Embedded Cross Trigger

Revision: r0p0

Technical Reference Manual

ARM®

ARM DDI 0291A

Embedded Cross Trigger
Technical Reference Manual

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Change
22 July 2003	A	First issue for r0p0

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

Contents

Embedded Cross Trigger Technical Reference Manual

List of Tables

Embedded Cross Trigger Technical Reference Manual

List of Figures
Embedded Cross Trigger Technical Reference Manual

Preface

This preface introduces the *Embedded Cross Trigger r0p0 Technical Reference Manual*. It contains the following sections:

- *About this book* on page x
- *Feedback* on page xiv.

About this book

This book provides user information for the *Embedded Cross Trigger* (ECT).

Intended audience

This book is written for hardware engineers implementing *System-on-Chip* (SoC) designs. It provides information to enable designers to integrate the ECT within their system.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the ARM Debug System, and ECT and its interfaces.

Chapter 2 *Functional Description*

Read this chapter for a description of the major functional blocks of the ECT.

Chapter 3 *Programmer's Model*

Read this chapter for a description of the registers and programming details of the ECT Cross Trigger Interface blocks.

Chapter 4 *Programmer's Model for Test*

Read this chapter for a description of the test registers of the ECT Cross Trigger Interface blocks.

Appendix A *Signal Descriptions*

Read this appendix for descriptions of signals that interface with the ECT.

Appendix B *Timing Requirements*

Read this chapter for descriptions of the timing requirements for the ECT interfaces.

Product revision status

The r*n*p*n*v*n* identifier indicates the revision status of the product described in this document, where:

r*n*	Identifies the major revision of the product.

p*n*	Identifies the minor revision or modification status of the product.
v*n*	Identifies a version that does not affect the external functionality of the product.

Typographical conventions

The following typographical conventions are used in this book:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`monospace` (with `mono` underlined)	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.

Other conventions

This document uses other conventions. They are described in the following sections:

- *Signals*
- *Bytes, halfwords, and words* on page xii
- *Bits, bytes, k, and M* on page xii
- *Register fields* on page xii.

Signals

When a signal is described as being asserted, the level depends on whether the signal is active HIGH or active LOW. Asserted means HIGH for active high signals and LOW for active low signals:

Prefix n	Active LOW signals are prefixed by a lowercase n except in the case of AHB or APB reset signals. These are named **HRESETn** and **PRESETn** respectively.

Prefix H	AHB signals are prefixed by an upper case H.
Prefix P	APB signals are prefixed by an upper case P.

Bytes, halfwords, and words

Byte	Eight bits.
Halfword	Two bytes (16 bits).
Word	Four bytes (32 bits).
Doubleword	Eight bytes (64 bits).
Quadword	16 contiguous bytes (128 bits).

Bits, bytes, k, and M

Suffix b	Indicates bits.
Suffix B	Indicates bytes.
Suffix k	When used to indicate an amount of memory means 1024. When used to indicate a frequency means 1000.
Suffix M	When used to indicate an amount of memory means 1024^2 = 1 048 576. When used to indicate a frequency means 1 000 000.

Register fields

You must not access reserved or unused address locations because this can result in unpredictable behavior of the device.

You must write as 0 any reserved or unused bits of registers, and ignore them on read unless otherwise stated in the relevant text.

All registers bits are reset to logic 0 by a system reset unless otherwise stated in the relevant text.

Unless otherwise stated in the relevant text, all registers support read and write accesses. A write access updates the contents of the register and a read access returns the contents of the register.

All registers defined in this document can only be accessed using word reads and word writes, unless otherwise stated in the relevant text.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information on developing code for the ARM family of processors.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the ARM Frequently Asked Questions list.

ARM publications

This book contains information that is specific to the ECT. Refer to the following documents for other relevant information:

- *AMBA Specification (Rev 2.0)* (ARM IHI 0011)
- *ETM9 Technical Reference Manual* (ARM DDI 0157)
- *ETM Specification* (ARM IHI 0014)
- *Embedded Trace Buffer Technical Reference Manual* (ARM DDI 0242)
- *ARM946E-S Technical Reference Manual* (ARM DDI 0155).

Feedback

ARM Limited welcomes feedback on both the ECT, and its documentation.

Feedback on the ECT

If you have any problems with the ECT, contact your supplier. To help us provide a rapid and useful response, please give:

- details of the release you are using
- details of the platform you are running on, such as the hardware platform, operating system type, and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tool, including the version number and date.

Feedback on this book

If you have any comments on this book, send an email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces the *Embedded Cross Trigger* (ECT) revision r0p0. It contains the following sections:

- *About the ARM debug system* on page 1-2
- *About the ECT* on page 1-6.

1.1 About the ARM debug system

The ARM debug system is described in the following sections:

- *Debug system overview*
- *Development tools* on page 1-5.

1.1.1 Debug system overview

The ARM debug system incorporates a number of *System-on-Chip* (SoC) peripherals that are specified by ARM. An example system can consist of the following blocks:

- *Embedded Trace Macrocell* (ETM)
- *Embedded Trace Buffer* (ETB)
- Embedded Cross Trigger (ECT)
- ARM microprocessor.

The ETMs consist of two peripherals:

- the embedded trace macrocell
- the ETB.

These macrocells are developed, tested, and licensed by ARM. Refer to the relevant documentation for a list of features.

The ECT provides an interface to the debug system. This enables ARM9/ETM9 subsystems to interact, that is *cross trigger*, with each other. The debug system enables debug support for multiple cores, together with cross triggering between the cores and their respective ETM9s.

