFO High Tidemark interrupt
- Receiver FIFO Flush interrupt

### B11.2.13  Postbox Combined

$I_{BDFSR}$  The Postbox Combined interrupt is provided to reduce the number of interrupt wires required to receive all events from a Postbox.

$R_{KQQJR}$  There is one Postbox Combined interrupt per Postbox implemented in the MHU.

$R_{QMSWR}$  The Postbox Combined interrupt is an external interrupt and has the output on the MHUS.

$R_{TFRCY}$  The Postbox Combined interrupt is a combined enabled interrupt.

$R_{MDSFB}$  The Postbox Combined interrupt combines the following interrupts:

- Channel Transfer Acknowledge interrupt of each DBCH implemented in the MHU.
- Sender FFCH Combined interrupt for each FFCH implemented in the MHU.

$R_{BWFWH}$  The Postbox Combined interrupt has the following register fields which are the interrupt enables:

- PDBCW<n>_CTRL.PBX_COMB_EN
- PFFCW<n>_CTRL.PBX_COMB_EN

$R_{DJMZN}$  The Postbox Combined interrupt has the following register fields which are the interrupt status:

- PBX_DBCH_INT_ST<n>.DBCH_INT_ST<m>
- PBX_FFCH_INT_ST<n>.FFCH_INT_ST<m>

$R_{PVJLY}$ PBX_DBCH_INT_ST<n>.DBCH_INT_ST<m> shows whether the Channel Transfer Acknowledge interrupt for that DBCH is asserted, where:

- <n> = floor(DBCH_num/32)
- <m> = DBCH_num - 32*n

$R_{CVQBN}$ PBX_FFCH_INT_ST<n>.FFCH_INT_ST<m> shows whether the Sender FFCH Combined interrupt for that FFCH is asserted, where:

- <n> = floor(FFCH_num/32)
- <m> = FFCH_num - 32*n

## B11.2.14 Mailbox Combined

$I_{NGCJF}$ The Mailbox Combined interrupt is provided to reduce the number of interrupt wires required to receive all events from a Mailbox.

$R_{BYGHD}$ There is one Mailbox Combined interrupt per Mailbox implemented in the MHU.

$R_{FTZCX}$ The Mailbox Combined interrupt is an external interrupt and has the output on the MHUR.

$R_{PXZQK}$ The Mailbox Combined interrupt is a combined enabled interrupt.

$R_{JRWCQ}$ The Mailbox Combined interrupt combines the following interrupts:

- Channel Transfer interrupt for each DBCH implemented in the MHU.
- Receiver FFCH Combined interrupt for each FFCH implemented in the MHU.
- Fast Channel Group Transfer interrupts, if they are implemented, for each Fast Channel Group implemented in the MHU.

$R_{SPGDV}$ The Mailbox Combined interrupt has the following register fields which are the interrupt enables:

- MDBCW<n>_CTRL.MBX_COMB_EN.
- MFFCW<n>_CTRL.MBX_COMB_EN.
- MBX_FCH_INT_EN.MBX_COMB_EN<m>, if Fast Channel Group Transfer interrupts are implemented.

$R_{JKNVD}$ The Mailbox Combined interrupt has the following register fields which are the interrupt status:

- MBX_DBCH_INT_ST<n>.DBCH_INT_ST<m>.
- MBX_FFCH_INT_ST<n>.FFCH_INT_ST<m>.
- MBX_FCG_INT_ST.FCH_GRP<m>, if Fast Channel Group Transfer interrupts are implemented.

$R_{DQPLS}$ MBX_DBCH_INT_ST<n>.DBCH_INT_ST<m> shows whether the Receiver Channel Transfer interrupt for that DBCH is asserted, where:

- <n> = floor(DBCH_num/32)
- <m> = DBCH_num - 32*n

$R_{XMQDD}$ MBX_FFCH_INT_ST<n>.FFCH_INT_ST<m> shows whether the Receiver FFCH Combined interrupt for that FFCH is asserted, where:

- <n> = floor(FFCH_num/32)
- <m> = FFCH_num - 32*n

$R_{PTPFH}$ MBX_FCG_INT_ST.FCH_GRP<m> shows whether the Fast Channel Group Transfer interrupt for that FCG is asserted, where <m> is the FCG number.

# Part C
## Programmers Model

# Chapter C1
# Programmers Model Overview

## C1.1  Register Access

$I_{\text{RBDNK}}$     This section covers the requirements for accesses to registers of the MHU.

### C1.1.1  Generic Rules

$R_{\text{VMLNR}}$     An implementation of the MHU support aligned accesses to all registers.

$S_{\text{HNDQC}}$     To ensure the order of and visibility accesses to the MHU, software must only use one of the following memory types to access the registers of the MHU:

- Device non-gathering, non-reordering, non-early write-response (DEV-nGnRnE).
- Device non-gathering, non-reordering, early write-response (DEV-nGnRnE).

For a definition of memory access types refer to [2].

$R_{\text{KCWRN}}$     An implementation of the MHU is required to support 32-bit access to all registers except for the following:

- PFFCW<n>_PAY
- PFCW<n>_PAY
- MFFCW<n>_PAY
- MFFCW<n>_FLG
- MFCW<n>_PAY

$R_{\text{FBFXY}}$     It is IMPLEMENTATION DEFINED whether accesses of other sizes are supported by an implementation to any registers except the following:

- PFFCW<n>_PAY
- PFCW<n>_PAY
- MFFCW<n>_PAY
- MFFCW<n>_FLG
- MFCW<n>_PAY

$R_{\text{SSGQS}}$     An access is considered an unsupported access if any of the following apply:

- The access is of a size not supported by the MHU implementation, for the register being accessed.
- The access is not aligned to the size of the access and the MHU implementation does not support unaligned accesses.

For more information on unsupported accesses refer to  C1.1.4 *Unsupported Accesses*.

$R_{\text{TBWRZ}}$     It is not required for the MHUS and MHUR to have support for the same access sizes and alignment.

## C1.1.2   FCH Payload Accesses

$I_{\text{SMVKS}}$     This section covers the rules specific for accesses to the PFCW<n>_PAY and MFCW<n>_PAY registers.

$R_{\text{ZTSGC}}$     When FCE is implemented with a Fast Channel word-size of 32-bits, the MHU must support 32-bit accesses to the PFCW<n>_PAY and MFCW<n>_PAY registers.

$R_{\text{HXVTN}}$     When FCE is implemented with a Fast Channel word-size of 64-bits, the MHU must support 64-bit accesses to the PFCW<n>_PAY and MFCW<n>_PAY registers.

$R_{\text{VHMXF}}$     It is IMPLEMENTATION DEFINED whether other size accesses are supported to the PFCW<n>_PAY and MFCW<n>_PAY register.

$R_{\text{MKVMT}}$     An access of an unsupported size to the PFCW<n>_PAY and MFCW<n>_PAY registers is considered an illegal access. For more information on illegal accesses refer to  C1.1.5 *Illegal Accesses*.

$S_{\text{PSJCJ}}$     Arm recommends that software access the PFCW<n>_PAY and MFCW<n>_PAY registers at the Fast Channel word-size so as to atomically update all bytes of the FCH.

## C1.1.3   FFCH Payload and Flag Accesses

$I_{\text{ZBZLM}}$     This section covers the rules specific for accesses to the PFFCW<n>_PAY, MFFCW<n>_PAY and MFFCW<n>_FLG registers.

$R_{\text{CDCVS}}$     When FE is implemented the MHU can support the following access sizes to the PFFCW<n>_PAY, MFFCW<n>_PAY and MFFCW<n>_FLG registers:

- 8-bit
- 16-bit
- 32-bit
- 64-bit

$R_{\text{PMJHB}}$     An implementation of the MHU which implements FE must support either 32-bit or 64-bit accesses to the PFFCW<n>_PAY, MFFCW<n>_PAY and MFFCW<n>_FLG registers. Support for other sizes is IMPLEMENTATION DEFINED.

$R_{\text{XVQKL}}$     It is allowed for an implementation to support different accesses sizes between the PFFCW<n>_PAY and MFFCW<n>_PAY registers and the PFFCW<n>_PAY and the MFFCW<n>_FLG registers.

$R_{\text{YCHPT}}$     The supported accesses sizes to the MFFCW<n>_PAY and MFFCW<n>_FLG registers must be the same.

$R_{\text{PYSBT}}$     It is IMPLEMENTATION DEFINED whether an access of an unsupported size to the PFFCW<n>_PAY register is:

- Considered an unsupported access.
- Truncated to a smaller supported size.

- Zero extended to a larger supported size.

When the access is truncated or zero extended to a supported size the MHU uses the new size as the requested size when calculating if it has space to push data onto the FIFO. This includes any update to the PFFCW<n>_ST.PPE field.

$R_{GPXQC}$     It is IMPLEMENTATION DEFINED whether an access of an unsupported size to the MFFCW<n>_PAY or MFFCW<n>_FLG register is:

- Considered an unsupported access.
- Truncated to a smaller supported size.
- Zero extended to a large supported size.

When the access is truncated or zero extended to a supported size, the MHU uses the new size when for one or more of the following operations:

- Calculating the number of bytes to read from the FIFO.
- Calculating the number of bytes to pop from the FIFO.

For more information on illegal accesses refer to *C1.1.5 Illegal Accesses*.

$S_{TJXXR}$     Arm recommends that both the Sender and Receiver only uses accesses with a supported size to access the PFFCW<n>_PAY, MFFCW<n>_PAY and MFFCW<n>_FLG registers, otherwise it can lead to corruption of or lost of Transfer data.

$R_{MCSPF}$     When 8-bit accesses are supported to the PFFCW<n>_PAY register the PBX_FFCH_CFG0.P8BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{ZLCQF}$     When 16-bit accesses are supported to the PFFCW<n>_PAY register the PBX_FFCH_CFG0.P16BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{MQYCM}$     When 32-bit accesses are supported to the PFFCW<n>_PAY register the PBX_FFCH_CFG0.P32BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{WTXTG}$     When 64-bit accesses are supported to the PFFCW<n>_PAY register the PBX_FFCH_CFG0.P64BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{JJCJC}$     When 8-bit accesses are supported to the MFFCW<n>_PAY and MFFCW<n>_FLG registers the MBX_FFCH_CFG0.M8BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{DZGTF}$     When 16-bit accesses are supported to the MFFCW<n>_PAY and MFFCW<n>_FLG registers the MBX_FFCH_CFG0.M16BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{HCKZC}$     When 32-bit accesses are supported to the MFFCW<n>_PAY and MFFCW<n>_FLG registers the MBX_FFCH_CFG0.M32BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$R_{YLNVH}$     When 64-bit accesses are supported to the MFFCW<n>_PAY and MFFCW<n>_FLG registers the MBX_FFCH_CFG0.M64BA_SPT field is set to `0b1` otherwise it is set to `0b0`.

$S_{SWNSD}$     Sender or Receiver can use the values of the PBX_FFCH_CFG0.P{8/16/32/64}BA_SPT and MBX_FFCH_CFG0.M{8/16/32/64}BA_SPT fields to determine the supported accesses sizes to the PFFCW<n>_PAY, MFFCW<n>_PAY and MFFCW<n>_FLG registers.

$I_{ZJGMB}$     When TZE or RME are implemented, there are equivalent fields in the SSC_FFCH_CFG0 and RSC_FFCH_CFG0 registers. These values are intended to be used by the software agent which owns the SSC and RSC block.

## C1.1.4 Unsupported Accesses

$I_{PPCMR}$     Access to the MHUS or MHUR can be considered to be an unsupported access, due to having one or more of the following:

- A size which is not supported by the register being accessed.

- An alignment which is not aligned to the size of the access, if the implementation of the MHU does not support unaligned accesses.

$R_{RKZZH}$     It is CONSTRAINED UNPREDICTABLE whether an implementation of the MHU treats an unsupported access as one of the following:

- RAZ/WI, and has no effect on any internal state of the MHU.
- Modifies the access to have an alignment and size, which is supported by the implementation of the MHU.

## C1.1.5 Illegal Accesses

$I_{FYSWJ}$     Accesses to the MHUS or MHU can be considered to be an illegal access, due to the access having the incorrect security to access the register. Refer to Chapter B9 *TrustZone and Realm Management Extension* for more information on the security requirements for accesses.

$R_{RYGLR}$     An illegal access is treated as RAZ/WI and has no effect on any internal state of the MHU. For example, a write access to the PFFCW<n>_PAY register which is consider illegal does not:

- Push any data onto the FIFO.
- Change the value of the PFFCW<n>_ST.PPE field.
- Generate any events.

## C1.2 Register Access Response

$R_{RFJVN}$     All read accesses must provide their response from the location storing the data.

$R_{YHYZB}$     A read (R1) to the MFFCW<n>_PAY register when the MFFCW<n>_CTRL.RA_EN field is set to `0b1,` cause data to be popped from the FIFO and one or more of the following to occur:

- The PFFCW<n>_ST.FFS, PFFCW<n>_PAY.FFS, MFFCW<n>_ST.FFL and MFFCW<n>_FLG.FFL fields are updated.
- The PFFCW<n>_ACK_CNT.ACK_CNT field is incremented by the number of bytes popped from the FIFO which had the both the EOT and ACK flags set.
- The Transfer Acknowledge event is generated, if the conditions are correct.
- A Sender Low Tidemark event is generated, if the conditions are correct.
- A Receiver Low Tidemark event is generated, if the conditions are correct.

It is not required for the updates to the PFFCW<n>_ST.FFS, PFFCW<n>_PAY.FLS, MFFCW<n>_ST.FFL, MFFCW<n>_FLG.FFL and PFFCW<n>_ACK_CNT.ACK_CNT fields to be made before the data is returned in response to the read(R1). However the following is required:

- The Transfer Acknowledge and Sender Low Tidemark events are generated only if a read (R2) of the PFFCW<n>_ACK_CNT, PFFCW<n>_ST.FFS or PFFCW<n>_PAY.FFS fields would observe the updated value due to the first read (R1).
- The Receiver Low Tidemark event is generated only if a read (R2) of MFFCW<n>_ST.FFL or MFFCW<n>_FLG.FFL fields would observe the updated value due to the first read (R1).

$R_{MMXTN}$     A read (R1) to the MFCW<n>_PAY register causes the Channel Transfer interrupt for the FCH to be updated.

$R_{JKGJW}$     A read (R1) to the PFFCW<n>_ACK_CNT register caused the value of the register to be updated as defined in $R_{MDTHX}$.

$R_{TZYBZ}$     The MHUS or MHUR must only provide a write response to a write after all effect, including side effects, except for writes to the following registers:

- PDBCW<n>_SET
- PFFCW<n>_PAY
- PFCW<n>_PAY
- MDBCW<n>_CLR
- MFFCW<n>_FIFO_POP
- MFCW<n>_PAY

$R_{JFXRY}$     It is IMPLEMENTATION DEFINED whether write accesses to the following registers provide an early write response:

- PDBCW<n>_SET
- PFFCW<n>_PAY
- PFCW<n>_PAY
- MDBCW<n>_CLR
- MFFCW<n>_FIFO_POP
- MFCW<n>_PAY

$I_{ZYHSQ}$     An early write response is defined as when a response to a write is generated before all effects of the write, for example setting values in another register, have occurred.

$R_{THJMQ}$     When an implementation of the MHU does not implement early write response, for a write W1, to the PDBCW<n>_SET register the following must all be completed before the write response is generated:

- PDBCW<n>_ST register is updated to reflect its new value.
- MDBCW<n>_ST and MDBCW<n>_ST_MSK register is updated to reflect its new value.
- Channel Transfer event is generated only when a read by the Receiver would observe the updated value of the MDBCW<n>_ST and MDBCW<n>_ST_MSK.

$R_{PYKHD}$      When an implementation of the MHU implements early write response, for a write W1, to the PDBCW<n>_SET register the following must be true:

- Any subsequent read by the Sender of the PDBCW<n>_ST register must observe the updates caused by W1.
- Any subsequent write by the Sender to the PDBCW<n>_SET register must occur after W1.
- A Channel Transfer event is generated only when a read by the Receiver of the MDBCW<n>_ST or MDBCW<n>_ST_MSK register would observe the updates caused by W1.

$R_{PWFZF}$      When an implementation of the MHU does not implement early write response, for a write W1, to the PFFCW<n>_PAY register the following must all be completed before the write response is generated:

- The values of the PFFCW<n>_PAY or PFFCW<n>_ST register is updated to reflect the changes to the PPE and FFS fields caused by W1.
- The values of the PFFCW<n>_FLG register is updated, if the PFFCW<n>_CTRL.TDM field is set to `0b01` or `0b10`.
- FIFO High Tide event is generated, if required.
- Channel Transfer event is generated only when a read by the Receiver of the MFFCW<n>_PAY register, will be able to observe the value of the bytes pushed onto the FIFO by W1.

$R_{XLYSQ}$      When an implementation of the MHU implements early write response, for a write W1, to the PFFCW<n>_PAY register the following all must be true:

- Any subsequent read by the Sender of the PFFCW<n>_PAY or PFFCW<n>_ST register must observe the changes to the PPE and FFS fields caused by W1.
- Any subsequent write by the Sender of the PFFCW<n>_PAY register must occur after W1.
- When the write data is pushed onto the FIFO the value of the SOT, EOT and ACK flags is the value of the fields in the PFFCW<n>_FLG register when the write response was generated.
- If PFFCW<n>_CTRL.TDM is set to `0b01` or `0b10` the value of the SOT and EOT fields in the PFFCW<n>_FLG register are updated when the write response is generated.
- A Channel Transfer event is only generated when a read by the Receiver of the MFFCW<n>_PAY register, will be able to observe the value of the bytes pushed onto the FIFO by W1.
- Any FIFO High Tide event is generated to the Sender when the write response is given.

$R_{TVNND}$      When an implementation of the MHU does not implement early write response, for a write W1, to the PFCW<n>_PAY register the following must all be completed before the write response is generated:

- PFCW<n>_PAY register is updated to reflect the value set by W1.
- MFCW<n>_PAY register is updated to reflect the value set by W1.
- A Channel Transfer event is generated only when a read by the Receiver of the MFCW<n>_PAY register would observe the updates caused by W1.

$R_{WZQFJ}$      When an implementation of the MHU implements early write response, for a write W1, to the PFCW<n>_PAY register the following must be true:

- Any subsequent read by the Sender of the PFCW<n>_PAY register must observe the updates caused by W1.
- Any subsequent write by the Sender of the PFCW<n>_PAY register must occur after W1.
- A Channel Transfer event is only generated when a read by the Receiver of the MFCW<n>_PAY register would observe the updates caused by W1.

$R_{VGNNJ}$      When an implementation of the MHU does not implement early write response, for a write W1, to the MDBCW<n>_CLR register the following must all be completed before the write response is generated:

- MDBCW<n>_ST and MDBCW<n>_ST_MSK registers are updated to reflect the updated value caused by W1.
- PDBCW<n>_ST register are updated to reflect the updated value caused by W1.
- Channel Transfer Acknowledge event is only generated when a read by the Sender of the PDBCW<n>_ST register would observe the updates caused by W1.

$R_{XGGMJ}$      When an implementation of the MHU implements early write response, for a write W1, to the MDBCW<n>_CLR register the following must be true:

- Any subsequent read by the Receiver of the MDBCW<n>_ST or MDBCW<n>_ST_MSK registers must observe the updates caused by W1.
- Any subsequent write by the Receiver to the MDBCW<n>_CLR register must occur after W1.
- Any Channel Transfer Acknowledge event is only generated when a read by the Sender of the PDBCW<n>_ST register would observe the updates caused by W1.

$I_{ZRXJG}$  A write to the MDBCW<n>_CLR register can causes the de-assertion of the Channel Transfer interrupt for a DBCH, if the write results in all fields in the MDBCW<n>_ST_MSK registers being `0b0`. Arm recommends that the Channel Transfer interrupt is de-asserted either at or before the point where a read of MDBCW<n>_ST_MSK register would return all fields as `0b0`.

$R_{CKBFN}$  When an implementation of the MHU does not implement early write response, for a write W1, to the MFFCW<n>_FIFO_POP register, and the MFFCW<n>_CTRL.RA_EN is `0b0`, the following must all be completed before the write response is generated:

- The number of bytes requested by W1 are popped from the FIFO.
- MFFCW<n>_ST.FFL and MFFCW<n>_FLG.FFL fields are updated to reflect the number of bytes popped from the FIFO caused by W1.
- PFFCW<n>_ACK_CNT.ACK_CNT, PFFCW<n>_ACK_CNT.ACK_CNT_OVRFLW are updated, as required by the popping of bytes requested by W1.
- PFFCW<n>_ST.FFS and PFFCW<n>_PAY.FFS fields are updated to reflect the number of bytes popped from the FIFO caused by W1.
- A Channel Transfer Acknowledge or Sender FIFO Low Tide events are generated only when a read of the PFFCW<n>_ACK_CNT.ACK_CNT, PFFCW<n>_ACK_CNT.ACK_CNT_OVRFLW, PFFCW<n>_ST.FFS or PFFCW<n>_PAY.FFS fields would observe the updated values caused by W1.

$R_{HPVGP}$  When an implementation of the MHU implements early write response, for a write W1, to the MFFCW<n>_FIFO_POP register, when MFFCW<n>_CTRL.RA_EN is `0b0`, the following must be true:

- Any subsequent read (R1) by the Receiver of the MFFCW<n>_PAY register will return the number of bytes equal to the size of the read, starting from the first byte after the bytes to be popped from the FIFO by write W1.

  For example, if W1 has a value of X and R1 has a size Y the bytes returned by the R1 is:

  - FIFO byte offset start = X.
  - FIFO byte offset end = X + Y - 1.

  Bytes in the FIFO are numbered from 0, with 0 being the head of the FIFO

- Any subsequent read (R1) by the Receiver of the MFFCW<n>_ST.FFL or MFFCW<n>_FLG.FFL fields returns a fill level observing the bytes popped by W1.
- Any Channel Transfer Acknowledge or Sender FIFO Low Tide events are only generated when a read by the Sender of the PFFCW<n>_ACK_CNT.ACK_CNT, PFFCW<n>_ACK_CNT.ACK_CNT_OVRFLW, PFFCW<n>_ST.FFS or PFFCW<n>_PAY.FFS fields would observe the bytes popped by W1.

$I_{LPDNM}$  When MFFCW<n>_CTRL.RA_EN is `0b1`, writes to the MFFCW<n>_FIFO_POP register have no effect and can complete instantly. This is not considered an early write response.

$R_{ZNRTW}$  When an implementation of the MHU does not implement early write response, for a write W1, to the MFCW<n>_PAY register the following must all be completed before providing the write response:

- MFCW<n>_PAY register must be updated to reflect the changes caused by W1.
- PFCW<n>_PAY register must be updated to reflect the changes caused by W1.

$R_{JGBZR}$  When an implementation of the MHU implement early write response, for a write W1, to the MFCW<n>_PAY register the following must be true:

- Any subsequent read by the Receiver of the MFCW<n>_PAY register must observe the updates caused by W1.
- Any subsequent write by the Receiver of the MFCW<n>_PAY register must occur after W1.

## C1.3 Reset Domains

$I_{BBPSM}$  This section contains information about the reset for fields, for fields which have an architected reset value.

$R_{WLPFV}$  The MHU has two reset domains:

- MHUS
- MHUR

$R_{CYRLT}$  Fields which have a reset domain of MHUS are set to its architected reset value, if it has one, when the MHUS enters a Non-operational state.

$R_{RZHKY}$  Fields which have a reset domain of MHUR are set to its architected reset value, if it has one, when the MHUR enters a Non-operational state.

$R_{BBJJG}$  Fields which have no reset domain either:

- Do not have an architected reset value, for example the PFCW<n>_PAY and MFCW<n>_PAY registers.
- Have a constant value, for example the PBX_FEAT_SPT0 or MBX_FEAT_SPT0 registers.
- Have a value which is calculated based on other fields which have a reset value, for example reads of the PFFCW<n>_PAY register return values based on the status of the FIFO.

# C1.4 Storage Resources

$I_{KPGMS}$ The MHU architecture allows for flexibility in the type and location of storage elements used in an implementation of the MHU. The MHU architecture also uses aliasing where two or more fields use the same storage resource to store the value.

## C1.4.1 Storage resource location

$I_{BYDLG}$ The location of a storage resource only applies for when the MHU implementation is distributed, as for a monolithic implementation the MHUS and MHUR are considered a single entity.

$R_{TPXFT}$ A storage resource refers to any logical structure which meets the following requirements:

- Has a method for reading the value of the logical structure.
- Has a method for writing the value of the logical structure based on a write enable.
- Has a method for being set to a known value when the storage element exits a Non-operational state, if the field it is storing the value for has a defined reset value.
- Retains the last value written to it, when it is not being written to and it has not entered a Non-operational state.

$I_{TSWFV}$ Examples of logical structures which meet these requirements are:

- Flip-flops
- RAM

$R_{LBHGN}$ Storage location for a field which is part of the MHUS reset domain must be in the MHUS.

$R_{CVPZP}$ Storage location for a field which is part of the MHUR reset domain must be in the MHUR.

$I_{DKMQB}$ A field is considered part of a reset domain even if the reset value is UNKNOWN.

## C1.4.2 Field Aliasing

$I_{KMHJD}$ Some fields in the MHU architecture are mapped onto the same storage location.

$R_{LGLBK}$ The field which share resources are:

- PDBCW<n>_ST.FLG<x> and MDBCW<n>_ST.FLG<x>.
- PFCW<n>_PAY.PAY and MFCW<n>_PAY.PAY.
- PFFCW<n>_PAY.PPE and PFFCW<n>_ST.PPE.

## C1.4.3 FIFO storage resources

$I_{ZSNCM}$ When FE is implemented there is implicit storage elements required to form the FIFO and FHB.

$R_{KLDLF}$ The location of the storage elements which make up the FIFO is IMPLEMENTATION DEFINED but must meet the requirements defined in Chapter B8 *FIFO Extension*

$R_{TDFGW}$ The FHB must be implemented in the MHUR as defined in Chapter B8 *FIFO Extension*.

# C1.5 Register Blocks

$I_{PTVSV}$    The registers of the MHU and grouped in pages and blocks.

$R_{WGHKQ}$    A page is 4KB in size and groups register which are related together.

$R_{FFNTS}$    Any location in a page which is not defined to include registers is Reserved and treated as RAZ/WI.

$R_{HFWKN}$    A block is 64KB in size and is formed of a number of pages.

$R_{XDLJJ}$    Any location within a block which is not allocated a page or the page is not implemented due to the configuration of the MHU, is Reserved and treated as RAZ/WI, with the exception of the IMPLEMENTATION DEFINED page within a block.

$I_{MQWYQ}$    For more information on the IMPLEMENTATION DEFINED page refer to C1.7 *IMPLEMENTATION DEFINED registers*.

$I_{KPDSX}$    Even though it may be possible for individual pages or smaller to be allocated to different software agents it is not an expected use cases. The MHU has been designed so that an agent has access to all pages within a block and isolation between agents is performed at the block level.

$R_{ZMTFX}$    The offset of registers within pages, and pages within blocks is fixed by the architecture.

$R_{JWYVT}$    The offset of blocks within an implementation of the MHU is IMPLEMENTATION DEFINED.

$I_{DKZDC}$    Arm recommends that the order of blocks within the MHUS or MHUR for an MHUv3.0 compliant MHU is as follows:

1. Sender/Receiver Security Control block, if TZE or RME are implemented.
2. Postbox/Mailbox block.
3. Sender/Receiver RAS block, if RASE is implemented.

Table C1.1 shows the layout of blocks within the MHUS or MHUR following these recommendations:

**Table C1.1: Example order of blocks for an MHUv3.0 implementation, with support for TZE and RME and RASE**

| Block Num | Block |
|-----------|-------|
| 0 | Sender/Receiver Security Control |
| 1 | Postbox/Mailbox |
| 2 | Sender/Receiver RAS |

# C1.6  Software Discovery

$I_{\text{MGFLR}}$   Each MHU implementation can implement a different selection and configuration of extensions of the MHU architecture. Software will need to identify the extensions and configurations of those extensions which the MHU implementation implements. To support this the MHU includes registers to enable software to discover the configuration of the MHU. These registers are referred to as the Configuration and Feature registers (CFR).

$S_{\text{JBNMC}}$   Software can use the CFRs or it can use an IMPLEMENTATION DEFINED method to discover the configuration of the MHU implementation.

$I_{\text{CLCVW}}$   An example IMPLEMENTATION DEFINED method would be software data structures or APIs, for example Device Tree or ACPI.

$R_{\text{HPBZW}}$   There is a set of CFRs for each block in the MHU located in the first page. These identification register provide the following information:

- Type of block being accessed.
- Features supported by the MHU.
- Configuration of the block.

$R_{\text{TSCKF}}$   The CFR are located within the first 64 32-bits words within the block.

$R_{\text{RRQYJ}}$   At offset `0x0000` of a block, with the exception of the Sender and Receiver RAS blocks, is the block identifier <x>_BLK_ID, where <x> is one of the following block prefixes: PBX, MBX, SSC, RSC.

$R_{\text{YTLNX}}$   The block identifier has a single 4-bit field called, BLK_ID which has one of the following values:

- `0x0` - Postbox.
- `0x1` - Mailbox.
- `0x2` - Sender Security Control.
- `0x3` - Receiver Security Control.

$S_{\text{JVXKP}}$   Software can use the block identifier to know the type of the block, except for the Sender and Receiver RAS block, it is accessing.

$R_{\text{LYSFN}}$   The Sender and Receiver RAS blocks do not include a <x>_BLK_ID register.

$S_{\text{RVDLZ}}$   Software must use an IMPLEMENTATION DEFINED method to discover whether a block is the Sender or Receiver RAS block.

$X_{\text{BJPNK}}$   The reason for the Sender and Receiver RAS blocks not including a <x>_BLK_ID register is that the expected user of the RAS blocks is different to the other blocks. RAS registers have their own standard, which have their own method of identification.

$R_{\text{FJMQR}}$   The CFRs include registers which provide information on the extensions which are implemented by the implementation of the MHU. These are referred to as Feature registers and have the following format <x>_FEAT_SPT<n>, where:

- <x> is the block prefixes: PBX, MBX, SSC, RSC.
- <n> is the register number and is in the range 0-1.

An example of a Feature registers is PBX_FEAT_SPT0 and PBX_FEAT_SPT1 which provides information on the features implemented by the Postbox.

$R_{\text{XLGHZ}}$   The CFRs includes registers which provide information on the configuration of the extensions, block or MHU. These are referred to as Configuration registers and have following format <x>_<y>_CFG<z>, where:

- <x> is the block prefixes: PBX, MBX, SSC, RSC.
- <y> is the identifier for the name of the feature the register provides configuration information on, for example DBCH or FFCH for Doorbell Extension and FIFO Channel Extension configuration information respectively.
- <z> is the register number starting from 0.

An example of a Configuration registers is MBX_DBCH_CFG0 and MBX_FFCH_CFG0 which provide configuration information about the DBCHs and FFCHs implemented in the Mailbox.

$R_{NMDWK}$        The CFRs are read-only, when accessed from the block they apply to.

$R_{DXXSQ}$        The values of the CFRs are constant.

# C1.7 IMPLEMENTATION DEFINED registers

$R_{XCBXX}$ The MHU provides the following regions for IMPLEMENTATION DEFINED registers:

- Offset `0x0FC8` of every block for the Implementation Identification Register.
- Offsets `0x0FD0-0x0FFF` of every block for IMPLEMENTATION DEFINED Identification Registers.
- Offsets `0xF000-0xFFFF` of every block for an IMPLEMENTATION DEFINED page.

## C1.7.1 Implementation Identification Registers

$I_{GWYRC}$ The MHU provides the Implementation Identification Register (IIDR) which provides the product ID, revision and variant and implementor identifier.

$I_{DZYND}$ The name of the IIDR register is prefixed by the block ID. For example, PBX_IIDR is the IIDR register in the Postbox.

$R_{JVTWG}$ The IIDR is a read-only register.

$R_{KLMQZ}$ An implementation must always set the fields of the IIDR registers to values which will identify the implementor, part and version of the implementation of the MHU.

$I_{NDSPJ}$ The variant and revision fields of the IIDR are considered to be major and minor revisions of the implementation of the MHU respectively.

## C1.7.2 IMPLEMENTATION DEFINED Identification Registers

$R_{LLBMK}$ The MHU defines a region of 12 registers for IMPLEMENTATION DEFINED identification registers located at `0xFD0-0xFFF` in the first 4KB of each block.

$R_{ZPLLS}$ The IMPLEMENTATION DEFINED identification registers are always read-only.

$R_{ZWGRB}$ The presence, names of the registers and fields and values of the IMPLEMENTATION DEFINED identification registers is IMPLEMENTATION DEFINED. If an offset within the IMPLEMENTATION DEFINED identification registers region is not implemented the offsets is Reserved and treated as RES0.

$I_{TRNJQ}$ Arm recommends that the format of the Identification Registers follows the format defined in [3] for Component and Peripheral Identification Registers, with the Class field set to `0xF`.

## C1.7.3 IMPLEMENTATION DEFINED Page

$I_{BMKWS}$ The IMPLEMENTATION DEFINED page provides a locations for the implementor of the MHU to include additional registers to extend the functionality of the MHU to meet the use case of the MHU. An example of where additional registers may be required is to add registers related to functional safety.

$I_{PVZWR}$ The names of the IMPLEMENTATION DEFINED pages are as follows:

- Postbox IMPLEMENTATION DEFINED page, PBX_IMPL_DEF_page, in the Postbox block.
- Mailbox IMPLEMENTATION DEFINED page, MBX_IMPL_DEF_page, in the Mailbox block.
- Sender Security Control IMPLEMENTATION DEFINED page, SSC_IMPL_DEF_page, in the Sender Security Control block.
- Receiver Security Control IMPLEMENTATION DEFINED page, RSC_IMPL_DEF_page, in the Receiver Security Control block.

$R_{TKRFW}$ Each block of the MHU has a single IMPLEMENTATION DEFINED page located at offset `0xF000` in the block

R<sub>RXYYB</sub>    The registers of the IMPLEMENTATION DEFINED page are IMPLEMENTATION DEFINED. This includes:

- Whether a register is present or not.
- The name of the register.
- The access permission of the registers, so long as they are not more permissive than the architecture defines for the IMPLEMENTATION DEFINED register.
- The names, behavior and reset value of any fields.
- The access permissions of any fields, so long as they are not more permissive than the register.

R<sub>SDQNR</sub>    When TZE or RME are implemented access to the IMPLEMENTATION DEFINED page are only allowed if the security of the access matches the requirement of the block as follows:

- For a PBX_IMPL_DEF_page or MBX_IMPL_DEF_page the security of the access must be of a security allowed to access the Postbox or Mailbox.
- For a SSC_IMPL_DEF_page or RSC_IMPL_DEF_page the security of the access must be:
  - Secure when TZE is implemented and RME is not implemented or RME is implemented and the sampled value of the **LEGACY_TZ_EN** input is 0b1.
  - Root when RME is implemented and the sampled value of the **LEGACY_TZ_EN** input is 0b0.

R<sub>WFCBR</sub>    It is allowed for registers in the IMPLEMENTATION DEFINED page to be restricted to accesses from a specific security world, for example only accessible by secure accesses, however the access must first past the register block security check.

R<sub>FHFPX</sub>    It is not required that IMPLEMENTATION DEFINED pages of different blocks contain the same registers.

R<sub>BFHGB</sub>    The architected functionality of the MHU must not be altered by the reset value of any register in IMPLEMENTATION DEFINED page.

X<sub>LBNRD</sub>    This allows generic MHU drivers to make use of the MHU implementation without having knowledge of the IMPLEMENTATION DEFINED features.

R<sub>SWBPM</sub>    The IMPLEMENTATION DEFINED page follows the same rules as any other page in the MHU and any unused locations are considered to be Reserved and treated as RES0.

# Chapter C2
# **Registers**

## C2.1  MHUS, MHU Sender

The MHUS characteristics are:

**Purpose**

Contains the blocks which are part of the MHU Sender

The offsets of the blocks, within the MHU Sender are IMPLEMENTATION DEFINED, however, Arm recommends the following offsets:

**When TZE is implemented for the MHUS**

0x0_0000 - SSC

0x1_0000 - PBX

**When TZE is not implemented for the MHUS**

0x0_0000 - PBX

**Attributes**

**When TZE is implemented for the MHUS**

The MHUS block is of size 128KB.

**When TZE is not implemented for the MHUS**

The MHUS block is of size 64KB.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| IMPDEF | SSC | Most permissive access: RW<br>Register condition: When TZE is implemented for the MHUS<br>Accessor condition: When TZE is implemented for the MHUS |
| IMPDEF | PBX | Most permissive access: RW |

## C2.1.1 PBX, Postbox

The PBX characteristics are:

**Purpose**

Contains the individual pages of the Postbox block.

Only accesses with the correct security can access the registers within the Postbox, otherwise the access is treated as an illegal access

The allowed security of an access to a register of the Postbox depends on:

- Whether RME is implemented for the MHUS.

- Whether TZE is implemented for the MHUS.

- Sampled value of MHUS **LEGACY_TZ_EN** input.

- Value of SSC_CTRL_page.SSC_PBX_SG0.SG_PBX0 field.

The following table list the allowed security states of an access:

| RME | TZE | Sampled **LEGACY_TZ_EN** | SSC_PBX_SG0.SG_PBX | Allowed access securities |
|-----|-----|--------------------------|--------------------|---------------------------|
| 0 | 0 | NA | NA | Any |
| 0 | 1 | NA | 0b0 | Secure |
| | | | 0b1 | Any |
| 1 | 1 | 0 | 0b00 | Root and Secure |
| | | | 0b01 | Any |
| | | | 0b10 | Root |
| | | | 0b11 | Root and Realm |
| | | 1 | 0bx0 | Root and Secure |
| | | | 0bx1 | Any |

**Configuration**

**Attributes**

The PBX block is of size 64KB.

This block is part of the MHUS block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| `0x0000` | PBX_CTRL_page | Most permissive access: RW |
| `0x1000` | PDBCW_page | Most permissive access: RW<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |

| Offset | Name | Notes |
|---|---|---|
| `0x2000` | PFFCW_page | Most permissive access: RW<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0x3000` | PFCW_page | Most permissive access: RW<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| `0xF000` | PBX_IMPL_DEF_page | Most permissive access: RW |

### C2.1.1.1  PBX_CTRL_page, Postbox CTRL page

The PBX_CTRL_page characteristics are:

**Purpose**

Allows access to the configuration registers of the Postbox.

**Configuration**

**Attributes**

The PBX_CTRL_page block is of size 4KB.

This block is part of the MHUS.PBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x000` | PBX_BLK_ID | Most permissive access: RO |
| `0x010` | PBX_FEAT_SPT0 | Most permissive access: RO |
| `0x014` | PBX_FEAT_SPT1 | Most permissive access: RO |
| `0x020` | PBX_DBCH_CFG0 | Most permissive access: RO<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |
| `0x030` | PBX_FFCH_CFG0 | Most permissive access: RO<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0x040` | PBX_FCH_CFG0 | Most permissive access: RO<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| `0x100` | PBX_CTRL | Most permissive access: RW |
| `0x400 + (4 * n)`for n in 3:0 | PBX_DBCH_INT_ST<n> | Most permissive access: RO<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |
| `0x410 + (4 * n)`for n in 1:0 | PBX_FFCH_INT_ST<n> | Most permissive access: RO<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0xFC8` | PBX_IIDR | Most permissive access: RO |

| Offset | Name | Notes |
|---|---|---|
| `0xFCC` | PBX_AIDR | Most permissive access: RO |
| `0xFD0 + (4 * n)for n in 11:0` | PBX_IMPL_DEF_ID<n> | Most permissive access: RO |

### C2.1.1.1.1 PBX_BLK_ID, Postbox Block Identifier

The PBX_BLK_ID characteristics are:

**Purpose**

Identifies the block as a Postbox.

**Attributes**

PBX_BLK_ID is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_BLK_ID bit assignments are:



**Bits [31:4]**

Reserved, RES0.

**BLK_ID, bits [3:0]**

Block Identifier

Identifies the type of block that resides in this 64KB.

Reads as 0b0000

Identifies the block as the Postbox block.

Access to this field is **RO**.

**Accessing PBX_BLK_ID**

Accesses to this register use the following encodings:

**Accessible at offset 0x000 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.2 PBX_FEAT_SPT0, Postbox Feature Support 0

The PBX_FEAT_SPT0 characteristics are:

**Purpose**

Provides information on the features implemented in the Postbox.

**Attributes**

PBX_FEAT_SPT0 is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_FEAT_SPT0 bit assignments are:

| 31          24 | 23      20 | 19      16 | 15      12 | 11        8 | 7         4 | 3         0 |
|----------------|------------|------------|------------|-------------|-------------|-------------|
| RES0           | RASE_SPT   | RME_SPT    | TZE_SPT    | FCE_SPT     | FE_SPT      | DBE_SPT     |

**Bits [31:24]**

Reserved, RES0.

**RASE_SPT, bits [23:20]**

Reliability, Availability and Serviceability Extension Support

The value of this field depends on the implementation of the MHU.

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

| RASE_SPT | Meaning |
|----------|---------|
| 0b0000   | MHU does not implement the RAS extension |
| 0b0010   | MHU implements the RAS extension but does not follow the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |
| 0b0011   | MHU implements the RAS extension and follows the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |

Access to this field is **RO**.

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

Realm Management Extension Support

The value of this field depends on the implementation of the MHU and an optional reset time sampled input **LEGACY_TZ_EN**.

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

| RME_SPT | Meaning |
|---------|---------|
| 0b0000  | MHU does not implement the Realm Management extension |

| RME_SPT | Meaning |
|---------|---------|
| 0b0001 | MHU implements Realm Management extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

When RME is implemented, for the half of the MHU, there can be a **LEGACY_TZ_EN** tie-off present on that side of the MHU.

The value of the **LEGACY_TZ_EN** tie-off is sampled at reset of the side of the MHU which the tie-off is associated with.

When the sampled value of the tie-off is 0b1 the value of this field is always 0x0, otherwise the value of this field is dependent on whether RME is implemented for this half of the MHU.

Access to this field is **RO**.

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

TrustZone Extension Support

The value of this field depends on the implementation of the MHU.

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

| TZE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the TrustZone extension |
| 0b0001 | MHU implements TrustZone extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

Access to this field is **RO**.

**FCE_SPT, bits [11:8]**

Fast Channel Extension Support

The value of this field depends on the implementation of the MHU.

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

| FCE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Fast Channel extension |
| 0b0001 | MHU implements Fast Channel extension |

Access to this field is **RO**.

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

FIFO Extension Support

The value of this field depends on the implementation of the MHU.

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

| FE_SPT | Meaning |
|--------|---------|
| 0b0000 | MHU does not implement the FIFO extension |
| 0b0001 | MHU implements FIFO extension |

Access to this field is **RO**.

**DBE_SPT, bits [3:0]**

Doorbell Extension Support

The value of this field depends on the implementation of the MHU.

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

| DBE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Doorbell extension |
| 0b0001 | MHU implements Doorbell extension |

Access to this field is **RO**.

**Accessing PBX_FEAT_SPT0**

Accesses to this register use the following encodings:

**Accessible at offset 0x010 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.3 *PBX_FEAT_SPT1, Postbox Feature Support 1*

The PBX_FEAT_SPT1 characteristics are:

**Purpose**

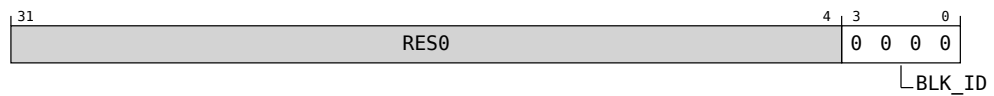Provides information on the features implemented in the Postbox.

**Attributes**

PBX_FEAT_SPT1 is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_FEAT_SPT1 bit assignments are:



**Bits [31:4]**

Reserved, RES0.

**AUTO_OP_SPT, bits [3:0]**

Auto Op Protocol Support

The value of this field depends on the implementation of the MHU.

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

| AUTO_OP_SPT | Meaning |
| --- | --- |
| 0b0000 | Auto Op(Min) is implemented |
| 0b0001 | Auto Op(Full) is implemented |

Access to this field is **RO**.

