KeyStone II Architecture
Debug and Trace

User Guide

Literature Number: SPRUHM4
July 2013

Release History

Release	Date	Chapter/Topic	Description/Comments
SPRUHM4	July 2013	All	Initial Release

Contents

Release History ..... *ø-ii*
List of Tables ..... *ø-vii*
List of Figures ..... *ø-ix*

Preface ø-xi
About This Manual ..... ø-xi
Notational Conventions ..... ø-xi
Related Documentation from Texas Instruments ..... ø-xii
Trademarks ..... ø-xii

Chapter 1

Introduction to Debug Architecture 1-1
1.1 Overview 1-2
1.2 Terminology 1-2
1.3 Features 1-3
1.4 The Debug Subsystem 1-5
1.4.1 Debug Pin Manager 1-6
1.5 Trace Components 1-9
1.6 System Trace Module 1-10
1.7 TI Embedded Trace Buffer (TETB) 1-10
1.7.1 Supported Modes 1-10
1.7.2 Applications 1-10
1.7.3 Message Format 1-11
1.8 SoC Cross-Triggering 1-12
1.8.1 Cross-Triggering with DSP Core Trace 1-14
1.8.2 Cross-Triggering with System Trace 1-14
1.8.2.1 Start Collecting Software Instrumentation Data Upon External Trigger 1-15
1.8.2.2 Stop Collecting Software Instrumentation Data Upon External Trigger 1-15
1.8.2.3 Start Event Monitoring Upon External Trigger 1-15
1.8.2.4 Stop Event Monitoring Upon External Trigger 1-16
1.9 The Suspend Mechanism 1-17
1.10 Application Integration 1-17
1.11 Board Design Integration 1-17

Chapter 2

DSP Debug 2-1
2.1 Overview 2-2
2.2 Halt Mode Debug 2-2
2.3 Real-Time Debug 2-3
2.4 Monitor Mode (Run Mode) Debug 2-3
2.5 Trace 2-4
2.6 Trace Port Width Configuration 2-4
2.7 Tuning Export Clock Frequency 2-5
2.7.1 Tuning Export Clock Frequency Through PLL 2-5
2.8 Trace Pin Configuration 2-5
2.9 Advanced Event Triggering 2-5

Chapter 3

ARM Debug 3-1
3.1 Overview . . . 3-2
3.2 Debug Modes . . . 3-2
3.3 Processor Trace Macro (PTM) . . . 3-2
3.4 CoreSight System Trace Macro (STM) . . . 3-3
3.5 Performance Monitor Unit (PMU) . . . 3-3
3.6 Cross Trigger Matrix (CTM) . . . 3-4

Chapter 4

System Trace 4-1
4.1 Overview . . . 4-2
4.2 DebugSS Trace and Monitoring . . . 4-4
4.3 Trace Port Width Configuration . . . 4-4
4.4 Tuning Export Clock Frequency . . . 4-4
4.5 Software Messages . . . 4-5
4.6 Hardware Messages . . . 4-5
4.7 Overflow . . . 4-5
4.8 Reset . . . 4-5
4.9 Message Interleaving . . . 4-6
4.10 Master ID Encoding . . . 4-6
4.11 Timestamping . . . 4-6
4.12 Message Format . . . 4-7

Chapter 5

Tracer Architecture and Operation 5-1
5.1 Purpose of the Tracer . . . 5-2
5.2 Features . . . 5-2
5.3 Tracer Architecture . . . 5-3
5.3.1 Hardware Instrumentation through CPTracers . . . 5-5
5.3.2 Event Interface . . . 5-7
5.3.3 Software Instrumentation . . . 5-7
5.3.4 Master ID for HW and SW Messages . . . 5-8
5.4 CBA Event Generation . . . 5-9
5.4.1 Event A - Master Request Event . . . 5-9
5.4.2 Event B - Arbitration Event . . . 5-9
5.4.3 Event C - Write Completion Event . . . 5-10
5.4.4 Event E - Read Completion Event . . . 5-10
5.4.5 Event F - Write Merge Event . . . 5-10
5.4.6 Event G - Read Discard Event . . . 5-10
5.5 Transaction Flow . . . 5-10
5.6 Tracer Connections . . . 5-13
5.7 FIFOs . . . 5-13
5.8 Trace Message Formats . . . 5-13
5.8.1 Status Messages . . . 5-13
5.8.2 Event Message . . . 5-14
5.8.3 Statistics Message . . . 5-15
5.8.4 STM Message . . . 5-15
5.9 CBA Counters and Statistics . . . 5-16
5.10 Throughput Counters . . . 5-16
5.11 Sliding Time Window . . . 5-16
5.12 Cross Triggering . . . 5-17

5.13 Filtering 5-17
5.14 Accumulated Wait Time Counter 5-17
5.15 Num Grant Counter 5-17
5.16 Overflow Status 5-17

Chapter 6

Programming Considerations 6-1
6.1 Programming Overview 6-2
6.2 STM Configuration 6-2
6.3 ETB Configuration 6-2
6.4 Tracer Configuration 6-2

Chapter 7

Registers 7-1
7.1 Debug Resource Manager Registers 7-2
7.1.1 Peripheral ID Register 7-2
7.1.2 DRM Configuration Register 7-3
7.1.3 DPM Claim Register 7-3
7.1.4 DPM Control Register N 7-4
7.2 Tracer Registers 7-5
7.2.1 Identification and Version Register 7-6
7.2.2 Transaction Qualifier Register 7-6
7.2.3 Module Control and Status Register 7-8
7.2.4 Sliding Time Window 7-9
7.2.5 Master ID Select Register A for Throughput0 and Event Trace 7-10
7.2.6 Master ID Select Register N for Throughput1 Calculation 7-11
7.2.7 Address Window Registers 7-12
7.2.8 Access Status Register 7-13
7.2.9 Address Mask Register 7-14
7.2.10 Destination Address Register 7-14
7.2.11 Message Priority Register 7-15
7.2.12 Ownership 7-15
7.2.13 Throughput0 7-16
7.2.14 Throughput1 7-16
7.2.15 Accumulated Wait Time 7-16
7.2.16 Number of Grants 7-17
7.2.17 Interrupt Raw Status 7-17
7.2.18 Interrupt Masked Status 7-18
7.2.19 Interrupt Mask Set 7-18
7.2.20 Interrupt Mask Clear 7-19
7.2.21 End of Interrupt (EOI) Register 7-19
7.3 STM Registers 7-20
7.3.1 Lock Access Register 7-20
7.3.2 Lock Status Register 7-20
7.3.3 Software Masters Control Register 0 – STM_SWMCTRL0 7-21
7.3.4 Software Masters Control Register 1 – STM_SWMCTRL1 7-22
7.3.5 Software Masters Control Register 2 – STM_SWMCTRL2 7-22
7.3.6 Software Masters Control Register 3 – STM_SWMCTRL3 7-23
7.3.7 Software Masters Control Register 4 – STM_SWMCTRL4 7-24
7.3.8 Hardware Masters Control Register 7-24
7.3.9 PTI Configuration Register 7-25
7.3.10 PTI Count Down Register 7-26
7.3.11 ATB Configuration Register 7-26
7.4 TETB Registers 7-27
7.4.1 Control Register (CTL) 7-27

7.4.2 TI Specific Control Register (TCTL), 0xC20 7-27
7.4.3 RAM Read Data Register (RRD) 7-28
7.4.4 RAM Read Pointer Register (RRP) 7-28

Chapter 8

Programming Guidelines 8-1
8.1 Application Support 8-2
8.2 Programming Overview 8-2
8.2.1 DSP Core Trace Specific Configuration 8-2
8.2.2 AET Specific Configuration 8-2
8.2.3 STM Specific Configuration 8-2
8.2.4 Debug Pin Manager Configuration 8-3

Index IX-1

List of Tables

Table 1-1 Terminology 1-2
Table 1-2 Emulation Interface with Different Debug Port Configurations 1-6
Table 1-3 Debug Pin Configurations 1-8
Table 4-1 STM Channel Memory Space 4-6
Table 4-2 STM Message Format 4-7
Table 5-1 MSTID mapping for Hardware Instrumentation (CPTRACERS) 5-8
Table 5-2 MSTID Mapping for Software Messages 5-8
Table 5-3 Tracer Events 5-9
Table 7-1 DRM Register Offsets 7-2
Table 7-2 Identification and Version Register Field Definitions 7-2
Table 7-3 DRM Configuration Register Field Definitions 7-3
Table 7-4 DPM Claim Register Field Definitions 7-3
Table 7-5 DPM Control Register N 7-4
Table 7-6 Tracer Register Offsets 7-5
Table 7-7 Identification and Version Register Field Definitions 7-6
Table 7-8 Transaction Qualifier Register Field Definitions 7-7
Table 7-9 Module Control and Status Register Field Definitions 7-8
Table 7-10 Sliding Time Window Register Field Definitions 7-9
Table 7-11 Master ID Select Group A Register Field Definitions 7-10
Table 7-12 Master ID Select Group B Register Field Definitions 7-10
Table 7-13 Master ID Select Group C Register Field Definitions 7-10
Table 7-14 Master ID Select Group D Field Definitions 7-11
Table 7-15 Master ID Select Group A Register Field Definitions 7-11
Table 7-16 Master ID Select Group B Register Field Definitions 7-11
Table 7-17 Master ID Select Group C Register Field Definitions 7-12
Table 7-18 Master ID Select Group D Register Field Definitions 7-12
Table 7-19 Start Address Register Field Definitions 7-12
Table 7-20 End Address Register Field Definitions 7-13
Table 7-21 Access Status Register Field Definitions 7-13
Table 7-22 Access Status Pacing Register Field Definitions 7-14
Table 7-23 Address Mask Register Field Definitions Field Definitions 7-14
Table 7-24 Destination Address Register Field Definitions 7-15
Table 7-25 Message Priority Register Field Definitions 7-15
Table 7-26 Ownership Register Field Definitions 7-15
Table 7-27 Throughput0 Register Field Definitions 7-16
Table 7-28 Throughput1 Register Field Definitions 7-16
Table 7-29 Accumulated Wait Time Field Definitions 7-17
Table 7-30 Number of Grants Register Field Definitions 7-17
Table 7-31 Interrupt Raw Status Register Field Definitions 7-17
Table 7-32 Interrupt Masked Status Register Field Definitions 7-18
Table 7-33 Interrupt Mask Set Register Field Definitions 7-18
Table 7-34 Interrupt Mask Clear Register Field Definitions 7-19
Table 7-35 End of Interrupt (EOI) Register Field Definitions 7-19
Table 7-36 STM Register Offsets 7-20
Table 7-37 Lock Access Register Field Definitions 7-20
Table 7-38 Lock Status Register Field Definitions 7-21
Table 7-39 Software Master Control Register 0 Field Definitions 7-21
Table 7-40 Software Master Control Register 1 Field Definitions 7-22
Table 7-41 Software Master Control Register 2 Field Definitions 7-22
Table 7-42 Software Master Control Register 3 Field Definitions 7-23
Table 7-43 Software Master Control Register 4 Field Definitions 7-24
Table 7-44 Hardware Master Control Register Field Definitions 7-24

Table 7-45 PTI Configuration Register Field Definitions 7-25
Table 7-46 PTI Count Down Register Field Definitions. 7-26
Table 7-47 ATB Configuration Register Field Definitions 7-26
Table 7-48 TETB Register Offsets 7-27
Table 7-49 Control Register Field Definitions 7-27
Table 7-50 TI Specific Control Register (TCTL) Field Definitions 7-27
Table 7-51 RAM Read Data (RRD) Register Field Definitions 7-28
Table 7-52 RAM Read Pointer (RRP) Register Field Definitions 7-28

List of Figures

Figure 1-1 Debug Subsystem Overview 1-5
Figure 1-2 Trace Components 1-9
Figure 1-3 TETB Message Format 1-11
Figure 1-4 SoC Cross Trigger Architecture 1-13
Figure 3-1 SoC Cross Trigger Architecture 3-5
Figure 4-1 SoC Trace Architecture 4-3
Figure 5-1 Tracer Connection 5-3
Figure 5-2 CBA Event Connections 5-4
Figure 5-3 Switch Fabric with CP_Tracer 5-6
Figure 5-4 Tracer Generation 5-10
Figure 5-5 Example—Event A to Tracer 5-11
Figure 5-6 Example—Event B to Tracer 5-12
Figure 5-7 Status Message Format 5-13
Figure 5-8 Status Message Format 5-14
Figure 5-9 Event Message Format 5-14
Figure 5-10 Statistic Message Format 5-15
Figure 5-11 STM Message Format 5-15
Figure 7-1 Peripheral ID Register 7-2
Figure 7-2 DRM Configuration Register 7-3
Figure 7-3 DPM Claim Register 7-3
Figure 7-4 DPM Control Register N 7-4
Figure 7-5 Identification and Version Register 7-6
Figure 7-6 Transaction Qualifier Register 7-6
Figure 7-7 Module Control and Status Register 7-8
Figure 7-8 Sliding Time Window Register 7-9
Figure 7-9 Master ID Select Group A Register 7-10
Figure 7-10 Master ID Select Group B Register 7-10
Figure 7-11 Master ID Select Group C Register 7-10
Figure 7-12 Master ID Select Group D Register 7-11
Figure 7-13 Master ID Select Group A Register 7-11
Figure 7-14 Master ID Select Group B Register 7-11
Figure 7-15 Master ID Select Group C Register 7-12
Figure 7-16 Master ID Select Group D Register 7-12
Figure 7-17 Start Address Register 7-12
Figure 7-18 End Address Register 7-13
Figure 7-19 Access Status Register 7-13
Figure 7-20 Access Status Pacing Register 7-13
Figure 7-21 Address Mask Register 7-14
Figure 7-22 Destination Address Register 7-14
Figure 7-23 Message Priority Register 7-15
Figure 7-24 Ownership Register 7-15
Figure 7-25 Throughput0 Register 7-16
Figure 7-26 Throughput1 Register 7-16
Figure 7-27 Accumulated Wait Time 7-16
Figure 7-28 Number of Grants Register 7-17
Figure 7-29 Interrupt Raw Status Register 7-17
Figure 7-30 Interrupt Masked Status Register 7-18
Figure 7-31 Interrupt Mask Set Register 7-18
Figure 7-32 Interrupt Mask Clear Register 7-19
Figure 7-33 End of Interrupt (EOI) Register 7-19
Figure 7-34 Lock Access Register 7-20
Figure 7-35 Lock Status Register 7-20

Figure 7-36 Software Master Control Register 0. 7-21
Figure 7-37 Software Master Control Register 1. 7-22
Figure 7-38 Software Master Control Register 2. 7-22
Figure 7-39 Software Master Control Register 3. 7-23
Figure 7-40 Software Master Control Register 4. 7-24
Figure 7-41 Hardware Master Control Register 7-24
Figure 7-42 PTI Configuration Register 7-25
Figure 7-43 PTI Count Down Register 7-26
Figure 7-44 ATB Configuration Register 7-26
Figure 7-45 Control Register 7-27
Figure 7-46 TI Specific Control Register (TCTL) 7-27
Figure 7-47 RAM Read Data (RRD) Register 7-28
Figure 7-48 RAM Read Pointer (RRP) Register 7-28

Preface

About This Manual

This user guide describes the capabilities of the trace features available through the debug architecture on KeyStone devices. Trace information can be gathered at the DSP core level or at the system level. The Debug Subsystem captures and exports trace data for both levels of trace. Trace is implemented as a non-intrusive debug tool within the KeyStone architecture, but can be selected to operate in both intrusive and non-intrusive mode depending on the amount of data the user wants to export.

Notational Conventions

This document uses the following conventions:

- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Terminal sessions and information the system displays are in `screen font`.
- Information you must enter is in **`boldface screen font`**.
- Elements in square brackets ([ ]) are optional.

Notes use the following conventions:

Note—Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

CAUTION—Indicates the possibility of service interruption if precautions are not taken.

WARNING—Indicates the possibility of damage to equipment if precautions are not taken.

Related Documentation from Texas Instruments

Antenna Interface 2 (AIF2) for KeyStone Devices User Guide	SPRUGV7
C66x CorePac User Guide	SPRUGW0
DDR3 Memory Controller for KeyStone Devices User Guide	SPRUGV8
External Memory Interface (EMIF16) for KeyStone Devices User Guide	SPRUGZ3
Fast Fourier Transform Coprocessor (FFTC) for KeyStone Devices User Guide	SPRUGS2
HyperLink for KeyStone Devices User Guide	SPRUGW8
Multicore Navigator for KeyStone Devices User Guide	SPRUGR9
Multicore Shared Memory Controller (MSMC) for KeyStone Devices User Guide	SPRUGW7
Packet Accelerator (PA) for KeyStone Devices User Guide	SPRUGS4
Peripheral Component Interconnect Express (PCIe) for KeyStone Devices User Guide	SPRUGS6
Phase Locked Loop (PLL) Controller for KeyStone Devices User Guide	SPRUGV2
Serial RapidIO (SRIO) for KeyStone Devices User Guide	SPRUGW1

Trademarks

Chapter 1

Introduction to Debug Architecture

1.1 "Overview" on page 1-2
1.2 "Terminology" on page 1-2
1.3 "Features" on page 1-3
1.4 "The Debug Subsystem" on page 1-5
1.5 "Trace Components" on page 1-9
1.6 "System Trace Module" on page 1-10
1.7 "TI Embedded Trace Buffer (TETB)" on page 1-10
1.8 "SoC Cross-Triggering" on page 1-12
1.9 "The Suspend Mechanism" on page 1-17
1.10 "Application Integration" on page 1-17
1.11 "Board Design Integration" on page 1-17

1.1 Overview

This user guide describes the capabilities of the various features available through the debug architecture on KeyStone II devices.

The SoC debug capabilities of KeyStone II devices are centered around the Debug subsystem module (DEBUGSS). The DEBUGSS module contains the ICEPick module which handles the external JTAG TAP and multiple secondary TAPs for the various processing cores of the device. It also provides DAP port for system wide memory access from debugger, Cross triggering, System trace, Peripheral suspend generation, Debug port (EMUx) pin management etc. The DEBUGSS module works in conjunction with the debug capability integrated in the processing cores (ARM CorePac and DSP CorePacs) to provide a comprehensive hardware platform for a rich debug and development experience.

The debug architecture allows the user to gather various types of trace data. Trace information can be gathered at the DSP core level or at the system level; the information records program flow, memory references, and application-specific data with timestamps. The Debug Subsystem captures and exports trace data for both levels of trace. Trace is implemented as a non-intrusive debug tool within the KeyStone architecture, but can be selected to operate in both intrusive and non-intrusive mode according to the amount of data the user wants to export. Within DSP Core Trace, there are Advanced Event Triggering facilities that allow the user to restrict the exported trace data to data of interest only; this minimizes intrusiveness.

Trace data is exported in packets of 20 bits from the debug port at the CPU rate. This allows all or some of the debug port pins to be allocated to trace. Therefore, the debug port pins can be shared between trace and other functions, with the number of pins allocated to trace and the corresponding trace port width varied at runtime.

TI provides a set of libraries known as CToolsLib that allows for easy integration of trace capabilities into existing software. For more information about CToolsLib, see Chapter 8 "Programming Guidelines" on page 8-1.

1.2 Terminology

Table 1-1 lists important acronyms and abbreviations used in this document.

Table 1-1 Terminology

Term	Definition
AET	Advance Event Trigger
ATB	Advanced Trace Bus
CBA	Central Bus Architecture
DAP	Debug Access Port
MPAX	Memory Protection and Address Extension
PDSP	Parallel DSP
PTI	Parallel Trace Interface
QMSS	Queue Manager SubSystem
STM	System Trace Module
TAP	TI Test Access Port
TBR	TI Trace Buffer and Router
TETB	TI Embedded Trace Buffer

1.3 Features

The debug architecture provides the following trace and cross triggering features at the SoC, ARM and C66x levels:

- SoC Features
  - Support for 1149.1(JTAG + Boundary scan) and 1149.6 (Boundary scan extensions). No support for 1149.7 (cJTAG)
  - Trace sources to DEBUGSS STM
    - Provides a way for hardware instrumentation and software messaging to supplement the processor core trace mechanisms.
    - Hardware instrumentation support of CPTracers to support logging of bus transactions for critical endpoints
    - Software messaging/instrumentation support for DSP and QMSS PDSP cores through DEBUGSS STM. ARM CorePac has support for this internally
  - Trace Sinks
    - Support for trace export (from all processor cores and DEBUGSS STM) through emulation pins (34 pin interface 1.8v LVCMOS interface). Concurrent trace of DSP and STM traces or ARM and STM traces via EMU pins is possible. Note that concurrent trace export of DSP and ARM is not supported (via EMU pins)
    - Support for 32KB DEBUGSS TBR to hold system trace. The data can be drained using EDMA to on-chip or DDR memory buffers. These intermediate buffers can subsequently be drained through the SoC high speed interfaces. The DEBUGSS TBR is dedicated to the DEBUGSS STM module
  - Cross triggering: Provides a way to propagate debug (trigger) events from one processor/subsystem/module to another
    - Cross triggering between multiple devices via EMU0/EMU1 pins
    - Cross triggering between multiple processing cores within the SoC like ARM/DSP CorePacs and non-processor entities like ARM STM (input only), CPTracers, CT-TBRs and DEBUGSS STM (input only)
  - Synchronized starting and stopping of processing cores
    - Global start of all ARM CorePac
    - Global start of all DSP CorePacs
    - Global stopping of all ARM and DSP CorePacs
  - Emulation mode aware peripherals (suspend features and debug access features)
  - Support system memory access via the DAP port (natively support 32 bit address, and it can support 36 bit address through configuration of MPAX inside MSMC). Debug access to any invalid memory location (reserved/clock-gated/power-down) shall not cause system hang.
  - Support WIR (wait-in-reset) debug boot mode for Non-secure devices.
  - Debug functionality shall survive all pin resets except power-on resets ($\overline{\text{POR}}$/$\overline{\text{RESETFULL}}$) and test reset ($\overline{\text{TRST}}$)
  - PDSP Debug features like access/control through DAP, Halt mode debug and software instrumentation (only QMSS PDSPs)

- ARM subsystem features
  - Support for Invasive debug like Halt mode debugging (breakpoint, watchpoints) and Monitor mode debugging
  - Support for Non-invasive debugging (Program trace, Performance Monitoring)
  - Support for A15 Performance Monitoring Unit (Cycle counters)
  - Support for per core CoreSight™ Program Trace Module (CS-PTM) with timing
  - Support for an integrated CoreSight™ System Trace Module (CS-STM) for hardware event and software instrumentation
  - A shared timestamp counter for all ARM cores and STM is integrated in ARMSS for trace data correlation
  - Support for a 16KB TBR to hold PTM/STM trace. The trace data is copied by EDMA to external memory for draining by SoC high speed serial interfaces.
  - Support for simultaneous draining of trace stream through EMUx pins and TBR to achieve higher aggregate trace throughput
  - Support for Debug authentication interface to disable debug accesses in secure devices
  - Support for Cross triggering between MPU cores, CS-STM and CT-TBR
  - Support for debug through warm reset
- DSP Features
  - Support for Halt-mode debug
  - Support for Real-time debug
  - Support for Monitor mode debug
  - Advanced Event Triggering for data/program counter (PC) watch-points, event monitoring and visibility into external events
  - Support for PC/Timing/Data/Event trace.
  - TI Embedded Trace Buffer of 4KB to store PC/Timing/Data/Event trace. The trace data is copied by EDMA to external memory for draining by SoC high speed serial interfaces or it can be drained through EMUx pins
  - Support for Cross triggering source/sink to other DSPs and SoC subsystems

1.4 The Debug Subsystem

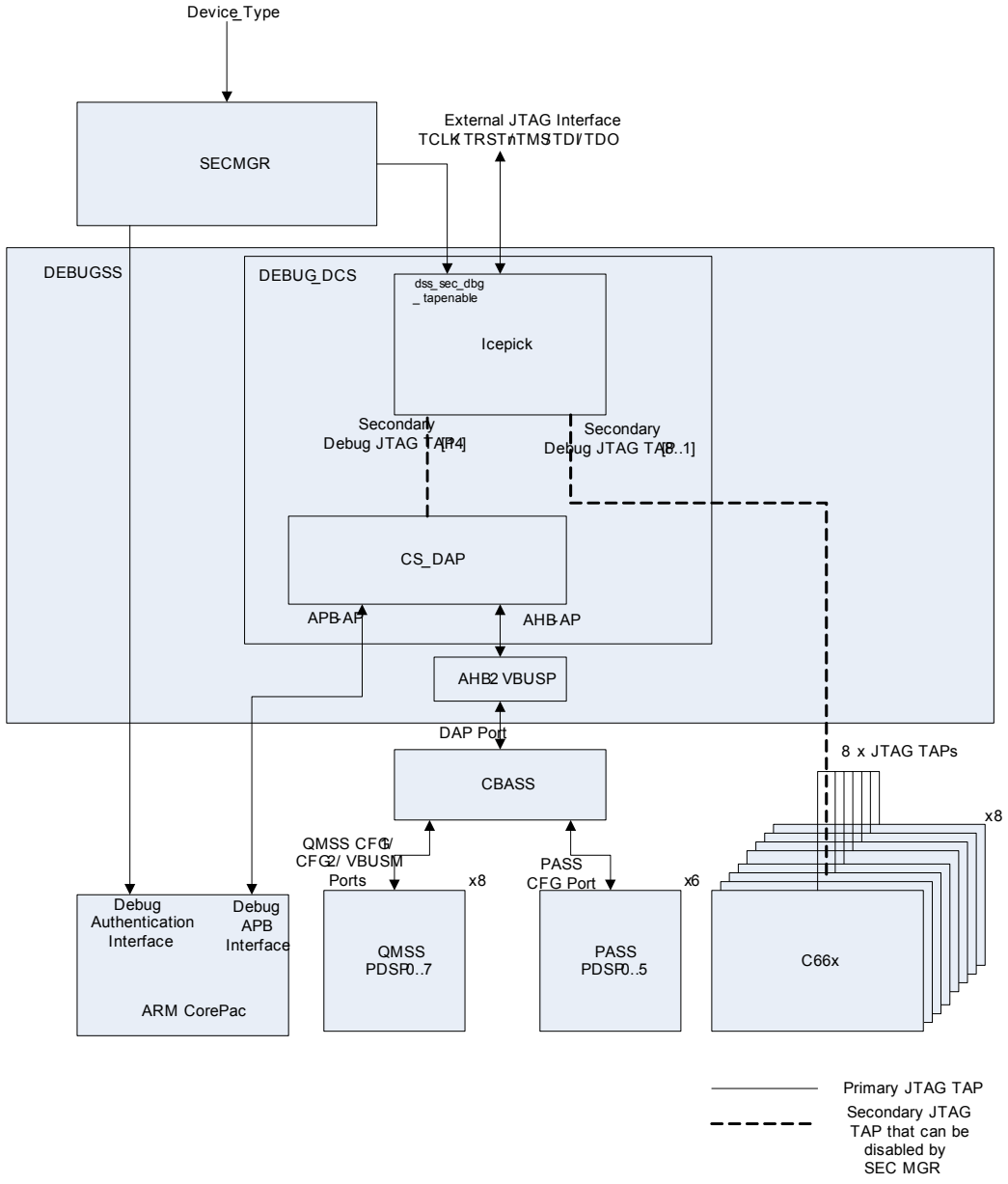
The Debug Subsystem is responsible for the capturing and exporting of trace data for DSP Core Trace, ARM Core Trace and System Trace. It is responsible for providing the mechanism for using embedded trace with the TETB or pin trace to export data to an external receiver.