Figure 1-1 on page 1-3 shows a dual-core debug system diagram. The diagram shows two main parts:

- the embedded trace functions
- the cross trigger functions.

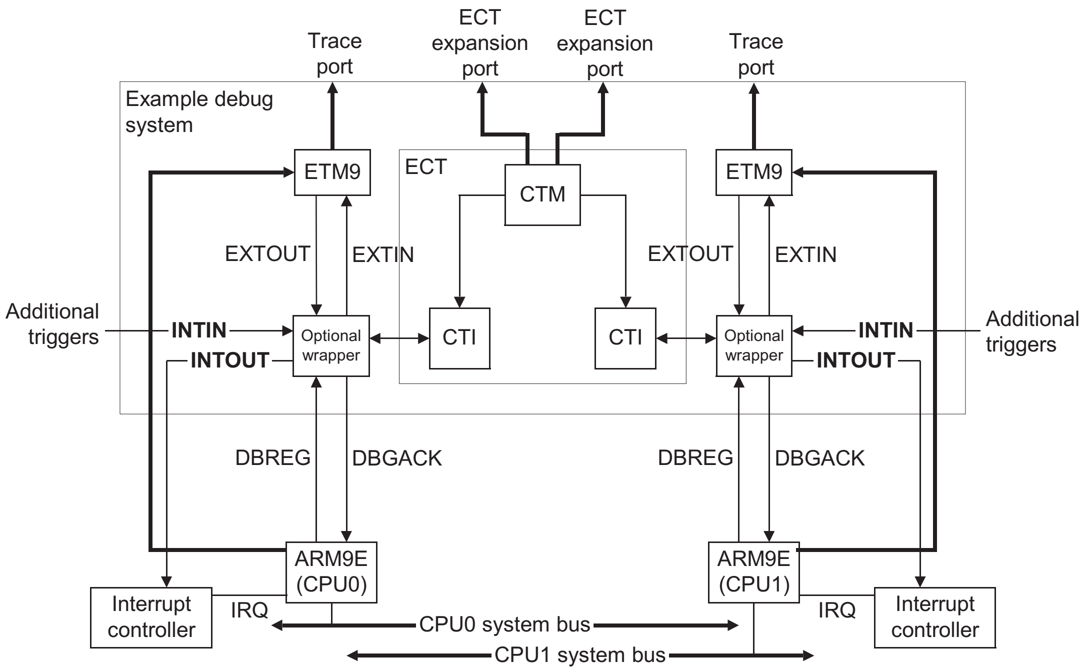

Figure 1-1 Example dual-core cross-trigger system

As well as the ETM9, and ECT (which contains the *Cross Trigger Interface* (CTI) and *Cross Trigger Matrix* (CTM)) there are the following components:

ARM9E CPU core.

Optional wrapper

A simple wrapper circuit to connect the CPU core debug signals to the CTI. For an example, refer to the file `Arm9CtiWrapper.v` in the `docs` directory of the ECT deliverable.

Interrupt Controller (IC)

The main function of the ECT (CTI and CTM) is to pass debug events from one processor to another. For example, the ECT can communicate debug state information from one core to another, so that program execution on both processors can be stopped at the same time if required.

Figure 1-1 on page 1-3 only contains two processors. However, you can expand the cross-trigger network to support more than two. You can also connect third-party CPU/DSP designs to the ECT providing that you have implemented a suitable wrapper. In this example, the system can have the trace function expanded to include an ETB (refer to *Embedded Trace Buffer Technical Reference Manual*).

You can connect additional sources of debug triggering signals to the ECT using the optional wrapper at the **INTIN** port as shown in Figure 1-1 on page 1-3.

You can connect the debug signals on the processor system directly to the CTI without the optional wrapper if:

- they use the same clock as the CTI
- the signals are glitch free and no special waveform shaping is required (for example edge detection).

Embedded trace

In Figure 1-1 on page 1-3 the embedded trace function is supported by the use of two ETM9s. The ETMs can output directly to two independent trace ports, or you can route them to the ETBs, where you can access the trace data using the *Test Access Port* (TAP).

CTM

This block combines the channel requests generated by the CTIs and broadcasts them to all other CTIs as channel triggers. This enables subsystems (ETM9 and ARM9) to cross trigger with each other.

You can also use the CTM to extend a system by connecting it to another CTM subsystem.

CTI

This block is the interface to the CTM. Its implementation is specific to the core and other trigger sources. It provides a common programmer's model for use by the debug tools.

ETM9

The ETM9 provides instruction and data trace information for ARM microprocessors. You can use this information with the debugger to enable easy cycle-accurate non-intrusive debugging on ARM core based embedded systems. The ETM9 provides extensive resources for event recognition to generate trigger events including:

- up to sixteen address comparators

- eight data comparators
- four 16-bit counters
- a 3-state sequencer.

For further details, see the *ETM9 Technical Reference Manual* and the *ETM Specification*.

1.1.2 Development tools

You can use a JTAG interface to control the debug system as shown in Figure 1-1 on page 1-3, and you can access trace information through a trace port. You can access the CTI modules as a memory-mapped peripheral using its AHB interface. The debug system is supported by development tools such as *RealView*® *Debugger* (RVD).

1.2 About the ECT

The ECT provides a mechanism for multiple microprocessor based subsystems to send and receive debug triggers to and from each other. It is an *Advanced Microcontroller Bus Architecture* (AMBA) compliant *System-on-Chip* (SoC) peripheral that provides a common programmer's model for use by the debug tools.