**Accessing PBX_FEAT_SPT1**

Accesses to this register use the following encodings:

**Accessible at offset 0x014 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.4 PBX_DBCH_CFG0, Postbox Doorbell Channel Configuration 0

The PBX_DBCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of DBE in Postbox.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PBX_DBCH_CFG0 are RAZ/WI.

**Attributes**

PBX_DBCH_CFG0 is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_DBCH_CFG0 bit assignments are:

| 31 | 8 | 7 | 0 |
|---|---|---|---|
| RES0 | | NUM_DBCH | |

**Bits [31:8]**

Reserved, RES0.

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

Number of Doorbell Channels

Number of DBCH implemented in the Postbox.

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

| NUM_DBCH | Meaning |
|---|---|
| `0x00..0x7F` | Number of DBCH is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing PBX_DBCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x020 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.5 PBX_FFCH_CFG0, Postbox FIFO Channel Configuration 0

The PBX_FFCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FE in Postbox.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PBX_FFCH_CFG0 are RAZ/WI.

**Attributes**

PBX_FFCH_CFG0 is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_FFCH_CFG0 bit assignments are:



**Bits [31:26]**

Reserved, RES0.

**FFCH_DEPTH, bits [25:16]**

FIFO Channel Depth

Depth of the FIFOs of the FFCHs in the Postbox.

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

| FFCH_DEPTH | Meaning |
|---|---|
| 0b0000000000..0 ↪b1111111111 | FIFO depth is N+1 bytes, where N is the value of this field. |

Access to this field is **RO**.

**Bits [15:12]**

Reserved, RES0.

**P64BA_SPT, bit [11]**

Postbox 64-bit Access Support

Whether 64-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P64BA_SPT | Meaning |
|---|---|
| 0b0 | 64-bit accesses are not supported |
| 0b1 | 64-bit accesses are supported |

Accesses must be aligned to an 64-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

When 64-bit accesses are supported the PFFCW<n>_PAY register occupies offsets 0x00-0x07 of the PFFCW.

When 64-bit accesses are not supported the PFFCW<n>_PAY register occupies offsets 0x00-0x03 of the PFFCW and offsets 0x04-0x07 of the PFFCW are RES0.

Access to this field is **RO**.

**P32BA_SPT, bit [10]**

Postbox 32-bit Access Support

Whether 32-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P32BA_SPT | Meaning |
|---|---|
| 0b0 | 32-bit accesses are not supported |
| 0b1 | 32-bit accesses are supported |

Accesses must be aligned to an 32-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**P16BA_SPT, bit [9]**

Postbox 16-bit Access Support

Whether 16-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P16BA_SPT | Meaning |
|---|---|
| 0b0 | 16-bit accesses are not supported |
| 0b1 | 16-bit accesses are supported |

Accesses must be aligned to an 16-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**P8BA_SPT, bit [8]**

Postbox 8-bit Access Support

Whether 8-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P8BA_SPT | Meaning |
|---|---|
| 0b0 | 8-bit accesses are not supported |

| P8BA_SPT | Meaning |
|---|---|
| `0b1` | 8-bit accesses are supported |

Accesses must be aligned to an 8-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

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

Number of FIFO Channels

The number of FFCHs in the Postbox.

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

| NUM_FFCH | Meaning |
|---|---|
| `0x00..0x3F` | Number of FFCHs is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing PBX_FFCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x030 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.6 PBX_FCH_CFG0, Postbox Fast Channel Configuration 0

The PBX_FCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FCE in the Postbox.

**Configuration**

This register is present only when FCE is implemented. Otherwise, direct accesses to PBX_FCH_CFG0 are RAZ/WI.
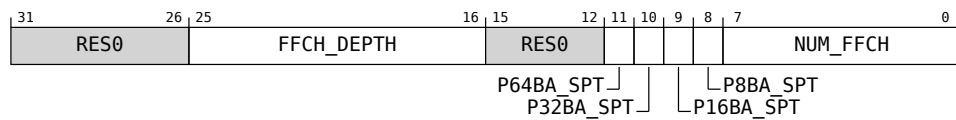
**Attributes**

PBX_FCH_CFG0 is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_FCH_CFG0 bit assignments are:
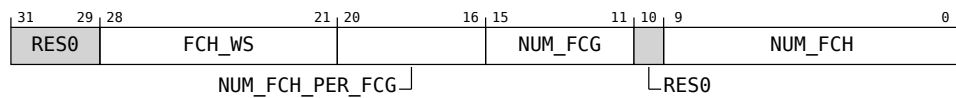


**Bits [31:29]**

Reserved, RES0.

**FCH_WS, bits [28:21]**

Fast Channel Word-Size

Number of bits each FCH implements.

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

| FCH_WS | Meaning |
|--------|---------|
| 0x20 | Fast Channel word-size is 32-bits |
| 0x40 | Fast Channel word-size is 64-bits |

Access to this field is **RO**.

**NUM_FCH_PER_FCG, bits [20:16]**

Number of Fast Channels per Fast Channel Group

Number of FCHs implemented in FCGs for Postbox.

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

| NUM_FCH_PER_FCG | Meaning |
|-----------------|---------|
| 0b00000..0b11111 | Number of FCHs per FCG is N+1, where N is the value of this field |

Access to this field is **RO**.

**NUM_FCG, bits [15:11]**

Number of Fast Channel Groups

Number of FCGs implemented in Postbox

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

| NUM_FCG | Meaning |
|---|---|
| 0b00000..0 ↪b11111 | Number of FCGs is N+1, where N is the value of this field |

Access to this field is **RO**.

**Bit [10]**

Reserved, RES0.

**NUM_FCH, bits [9:0]**

Number of Fast Channels

Number of FCHs implemented in Postbox.

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

| NUM_FCH | Meaning | Applies |
|---|---|---|
| 0b0000000000..0 ↪b0111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x40 |
| 0b0000000000..0 ↪b1111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x20 |

Access to this field is **RO**.

**Accessing PBX_FCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x040 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.7 PBX_CTRL, Postbox Control

The PBX_CTRL characteristics are:

**Purpose**

Configures the behavior of the Postbox.

**Attributes**

PBX_CTRL is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_CTRL bit assignments are:
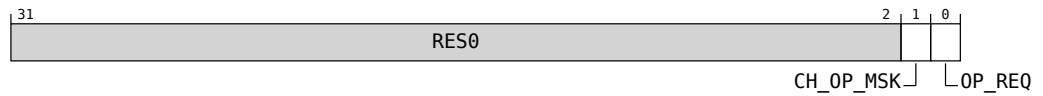


**Bits [31:2]**

Reserved, RES0.

**CH_OP_MSK, bit [1]**

Channel Operational Mask

Controls whether channels need to be idle to allow a controlled entry of the MHUS into a non-operational state.

| CH_OP_MSK | Meaning |
|-----------|---------|
| 0b0 | Channels must be idle to enter a non-operational state |
| 0b1 | Channels status is ignored when considering entry into a non-operational state |

This field has no effect on entry into a non-operation state in an uncontrolled manner for the MHUS.

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**OP_REQ, bit [0]**

Operational Request

Controls whether the MHUS is required to remain in an operational state.

| OP_REQ | Meaning |
|--------|---------|
| 0b0 | Postbox is not requested to remain in the operational state |
| 0b1 | Postbox is requested to remain in the operational state |

The reset behavior of this field is:

  • On a MHUS reset, this field resets to `0b0`.

**Accessing PBX_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x100 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RW**.

### C2.1.1.1.8   PBX_DBCH_INT_ST&lt;n&gt;, Postbox Doorbell Channel Interrupt Status n, n = 0 - 3

The PBX_DBCH_INT_ST&lt;n&gt; characteristics are:

**Purpose**

Indicates whether there is an interrupt outstanding for a DBCH

PBX_DBCH_INT_ST0 has status fields for DBCHs 0 to 31

PBX_DBCH_INT_ST1 has status fields for DBCHs 32 to 63

PBX_DBCH_INT_ST2 has status fields for DBCHs 64 to 95

PBX_DBCH_INT_ST3 has status fields for DBCHs 96 to 127

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PBX_DBCH_INT_ST&lt;n&gt; are RAZ/WI.

**Attributes**

PBX_DBCH_INT_ST&lt;n&gt; is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_DBCH_INT_ST&lt;n&gt; bit assignments are:
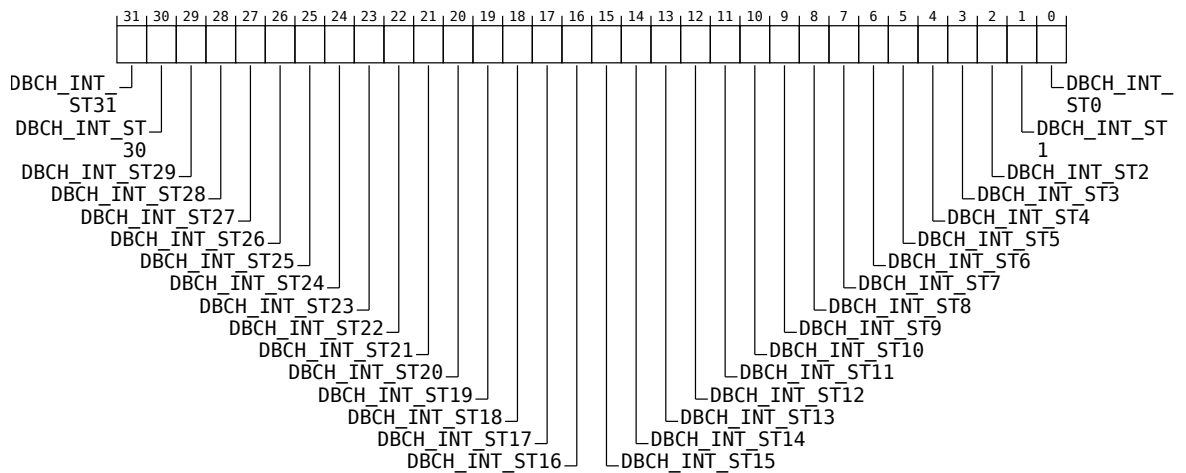


**DBCH_INT_ST&lt;x&gt;, bits [x], for x = 31 to 0**

Doorbell Channel Interrupt Status

Each bit indicates whether there is an outstanding interrupt for the DBCH, that is contributing to the Postbox Combined interrupt.

| DBCH_INT_ST&lt;x&gt; | Meaning |
|---|---|
| 0b0 | No interrupt outstanding for the DBCH or the DBCH is not configured to contribute into the Postbox Combined interrupt. |

| DBCH_INT_ST<x> | Meaning |
|---|---|
| `0b1` | Interrupt outstanding for the DBCH and the DBCH is configured to contribute into the Postbox Combined interrupt. |

Any fields which are not assigned to a DBCH are Reserved and treated as RES0

**Accessing PBX_DBCH_INT_ST<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0x400 + (4 * n) from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.9 PBX_FFCH_INT_ST<n>, Postbox FIFO Channel Interrupt Status n, n = 0 - 1

The PBX_FFCH_INT_ST<n> characteristics are:

**Purpose**

Indicates whether there is an interrupt outstanding for the FFCH.

PBX_FFCH_INT_ST0 has status fields for FFCHs 0 to 31

PBX_FFCH_INT_ST1 has status fields for FFCHs 32 to 63

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PBX_FFCH_INT_ST<n> are RAZ/WI.

**Attributes**

PBX_FFCH_INT_ST<n> is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_FFCH_INT_ST<n> bit assignments are:



**FFCH_INT_ST<x>, bits [x], for x = 31 to 0**

FIFO Channel Interrupt Status

Each bit indicates whether there is an outstanding interrupt for the FFCH, that is contributing to the Postbox Combined interrupt

| FFCH_INT_ST<x> | Meaning |
| --- | --- |
| 0b0 | No interrupt outstanding for the FFCH or the FFCH is not configured to contribute into the Postbox Combined interrupt. |
| 0b1 | Interrupt outstanding for the FFCH and the FFCH is configured to contribute into the Postbox Combined interrupt. |

Any fields which are not assigned to a FFCH are Reserved and treated as RAZ/WI

**Accessing PBX_FFCH_INT_ST<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0x410 + (4 * n) from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.10 PBX_IIDR, Postbox Implementer Identification Register

The PBX_IIDR characteristics are:

**Purpose**

This field provides information on the implementation of the MHU

**Attributes**

PBX_IIDR is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_IIDR bit assignments are:

| 31 PRODUCT_ID 20 | 19 VARIANT 16 | 15 REVISION 12 | 11 IMPLEMENTER 0 |
|---|---|---|---|

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

Product ID of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Variant or major revision of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Revision or minor version of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**IMPLEMENTER, bits [11:0]**

Implementer ID

Contains the JEP106 identification information as follows:

- 11:8 - JEP106 continuation code of implementer
- 7 - Always 0
- 6:0 - JEP106 identity code of implementer

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**Accessing PBX_IIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFC8 from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.11 PBX_AIDR, Postbox Architecture Identification Register

The PBX_AIDR characteristics are:

**Purpose**

Provides information on the version of the MHU architecture implemented in this implementation of the MHU.

**Attributes**

PBX_AIDR is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_AIDR bit assignments are:



**Bits [31:8]**

Reserved, RES0.

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

MHU Architecture Major Revision

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

| ARCH_MAJOR_REV | Meaning |
| --- | --- |
| 0b0000 | MHUv1 or unknown architecture versions |
| 0b0001 | MHUv2 |
| 0b0010 | MHUv3 |

All other values are Reserved

Access to this field is **RO**.

**ARCH_MINOR_REV, bits [3:0]**

MHU Architecture Minor Revision

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

| ARCH_MINOR_REV | Meaning |
| --- | --- |
| 0b0000 | Minor revision 0 of the major architecture |
| 0b0001 | Minor revision 1 of the major architecture |

All other values are Reserved

Access to this field is **RO**.

**Additional information**

The legal combinations for the ARCH_MAJOR_REV and ARCH_MINOR_REV fields are as follows:

| ARCH_MAJOR_REV | ARCH_MINOR_REV | Description |
| --- | --- | --- |
| 0x0 | 0x0 | MHUv1 |
| 0x1 | 0x0 | MHUv2.0 |
| 0x1 | 0x1 | MHUv2.1 |
| 0x2 | 0x0 | MHUv3.0 |

MHU implementations based on this architecture, have the ARCH_MAJOR_REV set to 0x2 and ARCH_MINOR_REV set to 0x0.

**Accessing PBX_AIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFCC from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.1.12 PBX_IMPL_DEF_ID<n>, IMPDEF Register, n = 0 - 11

The PBX_IMPL_DEF_ID<n> characteristics are:

**Purpose**

This register is for IMPLEMENTATION DEFINED Identification Registers which can be used to identify the implementation of the MHU Postbox

An implementation can do the following with this register:

- Chose whether a register is present or not.

- The name of the register.

- The access permission of the registers, so long as they are not more permissive than the architecture defines for the IMPLEMENTATION DEFINED register.

- The names, behavior and reset value of any fields.

- The access permissions of any fields, so long as they are not more permissive than the register.

If the PBX_IMPL_DEF_ID<n> is not present then the location is Reserved and treated as RES0
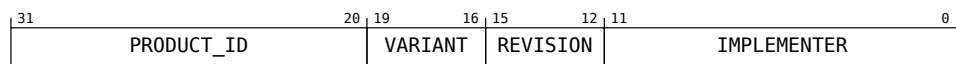
**Attributes**

PBX_IMPL_DEF_ID<n> is a 32-bit register.

This register is part of the MHUS.PBX.PBX_CTRL_page block.

**Field descriptions**

The PBX_IMPL_DEF_ID<n> bit assignments are:

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

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

IMPLEMENTATION DEFINED

**Accessing PBX_IMPL_DEF_ID<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0xFD0 + (4 * n) from MHUS.PBX.PBX_CTRL_page**

Access on this interface is **RO**.

### C2.1.1.2   PDBCW_page, Postbox Doorbell Channel Window Page

The PDBCW_page characteristics are:

**Purpose**

Allows access to the PDBCW.

**Configuration**

This Register Block is present only when DBE is implemented.

**Attributes**

The PDBCW_page block is of size 4KB.

This block is part of the MHUS.PBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
| --- | --- | --- |
| `0x0000 + (32 * n)`for n in ↪`127:0` | PDBCW<n>_ST | Most permissive access: RO |
| `0x000C + (32 * n)`for n in ↪`127:0` | PDBCW<n>_SET | Most permissive access: WO/RAZ |
| `0x0010 + (32 * n)`for n in ↪`127:0` | PDBCW<n>_INT_ST | Most permissive access: RO |
| `0x0014 + (32 * n)`for n in ↪`127:0` | PDBCW<n>_INT_CLR | Most permissive access: WO/RAZ |
| `0x0018 + (32 * n)`for n in ↪`127:0` | PDBCW<n>_INT_EN | Most permissive access: RW |
| `0x001C + (32 * n)`for n in ↪`127:0` | PDBCW<n>_CTRL | Most permissive access: RW |

### C2.1.1.2.1 PDBCW<n>_ST, Postbox Doorbell Channel Window <n> Status, n = 0 - 127

The PDBCW<n>_ST characteristics are:

**Purpose**

Status of the flags for DBCH<n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PDBCW<n>_ST are RAZ/WI.
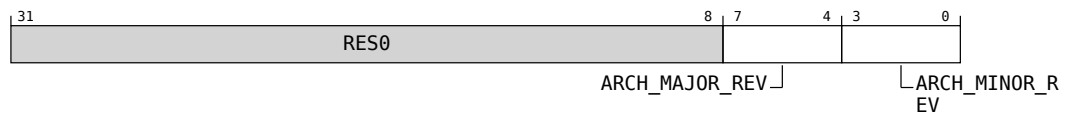
**Attributes**

PDBCW<n>_ST is a 32-bit register.

This register is part of the MHUS.PBX.PDBCW_page block.

**Field descriptions**

The PDBCW<n>_ST bit assignments are:
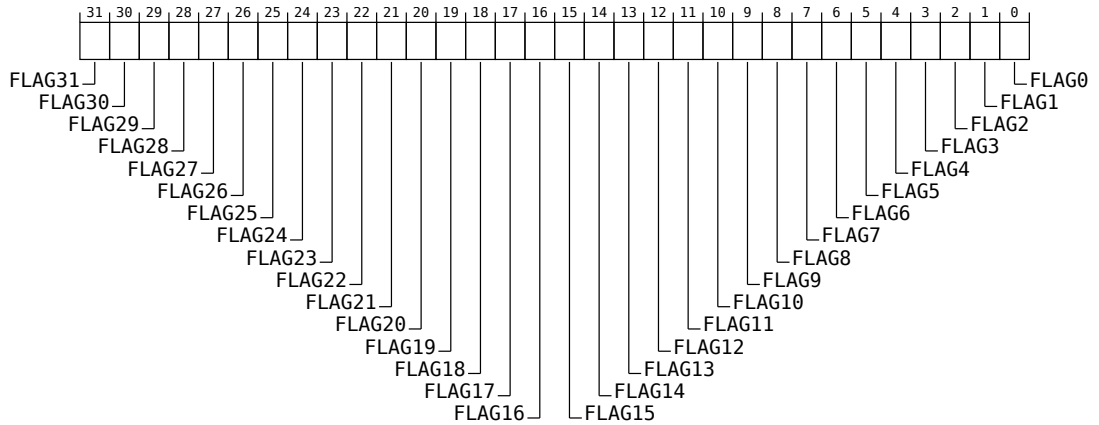


**FLAG<x>, bits [x], for x = 31 to 0**

Flag

Indicates the status of Flag bit <x> of the DBCH.

| FLAG<x> | Meaning |
|---------|---------|
| 0b0 | Flag<x> bit is not set |
| 0b1 | Flag<x> bit is set |

The reset behavior of this field is:

 • On a MHUR reset, this field resets to 0b0.

**Accessing PDBCW<n>_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0000 + (32 * n) from MHUS.PBX.PDBCW_page**

Access on this interface is **RO**.

### C2.1.1.2.2   PDBCW<n>_SET, Postbox Doorbell Channel Window <n> Set, n = 0 - 127

The PDBCW<n>_SET characteristics are:

**Purpose**

> Set flags of DBCH<n>.

**Configuration**

> This register is present only when DBE is implemented. Otherwise, direct accesses to PDBCW<n>_SET
> are RAZ/WI.

**Attributes**

> PDBCW<n>_SET is a 32-bit register.

> This register is part of the MHUS.PBX.PDBCW_page block.

**Field descriptions**

The PDBCW<n>_SET bit assignments are:
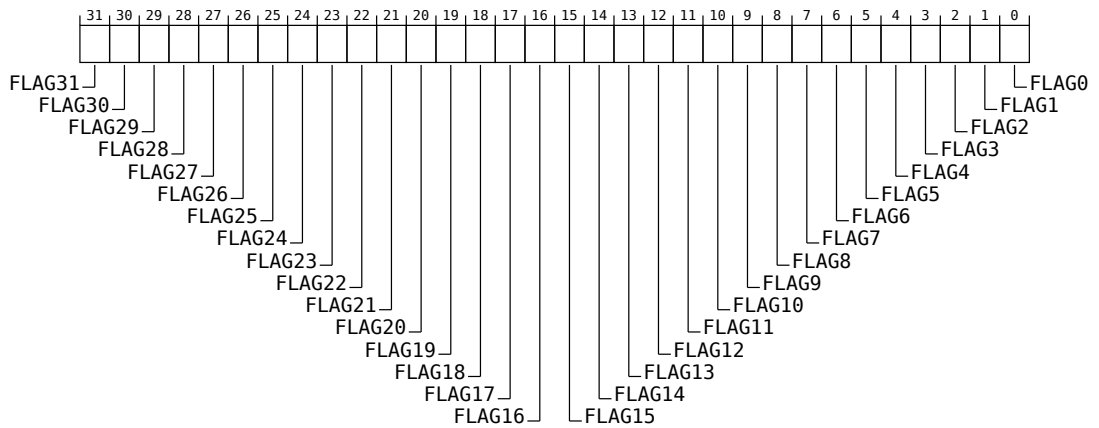


**FLAG<x>, bits [x], for x = 31 to 0**

| FLAG<x> | Meaning |
|---------|---------|
| 0b0 | No effect |
| 0b1 | Sets the associated bit in the PDBCW<n>_ST and MDBCW<n>_ST registers to 0b1 |

**Accessing PDBCW<n>_SET**

Accesses to this register use the following encodings:

**Accessible at offset 0x000C + (32 * n) from MHUS.PBX.PDBCW_page**

Access on this interface is **WO/RAZ**.

### C2.1.1.2.3 PDBCW<n>_INT_ST, Postbox Doorbell Channel Window <n> Interrupt Status, n = 0 - 127

The PDBCW<n>_INT_ST characteristics are:

**Purpose**

Provides the status of interrupts for DBCH <n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PDBCW<n>_INT_ST are RAZ/WI.

**Attributes**

PDBCW<n>_INT_ST is a 32-bit register.

This register is part of the MHUS.PBX.PDBCW_page block.

**Field descriptions**

The PDBCW<n>_INT_ST bit assignments are:
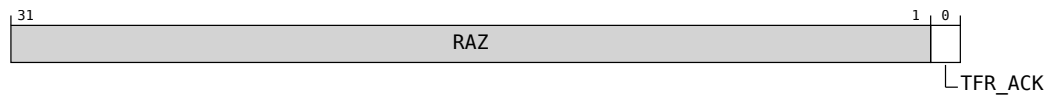


**Bits [31:1]**

Reserved, RAZ.

**TFR_ACK, bit [0]**

Transfer Acknowledge

Indicates whether a Transfer Acknowledge event has occurred.

| TFR_ACK | Meaning |
| --- | --- |
| 0b0 | Transfer Acknowledge event hsa not occurred |
| 0b1 | Transfer Acknowledge event has occurred |

**Accessing PDBCW<n>_INT_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0010 + (32 * n) from MHUS.PBX.PDBCW_page**

Access on this interface is **RO**.

### C2.1.1.2.4    PDBCW<n>_INT_CLR, Postbox Doorbell Channel Window <n> Interrupt Clear, n = 0 - 127

The PDBCW<n>_INT_CLR characteristics are:

**Purpose**

Allows clearing of any interrupts for DBCH <n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PDBCW<n>_INT_CLR are RAZ/WI.

**Attributes**

PDBCW<n>_INT_CLR is a 32-bit register.

This register is part of the MHUS.PBX.PDBCW_page block.

**Field descriptions**

The PDBCW<n>_INT_CLR bit assignments are:
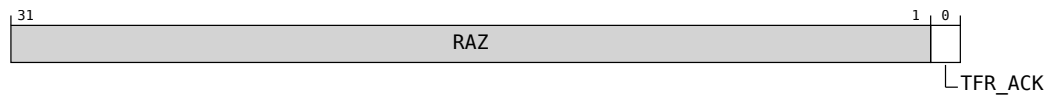


**Bits [31:1]**

Reserved, RAZ.

**TFR_ACK, bit [0]**

| TFR_ACK | Meaning |
|---------|---------|
| 0b0 | No effect |
| 0b1 | Clears the Transfer Acknowledge field in the PDBCW<n>_INT_ST register |

**Accessing PDBCW<n>_INT_CLR**

Accesses to this register use the following encodings:

**Accessible at offset 0x0014 + (32 * n) from MHUS.PBX.PDBCW_page**

Access on this interface is **WO/RAZ**.

### C2.1.1.2.5 PDBCW<n>_INT_EN, Postbox Doorbell Channel Window <n> Interrupt Enable, n = 0 - 127

The PDBCW<n>_INT_EN characteristics are:

**Purpose**

Configures the interrupts of DBCH <n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PDBCW<n>_INT_EN are RAZ/WI.

**Attributes**

PDBCW<n>_INT_EN is a 32-bit register.

This register is part of the MHUS.PBX.PDBCW_page block.

**Field descriptions**

The PDBCW<n>_INT_EN bit assignments are:
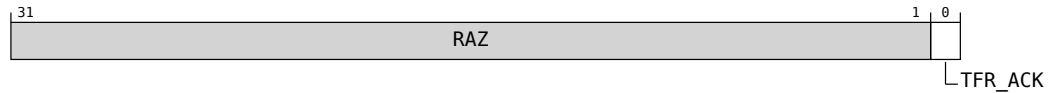


**Bits [31:1]**

Reserved, RAZ.

**TFR_ACK, bit [0]**

Transfer Acknowledge

Controls whether a Transfer Acknowledge event generates a Channel Transfer Acknowledge interrupt.

| TFR_ACK | Meaning |
| --- | --- |
| 0b0 | A Channel Transfer Acknowledge event does not generate a Channel Transfer Acknowledge interrupt. |
| 0b1 | A Channel Transfer Acknowledge event generates a Channel Transfer Acknowledge interrupt. |

The reset behavior of this field is:

• On a MHUS reset, this field resets to 0b0.

**Accessing PDBCW<n>_INT_EN**

Accesses to this register use the following encodings:

**Accessible at offset 0x0018 + (32 * n) from MHUS.PBX.PDBCW_page**

Access on this interface is **RW**.

### C2.1.1.2.6 PDBCW<n>_CTRL, Postbox Doorbell Channel Window <n> Control, n = 0 - 127

The PDBCW<n>_CTRL characteristics are:

**Purpose**

Configures the behavior of DBCH <n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to PDBCW<n>_CTRL are RAZ/WI.

**Attributes**

PDBCW<n>_CTRL is a 32-bit register.

This register is part of the MHUS.PBX.PDBCW_page block.

**Field descriptions**

The PDBCW<n>_CTRL bit assignments are:
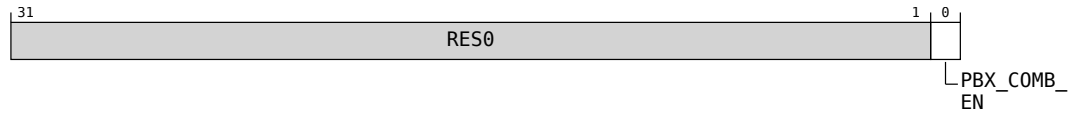


**Bits [31:1]**

Reserved, RES0.

**PBX_COMB_EN, bit [0]**

Postbox Combined Enable

Controls whether interrupts from the DBCH contribute to the Postbox Combined interrupt.

| PBX_COMB_EN | Meaning |
| --- | --- |
| 0b0 | DBCH interrupts do not contribute to the Postbox Combined interrupt |
| 0b1 | DBCH interrupts contribute to the Postbox Combined interrupt |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b1.

**Accessing PDBCW<n>_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x001C + (32 * n) from MHUS.PBX.PDBCW_page**

Access on this interface is **RW**.

### C2.1.1.3 PFFCW_page, Postbox FIFO Channel Windows Page

The PFFCW_page characteristics are:

**Purpose**

> Allows access to the PFFCW.
>
> Depending on the size of the access to the PFFCW_PAY register and whether the access size is supported the access is treated as follows:
>
> - 8-bit access and PBX_FFCH_CFG0.P8BA_SPT == 0b1 the access behaves as defined in PFFCW<n>_PAY8.
>
> - 8-bit access and PBX_FFCH_CFG0.P8BA_SPT == 0b0 the access is treated as an unsupported access.
>
> - 16-bit access and PBX_FFCH_CFG0.P16BA_SPT == 0b1 the access behaves as defined in PFFCW<n>_PAY16.
>
> - 16-bit access and PBX_FFCH_CFG0.P16BA_SPT == 0b0 the access is treated as an unsupported access.
>
> - 32-bit access and PBX_FFCH_CFG0.P32BA_SPT == 0b1 the access behaves as defined in PFFCW<n>_PAY32.
>
> - 32-bit access and PBX_FFCH_CFG0.P32BA_SPT == 0b0 the access is treated as an unsupported access.
>
> - 64-bit access and PBX_FFCH_CFG0.P64BA_SPT == 0b1 the access behaves as defined in PFFCW<n>_PAY64.
>
> - 64-bit access and PBX_FFCH_CFG0.P64BA_SPT == 0b0 the access is treated as an unsupported access.

**Configuration**

> This Register Block is present only when FE is implemented.

**Attributes**

> The PFFCW_page block is of size 4KB.
>
> This block is part of the MHUS.PBX block.

**Default access**

> Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
| --- | --- | --- |
| `0x000 + (64 * n)`for n in 63:0 | PFFCW<n>_PAY8 | Most permissive access: RW Register condition: When P8BA_SPT Accessor condition: When P8BA_SPT |
| `0x000 + (64 * n)`for n in 63:0 | PFFCW<n>_PAY8 | Most permissive access: RW Register condition: When P8BA_SPT Accessor condition: When P8BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x000 + (64 * n)for n in 63:0` | PFFCW<n>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P16BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | PFFCW<n>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P16BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | PFFCW<n>_PAY32 | Most permissive access: RW<br>Register condition: When P32BA_SPT<br>Accessor condition: When P32BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | PFFCW<n>_PAY32 | Most permissive access: RW<br>Register condition: When P32BA_SPT<br>Accessor condition: When P32BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | PFFCW<n>_PAY64 | Most permissive access: RW<br>Register condition: When P64BA_SPT<br>Accessor condition: When P64BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | PFFCW<n>_PAY64 | Most permissive access: RW<br>Register condition: When P64BA_SPT<br>Accessor condition: When P64BA_SPT |
| `0x001 + (64 * n)for n in 63:0` | PFFCW<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P8BA_SPT |
| `0x001 + (64 * n)for n in 63:0` | PFFCW<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P8BA_SPT |
| `0x002 + (64 * n)for n in 63:0` | PFFCW<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P8BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x002 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P8BA_SPT |
| `0x002 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P16BA_SPT |
| `0x002 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P16BA_SPT |
| `0x003 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P8BA_SPT |
| `0x003 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P8BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P64BA_SPT and P16BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | PFFCW\<n\>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P64BA_SPT and P16BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x004 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY32 | Most permissive access: RW<br>Register condition: When P32BA_SPT<br>Accessor condition: When P64BA_SPT and P32BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY32 | Most permissive access: RW<br>Register condition: When P32BA_SPT<br>Accessor condition: When P64BA_SPT and P32BA_SPT |
| `0x005 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x005 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x006 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x006 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x006 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P64BA_SPT and P16BA_SPT |
| `0x006 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY16 | Most permissive access: RW<br>Register condition: When P16BA_SPT<br>Accessor condition: When P64BA_SPT and P16BA_SPT |
| `0x007 + (64 * n)for n in 63:0` | PFFCW\<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x007 + (64 * n)for n in 63:0` | PFFCW<n>_PAY8 | Most permissive access: RW<br>Register condition: When P8BA_SPT<br>Accessor condition: When P64BA_SPT and P8BA_SPT |
| `0x0008 + (64 * n)for n in 63:0` | PFFCW<n>_FLG | Most permissive access: RW |
| `0x0010 + (64 * n)for n in 63:0` | PFFCW<n>_INT_ST | Most permissive access: RO |
| `0x0014 + (64 * n)for n in 63:0` | PFFCW<n>_INT_CLR | Most permissive access: WO/RAZ |
| `0x0018 + (64 * n)for n in 63:0` | PFFCW<n>_INT_EN | Most permissive access: RW |
| `0x0020 + (64 * n)for n in 63:0` | PFFCW<n>_CTRL | Most permissive access: RW |
| `0x0024 + (64 * n)for n in 63:0` | PFFCW<n>_ST | Most permissive access: RO |
| `0x0028 + (64 * n)for n in 63:0` | PFFCW<n>_ACK_CNT | Most permissive access: RO |
| `0x002C + (64 * n)for n in 63:0` | PFFCW<n>_TIDE | Most permissive access: RW |

### C2.1.1.3.1 PFFCW<n>_PAY8, Postbox FIFO Channel Window <n> Payload (8-bit access), n = 0 - 63

The PFFCW<n>_PAY8 characteristics are:

**Purpose**

> An 8-bit access to the PFFCW<n>_PAY register. Acesses must be aligned to any 8-bit boundary within the PFFCW_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

> 8-bit accesses are only supported, if PBX_FFCH_CFG0.P8BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

> If P64BA_SPT is set to 1 the PFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Postbox FIFO Channel Window <n>.

> If P64BA_SPT is set to 0 the PFFCW<n>_PAY register occupies offsets 0x00-0x04 and offsets 0x05 - 0x7 are reserved, within the Postbox FIFO Channel Window <n>.

> Write access push the byte that is written to this offset onto the FIFO, if the FIFO has at least one byte of free space.

> Read accesses return the number of invalid bytes in the FIFO and whether the previous push operation generated an error or not.

**Configuration**

> This register is present only when FE is implemented and P8BA_SPT. Otherwise, direct accesses to PFFCW<n>_PAY8 are RAZ/WI.

**Attributes**

> PFFCW<n>_PAY8 is a 8-bit register.

> This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_PAY8 bit assignments are:

**When access is a write:**



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

Payload to push onto FIFO

Causes the written byte of data to be pushed onto the FIFO, if there is at least one byte of free space in the FIFO and sets the PFFCW<n>_ST.PPE field to be set to 0b0.

If there is no free space in the FIFO, then no bytes are pushed onto the FIFO and the PFFCW<n>_ST.PPE field is set to 0b1

The value written to this field has no effect on the operation of the FIFO.

All data flags in the PFFCW<n>_FLG register are associated with the one byte pushed onto the FIFO

If the Transfer Delineation Mode for the FIFO is set to Partial or Auto Flag then on a write which sets PFFCW<n>_ST.PPE field to 0b0, the values of data flags in the PFFCW<n>_FLG register are updated.

**When access is a read:**

**PPE, bit [7]**

Previous Push Error

Indicates whether a previous push to the FIFO caused an error

An error occurs due to the FIFO not having enough space to push all the bytes provided in the last write to the PFFCW_PAY register.

| PPE | Meaning |
| --- | --- |
| 0b0 | No error has occurred on the last push operation |
| 0b1 | An error has occurred on the last push operation |

**FFS, bits [6:0]**

FIFO Free Space

Indicates the number of invalid bytes in the FIFO

| FFS | Meaning | Applies |
| --- | --- | --- |
| 0b0000000 | No free bytes in the FIFO | |
| 0b0000001..0↪b1111110 | Number of free bytes in the FIFO | |
| 0b1111111 | 127 bytes free in the FIFO | When FFCH_DEPTH <= 127 |
| 0b1111111 | 127 or more bytes free in the FIFO | When FFCH_DEPTH > 127 |

The maximum value returned is never greater than the FIFO Depth

**Accessing PFFCW<n>_PAY8**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**Read at offset 0x001 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**Read at offset 0x002 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**Read at offset 0x003 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**When P64BA_SPT**

**Read at offset 0x004 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**When P64BA_SPT**

**Read at offset 0x005 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**When P64BA_SPT**

**Read at offset 0x006 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**When P64BA_SPT**

**Read at offset 0x007 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY8;
```

**Write at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**Write at offset 0x001 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**Write at offset 0x002 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**Write at offset 0x003 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**When P64BA_SPT**

**Write at offset 0x004 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**When P64BA_SPT**

**Write at offset 0x005 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**When P64BA_SPT**

**Write at offset 0x006 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

**When P64BA_SPT**

**Write at offset 0x007 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY8 = value;
```

### C2.1.1.3.2 PFFCW<n>_PAY16, Postbox FIFO Channel Window <n> Payload (16-bit access), n = 0 - 63

The PFFCW<n>_PAY16 characteristics are:

**Purpose**

A 16-bit access to the PFFCW<n>_PAY register. Acesses must be aligned to any 16-bit boundary within the PFFCW_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

16-bit accesses are only supported, if PBX_FFCH_CFG0.P16BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If P64BA_SPT is set to 1 the PFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Postbox FIFO Channel Window <n>.

If P64BA_SPT is set to 0 the PFFCW<n>_PAY register occupies offsets 0x00-0x04 and offsets 0x05 - 0x7 are reserved, within the Postbox FIFO Channel Window <n>.

Write access pushes the two bytes that are written to this offset onto the FIFO, if the FIFO has at least two byte of free space.

Read accesses return the number of invalid bytes in the FIFO and whether the previous push operation generated an error or not.

**Configuration**

This register is present only when FE is implemented and P16BA_SPT. Otherwise, direct accesses to PFFCW<n>_PAY16 are RAZ/WI.

**Attributes**

PFFCW<n>_PAY16 is a 16-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_PAY16 bit assignments are:

**When access is a write:**

| 15 | 0 |
|---|---|
| PAY | |
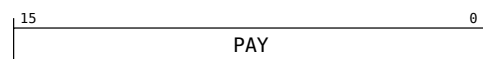
**PAY, bits [15:0]**

Payload to push onto FIFO

Causes these written bytes of data to be pushed onto the FIFO, if there is at least two byte of free space in the FIFO and sets the PFFCW<n>_ST.PPE field to be set to 0b0.

If there is less than two bytes of free space in the FIFO, then no bytes are pushed onto the FIFO and the PFFCW<n>_ST.PPE field is set to 0b1

The value written to this field has no effect on the operation of the FIFO.

The order in which bytes are pushed onto the FIFO depends on the value of the PFFCW<n>_CTRL.MSBF field as follows:

- 0b0 - Bytes are pushed onto the FIFO starting with the LSB.

- 0b1 - Bytes are pushed onto the FIFO starting with the MSB.

The MHU is a little endian device and considers the LSB to be the byte which is written to the lowest offsets accessed by the write.

SOT flag is always associated with the first byte pushed onto the FIFO and EOT and ACK flags are associated with the last byte pushed onto the FIFO

If the Transfer Delineation Mode for the FIFO is set to Partial or Auto Flag then on a write which sets PFFCW<n>_ST.PPE field to 0b0, the values of data flags in the PFFCW<n>_FLG register are updated.

**When access is a read:**



**PPE, bit [15]**

Previous Push Error

Indicates whether a previous push to the FIFO caused an error

An error occurs due to the FIFO not having enough space to push all the bytes provided in the last write to the PFFCW_PAY register.

| PPE | Meaning |
|-----|---------|
| 0b0 | No error has occurred on the last push operation |
| 0b1 | An error has occurred on the last push operation |

**Bits [14:11]**

Reserved, RES0.

**FFS, bits [10:0]**

FIFO Free Space

Indicates the number of invalid bytes in the FIFO

| FFS | Meaning |
|-----|---------|
| 0b00000000000 | No free bytes in the FIFO |
| 0b00000000001..0↪b01111111111 | Number of free bytes in the FIFO |
| 0b10000000000 | 1024 bytes free in the FIFO |

The maximum value returned is never greater than the FIFO Depth

**Accessing PFFCW<n>_PAY16**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY16;
```

**Read at offset 0x002 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY16;
```

**When P64BA_SPT**

**Read at offset 0x004 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY16;
```

**When P64BA_SPT**

**Read at offset 0x006 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY16;
```

**Write at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY16 = value;
```

**Write at offset 0x002 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY16 = value;
```

**When P64BA_SPT**

**Write at offset 0x004 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY16 = value;
```

**When P64BA_SPT**

**Write at offset 0x006 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY16 = value;
```

### C2.1.1.3.3 PFFCW<n>_PAY32, Postbox FIFO Channel Window <n> Payload (32-bit access), n = 0 - 63

The PFFCW<n>_PAY32 characteristics are:

**Purpose**

A 32-bit access to the PFFCW<n>_PAY register. Acesses must be aligned to any 32-bit boundary within the PFFCW_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

32-bit accesses are only supported, if PBX_FFCH_CFG0.P32BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If P64BA_SPT is set to 1 the PFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Postbox FIFO Channel Window <n>.

If P64BA_SPT is set to 0 the PFFCW<n>_PAY register occupies offsets 0x00-0x04 and offsets 0x05 - 0x7 are reserved, within the Postbox FIFO Channel Window <n>.

Write access pushes the four bytes that are written to this offset onto the FIFO, if the FIFO has at least four byte of free space.

Read accesses return the number of invalid bytes in the FIFO and whether the previous push operation generated an error or not.

**Configuration**

This register is present only when FE is implemented and P32BA_SPT. Otherwise, direct accesses to PFFCW<n>_PAY32 are RAZ/WI.

**Attributes**

PFFCW<n>_PAY32 is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_PAY32 bit assignments are:

**When access is a write:**

| 31 | 0 |
|---|---|
| PAY | |
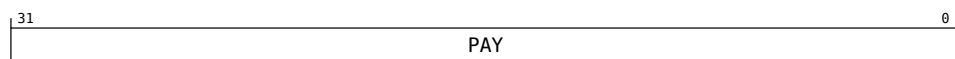
**PAY, bits [31:0]**

Payload to push onto FIFO

Causes these written bytes of data to be pushed onto the FIFO, if there is at least four byte of free space in the FIFO and sets the PFFCW<n>_ST.PPE field to be set to 0b0.

If there is less than four bytes of free space in the FIFO, then no bytes are pushed onto the FIFO and the PFFCW<n>_ST.PPE field is set to 0b1

The value written to this field has no effect on the operation of the FIFO.

The order in which bytes are pushed onto the FIFO depends on the value of the PFFCW<n>_CTRL.MSBF field as follows:
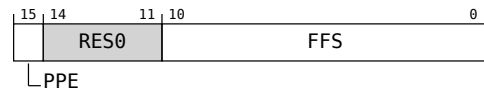
- 0b0 - Bytes are pushed onto the FIFO starting with the LSB.

- 0b1 - Bytes are pushed onto the FIFO starting with the MSB.

The MHU is a little endian device and considers the LSB to be the byte which is written to the lowest offsets accessed by the write.

SOT flag is always associated with the first byte pushed onto the FIFO and EOT and ACK flags are associated with the last byte pushed onto the FIFO

If the Transfer Delineation Mode for the FIFO is set to Partial or Auto Flag then on a write which sets PFFCW<n>_ST.PPE field to 0b0, the values of data flags in the PFFCW<n>_FLG register are updated.

**When access is a read:**



**PPE, bit [31]**

Previous Push Error

Indicates whether a previous push to the FIFO caused an error

An error occurs due to the FIFO not having enough space to push all the bytes provided in the last write to the PFFCW_PAY register.

| PPE | Meaning |
|-----|---------|
| 0b0 | No error has occurred on the last push operation |
| 0b1 | An error has occurred on the last push operation |

**Bits [30:11]**

Reserved, RES0.