The Debug Subsystem contains two main components used for debug:

- Debug Visibility Subsystem (DVS) for system trace management
- Data Resource Manager (DRM) for exporting trace data through the EMU pins (Debug Port) to an external receiver

Figure 1-1 shows a high level overview of the Debug Subsystem.

Figure 1-1 Debug Subsystem Overview



Each CorePac contains a Data Trace Formatter (DTF) to convert 10-bit trace packets into a 32-bit wide format for storage. The DTF also selects between the TETB and external pins for routing trace data (The STM performs the same feature for system trace data). The user must also program the DRM to select which trace port (one of the DSP Core Trace ports or the STM port) to connect to the external EMU pins.

Data is exported from the STM to the ETB through the Advanced Trace Bus (ATB) interface and from the STM to the Debug Port through the Parallel Trace Interface (PTI). The application must make sure that these modules are set up correctly for proper functionality.

1.4.1 Debug Pin Manager

The application software must make sure to program the DRM to appropriately export trace data to an external trace receiver. The Debug Pin Manager (DPM) of the DRM specifies this configuration by matching each of the debug pins to a specific function such as DSP Core Trace or triggering. The following table lists the available configurations each debug pin is allowed for each type of trace:

The device supports 34 emulation pins— EMU[33:0], which includes 19 dedicated EMU pins and 15 pins multiplexed with GPIO. These pins are shared by A15/DSP/STM trace, cross triggering and debug bootmodes.

The 34 pin dedicated emulation interface is defined in the following table.

Note—Note that If EMU[1:0] signals are shared for cross-triggering purposes in the board level, they SHOULD NOT be used for trace purposes.

Table 1-2 Emulation Interface with Different Debug Port Configurations (Part 1 of 2)

EMU Pins	Cross Triggering	ARM Trace		DSP Trace		STM	Debug Boot Mode
EMU33		TRCDT**a**[29]	TRCDT**b**[31]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU32		TRCDT**a**[28]	TRCDT**b**[30]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU31		TRCDT**a**[27]	TRCDT**b**[29]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU30		TRCDT**a**[26]	TRCDT**b**[28]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU29		TRCDT**a**[25]	TRCDT**b**[27]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU28		TRCDT**a**[24]	TRCDT**b**[26]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU27		TRCDT**a**[23]	TRCDT**b**[25]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU26		TRCDT**a**[22]	TRCDT**b**[24]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU25		TRCDT**a**[21]	TRCDT**b**[23]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU24		TRCDT**a**[20]	TRCDT**b**[22]			TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU23		TRCDT**a**[19]	TRCDT**b**[21]	TRCDT**a**[19]		TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU22		TRCDT**a**[18]	TRCDT**b**[20]	TRCDT**a**[18]		TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	

Table 1-2 Emulation Interface with Different Debug Port Configurations (Part 2 of 2)

EMU Pins	Cross Triggering	ARM Trace		DSP Trace		STM	Debug Boot Mode
EMU21		TRCDT**a**[17]	TRCDT**b**[19]	TRCDT**a**[17]	TRCDT**b**[19]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU20		TRCDT**a**[16]	TRCDT**b**[18]	TRCDT**a**[16]	TRCDT**b**[18]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU19		TRCDT**a**[15]	TRCDT**b**[17]	TRCDT**a**[15]	TRCDT**b**[17]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU18		TRCDT**a**[14]	TRCDT**b**[16]	TRCDT**a**[14]	TRCDT**b**[16]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU17		TRCDT**a**[13]	TRCDT**b**[15]	TRCDT**a**[13]	TRCDT**b**[15]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU16		TRCDT**a**[12]	TRCDT**b**[14]	TRCDT**a**[12]	TRCDT**b**[14]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU15		TRCDT**a**[11]	TRCDT**b**[13]	TRCDT**a**[11]	TRCDT**b**[13]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU14		TRCDT**a**[10]	TRCDT**b**[12]	TRCDT**a**[10]	TRCDT**b**[12]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU13		TRCDT**a**[9]	TRCDT**b**[11]	TRCDT**a**[9]	TRCDT**b**[11]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU12		TRCDT**a**[8]	TRCDT**b**[10]	TRCDT**a**[8]	TRCDT**b**[10]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU11		TRCDT**a**[7]	TRCDT**b**[9]	TRCDT**a**[7]	TRCDT**b**[9]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU10		TRCDT**a**[6]	TRCDT**b**[8]	TRCDT**a**[6]	TRCDT**b**[8]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU9		TRCDT**a**[5]	TRCDT**b**[7]	TRCDT**a**[5]	TRCDT**b**[7]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU8		TRCDT**a**[4]	TRCDT**b**[6]	TRCDT**a**[4]	TRCDT**b**[6]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU7		TRCDT**a**[3]	TRCDT**b**[5]	TRCDT**a**[3]	TRCDT**b**[5]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU6		TRCDT**a**[2]	TRCDT**b**[4]	TRCDT**a**[2]	TRCDT**b**[4]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU5		TRCDT**a**[1]	TRCDT**b**[3]	TRCDT**a**[1]	TRCDT**b**[3]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU4		TRCDT**a**[0]	TRCDT**b**[2]	TRCDT**a**[0]	TRCDT**b**[2]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU3		TRCCTRL	TRCCTRL	TRCCLKB	TRCCLKB	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU2		TRCCLK	TRCCLK	TRCCLKA	TRCCLKA	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	
EMU1	Trigger1		TRCDT**b**[1]		TRCDT**b**[1]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	dbgbootmode[1]
EMU0	Trigger0		TRCDT**b**[0]		TRCDT**b**[0]	TRCDT3, or TRCDT2, or TRCDT1, or TRCDT0, or TRCCLK or Tri-state	dbgbootmode[0]
End of Table 1-2							

The user can program the DPM to route a specific debug function to each debug port pin. STM data and clock can be configured on any of the debug port pins.

The user must make sure to claim ownership of the DPM and enable it before proceeding with any configuration of the module. An example of how to configure the DPM is included in the programming section of this user's guide.

Each debug port pin contains an associated DPM DSP Decode Register. An 8-bit programmable DPM control field for this register is associated with each debug port pin as shown below in Table 1-3.

Table 1-3 Debug Pin Configurations

DPM Control [x]	Debug Support	Debug Source
0x00	Tri-State	
0x01	Trigger	
0x02	System Trace	Trace Data[A]
0x03		Trace Data[B]
0x04		Trace Data[C]
0x05		Trace Data[D]
0x06		Trace Clock
0x07	Reserved	Reserved
0x08	Reserved	Reserved
0x09	Reserved	Reserved
0x0A	Reserved	Reserved
0x0B	DSP Core Trace	Trace Data[A]
0x0C		Trace Data[B]
0x0D		Trace Clock
0x0E	Reserved	Reserved
0x0F	Reserved	Reserved
End of Table 1-3		

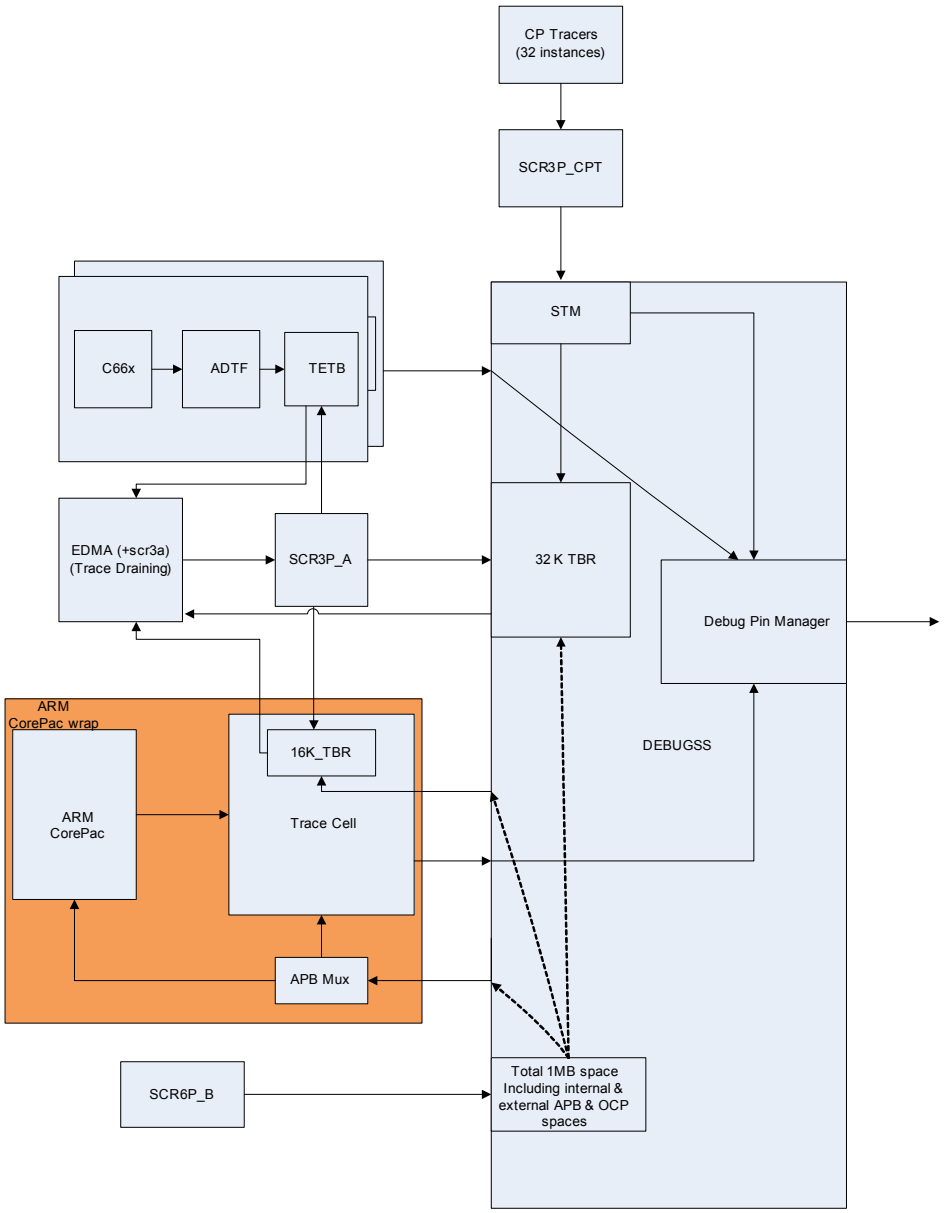
1.5 Trace Components

The below figure shows the overall trace architecture of the SoC with trace generation of ARM cores, DSP cores, DEBUGSS supported hardware (CPTracer) and software instrumentation. The figure also shows the trace draining using EMUx pins in addition to on-chip buffers for trace storage (TETBs for DSPs and CT-TBR for ARM and DEBUGSS STM)

In addition to the debug support provided on-chip, TI provides a set of APIs collectively known as CToolsLib that can fully utilize the debug features and assist with functions such as starting, stopping, and configuring various types of trace. For more information, see Chapter 8 "Programming Guidelines" on page 8-1.

Figure 1-2 shows the various trace components on the KeyStone II devices.

Figure 1-2 Trace Components



1.6 System Trace Module

The System Trace Module (STM) is part of the Debug Subsystem and implements the MIPI System Trace Protocol (STP). Its purpose is to collect system debug and performance data from the SoC and encapsulate it into a standard format. The STM supports both software messages and hardware messages. Hardware messages are those initiated from the individual Tracer modules as they monitor system activity. Software messages are those initiated by the application. Data from STM can also be routed to an external trace receiver through the Parallel Trace Interface (PTI) or stored on-chip in the shared trace buffer (TETB). For more information on hardware and software messages refer to the system trace section.

1.7 TI Embedded Trace Buffer (TETB)

The TETB module is an external trace buffer that can store data originally generated by Tracer modules or the DSP Core Trace modules. 32-bit wide data is captured from the DTF (Data Trace Formatter) and stored in TETB Trace RAM. There exists one TETB for each Corepac and one for the STM.

The TETB reads data from the RAM Read Data Register (RDR). This register holds the value of data read from the TETB Trace RAM that is sent over the VBUSP interface. The RAM Read Pointer Register (RRP) keeps track of the location in RAM currently being read.

1.7.1 Supported Modes

The TETB can be used in two modes: TI_MODE and non TI_MODE. The TI Specific Control Register (TCTL) specifies which mode to use. When TI_MODE is enabled, the contents of the trace RAM can be read from the RDR, regardless of whether trace capture is enabled or disabled. Reading from this register increments the RRP and triggers a RAM-access cycle to send data on the VBUSP interface. When TI_MODE is disabled, the only way to read the trace RAM is to disable trace capture. When trace capture is enabled and TI_MODE is disabled, a read from RDR returns 0xFFFFFFFF and the RRP Register does not increment.

The TETB can also capture trace data in Circular Mode and Stop Buffer Mode. Circular Mode continuously records trace data. When the TETB is full, the data is overwritten from the start location. Stop Buffer Mode sets the TETB to capture data until the buffer is full. In this mode, no additional data is be captured after the buffer is full.

1.7.2 Applications

The TETB with an on-chip trace buffer supports trace during run-time with no intrusion to the application. There are two primary application models that are facilitated by the TETBs:

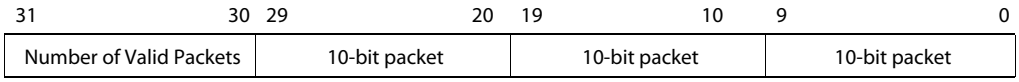
- Crash analysis: Here, the TETB contents are not accessed during run-time, but rather only at defined points. Typically this would be when an external host detects a system failure. In this case, the TETB can be set to trace continuously, and when a system fault occurs the host reads the TETB contents for offline analysis of the pre-fault execution.

- Continuous trace: The TETBs can be drained during run-time through the use of EDMA channels. This adds some system loading, as EDMA resources and memory bandwidth must be consumed for this purpose. However, continuous trace allows for trace data to be:
  - Stored in DDR to extend the size of the embedded trace buffer, which provides a larger history to the host.
  - Streamed out through a standard peripheral interface such as Ethernet, RapidIO, or PCIe, to be captured by an external host in the actual embedded system for real-time analysis.

1.7.3 Message Format

Figure 1-3 shows a format for the 32-bit data words sent to TETB:

Figure 1-3 TETB Message Format



Trace output from the DSP system is sent out in 10-bit packets to the TETB. As all of the packets may not contain valid data, bits [31-30] are used to keep track of the valid packets within the word.

Packets generated from the STM and sent to the TETB follow a different packet format, as STM packets can vary in size depending on data size and optional timestamp.

1.8 SoC Cross-Triggering

The device supports a cross-triggering feature, which provides a mean to propagate debug (trigger) events from one processor subsystem / module to another. For example, a given subsystem A can be programmed to generate a debug event, which can then be exported as global trigger across the device. Another subsystem B can be programmed to be sensitive to the trigger line input and to generate an action upon trigger detection.

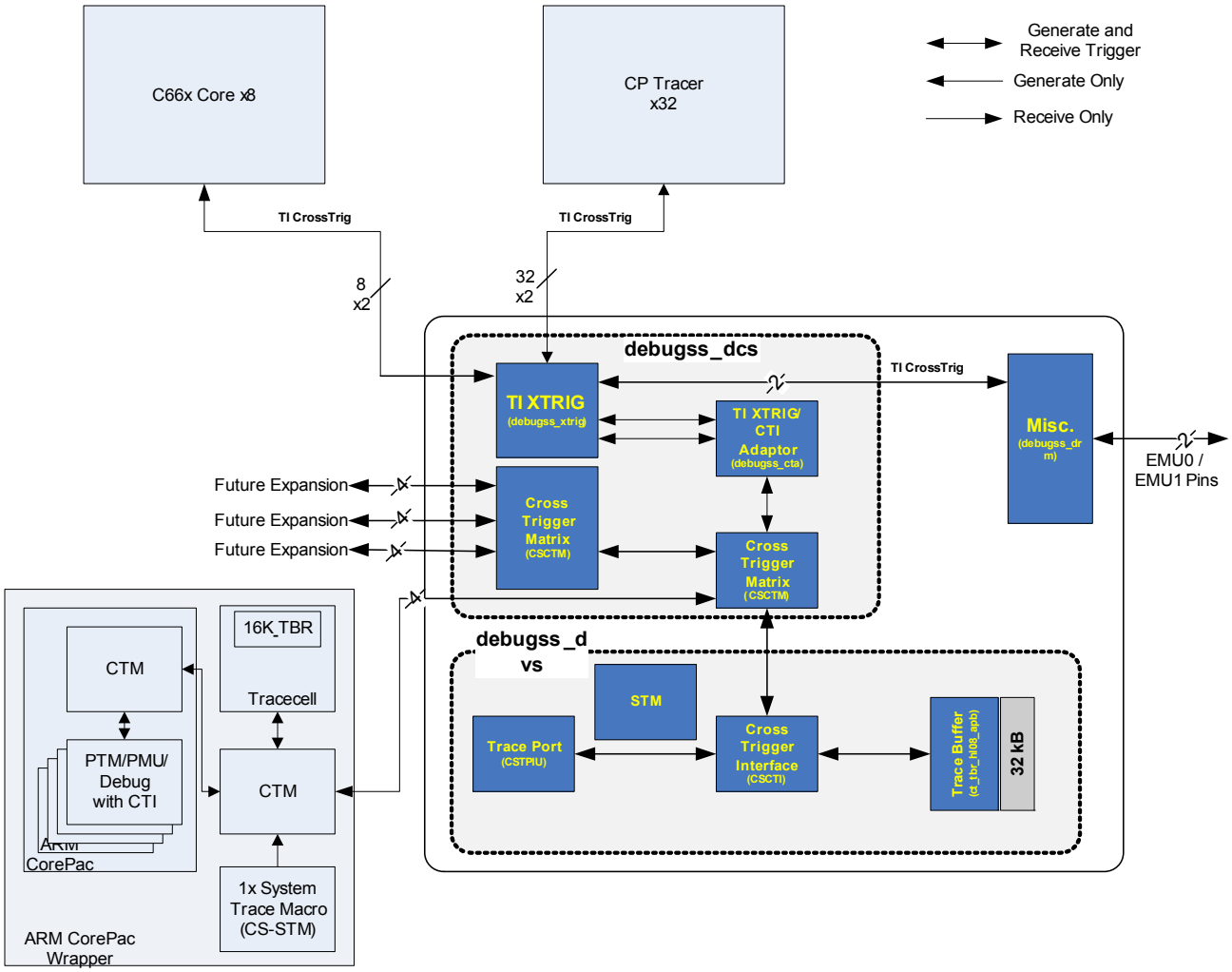
Examples of debug events are: processor entering debug state, watch point match, CS_PTM trigger, CS_ETB full and acquisition complete, etc.

Examples of debug actions are: debug request generation, restart (Cortex-A15 synchronous run), interrupt request generation, start/stop trace, etc.

Subsystems cross-triggering is consolidated at the device level by the XTRIGGER module, which is embedded in the debug subsystem. XTRIGGER also supports the off-chip trigger channels. Figure 1-4 shows the system level cross triggering connection

Note—XTRIGGER is not programmatically visible from the JTAG interface or any device processor. Thus, cross-triggering is programmed at the subsystem level.

Figure 1-4 SoC Cross Trigger Architecture



The debug subsystem (DEBUGSS) implements two cross-triggering schemes:

- TI cross-triggering scheme
- CoreSight cross-triggering scheme

The DEBUGSS has multiple modules to manage triggering across these cross trigger schemes.

- XTRIGGER module (TI cross trigger module) to support merge and distribution of TI asynchronous triggers (2 channels per interface). This is connected to all C66x, CPTracers and within DEBUGSS with STM/DRM
- CS_CTM (CoreSight Cross Trigger Matrix) to support merge and distribution of CoreSight triggers to CoreSight components inside and outside of the debug subsystem
- CT_CTA module (CTools Cross Trigger Adapter) to support conversion between TI asynchronous trigger format (2 channels) and CoreSight handshake based asynchronous trigger format

This cross-triggering architecture of the debug subsystem allows it to provide the following features:

- Migration of a trigger originating from a CoreSight trigger source and ending at a TI trigger sink
- Migration of a trigger originating from a TI trigger source and ending at a CoreSight trigger sink
- Migration of a trigger originating from a CoreSight trigger source and ending at a CoreSight trigger sink
- Migration of a trigger originating from a TI trigger source in to a TI trigger sink
- Examples of debug events are:
- Processor entering debug state
- Watch-Point match
- PTM trigger
- ETB full
- ETB acquisition complete

Examples of debug actions are:

- Debug request generation
- Restart
- Interrupt request generation
- Start Trace
- Stop Trace

The device implements two global cross-triggering lines: Trigger0 and Trigger1, also referenced as EMU0 and EMU1, respectively. Subsystems cross-triggering is consolidated at device level by the XTRIGGER module, which is located in the Debug Subsystem. The trigger lines are shared by all the device subsystems implementing cross-triggering and are used to facilitate co-emulation. A trigger event can therefore be propagated to any subsystem.

Based on DRM settings, the Trigger0 and Trigger1 lines may also be used as external triggers for cross-triggering. The user must make sure to program the DRM module in accordance with Table 1-2.

1.8.1 Cross-Triggering with DSP Core Trace

Cross-Triggering with DSP Core Trace is possible using EMU0 and EMU1 signals from the Debug Port. We can use these signals to trigger actions such as halt the target as a hardware breakpoint, or have the AET module on one DSP core trigger an action on another DSP core, the STM, or even on another device.

1.8.2 Cross-Triggering with System Trace

The MIPI-STM module interfaces with the SoC cross-triggering logic and implements two trigger input lines. These triggers can then be programmed by the user to start and stop collecting software instrumentation data:

- The EMU0 trigger line is coupled to SW Masters trace start
- The EMU1 trigger line is coupled to SW Masters trace stop

1.8.2.1 Start Collecting Software Instrumentation Data Upon External Trigger

When the user selects this tracing mode:

- All the MIPI-STM software instrumentation masters filtering parameters will be active
- In absence of trigger detection:
  - The software message matches will be acknowledged
  - The software message matches will not be captured into the snapshot buffer
- Upon detection of an external trigger routed to the EMU0 input:
  - The software message matches will be acknowledged
  - The software message matches will be captured into the snapshot buffer

Note—Hardware instrumentation masters are not sensitive to the MIPI-STM input triggers. The two input triggers are shared across all the software instrumentation masters.

For a use case where the ATB interface is disabled and re-enabled at later stage, the STM shall wait for a new trigger start detection to resume the trace capture.

For a use case where the PTI interface is disabled and re-enabled at later stage, the STM shall wait for a new trigger start detection to resume the trace capture.

1.8.2.2 Stop Collecting Software Instrumentation Data Upon External Trigger

When the user selects this tracing mode:

- All the MIPI-STM software instrumentation masters filtering parameters will be active
- In absence of trigger detection:
  - The software message matches will be acknowledged
  - The software message matches will be captured into the snapshot buffer
- Upon detection of an external trigger routed to the EMU1 input:
  - The software message matches will be acknowledged
  - The software message matches will not be captured into the snapshot buffer

Note—Hardware instrumentation masters are not sensitive to the MIPI-STM input triggers. The two input triggers are shared across all the software instrumentation masters.

1.8.2.3 Start Event Monitoring Upon External Trigger

When the user selects this tracing mode:

- All the system event filtering parameters will be active
- In absence of trigger detection:
  - The system events will be acknowledged
  - The system events will not be captured into the snapshot buffer
- Upon detection of an external trigger routed to the EMU0 input:
  - The system events will be acknowledged
  - The system events will be captured into the snapshot buffer

1.8.2.4 Stop Event Monitoring Upon External Trigger

When the user selects this tracing mode:

- All the system event filtering parameters will be active
- In absence of trigger detection:
  - The system events will be acknowledged
  - The system events will be captured into the snapshot buffer
- Upon detection of an external trigger routed to the EMU1 input:
  - The system events will be acknowledged
  - The system events will not be captured into the snapshot buffer

1.9 The Suspend Mechanism

The device supports a suspend feature, which provides a means to stop a closely-coupled hardware process running on a Peripheral-IP when the host processor enters debug state. The suspend mechanism is important for debug to ensure that Peripheral- IPs operate in a lock-step manner with a host controller processor. An entry is provided for each Peripheral-IP that needs to take into account the suspend signals from a number of processors. For each Peripheral-IP, the sensitivity to the suspend signals is defined within two possibilities (and so coded using one bit):

- Peripheral-IP is sensitive to the suspend line request
- Peripheral-IP ignores the suspend line request

Refer to each Peripheral-IP TRM chapter for more information about how to program the sensitivity.