The ECT consists of two modules:

The *Cross Trigger Interface* (CTI)

This block controls the Trigger Interface (TI). The CTI combines and maps the trigger requests, and broadcasts them to all other interfaces on the ECT as channel events. When the CTI receives a channel event it maps this onto a trigger output. This enables subsystems to cross trigger with each other. The receiving and transmitting of triggers is performed through the TI.

The *Cross Trigger Matrix* (CTM)

This block controls the distribution of channel events. It provides *Channel Interfaces* (CIs) for connection to either a CTI or CTM. This enables multiple ECTs to be connected to each other.

1.2.1 Features of the ECT

The ECT has the following features:

- cross triggering of multiple ARM9E cores
- support for glueless, multiple, stacked ECTs
- real-time access memory-mapped registers
- test registers for integration and production testing
- two channel interfaces, each with four event channels
- two trigger interfaces, each with eight input and output triggers
- standard handshaking interface that can be adapted to the user system with the use of a simple wrapper
- AHB mapped, you can access all the registers (4KB) through this interface
- ECT application trigger
- handshaking and synchronization bypass ports for lower latency triggering of synchronous systems

- lock register for protection against accidental register writes
- CTI Protection Enable Register to prevent User mode code access.

1.2.2 Trigger input events

The system supports the following examples of trigger input events:

DBGACK	Enables debug cross triggering between cores.
ETMEXTOUT	Enables triggering of one or more cores from an ETM trigger.
INTIN	Enables cross triggering of one or more cores from an external interrupt.
APPTRIG	An internal event generated inside the CTI. This is accessible using the AHB bus/tap controller and enables a debugger to force a channel event using the configuration registers.

The ECT has two types of event interfaces:

- two *Trigger Interfaces* (TIs) connecting to the microprocessor subsystems
- two *Channel Interfaces* (CIs) connecting to other ECT CIs enabling ECTs to be connected to each other.

The microprocessor subsystem can be an ARM9/ETM9 combination or a user-specific subsystem, for example DSP. The ECT combines the trigger requests generated by the subsystems and broadcasts them to other subsystems as channel events. This enables subsystems to cross trigger with each other.

This Technical Reference Manual provides enough information for developers to design a subsystem that can interface to the ECT.

1.2.3 Events

The ECT provides a standard interface enabling the following classes of cross trigger event to be supported using a simple wrapper, four external events and one internal event:

Conditioned	Keeps the output active for one clock cycle after an acknowledgement is received.
Level/pulse	Used when trigger destination is level sensitive, the signal is active for only one clock cycle.
Sticky	Keeps the output active until it is cleared by a corresponding clear register.

NoAck Used when the output follows the input trigger and does not require an acknowledgement.

Software programmable

Used by the application to trigger events under software control, this enables a debugger to force an event using the configuration registers.

If you want to add a further class you have to add a wrapper to the TI to shape your specific signals to the required format. For more information see *Cross trigger events* on page 2-20.

Chapter 2
Functional Description

This chapter describes the function of the blocks in the ECT. It contains the following sections:

2.1 Functional overview

This chapter describes how you can implement subsystems to enable a given core to interface to the ECT using a *Trigger Interface* (TI). The implementation of the ECT provides a common programmer's model for use by the debug tools.

This is described in the following sections:

- *About the ECT system*
- *Interfaces handshake protocol* on page 2-11
- *Synchronization* on page 2-11
- *Clock information* on page 2-13
- *Mapping* on page 2-14.

2.1.1 About the ECT system

The ECT function is supported by the use of the *Trigger Interface* (TI) and the *Channel Interface* (CI). The TI is the interface to the core subsystems.

Figure 2-1 on page 2-3 shows an example ECT system.

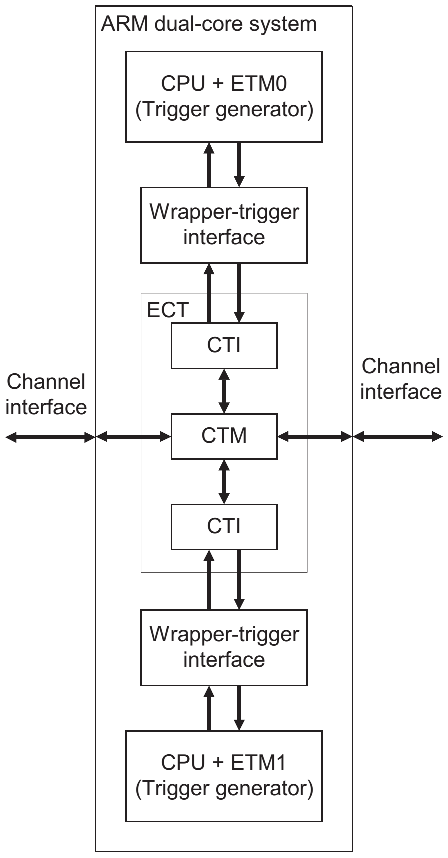

Figure 2-1 Example ECT system

Figure 2-1 shows the TI and the optional user-defined wrapper to the TI. The TI uses handshake signalling because it is robust with asynchronous systems, and it automatically adapts to changes in clock frequency of components connected to the ECT. The CI is generic so that you do not have to modify the ECT design if a different trigger generator subsystem is used. In most cases you require a simple wrapper to connect a trigger generator subsystem to the ECT.

Figure 2-2 on page 2-4 shows a simple ECT system.