**FFS, bits [10:0]**

FIFO Free Space

Indicates the number of invalid bytes in the FIFO

| FFS | Meaning |
|-----|---------|
| 0b00000000000 | No free bytes in the FIFO |
| 0b00000000001..0 ↪b01111111111 | Number of free bytes in the FIFO |
| 0b10000000000 | 1024 bytes free in the FIFO |

The maximum value returned is never greater than the FIFO Depth

**Accessing PFFCW<n>_PAY32**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY32;
```

**When P64BA_SPT**

**Read at offset 0x004 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY32;
```

**Write at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY32 = value;
```

**When P64BA_SPT**

**Write at offset 0x004 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY32 = value;
```

### C2.1.1.3.4 PFFCW<n>_PAY64, Postbox FIFO Channel Window <n> Payload (64-bit access), n = 0 - 63

The PFFCW<n>_PAY64 characteristics are:

**Purpose**

A 64-bit access to the PFFCW<n>_PAY register. Acesses must be aligned to any 64-bit boundary within the PFFCW_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

64-bit accesses are only supported, if PBX_FFCH_CFG0.P64BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

When PBX_FFCH_CFG0.P64BA_SPT is set to 0b1, the PFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Postbox FIFO Channel Window <n>.

Write access pushes the eight bytes that are written to this offset onto the FIFO, if the FIFO has at least eight byte of free space.

Read accesses return the number of invalid bytes in the FIFO and whether the previous push operation generated an error or not.

**Configuration**

This register is present only when FE is implemented and P64BA_SPT. Otherwise, direct accesses to PFFCW<n>_PAY64 are RAZ/WI.

**Attributes**

PFFCW<n>_PAY64 is a 64-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_PAY64 bit assignments are:

**When access is a write:**

| 63 | PAY | 32 |
|---|---|---|

| 31 | PAY | 0 |
|---|---|---|
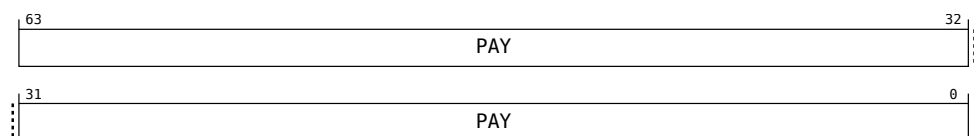
**PAY, bits [63:0]**

Payload to push onto FIFO

Causes these written bytes of data to be pushed onto the FIFO, if there is at least eight byte of free space in the FIFO and sets the PFFCW<n>_ST.PPE field to be set to 0b0.

If there is less than eight bytes of free space in the FIFO, then no bytes are pushed onto the FIFO and the PFFCW<n>_ST.PPE field is set to 0b1

The value written to this field has no effect on the operation of the FIFO.

The order in which bytes are pushed onto the FIFO depends on the value of the PFFCW<n>_CTRL.MSBF field as follows:
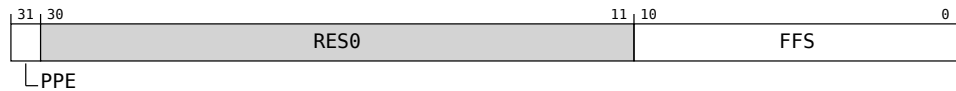
- 0b0 - Bytes are pushed onto the FIFO starting with the LSB.

- 0b1 - Bytes are pushed onto the FIFO starting with the MSB.

The MHU is a little endian device and considers the LSB to be the byte which is written to the lowest offsets accessed by the write.

SOT flag is always associated with the first byte pushed onto the FIFO and EOT and ACK flags are associated with the last byte pushed onto the FIFO

If the Transfer Delineation Mode for the FIFO is set to Partial or Auto Flag then on a write which sets PFFCW<n>_ST.PPE field to 0b0, the values of data flags in the PFFCW<n>_FLG register are updated.

**When access is a read:**



**PPE, bit [63]**

Previous Push Error

Indicates whether a previous push to the FIFO caused an error

An error occurs due to the FIFO not having enough space to push all the bytes provided in the last write to the PFFCW_PAY register.

| PPE | Meaning |
| --- | --- |
| 0b0 | No error has occurred on the last push operation |
| 0b1 | An error has occurred on the last push operation |

**Bits [62:11]**

Reserved, RES0.

**FFS, bits [10:0]**

FIFO Free Space

| FFS | Meaning |
| --- | --- |
| 0b00000000000 | No free bytes in the FIFO |
| 0b00000000001..0 ↪b01111111111 | Number of free bytes in the FIFO |
| 0b10000000000 | 1024 bytes free in the FIFO |

The maximum value returned is never greater than the FIFO Depth

**Accessing PFFCW<n>_PAY64**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
return PFFCW<n>_PAY64;
```

**Write at offset 0x000 + (64 * n) from MHUS.PBX.PFFCW_page**

```
PFFCW<n>_PAY64 = value;
```

### C2.1.1.3.5 PFFCW&lt;n&gt;_FLG, Postbox FIFO Channel Window &lt;n&gt; Flag, n = 0 - 63

The PFFCW&lt;n&gt;_FLG characteristics are:

**Purpose**

Provides access to the flags to be used for the next group of bytes pushed onto the FIFO by a write to the PFFCW&lt;n&gt;_PAY register.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW&lt;n&gt;_FLG are RAZ/WI.

**Attributes**

PFFCW&lt;n&gt;_FLG is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW&lt;n&gt;_FLG bit assignments are:
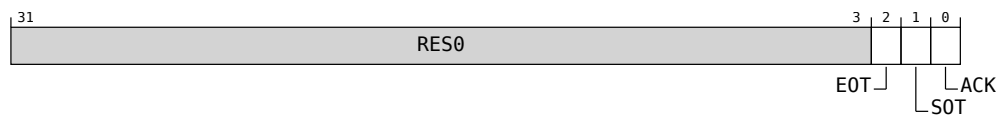


**Bits [31:3]**

Reserved, RES0.

**EOT, bit [2]**

EOT flag

The EOT field indicates that the next push operation to the FIFO will contain the last byte of a Transfer

This fields behavior depends on the value of the PFFCW&lt;n&gt;_CTRL.TDM field as follows:

| PFFCW&lt;n&gt;_CTRL.TDM | Behavior of the EOT field |
|---|---|
| 0b00 | Software manages the EOT field directly and hardware will never change the value. |
| 0b01 | The EOT field is set by software and hardware. The EOT is set to 0b0 when PFFCW&lt;n&gt;_CTRL.TDM is set to 0b01. When one or more bytes are pushed onto the FIFO, the EOT field is set to 0b0 irrespective of the value of the SOT or EOT fields before the push. |
| 0b10 | The EOT field is managed by hardware. The EOT flag is set to 0b0 when PFFCW&lt;n&gt;_CTRL.TDM is set to 0b10. The EOT flag value will toggle when one or more bytes are pushed onto the FIFO. |

| EOT | Meaning |
|---|---|
| 0b0 | Next push operation does not contain the last byte of the Transfer |
| 0b1 | Next push operation does contain the last byte of the Transfer |

The EOT flag is always associated with the last byte to be pushed onto the FIFO.

When multiple bytes are pushed onto the FIFO via a single write to the PFFCW<n>_PAY register, which byte is associated with which fields depends on the value of the PFFCW<n>_CTRL.MSBF field as follows:

- 0b0 - LSB is associated with the SOT flag and MSB is associated with the EOT and ACK flags.
- 0b1 - MSB is associated with the SOT flag and LSB is associated with the EOT and ACK flags.

The MHU is a little endian device and considers the LSB to be the byte which is written in the lowest offset accessed by the write.

The reset behavior of this field is:

- On a MHUS reset, this field resets to `0b0`.

Accessing this field has the following behavior:

- When PFFCW<n>_CTRL.TDM == '00', access to this field is **RW**
- Access is **RW** if all of the following are true:
    - PFFCW<n>_CTRL.TDM == '01'
    - any of the following are true:
        * EOT field will be set to == '1'
        * SOT field will be set to == '1'
- Access is **RO** if all of the following are true:
    - PFFCW<n>_CTRL.TDM == '01'
    - EOT field will be set to == '0'
    - SOT field will be set to == '0'
- When PFFCW<n>_CTRL.TDM == '10', access to this field is **RO**

**SOT, bit [1]**

SOT flag

The SOT flag indicates that the next push operation to the FIFO will contain the first byte of a Transfer

This fields behavior depends on the value of the PFFCW<n>_CTRL.TDM field as follows:

| PFFCW<n>_CTRL.TDM | Behavior of the SOT field |
| --- | --- |
| 0b00 | Software manages the SOT field directly and hardware will never change the value. |
| 0b01 | The SOT field is set by software and hardware. The SOT is set to 0b1 when PFFCW<n>_CTRL.TDM is set to 0b01. When one or more bytes are pushed onto the FIFO, the SOT field value will be set to the value of the EOT field when the push occurred. |
| 0b10 | The SOT field is managed by hardware. The SOT flag is set to 0b1 when PFFCW<n>_CTRL.TDM is set to 0b10 and is read-only. The SOT flag value will toggle when one or more bytes are push ed onto the FIFO. |

| SOT | Meaning |
| --- | --- |
| 0b0 | Next push operation does not contain the first byte of the Transfer |
| 0b1 | Next push operation does contain the first byte of the Transfer |

The SOT flag is always associated with the first byte to be pushed onto the FIFO.

When multiple bytes are pushed onto the FIFO via a single write to the PFFCW<n>_PAY register, which byte is associated with which fields depends on the value of the PFFCW<n>_CTRL.MSBF field as follows:

- 0b0 - LSB is associated with the SOT flag and MSB is associated with the EOT and ACK flags.
- 0b1 - MSB is associated with the SOT flag and LSB is associated with the EOT and ACK flags.

The MHU is a little endian device and considers the LSB to be the byte which is written in the lowest offset accessed by the write.

The reset behavior of this field is:

- On a MHUS reset, this field resets to `0b1`.

Accessing this field has the following behavior:

- When PFFCW<n>_CTRL.TDM == '00', access to this field is **RW**
- Access is **RW** if all of the following are true:
    - PFFCW<n>_CTRL.TDM == '01'
    - any of the following are true:
        * EOT field will be set to == '1'
        * SOT field will be set to == '1'
- Access is **RO** if all of the following are true:
    - PFFCW<n>_CTRL.TDM == '01'
    - EOT field will be set to == '0'
    - SOT field will be set to == '0'
- When PFFCW<n>_CTRL.TDM == '10', access to this field is **RO**

**ACK, bit [0]**

ACK Flag

The ACK flag requests that when the Receiver pops the byte and the byte is the last byte of the Transfer, from the FIFO, a Transfer Acknowledge event is generated.

The behavior of the ACK field is not effected by the value of PFFCW<n>_CTRL.TDM

| ACK | Meaning |
|-----|---------|
| `0b0` | Entry is not requested to generate a Transfer Acknowledge event when popped from the FIFO |
| `0b1` | Entry is requested to generated a Transfer Acknowledge event when popped from the FIFO |

The ACK flag is always associated with the same byte that is associated with the EOT flag.

The reset behavior of this field is:

- On a MHUS reset, this field resets to `0b0`.

**Accessing PFFCW<n>_FLG**

Accesses to this register use the following encodings:

**Accessible at offset 0x0008 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RW**.

### C2.1.1.3.6   PFFCW<n>_INT_ST, Postbox FIFO Channel Window <n> Interrupt Status, n = 0 - 63

The PFFCW<n>_INT_ST characteristics are:

**Purpose**

> Provides the status of interrupts for FFCH<n>.

**Configuration**

> This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_INT_ST are RAZ/WI.

**Attributes**

> PFFCW<n>_INT_ST is a 32-bit register.

> This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_INT_ST bit assignments are:
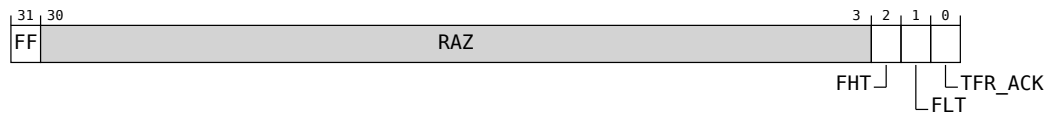


**FF, bit [31]**

FIFO Flush

Indicates whether the MHUR FIFO Flush mechanism has caused a flush of the FIFO for this FFCH.

| FF | Meaning |
|---|---|
| 0b0 | MHUR FIFO Flush mechanism has not caused a flush of the FIFO |
| 0b1 | MHU FIFO Flush mechanism has caused a flush of the FIFO |

**Bits [30:3]**

Reserved, RAZ.

**FHT, bit [2]**

FIFO High Tidemark

Indicates whether the Sender FIFO High Tidemark event has occurred.

| FHT | Meaning |
|---|---|
| 0b0 | Sender FIFO High Tidemark event has not occurred |
| 0b1 | Sender FIFO High Tidemark event has occurred |

**FLT, bit [1]**

FIFO Low Tidemark

Indicates whether the Sender FIFO Low Tidemark event has occurred.

| FLT | Meaning |
|-----|---------|
| 0b0 | Sender FIFO Low Tidemark event has not occurred |
| 0b1 | Sender FIFO Low Tidemark event has occurred |

**TFR_ACK, bit [0]**

Transfer Acknowledge

Indicates whether a Transfer Acknowledge event has occurred.

| TFR_ACK | Meaning |
|---------|---------|
| 0b0 | Transfer Acknowledge event has not occurred |
| 0b1 | Transfer Acknowledge event has occurred |

**Accessing PFFCW<n>_INT_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0010 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RO**.

### C2.1.1.3.7 PFFCW<n>_INT_CLR, Postbox FIFO Channel Window <n> Interrupt Clear, n = 0 - 63

The PFFCW<n>_INT_CLR characteristics are:

**Purpose**

Allows clearing of any interrupts for FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_INT_CLR are RAZ/WI.

**Attributes**

PFFCW<n>_INT_CLR is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_INT_CLR bit assignments are:
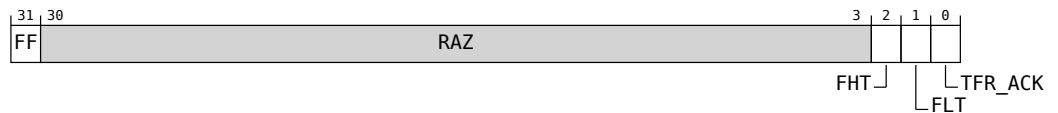


**FF, bit [31]**

FIFO Flush

| FF | Meaning |
|---|---|
| 0b0 | No effect |
| 0b1 | Clear FIFO flush event in the PFFCW<n>_INT_ST register |

**Bits [30:3]**

Reserved, RAZ.

**FHT, bit [2]**

FIFO High Tidemark

| FHT | Meaning |
|---|---|
| 0b0 | No effect |
| 0b1 | Clear FIFO High Tidemark event in the PFFCW<n>_INT_ST register |

**FLT, bit [1]**

FIFO Low Tidemark

| FLT | Meaning |
| --- | --- |
| 0b0 | No effect |
| 0b1 | Clear FIFO Low Tidemark event in the PFFCW<n>_INT_ST register |

**TFR_ACK, bit [0]**

Transfer Acknowledge

| TFR_ACK | Meaning |
| --- | --- |
| 0b0 | No effect |
| 0b1 | Clear Transfer Acknowledge event in the PFFCW<n>_INT_ST register |

**Accessing PFFCW<n>_INT_CLR**

Accesses to this register use the following encodings:

**Accessible at offset 0x0014 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **WO/RAZ**.

### C2.1.1.3.8 PFFCW<n>_INT_EN, Postbox FIFO Channel Window <n> Interrupt Enable, n = 0 - 63

The PFFCW<n>_INT_EN characteristics are:

**Purpose**

Configures the interrupts of FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_INT_EN are RAZ/WI.

**Attributes**

PFFCW<n>_INT_EN is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_INT_EN bit assignments are:
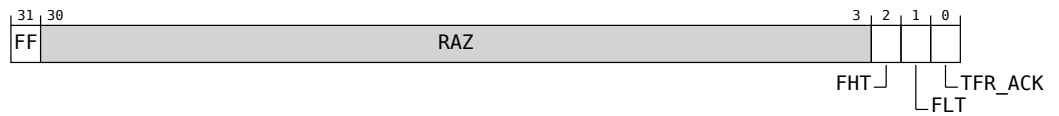


Shows the status of the events of the DBCH

**FF, bit [31]**

FIFO Flush

Controls whether a FIFO flush event generated by the MHUR generates an interrupt.

| FF | Meaning |
| --- | --- |
| 0b0 | A FIFO flush event from the MHUR does not generate an interrupt |
| 0b1 | A FIFO flush event from the MHUR generates an interrupt |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b1.

**Bits [30:3]**

Reserved, RAZ.

**FHT, bit [2]**

FIFO High Tidemark

Controls whether a Sender FIFO High Tidemark event generates an interrupt.

| FHT | Meaning |
| --- | --- |
| 0b0 | FIFO High Tidemark event does not generate an interrupt |

| FHT | Meaning |
|-----|---------|
| 0b1 | FIFO High Tidemark event generates an interrupt |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**FLT, bit [1]**

FIFO Low Tidemark

Controls whether a Sender FIFO Low Tidemark event generates an interrupt.

| FLT | Meaning |
|-----|---------|
| 0b0 | FIFO Low Tidemark event has does not generate an interrupt |
| 0b1 | FIFO Low Tidemark event generates an interrupt |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**TFR_ACK, bit [0]**

Transfer Acknolwedge

Controls whether a Transfer Acknowledge event generates an interrupt.

| TFR_ACK | Meaning |
|---------|---------|
| 0b0 | Transfer Acknowledge events does not generate an interrupt |
| 0b1 | Transfer Acknowledge event generates an interrupt |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**Accessing PFFCW<n>_INT_EN**

Accesses to this register use the following encodings:

**Accessible at offset 0x0018 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RW**.

### C2.1.1.3.9 PFFCW<n>_CTRL, Postbox FIFO Channel Window <n> Control, n = 0 - 63

The PFFCW<n>_CTRL characteristics are:

**Purpose**

> Configures the behavior of FFCH<n>.

**Configuration**

> This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_CTRL are RAZ/WI.

**Attributes**

> PFFCW<n>_CTRL is a 32-bit register.

> This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_CTRL bit assignments are:
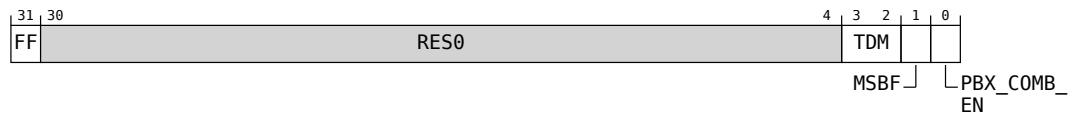


**FF, bit [31]**

FIFO Flush

Request a flush of the FFCH.

| FF | Meaning |
|----|---------|
| 0b0 | No request to flush the FIFO |
| 0b1 | Request to flush the FIFO |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**Bits [30:4]**

Reserved, RES0.

**TDM, bits [3:2]**

Transfer Delineation Mode

Selects which Transfer Delineation mode the MHU uses for the FFCH.

Transfer Delineation mode selects whether the MHU or software or a combination of both manages the SOT and EOT flags.

| TDM | Meaning |
|-----|---------|
| 0b00 | Software flag<br>Software is responsible for managing the SOT and EOT flags. |

| TDM | Meaning |
|---|---|
| 0b01 | Partial flag<br>Flags are managed partially by software and partially by the MHU.<br>Upon selecting partial flag Transfer Delineation mode the PFFCW<n>_FLG.{SOT,EOT} fields are set to 0b1 and 0b0 respectively.<br>No change occurs to the PFFCW<n>_FLG.ACK field.<br>To update either the PFFCW<n>_FLG.{SOT/EOT} fields at least one of them must be set to 0b1, otherwise the update to those fields are ignored.<br>When a push operation occurs the SOT and EOT fields are updated as follows:<br>  • SOT field is set to value of the EOT field before the push.<br>  • EOT field is always set to 0b0.<br>To update only the value of the PFFCW<n>_FLG.ACK field when using partial flag Transfer Delineation mode, software should set both the SOT and EOT fields to 0b0. |
| 0b10 | Auto flag<br>Flags are fully managed by the MHU.<br>Upon selecting auto flag Transfer Delineation mode the PFFCW<n>_FLG.{SOT,EOT} fields are set to 0b1 and 0b0 respectively.<br>No change occurs to the PFFCW<n>_FLG.ACK field.<br>When a push operation occurs the values of the SOT and EOT fields toggle.<br>The PFFCW<n>_FLG.{SOT/EOT} fields are read-only, but the PFFCW<n>_FLG.ACK field remains writeable.<br>To update the value of the PFFCW<n>_FLG.ACK field when using the auto flag Transfer Delineation mode software can write any value ot the SOT and EOT fields as it will be ignored. |

All other values are Reserved.

The reset behavior of this field is:

   • On a MHUS reset, this field resets to 0b00.

**MSBF, bit [1]**

Most Significant Byte First

Selects the order in which bytes are pushed onto the FIFO when multiple bytes are to be pushed due to a single write to the PFFCW<n>_PAY register.

| MSBF | Meaning |
| --- | --- |
| 0b0 | Least Significant Byte first |
| 0b1 | Most Significant Byte first |

FFCH are considered little-endian and the LSB is the lowest byte offset and the MSB is the highest byte offset within the access.

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**PBX_COMB_EN, bit [0]**

Postbox Combined Enable

Controls whether interrupts from the FFCH contribute to the Postbox Combined interrupt.

| PBX_COMB_EN | Meaning |
| --- | --- |
| 0b0 | FFCH interrupts do not contribute to the Postbox Combined interrupt. |
| 0b1 | FFCH interrupts contribute to the Postbox Combined interrupt. |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b1.

**Accessing PFFCW<n>_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x0020 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RW**.

### C2.1.1.3.10    PFFCW<n>_ST, Postbox FIFO Channel Window <n> Status, n = 0 - 63

The PFFCW<n>_ST characteristics are:

**Purpose**

Provides the status of FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_ST are RAZ/WI.

**Attributes**

PFFCW<n>_ST is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_ST bit assignments are:
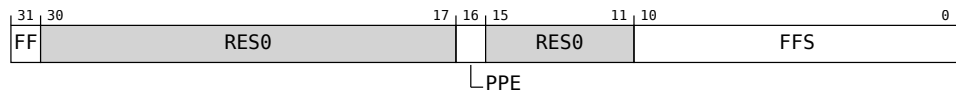


**FF, bit [31]**

FIFO Flush

Status of MHUS FIFO flush mechanism for the FFCH.

| FF | Meaning |
|---|---|
| 0b0 | The meaning of this value depends on the value of the PFFCW<n>_CTRL.FF field. **When PFFCW<n>_CTRL.FF is 0b0** Sender FIFO flush mechanism is idle **When PFFCW<n>_CTRL.FF is 0b1** Sender FIFO flush mechanism is performing a flush |
| 0b1 | Sender FIFO flush completed |

**Bits [30:17]**

Reserved, RES0.

**PPE, bit [16]**

Previous Push Error

Indicates whether a previous push to the FIFO caused an error

An error is when the previous write to the PFFCW<n>_PAY register could not push all bytes onto the FIFO due to the FIFO not having enough invalid bytes.

| PPE | Meaning |
|---|---|
| 0b0 | No error has occurred on the last push operation |

| PPE | Meaning |
| --- | --- |
| 0b1 | An error has occurred on the last push operation |

**Bits [15:11]**

Reserved, RES0.

**FFS, bits [10:0]**

FIFO Free Space

Indicates the number of invalid bytes in the FIFO

The maximum value returned is never greater than the FIFO Depth

**Accessing PFFCW<n>_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0024 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RO**.

### C2.1.1.3.11 PFFCW<n>_ACK_CNT, Postbox FIFO Channel Window <n> Acknowledge Counter, n = 0 - 63

The PFFCW<n>_ACK_CNT characteristics are:

**Purpose**

Provides a count of the number of Transfers which the Receiver has acknowledged, for FFCH<n>, since the last time the register was read.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_ACK_CNT are RAZ/WI.

**Attributes**

PFFCW<n>_ACK_CNT is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_ACK_CNT bit assignments are:
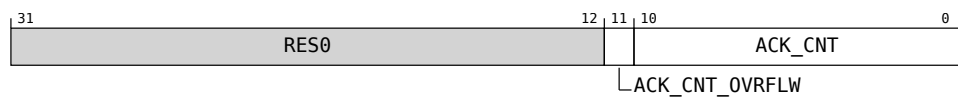


**Bits [31:12]**

Reserved, RES0.

**ACK_CNT_OVRFLW, bit [11]**

Acknowledge Counter Overflow

Indicate whether the Acknowledge counter has overflown.

| ACK_CNT_OVRFLW | Meaning |
|---|---|
| 0b0 | Acknowledge counter has not overflown |
| 0b1 | Acknowledge counter has overflown |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**ACK_CNT, bits [10:0]**

Acknowledge Count

Count of the number of Transfer Acknowledge events which have occurred since the last time the register was read.

When value of ACK_CNT_OVRFLW is set to 1 the value in this field no longer provides an accurate count of the number of Transfer Acknowledge events.

| ACK_CNT | Meaning |
|---|---|
| 0b00000000000..0 ↪b11111111111 | Number of Acknowledged Transfers |

The maximum value of this field is calculated by $2^{clog2((FFCH\_DEPTH/min\_transfer\_size)+1)} - 1$.

Where min_transfer_size is minimum size of a Transfer in bytes which the Sender can send.

The value of min_transfer_size is determined from the values of the P{8/16/32/64}BA_SPT fields in the PBX_FFCH_CFG0 register.

| P8BA_SPT | P16BA_SPT | P32BA_SPT | P64BA_SPT | Minimum Transfer Size (Bytes) |
|---|---|---|---|---|
| 1 | X | X | X | 1 |
| 0 | 1 | X | X | 2 |
| 0 | 0 | 1 | X | 4 |
| 0 | 0 | 0 | 1 | 8 |

When the counter reaches maximum value and a new Acknowledge occurs the ACK_CNT_OVRFLW is set to 0b1 and the ACK_CNT field wraps around.

The reset behavior of this field is:

- On a MHUS reset, this field resets to `0b00000000000`.

**Accessing PFFCW<n>_ACK_CNT**

Accesses to this register use the following encodings:

**Accessible at offset 0x0028 + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RO**.

### C2.1.1.3.12 PFFCW<n>_TIDE, Postbox FIFO Channel Window <n> Tidemark, n = 0 - 63

The PFFCW<n>_TIDE characteristics are:

**Purpose**

Allows configuration of the low and high tidemark thresholds for the Sender tidemark events for FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to PFFCW<n>_TIDE are RAZ/WI.

**Attributes**

PFFCW<n>_TIDE is a 32-bit register.

This register is part of the MHUS.PBX.PFFCW_page block.

**Field descriptions**

The PFFCW<n>_TIDE bit assignments are:

| 31      26 | 25            16 | 15      10 | 9            0 |
|------------|------------------|------------|----------------|
| RES0       | HIGH             | RES0       | LOW            |

**Bits [31:26]**

Reserved, RES0.

**HIGH, bits [25:16]**

High Tide Mark

Threshold value used in the generation of the Sender FIFO High Tide event.

The event is generated when a push to the FIFO occurs, and the following are both true:

- The FIFO fill level before the push was less than or equal to the value of this field.

- The FIFO fill level after the push is greater than the value of this field.

| HIGH | Meaning |
|------|---------|
| 0b0000000000..0 ↪b1111111111 | Threshold value in bytes |

The upper and lower offset of this field depend on the configuration of the MHU.

The upper offset of this field is equal to clog2(FFCH_DEPTH)+15.

The lower offset of this field depends on the supported access sizes to PFFCW<n>_PAY register as follows:

- When PBX_FFCH_CFG.P8BA_SPT is 0b1 the lower offset is 16.

- When PBX_FFCH_CFG.P8BA_SPT is 0b0 and PBX_FFCH_CFG.P16BA_SPT is 0b1 the lower offset is 17.

- When PBX_FFCH_CFG.P{8/16}BA_SPT are both 0b0 and PBX_FFCH_CFG.P32BA_SPT is 0b1 the lower offset is 18.

- When PBX_FFCH_CFG.P{8/16/32}BA_SPT are all 0b0 and PBX_FFCH_CFG.P64BA_SPT is 0b1 the lower offset is 19.

Any offsets above the upper offset or below the lower offset are RES0.

If the upper offset is less than the lower offset the entire field is RES0.

For calculations of the Sender High Tide event all offsets which are RES0 are used.

The reset behavior of this field is:

- On a MHUS reset, the expression `(FFCH_DEPTH - 1)[u:l]`.

Where:

u = tide_upr()

l = tide_lwr([P64BA_SPT,P32BA_SPT,P16BA_SPT,P8BA_SPT])

**Bits [15:10]**

Reserved, RES0.

**LOW, bits [9:0]**

Low Tide Mark

Threshold value used in the generation of the Sender FIFO Low Tide event.

The event is generated when a pop to the FIFO occurs, and the following are both true:

- The FIFO fill level before the pop was greater than the value of this field.

- The FIFO fill level after the pop is less than or equal the value of this field.

| LOW | Meaning |
|---|---|
| 0b0000000000..0 ↪b1111111111 | Threshold value in bytes |

The upper and lower offset of this field depend on the configuration of the MHU.

The upper offset of this field is equal to clog2(FFCH_DEPTH)-1.

The lower offset of this field depends on the supported access sizes to PFFCW<n>_PAY register as follows:

- When PBX_FFCH_CFG.P8BA_SPT is 0b1 the lower offset is 0.

- When PBX_FFCH_CFG.P8BA_SPT is 0b0 and PBX_FFCH_CFG.P16BA_SPT is 0b1 the lower offset is 1.

- When PBX_FFCH_CFG.P{8/16}BA_SPT are both 0b0 and PBX_FFCH_CFG.P32BA_SPT is 0b1 the lower offset is 2.

- When PBX_FFCH_CFG.P{8/16/32}BA_SPT are all 0b0 and PBX_FFCH_CFG.P64BA_SPT is 0b1 the lower offset is 3.

Any offsets above the upper offset or below the lower offset are RES0.

If the upper offset is less than the lower offset the entire field is RES0.

For calculations of the Sender High Tide event all offsets which are RES0 are used.

The reset behavior of this field is:

- On a MHUS reset, this field resets to `0b0000000000`.

**Accessing PFFCW<n>_TIDE**

Accesses to this register use the following encodings:

**Accessible at offset 0x002C + (64 * n) from MHUS.PBX.PFFCW_page**

Access on this interface is **RW**.

### C2.1.1.4 PFCW_page, Postbox Fast Channel Windows Page

The PFCW_page characteristics are:

**Purpose**

Allows access to the PFCW.

Depending on the Fast Channel word-size either the PFCW<n>_PAY32 or PFCW<n>_PAY64 registers are implmented.

**When the Fast Channel word-size is 32-bit**

PFCW<n>_PAY32 is implemented

There can be between 1 and 1024 PFCWs.

**When the Fast Channel word-size is 64-bit**

PFCW<n>_PAY64 is implemented

There can be between 1 and 512 PFCWs.

**Configuration**

This Register Block is present only when FCE is implemented.

**Attributes**

The PFCW_page block is of size 4KB.

This block is part of the MHUS.PBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| `0x000 + (4 * n)`for n in ↪1023:0 | PFCW<n>_PAY32 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x20<br>Accessor condition: When FCH_WS == 0x20 |
| `0x000 + (4 * n)`for n in ↪1023:0 | PFCW<n>_PAY32 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x20<br>Accessor condition: When FCH_WS == 0x20 |
| `0x000 + (8 * n)`for n in ↪511:0 | PFCW<n>_PAY64 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x40<br>Accessor condition: When FCH_WS == 0x40 |
| `0x000 + (8 * n)`for n in ↪511:0 | PFCW<n>_PAY64 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x40<br>Accessor condition: When FCH_WS == 0x40 |

### C2.1.1.4.1 PFCW<n>_PAY32, Postbox Fast Channel Window <n> Payload 32-bit, n = 0 - 1023

The PFCW<n>_PAY32 characteristics are:

**Purpose**

Access to payload of FCH <n>

Arm recommends that accesses to these registers are atomic.

---

**Note**

This is the 32-bit version of the PFCW<n>_PAY registers

---

**Configuration**

This register is present only when FCE is implemented and FCH_WS == 0x20. Otherwise, direct accesses to PFCW<n>_PAY32 are RAZ/WI.

**Attributes**

PFCW<n>_PAY32 is a 32-bit register.

This register is part of the MHUS.PBX.PFCW_page block.

**Field descriptions**

The PFCW<n>_PAY32 bit assignments are:

| 31 0 |
|---|
| PAY |

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

Payload for Channel <n>

A write to this register sets the value of the payload and generates a Transfer event.

A read to this register returns the current value of the payload.

The reset behavior of this field is:

• On a MHUR reset, this field resets to an architecturally UNKNOWN value.

**Accessing PFCW<n>_PAY32**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (4 * n) from MHUS.PBX.PFCW_page**

```
return PFCW<n>_PAY32;
```

**Write at offset 0x000 + (4 * n) from MHUS.PBX.PFCW_page**

```
PFCW<n>_PAY32 = value;
```

### C2.1.1.4.2  PFCW<n>_PAY64, Postbox Fast Channel Window <n> Payload 64-bit, n = 0 - 511

The PFCW<n>_PAY64 characteristics are:

**Purpose**

> Access to payload of FCH <n>
>
> Arm recommends that accesses to these registers are atomic.

> ---
> **Note**
>
> This is the 64-bit version of the PFCW<n>_PAY registers
> ---

**Configuration**

> This register is present only when FCE is implemented and FCH_WS == 0x40. Otherwise, direct accesses to PFCW<n>_PAY64 are RAZ/WI.

**Attributes**

> PFCW<n>_PAY64 is a 64-bit register.
>
> This register is part of the MHUS.PBX.PFCW_page block.

**Field descriptions**

The PFCW<n>_PAY64 bit assignments are:
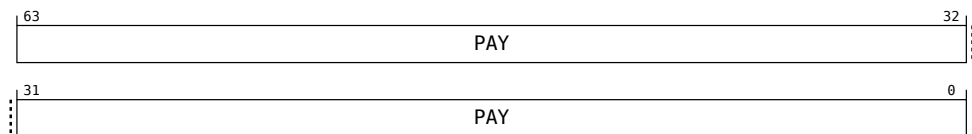


**PAY, bits [63:0]**

Payload for Channel <n>

A write to this register sets the value of the payload and generates a Transfer event.

A read to this register returns the current value of the payload.

The reset behavior of this field is:

 • On a MHUR reset, this field resets to an architecturally UNKNOWN value.

**Accessing PFCW<n>_PAY64**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (8 * n) from MHUS.PBX.PFCW_page**

```
return PFCW<n>_PAY64;
```

**Write at offset 0x000 + (8 * n) from MHUS.PBX.PFCW_page**