A shared Peripheral-IP can be sensitive to several host processors in terms of suspend. For each Peripheral-IP, the Debug Subsystem DRM module implements a Suspend Control Register to select which processor shall freeze the Peripheral-IP activity.

Each Suspend Control Register has the following bit fields:

- SUSPEND_SEL - suspend source selection
- SENS_CTRL - sensitivity control
- SUSPEND_DEFAULT - a status bit (Read-Only) that defines whether the Peripheral-IP shall use its default suspend settings, or shall let the user select the host processor
- The Suspend_Default value is hardwired during SoC integration and determines the functionality of the other two fields:
  - SUSPEND_DEFAULT = 0. The programmer can select one of the multiplexed suspend signals by writing to the 4-bit SUSPEND_SEL field, and can select whether this suspend signal shall reach the Peripheral-IP (SENS_CTRL = 1), or will be blocked (SENS_CTRL = 0).
  - SUSPEND_DEFAULT = 1. The programmer cannot control the suspend functionality. The peripheral is sensitive to a default suspend signal that cannot be blocked. The SUSPEND_SEL field is read-only and shows the number of the default suspend signal. The SENS_CTRL field is also read-only and shows 1.

1.10 Application Integration

For software support of the debug tools mentioned, see Chapter 8 "Programming Guidelines" on page 8-1.

1.11 Board Design Integration

In order to facilitate correct debug functionality between the device's emulation signals and a particular board JTAG interface, certain procedures must be followed. For information on board routing guidelines for JTAG and emulation signals, refer to the Emulation and Trace Headers TRM, SPRU655g.

Chapter 2

DSP Debug

2.1 "Overview" on page 2-2
2.2 "Halt Mode Debug" on page 2-2
2.3 "Real-Time Debug" on page 2-3
2.4 "Monitor Mode (Run Mode) Debug" on page 2-3
2.5 "Trace" on page 2-4
2.6 "Trace Port Width Configuration" on page 2-4
2.7 "Tuning Export Clock Frequency" on page 2-5
2.8 "Trace Pin Configuration" on page 2-5
2.9 "Advanced Event Triggering" on page 2-5

2.1 Overview

The following sections define the types of emulation actions possible for C66x. C66x CorePac supports invasive debug techniques like Halt mode, Monitor mode and Real time debug. C66x also supports advanced non-invasive debug mechanisms like Trace. The final section talks about the triggering options possible with AET (Advanced Event Triggering) supported by C66x.

2.2 Halt Mode Debug

Keystone II supports traditional stop-mode emulation, whereby host tooling can halt the core, as well as any of the peripherals. The core has a direct connection to the JTAG interface. Emulation has full visibility of the DSP memory map, and can access the chip memory through the core.

Each core has visibility requirements critical for successful software development and debug in the field. Functionality has been present in previous devices, and continues to be implemented here.

The following are requirements in a debugger environment (host control through JTAG)

- Run-time execution control (e.g. halt, step, run)
- Accurate reflection of memory and register contents
- Visibility into system stalls
  - Bank conflict
  - Cache miss/coherency overhead
  - Memory latency
  - DMA conflict
  - Cache state (coherency with external memory)
- Shared memory view:
  - Cache state (coherency between cores)
  - Conflict between cores (performance degradation)
- Shared atomic resources (memory structure, IP, etc.)
  - Ownership through Hardware Semaphore– view/force
  - IPC flags – view/force
  - Volatile shared structs – view/update
- Shared multichannel resources (e.g. DMA channels)
  - View/change context without impacting other cores
- Cross-triggering
  - Local halt, step, run
  - Global halt, step, run (synchronized)
  - Local hardware break point
  - Global software break point (assert and receive)
  - Global hardware break point (assert and receive)

2.3 Real-Time Debug

Real-Time Debug, a DSP only feature, provides a base set of capabilities for real-time execution control (run, step, halt, etc.) and register/memory visibility. This infrastructure component allows the user to debug application code while interrupts designated as real-time continue to be serviced. Registers and memory may be accessed while code is running or stopped. Users define the interrupts that are to be treated as real-time through a special debug interrupt enable register (DIER). The user can also mark code that must not be disturbed by real-time debug memory accesses within the application. These facilities provide the basic execution control and memory/register accessibility needed by the Code Composer Debugger.

Real-time debug encompasses three code execution states. These states generate bus activity that is presented to triggering logic. An applications developer may want to constrain triggering behavior to a limited set of these states. Before triggering behavior can be discussed, it is important to understand the three CPU code execution states related to debug:

- Normal code execution—running code prior to a debug event (normal code and interrupt service routines). Normal code execution is the behavior of the system as if the emulator is not connected to the chip where the peripheral resides. In other words, the CPU is running code without a debug event halting execution with the peripheral operating continuously.
- Secondary code execution—running code related to the service of a real-time interrupt after a debug event has halted code execution. The CPU execution is designated by the applications developer as real-time, allowing the service of interrupts designated as real-time (via the debug interrupt enable register) after code execution is halted.
- No code execution—not running code. This occurs when the emulation functions are enabled, a debug event halts code execution, and no real-time interrupt is being serviced after code execution is halted. Generally, break detection is always enabled for normal code execution. The applications developer may desire to either include or exclude secondary code execution. A programmable option is provided to either include or exclude secondary code execution in trigger generation.

2.4 Monitor Mode (Run Mode) Debug

It is possible to have a real-time debug thread of execution that provides application-level debugging features at run-time that complement the Stop-Mode/Real-Time Emulation features noted above. The debug thread would respond to commands sent by a host “debugger” and configure the AET logic to raise an AINT interrupt whenever a user-specified trigger condition occurred. The ISR for the AINT interrupt would be handled by the real-time debugger, which would implement whatever functionality was required to perform the required debug actions requested by the host debugger.

The debug thread software would be implemented as a library that is linked in by the customer application on the device, and would give the ability to:

- Suspend a specified task (on interruptible region)
- Resume execution of a the task
- “Single step” the task (i.e. set a pending interrupt flag, resume execution of the task and suspend execution of the task when the pending interrupt is serviced)
- View/modify memory, CPU registers

2.5 Trace

C66x trace provides for the recording of program flow, memory references, and application-specific data with a timestamp. Trace targets the debug of unstable code, performance analysis, and quality assurance. The use of dedicated hardware to both collect, buffer, transfer trace data to the host makes trace non-intrusive. Data is exported via trace packets (1 to 20-bits). These packets are converted to trace transmission packets via the trace export block. Transmission packets can be from 1 to 20 bits wide. This allows all or a percentage of the debug port pins to be allocated to trace. This allows the sharing of debug port pins between trace and other functions, with the number of pins allocated to trace and the corresponding trace port width varied at runtime.

The trace function can collect and export a record of the program flow and timing at the same rate generated by the CPU. Tracing data references must be restricted however as the export mechanism is generally limited to a number of pins insufficient to sustain tracing of all memory references. In the case of data trace, the Advanced Event Triggering facilities provide a means to restrict the trace data exported to data of interest to maintain the non-intrusive aspect of trace. This reduces the export bandwidth needs and facilitates the successful collection of the data of interest. Error indications are embedded in the debug stream in the event the export logic is unable to keep up with the data rate generated by the collection logic. This notification allows the user to scale back the amount of requested data collection.

The user can optionally select the export of all specified trace data. In this case the CPU is stalled to avoid the loss of trace data, with trace becoming intrusive to the application if trace related stalls are generated. The user is notified that trace stalls have occurred although the number of stalls and their location is not recorded.

The trace technology is modular with different collection elements for different types of data (PC, Timing, Memory References). This allows the customization of the trace function if the trace function is not encapsulated in the CPU mega-module. Collection elements for different types of data are modular allowing the customer to trade off functionality verses gates and pins required to meet the bandwidth requirements. Trace export shares debug pins with other functions.

Another option to export the trace data is via TETB buffers. Each C66x supports one dedicated TETB with 4KBytes memory. The trace data in TETB can be drained to some other memory (L2, shared, or external) while continuing to trace. And the trace data can be further exported by using an application interface such as Ethernet, SRIO, PCIe and etc. However, these measures introduce some intrusion to the normal operation of the system.

2.6 Trace Port Width Configuration

C66xCore Trace formats data into 1 to 20-bit packets. For external capture through the Debug Port, the trace port for each core has a width of 20 bits. This port mapping is software configurable to 16 or 18 data pins (TRACEDATA) plus two data pins dedicated to the export clock (TRACECLK). More information can be found in "Debug Pin Manager" on page 1-6.

2.7 Tuning Export Clock Frequency

The device implements two C66xCore Trace export clock (TRACECLK) generation schemes:

- Programmable PLL reference clock shared with application with programmable post-dividers

2.7.1 Tuning Export Clock Frequency Through PLL

The C66xCore Trace export clock can be derived from one of the following PLLs:

- MAIN PLL
- Peripheral PLL

Selection of PLL reference clock is done through the PLL Controller module. The PLL Controller module also provides a means to tune the DSP Core Trace export clock frequency by dividing it to 3 (default for SYSCLK2, used by C66x CorePacs Trace) or 5 (default for SYSCLK3, used by the System Trace Module) of PLL Controller level. These SYSCLK are configurable hence it is then possible to slow down the interface clock at Debug Subsystem level.

For more information about the Main PLL, see the *Phase Locked Loop (PLL) Controller for KeyStone Devices User Guide*.

WARNING—When using TI's provided tools (CCS with an XDDS560T or XDS560v2 Trace Pro) do not modify the PLL used for DSP Core Trace. This will cause conflicts with TI's software that uses a calibration scheme to set the trace data rate at the fastest rate possible the receiver can collect data at.

2.8 Trace Pin Configuration

The user can also configure the Debug Pin Manager to change the allocation of the EMU pins between ones allocated for data, trace clock 0, and trace clock 1.

2.9 Advanced Event Triggering

Triggers manage breakpoints, trace acquisition, data collection via an interrupt, timing measurement, and generate external triggers. They also control a state machine and counters used to create the intermediate events (loop counts and state machines). Advanced event triggering uses instruction and data bus comparators, auxiliary event detection, sequencers/state machines, and event counters to identify events of interest. These events can be combined to create simple or complex triggers using modules call trigger builders. Advanced Event Triggering logic is provided for monitoring program, memory bus, auxiliary input activity, remembering event sequences, counting event occurrences, or measuring the interval between events. Bus comparators can perform range and identity comparisons, detect exact transactions or the mere touching of a byte or range of bytes by memory references. External event detectors provide a means to monitor external triggers or internal states of interest (i.e. cache miss). A state machine with four states that allows transitions from any state to any other state provides for the identification of a sequence of triggers, with counters/timers (16-bit and/or 32-bit) providing for the implementation of loop counts or a predetermined number of events. The state machine and counter components are controlled by

triggers and generate events (state number and count equals zero). The state machine and counter events are part of the event pool used to create triggers. Trigger builder components are used to create triggers from the events created by all event generation sources.

Triggering facilities are used to identify specific system activity to generate breakpoints, an interrupt used for the collection of system data, or the identification of program activity that is observed through Trace.

Any system event routed to a C66x core can be routed (through software selection) to the AET. This is controlled through software by configuring the core's interrupt controller as well as the chip-level CP_INTCs. The C66x core's interrupt controller supports routing up to eight events to the AET logic for tracking by the emulation hardware. Events that can be routed include the system events, CPU interrupts and exception inputs, and interrupt and exception acknowledges. Routing the same event to AET on all C66x cores while configuring the same trigger action to be taken on all cores allows for 'global' events and actions. Likewise, cross-triggering is possible by having an AET trigger on one DSP core signal (through EMU0/1) an action on another core (or STM) within the chip or even on another device (assuming the device pins are connected properly).

For more information on how to program the AET for various application use, see.
Chapter 8 "Programming Guidelines" on page 8-1.

Chapter 3

ARM Debug

3.1 "Overview" on page 3-2
3.2 "Debug Modes" on page 3-2
3.3 "Processor Trace Macro (PTM)" on page 3-2
3.4 "CoreSight System Trace Macro (STM)" on page 3-3
3.5 "Performance Monitor Unit (PMU)" on page 3-3
3.6 "Cross Trigger Matrix (CTM)" on page 3-4

3.1 Overview

The Cortex-A15 processor supports the following native features:

- Halt mode and monitor mode debug
- Six hardware breakpoints and four watch points
- Asynchronous aborts
- Processor Trace and System Trace
- Performance monitoring
- Cross-triggering: allows stopping one CPU upon debug event (for example, breakpoint) detection in the other CPU

For more information about Cortex-A15 native debug support features, see the Cortex-A15 Technical Reference Manual.

3.2 Debug Modes

Cortex A15 support Invasive and Non-invasive debug methods.

The invasive debug component of the Debug architecture is intended primarily for run-control debugging. Invasive debug mode has support for

- Monitor debug-mode: In Monitor debug-mode, a debug event causes a debug exception to occur. A debug exception that relates to instruction execution generates a Prefetch Abort exception. A debug exception that relates to a data access generates a Data Abort exception. A software handler can then take control to examine or alter the processor state. Monitor debug mode is essential in real-time systems where the processor cannot be halted to collect debug information.
- Halting debug-mode: In Halting debug-mode, a debug event causes the processor to enter Debug state. In Debug state, the processor stops executing instructions from the location indicated by the program counter, but is instead controlled through the external debug interface, in particular using the Instruction Transfer Register, DBGITR

Non-invasive debug includes all debug features that permit data and program flow to be observed, but that do not permit modification of the main processor state. Non invasive debug mode support includes Processor Instruction Trace, Sample based profiling and Performance monitoring.

3.3 Processor Trace Macro (PTM)

Cortex A15 integrates a per core Processor Trace Macro (PTM or CS PTM). The PTM module performs real-time instruction flow tracing. It generates information that can be used by trace receiver to reconstruct the execution of all or part of a program.

The PTM identifies certain instructions in the program, and certain events, as waypoints. A waypoint is a point where instruction execution by the processor may involve a change in the program flow.

The PTM traces only the following waypoints:

- All indirect branches
- All conditional and unconditional direct branches
- All exceptions
- Any instruction that changes the instruction set state of the processor

- When halting debug mode is enabled, entering or leaving debug state
- Synchronization primitives

The PTM provides the ability to reconstruct the complete instruction flow.

For more information about CS_PTM, see ARM CoreSight ™ Program Flow Trace ™ Architecture Specification, v1.0 [4]

The Cortex-A15 processor trace characteristics are:

- Program trace only (no data trace)
- Separate PTM for each MPU core
- Two exclusive trace sinks:
  - TPIU: Trace exported to an external trace receiver
  - CT_TBR: Trace exported to on-chip buffer for draining through high speed serial interfaces
- Trace can be optionally:
  - Cycle accurate useful for profiling sections of code
  - Locally time-stamped using a 64-bit free-running graycode counter exported to the PTM of each MPU core
  - Interleaved dynamically between the MPU cores using MIPI Trace Wrapping Protocol (TWP) specification (for more information on MIPI TWP see the *MIPI Alliance Web site*)

Note—Depending on use case, the device may not be able to interleave two cycle accurate traces without overflow (bandwidth restriction).

3.4 CoreSight System Trace Macro (STM)

The STM integrated in the ARM CorePac provides 'extended stimulus port' registers designated to be accessible by software with minimum instrumentation cycles overhead.

Each 'extended stimulus port' occupies 256 consecutive bytes in the memory map and is write-only. Up to 64K instrumentation channels are available at MPU level.

The STM supports 'guaranteed' or 'invariant timing' transactions:

- Guaranteed transactions are guaranteed to be traced. This might involve stalling the MPU core to ensure the transaction is accepted by the CS_STM
- Invariant timing transactions are not guaranteed to be traced. These transactions will take an invariant amount of time regardless of the state of the CS_STM

3.5 Performance Monitor Unit (PMU)

The Cortex-A15 processor includes a per-core performance monitoring unit (PMU) that enables events, such as cache misses and instructions executed, to be counted over a period of time. The PMU gathers statistics about the operation of the processor and memory system.

The Cortex-A15 PMU events are listed in the Cortex A15 TRM.

3.6 Cross Trigger Matrix (CTM)

The CortexA15 CorePac provides support for cross-triggering between CPU cores using CTI and an integrated CTM (Cross Trigger Matrix). The CTI is programmable through AXI bus accesses or by debugger from APB bus directly.

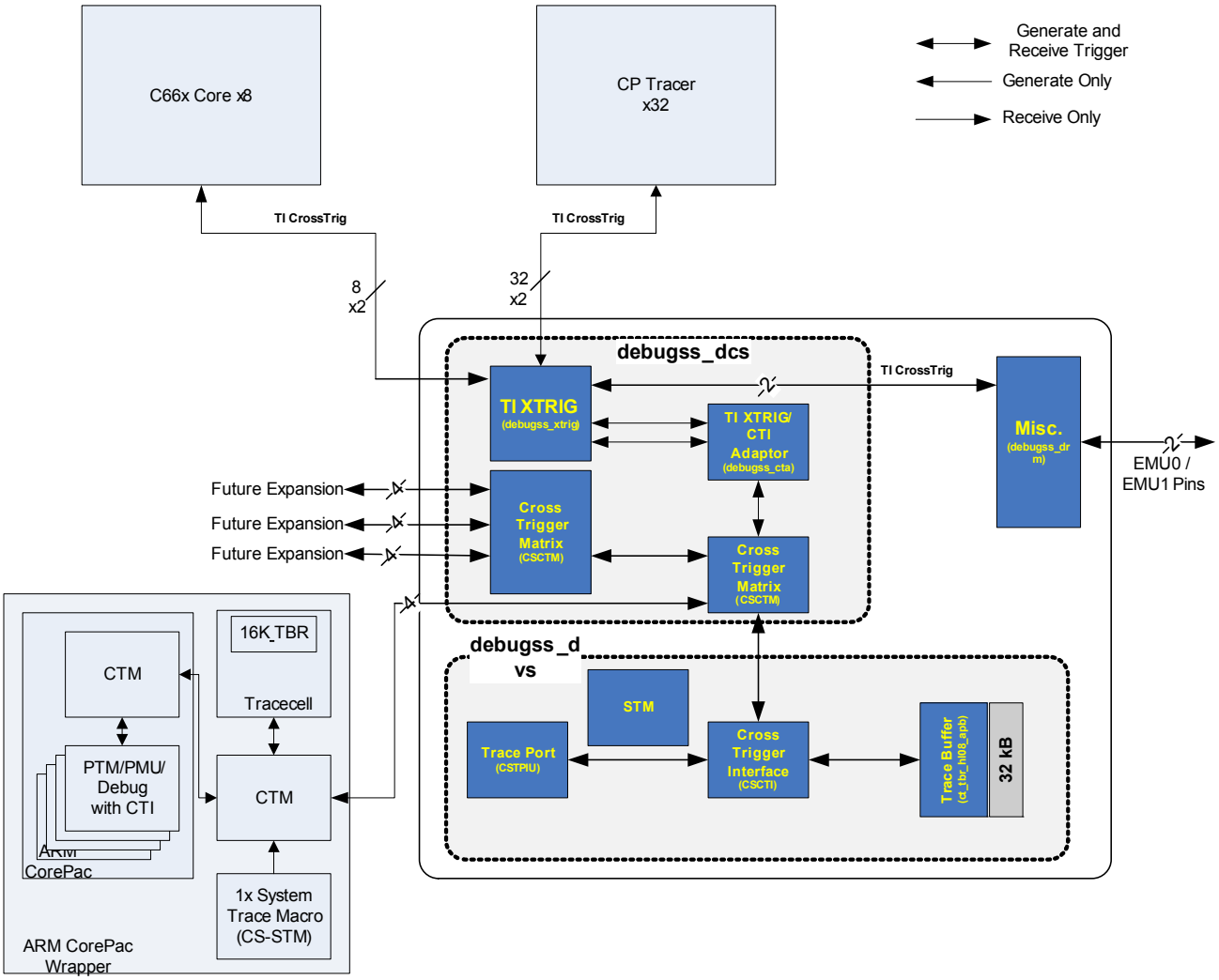
There is also another CTI integrated at wrapper level to collect trigger events from STM. ARM CorePac wrapper also integrates a CTM (cross trigger matrix) to carry cross-triggers between the processor cores, integrated STM, integrated CT-TBR and from DEBUGSS outside the ARM CorePac wrapper.

- Support for global or synchronous run when Cortex-A15 CPUs are halted
- Cross-triggering between PTMs and CT-TBR
  - PTM can generate a trigger that can be used to stop CT-TBR data capture
  - CT-TBR can send buffer-full or acquisition-complete events to the PTM

The ARM CorePac cross-triggering logic allows stopping one CPU core at debug event detection in the other MPU core. For example, a watch point detected by Core0 can break Core1 execution, and vice versa. This is also known as symmetric multiprocessor platform (SMP) debug and is supported through CTI programming.

Figure 3-1 highlights the ARM subsystem cross-triggering scheme along with the implementation at the SoC level. A debug event from any processor core can be programmed to have a specific debug action on the other core or drive a SOC trigger line and generate a debug action into another application subsystem or debug and trace component.

Figure 3-1 SoC Cross Trigger Architecture



Chapter 4

System Trace

4.1 "Overview" on page 4-2
4.2 "DebugSS Trace and Monitoring" on page 4-4
4.3 "Trace Port Width Configuration" on page 4-4
4.4 "Tuning Export Clock Frequency" on page 4-4
4.5 "Software Messages" on page 4-5
4.6 "Hardware Messages" on page 4-5
4.7 "Overflow" on page 4-5
4.8 "Reset" on page 4-5
4.9 "Message Interleaving" on page 4-6
4.10 "Master ID Encoding" on page 4-6
4.11 "Timestamping" on page 4-6
4.12 "Message Format" on page 4-7

4.1 Overview

As devices get more complicated and more IP is integrated into the device (SoC) there is a requirement to provide sufficient hooks in the silicon to provide enough visibility to facilitate debug in the system. Multiple cores (processing engines) executing concurrently and asynchronously stress the system memory in variable fashion during execution in response to system stimulus (events). Multiple 'DMA' transactions move a very large amount of data through the chip concurrently, converging at memory and accelerator endpoints. The data movement is a mixture of synchronous and asynchronous activity, both driving and driven by the program execution.

System trace can provide significant debug visibility into how multiple cores and masters are interacting and executing in an SoC. System trace provides visibility on bandwidth consumption by SoC masters as well as transaction trace information. Its purpose is to collect system debug and performance data from the SoC and encapsulate it into a standard format. These features are accomplished on KeyStone devices through both hardware and software messages.

The on-chip Tracer modules provide the functionality which generate hardware messages (as explained in the Tracer section of the user guide). All hardware messages originating from the various Tracer modules have the same Master ID (128) and hence are treated as a single hardware message source.

System trace also supports the capture of software messages as a minimally intrusive means to log information during run-time. This functionality is opposed to using printf() statements which are too intrusive and/or not available on many systems. To trigger SW messages, applications write data to be logged to a memory mapped location of the System Trace Module (STM). The STM will accordingly transport this data to the on-chip dedicated TETB or to an external receiver. The STM provides 256 dedicated channels for simultaneous logging of software messages.

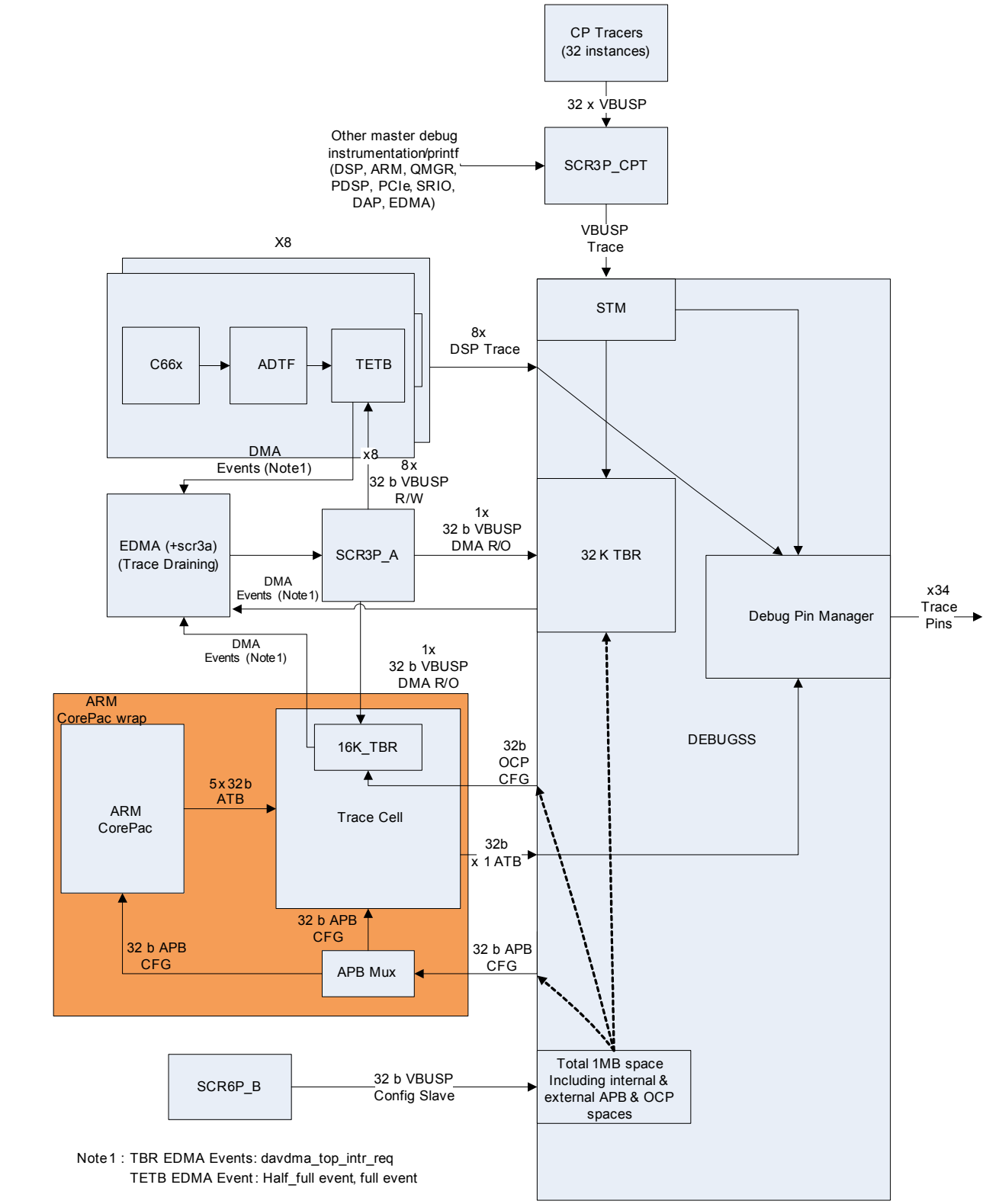
System trace data can be stored on-chip on the dedicated STM ETB or exported off-chip to an external trace receiver. When exporting data off-chip, the STM must interface with the DRM through the Parallel Trace Interface (PTI). The system trace port width and clock can both be configured through the PTI. More information on configuring the PTI can be found in Chapter 8 "Programming Guidelines" on page 8-1. Interfacing the STM with the dedicated ETB is performed through the ATB port.