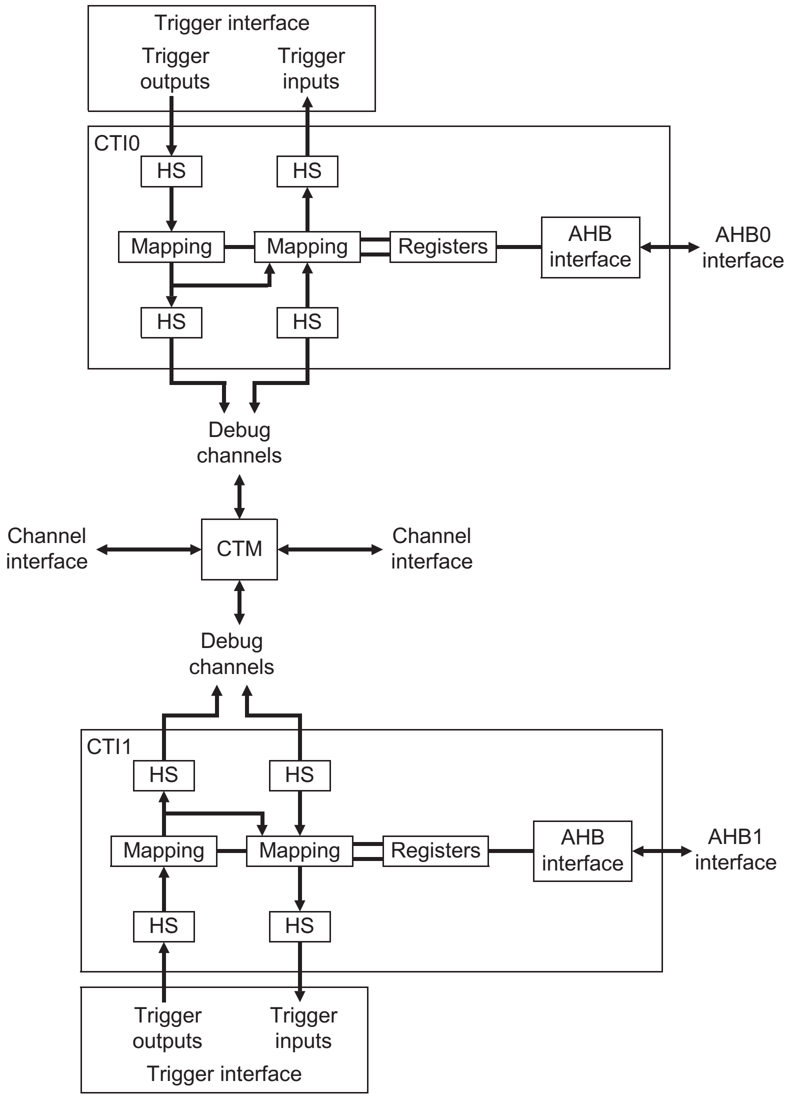

Figure 2-2 Simple ECT system

Figure 2-2 shows the following:

ECT	Comprising two CTIs and one CTM.
CTI	Interface between the cross trigger network (the CTM) and the processor system (for example, CPU and ETM).

CTM The cross trigger network providing interconnections between CTIs. The CTM contains four ports, each port containing Debug Channel In and Debug Channel Out connections. When a Debug Channel In receives a signal, it is output to the Debug Channel Out of the three other ports.

HS Handshaking circuitry.

AHB interface

Programs the configuration and status registers of the CTI. All transactions on the AHB slave programming bus of the ECTs are 32 bits wide. This eliminates endian issues when programming the ECTs.

The register block controls:

- the mapping from trigger inputs to debug channels
- the mapping from debug channels to trigger outputs.

You can control the registers using the AHB interface.

You can easily expand the system by using the two expansion ports provided by each ECT to stack the ECTs to each other without requiring external components (see *Stacking ECTs* on page 2-9).

Figure 2-3 on page 2-6 shows the following TI buses:

- **ECTTRIGIN[7:0]** for trigger in
- **ECTTRIGINACK[7:0]** for trigger in acknowledge
- **ECTTRIGOUT[7:0]** for trigger out
- **ECTTRIGOUTACK[7:0]** for trigger out acknowledge
- inputs **ECTTISBYPASSIN[7:0]** and **ECTTISBYPASSACK [7:0]** enable the buses to bypass the synchronization logic on the respective bits of **ECTTRIGIN** and **ECTTRIGOUTACK**
- inputs **ECTTIHSBYPASS[7:0]** enable the handshaking for the respective bits of **ECTTRIGOUT[7:0]** to be bypassed, when the outputs do not require an acknowledgement.