```
PFCW<n>_PAY64 = value;
```

### C2.1.1.5 PBX_IMPL_DEF_page, Postbox Implementation Defined page

The PBX_IMPL_DEF_page characteristics are:

**Purpose**

The PBX_IMPL_DEF_page is for an implementation of the MHU, to implement any IMPLEMENTATION DEFINED registers.

Registers defined in this block must follow the rules in  C1.7.3 *IMPLEMENTATION DEFINED Page*

**Configuration**

**Attributes**

The PBX_IMPL_DEF_page block is of size 4KB.

This block is part of the MHUS.PBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x0000 + (4 * n)`for n in 1023:0 | - | Most permissive access: ImplementationDefined |

## C2.1.2 SSC, Sender Security Control

The SSC characteristics are:

**Purpose**

Contains the individual pages of the Sender Security Control block.

Only accesses with the correct security can access the registers within the Sender Security Control block, otherwise the access is treated as an illegal access.

The allowed security of an access to a register of the Sender Security Control block depends on:

- Whether RME is implemented for the MHUS.
- Sampled value of MHUS **LEGACY_TZ_EN** input.

The following table list the allowed security states of an access:

| RME | TZE | **LEGACY_TZ_EN** | Allowed access security |
|-----|-----|------------------|-------------------------|
| 0 | 0 | NA | NA |
| 0 | 1 | NA | Secure |
| 1 | 1 | 0 | Root |
|   |   | 1 | Secure |

**Configuration**

This Register Block is present only when TZE is implemented for the MHUS.

**Attributes**

The SSC block is of size 64KB.

This block is part of the MHUS block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| `0x0000` | SSC_CTRL_page | Most permissive access: RW |
| `0xF000` | SSC_IMPL_DEF_page | Most permissive access: RW |

### C2.1.2.1   SSC_CTRL_page, Sender Security Control Page

The SSC_CTRL_page characteristics are:

**Purpose**

Allow assignment of resources of the MHUS to a Security Group.

**Configuration**

This Register Block is present only when TZE is implemented for the MHUS.

**Attributes**

The SSC_CTRL_page block is of size 4KB.

This block is part of the MHUS.SSC block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x000` | SSC_BLK_ID | Most permissive access: RO |
| `0x010` | SSC_FEAT_SPT0 | Most permissive access: RO |
| `0x014` | SSC_FEAT_SPT1 | Most permissive access: RO |
| `0x020` | SSC_DBCH_CFG0 | Most permissive access: RO<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |
| `0x030` | SSC_FFCH_CFG0 | Most permissive access: RO<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0x040` | SSC_FCH_CFG0 | Most permissive access: RO<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| `0x110 + (4 * n)`for n in 0 | SSC_PBX_SG<n> | Most permissive access: RW |
| `0xFC8` | SSC_IIDR | Most permissive access: RO |
| `0xFCC` | SSC_AIDR | Most permissive access: RO |
| `0xFD0 + (4 * n)`for n in 11:0 | SSC_IMPL_DEF_ID<n> | Most permissive access: RO |

### C2.1.2.1.1    SSC_BLK_ID, Sender Security Block Identifier

The SSC_BLK_ID characteristics are:

**Purpose**

> Identifies the block as a Sender Security.

**Configuration**

> This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_BLK_ID are RAZ/WI.

**Attributes**

> SSC_BLK_ID is a 32-bit register.

> This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_BLK_ID bit assignments are:
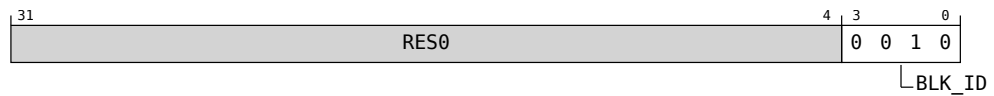


**Bits [31:4]**

Reserved, RES0.

**BLK_ID, bits [3:0]**

Block Identifier

Identifies the type of block that resides in this 64KB.

Reads as 0b0010

Identifies the block as the Sender Security block.

Access to this field is **RO**.

**Accessing SSC_BLK_ID**

Accesses to this register use the following encodings:

**Accessible at offset 0x000 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.2   SSC_FEAT_SPT0, Sender Security Feature Support 0

The SSC_FEAT_SPT0 characteristics are:

**Purpose**

Provides information on the features implemented in the Sender Security.

**Configuration**

This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_FEAT_SPT0 are RAZ/WI.

**Attributes**

SSC_FEAT_SPT0 is a 32-bit register.

This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_FEAT_SPT0 bit assignments are:

| 31　　　　　　　　　　24 | 23　　20 | 19　　16 | 15　　12 | 11　　　8 | 7　　　4 | 3　　　0 |
|---|---|---|---|---|---|---|
| RES0 | RASE_SPT | RME_SPT | TZE_SPT | FCE_SPT | FE_SPT | DBE_SPT |

**Bits [31:24]**

Reserved, RES0.

**RASE_SPT, bits [23:20]**

Reliability, Availability and Serviceability Extension Support

The value of this field depends on the implementation of the MHU.

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

| RASE_SPT | Meaning |
|---|---|
| 0b0000 | MHU does not implement the RAS extension |
| 0b0010 | MHU implements the RAS extension but does not follow the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |
| 0b0011 | MHU implements the RAS extension and follows the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |

Access to this field is **RO**.

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

Realm Management Extension Support

The value of this field depends on the implementation of the MHU and an optional reset time sampled input **LEGACY_TZ_EN**.

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

| RME_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Realm Management extension |
| 0b0001 | MHU implements Realm Management extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

When RME is implemented, for the half of the MHU, there can be a **LEGACY_TZ_EN** tie-off present on that side of the MHU.

The value of the **LEGACY_TZ_EN** tie-off is sampled at reset of the side of the MHU which the tie-off is associated with.

When the sampled value of the tie-off is 0b1 the value of this field is always 0x0, otherwise the value of this field is dependent on whether RME is implemented for this half of the MHU.

Access to this field is **RO**.

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

TrustZone Extension Support

The value of this field depends on the implementation of the MHU.

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

| TZE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the TrustZone extension |
| 0b0001 | MHU implements TrustZone extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

Access to this field is **RO**.

**FCE_SPT, bits [11:8]**

Fast Channel Extension Support

The value of this field depends on the implementation of the MHU.

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

| FCE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Fast Channel extension |

| FCE_SPT | Meaning |
|---------|---------|
| 0b0001  | MHU implements Fast Channel extension |

Access to this field is **RO**.

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

FIFO Extension Support

The value of this field depends on the implementation of the MHU.

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

| FE_SPT | Meaning |
|--------|---------|
| 0b0000 | MHU does not implement the FIFO extension |
| 0b0001 | MHU implements FIFO extension |

Access to this field is **RO**.

**DBE_SPT, bits [3:0]**

Doorbell Extension Support

The value of this field depends on the implementation of the MHU.

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

| DBE_SPT | Meaning |
|---------|---------|
| 0b0000  | MHU does not implement the Doorbell extension |
| 0b0001  | MHU implements Doorbell extension |

Access to this field is **RO**.

**Accessing SSC_FEAT_SPT0**

Accesses to this register use the following encodings:

**Accessible at offset 0x010 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.3 SSC_FEAT_SPT1, Sender Security Feature Support 1

The SSC_FEAT_SPT1 characteristics are:

**Purpose**

Provides information on the features implemented in the Sender Security.

**Configuration**

This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_FEAT_SPT1 are RAZ/WI.

**Attributes**

SSC_FEAT_SPT1 is a 32-bit register.

This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_FEAT_SPT1 bit assignments are:
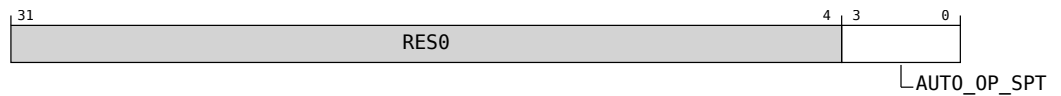


**Bits [31:4]**

Reserved, RES0.

**AUTO_OP_SPT, bits [3:0]**

Auto Op Protocol Support

The value of this field depends on the implementation of the MHU.

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

| AUTO_OP_SPT | Meaning |
| --- | --- |
| 0b0000 | Auto Op(Min) is implemented |
| 0b0001 | Auto Op(Full) is implemented |

Access to this field is **RO**.

**Accessing SSC_FEAT_SPT1**

Accesses to this register use the following encodings:

**Accessible at offset 0x014 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.4   SSC_DBCH_CFG0, Sender Security Doorbell Channel Configuration 0

The SSC_DBCH_CFG0 characteristics are:

**Purpose**

> Provides information on the configuration of DBE in MHUS.

**Configuration**

> This register is present only when TZE is implemented for the MHUS and DBE is implemented. Otherwise, direct accesses to SSC_DBCH_CFG0 are RAZ/WI.

**Attributes**

> SSC_DBCH_CFG0 is a 32-bit register.

> This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_DBCH_CFG0 bit assignments are:
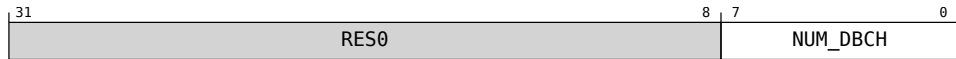
| 31                                    8 | 7          0 |
|------------------------------------------|--------------|
| RES0                                     | NUM_DBCH     |

**Bits [31:8]**

Reserved, RES0.

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

Number of Doorbell Channels

Number of DBCH implemented in the MHUS.

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

| NUM_DBCH | Meaning |
|----------|---------|
| 0x00..0x7F | Number of DBCH is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing SSC_DBCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x020 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.5   SSC_FFCH_CFG0, Sender Security FIFO Channel Configuration 0

The SSC_FFCH_CFG0 characteristics are:

**Purpose**

> Provides information on the configuration of FE in Sender Security.

**Configuration**

> This register is present only when TZE is implemented for the MHUS and FE is implemented. Otherwise, direct accesses to SSC_FFCH_CFG0 are RAZ/WI.

**Attributes**

> SSC_FFCH_CFG0 is a 32-bit register.

> This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_FFCH_CFG0 bit assignments are:



**Bits [31:26]**

Reserved, RES0.

**FFCH_DEPTH, bits [25:16]**

FIFO Channel Depth

Depth of the FIFOs of the FFCHs in the MHUS.

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

| FFCH_DEPTH | Meaning |
|---|---|
| 0b0000000000..0 ↪b1111111111 | FIFO depth is N+1 bytes, where N is the value of this field. |

Access to this field is **RO**.

**Bits [15:12]**

Reserved, RES0.

**P64BA_SPT, bit [11]**

Postbox 64-bit Access Support

Whether 64-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P64BA_SPT | Meaning |
|---|---|
| 0b0 | 64-bit accesses are not supported |
| 0b1 | 64-bit accesses are supported |

Accesses must be aligned to an 64-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

When 64-bit accesses are supported the PFFCW<n>_PAY register occupies offsets 0x00-0x07 of the PFFCW.

When 64-bit accesses are not supported the PFFCW<n>_PAY register occupies offsets 0x00-0x03 of the PFFCW and offsets 0x04-0x07 of the PFFCW are RES0.

Access to this field is **RO**.

**P32BA_SPT, bit [10]**

Postbox 32-bit Access Support

Whether 32-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P32BA_SPT | Meaning |
| --- | --- |
| 0b0 | 32-bit accesses are not supported |
| 0b1 | 32-bit accesses are supported |

Accesses must be aligned to an 32-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**P16BA_SPT, bit [9]**

Postbox 16-bit Access Support

Whether 16-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P16BA_SPT | Meaning |
| --- | --- |
| 0b0 | 16-bit accesses are not supported |
| 0b1 | 16-bit accesses are supported |

Accesses must be aligned to an 16-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**P8BA_SPT, bit [8]**

Postbox 8-bit Access Support

Whether 8-bit accesses to the PFFCW<n>_PAY register are supported.

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

| P8BA_SPT | Meaning |
| --- | --- |
| 0b0 | 8-bit accesses are not supported |

| P8BA_SPT | Meaning |
|----------|---------|
| 0b1 | 8-bit accesses are supported |

Accesses must be aligned to an 8-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

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

Number of FIFO Channels

The number of FFCHs in the MHUS.

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

| NUM_FFCH | Meaning |
|----------|---------|
| 0x00..0x3F | Number of FFCHs is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing SSC_FFCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x030 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.6 SSC_FCH_CFG0, Sender Security Fast Channel Configuration 0

The SSC_FCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FCE in the MHUS.

**Configuration**

This register is present only when TZE is implemented for the MHUS and FCE is implemented. Otherwise, direct accesses to SSC_FCH_CFG0 are RAZ/WI.
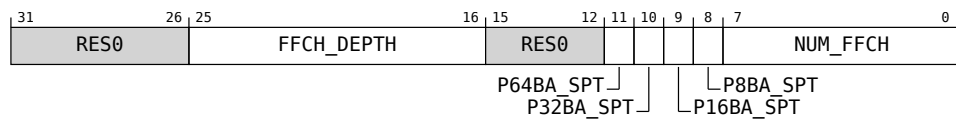
**Attributes**

SSC_FCH_CFG0 is a 32-bit register.

This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_FCH_CFG0 bit assignments are:

| 31 | 29 | 28 | | 21 | 20 | 16 | 15 | | 11 | 9 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RES0 | | FCH_WS | | | | | NUM_FCG | | | NUM_FCH | | |

└─NUM_FCH_PER_FCG

**Bits [31:29]**

Reserved, RES0.

**FCH_WS, bits [28:21]**

Fast Channel Word-Size

Number of bits each FCH implements.

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

| FCH_WS | Meaning |
|---|---|
| 0x20 | Fast Channel word-size is 32-bits |
| 0x40 | Fast Channel word-size is 64-bits |

Access to this field is **RO**.

**NUM_FCH_PER_FCG, bits [20:16]**

Number of Fast Channels per Fast Channel Group

Number of FCHs implemented in FCGs for MHUS.

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

| NUM_FCH_PER_FCG | Meaning |
|---|---|
| 0b00000..0b11111 | Number of FCHs per FCG is N+1, where N is the value of this field |

Access to this field is **RO**.

**NUM_FCG, bits [15:11]**

Number of Fast Channel Groups

Number of FCGs implemented in MHUS

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

| NUM_FCG | Meaning |
|---|---|
| 0b00000..0 ↪b11111 | Number of FCGs is N+1, where N is the value of this field |

Access to this field is **RO**.

**NUM_FCH, bits [9:0]**

Number of Fast Channels

Number of FCHs implemented in MHUS.

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

| NUM_FCH | Meaning | Applies |
|---|---|---|
| 0b0000000000..0 ↪b0111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x40 |
| 0b0000000000..0 ↪b1111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x20 |

Access to this field is **RO**.

**Accessing SSC_FCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x040 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.7 SSC_PBX_SG<n>, Sender Postbox Security Group <ngt;, n = 0 - 0

The SSC_PBX_SG<n> characteristics are:

**Purpose**

Configuration of the Security Group of the Postbox

**Configuration**

This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_PBX_SG<n> are RAZ/WI.
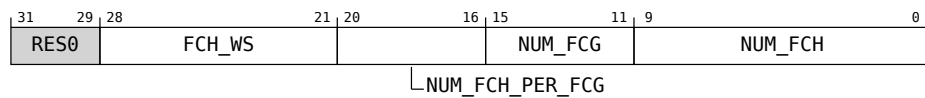
**Attributes**

SSC_PBX_SG<n> is a 32-bit register.

This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_PBX_SG<n> bit assignments are:

**When RME is implemented for the MHUS and sampled value of MHUS LEGACY_TZ_EN is 0b0:**
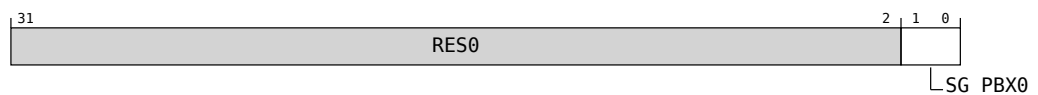


**Bits [31:2]**

Reserved, RES0.

**SG_PBX<m>, bits [2m+1:2m], for m = 0**

Security Group for Postbox

| SG_PBX<m> | Meaning |
|---|---|
| 0b00 | Postbox is assigned to the Secure Security Group |
| 0b01 | Postbox is assigned to the Non-secure Security Group |
| 0b10 | Postbox is assigned to the Root Security Group |
| 0b11 | Postbox is assigned to the Realm Security Group |

The reset behavior of this field is:

* On a MHUS reset, this field resets to 0b10.

**When RME is not implemented for the MHUS or (RME is implemented for the MHUS and sampled value of MHUS LEGACY_TZ_EN is 0b1):**
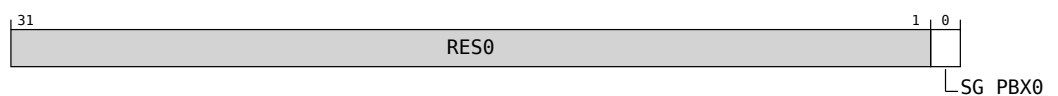


**Bits [31:1]**

Reserved, RES0.

**SG_PBX\<m\>, bits [m], for m = 0**

Security Group for Postbox

| SG_PBX\<m\> | Meaning |
| --- | --- |
| 0b0 | Postbox is assigned to the Secure Security Group |
| 0b1 | Postbox is assigned to the Non-secure Security Group |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b0.

**Accessing SSC_PBX_SG\<n\>**

Accesses to this register use the following encodings:

**Accessible at offset 0x110 + (4 * n) from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RW**.

### C2.1.2.1.8 SSC_IIDR, Sender Security Implementer Identification Register

The SSC_IIDR characteristics are:

**Purpose**

> This field provides information on the implementation of the MHU

**Configuration**

> This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_IIDR are RAZ/WI.

**Attributes**

> SSC_IIDR is a 32-bit register.

> This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_IIDR bit assignments are:

| 31              20 | 19      16 | 15      12 | 11                    0 |
|--------------------|------------|------------|-------------------------|
| PRODUCT_ID         | VARIANT    | REVISION   | IMPLEMENTER             |

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

Product ID of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Variant or major revision of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Revision or minor version of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**IMPLEMENTER, bits [11:0]**

Implementer ID

Contains the JEP106 identification information as follows:

- 11:8 - JEP106 continuation code of implementer
- 7 - Always 0
- 6:0 - JEP106 identity code of implementer

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**Accessing SSC_IIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFC8 from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.9 SSC_AIDR, Sender Security Architecture Identification Register

The SSC_AIDR characteristics are:

**Purpose**

Provides information on the version of the MHU architecture implemented in this implementation of the MHU.

**Configuration**

This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_AIDR are RAZ/WI.

**Attributes**

SSC_AIDR is a 32-bit register.

This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_AIDR bit assignments are:



**Bits [31:8]**

Reserved, RES0.

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

MHU Architecture Major Revision

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

| ARCH_MAJOR_REV | Meaning |
| --- | --- |
| 0b0000 | MHUv1 or unknown architecture versions |
| 0b0001 | MHUv2 |
| 0b0010 | MHUv3 |

All other values are Reserved

Access to this field is **RO**.

**ARCH_MINOR_REV, bits [3:0]**

MHU Architecture Minor Revision

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

| ARCH_MINOR_REV | Meaning |
| --- | --- |
| 0b0000 | Minor revision 0 of the major architecture |
| 0b0001 | Minor revision 1 of the major architecture |

All other values are Reserved

Access to this field is **RO**.

**Additional information**

The legal combinations for the ARCH_MAJOR_REV and ARCH_MINOR_REV fields are as follows:

| ARCH_MAJOR_REV | ARCH_MINOR_REV | Description |
| --- | --- | --- |
| 0x0 | 0x0 | MHUv1 |
| 0x1 | 0x0 | MHUv2.0 |
| 0x1 | 0x1 | MHUv2.1 |
| 0x2 | 0x0 | MHUv3.0 |

MHU implementations based on this architecture, have the ARCH_MAJOR_REV set to 0x2 and ARCH_MINOR_REV set to 0x0.

**Accessing SSC_AIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFCC from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.1.10   SSC_IMPL_DEF_ID<n>, IMPDEF Register, n = 0 - 11

The SSC_IMPL_DEF_ID<n> characteristics are:

**Purpose**

This register is for IMPLEMENTATION DEFINED Identification Registers which can be used to identify the implementation of the MHU Sender Security

An implementation can do the following with this register:

- Chose whether a register is present or not.

- The name of the register.

- The access permission of the registers, so long as they are not more permissive than the architecture defines for the IMPLEMENTATION DEFINED register.

- The names, behavior and reset value of any fields.

- The access permissions of any fields, so long as they are not more permissive than the register.

If the SSC_IMPL_DEF_ID<n> is not present then the location is Reserved and treated as RES0

**Configuration**

This register is present only when TZE is implemented for the MHUS. Otherwise, direct accesses to SSC_IMPL_DEF_ID<n> are RAZ/WI.

**Attributes**

SSC_IMPL_DEF_ID<n> is a 32-bit register.

This register is part of the MHUS.SSC.SSC_CTRL_page block.

**Field descriptions**

The SSC_IMPL_DEF_ID<n> bit assignments are:

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

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

IMPLEMENTATION DEFINED

**Accessing SSC_IMPL_DEF_ID<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0xFD0 + (4 * n) from MHUS.SSC.SSC_CTRL_page**

Access on this interface is **RO**.

### C2.1.2.2  SSC_IMPL_DEF_page, Sender Security Control Implementation Defined page

The SSC_IMPL_DEF_page characteristics are:

**Purpose**

> The SSC_IMPL_DEF_page is for an implementation of the MHU, to implement any IMPLEMENTATION DEFINED registers.
>
> Registers defined in this block must follow the rules in  C1.7.3 *IMPLEMENTATION DEFINED Page*

**Configuration**

> This Register Block is present only when TZE is implemented for the MHUS.

**Attributes**

> The SSC_IMPL_DEF_page block is of size 4KB.
>
> This block is part of the MHUS.SSC block.

**Default access**

> Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x0000 + (4 * n)`for n in 1023:0 | - | Most permissive access: ImplementationDefined |

## C2.2  MHUR, MHU Receiver

The MHUR characteristics are:

**Purpose**

Contains the blocks which are part of the MHU Receiver

The offsets of the blocks, within the MHU Receiver are IMPLEMENTATION DEFINED, however, Arm recommends the following offsets:

**When TZE is implemented for the MHUR**

0x0_0000 - RSC

0x1_0000 - MBX

**When TZE is not implemented for the MHUR**

0x0_0000 - MBX

**Attributes**

**When TZE is implemented for the MHUR**

The MHUR block is of size 128KB.

**When TZE is not implemented for the MHUR**

The MHUR block is of size 64KB.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| IMPDEF | RSC | Most permissive access: RW<br>Register condition: When TZE is implemented for the MHUR<br>Accessor condition: When TZE is implemented for the MHUR |
| IMPDEF | MBX | Most permissive access: RW |

## C2.2.1  MBX, Mailbox

The MBX characteristics are:

**Purpose**

Contains the individual pages of the Mailbox block.

Only accesses with the correct security can access the registers within the Mailbox, otherwise the access is treated as an illegal access

The allowed security of an access to a register of the Mailbox depends on:

- The value of RME is implemented for the MHUR.

- Whether TZE is implemented for the MHUR.

- Sampled value of MHUR **LEGACY_TZ_EN** input.

- Value of RSC_CTRL_page.RSC_MBX_SG0.SG_MBX0 field.

The following table list the allowed security states of an access:

| RME | TZE | Sampled **LEGACY_TZ_EN** | RSC_MBX_SG0.SG_MBX0 | Allowed access securities |
|-----|-----|--------------------------|---------------------|---------------------------|
| 0 | 0 | NA | NA | Any |
| 0 | 1 | NA | 0b0 | Secure |
|   |   |    | 0b1 | Any |
| 1 | 1 | 0 | 0b00 | Root and Secure |
|   |   |   | 0b01 | Any |
|   |   |   | 0b10 | Root |
|   |   |   | 0b11 | Root and Realm |
|   |   | 1 | 0bx0 | Root and Secure |
|   |   |   | 0bx1 | Any |

**Configuration**

**Attributes**

The MBX block is of size 64KB.

This block is part of the MHUR block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| `0x0000` | MBX_CTRL_page | Most permissive access: RW |
| `0x1000` | MDBCW_page | Most permissive access: RW<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |

| Offset | Name | Notes |
|--------|------|-------|
| `0x2000` | MFFCW_page | Most permissive access: RW<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0x3000` | MFCW_page | Most permissive access: RW<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| `0xF000` | MBX_IMPL_DEF_page | Most permissive access: RW |

### C2.2.1.1   MBX_CTRL_page, Mailbox Control page

The MBX_CTRL_page characteristics are:

**Purpose**

> Allows access to the configuration registers of the Mailbox.

**Configuration**

**Attributes**

> The MBX_CTRL_page block is of size 4KB.
>
> This block is part of the MHUR.MBX block.

**Default access**

> Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
| --- | --- | --- |
| 0x000 | MBX_BLK_ID | Most permissive access: RO |
| 0x010 | MBX_FEAT_SPT0 | Most permissive access: RO |
| 0x014 | MBX_FEAT_SPT1 | Most permissive access: RO |
| 0x020 | MBX_DBCH_CFG0 | Most permissive access: RO<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |
| 0x030 | MBX_FFCH_CFG0 | Most permissive access: RO<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| 0x040 | MBX_FCH_CFG0 | Most permissive access: RO<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| 0x100 | MBX_CTRL | Most permissive access: RW |
| 0x140 | MBX_FCH_CTRL | Most permissive access: RW<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| 0x144 | MBX_FCG_INT_EN | Most permissive access: RW<br>Register condition: When FCE is implemented and FCGI_SPT == 1<br>Accessor condition: When FCE is implemented |

| Offset | Name | Notes |
|---|---|---|
| `0x400 + (4 * n)for n in 3:0` | MBX_DBCH_INT_ST\<n\> | Most permissive access: RO<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |
| `0x410 + (4 * n)for n in 1:0` | MBX_FFCH_INT_ST\<n\> | Most permissive access: RO<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0x470` | MBX_FCG_INT_ST | Most permissive access: RO<br>Register condition: When FCE is implemented and FCGI_SPT == 1<br>Accessor condition: When FCE is implemented |
| `0x480 + (4 * n)for n in`<br>`↪31:0` | MBX_FCH_GRP\<n\>_INT_ST | Most permissive access: RO<br>Register condition: When FCE is implemented and FCGI_SPT == 1<br>Accessor condition: When FCE is implemented |
| `0xFC8` | MBX_IIDR | Most permissive access: RO |
| `0xFCC` | MBX_AIDR | Most permissive access: RO |
| `0xFD0 + (4 * n)for n in`<br>`↪11:0` | MBX_IMPL_DEF_ID\<n\> | Most permissive access: RO |

### C2.2.1.1.1    MBX_BLK_ID, Mailbox Block Identifier

The MBX_BLK_ID characteristics are:

**Purpose**

Identifies the block as a Mailbox.

**Attributes**

MBX_BLK_ID is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_BLK_ID bit assignments are:



**Bits [31:4]**

Reserved, RES0.

**BLK_ID, bits [3:0]**

Block Identifier

Identifies the type of block that resides in this 64KB.

Reads as 0b0001

Identifies the block as the Mailbox block.

Access to this field is **RO**.

**Accessing MBX_BLK_ID**

Accesses to this register use the following encodings:

**Accessible at offset 0x000 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.2 MBX_FEAT_SPT0, Mailbox Feature Support 0

The MBX_FEAT_SPT0 characteristics are:

**Purpose**

Provides information on the features implemented in the Mailbox.

**Attributes**

MBX_FEAT_SPT0 is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FEAT_SPT0 bit assignments are:

| 31 | 24 | 23 | 20 | 19 | 16 | 15 | 12 | 11 | 8 | 7 | 4 | 3 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | | RASE_SPT | | RME_SPT | | TZE_SPT | | FCE_SPT | | FE_SPT | | DBE_SPT | |

**Bits [31:24]**

Reserved, RES0.

**RASE_SPT, bits [23:20]**

Reliability, Availability and Serviceability Extension Support

The value of this field depends on the implementation of the MHU.

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

| RASE_SPT | Meaning |
|----------|---------|
| 0b0000 | MHU does not implement the RAS extension |
| 0b0010 | MHU implements the RAS extension but does not follow the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |
| 0b0011 | MHU implements the RAS extension and follows the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |

Access to this field is **RO**.

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

Realm Management Extension Support

The value of this field depends on the implementation of the MHU and an optional reset time sampled input **LEGACY_TZ_EN**.

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

| RME_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Realm Management extension |

| RME_SPT | Meaning |
|---------|---------|
| 0b0001 | MHU implements Realm Management extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

When RME is implemented, for the half of the MHU, there can be a **LEGACY_TZ_EN** tie-off present on that side of the MHU.

The value of the **LEGACY_TZ_EN** tie-off is sampled at reset of the side of the MHU which the tie-off is associated with.

When the sampled value of the tie-off is 0b1 the value of this field is always 0x0, otherwise the value of this field is dependent on whether RME is implemented for this half of the MHU.

Access to this field is **RO**.

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

TrustZone Extension Support

The value of this field depends on the implementation of the MHU.

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

| TZE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the TrustZone extension |
| 0b0001 | MHU implements TrustZone extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

Access to this field is **RO**.

**FCE_SPT, bits [11:8]**

Fast Channel Extension Support

The value of this field depends on the implementation of the MHU.

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

| FCE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Fast Channel extension |
| 0b0001 | MHU implements Fast Channel extension |

Access to this field is **RO**.

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

FIFO Extension Support

The value of this field depends on the implementation of the MHU.

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

| FE_SPT | Meaning |
| --- | --- |
| 0b0000 | MHU does not implement the FIFO extension |
| 0b0001 | MHU implements FIFO extension |

Access to this field is **RO**.

**DBE_SPT, bits [3:0]**

Doorbell Extension Support

The value of this field depends on the implementation of the MHU.

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

| DBE_SPT | Meaning |
| --- | --- |
| 0b0000 | MHU does not implement the Doorbell extension |
| 0b0001 | MHU implements Doorbell extension |

Access to this field is **RO**.

**Accessing MBX_FEAT_SPT0**

Accesses to this register use the following encodings:

**Accessible at offset 0x010 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.3 MBX_FEAT_SPT1, Mailbox Feature Support 1

The MBX_FEAT_SPT1 characteristics are:

**Purpose**

Provides information on the features implemented in the Mailbox.

**Attributes**

MBX_FEAT_SPT1 is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FEAT_SPT1 bit assignments are:



**Bits [31:4]**

Reserved, RES0.

**AUTO_OP_SPT, bits [3:0]**

Auto Op Protocol Support

The value of this field depends on the implementation of the MHU.

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

| AUTO_OP_SPT | Meaning |
| --- | --- |
| 0b0000 | Auto Op(Min) is implemented |
| 0b0001 | Auto Op(Full) is implemented |

Access to this field is **RO**.

**Accessing MBX_FEAT_SPT1**

Accesses to this register use the following encodings:

**Accessible at offset 0x014 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.4   MBX_DBCH_CFG0, Mailbox Doorbell Channel Configuration 0

The MBX_DBCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of DBE in Mailbox.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to
MBX_DBCH_CFG0 are RAZ/WI.

**Attributes**

MBX_DBCH_CFG0 is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_DBCH_CFG0 bit assignments are:

| 31                           8 | 7              0 |
|--------------------------------|------------------|
| RES0                           | NUM_DBCH         |

**Bits [31:8]**

Reserved, RES0.

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

Number of Doorbell Channels

Number of DBCH implemented in the Mailbox.

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

| NUM_DBCH | Meaning |
|----------|---------|
| 0x00..0x7F | Number of DBCH is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing MBX_DBCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x020 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.5  MBX_FFCH_CFG0, Mailbox FIFO Channel Configuration 0

The MBX_FFCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FE in Mailbox.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to MBX_FFCH_CFG0 are RAZ/WI.

**Attributes**

MBX_FFCH_CFG0 is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FFCH_CFG0 bit assignments are:



**Bits [31:26]**

Reserved, RES0.

**FFCH_DEPTH, bits [25:16]**

FIFO Channel Depth

Depth of the FIFOs of the FFCHs in the Mailbox.

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

| FFCH_DEPTH | Meaning |
| --- | --- |
| 0b0000000000..0 ↪b1111111111 | FIFO depth is N+1 bytes, where N is the value of this field. |

Access to this field is **RO**.

**Bits [15:12]**

Reserved, RES0.

**M64BA_SPT, bit [11]**

Mailbox 64-bit Access Support

Whether 64-bit accesses to the following registers are supported:

- MFFCW<n>_PAY
- MFFCW<n>_FLG

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

| M64BA_SPT | Meaning |
|---|---|
| 0b0 | 64-bit accesses are not supported |
| 0b1 | 64-bit accesses are supported |

Accesses must be aligned to an 64-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

When 64-bit accesses are supported the:

- MFFCW<n>_PAY register occupies offsets 0x00-0x07 of the MFFCW.

- MFFCW<n>_FLG register occupies offsets 0x008-0x0F of the MFFCW.

When 64-bit accesses are not supported:

- MFFCW<n>_PAY register occupies offsets 0x00-0x03 of the MFFCW and offsets 0x04-0x07 of the MFFCW are RES0.

- MFFCW<n>_FLG register occupies offsets 0x008-0x0B of the MFFCW and offsets 0x0C-0x0F of the MFFCW are RES0.

Access to this field is **RO**.

**M32BA_SPT, bit [10]**

Mailbox 32-bit Access Support

Whether 32-bit accesses to the following registers are supported:

- MFFCW<n>_PAY

- MFFCW<n>_FLG

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

| M32BA_SPT | Meaning |
|---|---|
| 0b0 | 32-bit accesses are not supported |
| 0b1 | 32-bit accesses are supported |

Accesses must be aligned to an 32-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**M16BA_SPT, bit [9]**

Mailbox 16-bit Access Support

Whether 16-bit accesses to the following registers are supported:

- MFFCW<n>_PAY

- MFFCW<n>_FLG

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

| M16BA_SPT | Meaning |
|-----------|---------|
| 0b0 | 16-bit accesses are not supported |
| 0b1 | 16-bit accesses are supported |

Accesses must be aligned to an 16-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**M8BA_SPT, bit [8]**

Mailbox 8-bit Access Support

Whether 8-bit accesses to the following registers are supported:

- MFFCW<n>_PAY

- MFFCW<n>_FLG

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

| M8BA_SPT | Meaning |
|----------|---------|
| 0b0 | 8-bit accesses are not supported |
| 0b1 | 8-bit accesses are supported |

Accesses must be aligned to an 8-bit boundary

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

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

Number of FIFO Channels

The number of FFCHs in the Mailbox.

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

| NUM_FFCH | Meaning |
|----------|---------|
| 0x00..0x3F | Number of FFCHs is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing MBX_FFCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x030 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.6 MBX_FCH_CFG0, Mailbox Fast Channel Configuration 0

The MBX_FCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FCE in the Mailbox.

**Configuration**

This register is present only when FCE is implemented. Otherwise, direct accesses to MBX_FCH_CFG0 are RAZ/WI.

**Attributes**

MBX_FCH_CFG0 is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FCH_CFG0 bit assignments are:
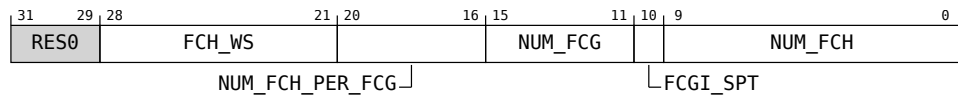


**Bits [31:29]**

Reserved, RES0.

**FCH_WS, bits [28:21]**

Fast Channel Word-Size

Number of bits each FCH implements.

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

| FCH_WS | Meaning |
|--------|---------|
| 0x20 | Fast Channel word-size is 32-bits |
| 0x40 | Fast Channel word-size is 64-bits |

Access to this field is **RO**.

**NUM_FCH_PER_FCG, bits [20:16]**

Number of Fast Channels per Fast Channel Group

Number of FCHs implemented in FCGs for Mailbox.

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

| NUM_FCH_PER_FCG | Meaning |
|-----------------|---------|
| 0b00000..0b11111 | Number of FCHs per FCG is N+1, where N is the value of this field |

Access to this field is **RO**.

**NUM_FCG, bits [15:11]**

Number of Fast Channel Groups

Number of FCGs implemented in Mailbox

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

| NUM_FCG | Meaning |
| --- | --- |
| 0b00000..0 ↪b11111 | Number of FCGs is N+1, where N is the value of this field |

Access to this field is **RO**.

**FCGI_SPT, bit [10]**

Fast Channel Group Interrupt Support

Indicates whether Fast Channel Group Transfer interrupt is implemented for each FCGs.

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

| FCGI_SPT | Meaning |
| --- | --- |
| 0b0 | Fast Channel Group Transfer interrupts are not implemented. |
| 0b1 | Fast Channel Group Transfer interrupts are implemented. |

Access to this field is **RO**.

**NUM_FCH, bits [9:0]**

Number of Fast Channels

Number of FCHs implemented in Mailbox.

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

| NUM_FCH | Meaning | Applies |
| --- | --- | --- |
| 0b0000000000..0 ↪b0111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x40 |
| 0b0000000000..0 ↪b1111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x20 |

Access to this field is **RO**.

**Accessing MBX_FCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x040 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### *C2.2.1.1.7  MBX_CTRL, Mailbox Control*

The MBX_CTRL characteristics are:

**Purpose**

Configures the behavior of the Mailbox.

**Attributes**

MBX_CTRL is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_CTRL bit assignments are:
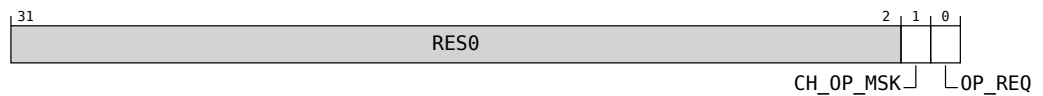


**Bits [31:2]**

Reserved, RES0.

**CH_OP_MSK, bit [1]**

Channel Operational Mask

Controls whether channels need to be idle to allow a controlled entry of the MHUR into a non-operational state.

| CH_OP_MSK | Meaning |
|---|---|
| 0b0 | Channels must be idle to enter a non-operational state |
| 0b1 | Channels status is ignored when considering entry into a non-operational state |

This field has no effect on entry into a non-operation state in an uncontrolled manner for the MHUR.

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**OP_REQ, bit [0]**

Operational Request

Controls whether the MHUR is required to remain in an operational state.

| OP_REQ | Meaning |
|---|---|
| 0b0 | Mailbox is not requested to remain in the operational state |
| 0b1 | Mailbox is requested to remain in the operational state |

The reset behavior of this field is:

• On a MHUR reset, this field resets to `0b0`.

**Accessing MBX_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x100 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RW**.

### C2.2.1.1.8   MBX_FCH_CTRL, Mailbox Fast Channel Control

The MBX_FCH_CTRL characteristics are:

**Purpose**

> Controls the Fast Channels in the Mailbox

**Configuration**

> This register is present only when FCE is implemented. Otherwise, direct accesses to MBX_FCH_CTRL are RAZ/WI.

**Attributes**

> MBX_FCH_CTRL is a 32-bit register.

> This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FCH_CTRL bit assignments are:
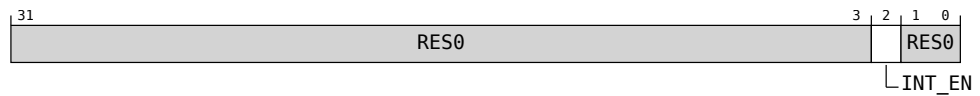


**Bits [31:3]**

Reserved, RES0.

**INT_EN, bit [2]**

Interrupt Enable

Enables interrupts for all FCHs in the Mailbox.

| INT_EN | Meaning |
| --- | --- |
| 0b0 | Interrupts for FCHs, in the Mailbox, are disabled |
| 0b1 | Interrupts for FCHs, in the Mailbox, are enabled |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b1.

**Bits [1:0]**

Reserved, RES0.

**Accessing MBX_FCH_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x140 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RW**.

### C2.2.1.1.9 MBX_FCG_INT_EN, Mailbox Fast Channel Group Interrupt Enable

The MBX_FCG_INT_EN characteristics are:

**Purpose**

Controls whether a FCG contributes to the Mailbox Combined interrupt

**Configuration**

This register is present only when FCE is implemented and FCGI_SPT == 1. Otherwise, direct accesses to MBX_FCG_INT_EN are RAZ/WI.

**Attributes**

MBX_FCG_INT_EN is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FCG_INT_EN bit assignments are:



**MBX_COMB_EN<m>, bits [m], for m = 31 to 0**

| MBX_COMB_EN<m> | Meaning |
| --- | --- |
| 0b0 | FCG<m> does not contribute to the Mailbox Combined interrupt |
| 0b1 | FCG<m> contributes to the Mailbox Combined interrupt |

The field is only implemented, if the associated FCG is implemented. Fields which are not implemented are RES0.

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b1.

**Accessing MBX_FCG_INT_EN**

Accesses to this register use the following encodings:

**Accessible at offset 0x144 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RW**.

### C2.2.1.1.10  MBX_DBCH_INT_ST<n>, Mailbox Doorbell Channel Interrupt Status <n>, n = 0 - 3

The MBX_DBCH_INT_ST<n> characteristics are:

**Purpose**

Indicates whether there is an interrupt outstanding for the DBCH

MBX_DBCH_INT_ST0 has status fields for DBCHs 0 to 31

MBX_DBCH_INT_ST1 has status fields for DBCHs 32 to 63

MBX_DBCH_INT_ST2 has status fields for DBCHs 64 to 95

MBX_DBCH_INT_ST3 has status fields for DBCHs 96 to 127

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MBX_DBCH_INT_ST<n> are RAZ/WI.
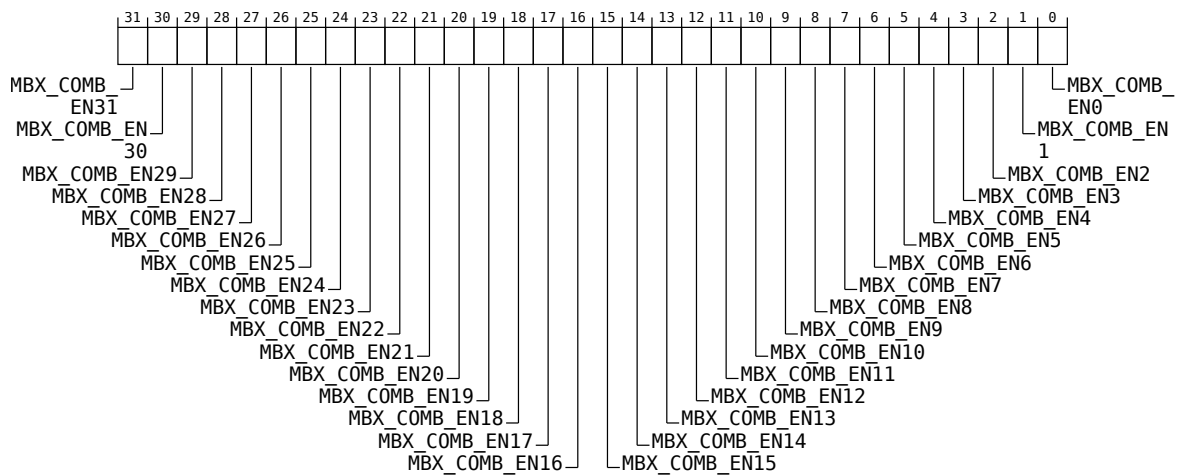
**Attributes**

MBX_DBCH_INT_ST<n> is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_DBCH_INT_ST<n> bit assignments are:



**DBCH_INT_ST<x>, bits [x], for x = 31 to 0**

Doorbell Channel Interrupt Status

Each bit indicates whether there is an outstanding interrupt for the DBCH, that is contributing to the Mailbox Combined interrupt.

| DBCH_INT_ST<x> | Meaning |
| --- | --- |
| 0b0 | No interrupt outstanding for the DBCH or the DBCH is not configured to contribute into the Mailbox Combined interrupt. |

| DBCH_INT_ST\<x\> | Meaning |
|---|---|
| 0b1 | Interrupt outstanding for the DBCH and the DBCH is configured to contribute into the Mailbox Combined interrupt. |

Any fields which are not assigned to a DBCH are Reserved and treated as RAZ/WI

**Accessing MBX_DBCH_INT_ST\<n\>**

Accesses to this register use the following encodings:

**Accessible at offset 0x400 + (4 * n) from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.11  MBX_FFCH_INT_ST<n>, Mailbox FIFO Channel Interrupt Status <n>, n = 0 - 1

The MBX_FFCH_INT_ST<n> characteristics are:

**Purpose**

Indicates whether there is an interrupt outstanding for the FFCH.

MBX_FFCH_INT_ST0 has status fields for FFCHs 0 to 31

MBX_FFCH_INT_ST1 has status fields for FFCHs 32 to 63

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to MBX_FFCH_INT_ST<n> are RAZ/WI.
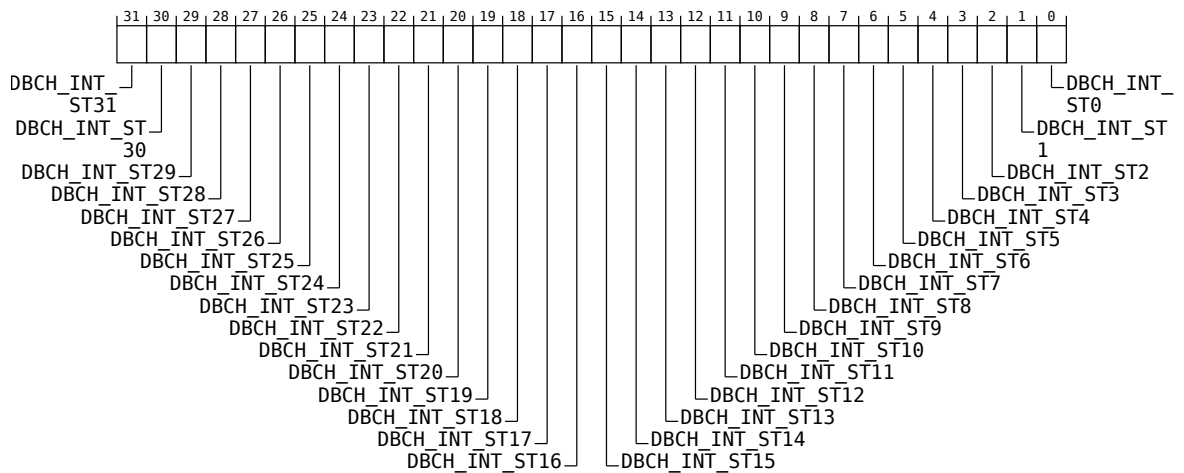
**Attributes**

MBX_FFCH_INT_ST<n> is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FFCH_INT_ST<n> bit assignments are:



**FFCH_INT_ST<x>, bits [x], for x = 31 to 0**

FIFO Channel Interrupt Status

Each bit indicates whether there is an outstanding interrupt for the FFCH, that is contributing to the Mailbox Combined interrupt.

| FFCH_INT_ST<x> | Meaning |
| --- | --- |
| 0b0 | No interrupt outstanding for the FFCH or the FFCH is not configured to contribute into the Mailbox Combined interrupt. |
| 0b1 | Interrupt outstanding for the FFCH and the FFCH is configured to contribute into the Mailbox Combined interrupt. |

Any fields which are not assigned to a FFCH are Reserved and treated as RAZ/WI

**Accessing MBX_FFCH_INT_ST<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0x410 + (4 * n) from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.12 MBX_FCG_INT_ST, Mailbox Fast Channel Group Interrupt Status

The MBX_FCG_INT_ST characteristics are:

**Purpose**

Provides the status of each Fast Channel Group Transfer interrupt

**Configuration**

This register is present only when FCE is implemented and FCGI_SPT == 1. Otherwise, direct accesses to MBX_FCG_INT_ST are RAZ/WI.

**Attributes**

MBX_FCG_INT_ST is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FCG_INT_ST bit assignments are:
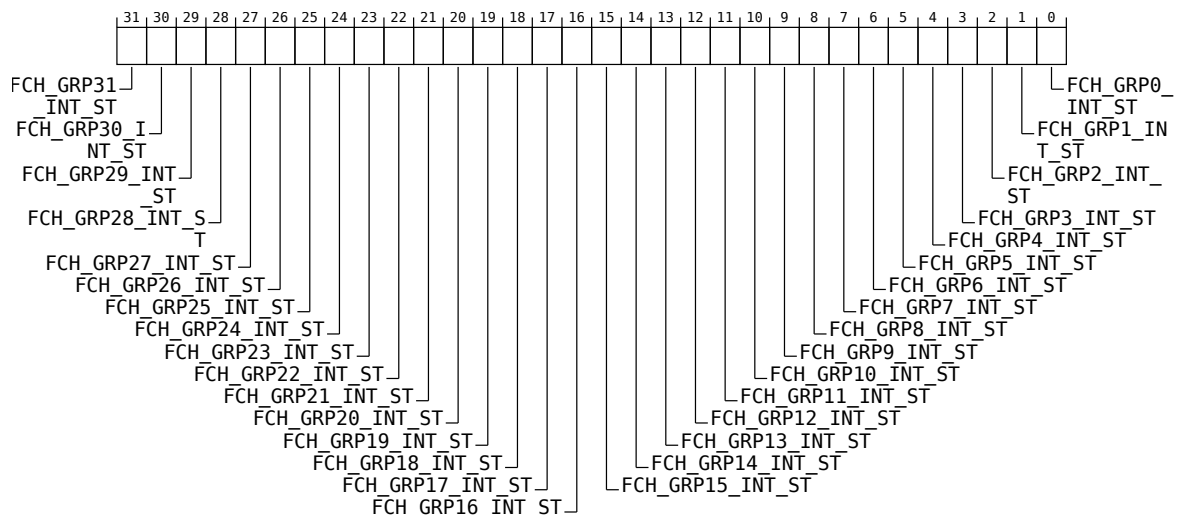


**FCH_GRP<n>_INT_ST, bits [n], for n = 31 to 0**

Fast Channel Group Interrupt Status

Each bit indicates whether there is an outstanding interrupt for an FCH which is part of FCG<m>, that is contributing to the Mailbox Combined interrupt.

| FCH_GRP<n>_INT_ST | Meaning |
| --- | --- |
| 0b0 | No Fast Channel Group Transfer interrupt for FCG<m> or FCG<m> is configured not to contribute to the Mailbox Combined interrupt. |
| 0b1 | Fast Channel Group Transfer interrupt for FCG<m> and FCG<m> is configured to contribute to the Mailbox Combined interrupt. |

Any fields which are not assigned to an implemented FCG are RES0.

**Accessing MBX_FCG_INT_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x470 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.13  MBX_FCH_GRP<n>_INT_ST, Mailbox Fast Channel Group <n> Interrupt Status, n = 0 - 31

The MBX_FCH_GRP<n>_INT_ST characteristics are:

**Purpose**

Provides the status of each FCH with the Fast Channel Group<n>

The number of MBX_FCH_GRP<n>_INT_ST is set by the value of MBX_FCH_CFG0.NUM_FCG, with any unused registers being RES0.

The number of fields within each register depends on the value of MBX_FCH_CFG0.NUM_FCH_PER_FCG, with any unused fields being RES0.

**Configuration**

This register is present only when FCE is implemented and FCGI_SPT == 1. Otherwise, direct accesses to MBX_FCH_GRP<n>_INT_ST are RAZ/WI.

**Attributes**

MBX_FCH_GRP<n>_INT_ST is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_FCH_GRP<n>_INT_ST bit assignments are:
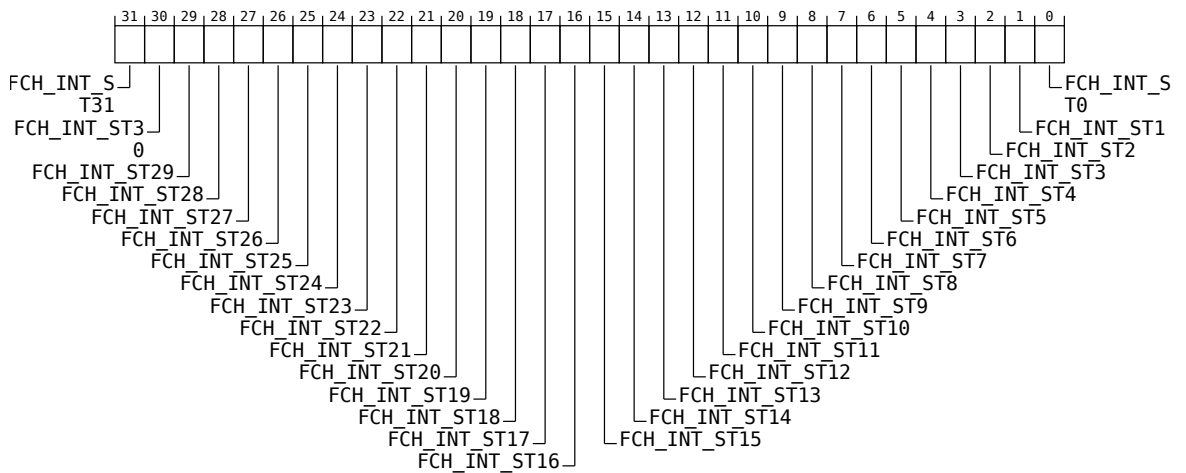


**FCH_INT_ST<m>, bits [m], for m = 31 to 0**

Fast Channel Interrupt Status

Each bit indicates whether there is an outstanding interrupt for the FCH, within FCG<n>, that is contributing to the Mailbox Combined interrupt.

| FCH_INT_ST<m> | Meaning |
| --- | --- |
| 0b0 | No interrupt outstanding for FCH<x> |
| 0b1 | Outstanding interrupt for FCH<x> |

Where x is the FCH number and is calculated by the following formula: n*NUM_FCH_PER_FCG + m.

Any fields which are not assigned to an implemented FCH are RES0.

**Accessing MBX_FCH_GRP<n>_INT_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x480 + (4 * n) from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.14   MBX_IIDR, Mailbox Implementer Identification Register

The MBX_IIDR characteristics are:

**Purpose**

> This field provides information on the implementation of the MHU

**Attributes**

> MBX_IIDR is a 32-bit register.

> This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_IIDR bit assignments are:

| 31                   20 | 19        16 | 15        12 | 11                          0 |
|-------------------------|--------------|--------------|-------------------------------|
| PRODUCT_ID              | VARIANT      | REVISION     | IMPLEMENTER                   |

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

Product ID of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Variant or major revision of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Revision or minor version of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**IMPLEMENTER, bits [11:0]**

Implementer ID

Contains the JEP106 identification information as follows:

- 11:8 - JEP106 continuation code of implementer
- 7 - Always 0
- 6:0 - JEP106 identity code of implementer

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**Accessing MBX_IIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFC8 from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.15 MBX_AIDR, Mailbox Architecture Identification Register

The MBX_AIDR characteristics are:

**Purpose**

Provides information on the version of the MHU architecture implemented in this implementation of the MHU.

**Attributes**

MBX_AIDR is a 32-bit register.

This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_AIDR bit assignments are:
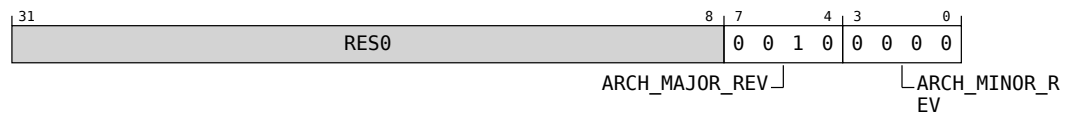


**Bits [31:8]**

Reserved, RES0.

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

MHU Architecture Major Revision

In the current implementation, this field is `0b0010`.

None

| ARCH_MAJOR_REV | Meaning |
|---|---|
| `0b0000` | MHUv1 or unknown architecture versions |
| `0b0001` | MHUv2 |
| `0b0010` | MHUv3 |

All other values are Reserved

Access to this field is **RO**.

**ARCH_MINOR_REV, bits [3:0]**

MHU Architecture Minor Revision

In the current implementation, this field is `0b0000`.

None

| ARCH_MINOR_REV | Meaning |
|---|---|
| `0b0000` | Minor revision 0 of the major architecture |
| `0b0001` | Minor revision 1 of the major architecture |

All other values are Reserved

Access to this field is **RO**.

**Additional information**

The legal combinations for the ARCH_MAJOR_REV and ARCH_MINOR_REV fields are as follows:

| ARCH_MAJOR_REV | ARCH_MINOR_REV | Description |
| --- | --- | --- |
| 0x0 | 0x0 | MHUv1 |
| 0x1 | 0x0 | MHUv2.0 |
| 0x1 | 0x1 | MHUv2.1 |
| 0x2 | 0x0 | MHUv3.0 |

MHU implementations based on this architecture, have the ARCH_MAJOR_REV set to 0x2 and ARCH_MINOR_REV set to 0x0.

**Accessing MBX_AIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFCC from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.1.16  MBX_IMPL_DEF_ID<n>, IMPDEF Register, n = 0 - 11

The MBX_IMPL_DEF_ID<n> characteristics are:

**Purpose**

>This register is for IMPLEMENTATION DEFINED Identification Registers which can be used to identify the implementation of the MHU Mailbox

>An implementation can do the following with this register:

>- Chose whether a register is present or not.

>- The name of the register.

>- The access permission of the registers, so long as they are not more permissive than the architecture defines for the IMPLEMENTATION DEFINED register.

>- The names, behavior and reset value of any fields.

>- The access permissions of any fields, so long as they are not more permissive than the register.

>If the MBX_IMPL_DEF_ID<n> is not present then the location is Reserved and treated as RES0

**Attributes**

>MBX_IMPL_DEF_ID<n> is a 32-bit register.

>This register is part of the MHUR.MBX.MBX_CTRL_page block.

**Field descriptions**

The MBX_IMPL_DEF_ID<n> bit assignments are:

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

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

IMPLEMENTATION DEFINED

**Accessing MBX_IMPL_DEF_ID<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0xFD0 + (4 * n) from MHUR.MBX.MBX_CTRL_page**

Access on this interface is **RO**.

### C2.2.1.2 MDBCW_page, Mailbox Doorbell Channel Windows Page

The MDBCW_page characteristics are:

**Purpose**

Allows access to the MDBCW.

**Configuration**

This Register Block is present only when DBE is implemented.

**Attributes**

The MDBCW_page block is of size 4KB.

This block is part of the MHUR.MBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
| --- | --- | --- |
| `0x0000 + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_ST | Most permissive access: RO |
| `0x0004 + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_ST_MSK | Most permissive access: RO |
| `0x0008 + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_CLR | Most permissive access: WO/RAZ |
| `0x0010 + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_MSK_ST | Most permissive access: RO |
| `0x0014 + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_MSK_SET | Most permissive access: WO/RAZ |
| `0x0018 + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_MSK_CLR | Most permissive access: WO/RAZ |
| `0x001C + (32 * n)`for n in ↪127:0 | MDBCW\<n\>_CTRL | Most permissive access: RW |

### C2.2.1.2.1  MDBCW<n>_ST, Mailbox Doorbell Channel Window <n> Status, n = 0 - 127

The MDBCW<n>_ST characteristics are:

**Purpose**

Status of the flags for DBCH<n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_ST are RAZ/WI.

**Attributes**

MDBCW<n>_ST is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_ST bit assignments are:
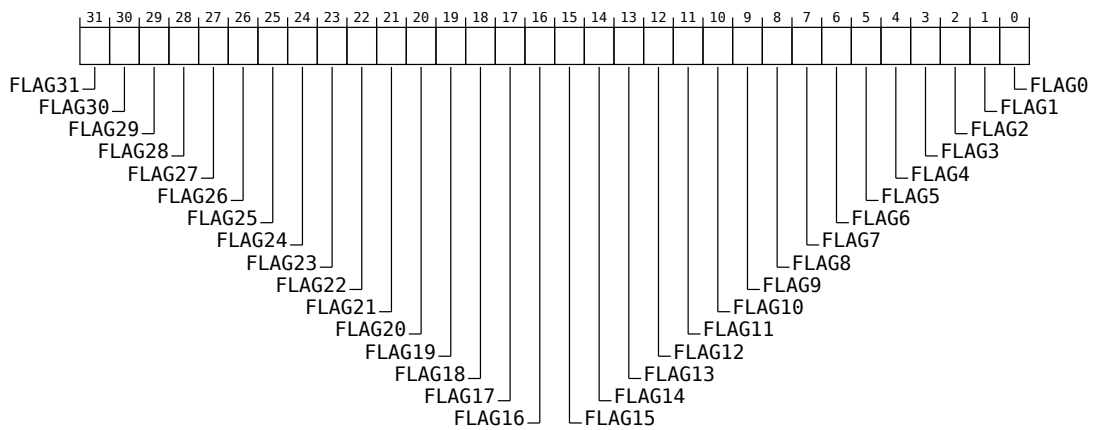


**FLAG<x>, bits [x], for x = 31 to 0**

Flag

| FLAG<x> | Meaning |
|---------|---------|
| 0b0 | Flag<x> bit is not set |
| 0b1 | Flag<x> bit is set |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**Accessing MDBCW<n>_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0000 + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **RO**.

### C2.2.1.2.2 MDBCW<n>_ST_MSK, Mailbox Doorbell Channel Window <n> Status Masked, n = 0 - 127

The MDBCW<n>_ST_MSK characteristics are:

**Purpose**

Masked status of DBCH<n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_ST_MSK are RAZ/WI.

**Attributes**

MDBCW<n>_ST_MSK is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_ST_MSK bit assignments are:
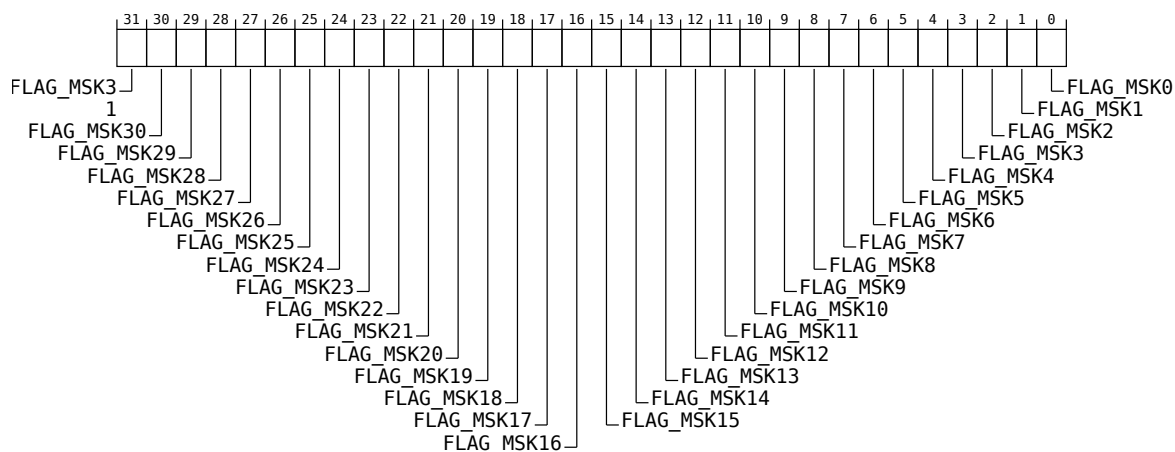


**FLAG_MSK<x>, bits [x], for x = 31 to 0**

Flag Masked

Provides the masked status of flag<x>.

| FLAG_MSK<x> | Meaning |
|---|---|
| 0b0 | Flag<x> and does not contribute to the DBCH Channel Transfer interrupt. |
| 0b1 | Flag Masked<x> and contribute to the DBCH Channel Transfer interrupt. |

The reset behavior of this field is:

• On a MHUR reset, this field resets to 0b0.

**Accessing MDBCW<n>_ST_MSK**

Accesses to this register use the following encodings:

**Accessible at offset 0x0004 + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **RO**.

### C2.2.1.2.3   MDBCW<n>_CLR, Mailbox Doorbell Channel Window <n> Clear, n = 0 - 127

The MDBCW<n>_CLR characteristics are:

**Purpose**

Clear flag of DBCH<n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_CLR are RAZ/WI.

**Attributes**

MDBCW<n>_CLR is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_CLR bit assignments are:
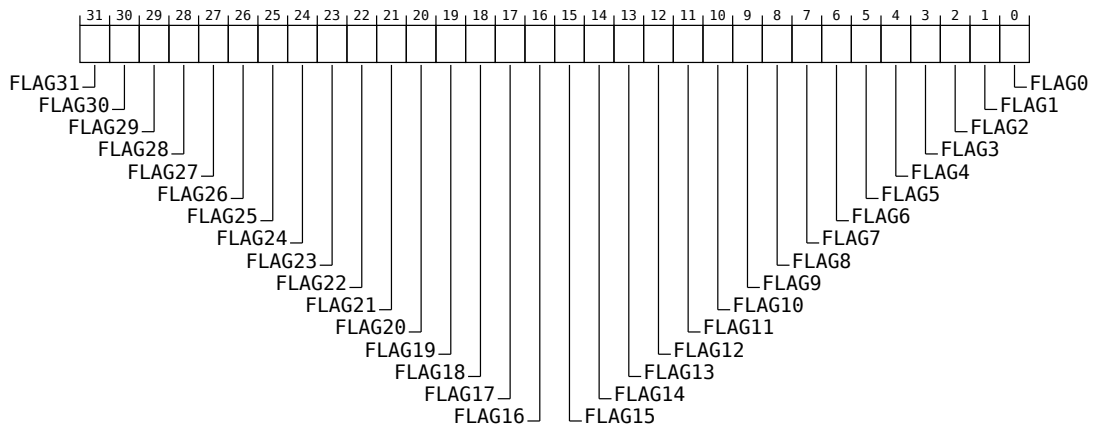


**FLAG<x>, bits [x], for x = 31 to 0**

| FLAG<x> | Meaning |
|---------|---------|
| 0b0 | No effect |
| 0b1 | Sets the associated bit in the PDBCW<n>_ST and MDBCW<n>_ST register to 0 |

**Accessing MDBCW<n>_CLR**

Accesses to this register use the following encodings:

**Accessible at offset 0x0008 + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **WO/RAZ**.

### C2.2.1.2.4 MDBCW<n>_MSK_ST, Mailbox Doorbell Channel Window <n> Mask Status, n = 0 - 127

The MDBCW<n>_MSK_ST characteristics are:

**Purpose**

Status of the mask for DBCH<n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_MSK_ST are RAZ/WI.

**Attributes**

MDBCW<n>_MSK_ST is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_MSK_ST bit assignments are:
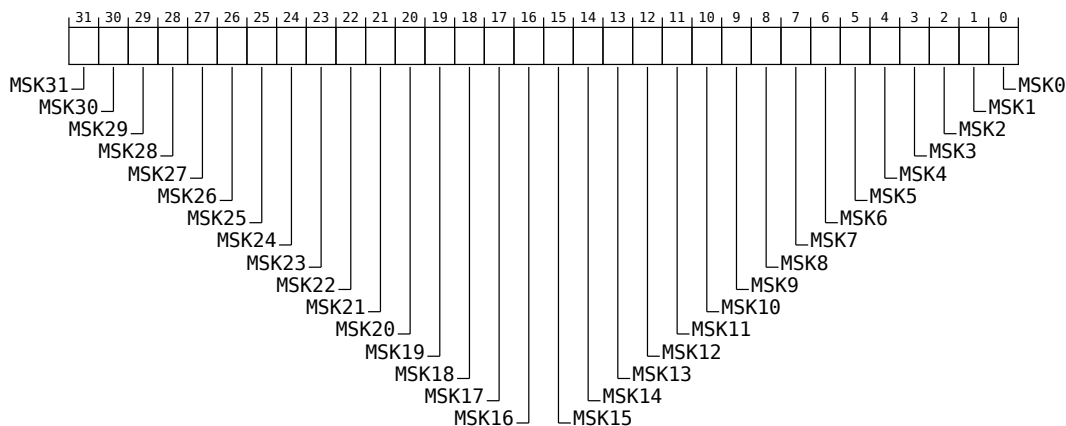


**MSK<x>, bits [x], for x = 31 to 0**

| MSK<x> | Meaning |
| --- | --- |
| 0b0 | Flag<x> in MDBCW<n>_ST register is not masked |
| 0b1 | Flag<x> in MDBCW<n>_ST register is masked |

The reset behavior of this field is:

• On a MHUR reset, this field resets to `0b0`.

**Accessing MDBCW<n>_MSK_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0010 + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **RO**.

### C2.2.1.2.5   MDBCW<n>_MSK_SET, Mailbox Doorbell Channel Window <n> Mask Set, n = 0 - 127

The MDBCW<n>_MSK_SET characteristics are:

**Purpose**

Sets a bit in the mask for DBCH<n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_MSK_SET are RAZ/WI.

**Attributes**

MDBCW<n>_MSK_SET is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_MSK_SET bit assignments are:
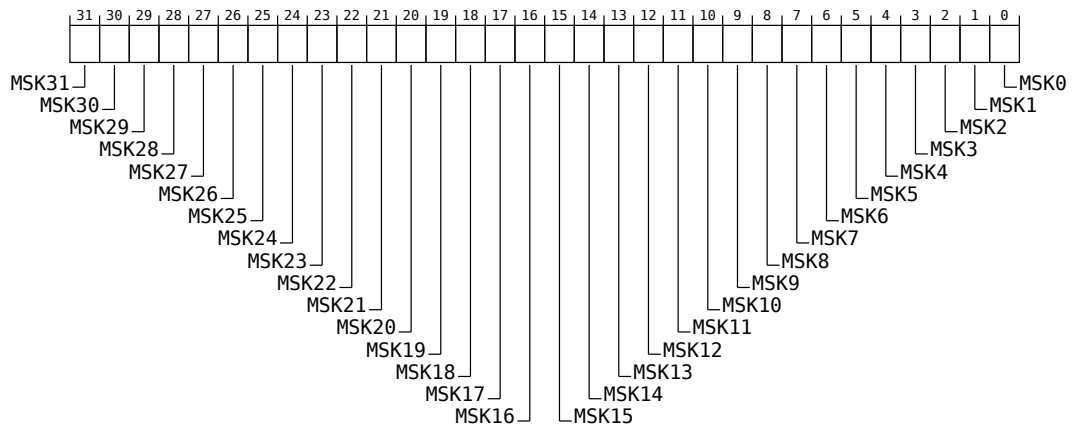


**MSK<x>, bits [x], for x = 31 to 0**

| MSK<x> | Meaning |
|--------|---------|
| 0b0 | No effect |
| 0b1 | Set associated bit in MDBCW<n>_MSK_ST register to 1 |

**Accessing MDBCW<n>_MSK_SET**

Accesses to this register use the following encodings:

**Accessible at offset 0x0014 + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **WO/RAZ**.

### C2.2.1.2.6 MDBCW<n>_MSK_CLR, Mailbox Doorbell Channel Window <n> Mask Clear, n = 0 - 127

The MDBCW<n>_MSK_CLR characteristics are:

**Purpose**

Clears a bit in the mask for DBCH<n>

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_MSK_CLR are RAZ/WI.

**Attributes**

MDBCW<n>_MSK_CLR is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_MSK_CLR bit assignments are:
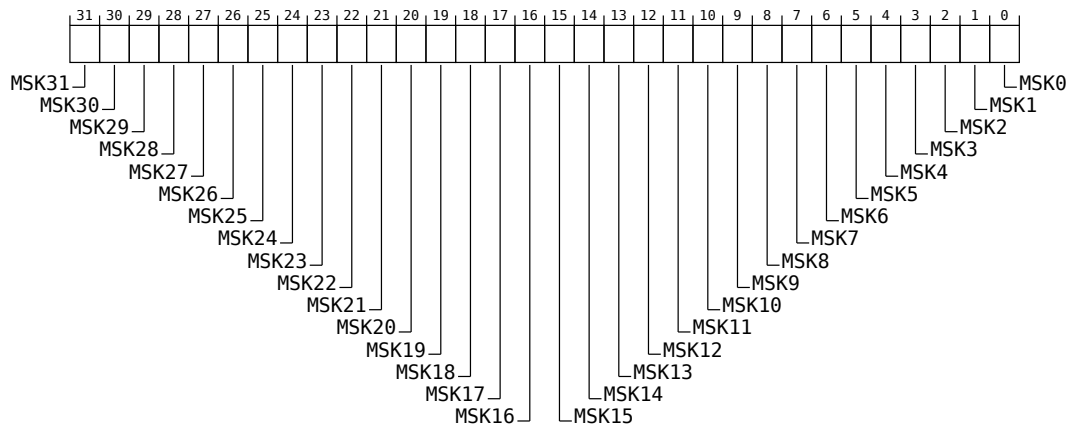


**MSK<x>, bits [x], for x = 31 to 0**

| MSK<x> | Meaning |
|--------|---------|
| 0b0 | No effect |
| 0b1 | Set associated bit in MDBCW<n>_MSK_ST register to 0 |

**Accessing MDBCW<n>_MSK_CLR**

Accesses to this register use the following encodings:

**Accessible at offset 0x0018 + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **WO/RAZ**.

### C2.2.1.2.7 MDBCW<n>_CTRL, Mailbox Doorbell Channel Window <n> Control, n = 0 - 127

The MDBCW<n>_CTRL characteristics are:

**Purpose**

Configures the behavior of DBCH <n>.

**Configuration**

This register is present only when DBE is implemented. Otherwise, direct accesses to MDBCW<n>_CTRL are RAZ/WI.

**Attributes**

MDBCW<n>_CTRL is a 32-bit register.

This register is part of the MHUR.MBX.MDBCW_page block.

**Field descriptions**

The MDBCW<n>_CTRL bit assignments are:
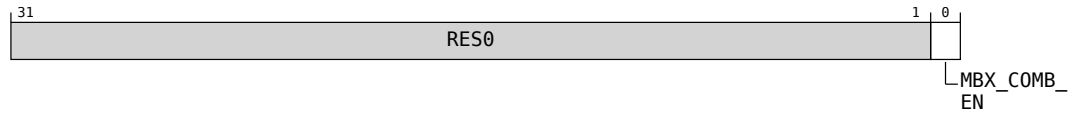


**Bits [31:1]**

Reserved, RES0.

**MBX_COMB_EN, bit [0]**

Mailbox Combined Enable

Controls whether events from this DBCH contribute to the Mailbox Combined interrupt.

| MBX_COMB_EN | Meaning |
| --- | --- |
| 0b0 | DBCH interrupts do not contribute to the Mailbox Combined interrupt |
| 0b1 | DBCH interrupts contribute to the Mailbox Combined interrupt |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b1.

**Accessing MDBCW<n>_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x001C + (32 * n) from MHUR.MBX.MDBCW_page**

Access on this interface is **RW**.

### C2.2.1.3  MFFCW_page, Mailbox FIFO Channel Window Page

The MFFCW_page characteristics are:

**Purpose**

Allows access to the MFFCW.

Depending on the size of the access to the MFFCW_PAY and MFFCW_FLG registers and whether the access size is supported the access is treated as follows:

- 8-bit access and MBX_FFCH_CFG0.M8BA_SPT == 0b1 the access behaves as defined in MFFCW<n>_PAY8 or MFFCW<n>_FLG8.

- 8-bit access and MBX_FFCH_CFG0.M8BA_SPT == 0b0 the access is treated as an unsupported access.

- 16-bit access and MBX_FFCH_CFG0.M16BA_SPT == 0b1 the access behaves as defined in MFFCW<n>_PAY16 or MFFCW<n>_FLG16.

- 16-bit access and MBX_FFCH_CFG0.M16BA_SPT == 0b0 the access is treated as an unsupported access.

- 32-bit access and MBX_FFCH_CFG0.M32BA_SPT == 0b1 the access behaves as defined in MFFCW<n>_PAY32 or MFFCW<n>_FLG32.

- 32-bit access and MBX_FFCH_CFG0.M32BA_SPT == 0b0 the access is treated as an unsupported access.

- 64-bit access and MBX_FFCH_CFG0.M64BA_SPT == 0b1 the access behaves as defined in MFFCW<n>_PAY64 or MFFCW<n>_FLG64.

- 64-bit access and MBX_FFCH_CFG0.M64BA_SPT == 0b0 the access is treated as an unsupported access.

**Configuration**

This Register Block is present only when FE is implemented.

**Attributes**

The MFFCW_page block is of size 4KB.

This block is part of the MHUR.MBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x000 + (64 * n)for n in 63:0` | MFFCW<n>_PAY8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M8BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | MFFCW<n>_PAY16 | Most permissive access: RO<br>Register condition: When M16BA_SPT<br>Accessor condition: When M16BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x000 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY32 | Most permissive access: RO<br>Register condition: When M32BA_SPT<br>Accessor condition: When M32BA_SPT |
| `0x000 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY64 | Most permissive access: RO<br>Register condition: When M64BA_SPT<br>Accessor condition: When M64BA_SPT |
| `0x001 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M8BA_SPT |
| `0x002 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M8BA_SPT |
| `0x002 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY16 | Most permissive access: RO<br>Register condition: When M16BA_SPT<br>Accessor condition: When M16BA_SPT |
| `0x003 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M8BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY16 | Most permissive access: RO<br>Register condition: When M16BA_SPT<br>Accessor condition: When M64BA_SPT and M16BA_SPT |
| `0x004 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY32 | Most permissive access: RO<br>Register condition: When M32BA_SPT<br>Accessor condition: When M64BA_SPT and M32BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x005 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO Register condition: When M8BA_SPT Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x006 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO Register condition: When M8BA_SPT Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x006 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY16 | Most permissive access: RO Register condition: When M16BA_SPT Accessor condition: When M64BA_SPT and M16BA_SPT |
| `0x007 + (64 * n)for n in 63:0` | MFFCW\<n>_PAY8 | Most permissive access: RO Register condition: When M8BA_SPT Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x008 + (64 * n)for n in 63:0` | MFFCW\<n>_FLG8 | Most permissive access: RO Register condition: When M8BA_SPT Accessor condition: When M8BA_SPT |
| `0x008 + (64 * n)for n in 63:0` | MFFCW\<n>_FLG16 | Most permissive access: RO Register condition: When M16BA_SPT Accessor condition: When M16BA_SPT |
| `0x008 + (64 * n)for n in 63:0` | MFFCW\<n>_FLG32 | Most permissive access: RO Register condition: When M32BA_SPT Accessor condition: When M32BA_SPT |
| `0x008 + (64 * n)for n in 63:0` | MFFCW\<n>_FLG64 | Most permissive access: RO Register condition: When M64BA_SPT Accessor condition: When M64BA_SPT |
| `0x009 + (64 * n)for n in 63:0` | MFFCW\<n>_FLG8 | Most permissive access: RO Register condition: When M8BA_SPT Accessor condition: When M8BA_SPT |

| Offset | Name | Notes |
|---|---|---|
| `0x00A + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M8BA_SPT |
| `0x00A + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG16 | Most permissive access: RO<br>Register condition: When M16BA_SPT<br>Accessor condition: When M16BA_SPT |
| `0x00B + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M8BA_SPT |
| `0x00C + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x00C + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG16 | Most permissive access: RO<br>Register condition: When M16BA_SPT<br>Accessor condition: When M64BA_SPT and M16BA_SPT |
| `0x00C + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG32 | Most permissive access: RO<br>Register condition: When M32BA_SPT<br>Accessor condition: When M64BA_SPT and M32BA_SPT |
| `0x00D + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x00E + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG8 | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x00E + (64 * n)for n in 63:0` | MFFCW\<n\>_FLG16 | Most permissive access: RO<br>Register condition: When M16BA_SPT<br>Accessor condition: When M64BA_SPT and M16BA_SPT |

| Offset | Name | Notes |
|--------|------|-------|
| `0x00F + (64 * n)for n in 63:0` | **MFFCW\<n>_FLG8** | Most permissive access: RO<br>Register condition: When M8BA_SPT<br>Accessor condition: When M64BA_SPT and M8BA_SPT |
| `0x0010 + (64 * n)for n in 63:0` | **MFFCW\<n>_INT_ST** | Most permissive access: RO |
| `0x0014 + (64 * n)for n in 63:0` | **MFFCW\<n>_INT_CLR** | Most permissive access: WO/RAZ |
| `0x0018 + (64 * n)for n in 63:0` | **MFFCW\<n>_INT_EN** | Most permissive access: RW |
| `0x0020 + (64 * n)for n in 63:0` | **MFFCW\<n>_CTRL** | Most permissive access: RW |
| `0x0024 + (64 * n)for n in 63:0` | **MFFCW\<n>_ST** | Most permissive access: RO |
| `0x0028 + (64 * n)for n in 63:0` | **MFFCW\<n>_FIFO_POP** | Most permissive access: WO/RAZ |
| `0x002C + (64 * n)for n in 63:0` | **MFFCW\<n>_TIDE** | Most permissive access: RW |

### C2.2.1.3.1 MFFCW<n>_PAY8, Mailbox FIFO Channel Window <n> Payload (8-bit access), n = 0 - 63

The MFFCW<n>_PAY8 characteristics are:

**Purpose**

> An 8-bit access to the MFFCW<n>_PAY register. Accesses must be aligned to any 8-bit boundary within the MFFCW<n>_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

> 8-bit accesses are only supported, if MBX_FFCH_CFG0.M8BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

> If M64BA_SPT is set to 1 the MFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Mailbox FIFO Channel Window <n>.

> If M64BA_SPT is set to 0 the MFFCW<n>_PAY register occupies offsets 0x00-0x03 and offsets 0x04-0x07 are reserved, within the Mailbox FIFO Channel Window <n>.

> A read of this registers reads the byte at the head of the FIFO, if the FIFO is not empty otherwise an IMPLEMENTATION DEFINED value is returned.

> If the MFFCW<n>_CTRL.RA_EN field is set to 0b1, the byte read from the FIFO is also popped from the FIFO.

> On a read of this registers the values of the data flags, associated with the byte read from the FIFO are stored in the Flag History Buffer.

**Configuration**

> This register is present only when FE is implemented and M8BA_SPT. Otherwise, direct accesses to MFFCW<n>_PAY8 are RAZ/WI.

**Attributes**

> MFFCW<n>_PAY8 is a 8-bit register.

> This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_PAY8 bit assignments are:

```
7                    0
┌────────────────────┐
│        PAY         │
└────────────────────┘
```
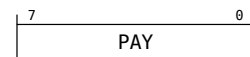
**PAY, bits [7:0]**

Payload read from FIFO

One byte is read from the FIFO, starting from the head of the FIFO.

A byte read from the FIFO can be either valid or invalid. A valid byte has an 8-bit data value and data flags associated with it. An invalid byte has an IMPLEMENTATION DEFINED data value and no data flags associated with it.

The data flags of the byte are stored in the Flag History Buffer(FHB) in entry 0 of the FHB.

With all other entries being marked as invalid.

If MFFCW<n>_CTRL.RA_EN is set to 0b1, when the byte is read from the FIFO:

- It is also removed from the FIFO.

- A Transfer Acknowledge event is generated if that byte was associated with the ACK and EOT data flags.

Otherwise the byte still remains in the FIFO and a subsequent read of the same size would return the same byte.

**Accessing MFFCW<n>_PAY8**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**Read at offset 0x001 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**Read at offset 0x002 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**Read at offset 0x003 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**When M64BA_SPT**

**Read at offset 0x004 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**When M64BA_SPT**

**Read at offset 0x005 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**When M64BA_SPT**

**Read at offset 0x006 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

**When M64BA_SPT**

**Read at offset 0x007 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY8;
```

### C2.2.1.3.2 MFFCW<n>_PAY16, Mailbox FIFO Channel Window <n> Payload (16-bit access), n = 0 - 63

The MFFCW<n>_PAY16 characteristics are:

**Purpose**

A 16-bit access to the MFFCW<n>_PAY register. Accesses must be aligned to any 16-bit boundary within the MFFCW<n>_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

16-bit accesses are only supported, if MBX_FFCH_CFG0.M16BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If M64BA_SPT is set to 1 the MFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Mailbox FIFO Channel Window <n>.

If M64BA_SPT is set to 0 the MFFCW<n>_PAY register occupies offsets 0x00-0x03 and offsets 0x04-0x07 are reserved, within the Mailbox FIFO Channel Window <n>.

A read of this register reads up to two bytes starting at the head of the FIFO, if the FIFO is not empty otherwise an IMPLEMENTATION DEFINED value is returned.

If the MFFCW<n>_CTRL.RA_EN field is set to 0b1, the bytes read from the FIFO is also popped from the FIFO.

On a read of this register, the values of the data flags, associated with the bytes read from the FIFO are stored in the Flag History Buffer.

**Configuration**

This register is present only when FE is implemented and M16BA_SPT. Otherwise, direct accesses to MFFCW<n>_PAY16 are RAZ/WI.

**Attributes**

MFFCW<n>_PAY16 is a 16-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_PAY16 bit assignments are:

| 15 | 0 |
|---|---|
| PAY | |
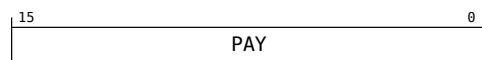
**PAY, bits [15:0]**

Payload read from FIFO

Two bytes are read from the FIFO, starting from the head of the FIFO.

A byte read from the FIFO can be either valid or invalid. A valid byte has an 8-bit data value and data flags associated with it. An invalid byte has an IMPLEMENTATION DEFINED data value and no data flags associated with it.

The data values of all bytes read from the FIFO, are arrange in the PAY field depending on the value of the MFFCW<n>_CTRL.MSBF field and the byte number.

**MFFCW<n>_CTRL.MSBF == 0b0**

Bytes are arranged starting with the first byte read from the FIFO in the LSB of the PAY field and the last byte read from the FIFO in the MSB of the PAY field.

**MFFCW<n>_CTRL.MSBF == 0b1**

Bytes are arranged starting with the first byte read from the FIFO in the MSB of the PAY field and the last byte read from the FIFO in the LSB of the PAY field.

The data flags of each byte are stored in the Flag History Buffer(FHB) starting with the first byte read in entry 0 of the FHB and the last byte read in entry 1 of the FHB.

With all other entries being marked as invalid.

If MFFCW<n>_CTRL.RA_EN is set to 0b1, when a byte is read from the FIFO:

- Tt is also removed from the FIFO.
- A Transfer Acknowledge event is generated if that byte was associated with the ACK and EOT data flags.

Otherwise the byte still remains in the FIFO and a subsequent read of the same size would return the same bytes.

**Accessing MFFCW<n>_PAY16**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY16;
```

**Read at offset 0x002 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY16;
```

**When M64BA_SPT**

**Read at offset 0x004 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY16;
```

**When M64BA_SPT**

**Read at offset 0x006 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY16;
```

### C2.2.1.3.3 MFFCW<n>_PAY32, Mailbox FIFO Channel Window <n> Payload (32-bit access), n = 0 - 63

The MFFCW<n>_PAY32 characteristics are:

**Purpose**

A 32-bit access to the MFFCW<n>_PAY register. Accesses must be aligned to any 32-bit boundary within the MFFCW<n>_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

32-bit accesses are only supported, if MBX_FFCH_CFG0.M32BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If M64BA_SPT is set to 1 the MFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Mailbox FIFO Channel Window <n>.

If M64BA_SPT is set to 0 the MFFCW<n>_PAY register occupies offsets 0x00-0x03 and offsets 0x04-0x07 are reserved, within the Mailbox FIFO Channel Window <n>.

A read of this register reads up to four bytes starting at the head of the FIFO, if the FIFO is not empty otherwise an IMPLEMENTATION DEFINED value is returned.

If the MFFCW<n>_CTRL.RA_EN field is set to 0b1, the bytes read from the FIFO is also popped from the FIFO.

On a read of this register, the values of the data flags, associated with the bytes read from the FIFO are stored in the Flag History Buffer.

**Configuration**

This register is present only when FE is implemented and M32BA_SPT. Otherwise, direct accesses to MFFCW<n>_PAY32 are RAZ/WI.

**Attributes**

MFFCW<n>_PAY32 is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_PAY32 bit assignments are:

| 31 | 0 |
|---|---|
| PAY | |
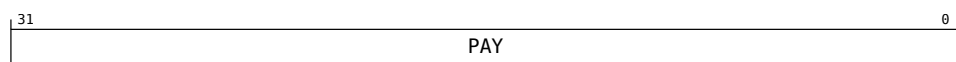
**PAY, bits [31:0]**

Payload read from FIFO

Four bytes are read from the FIFO, starting from the head of the FIFO.

A byte read from the FIFO can be either valid or invalid. A valid byte has an 8-bit data value and data flags associated with it. An invalid byte has an IMPLEMENTATION DEFINED data value and no data flags associated with it.

The data values of all bytes read from the FIFO, are arrange in the PAY field depending on the value of the MFFCW<n>_CTRL.MSBF field and the byte number.

**MFFCW<n>_CTRL.MSBF == 0b0**

Bytes are arranged starting with the first byte read from the FIFO in the LSB of the PAY field and the last byte read from the FIFO in the MSB of the PAY field.

**MFFCW<n>_CTRL.MSBF == 0b1**

Bytes are arranged starting with the first byte read from the FIFO in the MSB of the PAY field and the last byte read from the FIFO in the LSB of the PAY field.

The data flags of each byte are stored in the Flag History Buffer(FHB) starting with the first byte read in entry 0 of the FHB and the last byte read in entry 3 of the FHB.

With all other entries being marked as invalid.

If MFFCW<n>_CTRL.RA_EN is set to 0b1, when a byte is read from the FIFO:

- It is also removed from the FIFO.
- A Transfer Acknowledge event is generated if that byte was associated with the ACK and EOT data flags.

Otherwise the byte still remains in the FIFO and a subsequent read of the same size would return the same bytes.

**Accessing MFFCW<n>_PAY32**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY32;
```

**When M64BA_SPT**

**Read at offset 0x004 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY32;
```

### C2.2.1.3.4 MFFCW<n>_PAY64, Mailbox FIFO Channel Window <n> Payload (64-bit access), n = 0 - 63

The MFFCW<n>_PAY64 characteristics are:

**Purpose**

A 64-bit access to the MFFCW<n>_PAY register. Accesses must be aligned to any 64-bit boundary within the MFFCW<n>_PAY register, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to be an aligned access.

64-bit accesses are only supported, if MBX_FFCH_CFG0.M64BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

When MBX_FFCH_CFG0.M64BA_SPT is set to 0b1, the MFFCW<n>_PAY register occupies offsets 0x00-0x07 within the Mailbox FIFO Channel Window <n>.

A read of this register reads up to eight bytes starting at the head of the FIFO, if the FIFO is not empty otherwise an IMPLEMENTATION DEFINED value is returned.

If the MFFCW<n>_CTRL.RA_EN field is set to 0b1, the bytes read from the FIFO is also popped from the FIFO.

On a read of this register, the values of the data flags, associated with the bytes read from the FIFO are stored in the Flag History Buffer.

**Configuration**

This register is present only when FE is implemented and M64BA_SPT. Otherwise, direct accesses to MFFCW<n>_PAY64 are RAZ/WI.

**Attributes**

MFFCW<n>_PAY64 is a 64-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_PAY64 bit assignments are:
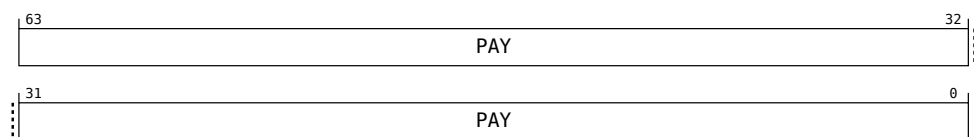


**PAY, bits [63:0]**

Payload read from FIFO

Eigth bytes are read from the FIFO, starting from the head of the FIFO.

A byte read from the FIFO can be either valid or invalid. A valid byte has an 8-bit data value and data flags associated with it. An invalid byte has an IMPLEMENTATION DEFINED data value and no data flags associated with it.

The data values of all bytes read from the FIFO, are arrange in the PAY field depending on the value of the MFFCW<n>_CTRL.MSBF field and the byte number.

**MFFCW<n>_CTRL.MSBF == 0b0**

Bytes are arranged starting with the first byte read from the FIFO in the LSB of the PAY field and the last byte read from the FIFO in the MSB of the PAY field.

**MFFCW<n>_CTRL.MSBF == 0b1**

Bytes are arranged starting with the first byte read from the FIFO in the MSB of the PAY field and the last byte read from the FIFO in the LSB of the PAY field.

The data flags of each byte are stored in the Flag History Buffer(FHB) starting with the first byte read in entry 0 of the FHB and the last byte read in entry 7 of the FHB.

With all other entries being marked as invalid.

If MFFCW<n>_CTRL.RA_EN is set to 0b1, when a byte is read from the FIFO:

- It is also removed from the FIFO.
- A Transfer Acknowledge event is generated if that byte was associated with the ACK and EOT data flags.

Otherwise the byte still remains in the FIFO and a subsequent read of the same size would return the same bytes.

**Accessing MFFCW<n>_PAY64**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_PAY64;
```

### C2.2.1.3.5   MFFCW<n>_FLG8, Mailbox FIFO Channel Window <n> Flag (8-bit access), n = 0 - 63

The MFFCW<n>_FLG8 characteristics are:

**Purpose**

An 8-bit access to the MFFCW<n>_FLG register. Accesses must be aligned to any 8-bit boundary within the MFFCW<n>_FLG register, otherwise the access is treated as RAZ/WI.

8-bit accesses are only supported, if MBX_FFCH_CFG0.M8BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If M64BA_SPT is set to 1 the MFFCW<n>_FLG register occupies offsets 0x08-0x0F within the Mailbox FIFO Channel Window <n>.

If M64BA_SPT is set to 0 the MFFCW<n>_FLG register occupies offsets 0x08-0x0B and offsets 0x0C-0x0F are reserved, within the Mailbox FIFO Channel Window <n>.

A read of this register returns:

- The contents of the Flag History Buffer.
- Current fill level of the FIFO.

This register is expected to be read after a read of the MFFCW<n>_PAY register to get the data flags associated with the bytes read from the FIFO. The read of this register must be of the same size as the read of the MFFCW<n>_PAY register, otherwise it can lead to loss or corruption of information.

**Configuration**

This register is present only when FE is implemented and M8BA_SPT. Otherwise, direct accesses to MFFCW<n>_FLG8 are RAZ/WI.

**Attributes**

MFFCW<n>_FLG8 is a 8-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_FLG8 bit assignments are:

```
 7         4 3 2 1 0
┌───────────┬─┬─┬───┐
│    FFL    │ │ │FLG0│
└───────────┴─┴─┴───┘
            RES0┘  └VFLG0
```
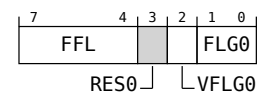
**FFL, bits [7:4]**

FIFO Fill Level

Indicates the number of bytes containing valid data in the FIFO.

| FFL | Meaning | Applies |
|---|---|---|
| 0b0000 | All bytes in the FIFO are invalid | |
| 0b0001..0b1110 | Number of valid bytes in the FIFO | |
| 0b1111 | 15 bytes are valid in the FIFO | When FFCH_DEPTH <= 15 |
| 0b1111 | 15 or more bytes are valid in the FIFO | When FFCH_DEPTH > 15 |

The maximum value returned is never greater than the FIFO Depth.

**Bit [3]**

Reserved, RES0.

**VFLG0, bit [2]**

Valid FLG0

Indicates whether FLG0 field contains valid data or not.

| VFLG0 | Meaning |
|-------|---------|
| 0b0 | FLG0 is not valid |
| 0b1 | FLG0 is valid |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**FLG0, bits [1:0]**

Flag 0

Provides the data flags, except for the ACK flag, held in FHB entry 0 and whether the data flags are valid or invalid.

The legal values of the FLG0 field are:

| FLG0 | Meaning |
|------|---------|
| 0b00 | Payload<br>Neither the first or last byte of a Transfer |
| 0b01 | Start Byte<br>First byte of a Transfer |
| 0b10 | End Byte<br>Last byte of a Transfer |
| 0b11 | Start and End Byte<br>First and last byte of a Transfer |

The value of the FLG0 field is only valid if VFLG0 is set to 0b1, otherwise the value of this field must be ignored by software.

**Accessing MFFCW<n>_FLG8**

Accesses to this register use the following encodings:

**Read at offset 0x008 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**Read at offset 0x009 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**Read at offset 0x00A + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**Read at offset 0x00B + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**When M64BA_SPT**

**Read at offset 0x00C + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**When M64BA_SPT**

**Read at offset 0x00D + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**When M64BA_SPT**

**Read at offset 0x00E + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

**When M64BA_SPT**

**Read at offset 0x00F + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG8;
```

### C2.2.1.3.6   MFFCW<n>_FLG16, Mailbox FIFO Channel Window <n> Flag (16-bit access), n = 0 - 63

The MFFCW<n>_FLG16 characteristics are:

**Purpose**

A 16-bit access to the MFFCW<n>_FLG register. Accesses must be aligned to any 16-bit boundary within the MFFCW<n>_FLG register, otherwise the access is treated as RAZ/WI.

16-bit accesses are only supported, if MBX_FFCH_CFG0.M16BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If M64BA_SPT is set to 1 the MFFCW<n>_FLG register occupies offsets 0x08-0x0F within the Mailbox FIFO Channel Window <n>.

If M64BA_SPT is set to 0 the MFFCW<n>_FLG register occupies offsets 0x08-0x0B and offsets 0x0C-0x0F are reserved, within the Mailbox FIFO Channel Window <n>.

A read of this register returns:

- The contents of the Flag History Buffer.

- Current fill level of the FIFO.

This register is expected to be read after a read of the MFFCW<n>_PAY register to get the data flags associated with the bytes read from the FIFO. The read of this register must be of the same size as the read of the MFFCW<n>_PAY register, otherwise it can lead to loss or corruption of information.

**Configuration**

This register is present only when FE is implemented and M16BA_SPT. Otherwise, direct accesses to MFFCW<n>_FLG16 are RAZ/WI.