A summary of the system trace features are as follows:

- Collects software-debug data from the Network on Chip (NoC)
- Collects software & hardware debug data from the SOC subsystems
- Collects NoC performance metrics
- Encapsulates instrumentation data
- Exports instrumentation data to an external trace receiver
  - Parallel Trace Interface [PTI]
- Stores instrumentation data into an on-chip shared trace buffer
  - Embedded Trace Buffer [ETB]

Figure 4-1 shows the overall trace architecture of the SoC with trace generation of ARM cores, DSP cores, DEBUGSS supported hardware (CPTracer) and software instrumentation. The figure also shows the trace draining using EMUx pins in addition to on-chip buffers for trace storage (TETBs for DSPs and CT-TBR for ARM and DEBUGSS STM).

Figure 4-1 SoC Trace Architecture



4.2 DebugSS Trace and Monitoring

The DEBUGSS integrates the DEBUGSS STM (also called MIPI STM or CT-STM) module to handle hardware instrumentation (CPTracer) messages and Software instrumentation (from various processing cores)

DEBUGSS System Trace Macro (STM)

DEBUGSS STM is a trace module that aids in software debugging. The main features of this module are:

- Implements MIPI STP protocol (rev 2.0) with the following characteristics:
  - Highly optimized for software-generated traces from ARM, DSP, QMSS PDSP, EDMA sources
  - Automatic time stamping of messages
  - Support for 8-, 16-, and 32-bit data types
- Collects the following information:
  - Software messages
  - Hardware instrumentation trace from CPTracer
- Exports trace data to:
  - Pin trace through the Parallel Trace Interface (EMUx pins) Or
  - Capture of trace into internal buffer through ATB interface to DEBUGSS CT_TBR
- Pin trace available in 1-, 2-, or 4-pin mode with single- or dual-edge clock, depending on the trace bandwidth requirements and characteristics of the trace receiver

4.3 Trace Port Width Configuration

For export to an external trace receiver, the STM has a configurable export width of 1, 2, or 4 data pins (STM_DATA) plus a dedicated export clock (STM_CLK). Data with and clock selection are configured through the PTI and are not to be changed on the fly.

4.4 Tuning Export Clock Frequency

The device implements two system trace export clock (STM_CLK) generation schemes:

- Programmable PLL reference clock shared with application with programmable post-dividers

Tuning Export Clock Frequency Through PLL

The MIPI-STM trace export clock (STM_CLK) can be derived from one of the following PLLs:

- MAIN PLL
- Peripheral PLL

Selection of PLL reference clock is done through the PLL Controller module. The PLL Controller module also provides a means to tune the STM export clock frequency by dividing it to 1, 2, or 4 of PLL Controller level. It is then possible to slow down the interface clock at Debug Subsystem level.

4.5 Software Messages

There are cases during development of a complex application, where the stop mode debug (breakpoint-based) doesn't work well because no task or processor can be halted without affecting the correctness of entire system, and certain run-time debug method has to be used. Typically, these are the printf() or the trace. However there are problems associated with this

- Printf() – too intrusive or won't be available for most embedded system.
- CPU trace – only limit to CPU, not all the processors in the system.

STM-based SW-message is a software instrumentation method, which is light-weight in term of intrusiveness, and is very easy to use. With it, an application directly writes data to be logged to some memory mapped location of STM, and STM will automatically capture and export the data to external receiver or to the on-chip CT-TBR buffer. To have this, all C66x/A15/QMGR PDSPs have a write access path to the memory space of STM, and all the processors should have a distinguished MstID[7:0]. In current STM implementation, MstID[7]=0 for SW message. STM supports 256 channels for each SW master. If up to 256 tasks (threads) running on same processor use different channels, they can log data to STM simultaneously without any mutex/semaphore protection.

STM provides multiple registers for configuring which software masters are enabled for generating trace messages. The cTools libraries then provide various APIs for generating the software messages.

4.6 Hardware Messages

Hardware messages are generated by the Tracer modules. For more information on hardware messages, refer to the Tracer section of the user guide.

4.7 Overflow

When there is overflow of input data to the STM from hardware messages, the STM will stall input from any source to avoid overflowing its buffers.

4.8 Reset

The STM will be reset with a hardware power on reset. All STM registers are asynchronously reset. The STM shall remain operational and with its setup preserved when a warm reset is triggered in order to export the trace history to the trace controller. This trace history may contain valuable information from a debug perspective allowing the user understanding the root cause of the warm reset (e.g. security violation). Making the STM non sensitive to warm reset will also allow tracing the transactions processed right after the reset without re-configuration of the System Trace.

The application or debugger software can reset the STM at any time by writing a 1 to the SoftReset bit of the STM System Configuration Register. This will put the STM in the same reset-state as a hardware reset.

4.9 Message Interleaving

The STM architecture allows interleaving of software and hardware master's messages. These flows are typically asynchronous and could result in concurrent or very close accesses arbitrated by the instrumentation NoC and signaled by MASTER message. Therefore the STM buffer shall allow absorbing write accesses peaks resulting from interleaving.

4.10 Master ID Encoding

The STM uses a Master ID field to differentiate between hardware and software messages. It uses the following field encodings:

- MReqMstID[7]: Differentiates software masters versus hardware masters (0 signifies a software master and 1 signifies a hardware master)
- MReqMstID[6-2]: Signifies the ID that has been assigned to the master by the NOC configuration

For master ID information refer to the chip specification.

4.11 Timestamping

The STM supports automatic timestamping upon receiving system trace messages. The timestamp is exported along with the message. Table 4-1 shows the structure of the STM memory space for multiple channels.

Table 4-1 STM Channel Memory Space

Byte Address		Data	Timestamp	Channel	Size
0x00000	0x007FF			0	4 Kbytes
0x00800	0x00FFF				
0x01000	0x017FF			1	4 Kbytes
0x01800	0x01FFF				
0x02000	0x027FF			2	4 Kbytes
0x02800	0x02FFF				
0x03000	0x037FF			3	4 Kbytes
0x03800	0x03FFF				
0xFF000	0xFF7FF			255	4 Kbytes
0xFF800	0xFFFFF				
End of Table 4-1					

The application software shall write to:

- The lower 2-kbyte window to trigger a data message without timestamping
- The upper 2-kbyte window to trigger a data message with timestamping

One-Kbyte address spaces can also be allocated for each channel, in which case writing to the upper 512 bytes would trigger a data message with timestamping. The length configuration is not exclusive - one master may instrument its application code through 4 kbyte channels and another may do so through 1 kbyte channels simultaneously. Both hardware and software masters may operate this way.

The STM also supports local timestamping where the timestamped messages are stored to the ETB. The user should enable local time-stamping mode to use this feature. In this case, the STM can receive one hardware message (32 bits) per cycle. Considering the hardware message only case, the STM adds 4 bits additional information to each hardware message or 12 bits additional information to each hardware message if the STM is configured to add a timestamp. The STM can send processed hardware messages to the TETB at 32 bits per cycle. Therefore, the sustainable rate is 8/9 hardware messages per cycle, assuming the timestamp is not added. The STM can buffer up to 128 full messages at a time. We can assume the STM consumes 8/9 messages per cycle and buffers 1/9 messages per cycle. Consequently, it can process up to 128*9 = 1152 back-to-back hardware messages. When the timestamp is added to the hardware message, the sustainable rate is 8/11 hardware messages per cycle and the STM should be able to process up to 128 / (1-8/11) = 469 hardware messages back-to-back.

When the STM runs out of buffer space, it will hold the reception of messages from the Tracer modules. This may cause overflow of hardware messages within the Tracer modules. The hardware messages do carry an overflow indication.

4.12 Message Format

Table 4-2 illustrates the message format for packets processed by the STM.

Table 4-2 STM Message Format

ID	Data	Mnemonic	Description
0001	M[7-0]	MASTER	Master ID
0010	O[7-0]	OVRF	Number of overflows
0011	C[7-0]	C8	Channel
0110	D[31-0]	D32	Data
1010	D[31-0] T[7-0]	D32TS	Data and timestamp

This message format is used by both software and hardware masters. Data words are stored internally in 44-bit packets within an internal FIFO before being exported through the ATB (for ETB) or PTI (for external trace receiver) interfaces.

Chapter 5

Tracer Architecture and Operation

This chapter provides an overview of the Tracer architecture and describes the operational concepts of the Tracer module, including event generation, message formatting/exporting, and integration with both the System Trace Module and TI Embedded Trace Buffer.

5.1 "Purpose of the Tracer" on page 5-2
5.2 "Features" on page 5-2
5.3 "Tracer Architecture" on page 5-3
5.4 "CBA Event Generation" on page 5-9
5.5 "Transaction Flow" on page 5-10
5.6 "Tracer Connections" on page 5-13
5.7 "FIFOs" on page 5-13
5.8 "Trace Message Formats" on page 5-13
5.9 "CBA Counters and Statistics" on page 5-16
5.10 "Throughput Counters" on page 5-16
5.11 "Sliding Time Window" on page 5-16
5.12 "Cross Triggering" on page 5-17
5.13 "Filtering" on page 5-17
5.14 "Accumulated Wait Time Counter" on page 5-17
5.15 "Num Grant Counter" on page 5-17
5.16 "Overflow Status" on page 5-17

5.1 Purpose of the Tracer

Tracer modules are used in conjunction with their supported SoC masters to generate system trace events. Corresponding messages are then generated based on these events and can be read to obtain information about the status and performance of the system.

5.2 Features

The following features are available for the Tracer modules:

- One hardware Tracer module for each supported slave interface, each containing the following:
  - Support for events A through G as described in Table 6-1 "Tracer Events"
  - Unique event inputs A, F, and G for each supported master
  - Event inputs B, C, and E shared among supported masters
  - Dedicated FIFOs for events A, B, C, and E
  - Message filters
  - Two throughput counters
  - Accumulated Wait Time counter
  - Num Grant counter
  - Message Formatter
  - VBUSP slave port for CorePac access to Tracer MMRs
  - VBUSP master port for access to STM port of Debug Subsystem
- Multiple Tracer modules on each Keystone II device as follows:
  - One for each core
  - One for each SRAM bank
  - One for EMIF
  - One for RAC
  - One for RAC_FE
  - One for TAC
  - One for Semaphore
  - Two for QM_SS
  - One for CFG SCR
  - Additional device specific Tracer modules. Please refer to individual data manuals for more information regarding the types of Tracers supported for a particular device.
- CBA event generators embedded in the SCR that generate master events for CorePacs, IPs, SRAMs, and the CFG SCR
- CBA slave event interfaces for each Tracer
- Capability to export three different types of messages from captured event data (event, statistics, and status messages)
- Tracers run on either CPU/2 or CPU/3 clock
- Multiple master connections to each slave Tracer module

5.3 Tracer Architecture

The Tracer module generates and exports system trace messages based on events triggered by the SoC masters that support it. The Tracer module is used to collect real-time data from a system by setting up various counters and filter modes provided by the module. It uses synchronous CBA event generators through four different event interfaces connected to the SCR to format and export trace messages. It also has the capability to provide various filter modes to the events and export the produced messages to the TI External Trace Buffer (TETB) inside the Debug Subsystem.

The Tracer also has statistics counters to monitor the real-time performance of the CBA infrastructure. These counters are also triggered by the CBA event generators built into the SCR. All event messages are sent through a VBUSP interface. Each Tracer connection is specified by a unique SID.

Figure 5-1 shows how the Tracer functional block is connected to other components of the SoC. The Tracer module supports connections from multiple supported masters to its CBA event inputs. Various masters trigger CBA event generators on their own respective modules according to which events are occurring. In addition to the original transaction requests sent from the masters to the targeted slave, these events are sent to the CBA slave interfaces on the Tracer module, where the Tracer forms messages to send to the STM for reformatting. Messages originating from the Tracers are 32 bits wide and are reformatted into larger packets within the STM. An additional 4 bits are used by the STM to signify that the message is a hardware message. The STM then sends messages to the TETB or to an external trace receiver through the Debug Port.

Figure 5-1 Tracer Connection

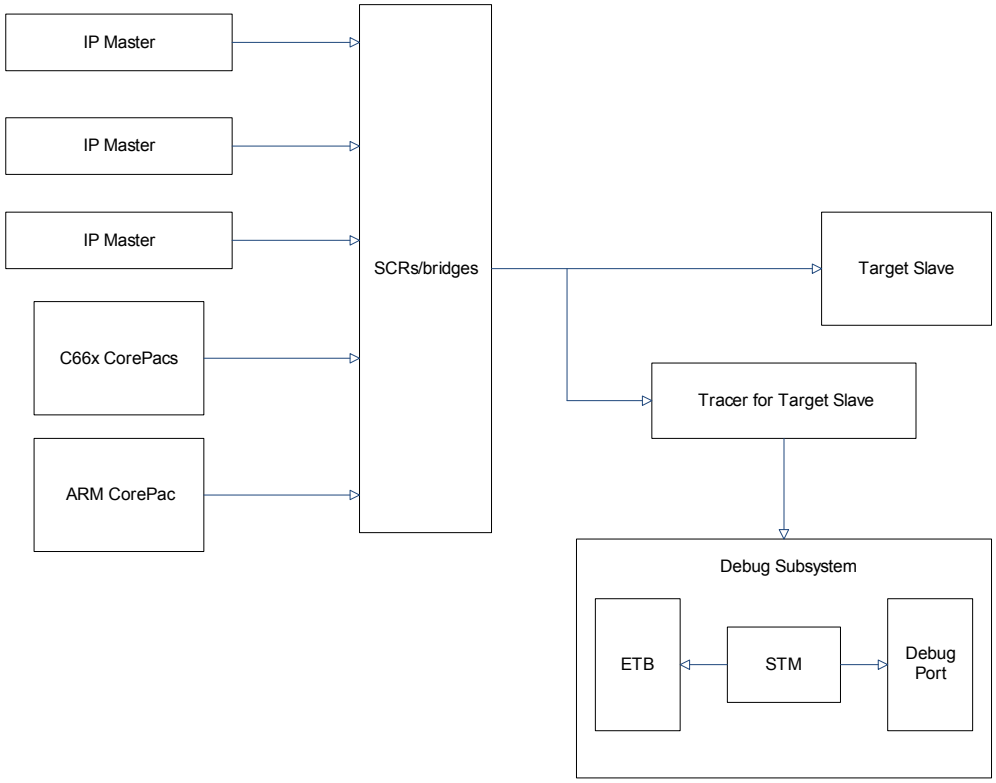
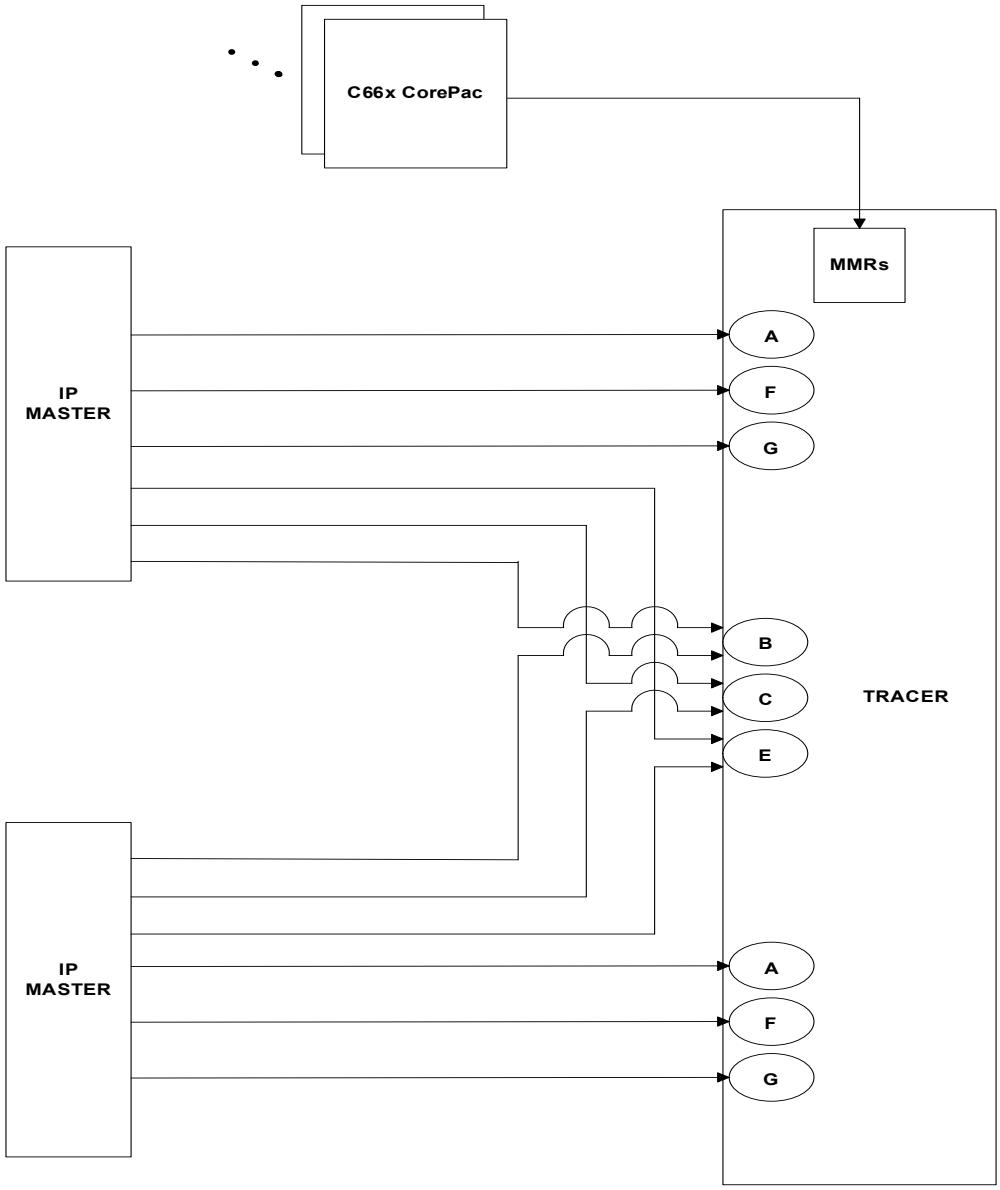


Figure 5-2 shows the individual CBA event connections from multiple IP masters to the Tracer for a particular slave. Each Tracer module has unique event A, F, and G inputs for each IP master it supports. For events B, C, and E, there is only one input for each of these events per Tracer module that is shared among supported IPs.

Figure 5-2 CBA Event Connections



5.3.1 Hardware Instrumentation through CPTracers

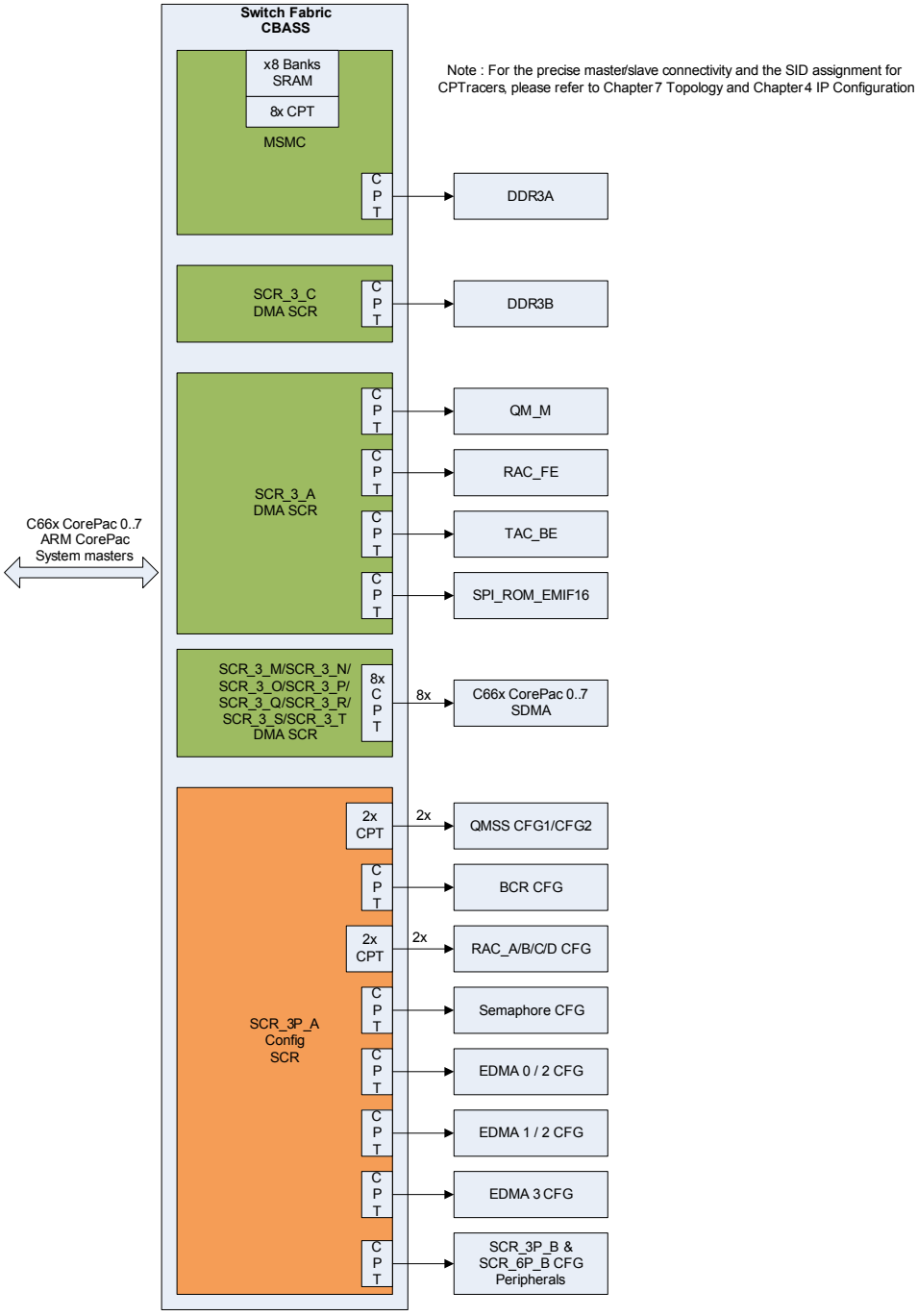
There are a large number of masters in the SoC, all of which drive data transactions through the switch fabric, a certain minimum amount of visibility is required to balance priority and scheduling for successful system integration and debug. The following are requirements both in a debugger environment (host control through JTAG) and in the embedded system (host control through application software)

- Resource (memory) usage – Ability to track the bandwidth consumed by the system masters.
  - Bandwidth (#bytes/time)
  - Provide two configurable system master groups, and bandwidth tracking for each group
- Transaction trace – Ability to monitor the data transactions occurring by each master, to selected slave interfaces through tracing of key transaction points:
  - Arbitration won
  - Last data phase (transaction complete)
  - Two filtering functions for transaction traces to bring out the specific transactions:
    - Transaction-qualifier-filtering: read/write
    - Address-range-based filtering
- Cross-triggering to turn on/off exporting capability from DSP/A15.

To support this visibility, the CP_Tracer module is integrated into the chip (Figure 5-3) to monitor and trace the transactions to each of the high-speed memory slave ports of the main SCR(s).This includes each of the DSP SDMA ports and DDR3A/DDR3B EMIF slave port and each memory bank in MSMC SRAM. The CP_Tracer is also used for tracing and monitoring the transactions sent to the configuration SCR peripherals.

Note—There is a bottleneck in the STM, such that it is not possible to activate all CPTracer instances in the device at the same time

Figure 5-3 Switch Fabric with CP_Tracer



The sliding time window specifies the measurement interval for all the CBA statistic counters implemented in the CP_Tracer module. The sliding time window is specified in terms of the CP_Tracer clock cycles. All the counters that are enabled start counting at the first transaction after the sliding window begins. When the sliding window timer expires, the counter values are loaded into the respective registers and the count starts

again. If enabled, an interrupt is also generated when the sliding time window expires. The host CPU and/or EDMA can read the statistics counters upon assertion of the interrupt. If enabled, the counter value can also be exported to STM automatically after the sliding time window is expired.