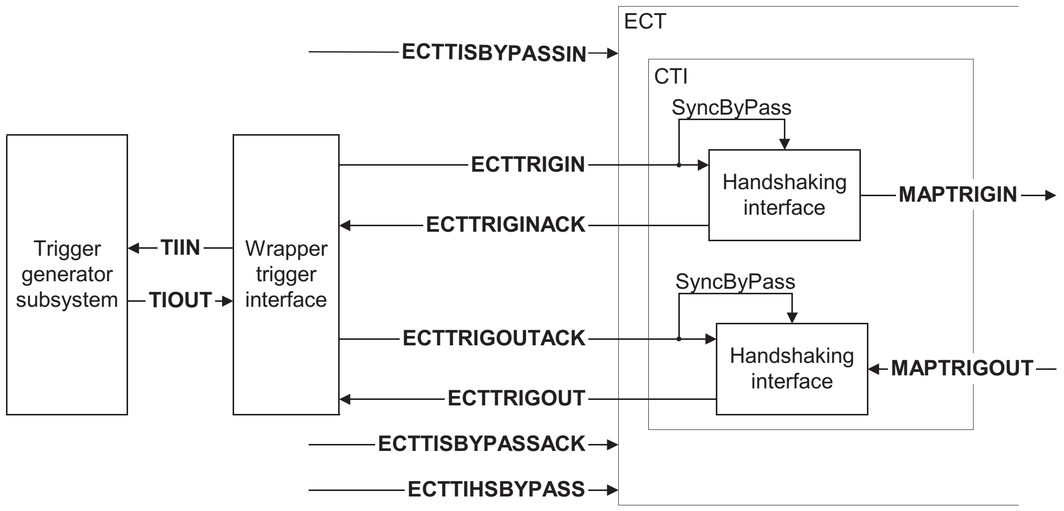

Figure 2-3 Trigger interface

Figure 2-4 shows an example of the typical signals that make up the trigger interface input (**TIIN**) and trigger interface output (**TIOUT**) buses to the wrapper.

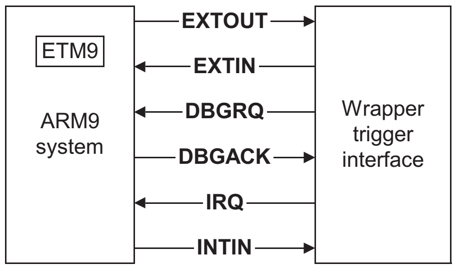

Figure 2-4 Example of ARM core subsystem connections

Depending on the subsystem:

1. An event on **EXTOUT**, **DBGACK**, and **INTIN** (all of which can make up the **TIIN** bus) can be propagated into the ECT through the wrapper block onto one of the **ECTTRIGIN** signals.
2. The **ECTTRIGIN** signal is propagated though the ECT onto the other TIs and CIs.
3. This event can be forwarded to the next core as:
   - a debug request, **DBGRQ**

- an ETM event, **EXTIN**
- an interrupt, **IRQ**.

Each type of cross trigger event has a specific behavior that requires the wrapper to function in a certain way, as described in *Cross trigger events* on page 2-20.

Figure 2-5 and Figure 2-6 on page 2-8 show that a single ECT within a system has two trigger interfaces and two channel interfaces.