**Attributes**

MFFCW<n>_FLG16 is a 16-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_FLG16 bit assignments are:
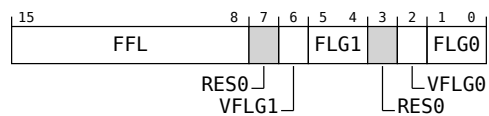


**FFL, bits [15:8]**

FIFO Fill Level

Indicates the number of bytes containing valid data in the FIFO.

| FFL | Meaning | Applies |
|---|---|---|
| 0x00 | All bytes in the FIFO are invalid | |
| 0x01..0xFE | Number of valid bytes in the FIFO | |
| 0xFF | 255 bytes are valid in the FIFO | When FFCH_DEPTH <= 255 |

| FFL | Meaning | Applies |
| --- | --- | --- |
| 0xFF | 255 or more bytes are valid in the FIFO | When FFCH_DEPTH > 255 |

The maximum value returned is never greater than the FIFO Depth.

**Bits [7, 3]**

Reserved, RES0.

**VFLG<m>, bits [2+4m], for m = 1 to 0**

Valid Flag <m>

Indicates whether FLG<m> field contains valid data or not.

| VFLG<m> | Meaning |
| --- | --- |
| 0b0 | FLG<m> is not valid |
| 0b1 | FLG<m> is valid |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**FLG<m>, bits [5:4, 1:0], for m = 1 to 0, where each field is 2 bits wide**

Flag <m>

Provides the data flags, except for the ACK flag, held in FHB entry 0 and 1 and whether the data flags are valid or invalid.

The association of the FLG<m> field to the FHB entries depends on the value of the MFFCW<n>_CTRL.MSBF field when the read of the MFFCW<n>_FLG register occurrs.

**MFFCW<n>_CTRL.MSBF == 0b0**

FLG0 is associatied with FHB entry 0 and FLG1 is associated with FHB entry 1

**MFFCW<n>_CTRL.MSBF == 0b1**

FLG0 is associatied with FHB entry 1 and FLG1 is associated with FHB entry 0

The legal values of the FLG<m> field, when VFLG<m> is set to 0b1 are:

| FLG<m> | Meaning |
| --- | --- |
| 0b00 | Payload<br>Neither the first or last byte of a Transfer |
| 0b01 | Start Byte<br>First byte of a Transfer |
| 0b10 | End Byte<br>Last byte of a Transfer |
| 0b11 | Start and End Byte<br>First and last byte of a Transfer |

The value of the FLG<m> field is only valid if VFLG<m> is set to 0b1, otherwise the value of this field must be ignored by software.

**Accessing MFFCW<n>_FLG16**

Accesses to this register use the following encodings:

**Read at offset 0x008 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG16;
```

**Read at offset 0x00A + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG16;
```

**When M64BA_SPT**

**Read at offset 0x00C + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG16;
```

**When M64BA_SPT**

**Read at offset 0x00E + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG16;
```

### C2.2.1.3.7  MFFCW<n>_FLG32, MailboxPostbox FIFO Channel Window <n> Flag (32-bit access), n = 0 - 63

The MFFCW<n>_FLG32 characteristics are:

**Purpose**

A 32-bit access to the MFFCW<n>_FLG register. Accesses must be aligned to any 32-bit boundary within the MFFCW<n>_FLG register, otherwise the access is treated as RAZ/WI.

32-bit accesses are only supported, if MBX_FFCH_CFG0.M32BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

If M64BA_SPT is set to 1 the MFFCW<n>_FLG register occupies offsets 0x08-0x0F within the Mailbox FIFO Channel Window <n>.

If M64BA_SPT is set to 0 the MFFCW<n>_FLG register occupies offsets 0x08-0x0B and offsets 0x0C-0x0F are reserved, within the Mailbox FIFO Channel Window <n>.

A read of this register returns:

- The contents of the Flag History Buffer.

- Current fill level of the FIFO.

This register is expected to be read after a read of the MFFCW<n>_PAY register to get the data flags associated with the bytes read from the FIFO. The read of this register must be of the same size as the read of the MFFCW<n>_PAY register, otherwise it can lead to loss or corruption of information.

**Configuration**

This register is present only when FE is implemented and M32BA_SPT. Otherwise, direct accesses to MFFCW<n>_FLG32 are RAZ/WI.

**Attributes**

MFFCW<n>_FLG32 is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_FLG32 bit assignments are:
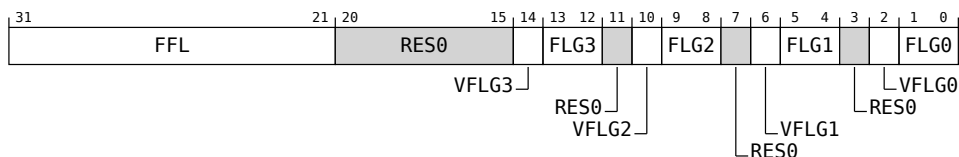


**FFL, bits [31:21]**

FIFO Fill Level

Indicates the number of bytes containing valid data in the FIFO.

| FFL | Meaning |
| --- | --- |
| 0b00000000000 | All bytes in the FIFO are invalid |
| 0b00000000001..0<br>↪b01111111111 | Number of valid bytes in the FIFO |

| FFL | Meaning |
|-----|---------|
| `0b10000000000` | 1024 or more bytes are valid in the FIFO |

The maximum value returned is never greater than the FIFO Depth.

**Bits [20:15, 11, 7, 3]**

Reserved, RES0.

**VFLG<m>, bits [2+4m], for m = 3 to 0**

Valid Flag <m>

Indicates whether FLG<m> field contains valid data or not.

| VFLG<m> | Meaning |
|---------|---------|
| `0b0` | FLG<m> is not valid |
| `0b1` | FLG<m> is valid |

The reset behavior of this field is:

- On a MHUR reset, this field resets to `0b0`.

**FLG<m>, bits [13:12, 9:8, 5:4, 1:0], for m = 3 to 0, where each field is 2 bits wide**

Flag <m>

Provides the data flags, except for the ACK flag, held in FHB entry 0 to 3 and whether the data flags are valid or invalid.

The association of the FLG<m> field to the FHB entries depends on the value of the MFFCW<n>_CTRL.MSBF field when the read of the MFFCW<n>_FLG register occurrs.

**MFFCW<n>_CTRL.MSBF == 0b0**

FLG0 is associatied with FHB entry 0 and FLG3 is associated with FHB entry 3

**MFFCW<n>_CTRL.MSBF == 0b1**

FLG0 is associatied with FHB entry 3 and FLG3 is associated with FHB entry 0

The legal values of the FLG<m> field, when VFLG<m> is set to 0b1 are:

| FLG<m> | Meaning |
|--------|---------|
| `0b00` | Payload<br>Neither the first or last byte of a Transfer |
| `0b01` | Start Byte<br>First byte of a Transfer |
| `0b10` | End Byte<br>Last byte of a Transfer |
| `0b11` | Start and End Byte<br>First and last byte of a Transfer |

The value of the FLG<m> field is only valid if VFLG<m> is set to 0b1, otherwise the value of this field must be ignored by software.

**Accessing MFFCW<n>_FLG32**

Accesses to this register use the following encodings:

**Read at offset 0x008 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG32;
```

**When M64BA_SPT**

**Read at offset 0x00C + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG32;
```

### C2.2.1.3.8 MFFCW\<n>_FLG64, Mailbox FIFO Channel Window \<n> Flag (64-bit access), n = 0 - 63

The MFFCW\<n>_FLG64 characteristics are:

**Purpose**

A 64-bit access to the MFFCW\<n>_FLG register. Accesses must be aligned to any 64-bit boundary within the MFFCW\<n>_FLG register, otherwise the access is treated as RAZ/WI.

64-bit accesses are only supported, if MBX_FFCH_CFG0.M64BA_SPT is set to 0b1, otherwise it is an unsupported access and it is IMPLEMENTATION DEFINED whether the access is treated as RAZ/WI or modified to a supported size.

When MBX_FFCH_CFG0.M64BA_SPT is set to 0b1, the MFFCW\<n>_FLG register occupies offsets 0x08-0x0F within the Mailbox FIFO Channel Window \<n>.

A read of this register returns:

- The contents of the Flag History Buffer.

- Current fill level of the FIFO.

This register is expected to be read after a read of the MFFCW\<n>_PAY register to get the data flags associated with the bytes read from the FIFO. The read of this register must be of the same size as the read of the MFFCW\<n>_PAY register, otherwise it can lead to loss or corruption of information.

**Configuration**

This register is present only when FE is implemented and M64BA_SPT. Otherwise, direct accesses to MFFCW\<n>_FLG64 are RAZ/WI.
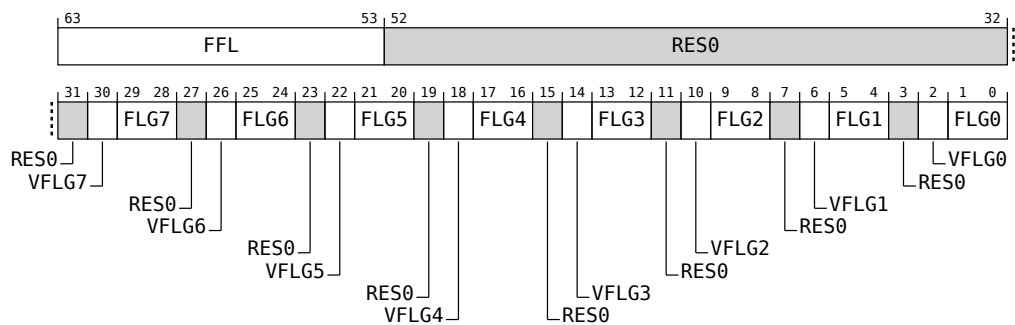
**Attributes**

MFFCW\<n>_FLG64 is a 64-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW\<n>_FLG64 bit assignments are:



**FFL, bits [63:53]**

FIFO Fill Level

Indicates the number of bytes containing valid data in the FIFO.

| FFL | Meaning |
| --- | --- |
| 0b00000000000 | All bytes in the FIFO are invalid |

| FFL | Meaning |
|-----|---------|
| `0b00000000001..0` <br> ↪`b01111111111` | Number of valid bytes in the FIFO |
| `0b10000000000` | 1024 or more bytes are valid in the FIFO |

The maximum value returned is never greater than the FIFO Depth.

**Bits [52:31, 27, 23, 19, 15, 11, 7, 3]**

Reserved, RES0.

**VFLG<m>, bits [2+4m], for m = 7 to 0**

Valid Flag <m>

Indicates whether FLG<m> field contains valid data or not.

| VFLG<m> | Meaning |
|---------|---------|
| `0b0` | FLG<m> is not valid |
| `0b1` | FLG<m> is valid |

The reset behavior of this field is:

  • On a MHUR reset, this field resets to `0b0`.

**FLG<m>, bits [29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0], for m = 7 to 0, where each field is 2 bits wide**

Flag <m>

Provides the data flags, except for the ACK flag, held in FHB entry 0 to 7 and whether the data flags are valid or invalid.

The association of the FLG<m> field to the FHB entries depends on the value of the MFFCW<n>_CTRL.MSBF field when the read of the MFFCW<n>_FLG register occurrs.

**MFFCW<n>_CTRL.MSBF == 0b0**

FLG0 is associatied with FHB entry 0 and FLG7 is associated with FHB entry 7

**MFFCW<n>_CTRL.MSBF == 0b1**

FLG0 is associatied with FHB entry 7 and FLG7 is associated with FHB entry 0

The legal values of the FLG<m> field, when VFLG<m> is set to 0b1 are:

| FLG<m> | Meaning |
|--------|---------|
| `0b00` | Payload <br> Neither the first or last byte of a Transfer |
| `0b01` | Start Byte <br> First byte of a Transfer |
| `0b10` | End Byte <br> Last byte of a Transfer |
| `0b11` | Start and End Byte <br> First and last byte of a Transfer |

The value of the FLG\<m\> field is only valid if VFLG\<m\> is set to 0b1, otherwise the value of this field must be ignored by software.

**Accessing MFFCW\<n\>_FLG64**

Accesses to this register use the following encodings:

**Read at offset 0x008 + (64 * n) from MHUR.MBX.MFFCW_page**

```
return MFFCW<n>_FLG64;
```

### C2.2.1.3.9 MFFCW<n>_INT_ST, Mailbox FIFO Channel Window <n> Interrupt Status, n = 0 - 63

The MFFCW<n>_INT_ST characteristics are:

**Purpose**

Provides the status of the interrupts for the FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW<n>_INT_ST are RAZ/WI.

**Attributes**

MFFCW<n>_INT_ST is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_INT_ST bit assignments are:
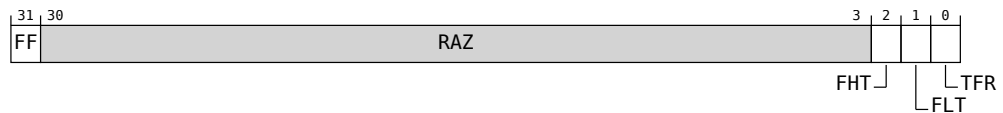


**FF, bit [31]**

FIFO Flush

Indicates whether the MHUS FIFO Flush mechanism has caused a flush of the FIFO for this FFCH.

| FF | Meaning |
| --- | --- |
| 0b0 | MHUS FIFO Flush mechanism has not caused a flush of the FIFO |
| 0b1 | MHUS FIFO Flush mechanism has caused a flush of the FIFO |

**Bits [30:3]**

Reserved, RAZ.

**FHT, bit [2]**

FIFO High Tidemark

Indicates whether the Receiver FIFO High Tidemark event has occurred.

| FHT | Meaning |
| --- | --- |
| 0b0 | Receiver FIFO High Tidemark has not occurred |
| 0b1 | Receiver FIFO High Tidemark event has occurred |

**FLT, bit [1]**

FIFO Low Tidemark

Indicates whether the Receiver FIFO Low Tidemark event has occurred.

| FLT | Meaning |
|-----|---------|
| 0b0 | Receiver FIFO Low Tidemark event has not occurred |
| 0b1 | Receiver FIFO Low Tidemark event has occured |

**TFR, bit [0]**

Transfer

Indicates whether a Transfer event has occurred.

| TFR | Meaning |
|-----|---------|
| 0b0 | Transfer event has not occurred |
| 0b1 | Transfer event has occurred |

**Accessing MFFCW<n>_INT_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0010 + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **RO**.

### C2.2.1.3.10 MFFCW&lt;n&gt;_INT_CLR, Mailbox FIFO Channel Window &lt;n&gt; Interrupt Clear, n = 0 - 63

The MFFCW&lt;n&gt;_INT_CLR characteristics are:

**Purpose**

Allows clearing of any interrupts for FFCH&lt;n&gt;.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW&lt;n&gt;_INT_CLR are RAZ/WI.

**Attributes**

MFFCW&lt;n&gt;_INT_CLR is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW&lt;n&gt;_INT_CLR bit assignments are:



Shows the status of the events of the DBCH

**FF, bit [31]**

FIFO Flush

| FF | Meaning |
|---|---|
| 0b0 | No effect |
| 0b1 | Clear FIFO flush event in the MFFCW&lt;n&gt;_INT_ST register |

**Bits [30:3]**

Reserved, RAZ.

**FHT, bit [2]**

FIFO High Tidemark

| FHT | Meaning |
|---|---|
| 0b0 | No effect |
| 0b1 | Clear FIFO High Tidemark event in the MFFCW&lt;n&gt;_INT_ST register |

**FLT, bit [1]**

FIFO Low Tidemark

| FLT | Meaning |
|-----|---------|
| 0b0 | No effect |
| 0b1 | Clear FIFO Low Tidemark event in the MFFCW<n>_INT_ST register |

**TFR, bit [0]**

Transfer

| TFR | Meaning |
|-----|---------|
| 0b0 | No effect |
| 0b1 | Clear Transfer event in the MFFCW<n>_INT_ST register |

**Accessing MFFCW<n>_INT_CLR**

Accesses to this register use the following encodings:

**Accessible at offset 0x0014 + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **WO/RAZ**.

### C2.2.1.3.11 MFFCW<n>_INT_EN, Mailbox FIFO Channel Window <n> Interrupt Enable, n = 0 - 63

The MFFCW<n>_INT_EN characteristics are:

**Purpose**

Configures the interrupts of FFCH<n>.

**Configuration**

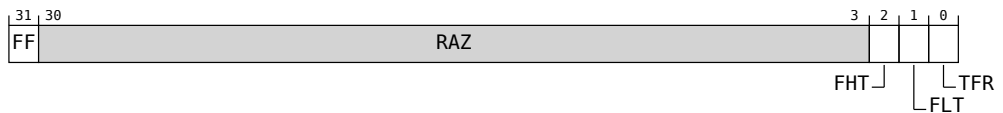This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW<n>_INT_EN are RAZ/WI.

**Attributes**

MFFCW<n>_INT_EN is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_INT_EN bit assignments are:
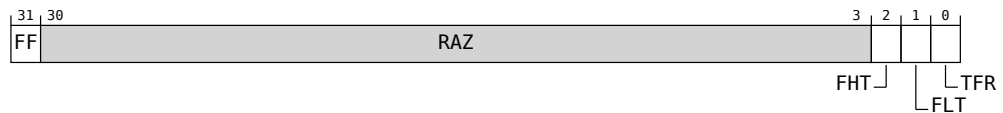


Shows the status of the events of the DBCH

**FF, bit [31]**

FIFO Flush

Controls whether a FIFO flush event generated by the MHUS generates an interrupt.

| FF | Meaning |
| --- | --- |
| 0b0 | A FIFO flush event from the MHUS does not generate an interrupt |
| 0b1 | A FIFO flush event from the MHUS generates an interrupt |

The reset behavior of this field is:

- On a MHUS reset, this field resets to 0b1.

**Bits [30:3]**

Reserved, RAZ.

**FHT, bit [2]**

FIFO High Tidemark

Controls whether a Receiver FIFO High Tidemark event generates an interrupt.

| FHT | Meaning |
| --- | --- |
| 0b0 | FIFO High Tidemark event does not generate an interrupt |

| FHT | Meaning |
|-----|---------|
| 0b1 | FIFO High Tidemark event generates an interrupt |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**FLT, bit [1]**

FIFO Low Tidemark

Controls whether a Receiver FIFO Low Tidemark event generates an interrupt.

| FLT | Meaning |
|-----|---------|
| 0b0 | FIFO Low Tidemark event has does not generate an interrupt |
| 0b1 | FIFO Low Tidemark event generates an interrupt |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**TFR, bit [0]**

Transfer

Controls whether a Transfer event generates an interrupt.

| TFR | Meaning |
|-----|---------|
| 0b0 | Transfer event does not generate an interrupt |
| 0b1 | Transfer event generates interrupt |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b1.

**Accessing MFFCW<n>_INT_EN**

Accesses to this register use the following encodings:

**Accessible at offset 0x0018 + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **RW**.

### C2.2.1.3.12   MFFCW<n>_CTRL, Mailbox FIFO Channel Window <n> Control, n = 0 - 63

The MFFCW<n>_CTRL characteristics are:

**Purpose**

Configures the behavior of FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW<n>_CTRL are RAZ/WI.

**Attributes**

MFFCW<n>_CTRL is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_CTRL bit assignments are:
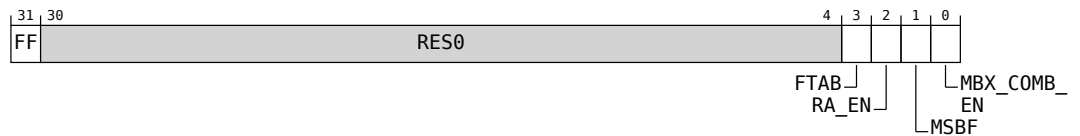


**FF, bit [31]**

FIFO Flush

Request a flush of the FFCH

| FF | Meaning |
| --- | --- |
| 0b0 | No request to flush the FIFO |
| 0b1 | Request to flush the FIFO |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**Bits [30:4]**

Reserved, RES0.

**FTAB, bit [3]**

Future Transfer Auto Buffering

| FTAB | Meaning |
| --- | --- |
| 0b0 | Future Transfer Auto Buffering is not enabled. The value of flags associated with bytes read from the FIFO have no effect on the number of bytes read or popped from the FIFO. |

| FTAB | Meaning |
|------|---------|
| 0b1 | Future Transfer Auto Buffering is enabled<br>When a read of the MFFCW<n>_PAY register with an access size greater than 1 occurs, any bytes read from the FIFO after the first byte with the EOT field set to 0b1 has been detected, are:<br>• Not read or popped from the FIFO.<br>• The bytes are set to an UNKNOWN value in the read data response.<br>• Entries in the Flag History Buffer are set to invalid.<br>Only bytes up to the first byte read from the FIFO with the EOT field set to 0b1 are popped from the FIFO. |

Must be used with the RA_EN field set to 0b1, otherwise the field is considered to be 0b0

The reset behavior of this field is:

• On a MHUR reset, this field resets to 0b0.

**RA_EN, bit [2]**

Controls whether Read to acknowledge is enabled.

| RA_EN | Meaning |
|-------|---------|
| 0b0 | Read acknowledge is not enabled<br>Bytes are popped from the FIFO by writing to MFFCW<n>_FIFO_POP register.<br>THe number of bytes popped from the FIFO is determined by the value written to the MFFCW<n>_FIFO_POP register. |
| 0b1 | Read acknowledge is enabled<br>Bytes are popped from the FIFO by reading the MFFCW<n>_PAY register.<br>The number of bytes popped from the FIFO is determined by the:<br>• Size of the access.<br>• FTAB field value.<br>• Value of the EOT flag for a byte which is popped from the FIFO due to this read, when the FTAB field is set 0b1. |

The reset behavior of this field is:

• On a MHUR reset, this field resets to 0b0.

**MSBF, bit [1]**

Most Significant Byte First

Selects the order in which bytes are read from the FIFO when multiple bytes are to be read due to a single read of the MFFCW<n>_PAY register.

| MSBF | Meaning |
|------|---------|
| 0b0 | Least Significant Byte first |
| 0b1 | Most Significant Byte first |

FFCH are considered little endian and the LSB is the byte lowest offset within the access and the MSB is the highest byte offset within the access.

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**MBX_COMB_EN, bit [0]**

Mailbox Combined Enable

Controls whether interrupts from the FFCH contribute to the Mailbox Combined interrupt.

| MBX_COMB_EN | Meaning |
|-------------|---------|
| 0b0 | FFCH interrupts does not contribute to the Mailbox Combined interrupt |
| 0b1 | FFCH interrupts contributes to the Mailbox Combined interrupt |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b1.

**Accessing MFFCW<n>_CTRL**

Accesses to this register use the following encodings:

**Accessible at offset 0x0020 + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **RW**.

### C2.2.1.3.13   MFFCW&lt;n&gt;_ST, Mailbox FIFO Channel Window &lt;n&gt; Status, n = 0 - 63

The MFFCW&lt;n&gt;_ST characteristics are:

**Purpose**

> Provides the status of the FFCH&lt;n&gt;.

**Configuration**

> This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW&lt;n&gt;_ST
> are RAZ/WI.

**Attributes**

> MFFCW&lt;n&gt;_ST is a 32-bit register.

> This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW&lt;n&gt;_ST bit assignments are:

| 31 | 30          11 | 10          0 |
|----|----------------|---------------|
| FF | RES0 | FFL |

**FF, bit [31]**

FIFO Flush

Status of MHUR FIFO Flush mechanism for the FFCH.

| FF | Meaning |
|------|---------|
| 0b0 | The meaning of this value depends on the value of the FF field in the corresponding MFFCW&lt;n&gt;_CTRL register. **When MFFCW&lt;n&gt;_CTRL.FF is 0b0** Receiver FIFO flush mechanism is idle. **When MFFCW&lt;n&gt;_CTRL;.FF is 0b1** Receiver FIFO flush mechanism is performing a flush. |
| 0b1 | FIFO flush complete |

The reset behavior of this field is:

> • On a MHUR reset, this field resets to 0b0.

**Bits [30:11]**

Reserved, RES0.

**FFL, bits [10:0]**

FIFO Fill Level

Indicates the number of valid bytes in the FIFO.

The maximum value returned is never greater than the FIFO Depth.

**Accessing MFFCW&lt;n&gt;_ST**

Accesses to this register use the following encodings:

**Accessible at offset 0x0024 + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **RO**.

### C2.2.1.3.14 MFFCW<n>_FIFO_POP, Mailbox FIFO Channel Window <n> FIFO POP, n = 0 - 63

The MFFCW<n>_FIFO_POP characteristics are:

**Purpose**

Request bytes to be popped from the FIFO of FFCH<n>.

**Configuration**

This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW<n>_FIFO_POP are RAZ/WI.
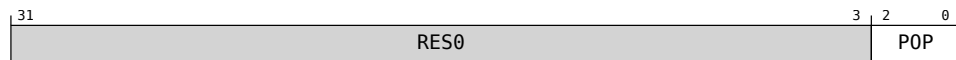
**Attributes**

MFFCW<n>_FIFO_POP is a 32-bit register.

This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_FIFO_POP bit assignments are:

| 31 | 3 | 2 | 0 |
|---|---|---|---|
| RES0 | | POP | |

**Bits [31:3]**

Reserved, RES0.

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

Number of bytes to pop from the FIFO.

This is the maximum number of bytes popped from the FIFO.

The number of bytes actually popped from the FIFO is as follows:

- When MFFCW<n>_CTRL.RA_EN is set to 0b1 no bytes are popped from the FIFO.

- When MFFCW<n>_CTRL.RA_EN is set to 0b0 the number of bytes is the smallest of the following:

    - Value written to this field.

    - Number of valid bytes in the FIFO.

| POP | Meaning | Applies |
|---|---|---|
| 0b000 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | When !M8BA_SPT |
| 0b000 | Maximum of 1 byte is popped from the FIFO | When M8BA_SPT |
| 0b001 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | When !M16BA_SPT |
| 0b001 | Maximum of 2 bytes are popped from the FIFO | When M16BA_SPT |

| POP | Meaning | Applies |
|---|---|---|
| 0b010 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | |
| 0b011 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | When !M32BA_SPT |
| 0b011 | Maximum of 4 bytes are popped from the FIFO | When M32BA_SPT |
| 0b100 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | |
| 0b101 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | |
| 0b110 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | |
| 0b111 | It is IMPLEMENTATION DEFINED whether:<br>• No bytes are popped from the FIFO.<br>• The value is treated as another supported value. | When !M64BA_SPT |
| 0b111 | Maximum of 8 bytes are popped from the FIFO | When M64BA_SPT |

**Accessing MFFCW<n>_FIFO_POP**

Accesses to this register use the following encodings:

**Accessible at offset 0x0028 + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **WO/RAZ**.

### C2.2.1.3.15   MFFCW<n>_TIDE, Mailbox FIFO Channel Window <n> Tidemark, n = 0 - 63

The MFFCW<n>_TIDE characteristics are:

**Purpose**

> Allows configuration of the low and high tidemark thresholds for the Receiver tidemark events

**Configuration**

> This register is present only when FE is implemented. Otherwise, direct accesses to MFFCW<n>_TIDE are RAZ/WI.
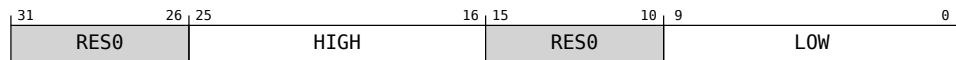
**Attributes**

> MFFCW<n>_TIDE is a 32-bit register.

> This register is part of the MHUR.MBX.MFFCW_page block.

**Field descriptions**

The MFFCW<n>_TIDE bit assignments are:

| 31    26 | 25          16 | 15    10 | 9          0 |
|----------|----------------|----------|--------------|
| RES0     | HIGH           | RES0     | LOW          |

**Bits [31:26]**

Reserved, RES0.

**HIGH, bits [25:16]**

High Tide Mark

Threshold value used in the generation of the Receiver FIFO High Tide event.

The event is generated when a push to the FIFO occurs, and the following are both true:

- The FIFO fill level before the push was less than or equal to the value of this field.

- The FIFO fill level after the push is greater than the value of this field.

| HIGH | Meaning |
|------|---------|
| 0b0000000000..0 ↪b1111111111 | Threshold value in bytes |

The upper and lower offset of this field depend on the configuration of the MHU.

The upper offset of this field is equal to clog2(FFCH_DEPTH)+15.

The lower offset of this field depends on the supported access sizes to MFFCW<n>_PAY register as follows:

- When MBX_FFCH_CFG.M8BA_SPT is 0b1 the lower offset is 16.

- When MBX_FFCH_CFG.M8BA_SPT is 0b0 and MBX_FFCH_CFG.M16BA_SPT is 0b1 the lower offset is 17.

- When MBX_FFCH_CFG.M{8/16}BA_SPT are both 0b0 and MBX_FFCH_CFG.M32BA_SPT is 0b1 the lower offset is 18.

- When MBX_FFCH_CFG.M{8/16/32}BA_SPT are all 0b0 and MBX_FFCH_CFG.M64BA_SPT is 0b1 the lower offset is 19.

Any offsets above the upper offset or below the lower offset are RES0.

If the upper offset is less than the lower offset the entire field is RES0.

For calculations of the Receiver High Tide event all offsets which are RES0 are used.

The reset behavior of this field is:

- On a MHUR reset, the expression `(FFCH_DEPTH - 1)[u:l]`.

Where:

u = tide_upr()

l = tide_lwr([M64BA_SPT,M32BA_SPT,M16BA_SPT,M8BA_SPT])

**Bits [15:10]**

Reserved, RES0.

**LOW, bits [9:0]**

Low Tide Mark

Threshold value used in the generation of the Receiver FIFO Low Tide event.

The event is generated when a pop to the FIFO occurs, and the following are both true:

- The FIFO fill level before the pop was greater than the value of this field.
- The FIFO fill level after the pop is less than or equal the value of this field.

| LOW | Meaning |
|-----|---------|
| 0b0000000000..0 ↪b1111111111 | Threshold value in bytes |

The upper and lower offset of this field depend on the configuration of the MHU.

The upper offset of this field is equal to clog2(FFCH_DEPTH)-1.

The lower offset of this field depends on the supported access sizes to MFFCW<n>_PAY register as follows:

- When MBX_FFCH_CFG.M8BA_SPT is 0b1 the lower offset is 0.
- When MBX_FFCH_CFG.M8BA_SPT is 0b0 and MBX_FFCH_CFG.M16BA_SPT is 0b1 the lower offset is 1.
- When MBX_FFCH_CFG.M{8/16}BA_SPT are both 0b0 and MBX_FFCH_CFG.M32BA_SPT is 0b1 the lower offset is 2.
- When MBX_FFCH_CFG.M{8/16/32}BA_SPT are all 0b0 and MBX_FFCH_CFG.M64BA_SPT is 0b1 the lower offset is 3.

Any offsets above the upper offset or below the lower offset are RES0.

If the upper offset is less than the lower offset the entire field is RES0.

For calculations of the Receiver Low Tide event all offsets which are RES0 are used.

The reset behavior of this field is:

- On a MHUR reset, this field resets to `0b0000000000`.

**Accessing MFFCW<n>_TIDE**

Accesses to this register use the following encodings:

**Accessible at offset 0x002C + (64 * n) from MHUR.MBX.MFFCW_page**

Access on this interface is **RW**.

### C2.2.1.4  MFCW_page, Mailbox Fast Channel Windows Page

The MFCW_page characteristics are:

**Purpose**

Allows access to the MFCW.

Depending on the Fast Channel word-size either the MFCW<n>_PAY32 or MFCW<n>_PAY64 registers are implmented.

**When the Fast Channel word-size is 32-bit**

MFCW<n>_PAY32 is implemented

There can be between 1 and 1024 MFCWs.

**When the Fast Channel word-size is 64-bit**

MFCW<n>_PAY64 is implemented

There can be between 1 and 512 MFCWs.

**Configuration**

This Register Block is present only when FCE is implemented.

**Attributes**

The MFCW_page block is of size 4KB.

This block is part of the MHUR.MBX block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x000 + (4 * n)`for n in ↪`1023:0` | MFCW<n>_PAY32 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x20<br>Accessor condition: When FCH_WS == 0x20 |
| `0x000 + (4 * n)`for n in ↪`1023:0` | MFCW<n>_PAY32 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x20<br>Accessor condition: When FCH_WS == 0x20 |
| `0x000 + (8 * n)`for n in ↪`511:0` | MFCW<n>_PAY64 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x40<br>Accessor condition: When FCH_WS == 0x40 |
| `0x000 + (8 * n)`for n in ↪`511:0` | MFCW<n>_PAY64 | Most permissive access: RW<br>Register condition: When FCH_WS == 0x40<br>Accessor condition: When FCH_WS == 0x40 |

### C2.2.1.4.1   MFCW\<n\>_PAY32, Mailbox Fast Channel Window \<n\> Payload 32-bit, n = 0 - 1023

The MFCW\<n\>_PAY32 characteristics are:

**Purpose**

Access to payload of FCH\<n\>

Arm recommends that accesses to these registers are atomic.

---

**Note**

This is the 32-bit version of the MFCW\<n\>_PAY registers

---

**Configuration**

This register is present only when FCE is implemented and FCH_WS == 0x20. Otherwise, direct accesses to MFCW\<n\>_PAY32 are RAZ/WI.

**Attributes**

MFCW\<n\>_PAY32 is a 32-bit register.

This register is part of the MHUR.MBX.MFCW_page block.

**Field descriptions**

The MFCW\<n\>_PAY32 bit assignments are:
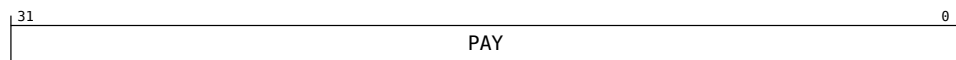
| 31                                                    0 |
|---|
| PAY |

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

Payload for Channel \<n\>

A write to this register sets the value of the payload.

A read to this register:

- Returns the current value of the payload.
- Acknowledges the Transfer.
- Clears the Transfer interrupt for the FCH, if it is set.

The reset behavior of this field is:

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

**Accessing MFCW\<n\>_PAY32**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (4 * n) from MHUR.MBX.MFCW_page**

```
return MFCW<n>_PAY32;
```

**Write at offset 0x000 + (4 * n) from MHUR.MBX.MFCW_page**

```
MFCW<n>_PAY32 = value;
```

### C2.2.1.4.2  MFCW<n>_PAY64, Mailbox Fast Channel Window <n> Payload 64-bit, n = 0 - 511

The MFCW<n>_PAY64 characteristics are:

**Purpose**

Access to payload of FCH<n>

Arm recommends that accesses to these registers are atomic.

---
**Note**

This is the 64-bit version of the MFCW<n>_PAY registers

---

**Configuration**

This register is present only when FCE is implemented and FCH_WS == 0x40. Otherwise, direct accesses to MFCW<n>_PAY64 are RAZ/WI.

**Attributes**

MFCW<n>_PAY64 is a 64-bit register.

This register is part of the MHUR.MBX.MFCW_page block.

**Field descriptions**

The MFCW<n>_PAY64 bit assignments are:
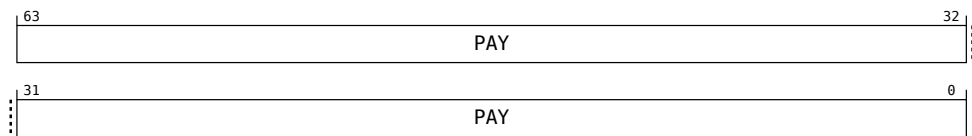
| 63 | PAY | 32 |
|----|-----|----|

| 31 | PAY | 0 |
|----|-----|---|

**PAY, bits [63:0]**

Payload for Channel <n>

A write to this register sets the value of the payload.

A read to this register:

- Returns the current value of the payload.

- Acknowledges the Transfer.

- Clears the Transfer interrupt for the FCH, if it is set.

The reset behavior of this field is:

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

**Accessing MFCW<n>_PAY64**

Accesses to this register use the following encodings:

**Read at offset 0x000 + (8 * n) from MHUR.MBX.MFCW_page**

```
return MFCW<n>_PAY64;
```

**Write at offset 0x000 + (8 * n) from MHUR.MBX.MFCW_page**

```
MFCW<n>_PAY64 = value;
```

### C2.2.1.5 MBX_IMPL_DEF_page, Mailbox Implementation Defined page

The MBX_IMPL_DEF_page characteristics are:

**Purpose**

> The MBX_IMPL_DEF_page is for an implementation of the MHU, to implement any IMPLEMENTA-TION DEFINED registers.

> Registers defined in this block must follow the rules in  C1.7.3 *IMPLEMENTATION DEFINED Page*

**Configuration**

**Attributes**

> The MBX_IMPL_DEF_page block is of size 4KB.

> This block is part of the MHUR.MBX block.

**Default access**

> Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x0000 + (4 * n)for n in 1023:0` | - | Most permissive access: ImplementationDefined |

## C2.2.2  RSC, Receiver Security Control

The RSC characteristics are:

**Purpose**

Contains the individual pages of the Receiver Security Control block.

Only accesses with the correct security can access the registers within the Receiver Security Control block, otherwise the access is treated as an illegal access.

The allowed security of an access to a register of the Receiver Security Control block depends on:

- Whether RME is implemented for the MHUR.
- Sampled value of MHUR **LEGACY_TZ_EN** input.

The following table list the allowed security states of an access:

| RME | TZE | **LEGACY_TZ_EN** | Allowed access security |
|-----|-----|------------------|-------------------------|
| 0 | 0 | NA | NA |
| 0 | 1 | NA | Secure |
| 1 | 1 | 0 | Root |
|   |   | 1 | Secure |

**Configuration**

This Register Block is present only when TZE is implemented for the MHUR.

**Attributes**

The RSC block is of size 64KB.

This block is part of the MHUR block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|--------|------|-------|
| `0x0000` | RSC_CTRL_page | Most permissive access: RW |
| `0xF000` | RSC_IMPL_DEF_page | Most permissive access: RW |

### C2.2.2.1   RSC_CTRL_page, Receiver Security Control Page

The RSC_CTRL_page characteristics are:

**Purpose**

Allow assignment of resources of the MHUR to a Security Group.

**Configuration**

This Register Block is present only when TZE is implemented for the MHUR.

**Attributes**

The RSC_CTRL_page block is of size 4KB.

This block is part of the MHUR.RSC block.

**Default access**

Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x000` | RSC_BLK_ID | Most permissive access: RO |
| `0x010` | RSC_FEAT_SPT0 | Most permissive access: RO |
| `0x014` | RSC_FEAT_SPT1 | Most permissive access: RO |
| `0x020` | RSC_DBCH_CFG0 | Most permissive access: RO<br>Register condition: When DBE is implemented<br>Accessor condition: When DBE is implemented |
| `0x030` | RSC_FFCH_CFG0 | Most permissive access: RO<br>Register condition: When FE is implemented<br>Accessor condition: When FE is implemented |
| `0x040` | RSC_FCH_CFG0 | Most permissive access: RO<br>Register condition: When FCE is implemented<br>Accessor condition: When FCE is implemented |
| `0x110 + (4 * n)for n in 0` | RSC_MBX_SG<n> | Most permissive access: RW |
| `0xFC8` | RSC_IIDR | Most permissive access: RO |
| `0xFCC` | RSC_AIDR | Most permissive access: RO |
| `0xFD0 + (4 * n)for n in 11:0` | RSC_IMPL_DEF_ID<n> | Most permissive access: RO |

### C2.2.2.1.1 RSC_BLK_ID, Receiver Security Block Identifier

The RSC_BLK_ID characteristics are:

**Purpose**

Identifies the block as a Receiver Security.

**Configuration**

This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_BLK_ID are RAZ/WI.

**Attributes**

RSC_BLK_ID is a 32-bit register.

This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_BLK_ID bit assignments are:
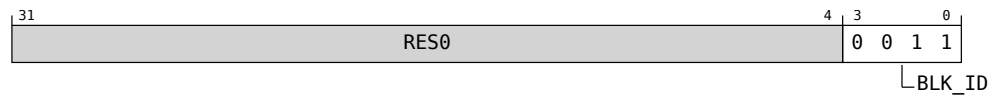


**Bits [31:4]**

Reserved, RES0.

**BLK_ID, bits [3:0]**

Block Identifier

Identifies the type of block that resides in this 64KB.

Reads as 0b0011

Identifies the block as the Receiver Security block.

Access to this field is **RO**.

**Accessing RSC_BLK_ID**

Accesses to this register use the following encodings:

**Accessible at offset 0x000 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.2   RSC_FEAT_SPT0, Receiver Security Feature Support 0

The RSC_FEAT_SPT0 characteristics are:

**Purpose**

> Provides information on the features implemented in the Receiver Security.

**Configuration**

> This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_FEAT_SPT0 are RAZ/WI.

**Attributes**

> RSC_FEAT_SPT0 is a 32-bit register.

> This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_FEAT_SPT0 bit assignments are:

| 31 24 | 23 20 | 19 16 | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|---|---|
| RES0 | RASE_SPT | RME_SPT | TZE_SPT | FCE_SPT | FE_SPT | DBE_SPT |

**Bits [31:24]**

Reserved, RES0.

**RASE_SPT, bits [23:20]**

Reliability, Availability and Serviceability Extension Support

The value of this field depends on the implementation of the MHU.

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

| RASE_SPT | Meaning |
|---|---|
| 0b0000 | MHU does not implement the RAS extension |
| 0b0010 | MHU implements the RAS extension but does not follow the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |
| 0b0011 | MHU implements the RAS extension and follows the recommendations in B10.7 *Recommend implementation of RAS using Arm RAS extensions* |

Access to this field is **RO**.

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

Realm Management Extension Support

The value of this field depends on the implementation of the MHU and an optional reset time sampled input **LEGACY_TZ_EN**.

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

| RME_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Realm Management extension |
| 0b0001 | MHU implements Realm Management extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

When RME is implemented, for the half of the MHU, there can be a **LEGACY_TZ_EN** tie-off present on that side of the MHU.

The value of the **LEGACY_TZ_EN** tie-off is sampled at reset of the side of the MHU which the tie-off is associated with.

When the sampled value of the tie-off is 0b1 the value of this field is always 0x0, otherwise the value of this field is dependent on whether RME is implemented for this half of the MHU.

Access to this field is **RO**.

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

TrustZone Extension Support

The value of this field depends on the implementation of the MHU.

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

| TZE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the TrustZone extension |
| 0b0001 | MHU implements TrustZone extension |

The value of this field only applies to the half of the MHU which the register is associated with.

It is valid for the different halves of the MHU to implement different values for this field.

For fields in the PBX_FEAT_SPT0 or SSC_FEAT_SPT0 the value applies to the MHUS only.

For fields in the MBX_FEAT_SPT0 or RSC_FEAT_SPT0 the value applies to the MHUR only.

Access to this field is **RO**.

**FCE_SPT, bits [11:8]**

Fast Channel Extension Support

The value of this field depends on the implementation of the MHU.

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

| FCE_SPT | Meaning |
|---------|---------|
| 0b0000 | MHU does not implement the Fast Channel extension |

| FCE_SPT | Meaning |
|---|---|
| 0b0001 | MHU implements Fast Channel extension |

Access to this field is **RO**.

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

FIFO Extension Support

The value of this field depends on the implementation of the MHU.

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

| FE_SPT | Meaning |
|---|---|
| 0b0000 | MHU does not implement the FIFO extension |
| 0b0001 | MHU implements FIFO extension |

Access to this field is **RO**.

**DBE_SPT, bits [3:0]**

Doorbell Extension Support

The value of this field depends on the implementation of the MHU.

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

| DBE_SPT | Meaning |
|---|---|
| 0b0000 | MHU does not implement the Doorbell extension |
| 0b0001 | MHU implements Doorbell extension |

Access to this field is **RO**.

**Accessing RSC_FEAT_SPT0**

Accesses to this register use the following encodings:

**Accessible at offset 0x010 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.3 RSC_FEAT_SPT1, Receiver Security Feature Support 1

The RSC_FEAT_SPT1 characteristics are:

**Purpose**

> Provides information on the features implemented in the Receiver Security.

**Configuration**

> This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_FEAT_SPT1 are RAZ/WI.
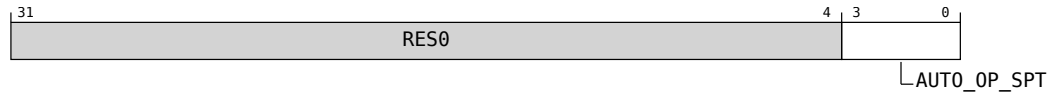
**Attributes**

> RSC_FEAT_SPT1 is a 32-bit register.