5.3.2 Event Interface

The CBA event generation logic embedded in an SCR can generate events based on the transaction occurring at a master interface. The events are generated for every transaction targeted towards a particular slave interface. The master interfaces and the target slave interface for which the events are generated can be configured when the SCR is generated. Care should be taken such that the same transaction does not generate multiple events as it traverses through different SCRs in a system.

CP_Tracer has interfaces to monitor and log the following events:

- New request event from master (Event A)
- New request event to slave (Event B)
- Last write data event from master (Event C)
- Last read data event to master (Event E)
- Write Merged (Event F)
- Command Discarded (Event G)

CP_Tracer module provides the following filtering features:

- Filtering based on mstid on events B and E
- Filtering based on read/write on event B
- Filtering based on dtype on event B
- Filtering based on address range (inclusive of addresses within the range and exclusive outside the range) on event B
- Filtering based on EMU0/1 control inputs on all events B, C and E

5.3.3 Software Instrumentation

There are cases during development of a complex application, where the stop mode debug (breakpoint-based) doesn't work well because no task or processor can be halted without affecting the correctness of entire system, and certain run-time debug method has to be used. Typically, these are the printf() or the trace. However there are problems associated with this:

- Printf() – too intrusive or won't be available for most embedded system.
- CPU trace – only limit to CPU, not all the processors in the system.

STM-based SW-message is a software instrumentation method, which is light-weight in term of intrusiveness, and is very easy to use. With it, an application directly writes data to be logged to some memory mapped location of STM, and STM will automatically capture and export the data to external receiver or to the on-chip CT-TBR buffer. To have this, all DSPs/A15s/QMGR PDSPs have a write access path to the memory space of STM, and all the processors should have a distinguished MstID[7:0]. Current STM implementation require MstID[7]=0 for SW message. Current design of STM supports 256 channels for each SW master. If up to 256 tasks (threads) running on same processor use different channels, they can log data to STM simultaneously without any mutex/semaphore protection.

5.3.4 Master ID for HW and SW Messages

The tables below describe master ID for the hardware and software masters of the STM.

Table 5-1 MSTID mapping for Hardware Instrumentation (CPTRACERS)

CPTracer Name	MSTID [7:0]	Clock domain	SID[4:0]	Description
CPT_MSMCx_MST, where x = 0..3	0x94-0x97	CPU/1	0x0..3	MSMC SRAM Bank 0 to MSMC SRAM Bank 3 monitors
CPT_MSMC4_MST	0xB1	CPU/1	0x4	MSMC SRAM Bank 4
CPT_MSMCx_MST, where x = 5..7	0xAE - 0xB0	CPU/1	0x5..7	MSMC SRAM Bank 5to MSMC SRAM Bank 7 monitors
CPT_DDR3A_MST	0x98	CPU/1	0x8	MSMC DDR3A port monitor
CPT_L2_x_MST, where x = 0..7	0x8C - 0x93	CPU/3	0x9..0x10	DSP 0 to 7 SDMA port monitors
CPT_TPCC0_4_MST	0xA4	CPU/3	0x11	EDMA 0 and EDMA 4 CFG port monitor
CPT_TPCC1_2_3_MST	0xA5	CPU/3	0x12	EDMA 1, EDMA2 and EDMA3 CFG port monitor
CPT_INTC_MST	0xA6	CPU/3	0x13	INTC port monitor (for INTC 0/1/2 and GIC400)
CPT_SM_MST	0x99	CPU/3	0x14	Semaphore CFG port monitors
CPT_QM_CFG1_MST	0x9A	CPU/3	0x15	QMSS CFG1 port monitor
CPT_QM_CFG2_MST	0xA0	CPU/3	0x16	QMSS CFG2 port monitor
CPT_QM_M_MST	0x9B	CPU/3	0x17	QM_M CFG/DMA port monitor
CPT_SPI_ROM_EMIF16_MST	0xA7	CPU/3	0x18	SPI ROM EMIF16 CFG port monitor
CPT_CFG_MST	0x9C	CPU/3	0x19	SCR_3P_B and SCR_6P_B CFG peripheral port monitors
CPT_NETCP_1_CFG	0xB2	CPU/3	0x1A	NetCP_1 CFG port CPtracer
CPT_RAC_FEI_MST	0x9D	CPU/3	0x1A	RAC_FE port monitor
CPT_RAC_CFG1_MST	0x9E	CPU/3	0x1B	RAC A/B CFG port monitor
CPT_TAC_BE_MST	0x9F	CPU/3	0x1C	TAC_BE port monitor
CPT_BCR_CFG_MST	0xA3	CPU/3	0x1D	BCR (RAC Broadcaster) CFG port monitor
CPT_RAC_CFG2_MST	0xA2	CPU/3	0x1E	RAC C/D CFG port monitor
CPT_DDR3B_MST	0xA1	CPU/3	0x1F	DDR 3B port monitor (on SCR 3C)
End of Table 5-1				

Table 5-2 MSTID Mapping for Software Messages

Core Name	MSTID [7:0]	Description
DSP0	0x0	DSP MDMA Master ID
DSP1	0x1	
DSP2	0x2	
DSP3	0x3	
DSP4	0x4	
DSP5	0x5	
DSP6	0x6	
DSP7	0x7	
A15 Core0	0x8	ARM Master IDs
A15 Core1	0x9	
A15 Core2	0xA	
A15 Core3	0xB	
QMSS PDSPs	0x46	All QMSS PDSPs share the same master ID. Differentiating between the 8 PDSPs is done through channel number used
End of Table 5-2		

5.4 CBA Event Generation

For any transaction that targets a supported slave interface, an event can be generated to monitor the transaction from the master to the slave through the CBA event generators of Tracer. The user can determine how these events are generated and how they are exported.

When transaction requests are generated from an SoC master to a particular slave, that master can send event signals to the Tracer module for the targeted slave. The Tracer module in turn will route information to the TETB within the Debug Subsystem or to an external trace receiver through the Debug Port. It is important to make sure the same transaction does not generate multiple events as it traverses the different SCRs in a system. Table 5-3 shows the different event types that can be triggered by Tracer.

Table 5-3 Tracer Events

Event	Action	Function
A	Master requesting to slave	This event triggers when there is a new request from the master decoded to the slave.
B	New request to slave	This event triggers when a transaction is sent to the slave.
C	Last write data to slave	This event triggers when the last write data is sent to the slave, thus completing the write burst.
E	Last read data from slave	This event triggers when the last read data arrives at the slave interface, thus completing the read burst.
F	Merge	Indicates that a write request from <mst> to <slv> has been merged with another request
G	Discard	Indicates that a read request from <mst> to <slv> has been discarded.
End of Table 5-3		

5.4.1 Event A - Master Request Event

Event A represents a master request transaction and is used for generating status and statistics messages from the Tracer. For example, consider a master issuing a write request to a target slave. The master could generate a transaction write request and at the same time generate an Event A on its CBA master interface which would then be sent to the CBA slave interface of the Tracer module for the targeted slave.

5.4.2 Event B - Arbitration Event

Event B represents a standard request to a slave that has been granted arbitration. With this event we can monitor many different transactions, so the user has the ability to establish filter modes to enable monitoring of the following:

- Trigger event when a transaction is sent to a slave. This signal interface does not track which master initiated the transaction. The Tracer uses dedicated event A CBA interfaces for each supported master for tracking the frequency of transactions from specific masters.
- Trigger event when the last transaction is arbitrated for a given command.
- Send the master ID associated with the given transaction.
- Send the arbitration direction associated with the given transaction.
- Send the dtype (0 for CPU Data Access, 1 for CPU Instruction Access, and 2 for DMA Access) associated with the given transaction.
- Send the transaction ID associated with the given transaction.
- Send the destination address associated with the given transaction.

- Send the byte count associated with the given transaction. This byte count is added to one of the throughput counters based upon how we configure the Transaction Qualifier Register. The throughput can be monitored based on a configurable timing window. The throughput gathered during the last timing window can then be read from one of two throughput registers, Throughput0 or Throughput1.

  The Transaction Qualifier Register and the Master ID Select Register N can be used to set any filtering modes for generating Event B messages.

5.4.3 Event C - Write Completion Event

Event C is triggered when the last set of write data is sent to the targeted slave. This event also signifies the end of the write burst.

5.4.4 Event E - Read Completion Event

Event E is triggered when the last set of read data is sent to the targeted slave, completing the read burst. The associated master ID and transaction ID is also sent with this event. The MasterID Select Register N can be used to set any filtering modes required for this event.

5.4.5 Event F - Write Merge Event

Event F indicates that a write request from <mst> to <slv> has been merged with another request.

5.4.6 Event G - Read Discard Event

Event G indicates that a read request from <mst> to <slv> has been discarded.

5.5 Transaction Flow

Figure 5-4 shows the overall transaction flow of a Tracer event. The event is generated from the master interface with the transaction request. The Tracer creates the hardware message from the event that is routed to the STM to be stored on-chip or exported externally.

Figure 5-4 Tracer Generation

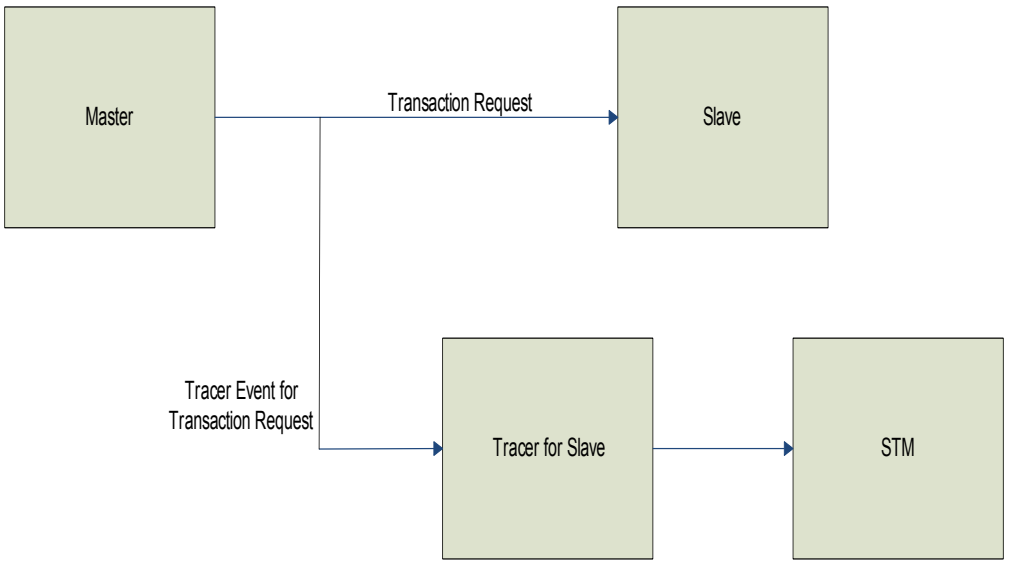


Figure 5-5 illustrates how an Event A can be generated and propagated to a Tracer module through a simple write to L2 memory.

Figure 5-5 Example—Event A to Tracer

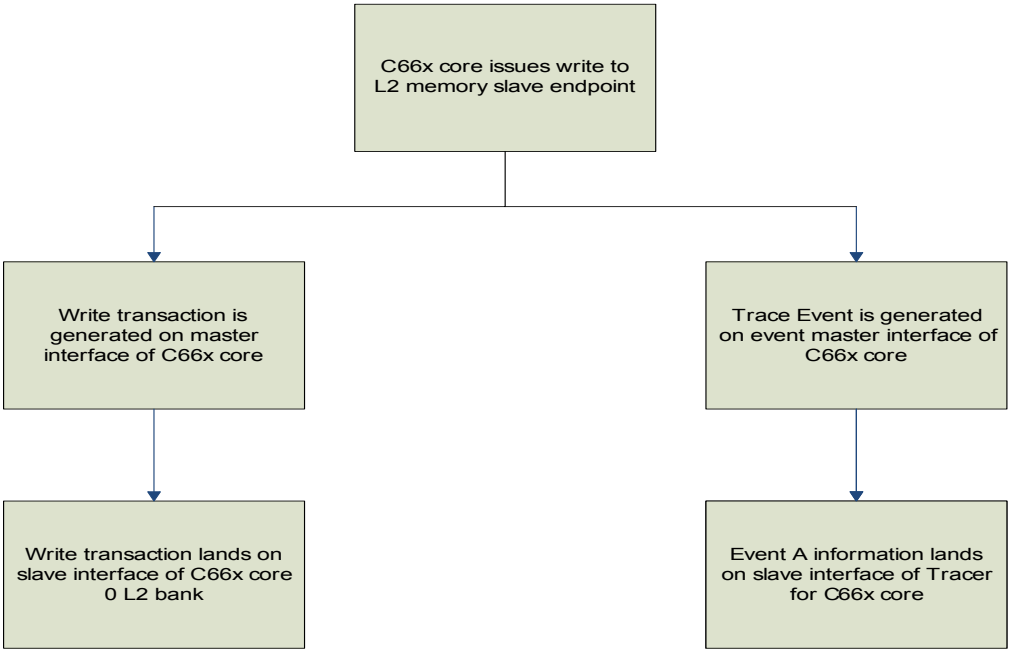
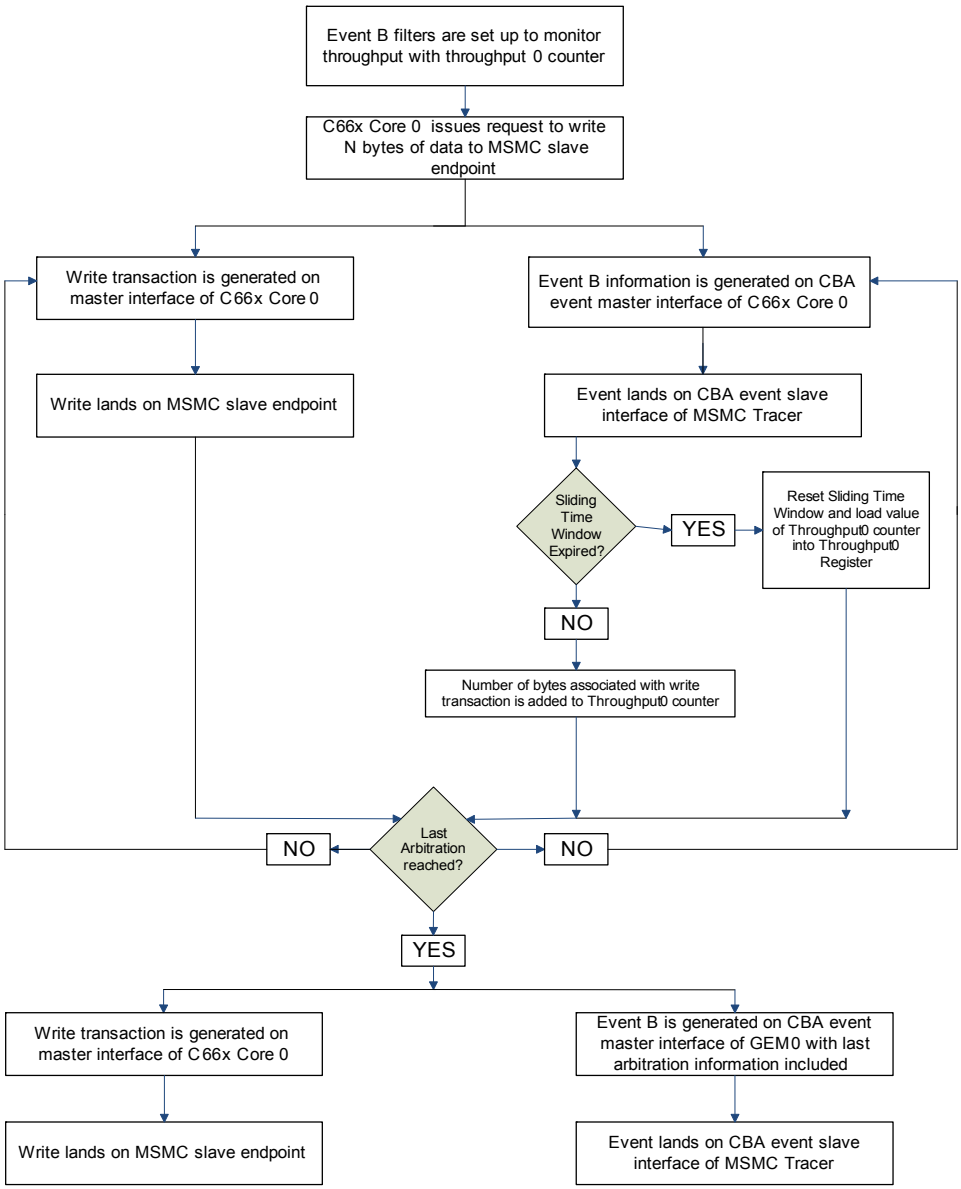


Figure 5-6 illustrates how an Event B is be generated and propagated through the Tracer module through a write to MSMC RAM.

Figure 5-6 Example—Event B to Tracer



Note—There is only one Event B interface for the Tracer module for the MSMC target that is shared amongst masters, whereas there are unique Event A interfaces on this module for each supported master.

5.6 Tracer Connections

The SID specifies the unique Tracer which has generated a particular message. The SID can also be determined from the generated messages of the STM. For information about the types of Tracers supported and their associated SID mapping, please refer to the device specific data manuals.

There exists different Tracer connections from various master ports to monitored slave interfaces for a particular device. For more information on master to slave connections for various Tracer modules of a particular device, please refer to the device specific data manual.

5.7 FIFOs

The Tracer modules implement the following FIFOs:

Event FIFO: This FIFO queues the event signals arriving at the CBA event slave interface of the Tracer.

Message FIFO: This FIFO queues the messages that are constructed by the Tracer logic after an event has been received. As long as there is at least one message pending in this FIFO, the VBUSP trace transfer controller will issue a 32-bit transfer to the STM.

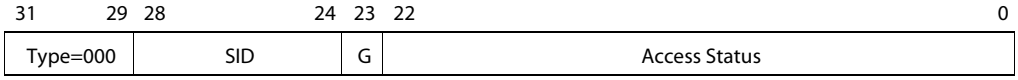
5.8 Trace Message Formats

The Tracer modules generate different message types according to the events they receive at their CBA event interfaces. These messages contain all the information that the corresponding events were generated and filtered on. Event A is recorded through status messages, whereas events B, C, and E are recorded through event messages. The following sections describe the formats for the different messages generated by the Tracers, STM, and TETB.

5.8.1 Status Messages

Status messages (Figure 5-7) are generated for Event A and provide information on request interaction between masters and slaves.

Figure 5-7 Status Message Format

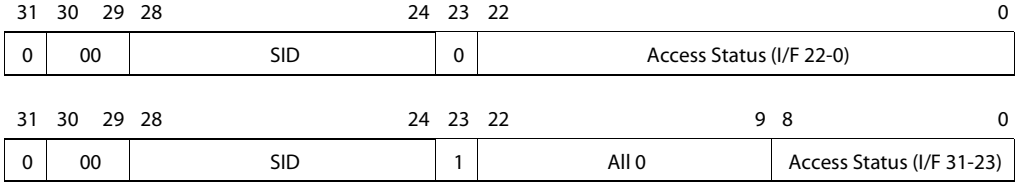


The SID specifies which unique Tracer this message is generated from.

The Access Status field is used to associate the transaction with a particular master event I/F. Each bit of the Access Status field signifies whether a request has happened on that particular Event A interface since the last status message was exported. A '0' means that no request event happened on that Event A interface since the last status message was exported. A '1' means that one or more request event happened on an event A interface since the last status message was exported.

As we can have up to 32 interfaces from which status messages are generated from and a limited amount of bits for representation, the status message can appear in two formats, shown in Figure 5-8.

Figure 5-8 Status Message Format

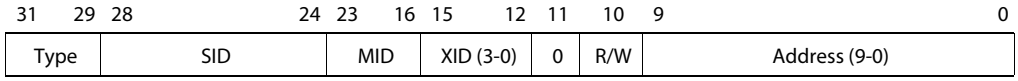


In the first configuration, bits 0-22 represent master interfaces 0-22. In the second, bits 0-8 represent the master interfaces 23-31. The two types are differentiated by the G field, bit 23.

5.8.2 Event Message

Event messages are generated for events B, C, and E. Figure 5-9 shows the message format for event messages.

Figure 5-9 Event Message Format



- The type field is encoded as following:
- 0b001: event B
- 0b010: event C
- 0b011: event E

The other fields are as follows:

- SID specifies which unique Tracer this message is generated from
- MID specifies the unique master ID from which the transaction originated to this Tracer slave endpoint.
- XID specifies the transaction ID.
- O indicates overflow status.
- R/W can be used to determine whether the transaction was read or write.
- The Address field can be used to determine the 10 bits of the associated 48-bit address that are exported with the event. The Address Mask Register is used to determine which 10 bits are exported.

Not all fields are valid for all event messages. For each event message the following fields are valid. All other fields are read as 0:

- Event B: Type, SID, MID, XID, O, RW, Address
- Event C: Type, SID, O
- Event E: Type, SID, MID, XID, O

5.8.3 Statistics Message

Each statistics message contains 4*32 bits as shown in Figure 5-10:

Figure 5-10 Statistic Message Format

31 29	28 24	23 0
Type=100	SID	Throughput_count0 (23-0)
Type=101	SID	Throughput_count1 (23-0)
Type=110	SID	Total_wait_time (23-0)
Type=111	SID	Total_granted (23-0)

Throughput_count0—This field equals the throughput from register Throughput0.

Throughput_count1—This field equals the throughput from register Throughput1.

Total_wait_time—This field is read from the Accumulated Wait time counter.

Total_granted—This field gives the reading of the grant counter which counts the number of times arbitration has been granted. The value is essentially equal to the number of B events that have occurred with arb_last asserted and is read from the Num Grant Counter.

Each of these fields is implemented by a dedicated counter in each Tracer and each start counting at the first transaction after the sliding time window begins. The sliding time window is explained in Section 5.11.

5.8.4 STM Message

The various messages from the Tracer modules are routed to the STM in the Debug Subsystem. Figure 5-11 shows how the messages originating from the Tracer modules will be encapsulated within STM.

Figure 5-11 STM Message Format

35 32	31 0
HWDAT	Tracer Message (Status, Event, or Statistics)

STM will pack incoming Tracer module data into 36-bit messages, 32 of which are data from the Tracer module and the most significant four bits are always 0b0110 to signify a HWDAT (Hardware Data) message.

5.9 CBA Counters and Statistics

Each Tracer's statistics counters allow the following statistics to be calculated:

- Bus bandwidth to slave used by one or more selected bus masters (bytes/sec.) = throughput for bus master X / sliding time window duration
- Average access size = throughput byte count / num accesses granted
- Bus utilization (transactions per second): = Num accesses granted / sliding time window duration
- Percentage of time there was contention for the bus = (accumulated wait time / sliding window length in cycles) * 100
- Minimum Average Latency 1 = Accumulated Wait Time / number of accesses
- Percentage of bus throughput used by bus master X = (throughput for bus master X / throughput for all bus masters) * 100

  where sliding time window duration = sliding time window period in cycles / # cycles per second

Note—The Tracers cannot measure average latency and average wait time. The Minimum Average Latency is not a true average arbitration latency, since it ignores the cycle counts where multiple bus masters are waiting at the same time. It will typically be lower than the true average latency.

5.10 Throughput Counters

Each Tracer module contains two throughput counters that can be enabled and filtered individually through specific registers. These counters are used in correspondence with Event B and exported as part of the statistics message. The throughput counters accumulate and record the total number of bytes forwarded to the target slave during the specified time duration. Filtering can be performed by address range, emu type, dtype, read/write transaction type, and master ID.

5.11 Sliding Time Window

Each Tracer module implements a sliding time window, measured in Tracer clock cycles, which specifies the total interval in which all CBA statistics counters are recorded. The Tracer clock cycles are tied to DMA cycles. In consequence, excluding CPU instruction or Data cycles in calculation will have no affect on the Tracer clock cycles computed. Excluding DMA cycles will exclude all Tracer cycles. When the sliding time window begins, all counters which are configured and enabled begin incrementing. When the sliding time window expires, these counter values are loaded

into their respective registers and the counts repeat again based on the sliding time window interval. When a periodic stream is enabled (statistic counters or access status) messages are generated on expiration of the sliding time window even if there was no bus activity.

The interrupt raw bit of the Interrupt Raw Status Register is set when the window expires.

5.12 Cross Triggering

By using EMU0 and EMU1 signals from any C66x core we have the ability to start and stop trace recording at specific times with the STM instead of around transaction events. Both EMU0/1 assert respective signals in the Debug Subsystem that control trace start and stop. The Transaction Qualifier Register allows us to determine whether we want to trace transactions between the assertion of EMU0 and EMU1 with the Qualif_EMU field.

5.13 Filtering

For generating statistics and event messages, there are several filtering modes which can be applied before exporting messages. Filtering can be applied based on:

- mstid on events B and E
- read/write on event B
- dtype on event B
- address range (inclusive of addresses within the range and exclusive outside the range) on event B. If any bytes of a transaction fall within an address range then the transaction will pass the inclusive filter or fail the exclusive filter.
- EMU0/1 control inputs on all events B, C and E

5.14 Accumulated Wait Time Counter

The accumulated-wait-time counter is a 24-bit counter used to accumulate the number of cycles spent waiting for a request to be serviced. It is exported as part of the statistics trace message. When a new request is issued from a master device to a slave, this event will be captured on the Event A interface of the Tracer for the corresponding slave and the number of pending requests will be incremented. When a transaction is sent to the slave for this request, this event will be captured on the Event B interface of the Tracer and the number of pending requests will be decremented. As long as there are pending requests, the accumulation cycle count will be incremented. Events E and F will also decrement the pending count.