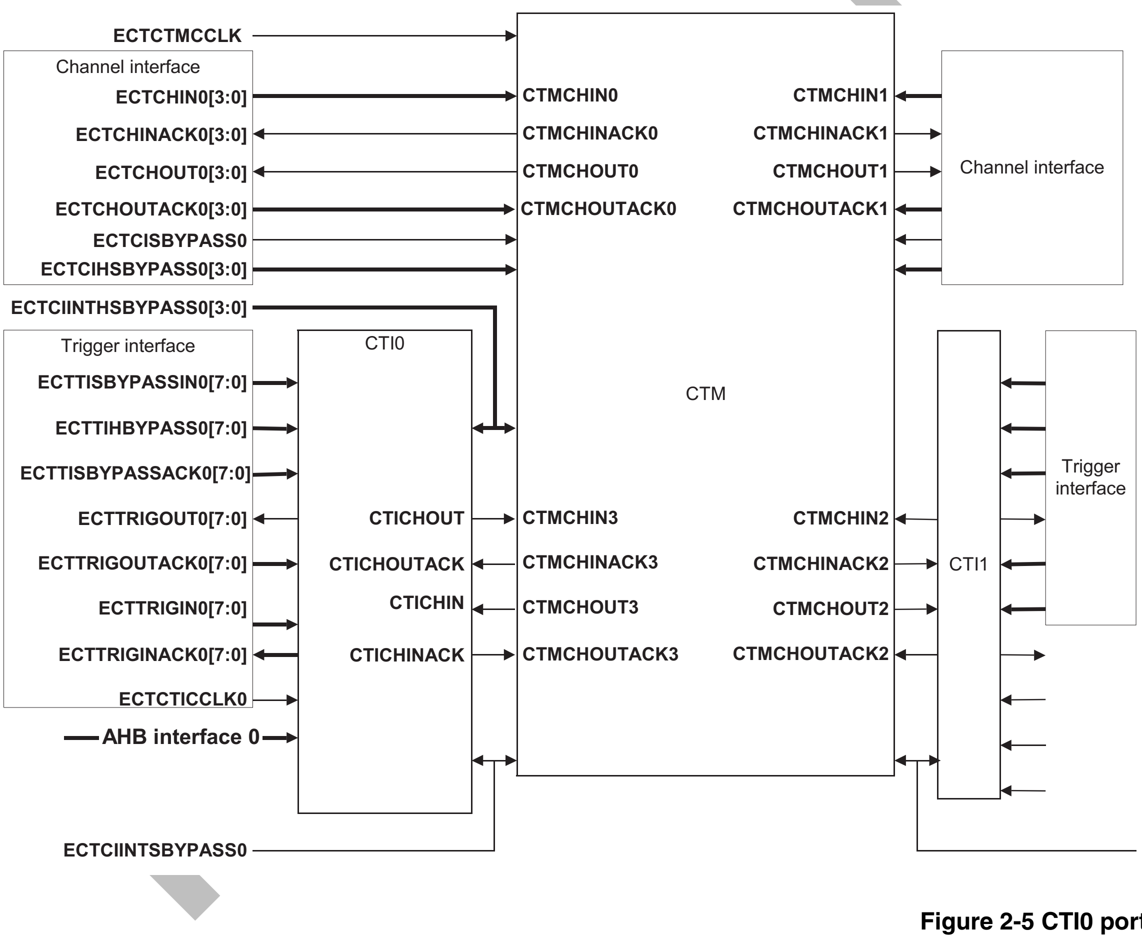

Figure 2-5 CTI0 ports

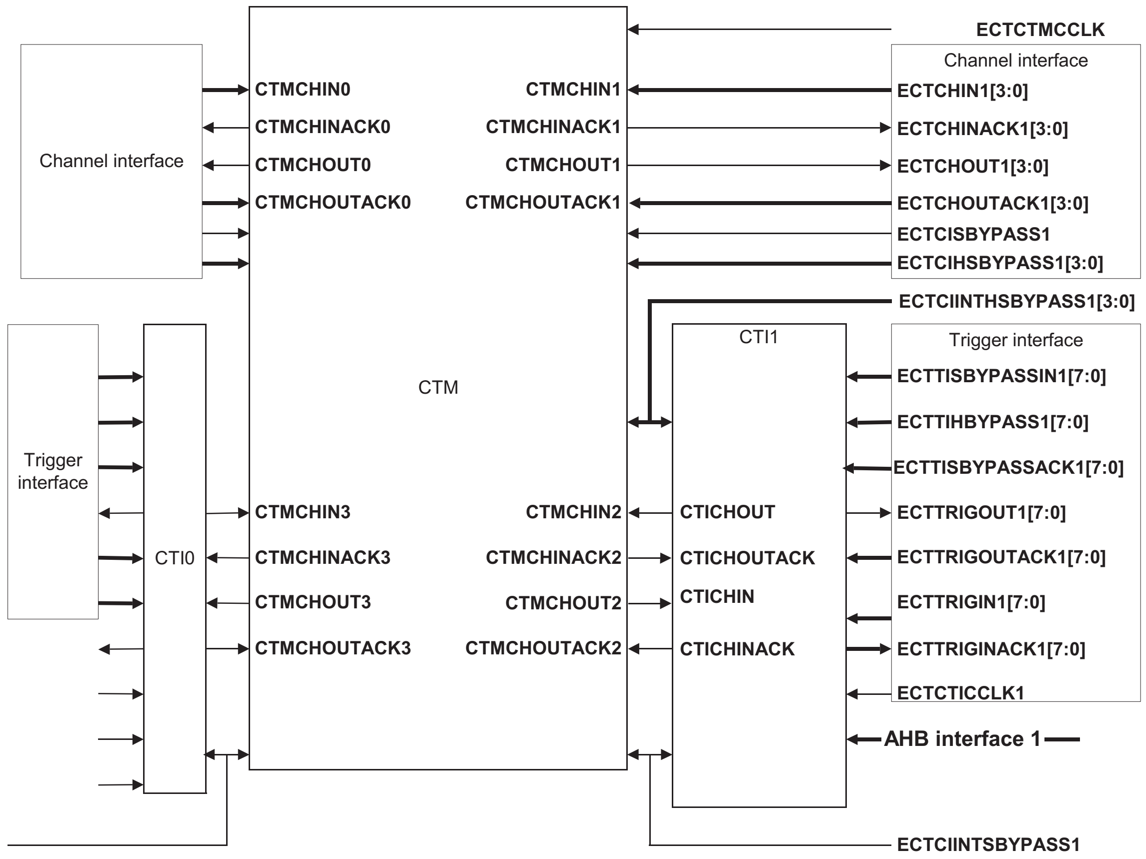

Figure 2-6 CTI1 ports

The CTI takes the trigger inputs from the TI and enables them onto channels within the ECT. The ECT supports up to four channels equating to **ECTCHIN[3:0]** and **ECTCHINOUT[3:0]**.

The CI comprises the following buses:

- **ECTCHOUT[3:0]** for channel out
- **ECTCHOUTACK[3:0]** for channel out acknowledge
- **ECTCHIN[3:0]** for channel in
- **ECTCHINACK[3:0]** for channel in acknowledge

- inputs **ECTCISBYPASS[3:0]** to enable the buses to bypass the handshaking and synchronization logic of **ECTCHIN[3:0]**
- inputs **ECTCIINTHSBYPASS[3:0]** and **ECTCIINTSBYPASS[3:0]** to bypass the same logic for the internal Channel Interfaces between the CTI and CTM.

Stacking ECTs

Figure 2-7 shows how the CI enables the ECT to be connected to other ECT systems. Because there are two CIs you can stack the ECTs. This provides a very flexible solution for expanding debug access for multi-core products. You can stack the ECTs within a single system or across systems.