> This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_FEAT_SPT1 bit assignments are:



**Bits [31:4]**

Reserved, RES0.

**AUTO_OP_SPT, bits [3:0]**

Auto Op Protocol Support

The value of this field depends on the implementation of the MHU.

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

| AUTO_OP_SPT | Meaning |
|---|---|
| 0b0000 | Auto Op(Min) is implemented |
| 0b0001 | Auto Op(Full) is implemented |

Access to this field is **RO**.

**Accessing RSC_FEAT_SPT1**

Accesses to this register use the following encodings:

**Accessible at offset 0x014 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.4 RSC_DBCH_CFG0, Receiver Security Doorbell Channel Configuration 0

The RSC_DBCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of DBE in MHUR.

**Configuration**

This register is present only when TZE is implemented for the MHUR and DBE is implemented. Otherwise, direct accesses to RSC_DBCH_CFG0 are RAZ/WI.

**Attributes**

RSC_DBCH_CFG0 is a 32-bit register.

This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_DBCH_CFG0 bit assignments are:

| 31 8 | 7 0 |
|---|---|
| RES0 | NUM_DBCH |

**Bits [31:8]**

Reserved, RES0.

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

Number of Doorbell Channels

Number of DBCH implemented in the MHUR.

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

| NUM_DBCH | Meaning |
|---|---|
| 0x00..0x7F | Number of DBCH is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing RSC_DBCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x020 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.5 RSC_FFCH_CFG0, Receiver Security FIFO Channel Configuration 0

The RSC_FFCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FE in Receiver Security.

**Configuration**

This register is present only when TZE is implemented for the MHUR and FE is implemented. Otherwise, direct accesses to RSC_FFCH_CFG0 are RAZ/WI.
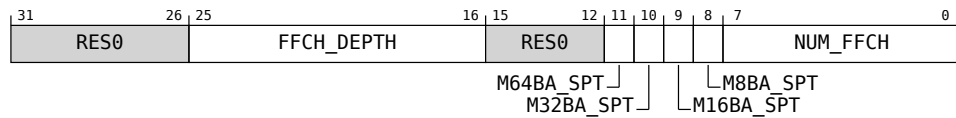
**Attributes**

RSC_FFCH_CFG0 is a 32-bit register.

This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_FFCH_CFG0 bit assignments are:



**Bits [31:26]**

Reserved, RES0.

**FFCH_DEPTH, bits [25:16]**

FIFO Channel Depth

Depth of the FIFOs of the FFCHs in the MHUR.

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

| FFCH_DEPTH | Meaning |
| --- | --- |
| 0b0000000000..0 ↪b1111111111 | FIFO depth is N+1 bytes, where N is the value of this field. |

Access to this field is **RO**.

**Bits [15:12]**

Reserved, RES0.

**M64BA_SPT, bit [11]**

Mailbox 64-bit Access Support

Whether 64-bit accesses to the following registers are supported:

- MFFCW<n>_PAY
- MFFCW<n>_FLG

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

| M64BA_SPT | Meaning |
|---|---|
| 0b0 | 64-bit accesses are not supported |
| 0b1 | 64-bit accesses are supported |

Accesses must be aligned to an 64-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

When 64-bit accesses are supported the:

- MFFCW<n>_PAY register occupies offsets 0x00-0x07 of the MFFCW.

- MFFCW<n>_FLG register occupies offsets 0x008-0x0F of the MFFCW.

When 64-bit accesses are not supported:

- MFFCW<n>_PAY register occupies offsets 0x00-0x03 of the MFFCW and offsets 0x04-0x07 of the MFFCW are RES0.

- MFFCW<n>_FLG register occupies offsets 0x008-0x0B of the MFFCW and offsets 0x0C-0x0F of the MFFCW are RES0.

Access to this field is **RO**.

**M32BA_SPT, bit [10]**

Mailbox 32-bit Access Support

Whether 32-bit accesses to the following registers are supported:

- MFFCW<n>_PAY

- MFFCW<n>_FLG

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

| M32BA_SPT | Meaning |
|---|---|
| 0b0 | 32-bit accesses are not supported |
| 0b1 | 32-bit accesses are supported |

Accesses must be aligned to an 32-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**M16BA_SPT, bit [9]**

Mailbox 16-bit Access Support

Whether 16-bit accesses to the following registers are supported:

- MFFCW<n>_PAY

- MFFCW<n>_FLG

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

| M16BA_SPT | Meaning |
|-----------|---------|
| 0b0 | 16-bit accesses are not supported |
| 0b1 | 16-bit accesses are supported |

Accesses must be aligned to an 16-bit boundary.

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

**M8BA_SPT, bit [8]**

Mailbox 8-bit Access Support

Whether 8-bit accesses to the following registers are supported:

- MFFCW<n>_PAY
- MFFCW<n>_FLG

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

| M8BA_SPT | Meaning |
|----------|---------|
| 0b0 | 8-bit accesses are not supported |
| 0b1 | 8-bit accesses are supported |

Accesses must be aligned to an 8-bit boundary

The value of this field has no effect on the supported access sizes to any other registers.

Access to this field is **RO**.

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

Number of FIFO Channels

The number of FFCHs in the MHUR.

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

| NUM_FFCH | Meaning |
|----------|---------|
| 0x00..0x3F | Number of FFCHs is N+1, where N is the value of this field |

Access to this field is **RO**.

**Accessing RSC_FFCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x030 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.6 RSC_FCH_CFG0, Receiver Security Fast Channel Configuration 0

The RSC_FCH_CFG0 characteristics are:

**Purpose**

Provides information on the configuration of FCE in the MHUR.

**Configuration**

This register is present only when TZE is implemented for the MHUR and FCE is implemented. Otherwise, direct accesses to RSC_FCH_CFG0 are RAZ/WI.
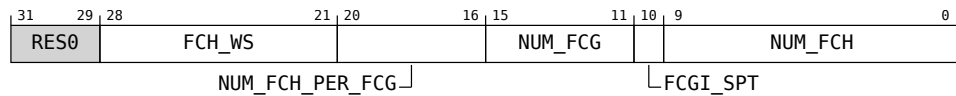
**Attributes**

RSC_FCH_CFG0 is a 32-bit register.

This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_FCH_CFG0 bit assignments are:



**Bits [31:29]**

Reserved, RES0.

**FCH_WS, bits [28:21]**

Fast Channel Word-Size

Number of bits each FCH implements.

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

| FCH_WS | Meaning |
| --- | --- |
| 0x20 | Fast Channel word-size is 32-bits |
| 0x40 | Fast Channel word-size is 64-bits |

Access to this field is **RO**.

**NUM_FCH_PER_FCG, bits [20:16]**

Number of Fast Channels per Fast Channel Group

Number of FCHs implemented in FCGs for MHUR.

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

| NUM_FCH_PER_FCG | Meaning |
| --- | --- |
| 0b00000..0b11111 | Number of FCHs per FCG is N+1, where N is the value of this field |

Access to this field is **RO**.

**NUM_FCG, bits [15:11]**

Number of Fast Channel Groups

Number of FCGs implemented in MHUR

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

| NUM_FCG | Meaning |
|---|---|
| 0b00000..0 ↪b11111 | Number of FCGs is N+1, where N is the value of this field |

Access to this field is **RO**.

**FCGI_SPT, bit [10]**

Fast Channel Group Interrupt Support

Indicates whether Fast Channel Group Transfer interrupt is implemented for each FCGs.

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

| FCGI_SPT | Meaning |
|---|---|
| 0b0 | Fast Channel Group Transfer interrupts are not implemented. |
| 0b1 | Fast Channel Group Transfer interrupts are implemented. |

Access to this field is **RO**.

**NUM_FCH, bits [9:0]**

Number of Fast Channels

Number of FCHs implemented in MHUR.

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

| NUM_FCH | Meaning | Applies |
|---|---|---|
| 0b0000000000..0 ↪b0111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x40 |
| 0b0000000000..0 ↪b1111111111 | Number of FCH is N+1, where N is the value of this field | When FCH_WS == 0x20 |

Access to this field is **RO**.

**Accessing RSC_FCH_CFG0**

Accesses to this register use the following encodings:

**Accessible at offset 0x040 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.7 RSC_MBX_SG<n>, Receiver Mailbox Security Group <n>, n = 0 - 0

The RSC_MBX_SG<n> characteristics are:

**Purpose**

Configuration of the Security Group of the Mailbox

**Configuration**

This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_MBX_SG<n> are RAZ/WI.
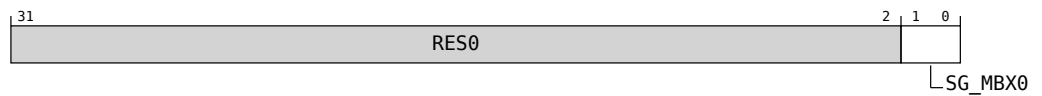
**Attributes**

RSC_MBX_SG<n> is a 32-bit register.

This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_MBX_SG<n> bit assignments are:

**When RME is implemented for the MHUR and sampled value of MHUR LEGACY_TZ_EN is 0b0:**



**Bits [31:2]**

Reserved, RES0.
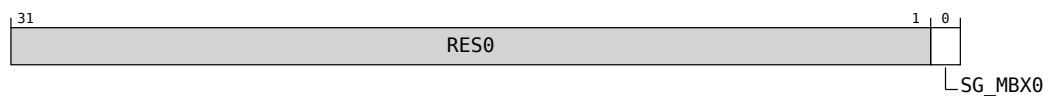
**SG_MBX<m>, bits [2m+1:2m], for m = 0**

Security Group for Mailbox

| SG_MBX<m> | Meaning |
| --- | --- |
| 0b00 | Mailbox is assigned to the Secure Security Group |
| 0b01 | Mailbox is assigned to the Non-secure Security Group |
| 0b10 | Mailbox is assigned to the Root Security Group |
| 0b11 | Mailbox is assigned to the Realm Security Group |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b10.

**When RME is not implemented for the MHUR or (RME is implemented for the MHUR and sampled value of MHUR LEGACY_TZ_EN is 0b1):**



**Bits [31:1]**

Reserved, RES0.

**SG_MBX<m>, bits [m], for m = 0**

Security Group for Mailbox

| SG_MBX<m> | Meaning |
|---|---|
| 0b0 | Mailbox is assigned to the Secure Security Group |
| 0b1 | Mailbox is assigned to the Non-secure Security Group |

The reset behavior of this field is:

- On a MHUR reset, this field resets to 0b0.

**Accessing RSC_MBX_SG<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0x110 + (4 * n) from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RW**.

### C2.2.2.1.8   RSC_IIDR, Receiver Security Implementer Identification Register

The RSC_IIDR characteristics are:

**Purpose**

> This field provides information on the implementation of the MHU

**Configuration**

> This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_IIDR are RAZ/WI.

**Attributes**

> RSC_IIDR is a 32-bit register.

> This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_IIDR bit assignments are:

| 31                     20 | 19      16 | 15    12 | 11                       0 |
|---------------------------|------------|----------|----------------------------|
| PRODUCT_ID                | VARIANT    | REVISION | IMPLEMENTER                |

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

Product ID of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Variant or major revision of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

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

Revision or minor version of the MHU implementation

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**IMPLEMENTER, bits [11:0]**

Implementer ID

Contains the JEP106 identification information as follows:

- 11:8 - JEP106 continuation code of implementer
- 7 - Always 0
- 6:0 - JEP106 identity code of implementer

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

**Accessing RSC_IIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFC8 from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.9 *RSC_AIDR, Receiver Security Architecture Identification Register*

The RSC_AIDR characteristics are:

**Purpose**

Provides information on the version of the MHU architecture implemented in this implementation of the MHU.

**Configuration**

This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_AIDR are RAZ/WI.
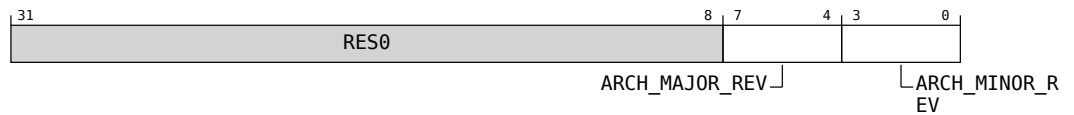
**Attributes**

RSC_AIDR is a 32-bit register.

This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_AIDR bit assignments are:



**Bits [31:8]**

Reserved, RES0.

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

MHU Architecture Major Revision

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

| ARCH_MAJOR_REV | Meaning |
| --- | --- |
| 0b0000 | MHUv1 or unknown architecture versions |
| 0b0001 | MHUv2 |
| 0b0010 | MHUv3 |

All other values are Reserved

Access to this field is **RO**.

**ARCH_MINOR_REV, bits [3:0]**

MHU Architecture Minor Revision

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

| ARCH_MINOR_REV | Meaning |
| --- | --- |
| 0b0000 | Minor revision 0 of the major architecture |
| 0b0001 | Minor revision 1 of the major architecture |

All other values are Reserved

Access to this field is **RO**.

**Additional information**

The legal combinations for the ARCH_MAJOR_REV and ARCH_MINOR_REV fields are as follows:

| ARCH_MAJOR_REV | ARCH_MINOR_REV | Description |
|---|---|---|
| 0x0 | 0x0 | MHUv1 |
| 0x1 | 0x0 | MHUv2.0 |
| 0x1 | 0x1 | MHUv2.1 |
| 0x2 | 0x0 | MHUv3.0 |

MHU implementations based on this architecture, have the ARCH_MAJOR_REV set to 0x2 and ARCH_MINOR_REV set to 0x0.

**Accessing RSC_AIDR**

Accesses to this register use the following encodings:

**Accessible at offset 0xFCC from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.1.10   RSC_IMPL_DEF_ID<n>, IMPDEF Register, n = 0 - 11

The RSC_IMPL_DEF_ID<n> characteristics are:

**Purpose**

> This register is for IMPLEMENTATION DEFINED Identification Registers which can be used to identify the implementation of the MHU Receiver Security

> An implementation can do the following with this register:

> • Chose whether a register is present or not.

> • The name of the register.

> • The access permission of the registers, so long as they are not more permissive than the architecture defines for the IMPLEMENTATION DEFINED register.

> • The names, behavior and reset value of any fields.

> • The access permissions of any fields, so long as they are not more permissive than the register.

> If the RSC_IMPL_DEF_ID<n> is not present then the location is Reserved and treated as RES0

**Configuration**

> This register is present only when TZE is implemented for the MHUR. Otherwise, direct accesses to RSC_IMPL_DEF_ID<n> are RAZ/WI.

**Attributes**

> RSC_IMPL_DEF_ID<n> is a 32-bit register.

> This register is part of the MHUR.RSC.RSC_CTRL_page block.

**Field descriptions**

The RSC_IMPL_DEF_ID<n> bit assignments are:

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

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

IMPLEMENTATION DEFINED

**Accessing RSC_IMPL_DEF_ID<n>**

Accesses to this register use the following encodings:

**Accessible at offset 0xFD0 + (4 * n) from MHUR.RSC.RSC_CTRL_page**

Access on this interface is **RO**.

### C2.2.2.2 RSC_IMPL_DEF_page, Receiver Security Control Implementation Defined page

The RSC_IMPL_DEF_page characteristics are:

**Purpose**

> The RSC_IMPL_DEF_page is for an implementation of the MHU, to implement any IMPLEMENTATION DEFINED registers.
>
> Registers defined in this block must follow the rules in  C1.7.3 *IMPLEMENTATION DEFINED Page*

**Configuration**

> This Register Block is present only when TZE is implemented for the MHUR.

**Attributes**

> The RSC_IMPL_DEF_page block is of size 4KB.
>
> This block is part of the MHUR.RSC block.

**Default access**

> Accesses to the address space of this block that do not resolve to a register or a further block are treated as RAZ/WI.

**Contents**

| Offset | Name | Notes |
|---|---|---|
| `0x0000 + (4 * n)`for n in 1023:0 | - | Most permissive access: ImplementationDefined |

# Part D
## Appendix

# Chapter D1
# Postbox and Mailbox Interrupt Logic

## D1.1  Postbox Interrupt Logic

$I_{DFKWG}$      Figure D1.1 shows an example logic structure of the interrupt generation logic of a Postbox.
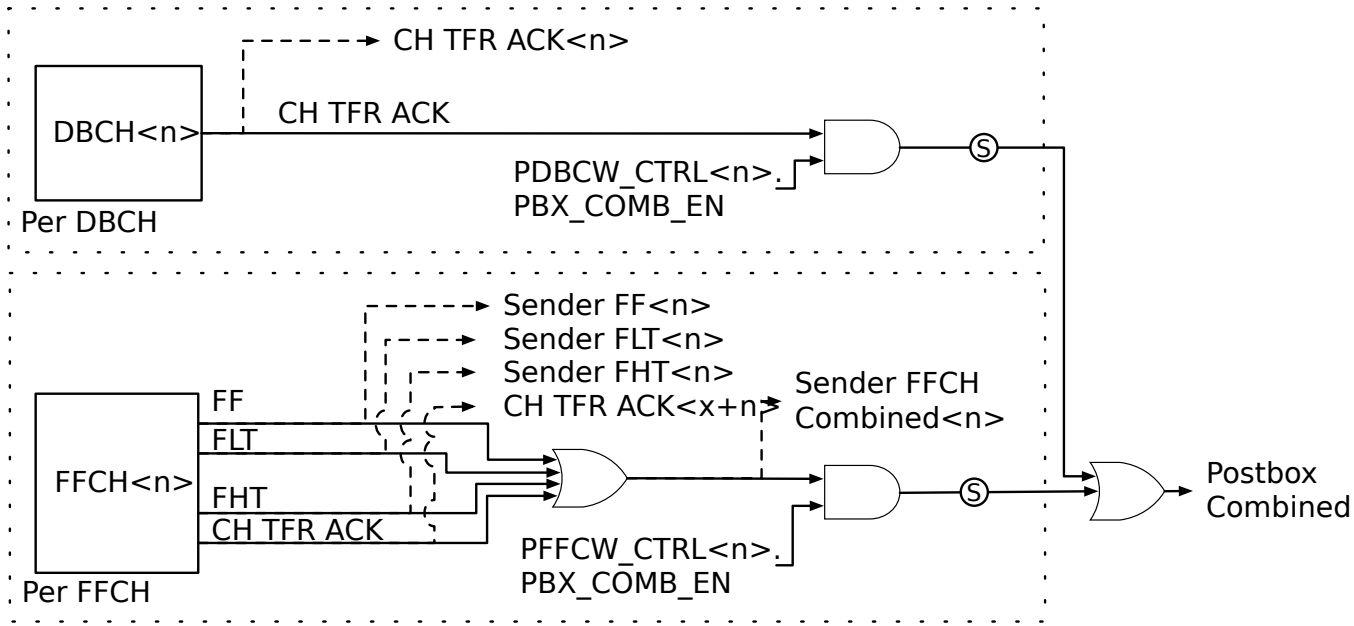
**Figure D1.1: Postbox interrupt logic**

In Figure D1.1 the following apply:

- Dotted lines indicate optional output of logic.
- Dashed boxes indicate units which are repeated for each channel.
- <n> indicates the channel number, which starts from 0 for each type of channel.
- <x> is the number of DBCHs implemented in the Postbox.
- Circles with 'S' inside indicate points where status are generated for Postbox Combined interrupts.

## D1.2 Mailbox Interrupt Logic

I<sub>MDDPZ</sub> Figure D1.2 shows an example logic structure of the interrupt generation logic of a Mailbox.
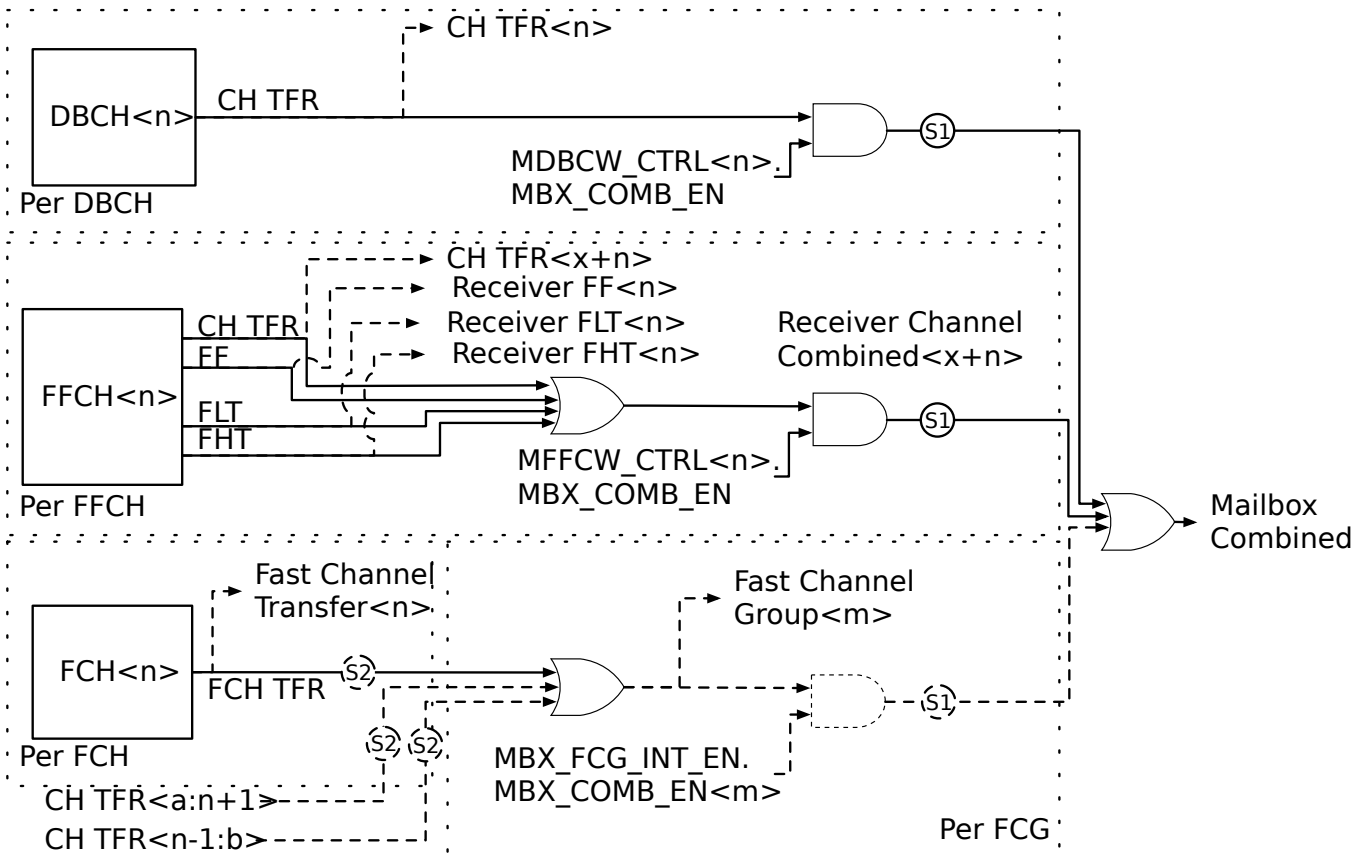


**Figure D1.2: Mailbox interrupt logic**

In Figure D1.2 the following apply:

- Dotted lines indicate optional output of logic.
- Dashed boxes indicate units which are repeated for each channel.
- <n> indicates the channel number, which starts from 0 for each type of channel.
- <x> is the number of DBCHs implemented in the Postbox.
- <m> is the FCG number.
- <a> is the upper most Fast Channel Transfer interrupt which feeds into the Fast Channel Group Transfer interrupt <m>.
- <b> is lowest most Fast Channel Transfer interrupt which feeds into the Fast Channel Group Transfer interrupt <m>.
- Circles with 'S1' inside indicate points where status are generated for Mailbox Combined interrupt.
- Circles with 'S2' inside indicate points where status are generated for the Fast Channel Group Transfer interrupt.

# Chapter D2
# Transport Protocols

$I_{GFTRL}$      This sections covers the expected transport protocols used by the Sender and Receiver.

$I_{ZDPMD}$      A transport protocol is a method for a Sender to send Transfer to the Receiver using the channels of the MHU.

$R_{PZXLB}$      MHUv3 supports the following transport protocols:

- Doorbell
- FIFO
- Stream FIFO
- Last-value

$S_{PNTZP}$      Software must use one of the transport protocols described in this section to send Transfers using the MHUv3, otherwise it is not guaranteed that in future versions of the MHU architecture the same method will still work.

# D2.1 Doorbell

$I_{DFJRP}$     The Doorbell transport protocol enables the Sender to indicate to the Receiver that a specific event has occurred.

$I_{CHFNH}$     It is software IMPLEMENTATION DEFINED:

- The event associated with the doorbell.
- If there is any out-band payload associated with the event.
- The location of any out-band payload.

$R_{CBFZB}$     The Doorbell transport protocol can only be used with a DBCH.

$I_{BQHLS}$     Each DBCH has 32 flags bits each of which can be used to indicate a different Doorbell event has occurred. Each time the Sender sets one or more of the flags bits of the DBCH it is considered its own Transfer.

## D2.1.1 Requirements

$S_{RRSPW}$     To use the Doorbell protocol the Channel Transfer and Channel Transfer Acknowledge events of the DBCH are used and the Sender and Receiver must configure the DBCH so that the Sender and Receiver can receive these events.

### D2.1.1.1 Sender Requirements

$S_{JPZDF}$     The Sender must only write to the PDBCW<n>_SET register after the event which the Doorbell is related to has occurred and any out-band payload is guaranteed to be visible to the Receiver.

$S_{RPHDK}$     Once a flag bit is set, the Sender must wait for the Receiver to acknowledge the flag by clearing it, before the Sender uses the same flag bit to send a new Transfer.

$S_{WFNDP}$     The Sender knows that the Receiver has acknowledged one or more previous Transfers by waiting for the flag bit or bits in the PDBCW<n>_ST register to read as `0b0`.

$S_{LQMQX}$     The Sender can use the Channel Transfer Acknowledge event to know when the Receiver has acknowledged one or more previous Transfer and therefore the values of one or more flag bits in the PDBCW<n>_ST register have been updated. This removes the need for the Sender to poll the PDBCW<n>_ST register.

### D2.1.1.2 Receiver Requirements

$S_{DFTCQ}$     The Receiver acknowledges receipt of the Transfer by clearing the flag in the MDBCW<n>_ST register by writing the MDBCW<n>_CLR register with the corresponding bit set to `0b1`.

$S_{JHYLC}$     Any flag bit of the MDBCW<n>_ST register used, which are required to generate a Channel Transfer event, the Receiver must have the corresponding bit in the MDBCW<n>_MSK_ST register set to `0b0`.

$I_{CWJPP}$     If a flag bit in the MDBCW<n>_ST register is not required to generate a Channel Transfer event, the Receiver must set the corresponding bit in the MDBCW<n>_MSK_ST register to be set to `0b1`.

## D2.1.2 Procedure

$R_{XHYYQ}$     In  Figure D2.1 the sequence of events to use the Doorbell transport protocol is shown. Arrows with dotted lines indicate optional steps.
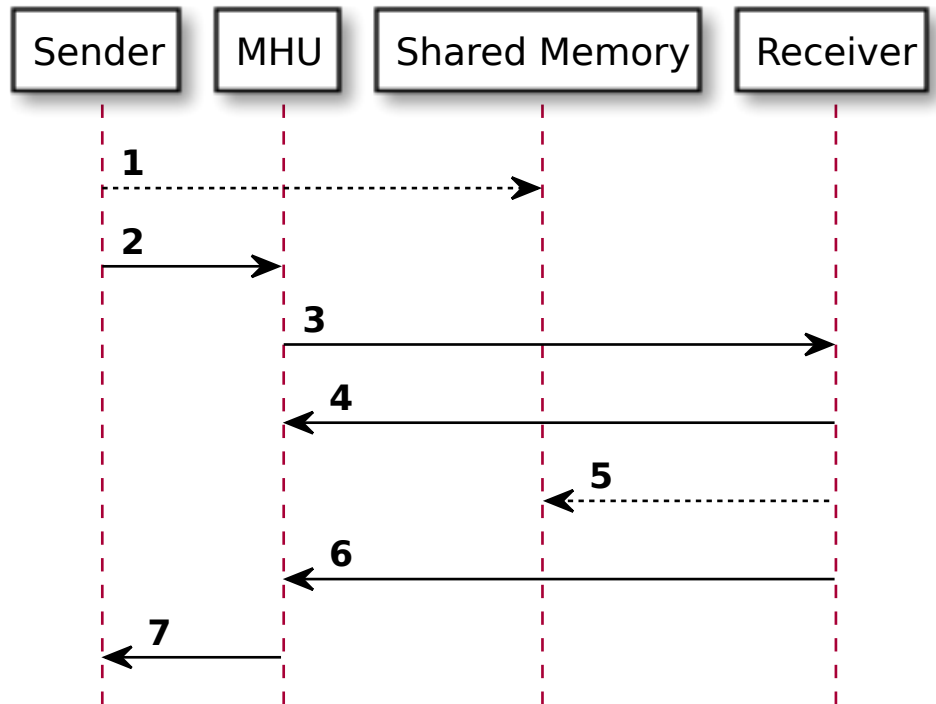
**Figure D2.1: Doorbell transport protocol**

The sequence of events are:

1. Sender writes any optional out-band payload for the Transfer to the out-band memory.
2. Sender writes `0b1`, to the correct flag in the PDBCW<n>_SET register.
3. MHU generates a Channel Transfer event. The Channel Transfer events can generate one or more interrupts depending on the settings the DBCH in the Mailbox:

   - Channel Transfer interrupt, if the corresponding bit in the MDBCW<n>_MSK_ST register is set to `0b0`.
   - Mailbox Combined interrupt, if the corresponding bit in the MDBCW<n>MSK_ST register is set to `0b0` .and MDBCW<n>_CTRL.MBX_COMB_EN is set to `0b1`.

4. Receiver reads MDBCW<n>_ST register to identify which Doorbell(s) are outstanding.
5. Receiver reads any optional payload for the Transfer.
6. Receiver clears the flag by writing `0b1`, to the corresponding bit in the MDBCW<n>_CLR register.
7. MHU generates a Channel Transfer Acknowledge event. The Channel Transfer Acknowledge event can generate one or more interrupts of the following interrupts:

   - Channel Transfer Acknowledge interrupt, if PDBCW<n>_INT_EN.CH_TFR_ACK is set to `0b1`.
   - Postbox Combined interrupt, if PDBCW<n>_CTRL.PBX_COMB_EN is set to `0b1`.

The steps performed by the Sender and Receiver in  Figure D2.1 can be run concurrently or sequentially. It is even possible that the Sender is performing the sequences for Transfer Y whilst the Receiver is performing the sequences for Transfer X, where Transfer X is sent by before Transfer Y.

When the Receiver performs step 6 it is enabling the Sender to send another Transfer using the flag bit that was cleared. It is not required that Receiver has acted on the previous Transfer and, if there is any out-band payload, has finished with its use of the payload.

## D2.2 FIFO

$I_{NYHNP}$  FIFO transport protocol enables:

- Multiple Transfers in-flight using a single channel.
- Transfers with variable length.
- Transfers with in-band payload and an optional out-band payload.

$S_{GSFDN}$  It is software IMPLEMENTATION DEFINED:

- What the values of the data mean.
- How many bytes of in-band and out-band data there are in a Transfer, but there must be at least one byte of in-band payload.
- Whether the number of bytes in-band and out-band payload is the same for all Transfers.
- The location of any out-band payload.
- Whether the Sender of the Transfer receives interrupts when the Receiver acknowledges a Transfer or not.

$I_{VXWVD}$  The number of outstanding Transfers is limited by the amount of in-band data each Transfer uses and the depth of the FIFO.

$S_{CYMNN}$  The maximum amount of in-band data a Transfer uses must not exceed the FIFO depth, otherwise data loss can occur.

### D2.2.1 Requirements

$I_{PYDYK}$  This sections covers the requirements to use the FIFO transport protocol.

$R_{DBHRH}$  The FIFO transport protocol must only be used with a FFCH.

$S_{NXYYW}$  The Sender and Receiver must use the same value for PFFCW<n>_CTRL.MSBF and MFFCW<n>_CTRL.MSBF fields for the same Transfer when pushing, reading or popping bytes from the FIFO.

$S_{JTWFG}$  The method by which the Sender and Receiver agree on what value to set the PFFCW<n>_CTRL.MSBF and MFFCW<n>_CTRL.MSBF fields to is software IMPLEMENTATION DEFINED.

$S_{CJXMN}$  The Sender can write zero or more bytes to an out-band payload location. It is software IMPLEMENTATION DEFINED when the Receiver reads the out-band payload and depends on the following:

- How and when the Receiver knows the address to use to access the out-band payload.
- When and how the Sender knows that the out-band payload location is no longer required by the Receiver.

#### D2.2.1.1 Sender Requirements

$S_{KBKHN}$  The Sender must not push new data onto the FIFO if the FIFO is full.

$I_{LBLHV}$  The Sender can determine the amount of free space in the FIFO using the value in the PFFCW<n>_ST.FFS field or the Sender Low and High Tide events.

$S_{MVQWK}$  The Sender must only push bytes onto the FIFO which belong to the same Transfer using a single write to the PFFCW<n>_PAY register.

$S_{TZFZR}$  The Sender is only allowed to change the value of the PFFCW<n>_CTRL.MSBF field before it pushes any bytes onto the FIFO for the Transfer.

$I_{BYYRD}$  Depending on the Transfer Delineation mode being used the Sender must obey different rules for the following:

- Management of the SOT, EOT and ACK flags.
- Number of writes to the PFFCW<n>_PAY register, to push data onto the FIFO, for a single Transfer.

$S_{BYDXR}$      When using software flag Transfer Delineation mode the Sender must:

- Set the PFFCW<n>_FLG.SOT field to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the first byte of the Transfer.
- Have the first byte of the Transfer in either the:

    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Set the PFFCW<n>_FLG.SOT field to `0b0` before it pushes one or more bytes onto the FIFO, where none of the bytes are the first byte of the Transfer.
- Set the PFFCW<n>_FLG.ACK field to the correct value before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.

    - If the Sender wants a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b1`.
    - If the Sender does not want a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b0`.

- Set the PFFCW<n>_FLG.EOT field to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.
- Have the last byte of the Transfer in either the:

    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Set the PFFCW<n>_FLG.EOT field to `0b0` before it pushes one or more bytes onto the FIFO, where none of the bytes are the last byte of the Transfer.

$S_{TKHBM}$      When using software flag Transfer Delineation mode the Sender can perform as many writes to the PFFCW<n>_PAY register as required to push all data of the Transfer onto the FIFO. So long as there is space in the FIFO for all bytes, requested by the write to be pushed, to be pushed onto the FIFO. It is not required that each write to the PFFCW<n>_PAY register, when using software flag Transfer Delineation mode, are of the same size.

$S_{ZFYHY}$      It might be required that the Sender either:

- Has to wait between writes to the PFFCW<n>_PAY register for FIFO to have enough space to push next N bytes of the Transfer.
- Has to reduce the number of bytes to be pushed onto the FIFO.

$S_{YZTJD}$      When using partial flag Transfer Delineation mode the Sender must:

- Set the PFFCW<n>_FLG.{SOT/EOT} fields to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the first byte of and one of the bytes is the last byte of the Transfer. This also applies if they are the same byte.
- Set the PFFCW<n>_FLG.EOT field to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer and none of the bytes are the first byte of the Transfer.
- Set the PFFCW<n>_FLG.ACK field to the correct value before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.

    - If the Sender wants a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b1`.
    - If the Sender does not want a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b0`.

- Have the first byte of the Transfer in either the:

    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Have the last byte of the Transfer in either the:

- MSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
- LSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

$S_{HWSBQ}$    When using partial flag Transfer Delineation mode the Sender can perform as many writes to the PFFCW<n>_PAY register as required to push all data of the Transfer onto the FIFO. So long as there is space in the FIFO for all bytes, requested by the write to be pushed, to be pushed onto the FIFO. It is not required that each write to the PFFCW<n>_PAY register, when using partial flag Transfer Delineation mode, are of the same size.

$S_{LHVZP}$    It might be required that the Sender either:

- Has to wait between writes to the PFFCW<n>_PAY register for FIFO to have enough space to push next N bytes of the Transfer.
- Has to reduce the number of bytes to be pushed onto the FIFO.

$S_{WZXHD}$    When using auto flag Transfer Delineation mode the Sender must:

- Set the PFFCW<n>_FLG.ACK field to the correct value before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.

  - If the Sender wants a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b1`.
  - If the Sender does not want a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b0`.

- Have the first byte of the Transfer in either the:

  - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
  - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Have the last byte of the Transfer in either the:

  - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
  - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Never write to the PFFCW<n>_PAY register, where one of the bytes is the first byte and one of the bytes is the last byte of the Transfer, including if they are the same byte.

$S_{GMTJJ}$    When using auto flag Transfer Delineation mode the Sender must perform two writes to the PFFCW<n>_PAY register to push the data of the Transfer onto the FIFO, and there must be space in the FIFO to push all bytes, requested by the write to be pushed onto the FIFO, to be pushed onto the FIFO. It is not required that each write to the PFFCW<n>_PAY register, when using auto flag Transfer Delineation mode, are of the same size.

$S_{TRCYX}$    It might be required that the Sender either:

- Has to wait between writes to the PFFCW<n>_PAY register for FIFO to have enough space to push next N bytes of the Transfer.
- Has to reduce the number of bytes to be pushed onto the FIFO.

$S_{WLJYF}$    The Channel Transfer Acknowledge event of the FFCH are used by the Sender to know when a Transfer has been acknowledge by the Receiver. The Sender can either:

- Configure the FFCH to generate the Channel Transfer Acknowledge interrupt and when the interrupt is generated to read the value of the PFFCW<n>_ACK_CNT.ACK_CNT field to know how many Transfer have been acknowledged.
- Poll the PFFCW<n>_INT_ST.CH_TFR_ACK field for a value of `0b1` and then read the value of the PFFCW<n>_ACK_CNT.ACK_CNT to know how many Transfers have been acknowledged.
- Poll the PFFCW<n>_ACK_CNT.ACK_CNT field for a non-zero value.

$I_{PZRPH}$    For the rest of this section it is assumed the Channel Transfer Acknowledge interrupt is used, however, any of the other methods could be used.

## D2.2.1.2 Receiver Requirements

$S_{XTHDX}$  The Receiver is only allowed to change the value of the MFFCW\<n>_CTRL.MSBF field before it reads or pops any bytes from the FIFO for the Transfer.

$S_{ZCZGR}$  The Receiver must use the values of MFFCW\<n>_FLG register to track the flag values of the bytes it has read from the MFFCW\<n>_PAY register. It uses these values to know:

- Whether a byte is valid or invalid.
- Which bytes belong to which Transfers.
- To detect partial Transfers.

$S_{WXZQC}$  Software should only use the value of MFFCW\<n>_FLG register after it has read at least one byte from the MFFCW\<n>_PAY register.

$S_{NDPQV}$  The Receiver must read the contents of the MFFCW\<n>_FLG register each time data is read from the FIFO, otherwise it can lead to inclusions of data values which are not part of the Transfer.

$S_{ZVWNJ}$  Table D2.1 shows the value of SOT and EOT fields for the current and previous byte, where both bytes are valid. Any non-valid bytes are to be ignore. When there has been no previous valid byte then it should be considered that EOT was set to `0b1`.

**Table D2.1: Meanings of current and previous SOT and EOT fields**

| SOT' | EOT' | SOT | EOT | Error | Description |
|------|------|-----|-----|-------|-------------|
| X | 0 | 0 | 0 | No | Current and previous bytes are part of the same Transfer and there are more bytes to read |
| X | 0 | 0 | 1 | No | Current and previous bytes are part of the same Transfer and there are no more bytes to read |
| X | 0 | 1 | 0 | Yes | New Transfer started without completing previous Transfer |
| X | 0 | 1 | 1 | Yes | New Transfer started without completing previous Transfer |
| X | 1 | 0 | 0 | Yes | Implicit new Transfer started with the beginning of the Transfer missing |
| X | 1 | 0 | 1 | Yes | Implicit new Transfer started with the beginning of the Transfer missing |
| X | 1 | 1 | 0 | No | New Transfer started and there are more bytes to read |
| X | 1 | 1 | 1 | No | New Transfer started and ended |

SOT' and EOT' are the values of SOT and EOT for the previous byte.

$S_{SWKMK}$  Whenever the Receiver detects a sequence of SOT', EOT', SOT and EOT flags for two valid bytes it did not expect it should take IMPLEMENTATION DEFINED actions to resolve the corrupt Transfers.

$I_{YCRWL}$  The MFFCW\<n>_FLG register contains the flags associated with the bytes last read from the FIFO by the last read of the MFFCW\<n>_PAY register.

$S_{DSBPF}$  To use the FIFO protocol the Channel Transfer event of the FFCH is used and the Receiver must either:

- Configure the FFCH to generate Channel Transfer interrupts so that it can receive these events.
- Poll the MFFCW\<n>_PAY and MFFCW\<n>_FLG register until valid bytes are detected.
- Poll the MFFCW\<n>_ST.FFL until a non-zero value is detected.

$I_{LDPZJ}$  For the rest of this section it is assumed the Channel Transfer interrupt is used, however, any of the other methods could be used.

$S_{HYQCN}$      When the Receiver is using Read-Acknowledge without enabling Future Transfer Auto Buffering, it may be required to buffer bytes read out of the MFFCW<n>_PAY register whilst processing the current Transfer, which are part of the next Transfer. These buffered bytes are required to be processed when the Receiver processes the next Transfer and are considered the first bytes of the next Transfer. Only bytes which have a flag value of 0b01 or 0b00 and come after a byte with 0b01 are required to be buffered. Failure to buffer these bytes will result in a lost of data.

$S_{YCZWQ}$      When the Receiver is not using Read-Acknowledge it must only pop the bytes up to the end of current Transfer from the FIFO. Popping the incorrect number of bytes from the FIFO can lead to loss or corruption of Transfer data.

## D2.2.2 Procedure

$S_{JFNPF}$      In Figure D2.2 shows the sequence of events to use the FIFO transport protocol. Arrows with dotted lines indicate optional steps.

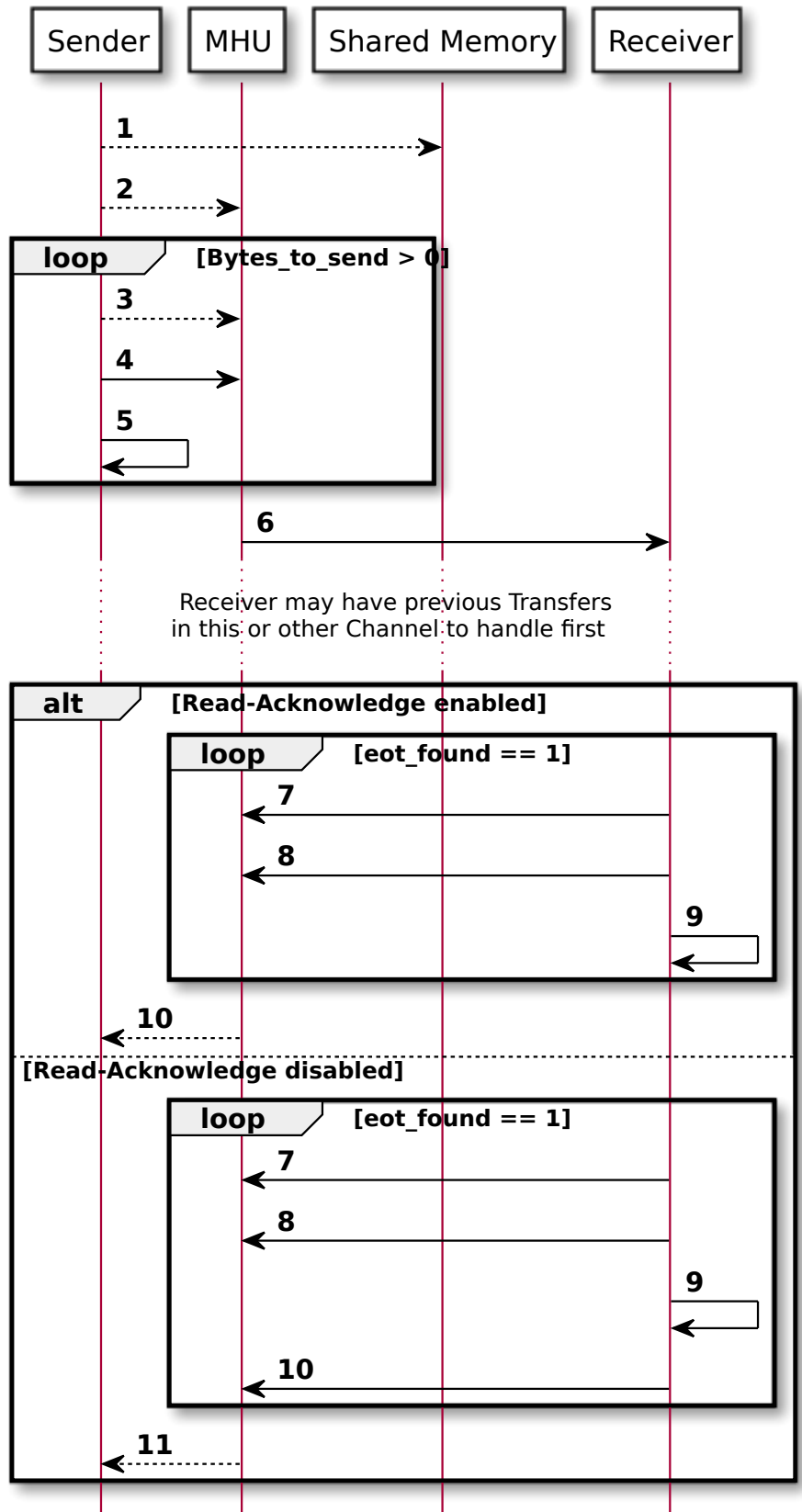**Figure D2.2: FIFO transport protocol**

The sequence of events shown in Figure D2.2:

1. Sender writes any out-band payload to shared memory.
2. Sender configures the Transfer Delineation Mode by writing to the PFFCW<n>_CTRL.TDM field.