5.15 Num Grant Counter

This is a 24-bit counter that is used to count the number of times arbitration has been granted (the number of event B that occurred with arb_last asserted). This counter is reset when the sliding timer window expires. This counter is enabled by a software register bit. This counter value is exported as part of the statistics trace message.

5.16 Overflow Status

Due to limits in the Tracer event FIFO size and the STM bandwidth, overflow may occur and event traces could be generated faster than they are drained. To deal with overflow, Tracer modules implement an overflow bit for events B, C, and E. When set to 1 in the corresponding trace message, the bit indicates that there has been some event loss between the current event and the previous event of the same type. A value of 0 indicates that no event loss has occurred.

Chapter 6

Programming Considerations

This chapter contains general instructions about how to initialize and set up the Tracer modules for message generation and export. Note that the CToolsLib exists to provide APIs for all the functionality described below and that the information that follows is only an overview of the internal operation and custom tool creation. See Chapter 8 “Programming Guidelines” on page 8-1 for more information on CToolsLib.

6.1 "Programming Overview" on page 6-2
6.2 "STM Configuration" on page 6-2
6.3 "ETB Configuration" on page 6-2
6.4 "Tracer Configuration" on page 6-2

6.1 Programming Overview

The STM must be configured in addition to setting up each Tracer. The following sequence is a basic programming model for configuring a Tracer:

1. Enable the Debug_SS subsystem.
2. Enable the STM.
3. Optionally, enable TETB if storing data on-chip.
4. Enable and configure the Tracer to create events and generate messages based on the specific transactions that need to be monitored for individual master-to-slave interfaces.
5. Initiate the desired transactions that need to be monitored.
6. Wait for the messages to be sent to the TETB or external trace receiver.
7. Read and parse the messages from the TETB, or view them with a Data Visualization Tool through CCS.

6.2 STM Configuration

The following sequence is a basic programming model for configuring the STM:

1. Enable STM writes by writing 0xC5ACCE55 to the Lock Access Register.
2. Claim ownership of the STM by writing 0x40000000 to the Software Master Control Register (SWMCTRL0).
3. Write 0xFFFFFF80 to the Hardware Master Control Register (HWMCTRL) register to enable trace for different hardware masters. This value will ensure that hardware master trace is enabled when the following are true:
   a. Bit [7] of Master Request Master ID Register = 1
   b. Bits [6-2] of Master Request Master ID Register matches HwMasterID0 of HWMCTRL
4. Enable the STM ATB by writing 0x00010000 to the ATB Configuration Register. Configuring the ATB is necessary for directing system trace data from the STM to the ETB. When ATB is enabled the following are true:
   a. Local timestamp is enabled
   b. Master Refresh is enabled
   c. PTI interface from STM is disabled
5. After configuring STM, write 0x80000000 to SWMCTRL0 to enable STM.

6.3 ETB Configuration

The following sequence is a basic programming model for configuring the ETB:

1. Enable trace capture in ETB by setting bit 0 of the ETB Control Register (ETB_CTL) to 1.
2. Enable TI MODE by setting bit 0 of TCTL to 1. This will allow the contents of the ETB Trace RAM to be stored in the RAM Read Data Register when trace capture is enabled.

6.4 Tracer Configuration

The following sequence is a basic programming model for configuring the Tracer:

1. Configure Tracer to capture specific events or throughput through the Transaction Qualifier Register.
2. Configure the pacing of exported messages through the Access Status Pacing Register.

3. Configure the Sliding Time Window Register to set the timeframe for which statistics are to be captured.
4. Configure the destination address for outgoing VBUSP transactions from the selected master interface to the address of the DEBUG_SS_STM data space (0x20000000).
5. Configure message export through the Module Control and Status Register.

After each of the modules is configured, the user can perform the procedures that the application requires and set the events which are to be triggered on (for example memory transfers). Tracer events will be generated and the user can poll the Access Status Register to determine if a particular transaction occurred on a particular master event I/F during the previous time sliding window. Access status is set by the hardware when event A arrives at the master event I/F. Access status is reset to 0 when the sliding time window expires.

After events are generated, if they are to be sent to the TETB then the messages can be read from the TETB RDR Register. The messages in the TETB RAM are generated by the STM in STM packet format. To read the messages correctly and interpret the information generated by the Tracers, the messages must be parsed after reading them from the RDR Register. The most significant four bits signify the data type. Typically, for Tracer messages they will take the value 0b0110 representing HWDAT. The master ID and data from event and status messages can then be gathered correctly.

Chapter 7

Registers

This chapter contains the register definitions for Tracers as well as definitions for supporting registers of STM and TETB.

7.1 "Debug Resource Manager Registers" on page 7-2
7.2 "Tracer Registers" on page 7-5
7.3 "STM Registers" on page 7-20
7.4 "TETB Registers" on page 7-27

7.1 Debug Resource Manager Registers

The following sections describe the registers required to configure the Debug Subsystem and debug pins for a specific purpose. Table 7-1 lists the register offsets for these registers.

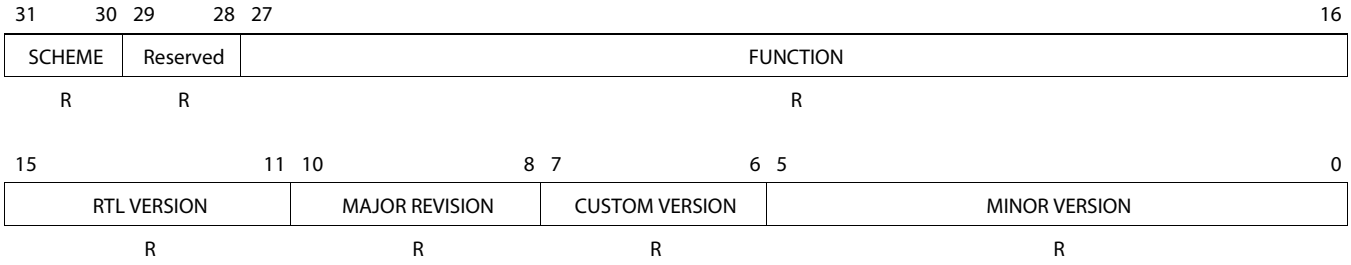
Table 7-1 DRM Register Offsets

Offset	Name
0x0	Peripheral ID Register
0x10	DRM Config Register
0x18	DRM Capability Register A
0x1C	DRM Capability Register B
0x50	DPM Claim Register
0x80 - 0xD0	DPM Control Register 0 - 19
End of Table 7-1	

7.1.1 Peripheral ID Register

The peripheral ID Register contains general information about the module.

Figure 7-1 Peripheral ID Register



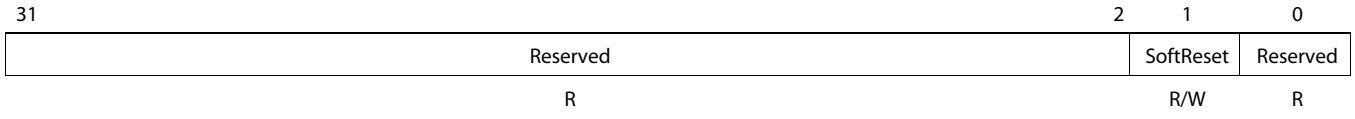
Legend: R = Read only; R/W = Read/Write

Table 7-2 Identification and Version Register Field Definitions

Bit	Field	Description
31-30	SCHEME	ID Scheme
29-28	Reserved	Reserved, read as 0
27-16	ID	Module Identification value
15-11	RTL VERSION	Module RTL version
10-8	MAJOR VERSION	Module major version
7-6	CUSTOM_VER	Custom Version
5-0	MINOR VERSION	Module minor version
End of Table 7-2		

7.1.2 DRM Configuration Register

Figure 7-2 DRM Configuration Register



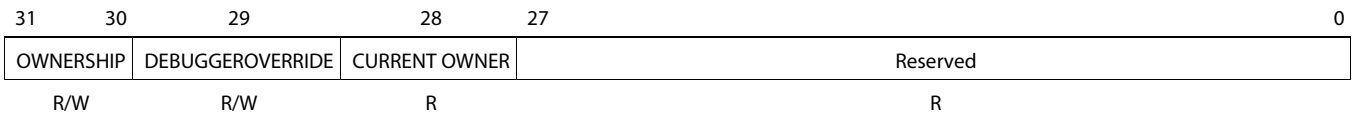
Legend: R/W = Read/Write

Table 7-3 DRM Configuration Register Field Definitions

Bit	Field	Description
31-2	Reserved	Reserved
1	SOFTRESET	1 - Writing a 1 to this field triggers DRM module reset 0 - Reads always return a 0
0	Reserved	Reserved
End of Table 7-3		

7.1.3 DPM Claim Register

Figure 7-3 DPM Claim Register



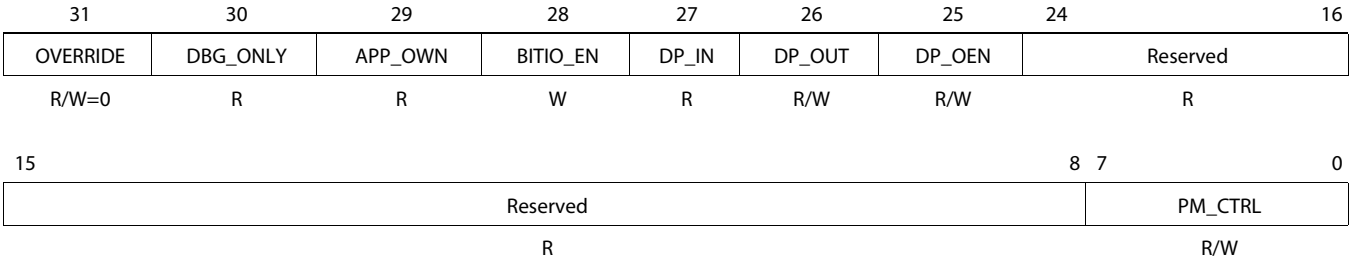
Legend: R/W = Read/Write

Table 7-4 DPM Claim Register Field Definitions

Bit	Field	Description
31-30	OWNERSHIP	Read to get current ownership status. Encoding is as follows: 00 - Available 01 - Claimed 10 - Enabled 11 - Reserved Send command to modify ownership state. A successful command causes this bit values to reflect the new state
29	DEBUGGEROVERRIDE	Used in conjunction with CLAIM request through ownership field. 0 - When written with DebuggerOverride=0, a claim request will be granted only if the unit is available. 1 - When written with DebuggerOverride=1, a claim request by the debugger will be granted regardless of the current ownership status of the unit. Reads always return a 1
28	CURRENT OWNER	0 - Indicates that the DP pin is debugger owned 1 - Indicates that the DP pin is application owned
27-0	Reserved	Reserved
End of Table 7-4		

7.1.4 DPM Control Register N

Figure 7-4 DPM Control Register N



31	30	29	28	27	26	25	24–16
OVERRIDE	DBG_ONLY	APP_OWN	BITIO_EN	DP_IN	DP_OUT	DP_OEN	Reserved
R/W=0	R	R	W	R	R/W	R/W	R

15–8	7–0
Reserved	PM_CTRL
R	R/W

Legend: R = Read only; R/W = Read/Write

Table 7-5 DPM Control Register N

Bit	Field	Description
31	OVERRIDE	0 - Indicates that reallocation to non-debug functions is allowed 1 - Indicates that reallocation to non-debug functions is not allowed
30	DBG_ONLY	0 - Indicates the DP pin is not solely used for debug function 1 - Indicates the DP pin is for debug function only
29	APP_OWN	0 - Indicates the DP pin is not application owned 1 - Indicates the DP pin is application owned.
28	BITIO_EN	Bit IO Enable bit
27	DP_IN	This bit shall indicate the current value of DP_In
26	DP_OUT	Writing 1 to this bit sets this pin as output.
25	DP_OEN	Writing 1 to this bit sets the output-enable for this pin
24-8	Reserved	Reserved
7-0	PM_CTRL	Pin manager control bits. For more information, see "Debug Pin Manager" on page 1-6.
End of Table 7-5		

7.2 Tracer Registers

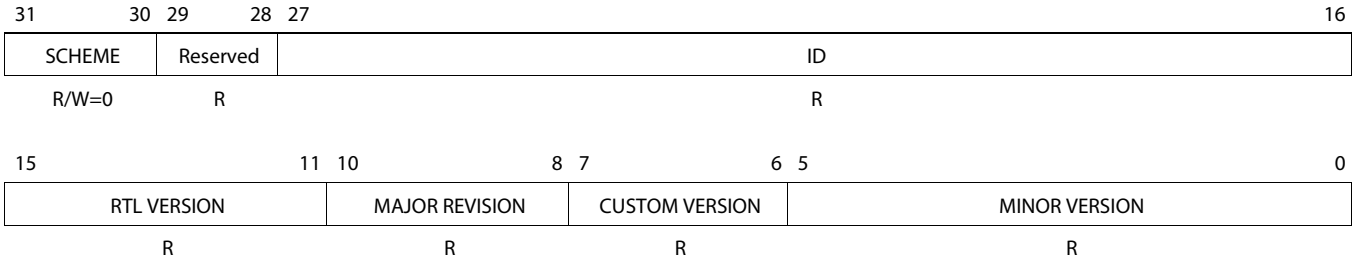
The following sections describe the registers required to configure the Tracer module and its events. Table 7-6 lists the register offsets for Tracer.

Table 7-6 Tracer Register Offsets

Offset	Name
0x0	Identification and Version Register
0x04	Transaction Qualifier Register
0x08	Module Control and Status Register
0x0C	Sliding Time Window
0x10	Master ID Select Register A
0x14	Master ID Select Register B
0x18	Master ID Select Register C
0x1C	Master ID Select Register D
0x20	Master ID Select Register A
0x24	Master ID Select Register B
0x28	Master ID Select Register C
0x2C	Master ID Select Register D
0x30	End Address Register
0x34	Start Address Register
0x38	Access Status Register
0x3C	Access Status Pacing Register
0x40	Address Mask Register
0x44	Destination Address Register
0x48	Message Priority Register
0x4C	Ownership
0x50	Throughput0
0x54	Throughput1
0x58	Accumulated Wait Time
0x5C	Number of Grants
0x60	Interrupt Raw Status Register
0x64	Interrupt Masked Status Register
0x68	Interrupt Mask Set Register
0x6C	Interrupt Mask Clear Register
0x70	End of Interrupt Register
End of Table 7-6	

7.2.1 Identification and Version Register

Figure 7-5 Identification and Version Register



Legend: R = Read only; R/W = Read/Write

The Identification and Version Register provides information on the various revisions of the Tracer module.

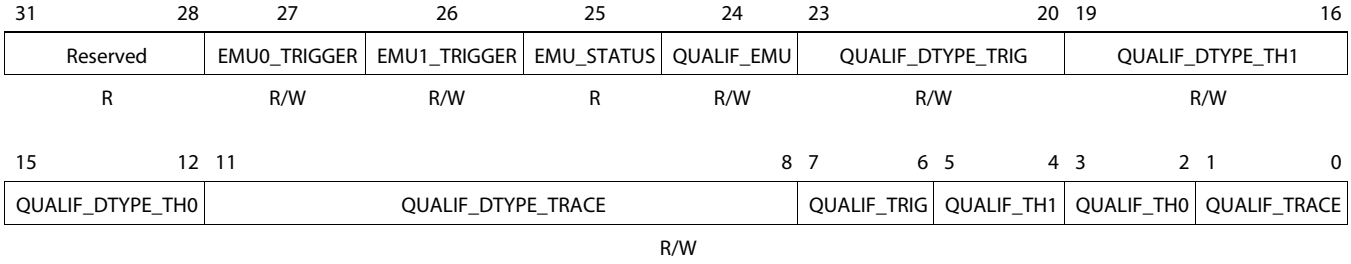
Table 7-7 Identification and Version Register Field Definitions

Bit	Field	Description
31-30	SCHEME	ID Scheme
29-28	Reserved	Reserved, read as 0
27-16	ID	Module Identification value1
15-11	RTL VERSION	Module RTL version
10-8	MAJOR VERSION	Module major version
7-6	CUSTOM_VER	Custom Version
5-0	MINOR VERSION	Module minor version
End of Table 7-7		

7.2.2 Transaction Qualifier Register

The Transaction Qualifier Register is used to set filtering modes for event B. Options on selecting which transactions are monitored with the two available throughput counters are provided, as well as cross triggering options and general filtering options for event B.

Figure 7-6 Transaction Qualifier Register



Legend: R = Read only; R/W = Read/Write

Table 7-8 Transaction Qualifier Register Field Definitions (Part 1 of 2)

Bit	Field	Description
31-28	Reserved	Reserved
27	EMU0_TRIGGER	1 - emu0_out is asserted when event B has occurred 0 - emu0_out is not asserted by Tracer
26	EMU1_TRIGGER	1 - emu1_out is asserted when qualified event B has occurred 0 - emu1_out is not asserted by Tracer
25	EMU_STATUS	Set by the hardware and read only 1 - Tracer is enabled to monitor CBA events 0 - Tracer is not enabled to monitor CBA events. When Qualif_EMU = 0, EMU_status is set to 1. When Qualif_EMU= 1, EMU_status is set to 1 when EMU0 is asserted. EMU_status is set to 0 when EMU1 is asserted.
24	QUALIF_EMU	1 - only the CBA events happening between assertion of EMU0 and assertion of EMU1 are sent to qualifier. 0 - all CBA events sent to qualifier
23-20	QUALIF_DTYPE_TRIG	0bxx0x - Capture event Bs with dtype 1 (CPU Instruction Access) for emu0/1_out triggering 0bxx1x - Exclude event Bs with dtype 1 (CPU Instruction Access) from emu0/1_out triggering 0bx0xx - Capture event Bs with dtype 2 (DMA Access) for emu0/1_out triggering 0bx1xx - Exclude event Bs with dtype 2 (DMA Access) from emu0/1_out triggering 0b0xxx - Capture event Bs with dtype 3 (Reserved) for emu0/1_out triggering 0b1xxx - Exclude event Bs with dtype 3 (Reserved) from emu0/1_out triggering
19-16	QUALIF_DTYPE_TH1	0bxxx1 - Exclude event Bs with dtype 0 (CPU Data Access) from throughput 1 calculation 0bxx0x - Capture event Bs with dtype 1 (CPU Instruction Access) for throughput 1 calculation 0bxx1x - Exclude event Bs with dtype 1 (CPU Instruction Access) from throughput 1 calculation 0bx0xx - Capture event Bs with dtype 2 (DMA Access) for throughput 1 calculation 0bx1xx - Exclude event Bs with dtype 2 (DMA Access) from throughput 1 calculation 0b0xxx - Capture event Bs with dtype 3 (Reserved) for throughput 1 calculation 0b1xxx - Exclude event Bs with dtype 3 (Reserved) from throughput 1 calculation
15-12	QUALIF_DTYPE_TH0	0bxxx0 - Capture event Bs with dtype 0 (CPU Data Access) for throughput 0 calculation 0bxxx1 - Exclude event Bs with dtype 0 (CPU Data Access) from throughput 0 calculation 0bxx0x - Capture event Bs with dtype 1 (CPU Instruction Access) for throughput 0 calculation 0bxx1x - Exclude event Bs with dtype 1 (CPU Instruction Access) from throughput 0 calculation 0bx0xx - Capture event Bs with dtype 2 (DMA Access) for throughput 0 calculation 0bx1xx - Exclude event Bs with dtype 2 (DMA Access) from throughput 0 calculation 0b0xxx - Capture event Bs with dtype 3 (Reserved) for throughput 0 calculation 0b1xxx - Exclude event Bs with dtype 3 (Reserved) from throughput 0 calculation
11-8	QUALIF_DTYPE_TRACE	0bxxx0 - Capture event Bs with dtype 0 (CPU Data Access) for event B export 0bxxx1 - Exclude event Bs with dtype 0 (CPU Data Access) from event B export 0bxx0x - Capture event Bs with dtype 1 (CPU Instruction Access) for event B export 0bxx1x - Exclude event Bs with dtype 1 (CPU Instruction Access) from event B export 0bx0xx - Capture event Bs with dtype 2 (DMA Access) for event B export 0bx1xx - Exclude event Bs with dtype 2 (DMA Access) from event B export 0b0xxx - Capture event Bs with dtype 3 (Reserved) for event B export 0b1xxx - Exclude event Bs with dtype 3 (Reserved) from event B export

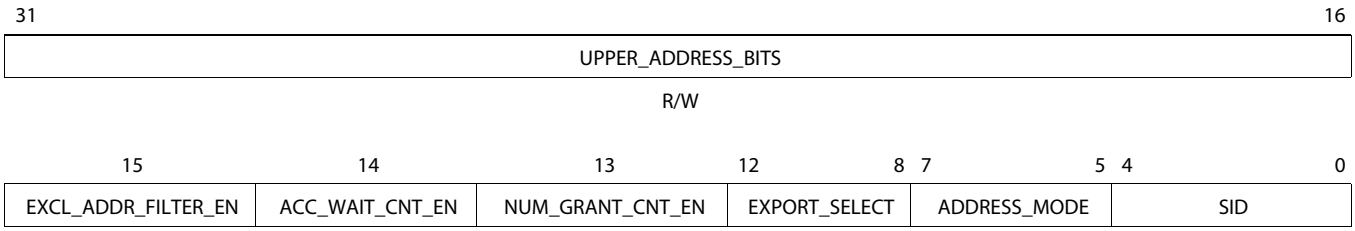
Table 7-8 Transaction Qualifier Register Field Definitions (Part 2 of 2)

Bit	Field	Description
7-6	QUALIF_TRIG	0b11 - Capture both read and write transactions for emu0/1_out trigger
		0b01 - Only capture read transactions for emu0/1_out trigger
		0b10 - Only capture write transactions for emu0/1_out trigger
		0b00 - Do not capture either read or write transaction for emu0/1_out trigger (Disables emu0/1_out triggering)
5-4	QUALIF_TH1	0b11 - Capture both read and write transactions for throughput 1
		0b01 - Only capture read transactions for throughput 1
		0b10 - Only capture write transactions for throughput 1
		0b00 - Do not capture either read or write transaction for throughput 1 (Disables throughput count 1)
3-2	QUALIF_TH0	0b11 - Capture both read and write transactions for throughput
		00b01 - Only capture read transactions for throughput
		00b10 - Only capture write transactions for throughput
		00b00 - Do not capture either read or write transaction for throughput 0 (Disables throughput count 0)
1-0	QUALIF_TRACE	0b11 - Capture both read and write transactions for event B trace export
		0b01 - Only capture read transactions for event B trace export
		0b10 - Only capture write transactions for event B trace export
		t0b00 - Do not capture either read or write transaction for event B trace export
End of Table 7-8		

7.2.3 Module Control and Status Register

The Module Control and Status Register provides address filtering, counter enabling, and export options to select which captured messages are forwarded from the internal Tracer FIFO to the STM.

Figure 7-7 Module Control and Status Register



Legend: R = Read only; R/W = Read/Write

Table 7-9 Module Control and Status Register Field Definitions (Part 1 of 2)

Bit	Field	Description
31-16	UPPER_ADDRESS_BITS	These are the upper address bits used for all address filtering. They function as the MSBs [47-32] of both start address and end address. See the definition of the address_mode field for which bits of this address are valid. Only the valid bits as indicated by the address_mode field are writable, all other bits are read-only 0.
15	EXCL_ADDR_FILTER_EN	1 - Enable exclusive address filtering
		0 - Disable exclusive address filtering
14	ACC_WAIT_CNT_EN	1 - Enables accumulated wait counter
		0 - Disables accumulated wait counter
13	NUM_GRANT_CNT_EN	1 - Enables num grant counter
		0 - Disables num grant counter

Table 7-9 Module Control and Status Register Field Definitions (Part 2 of 2)

Bit	Field	Description
12-8	EXPORT_SELECT	0b00000 - no export capability 0bxxxx1 - export statistics message when the sliding time window is expired. 0bxxx1x - export event trace based on filtering for event B 0bxx1xx - export event trace for event C 0bx1xxx - export event trace for event E 0b1xxxx - export access status message based on pacing
7-5	ADDRESS_MODE	This register reflects the value of the address_mode tie off and is read-only. The value indicates how many address bits on the event_<slv>_arb_address are valid. For example, if it indicates 32 bits then bits 31-0 are used for address filtering and all others must be tied to 0. 36 bits means 35-0 are used, and so on. 0'b000 - 32 bit address 0'b001 - 36 bit address 0'b010 - 40 bit address 0'b011 - 44 bit address 0'b100 - 48 bit address Other values Reserved Note: event_<slv>_arb_address bits that are not used must be tied to zero. If they are non-zero, address filtering will not work correctly.
4-0	SID	Read only. It is used to differentiate the Tracer modules if a device has multiple Tracers on chip. It is a tie-off value.
End of Table 7-9		

7.2.4 Sliding Time Window

The Sliding Time Window specifies the duration in which to record statistics events which have been configured by the user. When the window expires the values of all Tracer counters are loaded into their respective registers.

Figure 7-8 Sliding Time Window Register

31	0
SLIDING TIME WINDOW	
R/W	

Legend: R/W = Read/Write

Table 7-10 Sliding Time Window Register Field Definitions

Bit	Field	Description
31-0	SLIDING TIME WINDOW	Specifies a sliding window in terms of the Tracer clock over which the statistics are to be collected. An interrupt is generated (if enabled) when the sliding time window expires. Value is in Tracer clocks Counter starts as soon as a non-zero value is written to this register Writing a 0x00000000 to this register will disable the sliding time window counter Counter is disabled at reset despite having a non-zero value
End of Table 7-10		

7.2.5 Master ID Select Register A for Throughput0 and Event Trace

The Master ID Select Registers A through D for Throughput0 and Event Trace allows the user to select which master IDs are used in Throughput0 calculation. It also allows selection of which master IDs for event B and E are exported to STM.