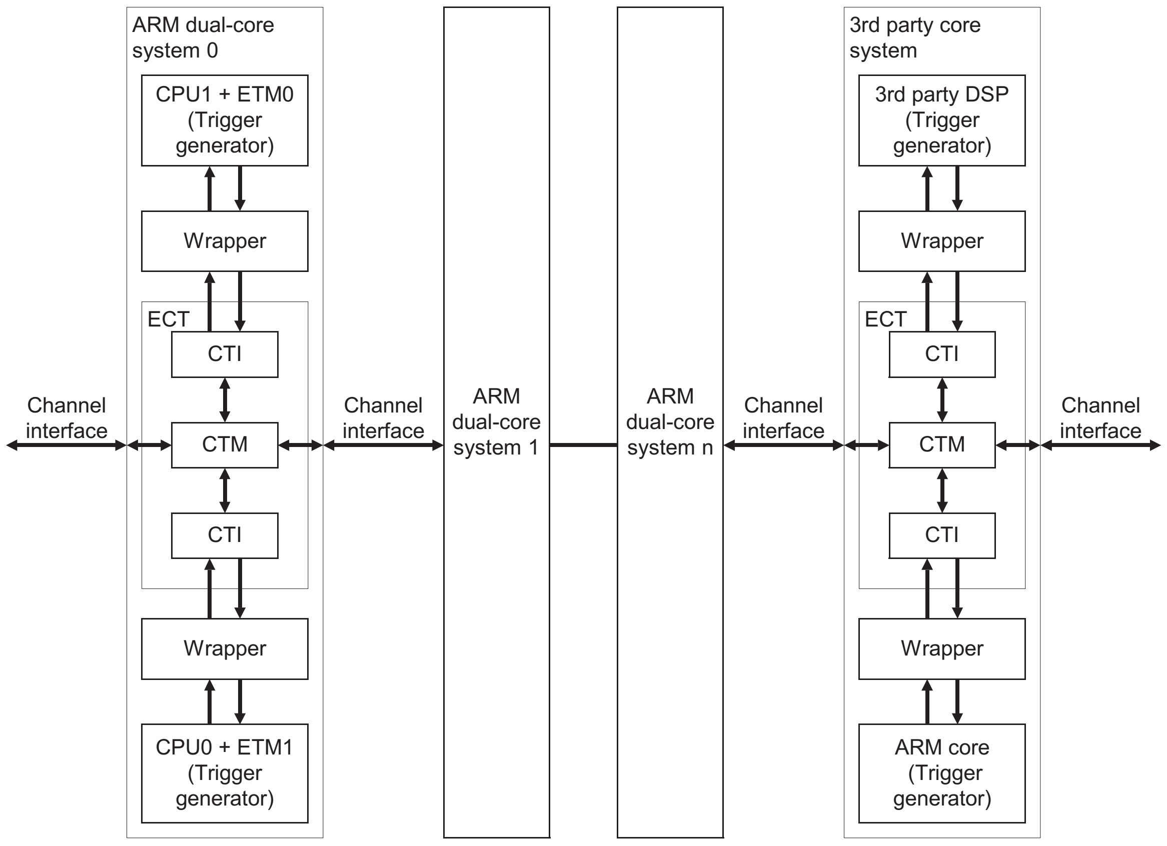

Figure 2-7 Stacked ECT systems

When you stack a number of ECTs together you must ensure that no timing violating paths are created. It is expected that timing violations will only occur when a large number of ECTs are used and all the synchronization bypasses are enabled. If violations are created you can pipeline the signal path by enabling the synchronization logic on some of the ECTs, so avoiding the timing violation.

Figure 2-8 shows an example of how the generic debug CI is connected, as shown in Figure 2-7 on page 2-9. This figure shows an example of the connections required between the two CTMs.

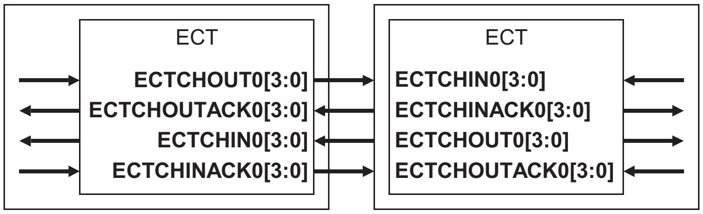

Figure 2-8 Channel interface connections

You can connect the CI port to another CI port, as long as the connections are swapped, for example **ECTCHOUT** connected to **ECTCHIN**, and **CTMCHOUTACK** connected to **CTMCHINACK**.

When one TI sends a trigger:

1. It is propagated to the CTM, through the CTI.
2. The CTM propagates it to the other TIs and ECTs using its CIs.
3. These other ECTs send the trigger to all the TIs connected to it.

For example, if the CPU0 in Figure 2-7 on page 2-9 generates a trigger, and if its corresponding mapping logic in the CTI is enabled, this trigger event is mapped into a channel event and enters the CTM in the ARM dual-core system 0. The event is then sent to the CTI for CPU1, as well as all other ECTs in the system through the channel interface. The combinatorial path between one TI and another on a separate subsystem is increased by the propagation delay across the other CTMs. The synchronization between the systems is managed by the handshaking internal to the CTM components that you can bypass if required.

ECTDGBEN turns off the CI and TI inputs and outputs in the CTI modules, effectively turning off the mapping logic. To save power you can use this signal to ensure the ECT interfaces are off in non-debug mode.

2.1.2 Interfaces handshake protocol

There is no edge detection in the ECT. An event signal is transmitted as a level. If you want to have edge detection or single pulse output you must implement the required shaping logic in the external wrapper.

To avoid any incompatibility, the following protocol has been defined:

- Only logic 1 is interpreted as an event.
- If the handshake is enabled, an output must stay active until an acknowledgement by hardware (or optionally acknowledged by software for the TI) is received, even if the acknowledge signal driver is deactivated.

If the handshaking is not bypassed the ECT can only handle one-shot events. If events are close to one another (multi-shot) and mapped to the same channel they are possibly merged into one event on an output trigger. For debug events such as breakpoint, trace start, and trace stop this does not cause a problem because input events mapped to the same triggers are required to do the same thing.