   This is not required for every transfer and can be set when performing the initial configuration of the FFCH or when the conditions change to require a different mode be used.

**While Bytes to Send > 0**

3. Sender writes to the PFFCW<n>_FLG register to set the data flags correctly.

   Depending on the Transfer Delineation Mode the Sender:

   - Software flag

     If next write to the PFFCW<n>_PAY register includes the:

     - First byte of the Transfer set the PFFCW<n>_FLG.SOT field to `0b1`, otherwise set the field to `0b0`.
     - Last byte of the Transfer set the PFFCW<n>_FLG.EOT field to `0b1`, otherwise set the field to `0b0`.
     - Last byte of the Transfer and an acknowledgement is required for this Transfer set the PFFCW<n>_FLG.ACK field to `0b1`, otherwise set it to `0b0`.

   - Partial flag

     If next write to the PFFCW<n>_PAY register includes the:

     - First byte of the Transfer this step is not required.
     - First and last byte of the Transfer set the PFFCW<n>_FLG.{SOT,EOT} fields both to `0b1`.
     - Last byte but not the first byte of the Transfer set the PFFCW<n>_FLG.EOT field to `0b1`.
     - Last byte of the Transfer and an acknowledgement is required for this Transfer set the PFFCW<n>_FLG.ACK field to `0b1`, otherwise set it to `0b0`.

   - Auto flag

     Sets the PFFCW<n>_FLG.ACK field to `0b1` if the Sender wants an acknowledgement for this Transfer, otherwise sets it `0b0` if the Sender does not want an acknowledgement for this Transfer.

4. Sender performs a write access to the PFFCW<n>_PAY register with the payload.

   The Sender use an access equal to the amount of data that the Sender wants to push onto the FIFO. The order of the bytes in the payload written to the PFFCW<n>_PAY register depends on the setting of the PFFCW<n>_CTRL.MSBF field.

5. Sender decreases the number of bytes to send by the number of bytes written to the PFFCW<n>_PAY register. If the number of bytes to send is 0 the Sender exits the loop, otherwise repeat steps 3 and 4 until all bytes of the Transfer have been pushed onto the FIFO.

6. MHU generates a Channel Transfer event, when one or more bytes are pushed onto the FIFO and the EOT flag for one of the bytes is set to `0b1`. The Channel Transfer event becomes one or more interrupts depending on the settings of the interrupt enables for the FFCH in the Mailbox:

   - Channel Transfer interrupt, if MFFCW<n>_INT_EN.CH_TFR is set to `0b1`.
   - Receiver FFCH Combined interrupt, if MFFCW<n>_INT_EN.CH_TFR is set to `0b1`.
   - Mailbox Combined interrupt, if MFFCW<n>_INT_EN.CH_TFR and MFFCW<n>_INT_EN.MBX_COMB_EN are both set to `0b1`.

**Read-Acknowledge enabled**

7. Receiver reads data from the MFFCW<n>_PAY register.
8. Receiver reads the flag information from the MFFCW<n>_FLG register.

9. Receiver checks the flag information for each byte read from the MFFCW<n>_FLG register.

   Bytes read from the MFFCW<n>_PAY register are treated as defined in Table D2.1.

   If any valid byte read from the MFFCW<n>_PAY register associated with a flag value indicates the start of a new Transfer, the Receiver exists the loop, otherwise repeats steps 6-8.

10. MHU generates Transfer Acknowledge event, for any bytes which are popped from the FIFO which have the EOT and ACK fields set to `0b1`. The Transfer Acknowledge event can generate one or more interrupts depending on the settings for the FFCH in the Postbox:

    - Channel Transfer Acknowledge interrupt, if the PFFCW<n>_INT_EN.CH_TFR_ACK is set to `0b1`.
    - Sender FFCH Combined interrupt, if the PFFCW<n>_INT_EN.CH_TFR_ACK is set to `0b1`.
    - Postbox Combined interrupt, if the PFFCW<n>_INT_EN.CH_TFR_ACK and PFFCW<n>_INT_EN.PBX_COMB_EN are both set to `0b1`.

**Read-Acknowledge disabled**

7. Receiver reads data from the MFFCW<n>_PAY register.
8. Receiver reads the flag information from the MFFCW<n>_FLG register.
9. Receiver checks the flag information for each byte read from the MFFCW<n>_FLG register.

   Bytes read from the MFFCW<n>_PAY register are treated as defined in Table D2.1.

10. Receiver writes to the MFFCW<n>_FIFO_POP register with a value equal to the number of bytes the Receiver wants to pop from the FIFO.

    If any valid byte read from the MFFCW<n>_PAY register associated with a flag indicates the start of a new Transfer, Receiver exists the loop, otherwise repeats steps 7-10.

11. MHU generates Transfer Acknowledge event, for any bytes which are popped from the FIFO which have the EOT and ACK fields set to `0b1`. The Transfer Acknowledge event can generate one or more interrupts depending on the settings of the FFCH in the Postbox:

    - Channel Transfer Acknowledge interrupt, if the PFFCW<n>_INT_EN.CH_TFR_ACK is set to `0b1`.
    - Sender FFCH Combined interrupt, if the PFFCW<n>_INT_EN.CH_TFR_ACK is set to `0b1`.
    - Postbox Combined interrupt, if the PFFCW<n>_INT_EN.CH_TFR_ACK and PFFCW<n>_INT_EN.PBX_COMB_EN are both set to `0b1`.

$I_{CRXYR}$ The steps performed by the Sender and Receiver in Figure D2.2 can be run concurrently or sequentially. It is even possible that the Sender is performing the sequences for Transfer Y whilst the Receiver is performing the sequences for Transfer X, where Transfer X is sent by before Transfer Y.

# D2.3 Streaming FIFO

$I_{FFGQP}$  Streaming FIFO transport protocol enables Transfers of undefined lengths, even larger than the size of the FIFO. It uses the FIFO Free Space and FIFO Fill level to allow the:

- Sender to know when bytes can be pushed onto the FIFO.
- Receiver to know when bytes are ready to be read and popped from the FIFO.

$S_{MHGLB}$  It is software IMPLEMENTATION DEFINED:

- What the values of the data mean.
- How many bytes of in-band and out-band data there are in a Transfer, but there must be at least one byte of in-band payload.
- Whether the number of bytes in-band and out-band payload is the same for all Transfers.
- The location of any out-band payload.
- Whether the Sender of the Transfer receives interrupts when the Receiver acknowledges a Transfer.
- Number of bytes which the Sender can push onto the FIFO before it stops.
- Number of bytes which must be in the FIFO before the Receiver will read them from the FIFO.

$S_{PSDQM}$  The number of outstanding Transfers is limited by the amount of in-band data each Transfer uses and the depth of the FIFO.

$S_{JWPHV}$  There is no limit on the amount of in-band data used by a Transfer.

## D2.3.1 Requirements

$I_{MLXCY}$  This section covers the requirements to use the streaming FIFO transport protocol.

$R_{NLJCL}$  The streaming FIFO transport protocol must only be used with a FFCH.

$S_{HXJXH}$  The Sender and Receiver must use the same value for PFFCW<n>_CTRL.MSBF and MFFCW<n>_CTRL.MSBF fields for the same Transfer when pushing, reading or popping bytes from the FIFO.

$S_{DBSHV}$  The method by which the Sender and Receiver agree on what value to set the PFFCW<n>_CTRL.MSBF and MFFCW<n>_CTRL.MSBF fields to is software IMPLEMENTATION DEFINED.

$S_{PGGMV}$  To use the streaming FIFO protocol the FIFO Low and High Tide events are used by both the Sender and Receiver and the Sender and Receiver must either:

- Configure the FFCH so that it can receive these events.
- Use the values of the FIFO Free Space or FIFO Fill Level fields provided by the MHU.

$S_{RTFKF}$  When using the Sender and Receiver FIFO Low and High Tidemark events there can be a delay in-between the event being generated and the Sender or Receiver detecting the event, via an interrupt.

To avoid lost of data, Arm recommends that the Sender sets the PFFCW<n>_TIDE.HIGH field to a value which generates the Sender High Tide event early enough so that the Sender can stop push new data onto the FIFO before the FIFO is full.

$S_{LLQSH}$  The Sender can write zero or more bytes to an out-band payload location. It is software IMPLEMENTATION DEFINED when the Receiver reads the out-band payload and depends on the following:

- How and when the Receiver knows the address to use to access the out-band payload.
- When and how the Sender knows that the out-band payload location is no longer required by the Receiver.

### D2.3.1.1 Sender Requirements

$S_{GVWTL}$  The Sender must not push new data onto the FIFO if the FIFO is full.

$I_{QWBRJ}$ The Sender can determine the amount of free space in the FIFO using the value in the PFFCW<n>_ST.FFS or PFFCW<n>_PAY.FFS fields or FIFO High Tide event.

---

**Note**

The values of the PFFCW<n>PAY.FFS and PFFCW<n>_FFS fields can be less than the actual value, if the size of the read access to the register does not have enough bits to encode the FFS field in the returned data. However, the value will always be able to show that it is possible to perform at least one more push operation of any supported size.

For more information on when this occurs refer to Chapter B8 *FIFO Extension*.

---

$S_{HLMGD}$ The Sender must only push bytes onto the FIFO which belong to the same Transfer using a single write to the PFFCW<n>_PAY register.

$S_{TLLPC}$ The Sender is only allowed to change the value of the PFFCW<n>_CTRL.MSBF field before it pushes any bytes onto the FIFO for the Transfer.

$I_{ZKWKB}$ Depending on the Transfer Delineation mode being used the Sender must obey different rules for the following:

- Management of the SOT, EOT and ACK flags.
- Number of writes to the PFFCW<n>_PAY register, to push data onto the FIFO, for a single Transfer.

$S_{PPFBY}$ When using software flag Transfer Delineation mode the Sender must:

- Set the PFFCW<n>_FLG.SOT field to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the first byte of the Transfer.
- Have the first byte of the Transfer in either the:

  - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
  - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Set the PFFCW<n>_FLG.SOT field to `0b0` before it pushes one or more bytes onto the FIFO, where none of the bytes are the first byte of the Transfer.
- Set the PFFCW<n>_FLG.ACK field to the correct value before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.

  - If the Sender wants a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b1`.
  - If the Sender does not want a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b0`.

- Set the PFFCW<n>_FLG.EOT field to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.
- Have the last byte of the Transfer in either the:

  - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
  - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Set the PFFCW<n>_FLG.EOT field to `0b0` before it pushes one or more bytes onto the FIFO, where none of the bytes are the last byte of the Transfer.

$S_{BMPSZ}$ When using software flag Transfer Delineation mode the Sender can perform as many writes to the PFFCW<n>_PAY register as required to push all data of the Transfer onto the FIFO. So long as there is space in the FIFO for all bytes, requested by the write to be pushed, to be pushed onto the FIFO. It is not required that each write to the PFFCW<n>_PAY register, when using software flag Transfer Delineation mode, are of the same size.

$S_{\text{GNXLF}}$  It might be required that the Sender either:

- Has to wait between writes to the PFFCW<n>_PAY register for FIFO to have enough space to push next N bytes of the Transfer.
- Has to reduce the number of bytes to be pushed onto the FIFO.

$S_{\text{BYCDH}}$  When using partial flag Transfer Delineation mode the Sender must:

- Set the PFFCW<n>_FLG.{SOT/EOT} fields to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the first byte of and one of the bytes is the last byte of the Transfer. This also applies if they are the same byte.
- Set the PFFCW<n>_FLG.EOT field to `0b1` before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer and none of the bytes are the first byte of the Transfer.
- Set the PFFCW<n>_FLG.ACK field to the correct value before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.

    - If the Sender wants a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b1`.
    - If the Sender does not want a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b0`.

- Have the first byte of the Transfer in either the:

    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Have the last byte of the Transfer in either the:

    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

$S_{\text{VDPZG}}$  When using partial flag Transfer Delineation mode the Sender can perform as many writes to the PFFCW<n>_PAY register as required to push all data of the Transfer onto the FIFO. So long as there is space in the FIFO for all bytes, requested by the write to be pushed, to be pushed onto the FIFO. It is not required that each write to the PFFCW<n>_PAY register, when using partial flag Transfer Delineation mode, are of the same size.

$S_{\text{YFGLQ}}$  It might be required that the Sender either:

- Has to wait between writes to the PFFCW<n>_PAY register for FIFO to have enough space to push next N bytes of the Transfer.
- Has to reduce the number of bytes to be pushed onto the FIFO.

$S_{\text{CYYYD}}$  When using auto flag Transfer Delineation mode the Sender must:

- Set the PFFCW<n>_FLG.ACK field to the correct value before it pushes one or more bytes onto the FIFO, where one of the bytes is the last byte of the Transfer.

    - If the Sender wants a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b1`.
    - If the Sender does not want a Channel Transfer Acknowledge event to be generated, when the last byte of the Transfer is popped from the FIFO, the Sender sets the PFFCW<n>_FLG.ACK field to `0b0`.

- Have the first byte of the Transfer in either the:

    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Have the last byte of the Transfer in either the:

    - MSB position when PFFCW<n>_CTRL.MSBF is set to `0b0`.
    - LSB position when PFFCW<n>_CTRL.MSBF is set to `0b1`.

- Never write to the PFFCW<n>_PAY register, where one of the bytes is the first byte and one of the bytes is the last byte of the Transfer, including if they are the same byte.

$S_{KJSBD}$ When using auto flag Transfer Delineation mode the Sender must perform two writes to the PFFCW<n>_PAY register to push the data of the Transfer onto the FIFO, and there must be space in the FIFO to push all bytes, requested by the write to be pushed onto the FIFO, to be pushed onto the FIFO. It is not required that each write to the PFFCW<n>_PAY register, when using auto flag Transfer Delineation mode, are of the same size.

$S_{LVDHS}$ It might be required that the Sender either has to:

- Wait between writes to the PFFCW<n>_PAY register for FIFO to have enough space to push next N bytes of the Transfer.
- Reduce the number of bytes to be pushed onto the FIFO.

$S_{JVHSS}$ To use the streaming FIFO protocol the Sender must keep track of the number of valid bytes that are in the FIFO. The Sender must either:

- Configure the FFCH to generate both the Sender FIFO Low and High Tide events and set the PFFCW<n>_TIDE.{HIGH/LOW} fields to appropriate values.
- Check the amount of free space in the FIFO and only writing to the PFFCW<n>_PAY if the free space is below the threshold to push new data onto the FIFO. The Sender can check the amount of free space by reading the value PFFCW<n>_ST.FFS field.

$I_{HCNRC}$ For the rest of this section it is assumed the Sender FIFO Low and High Tidemark interrupts are used, however, any of the other methods could be used.

$S_{DLCST}$ It is the responsibility of the Sender to read the PFFCW<n>_ACK_CNT register when the Transfer Acknowledge interrupt is generated to know how many Transfers have been acknowledged since the last time the Sender read the register.

### D2.3.1.2 Receiver Requirements

$S_{FPNTK}$ The Receiver is only allowed to change the value of the MFFCW<n>_CTRL.MSBF field before it reads or pops any bytes from the FIFO for the Transfer.

$S_{KKHKR}$ The Receiver can know when there are valid bytes in the FIFO by reading the value of MFFCW<n>_ST.FFL or MFFCW<n>_FLG.FFL fields or Receiver High Tide event.

---

**Note**

The values of the MFFCW<n>_ST.FFL and MFFCW<n>_FLG.FFL fields can be less than the actual value, if the size of the read access to the register does not have enough bits to encode the FFL field in the returned data. However, the value will always be able to show that it is possible to perform at least one more pop operation of any supported size.

For more information on when this occurs refer to  Chapter B8 *FIFO Extension*.

---

$S_{JTMXN}$ The Receiver must use the values of MFFCW<n>_FLG register to track the flag values of the bytes it has read from the MFFCW<n>_PAY register. It uses these values to know:

- Whether a byte is valid or invalid.
- Which bytes belong to which Transfers.
- To detect partial Transfers.

$S_{XDLDG}$ Software should only use the value of MFFCW<n>_FLG register after it has read at least one byte from the MFFCW<n>_PAY register.

$I_{WJDGJ}$    Table D2.2 shows the value of SOT and EOT fields for the current and previous byte, where both bytes are valid. A valid byte is a byte where the Valid flag is set. Any non-valid bytes are to be ignore. When there has been no previous valid byte then it should be considered that EOT was set to `0b1`.

**Table D2.2: Meanings of current and previous SOT and EOT fields**

| SOT' | EOT' | SOT | EOT | Error | Description |
|------|------|-----|-----|-------|-------------|
| X | 0 | 0 | 0 | No | Current and previous bytes are part of the same Transfer and there are more bytes to read |
| X | 0 | 0 | 1 | No | Current and previous bytes are part of the same Transfer and there are no more bytes to read |
| X | 0 | 1 | 0 | Yes | New Transfer started without completing previous Transfer |
| X | 0 | 1 | 1 | Yes | New Transfer started without completing previous Transfer |
| X | 1 | 0 | 0 | Yes | Implicit new Transfer started with the beginning of the Transfer missing |
| X | 1 | 0 | 1 | Yes | Implicit new Transfer started with the beginning of the Transfer missing |
| X | 1 | 1 | 0 | No | New Transfer started and there are more bytes to read |
| X | 1 | 1 | 1 | No | New Transfer started and ended |

SOT' and EOT' are the value of SOT and EOT for the previous byte.

$S_{NNGMM}$    Whenever the Receiver detects a sequence of SOT', EOT', SOT and EOT flags for two valid bytes it did not expect it should take IMPLEMENTATION DEFINED actions to resolve the corrupt Transfers.

$S_{HCWMX}$    When the Receiver is using Read-Acknowledge without enabling Future Transfer Auto Buffering, it may be required to buffer bytes read out of the MFFCW<n>_PAY register whilst processing the current Transfer, which are part of the next Transfer. These buffered bytes are required to be processed when the Receiver processes the next Transfer and are considered the first bytes of the next Transfer. Only valid bytes which have a flag value of `0b01` or `0b00` and come after a byte with `0b01` are required to be buffered. Failure to buffer these bytes will result in a lost of data.

$S_{XZBDZ}$    When the Receiver is not using Read-Acknowledge it must only pop the bytes up to the end of current Transfer from the FIFO. Popping the incorrect number of bytes from the FIFO can lead to loss or corruption of Transfer data.

$S_{JFDWD}$    To use the streaming FIFO protocol the Receiver must keep track of the number of valid bytes in the FIFO. The Receiver must either:

- Configure the FFCH to generate both the Receiver FIFO Low and High Tide events and set the MFFCW<n>_TIDE.{HIGH/LOW} fields to appropriate values.
- Poll the value of the MFFCW<n>_ST.FFL field until the value is above the threshold for the Receiver to start processing Transfer.

$I_{HFNHG}$    For the rest of this section it is assumed the Sender FIFO Low and High Tidemark interrupts are used, however, any of the other methods could be used.

## D2.3.2  Procedure

$S_{WBNDP}$    Figure D2.3 shows the sequence of events to use the streaming FIFO transport protocol for the Sender. Arrows with dotted lines indicate optional steps.
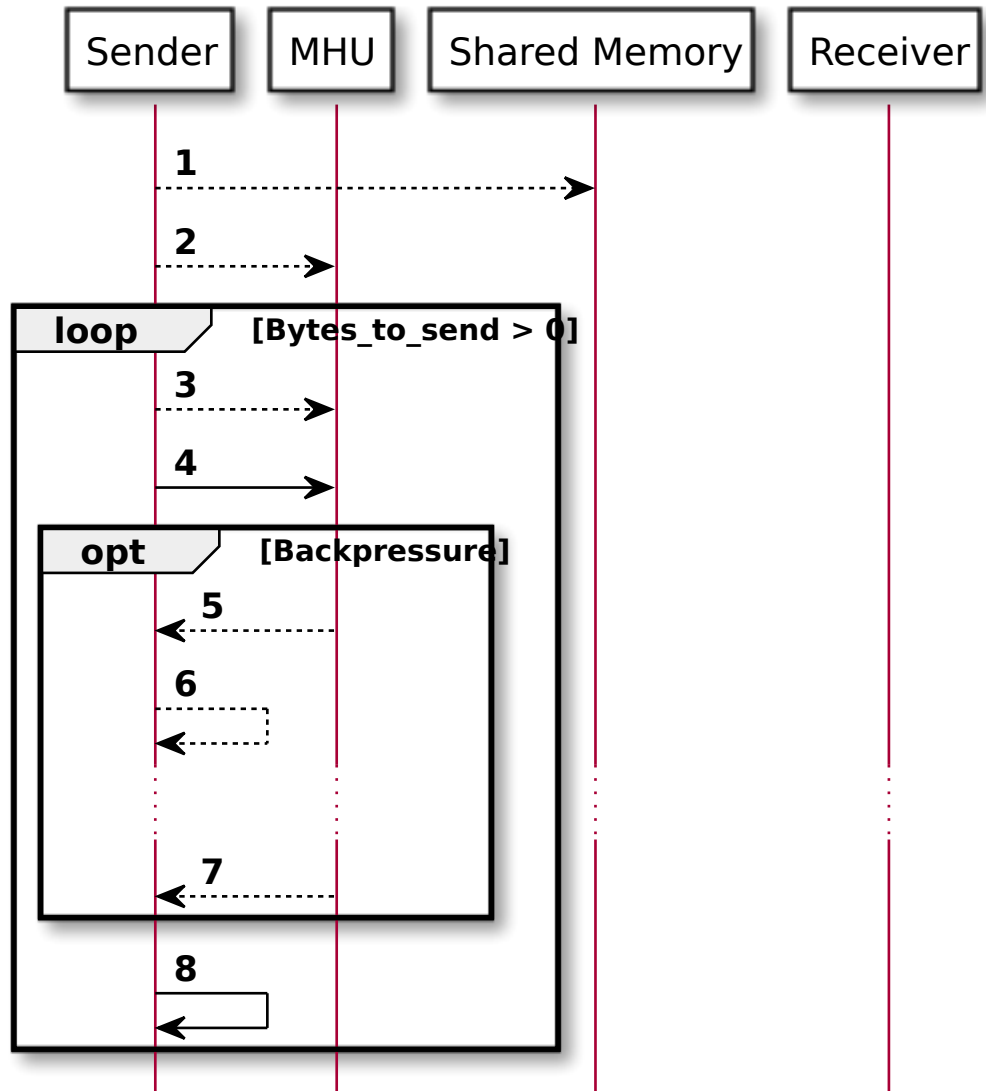
**Figure D2.3: Streaming FIFO transport protocol for Sender**

The sequence of events shown in  Figure D2.3:

1. Sender writes any out-band payload to shared memory.
2. Sender configures the Transfer Delineation Mode by writing to the PFFCW<n>_CTRL.TDM field.

   This is not required for every transfer and can be set when performing the initial configuration of the FFCH or when the conditions change to require a different mode be used.

**While Bytes to Send > 0**

3. Sender writes to the PFFCW<n>_FLG register to set the data flags correctly.

   Depending on the Transfer Delineation Mode the Sender:

   - Software Flag

     If next write to the PFFCW<n>_PAY register includes the:

     - First byte of the Transfer set the PFFCW<n>_FLG.SOT field to 0b1, otherwise set the field to 0b0.

- Last byte of the Transfer set the PFFCW<n>_FLG.EOT field to `0b1`, otherwise set the field to `0b0`.
- Last byte of the Transfer and an acknowledgement is required for this Transfer set the PFFCW<n>_FLG.ACK field to `0b1`, otherwise set it to `0b0`.

- Partial flag

  If next write to the PFFCW<n>_PAY register includes the:

  - First byte of the Transfer this step is not required.
  - First and last byte of the Transfer set the PFFCW<n>_FLG.{SOT,EOT} fields both to `0b1`.
  - Last byte but not the first byte of the Transfer set the PFFCW<n>_FLG.EOT field to `0b1`.
  - Last byte of the Transfer and an acknowledgement is required for this Transfer set the PFFCW<n>_FLG.ACK field to `0b1`, otherwise set it to `0b0`.

- Auto flag

  Sets the PFFCW<n>_FLG.ACK field to `0b1` if the Sender wants an acknowledgement for this Transfer, otherwise sets it `0b0` if the Sender does not want an acknowledgement for this Transfer.

4. Sender performs a write access to the PFFCW<n>_PAY register with the payload.

   The Sender use an access equal to the amount of data that the Sender want to push onto the FIFO. The order of the bytes in the payload written to the PFFCW<n>_PAY register depends on the setting of the PFFCW<n>_CTRL.MSBF field.

5. MHU generates a Sender FIFO High Tidemark event, when the number of valid bytes in the FIFO interrupt is greater than the PFFCW<n>_TIDE.HIGH field. The Sender FIFO High Tidemark event can become one or more interrupts depending on the settings for the channel in the Postbox:

   - Sender FIFO High Tide interrupt, if the PFFCW<n>_INT_EN.FHT is set to `0b1`.
   - Sender FFCH Combined interrupt, if the PFFCW<n>_INT_EN.FHT is set to `0b1`.
   - Postbox Combined interrupt, if the PFFCW<n>_INT_EN.FHT and PFFCW<n>_INT_EN.PBX_COMB_EN are both set to `0b1`.

6. Sender stops pushing new data into the FIFO. The Sender waits until free space in the FIFO is available by either polling the PFFCW<n>_ST.FFS or using the FIFO Low Tidemark interrupt.

7. After a period of time, the number of valid bytes in the FIFO becomes less than or equal to the value of the PFFCW<n>_TIDE.LOW field. The MHU generates a Sender FIFO Low Tide event. The Sender FIFO Low Tidemark event can become one or more interrupts depending on the settings for the FFCH in the Postbox:

   - Sender FIFO Low Tide interrupt, if the PFFCW<n>_INT_EN.FLT is set to `0b1`.
   - Sender FFCH Combined interrupt, if the PFFCW<n>_INT_EN.FLT is set to `0b1`.
   - Postbox Combined interrupt, if the PFFCW<n>_INT_EN.FLT and PFFCW<n>_INT_EN.PBX_COMB_EN are both set to `0b1`.

At this point the Sender resuming sending payload data.

8. Sender decreases the number of bytes to send by the number of bytes written to the PFFCW<n>_PAY register. If the number of bytes to send is 0 the Sender exits the loop, otherwise returns to step 3 to modifying the flags in the PFFCW<n>_FLG registers as required to send more bytes of the Transfer.

$S_{\text{HCHCQ}}$      Figure D2.4 shows the sequence of events to use the streaming FIFO transport protocol for the Receiver. Arrows with dotted lines indicate optional steps.
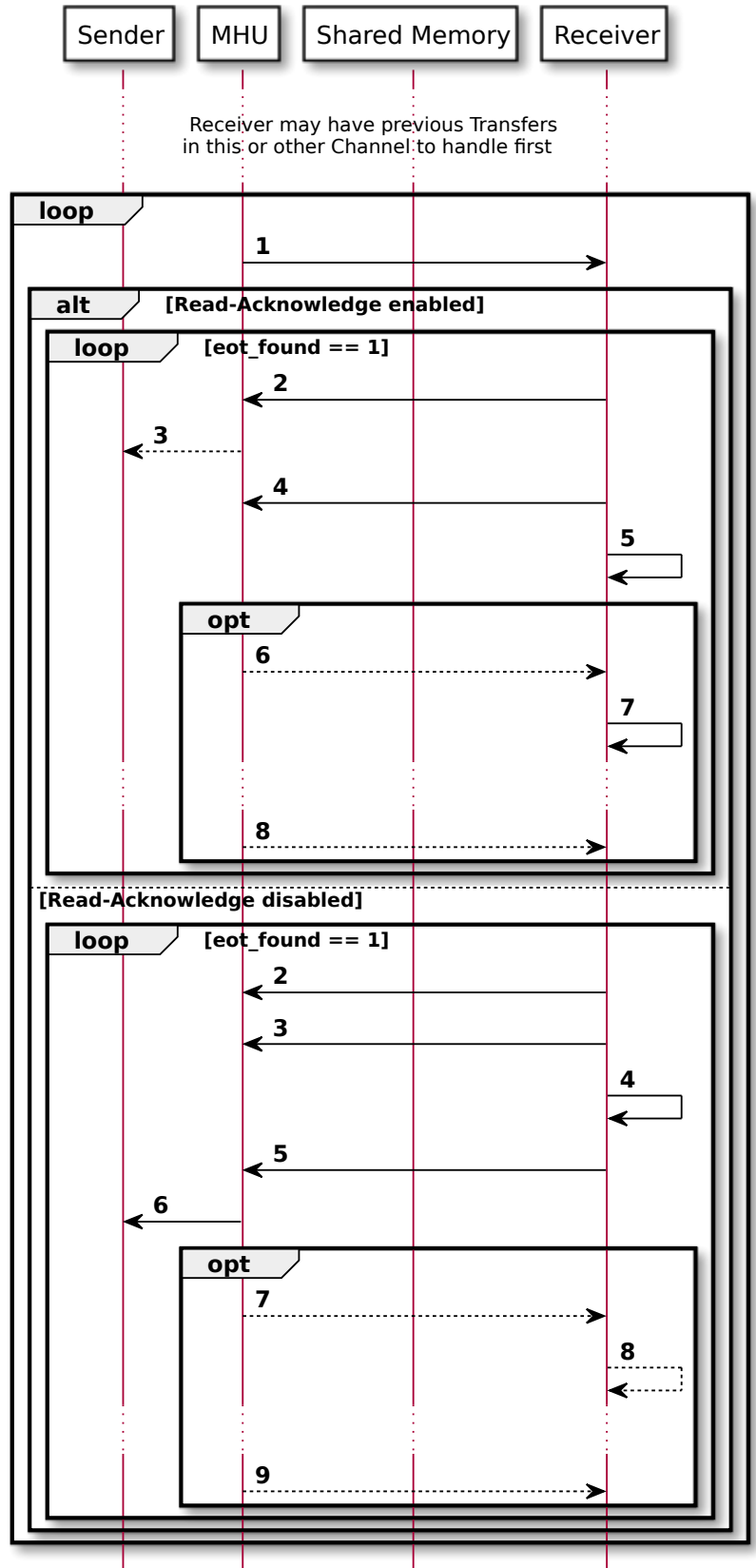
**Figure D2.4: Streaming FIFO transport protocol for Receiver**

The sequence of events shown in  Figure D2.4:

1. Receiver waits for a indication that there are enough bytes in the FIFO to being processing. The Receiver uses either the:

   - Receiver FIFO High Tidemark event.
   - Value of the FFL field in either the MFFCW<n>_FLG or MFFCW<n>_ST register.

2. MHU generates a Receiver FIFO High Tidemark event when the number of valid bytes in the FIFO is greater than the MFFCW<n>_TIDE.HIGH field. The Receiver FIFO High Tidemark event can become one or more interrupts depending on the settings for the channel in the Mailbox:

   - Receiver FIFO High Tide interrupt, if the MFFCW<n>_INT_EN.FHT is set to `0b1`.
   - Receiver FFCH Combined interrupt, if the MFFCW<n>_INT_EN.FHT is set to `0b1`.
   - Mailbox Combined interrupt, if the MFFCW<n>_INT_EN.FHT and MFFCW<n>_INT_EN.MBX_COMB_EN are both set to `0b1`.

**Read-Acknowledge enabled**

3. Receiver reads data from the MFFCW<n>_PAY register.
4. MHU generates Channel Transfer Acknowledge event for any bytes which are popped from the FIFO which have the EOT and ACK fields set to `0b1`. The Channel Transfer Acknowledge event can become one or more interrupts depending on the settings for the FFCH in the Postbox:

   - Channel Transfer Acknowledge interrupt, if the PFFCW<n>_CH_TFR_ACK is set to `0b1`.
   - Receiver FFCH Combined interrupt, if the PFFCW<n>_CH_TFR_ACK is set set to `0b1`.
   - Postbox Combined interrupt, if the PFFCW<n>_CH_TFR_ACK and PFFCW<n>_INT_EN.PBX_COMB_EN are both set to `0b1`.

5. Receiver reads the flag information from the MFFCW<n>_FLG register.
6. Receiver checks the flag information for each byte read from the MFFCW<n>_FLG register.

   Bytes read from the MFFCW<n>_PAY register are treated as defined in  Table D2.2.

7. If the number of bytes in the FIFO becomes less than or equal to the value of the MFFCW<n>_TIDE.LOW field, the MHU generates a Receiver Low Tide event. The Receiver Low Tide event can become one or more interrupts depending on the settings for the FFCH in the Mailbox:

   - Receiver FIFO Low Tide interrupt, if the MFFCW<n>_INT_EN.FLT is set to `0b1`.
   - Receiver FFCH Combined interrupt, if the MFFCW<n>_INT_EN.FLT is set to `0b1`.
   - Mailbox Combined interrupt, if the MFFCW<n>_INT_EN.FLT and MFFCW<n>_INT_EN.MBX_COMB_EN are both set to `0b1`.

8. Receiver stops popping new data from the FIFO. The Receiver returns to steps 2 and waits until enough valid data is available in the FIFO.

**Read-Acknowledge disabled**

3. Receiver reads data from the MFFCW<n>_PAY register.
4. Receiver reads the flag information from the MFFCW<n>_FLG register.
5. Receiver checks the flag information for each byte read from the MFFCW<n>_FLG register.

   Bytes read from the MFFCW<n>_PAY register are treated as defined in  Table D2.2.

6. Receiver writes to the MFFCW<n>_FIFO_POP register. The value of the write must be equal to the number of bytes the Receiver wants to pop from the FIFO.

7. MHU generates Channel Transfer Acknowledge event for any bytes which are popped from the FIFO which have the EOT and ACK fields set to `0b1`. The Channel Transfer Acknowledge event can become one or more interrupts depending on the settings for the FFCH in the Postbox:

   - Channel Transfer Acknowledge interrupt, if the PFFCW<n>_CH_TFR_ACK is set to `0b1`.
   - Receiver FFCH Combined interrupt, if the PFFCW<n>_CH_TFR_ACK is set set to `0b1`.

- Postbox Combined interrupt, if the PFFCW<n>_CH_TFR_ACK and PFFCW<n>_INT_EN.PBX_COMB_EN are both set to `0b1`.

8. If the number of bytes in the FIFO becomes less than or equal to the value of the MFFCW<n>_TIDE.LOW field, the MHU generates a Receiver Low Tide event. The Receiver Low Tide event can become one or more interrupts depending on the settings for the FFCH in the Mailbox:

- Receiver FIFO Low Tide interrupt, if the MFFCW<n>_INT_EN.FLT is set to `0b1`.
- Receiver FFCH Combined interrupt, if the MFFCW<n>_INT_EN.FLT is set to `0b1`.
- Mailbox Combined interrupt, if the MFFCW<n>_INT_EN.FLT and MFFCW<n>_INT_EN.MBX_COMB_EN are both set to `0b1`.

9. Receiver stops popping new data from the FIFO. The Receiver returns to steps 2 and waits until enough valid data is available in the FIFO.

$I_{HSXRQ}$    The sequences in Figure D2.3 and Figure D2.4 can be run concurrently or sequentially. It is even possible that the Sender is performing the sequences for Transfer Y whilst the Receiver is performing the sequences for Transfer X, where Transfer X is sent by before Transfer Y.

## D2.4 Last-value

$I_{QHWFR}$   The Last-value transport protocol enables a Sender to send a value to the a Receiver which can be updated at any point, even if the Receiver has not seen a previously written value.

$I_{GTQWX}$   The value seen by the Receiver can be either the new or old value when the Sender updates the data value.

### D2.4.1 Requirements

$I_{TDMNQ}$   This section covers the requirements of the last value transport protocol.

$R_{LGYHM}$   The last-value transport protocol must only be used with FCH.

$R_{CWVBR}$   Only in-band data is supported up to the Fast Channel word-size.

$R_{JLSXH}$   Both the Sender and Receiver systems must support a single-copy atomic size greater than or equal to the Fast Channel word-size.

$S_{PPYYS}$   All updates to a FCH must be performed atomically, otherwise it is UNPREDICTABLE what value the Receivers will read.

$S_{YLRFM}$   It is software IMPLEMENTATION DEFINED:

   • The value of the data sent by the Sender to the Receiver.
   • When the Sender is allowed to update the value of the data.

$S_{XLBVH}$   Sender and Receiver must be able to handle all the following scenarios:

   • Sender writes the FCH once, with data D1, after which the Receiver then reads the FCH and gets value D1.
   • Sender writes the FCH twice, with data D1 and D2, and the Receiver reads either D1 or D2 depending on the sequence of the Sender writes and Receiver reads.

### D2.4.2 Procedure

$S_{XJPQT}$   In  Figure D2.5 shows the sequence of events to use the Last-value transport protocol when only sending a single-word of in-band data. Arrows with dotted lines indicate optional steps.
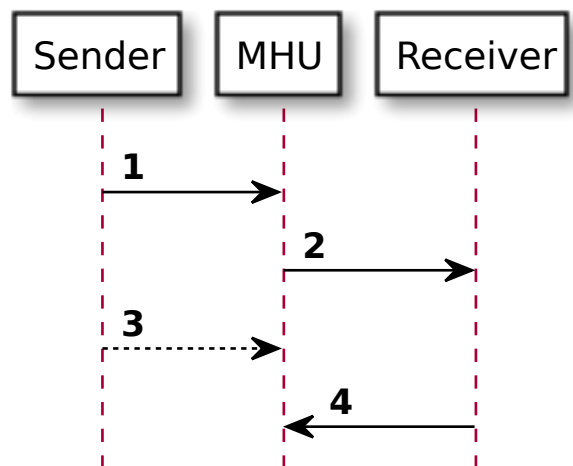


**Figure D2.5: Last-value transport protocol**

The sequence of events are:

1. Sender atomically writes to the in-band data value to the PFCW<n>_PAY registers atomically.
2. MHU generates Channel Transfer event. The Channel Transfer event can become one or more of the following interrupts depending on the Mailbox configuration:

   - Fast Channel Transfer interrupt, if MBX_FCH_CTRL.INT_EN is set to `0b1`.
   - Fast Channel Group Transfer <m> interrupt, if MBX_FCH_CTRL.INT_EN is set to `0b1`.
   - Mailbox Combined interrupt, if MBX_FCH_CTRL.INT_EN and MBX_FCG_INT_EN.MBX_COMB_EN<m> are both set to `0b1`.

3. Sender atomically updates the in-band data value, no interrupt is generated as Receiver hasn't acknowledged the previous write.
4. Receiver reads the MFCW<n>_PAY registers atomically to get value of in-band data.

Where <m> is the FCG the FCH is part of.

$I_{SJTZX}$   Step 3 is optional and can happen at any time:

- If it happens before step 4 then the in-band value is updated.
- If it happens after step 4 then a new Transfer is triggered.
- If it happens at the same time as step 4, the Receiver reads the old value and a new Transfer is started for the new value.

$I_{RPFKH}$   The steps performed by the Sender and Receiver in Figure D2.5 can be run concurrently or sequentially. It is even possible that the Sender is performing the sequences for Transfer Y whilst the Receiver is performing the sequences for Transfer X, where Transfer X is sent by before Transfer Y.

# Glossary

**Agent**

Generic term used to refer to a software entity which sends or receives a Transfer.

**Channel**

Generic term for the resource which a Sender uses to send a Transfer to the Receiver.

**Doorbell Channel (DBCH)**

A Doorbell Channel is a type of channel, which is used to send Transfers which are a single event to the Receiver.

**Fast Channel (FCH)**

A Fast Channel is a type of channel, which is used to send a Transfer to the Receiver.

**Fast Channel Group (FCG)**

A group of 1-32 FCH.

**Fast Channel word-size**

The size of data payload of an Fast Channel

**FIFO Channel (FFCH)**

A FIFO Channel is a type of channel, which is used to send one or more Transfers, where each Transfers can be 32-bit or 64-bits and can be any value.

**Flag History Buffer (FHB)**

Buffer used to record the flags of the bytes read from the FIFO of an FFCH requested by the last read by the Receiver

**Illegal Access**

An access which has incorrect security to access the register  C1.1.5 *Illegal Accesses*

**In-band**

A transfer payload sent using a channel of the MHU.

**Inter-Process Communication (IPC)**

A method that allows two separate processes to communicate with one another, using interrupts to indicate when there is a new communication or an update about the communication.

**Inter-Processor Interrupt (IPC)**

An interrupt that is used to indicates to another processor that the interrupting processor requires action from the other processor.

**Mailbox (MBX)**

A Mailbox is collection of channels and control registers, used by the Receiver to receiver a Transfer from the Sender.

**Message**

A message is a self-contain entity which is sent by a Sender to a Receiver. A Message is formed of one or more Transfers.

**Message protocol**

    A protocol which defines the format of messages that are sent between Sender and Receiver.

**MHU Receiver (MHUR)**

    The side of the MHU used by the Receiver of the message.

**MHU Sender (MHUS)**

    The side of the MHU used by the Sender of the message.

**Non-operational State**

    MHU Sender or Receiver is in a state where it cannot send or receiver Transfer

**Operational State**

    MHU Sender or Receiver is in a state where it can send or receiver Transfers

**Out-band**

    A transfer payload sent using a method other than a channel of the MHU, for example shared memory.

**Postbox (PBX)**

    A Postbox is a collection of channels and control registers, used by the Sender to send a Transfer to the Receiver.

**Receiver Agent**

    Piece of software receiving the message, referred to as the Receiver for short.

**Receiver Security Control (RSC)**

    Security control block which controls security of the MHUR

**RES0**

    A Reserved bit. Used for fields for register descriptions, and for fields in architecturally-defined data structures that are held in memory. For a full definition refer to the Arm ARM [1]

**RES1**

    A Reserved bit. Used for fields for register descriptions, and for fields in architecturally-defined data structures that are held in memory. For a full definition refer to the Arm ARM [1]

**Security Assignment Agent**

    Piece of software which assigns the resources of the MHU to different Security Groups.

**Sender Agent**

    Piece of software sending the message, referred to as the Sender for short.

**Sender Security Control (SSC)**

    Security control block which controls security of the MHUS

**System Control Processor**

    A processor that controls the clock, power and resets domains of the SoC.

**Transfer**

    A transfer is defined as a single interrupt generated by the MHU. A transfer can be nothing more than an interrupt, but it can also have an associated data payload (transfer payload).

**Transport protocol**

    A protocol which defines how transfers are indicated by the Sender to the Receiver, using the MHU, and the location of the transfer payload, if applicable.

---

**Unsupported Access**

An access which is of an unsupported alignment, size or memory type