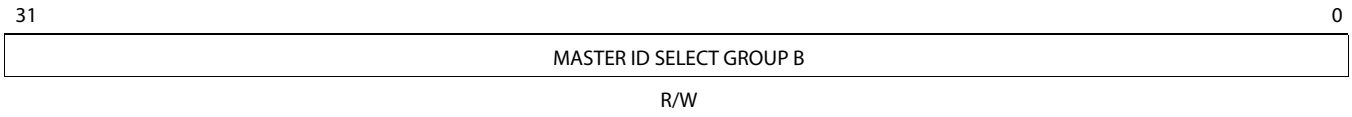
Figure 7-9 Master ID Select Group A Register

Legend: R/W = Read/Write

Table 7-11 Master ID Select Group A Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP A	Master ID select Group A contains the master ID select for master ID 0 to 31. Bit 0 enables mstid 0, bit 1 enables mstid 1 etc. If enabled, the corresponding mstid is selected for throughput0 calculation. The selected mstid is also used to filter event B and E for event trace export.

Figure 7-10 Master ID Select Group B Register



Legend: R/W = Read/Write

Table 7-12 Master ID Select Group B Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP B	Master ID select Group B contains the master ID select for master ID 32 to 63. Bit 0 enables mstid 32, bit 1 enables mstid 33 etc. If enabled, the corresponding mstid is selected for throughput0 calculation. The selected mstid is also used to filter event B and E for event trace export.
End of Table 7-12		

Figure 7-11 Master ID Select Group C Register

Legend: R/W = Read/Write

Table 7-13 Master ID Select Group C Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP C	Master ID select Group C contains the master ID select for master ID 64 to 95. Bit 0 enables mstid 64, bit 1 enables mstid 65 etc. If enabled, the corresponding mstid is selected for throughput0 calculation. The selected mstid is also used to filter event B and E for event trace export.

Figure 7-12 Master ID Select Group D Register

Legend: R/W = Read/Write

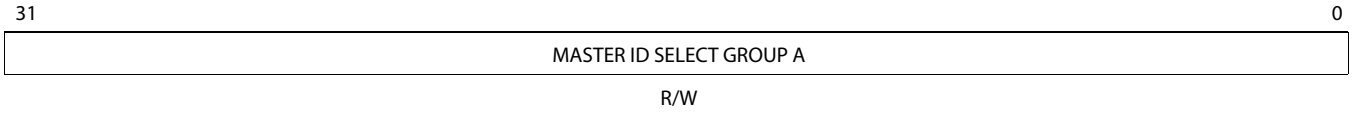
Table 7-14 Master ID Select Group D Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP D	Master ID select Group D contains the master ID select for master ID 96 to 127. Bit 0 enables mstid 96, bit 1 enables mstid 97 etc. If enabled, the corresponding mstid is selected for throughput0 calculation. The selected mstid is also used to filter event B and E for event trace export.
End of Table 7-14		

7.2.6 Master ID Select Register N for Throughput1 Calculation

The Master ID Select Registers A-D for Throughput1 calculation allows the user to select which master IDs are included in calculation for Throughput 1.

Figure 7-13 Master ID Select Group A Register

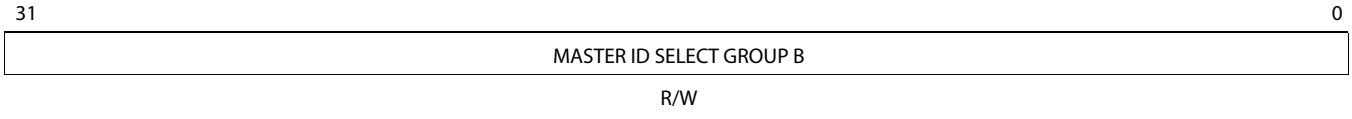


Legend: R/W = Read/Write

Table 7-15 Master ID Select Group A Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP A	Master ID select Group A contains the master ID select for master ID 0 to 31. Bit 0 enables mstid 0, bit 1 enables mstid 1 etc. If enabled, the corresponding mstid is selected for throughput1 calculation.
End of Table 7-15		

Figure 7-14 Master ID Select Group B Register

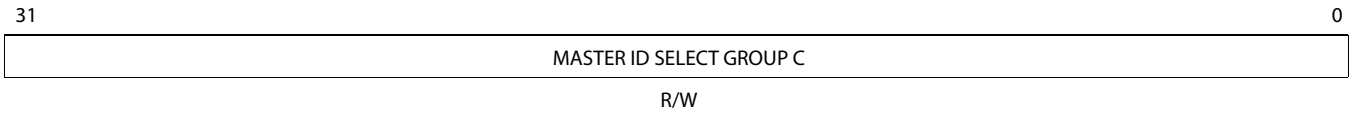


Legend: R/W = Read/Write

Table 7-16 Master ID Select Group B Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP B	Master ID select Group B contains the master ID select for master ID 32 to 63. Bit 0 enables mstid 32, bit 1 enables mstid 33 etc. If enabled, the corresponding mstid is selected for throughput1 calculation.
End of Table 7-16		

Figure 7-15 Master ID Select Group C Register

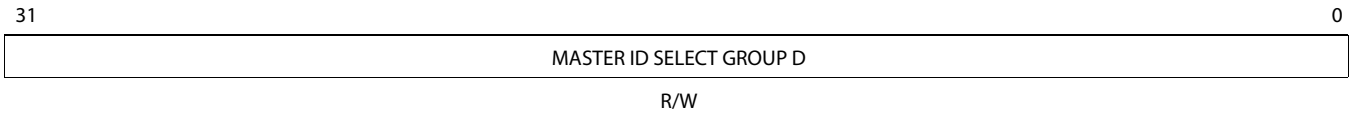


Legend: R/W = Read/Write

Table 7-17 Master ID Select Group C Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP C	Master ID select Group C contains the master ID select for master ID 64 to 95. Bit 0 enables mstid 64, bit 1 enables mstid 65 etc. If enabled, the corresponding mstid is selected for throughput1 calculation.
End of Table 7-17		

Figure 7-16 Master ID Select Group D Register



Legend: R/W = Read/Write

Table 7-18 Master ID Select Group D Register Field Definitions

Bit	Field	Description
31-0	MASTER ID SELECT GROUP D	Master ID select Group D contains the master ID select for master ID 96 to 127. Bit 0 enables mstid 96, bit 1 enables mstid 97 etc. If enabled, the corresponding mstid is selected for throughput1 calculation.
End of Table 7-18		

7.2.7 Address Window Registers

The Start and End Address Registers are used in conjunction to set the address window for filtering throughput statistics capture (for Throughput0 and Throughput1) and message exporting of event B. The address range can be set to inclusive or exclusive. Transactions with address ranges falling within the range (or outside of the range) will be added to throughput or exported. When End Address is set to 0x0, address filtering is disabled.

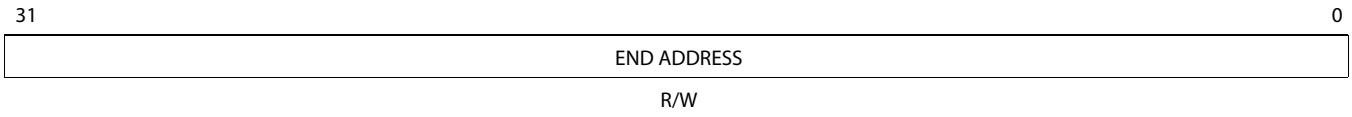
Figure 7-17 Start Address Register

Legend: R/W = Read/Write

Table 7-19 Start Address Register Field Definitions

Bit	Field	Description
31-0	START ADDRESS	Memory address for the low address range.
End of Table 7-19		

Figure 7-18 End Address Register



Legend: R/W = Read/Write

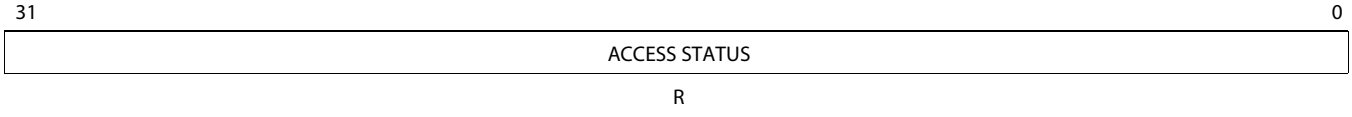
Table 7-20 End Address Register Field Definitions

Bit	Field	Description
31-0	END ADDRESS	Memory address. Combined with the start address register, it provides the address range for filtering function for both throughput statistics and event B export. Only the transaction to the address range between and including [start address, end address] will be captured for throughput. For event trace export, the event which overlaps [start address, end address] will be captured to export. When the end address is set to 0x0, it disables the address filtering function. When the address filtering is disabled, the transactions are captured regardless of the memory range.
End of Table 7-20		

7.2.8 Access Status Register

The Access Status Register is used to monitor when transactions arrive on any of the 32 master event I/Fs of Tracer.

Figure 7-19 Access Status Register

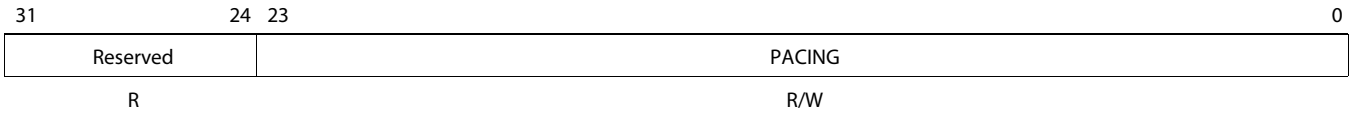


Legend: R = Read only

Table 7-21 Access Status Register Field Definitions

Bit	Field	Description
31-0	ACCESS STATUS	Each bit is correspondent to one master event I/F. 1 - means a transaction arrived in that master event I/F during the previous time sliding window 0 - means no transactions arrived in that master event I/F during the previous time sliding window The access status is set by the hardware when event A arrives at the master event I/F. The access status is reset to 0x0 when the sliding window is expired.
End of Table 7-21		

Figure 7-20 Access Status Pacing Register



Legend: R/W = Read/Write

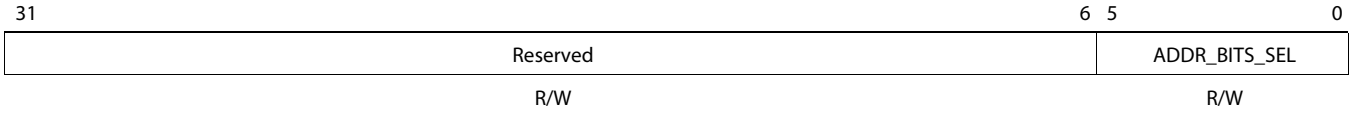
Table 7-22 Access Status Pacing Register Field Definitions

Bit	Field	Description
31-24	Reserved	Reserved
23-0	PACING	0x000000 - If export_select[4] is set and an access has occurred since the previous access status message was sent, then the access status message is sent out whenever the sliding time window expires. 0xXXXXXx - If this field is non-zero, export_select[4] is set, and an access has occurred since the previous access status message was sent, then the access status message is sent out as long as X cycles have occurred since the last message where X is the value of this field. NOTE: if the sliding time window expires, then the access status message will be sent out regardless of pacing, and the pacing counter will be reset. If this value is greater than the sliding time window then the access status message will only be sent out whenever the sliding time window expires.
End of Table 7-22		

7.2.9 Address Mask Register

The Address Mask Register is used to select which address bits of the event address to export.

Figure 7-21 Address Mask Register



Legend: R/W = Read/Write

Table 7-23 Address Mask Register Field Definitions Field Definitions

Bit	Field	Description
31-6	Reserved	Reserved
5-0	ADDR_BITS_SEL	Event messages export 10 bits of the address associated with the event. This field indicates which 10 contiguous bits of the possible 48 bits are exported. The value indicates the lowest bit and the next 9 bits are exported also: 0 – Bits 09-0 are exported 1 – Bits 10-1 are exported 2 – Bits 11-2 are exported 3-36 - ….. 37 – Bits 46-37 are exported 38 – Bits 47-38 are exported 63-38 – Reserved. Behavior undefined.
End of Table 7-23		

7.2.10 Destination Address Register

The Destination Address Register specifies the destination address for outgoing transactions from the Tracer VBUSP master interface. Normally we want to set this address to 0x20000000 to route packets from Tracer to the DEBUG_SS_STM data space.

Figure 7-22 Destination Address Register

Legend: R/W = Read/Write

Table 7-24 Destination Address Register Field Definitions

Bit	Field	Description
31-2	DESTINATION ADDRESS 31-2	Destination Address bits 31-2 for the outgoing VBUSP transactions from the master interface. The address must be word aligned, so the 2 LSBS (1-0) are always 0
1-0	Reserved	Reserved. 2 LSBs of the destination address must be aligned
End of Table 7-24		

7.2.11 Message Priority Register

The Message Priority Register sets the priorities associated with exporting different message types.

Figure 7-23 Message Priority Register

31 15	14 12	11 9	8 6	5 3	2 0
Reserved	STAT_MSG_PRIO	EVENT_E_MSG_PRIO	EVENT_C_MSG_PRIO	EVENT_B_MSG_PRIO	ACCESS_MSG_PRIO
R	R/W	R/W	R/W	R/W	R/W

Legend: R = Read only; R/W = Read/Write

Table 7-25 Message Priority Register Field Definitions

Bit	Field	Description
31-15	Reserved	Reserved. Read as 0
14-12	STAT_MSG_PRIO	Vbusp priority and epriority associated with statistics messages
11-9	EVENT_E_MSG_PRIO	Vbusp priority and epriority associated with event e message export
8-6	EVENT_C_MSG_PRIO	Vbusp priority and epriority associated with event c message export
5-3	EVENT_B_MSG_PRIO	Vbusp priority and epriority associated with event b message export
2-0	ACCESS_MSG_PRIO	Vbusp priority and epriority associated with access status message export
End of Table 7-25		

7.2.12 Ownership

The Ownership Register is used for software to claim ownership of the Tracer module for the purpose of avoiding race conditions.

Figure 7-24 Ownership Register

31 1	0
Reserved	OWN
R	R/W

Legend: R = Read only

Table 7-26 Ownership Register Field Definitions

Bit	Field	Description
31-1	Reserved	Reserved. Read as 0
0	OWN	This bit can be used by software to indicate that this Tracer is “owned” by a process for configuration. It is simply a read/write register that is read as the last value written. It provides no interlock capability so software is responsible for implementing a semaphore using this bit and handle any possible race conditions that may occur.
End of Table 7-26		

7.2.13 Throughput0

Throughput0 register is loaded with the value of the Throughput0 counter after each expiration of the sliding time window. From this register the user can read the value of the accumulated throughput based on filtered events during the last sliding time window.

Figure 7-25 Throughput0 Register

Legend: R = Read only

Table 7-27 Throughput0 Register Field Definitions

Bit	Field	Description
31-24	Reserved	Reserved
23-0	THROUGHPUT0	The accumulated throughput during the last sliding time window based on all throughput0 qualifiers and filters.
End of Table 7-27		

7.2.14 Throughput1

Throughput1 register is loaded with the value of the Throughput1 counter after each expiration of the sliding time window. From this register the user can read the value of the accumulated throughput based on filtered events during the last sliding time window.

Figure 7-26 Throughput1 Register

Legend: R = Read only

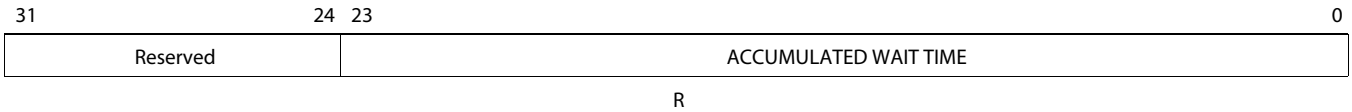
Table 7-28 Throughput1 Register Field Definitions

Bit	Field	Description
31-24	Reserved	Reserved
23-0	THROUGHPUT1	The accumulated throughput during the last sliding time window based on all throughput1 qualifiers and filters.
End of Table 7-28		

7.2.15 Accumulated Wait Time

The accumulated Wait Time register is loaded with the value of the Accumulation counter after each expiration of the sliding time window. This value gives the accumulated throughput based on filtered events.

Figure 7-27 Accumulated Wait Time



Legend: R = Read only

Table 7-29 Accumulated Wait Time Field Definitions

Bit	Field	Description
31-24	Reserved	Reserved
23-0	ACCUMULATED WAIT TIME	The accumulated throughput during the last sliding time window based on all throughput1 qualifiers and filters.
End of Table 7-29		

7.2.16 Number of Grants

The Number of Grants register is loaded with the value of the Num Grant counter after each expiration of the sliding time window. This value gives the number of times that a transaction was granted arbitration. For instance, as Event A signifies a master generating a request to a slave, and Event B for the same transaction signifies the transaction being sent to the slave, then every time an Event B with arrives with an active arb_last signal, the grant counter will be incremented.

Figure 7-28 Number of Grants Register

Legend: R = Read only

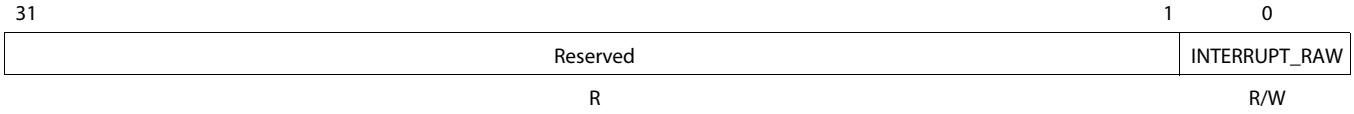
Table 7-30 Number of Grants Register Field Definitions

Bit	Field	Description
31-24	Reserved	Reserved
23-0	NUMBER OF GRANTS	The number of times that arbiter made grants during a sliding timing window. It is set when the sliding timing window is expired.
End of Table 7-30		

7.2.17 Interrupt Raw Status

The Interrupt Raw Status Register is used to determine if the sliding time window has expired.

Figure 7-29 Interrupt Raw Status Register



Legend: R = Read only

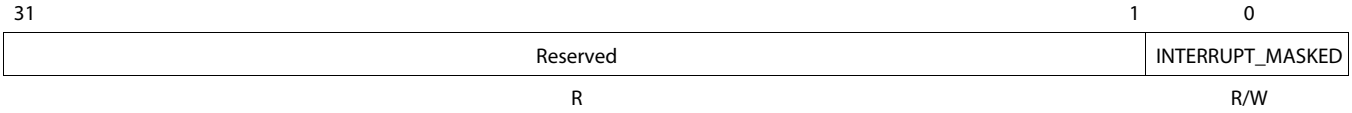
Table 7-31 Interrupt Raw Status Register Field Definitions

Bit	Field	Description
31-1	Reserved	Reserved
0	INTERRUPT_RAW	This bit is set when the sliding time window expires regardless of the interrupt mask set bit in the Interrupt Mask Set register. This bit remains set until the host CPU clears it. Writing a 1 to this bit sets it and writing a 0 has no effect.
End of Table 7-31		

7.2.18 Interrupt Masked Status

The Interrupt Masked Status Register is used to determine if the sliding time window has expired.

Figure 7-30 Interrupt Masked Status Register



Legend: R = Read only

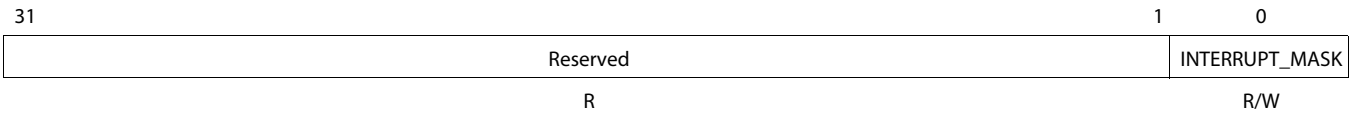
Table 7-32 Interrupt Masked Status Register Field Definitions

Bit	Field	Description
31-1	Reserved	Reserved
0	INTERRUPT_MASKED	This bit is set when the sliding time window expires and the interrupt mask set bit in the Interrupt Mask Set register is set. This bit remains set until the host CPU clears the raw status bit. This bit also drives the module output port. Writing a 1 to this bit clears the raw status (and therefore this status) and writing a 0 has no effect.
End of Table 7-32		

7.2.19 Interrupt Mask Set

The interrupt Mask Set register is used to enable the Interrupt_Masked status bit of the Interrupt Masked Status register.

Figure 7-31 Interrupt Mask Set Register



Legend: R = Read only; R/W = Read/Write

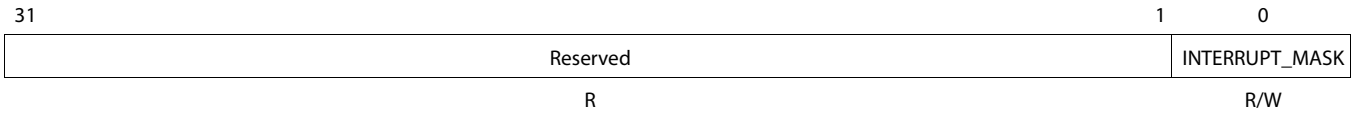
Table 7-33 Interrupt Mask Set Register Field Definitions

Bit	Field	Description
31-1	Reserved	Reserved
0	INTERRUPT MASK	When set to 1, this bit enables the Interrupt_Masked Status bit. Writing a 1 to this bit sets it and writing a 0 has no effect.
End of Table 7-33		

7.2.20 Interrupt Mask Clear

The interrupt Mask Clear register is used to clear the Interrupt_Masked status bit of the Interrupt Masked Status register.

Figure 7-32 Interrupt Mask Clear Register



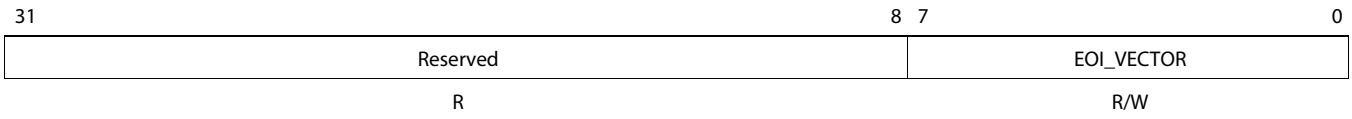
Legend: R = Read only; R/W = Read/Write

Table 7-34 Interrupt Mask Clear Register Field Definitions

Bit	Field	Description
31-1	Reserved	Reserved
0	INTERRUPT MASK	Writing a 1 to this bit clears the Interrupt Mask and disables the Interrupt_Masked bit. Writing a 0 has no effect
End of Table 7-34		

7.2.21 End of Interrupt (EOI) Register

Figure 7-33 End of Interrupt (EOI) Register



Legend: R = Read only; R/W = Read/Write

Table 7-35 End of Interrupt (EOI) Register Field Definitions

Bit	Field	Description
31-8	Reserved	Reserved
7-0	EOI_VECTOR	Writing a value updates the eoi_vector output and generates a single cycle eoi_write pulse that goes to external interrupt distribution logic. The specific eoi_vector value to write depends on the chip setup of this distribution logic.
End of Table 7-35		

7.3 STM Registers

The following sections describe the registers required to configure the STM to work correctly with the Tracer modules. Table 7-36 lists the register offsets for the STM.

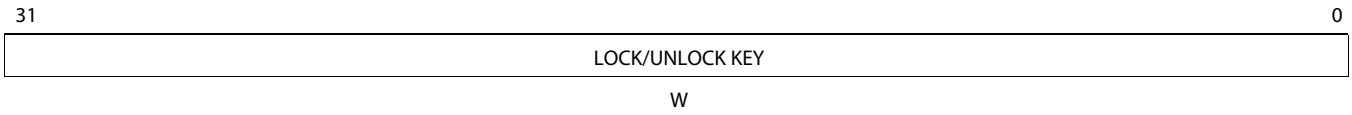
Table 7-36 STM Register Offsets

Offset	Name
0xFB0	Lock Access Register
0xFB4	Lock Status Register
0x024	Software Masters Control Register 0
0x028	Software Masters Control Register 1
0x02C	Software Masters Control Register 2
0x030	Software Masters Control Register 3
0x034	Software Masters Control Register 4
0x038	Hardware Master Control Register
0x03C	PTI Configuration Register
0x040	PTI Count Down Register
0x044	ATB Configuration Register
End of Table 7-36	

7.3.1 Lock Access Register

The Lock Access Register is used to allow write access to other STM registers by application software.

Figure 7-34 Lock Access Register



Legend: W = Write only

Table 7-37 Lock Access Register Field Definitions

Bit	Field	Description
31-0	LOCK/UNLOCK KEY	A write to this field of value 0xC5ACCE55 will grant access to other registers within the STM module. Writing any other value to this field will prevent writes to any other register within STM. Upon POR this register is reset to 0.
End of Table 7-37		

7.3.2 Lock Status Register

The Lock Status Register provides the locking mechanism used in the Lock Access Register.

Figure 7-35 Lock Status Register

Legend: R = Read only

Table 7-38 Lock Status Register Field Definitions

Bit	Field	Description
31-3	Reserved	Reserved
2	8-BIT LOCK	0 - Indicates we are using 32 bits for the lock 1 - Indicates we are using 8 bits for the lock
1	LOCK STATUS	0 - Unlocked 1 - Locked
0	LOCK IMPLEMENTED	0 - Indicates that no locking mechanism exists 1 - Indicates a locking mechanism exists
End of Table 7-38		

7.3.3 Software Masters Control Register 0 – STM_SWMCTRL0

The Software Masters Control Register 0 implements ownership status and cross triggering from external lines EMU0 and EMU1.

Figure 7-36 Software Master Control Register 0

31 30	29	28	27 17	16
OWNERSHIP	DEBUGGER OVERRIDE	CURRENT OWNER	Reserved	DEBUGGERTRACEEN
R/W	R/W	R/W	R	R/W

15	14	13 0
TRIGGER1ENABLE	TRIGGER0ENABLE	Reserved
R/W	R/W	R

Legend: R = Read only; R/W = Read/Write

Table 7-39 Software Master Control Register 0 Field Definitions

Bit	Field	Description
3-0	OWNERSHIP	A read returns the current ownership status. A write performs the following: 00 - change state to AVAILABLE 01 - change state to CLAIMED 10 - change state to ENABLED
29	DEBUGGER OVERRIDE	A read from this register will return a 1. A write performs the following: 0 - CLAIM request is granted only if unit is available 1 - CLAIM request is granted regardless of ownership status of unit
28	CURRENT OWNER	0 - Debugger owns resource 1 - Application owns resource
27-17	Reserved	Reserved
16	DEBUGGERTRACEEN	0 - Software master access qualified by MReqDebug doesn't generate STP message 1 - Software master access qualified by MReqDebug generates STP message
15	TRIGGER1ENABLE	0 - STM is not sensitive to EMU1 trigger 1 - Enables stop capturing of software messages from EMU1 detection [HIGH TO LOW]
14	TRIGGER0ENABLE	0 - STM is not sensitive to EMU0 trigger 1 - Enables start capturing of software messages from EMU0 detection [HIGH TO LOW]
13-0	Reserved	Reserved
End of Table 7-39		

7.3.4 Software Masters Control Register 1 – STM_SWMCTRL1

The Software Masters Control Register 1 sets SWMasterID values.

Figure 7-37 Software Master Control Register 1

31	30 24	23	22 16
Reserved	SWMasterID3	Reserved	SWMasterID2
R-0	R/W-0	R-0	R/W-0

15	14 8	7	6 0
Reserved	SWMasterID1	Reserved	SWMasterID0
R-0	R/W-0	R-0	R/W-0

Legend: R = Read only; R/W = Read/Write

Table 7-40 Software Master Control Register 1 Field Definitions

Bit	Field	Description
31	Reserved	Reserved
30-24	SWMasterID3	Software master trace enabled when MReqMstID[7] = 0 and MReqMstID[6:0] matches SWMasterID3
23	Reserved	Reserved
22-16	SWMasterID2	Software master trace enabled when MReqMstID[7] = 0 and MReqMstID[6:0] matches SWMasterID2
15	Reserved	Reserved
14-8	SWMasterID1	Software master trace enabled when MReqMstID[7] = 0 and MReqMstID[6:0] matches SWMasterID1
7	Reserved	Reserved
6-0	SWMasterID0	Software master trace enabled when MReqMstID[7] = 0 and MReqMstID[6:0] matches SWMasterID0
End of Table 7-40		

7.3.5 Software Masters Control Register 2 – STM_SWMCTRL2

The Software Masters Control Register 2 sets MRegMstID[1:0] mask values.

Figure 7-38 Software Master Control Register 2

31 26	25 24	23 18	17 16
Reserved	CoreIDMask3	Reserved	CoreIDMask2
R-0	R/W-0	R-0	R/W-0

15 10	9 8	7 2	1 0
Reserved	CoreIDMask1	Reserved	CoreIDMask0
R-0	R/W-0	R-0	R/W-0

Legend: R = Read only; R/W = Read/Write

Table 7-41 Software Master Control Register 2 Field Definitions (Part 1 of 2)

Bit	Field	Description
31-26	Reserved	Reserved
25-24	CoreIDMask3	MReqMstID[1:0] mask. Shall be set when the user wants to trace all SWMasterID3 processes.
23-18	Reserved	Reserved
17-16	CoreIDMask2	MReqMstID[1:0] mask. Shall be set when the user wants to trace all SWMasterID2 processes.
15-10	Reserved	Reserved
9-8	CoreIDMask1	MReqMstID[1:0] mask. Shall be set when the user wants to trace all SWMasterID1 processes.

Table 7-41 Software Master Control Register 2 Field Definitions (Part 2 of 2)

Bit	Field	Description
7-2	Reserved	Reserved
1-0	CoreIDMask0	MReqMstID[1:0] mask. Shall be set when the user wants to trace all SWMasterID0 processes.
End of Table 7-41		

7.3.6 Software Masters Control Register 3 – STM_SWMCTRL3

The Software Masters Control Register 3 sets SWDomainID values.

Figure 7-39 Software Master Control Register 3

31–27	26–24	23–19	18–16
Reserved	SWDomainID3	Reserved	SWDomainID2
R-0	R/W-0	R-0	R/W-0

15–11	10–8	7–3	2–0
Reserved	SWDomainID1	Reserved	SWDomainID0
R-0	R/W-0	R-0	R/W-0

Legend: R = Read only; R/W = Read/Write

Table 7-42 Software Master Control Register 3 Field Definitions

Bit	Field	Description
31-27	Reserved	Reserved
26-24	SWDomainID3	Software master trace enabled when MReqMstID[6:0] matches SWMasterID3 and MReqDomain[2:0] matches SWDomainID3.
23-19	Reserved	Reserved
18-16	SWDomainID3	Software master trace enabled when MReqMstID[6:0] matches SWMasterID3 and MReqDomain[2:0] matches SWDomainID3.
15-11	Reserved	Reserved
10-8	SWDomainID3	Software master trace enabled when MReqMstID[6:0] matches SWMasterID3 and MReqDomain[2:0] matches SWDomainID3.
7-3	Reserved	Reserved
2-0	SWDomainID3	Software master trace enabled when MReqMstID[6:0] matches SWMasterID3 and MReqDomain[2:0] matches SWDomainID3.
End of Table 7-42		

7.3.7 Software Masters Control Register 4 – STM_SWMCTRL4

The Software Masters Control Register 4 sets MRegDomain[2:0] mask values.

Figure 7-40 Software Master Control Register 4

31 27	26 24	23 19	18 16
Reserved	DomainMask3	Reserved	DomainMask2
R-0	R/W-0	R-0	R/W-0

15 11	10 8	7 3	2 0
Reserved	DomainMask1	Reserved	DomainMask0
R-0	R/W-0	R-0	R/W-0

Legend: R = Read only; R/W = Read/Write

Table 7-43 Software Master Control Register 4 Field Definitions

Bit	Field	Description
31-27	Reserved	Reserved
26-24	DomainMask3	MReqDomain[2:0] mask. Shall be set when the user wants to trace all SWMasterID3 domains
23-19	Reserved	Reserved
18-16	DomainMask2	MReqDomain[2:0] mask. Shall be set when the user wants to trace all SWMasterID2 domains
15-11	Reserved	Reserved
10-8	DomainMask1	MReqDomain[2:0] mask. Shall be set when the user wants to trace all SWMasterID1 domains
7-3	Reserved	Reserved
2-0	DomainMask0	MReqDomain[2:0] mask. Shall be set when the user wants to trace all SWMasterID0 domains
End of Table 7-43		

7.3.8 Hardware Masters Control Register

The Hardware Master Control Register is used to enable trace capturing for hardware masters based on mapping of the MReqMstID register.

Figure 7-41 Hardware Master Control Register

31	30 26	25 23	22 18	17 16
Reserved	HWMasterID3	Reserved	HWMasterID2	Reserved
R-0	R/W-0	R-0	R/W-0	R-0

15	14 10	9 7	6 2	1 0
Reserved	HWMasterID1	Reserved	HWMasterID0	Reserved
R-0	R/W-0	R-0	R/W-0	R-0

Legend: R = Read only; R/W = Read/Write

Table 7-44 Hardware Master Control Register Field Definitions (Part 1 of 2)

Bit	Field	Description
31	Reserved	Reserved
30-26	HWMasterID3	Hardware master trace enabled when MRqMstID[7] = 1 and MReqMstID[6:2] matches HWMasterID3
25-23	Reserved	Reserved
22-18	HWMasterID2	Hardware master trace enabled when MRqMstID[7] = 1 and MReqMstID[6:2] matches HWMasterID2
17-15	Reserved	Reserved

Table 7-44 Hardware Master Control Register Field Definitions (Part 2 of 2)

Bit	Field	Description
14-10	HWMasterID1	Hardware master trace enabled when MRqMstID[7] = 1 and MReqMstID[6:2] matches HWMasterID1
9-7	Reserved	Reserved
6-2	HWMasterID0	Hardware master trace enabled when MRqMstID[7] = 1 and MReqMstID[6:2] matches HWMasterID0
1-0	Reserved	Reserved
End of Table 7-44		

7.3.9 PTI Configuration Register

The PTI Configuration Register is used to configure the Parallel Trace Interface.

Figure 7-42 PTI Configuration Register

Legend: R = Read only; R/W = Read/Write

Table 7-45 PTI Configuration Register Field Definitions (Part 1 of 2)

Bit	Field	Description
31-9	Reserved	Reserved
8	PTIEnable	PTI enable/disable. 0 = PTI is disabled. "Programming takes effect when FIFO is empty "PTI exports NULL to insure TRC_CLK low "A software action is required to get PTI back active 1 = PTI is enabled. "Behavior depends on PTIAutoIdle setting "ATB interface shall be disabled
7	PTIClkGating	PTI clock mode 0 = Continuous mode enabled. NULL messages are exported on each TRC_CLK edge whenever STM FIFO is empty 1 = TRC_CLK gating enabled. Trace export stops with TRC_CLK low whenever STM FIFO is empty and a programmable number of NULL messages have been emitted.
6	PTIAutoIdle	PTI auto idle enable/disable. 0 = PTI Auto Idle is disabled. "PTI is kept active when FIFO is empty "PTI can only be disabled by software 1 = PTI Auto Idle is enabled. "PTI is active as soon as trace data is available from FIFO "PTI is disabled when FIFO is empty and PTI Count Down has reached zero and TRC_CLK is low. "PTI exports NULL until PTI count down has reached zero and TRC_CLK is low "PTI goes back active as soon trace data is available from FIFO
5-4	PTISize	PTI size. 0h = 1-bit PTI size 1h = 2-bit PTI size 2h = 4-bit PTI size 3h = Reserved (same behavior as 4-bit size)

Table 7-45 PTI Configuration Register Field Definitions (Part 2 of 2)

Bit	Field	Description
3	PTIClkEdge	PTI clock edge mode. 0 = Dual edge operating mode 1 = Single edge operating mode
2-0	Reserved	Reserved
End of Table 7-45		

7.3.10 PTI Count Down Register

The PTI Count Down Register is used to configure the number of NULL messages.

Figure 7-43 PTI Count Down Register

31	8	7	0
Reserved		NullCount	
R-0		R/W-4h	

Legend: R = Read only; R/W = Read/Write

Table 7-46 PTI Count Down Register Field Definitions

Bit	Field	Description
31-8	Reserved	Reserved
7-0	NullCount	Number of NULL messages to be emitted after last DATA. Once count down value reaches zero, the next instrumentation access will trig a MASTER message regardless of previous access master and channel. NullCount [2:0] field is static and set to "100". Programming that field has no effect. NullCount has a minimum programming value of 4 and can be programmed by step of 8.
End of Table 7-46		

7.3.11 ATB Configuration Register

The ATB Configuration Register is used to enable the ATB interface within the STM to allow for trace messages to be sent to the TETB.

Figure 7-44 ATB Configuration Register

31	17	16	15	0
Reserved		ATB_ENABLE	Reserved	
R		R/W	R	

Legend: R = Read only; R/W = Read/Write

Table 7-47 ATB Configuration Register Field Definitions

Bit	Field	Description
31-17	Reserved	Reserved
16	ATB ENABLE	0 - ATB interface is disabled 1 - ATB interface is enabled
15-0	Reserved	Reserved
End of Table 7-47		

7.4 TETB Registers

The following sections describe the registers required to configure the TETB to work correctly with the Tracer modules. Table 7-48 lists the register offsets for the TETB.

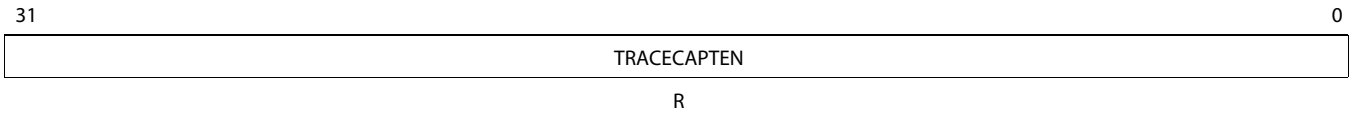
Table 7-48 TETB Register Offsets

Offset	Name
0x020	Control Register (CTL)
0xC20	TI Specific Control Register (TCTL)
0x010	RAM Read Data Register (RRD)
0x014	RAM Read Pointer Register (RRP)
End of Table 7-48	

7.4.1 Control Register (CTL)

The Control Register is used to enable trace capturing for the TETB.

Figure 7-45 Control Register



Legend: R = Read only

Table 7-49 Control Register Field Definitions

Bit	Field	Description
31-0	TRACECAPTEN	0 - Disable Trace Capture 1 - Enable Trace Capture
End of Table 7-49		

7.4.2 TI Specific Control Register (TCTL), 0xC20

The TCTL allows configuration of the buffering mode and TI_MODE.

Figure 7-46 TI Specific Control Register (TCTL)

Legend: R = Read only

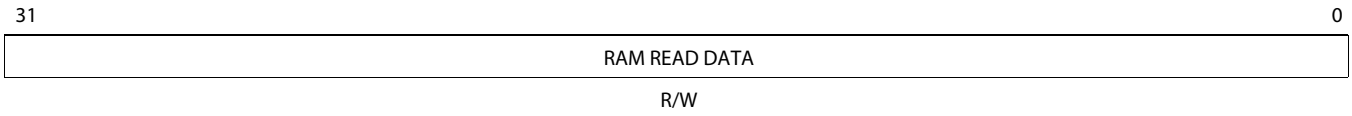
Table 7-50 TI Specific Control Register (TCTL) Field Definitions

Bit	Field	Description
31-2	Reserved	Reserved
1	STOP_BUF	0 - Circular Buffer Mode – When the TETB buffer is full, the buffer will wrap around and start overwriting data starting from the beginning of the buffer. 1 - Stop Buffer Mode – When the TETB buffer is full, no more data will be recorded.
0	TI_MODE	0 - The RAM Read Pointer Register is not updated unless trace capture is enabled 1 - The RAM Read Pointer Register will be updated
End of Table 7-50		

7.4.3 RAM Read Data Register (RRD)

The RAM Read Data Register is used to obtain the contents of the TETB. It contains the 32-bit word obtained from reading the TETB RAM at the location pointed to by the RAM Read Pointer Register.

Figure 7-47 RAM Read Data (RRD) Register



Legend: R/W = Read/Write

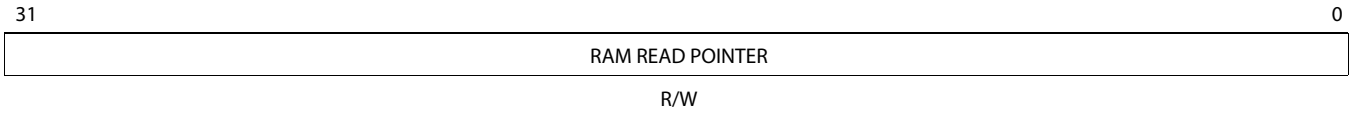
Table 7-51 RAM Read Data (RRD) Register Field Definitions

Bit	Field	Description
31-0	RAM READ DATA	Read contents from TETB RAM
End of Table 7-51		

7.4.4 RAM Read Pointer Register (RRP)

The RAM Read Pointer Register contains a 32-bit read pointer for where to read entries from the TETB RAM over the VBUSP interface.

Figure 7-48 RAM Read Pointer (RRP) Register



Legend: R/W = Read/Write

Table 7-52 RAM Read Pointer (RRP) Register Field Definitions

Bit	Field	Description
31-0	RAM READ POINTER	Sets read pointer used to read entries from TETB RAM
End of Table 7-52		

Chapter 8

Programming Guidelines

8.1 "Application Support" on page 8-2
8.2 "Programming Overview" on page 8-2

8.1 Application Support

TI includes a set of system level debug facilities in the Debug Subsystem of devices known as Chip Tools (CTools). For easy integration of DSP Core Trace and System Trace into applications, a set of libraries commonly referred to as CToolsLib are provided. CToolsLib is a collection of embedded target APIs/libraries focused on enabling easy access to the CTools. CToolsLib encompasses the following libraries to assist in trace integration:

- DSP Trace Library
- AET Library
- STM Library
- CP Tracer Library
- ETB/TBR Library
- PTM/ETM Library
- CTools Use-Case Library

For more information about these libraries, downloadable files, and other useful links, please visit the CToolsLib Wiki site:

http://processors.wiki.ti.com/index.php/CToolsLib

8.2 Programming Overview

The following sections demonstrate the capabilities of the cTools libraries for various debug tools. Care should be taken to make sure the power and sleep controller for the Debug Subsystem is enabled before proceeding with further debug configuration.

8.2.1 DSP Core Trace Specific Configuration

DSPTraceLib enables you to do the following:

- Claim ownership of the Trace Export Block
- Configure Pin Manager
- Claim and enable trace streams
- Configure type of trace capture
- Start/stop trace recording

8.2.2 AET Specific Configuration

CToolsLib for AET provides APIs for configuring the AET logic for various trace jobs. The AET library enables you to do the following:

- Initialize AET
- Claim AET
- Set up AET to trigger on a certain set of conditions
- Start/Stop triggering with AET

8.2.3 STM Specific Configuration

CToolsLib provides various APIs for STM configuration as follows:

- Sending various types of software messages.
- Configuring the Tracers and generating various types of hardware messages

In addition, the STM can be configured for the following features:

1. Setting STM for optimized or unoptimized operation. In optimized operation only a pointer to the message string is transported, unlike in unoptimized operation where the entire null terminated string is transported.
2. Configuring STM channel resolution to either 1024 or 4096 bytes for specified channels

8.2.4 Debug Pin Manager Configuration

To export trace data off chip to an external trace receiver, the Debug Pin Manager of the DRM must be programmed to specify which pins will be used for export and for what purpose. With TI's CToolsLib, this configuration is performed automatically. Otherwise, to set up a special configuration, the Debug Pin Manager must be programmed as demonstrated in the following text:

1. Read Peripheral ID register to make sure DRM can be accessed successfully
2. Read DRM_CAPABILITIES register to make sure DRM is properly configured
3. Claim ownership of the DPM by writing 0x01 to the ownership field of the DPM Claim Register
4. Enable the DPM by writing 0x10 to the ownership field of the DPM Claim Register
5. Configure corresponding DPM Control Register for each debug pin in use
6. The user can soft reset the entire DRM by writing to the Soft Reset field of the DRM Config Register.

CAUTION—Do not program the DPM manually if you are using any of the CTools Libraries.

Index

A

accelerators
  RAC (Receive Accelerator Coprocessor), 5-2
  TAC (Transmit Accelerator Coprocessor), 5-2
AET (Advanced Event Triggering), 1-14, 2-6, 8-2
AIF2 (Antenna Interface), ø-xii
architecture, ø-xi, 1-1 to 1-3, 4-6, 5-1, 5-3

B

buffer, 1-2, 1-10 to 1-11, 1-15 to 1-16, 4-2, 4-6 to 4-7, 5-1, 5-3, 7-27
bus(es), 1-2, 1-6, 2-5, 5-16 to 5-17

C

capture mode, 1-10, 1-15, 2-4, 4-2, 6-2, 7-7 to 7-8, 7-12, 7-27, 8-2
CBA (Common Bus Architecture), 1-2, 5-2 to 5-4, 5-9, 5-13, 5-16, 7-7
clock, 1-7 to 1-8, 2-4 to 2-5, 4-2, 4-4, 5-2, 5-16, 7-9
configuration, 1-6 to 1-7, 2-4 to 2-5, 4-4 to 4-6, 5-14, 6-2, 7-3, 7-15, 7-20, 7-26 to 7-27, 8-2 to 8-3
configuration register, 4-5, 6-2, 7-3, 7-20, 7-26
consumption, 4-2
CPU, 1-2, 2-6, 5-2, 5-9, 5-16, 7-7, 7-17 to 7-18

D

DDR (Double Data Rate), 1-11
  DDR3, ø-xii
debug, ø-xi, 1-1 to 1-3, 1-5 to 1-10, 1-12, 1-14, 1-17, 2-4 to 2-5, 4-2, 4-4 to 4-5, 5-2 to 5-3, 5-9, 5-15, 5-17, 6-2 to 6-3, 7-2, 7-4, 7-14, 8-2 to 8-3
debug mode, ø-xi, 1-1 to 1-3, 1-5 to 1-10, 1-12, 1-14, 1-17, 2-4 to 2-5, 4-2, 4-4 to 4-5, 5-2 to 5-3, 5-9, 5-15, 5-17, 6-2 to 6-3, 7-2, 7-4, 7-14, 8-2 to 8-3
detection, 1-12, 1-15 to 1-16, 2-5, 7-21
DMA (direct memory access), 5-9, 5-16, 7-7
DSP, ø-xi, 1-2 to 1-3, 1-5 to 1-6, 1-8, 1-10 to 1-11, 1-14, 2-1, 2-4 to 2-6, 3-2, 8-2

E

EDMA (Enhanced DMA Controller), 1-11
EMIF (External Memory Interface), 5-2
EMU (emulation), 1-5 to 1-6, 1-14, 1-17, 2-5 to 2-6, 5-16 to 5-17, 7-6 to 7-7
emulation, 1-14, 1-17, 2-6
EOI (End of Interrupt), 7-19
ETB (Embedded Trace Buffer), 1-6, 1-14, 4-2, 4-7, 6-2, 8-2

F

fault, 1-10
FFTC (Fast Fourier Transform Coprocessor), ø-xii

H

host controller, 1-17
HyperLink (formerly MCM), ø-xii

I

inputs, 2-6, 5-2 to 5-4, 5-17
integration, 1-2, 1-17, 5-1, 8-2
interface, ø-xii, 1-2, 1-6, 1-10 to 1-11, 1-15, 1-17, 2-5, 4-2, 4-4, 5-2 to 5-3, 5-9 to 5-10, 5-12 to 5-13, 5-17, 6-2 to 6-3, 7-14 to 7-15, 7-26, 7-28
interrupt, 1-14, 2-5 to 2-6, 5-17, 7-5, 7-9, 7-17 to 7-19

J

JTAG (IEEE
  Joint Action Test Group), 1-17

M

memory
  DMA, 5-9, 5-16, 7-7
  EMIF, 5-2
  general, ø-xii, 1-2, 1-11, 2-5, 4-2, 4-6, 5-11, 6-3, 7-12 to 7-13
  L2 (Level-Two Unified Memory), 5-11
  MSMC, ø-xii, 5-12
message, 1-11, 1-15, 4-2, 4-6 to 4-7, 5-1 to 5-3, 5-10, 5-13 to 5-17, 6-1, 6-3, 7-5, 7-9, 7-12, 7-14 to 7-15, 7-21, 8-3
metrics, 4-2
mode
  capture, 1-10, 1-15, 2-4, 4-2, 6-2, 7-7 to 7-8, 7-12, 7-27, 8-2
  debug, ø-xi, 1-1 to 1-3, 1-5 to 1-10, 1-12, 1-14, 1-17, 2-4 to 2-5, 4-2, 4-4 to 4-5, 5-2 to 5-3, 5-9, 5-15, 5-17, 6-2 to 6-3, 7-2, 7-4, 7-14, 8-2 to 8-3
module, 1-2, 1-7, 1-10, 1-12, 1-14, 1-17, 2-5, 4-2, 4-4, 5-1 to 5-4, 5-9, 5-11 to 5-12, 5-15 to 5-16, 6-3, 7-2 to 7-3, 7-5 to 7-6, 7-8, 7-15, 7-18, 7-20
MSMC (Multicore Shared Memory Controller), ø-xii, 5-12
Multicore Navigator (formerly CPPI), ø-xii

O

on-chip, 1-9 to 1-10, 4-2, 5-10, 6-2
output(s), 1-11, 7-4, 7-18 to 7-19

override, 7-4, 7-21

P

PASS (Packet Accelerator Subsystem), 5-17
PCIe (Peripheral Component Interconnect Express), ø-xii, 1-11
performance, 1-10, 4-2, 5-2 to 5-3
PLL (Phase-Locked Loop), ø-xii, 2-5
POR, 7-20
port, 1-2, 1-5 to 1-8, 1-14, 2-4, 4-2, 4-4, 5-2 to 5-3, 5-9, 7-18

Q

QM_SS (Queue Manager Subsystem), 5-2

R

RAC (Receive Accelerator Coprocessor), 5-2
RAM, 1-10, 5-12, 6-2 to 6-3, 7-27 to 7-28
RapidIO, ø-xii, 1-11
reset, 4-5, 5-17, 6-3, 7-3, 7-9, 7-13 to 7-14, 7-20, 8-3

S

SCR (Switched Central Resource), 5-2 to 5-3
security
  general, 4-5
semaphore, 5-2, 7-15
signal, 1-17, 2-6, 5-9, 7-17
sleep mode, 8-2
SRAM (Static RAM), 5-2
SRIO (Serial RapidIO) subsystem, ø-xii
status register, 5-17, 6-3, 7-5, 7-8, 7-13, 7-17 to 7-21

T

TAC (Transmit Accelerator Coprocessor), 5-2
timers, 5-17
timestamp, 1-11, 4-6 to 4-7, 6-2
Trace, ø-xi, 1-2 to 1-3, 1-5 to 1-6, 1-8 to 1-11, 1-14 to 1-15, 1-17, 2-1, 2-4 to 3-2, 4-1 to 4-7, 5-1 to 5-3, 5-9, 5-13, 5-17, 6-2, 7-6 to 7-11, 7-13, 7-24, 7-26 to 7-27, 8-2 to 8-3
trace buffers
  ETB (Embedded Trace Buffer), 1-6, 1-14, 4-2, 4-7, 6-2, 8-2

V

version, 7-2, 7-5 to 7-6

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265