Events arising from different interfaces but mapped to the same channel might also be merged. This is acceptable because the mapping logic would have been programmed to allow this. Events can be merged because blocks handshake between asynchronous clock domains or channels events are mapped onto the same output trigger.

If the events broadcast on the ECT emanate from synchronous clock domains then you can bypass the handshaking logic. The output does not receive an acknowledgement (Noack trigger class). In such cases you can use the ECT to transmit multiple shot events. The output has to remain active for at least one clock cycle to ensure it is captured at the destination interface.

2.1.3 Synchronization

Systems where events broadcast on the ECT can emanate from asynchronous clock domains must observe the following rules:

1. *Register input events* on page 2-12. Any signal that is an input of the ECT must be glitch free to avoid any false event. The wrapper can perform glitch removal as necessary.
2. *Synchronize output events* on page 2-12. The outputs of the ECT must be synchronized to the local clock domain before use by the subsystem, if such synchronization is not already present in the subsystem. The wrapper performs synchronization when necessary.

Register input events

The concern here is that a glitch on an output of the subsystem is propagated in the ECT and can be interpreted as an event. Also, the synthesis tools do not provide the necessary mechanism to constrain signals so that they are glitch free.

As an example, in an ARM9E design, **DBGACK** is the output of a combinatorial circuit (three-input OR gate). There is no way to ensure that a change of state on the inputs does not result in a glitch on the **DBGACK** signal, which can be fetched by an asynchronous circuit.

Considering design approaches suitable for delivery as reusable IP, the events from the processor cores must be registered inside the wrapper before being sent to the ECT. A single register can be used in this case or you can enable synchronization logic in the ECT for that specific trigger input adding an extra clock cycle to the latency.

Synchronize output events

Figure 2-9 shows that a standard two-register synchronization is used.

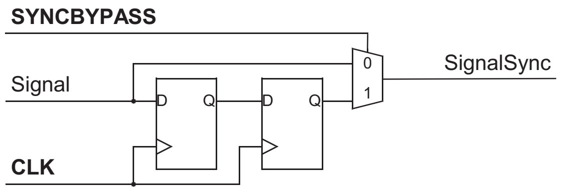

Figure 2-9 Standard synchronization

Note

It is important that you consult the design rules on the target process and the standard cell libraries to see if there are any restrictions or recommendation in relation to synchronizers. For example, the library rules might require the use of specially designed synchronization registers, or require that registers forming a synchronization chain must be placed in close proximity to each other.

Latency

The latency corresponds to the number of cycles necessary from the time one signal enters a ECT until it is propagated to another processor. The path in the CTI and CTM is combinatorial, so that the latency is only due to the handshaking circuits. This means that, in the worst case, the latency is the sum of:

- one clock cycle of the CTI sending the event to the matrix (removal of the glitch)
- the combinatorial delay caused by the propagation of a **ECTTRIGIN** event to a **ECTTRIGOUT** output
- two clock cycles of the CTI receiving the event from the matrix (synchronization to the clock of the receiving device).

In certain cases, it might be possible to reduce this latency:

- If the signal sent by the processor is the output of a flip-flop. In this case you do not have to register the signal because it is glitch-free.
- If the core being interfaced to already features synchronization logic internally. For example the processor has been specified to receive an asynchronous signal.
- If all the subsystems are synchronous. In this case the TI bypass signals, **ECTTISBYPASSACK** and **ECTTISBYPASSIN**, and the CI bypass signals **ECTCISBYPASS** and **ECTCIINTSBYPASS**, can be activated. However, this path must respect the layout and synthesis timing constraints.

2.1.4 Clock information

When a processor clock is stopped (waiting for an interrupt for example), the corresponding CTI can receive an event from the CTM. When the CTI clock is the same as the subsystem and the handshaking is not bypassed, the CTM keeps the signal active until an acknowledgement is received, which only occurs when the clock is started again. In this case, out-of-date events can happen on the core. This does not inhibit the channel being used by other processors.

However, if the CTI clock differs from the local processor clock (for example, gated differently), it is possible for the CTI to raise an event to the core using **ECTTRIGOUT**, while the processor clock is off. If it has to be avoided, the processor must disable its CTI before stopping the clock.

It is assumed that in most systems the **ECTCTICLK** is connected to the same clock as its local processor and the **ECTCTMCLK** is connected to the fastest processor clock in the system so reducing the trigger latency and the requirement for clock enables ports.

If a CTI is going to be disabled while the processor enters a clock stopped mode, ARM recommends that:

- the CTI turns off the event-to-channel mapping, so that unwanted events are not generated to the CTM
- the channel-to-event mapping circuit is turned off or disabled for a few clock cycles after the clock is on.

2.1.5 Mapping

The mapping blocks control the driving of the CTIs output ports.

2.2 Trigger interface

The *Trigger Interface* (TI) is described in the following sections:

- *About the TI*
- *ECTTRIGIN handshake* on page 2-17
- *Trigger interface wrapper input* on page 2-17
- *ECTTRIGOUT handshake* on page 2-18
- *Cross trigger events* on page 2-20
- *Example wrapper* on page 2-23.

2.2.1 About the TI

The TI is generic. This means that you do not have to modify the CTI design if you are using a different subsystem. In most cases you require a simple wrapper to connect a subsystem to the CTI. Figure 2-10 on page 2-16 shows an overview of the CTI, part of the ECT structure. The CTI interfaces to a subsystem with the CTM, which in turn communicates with the ECT interfaces. The CTI enables the subsystem to broadcast and respond to (enabled) events on the CTM.

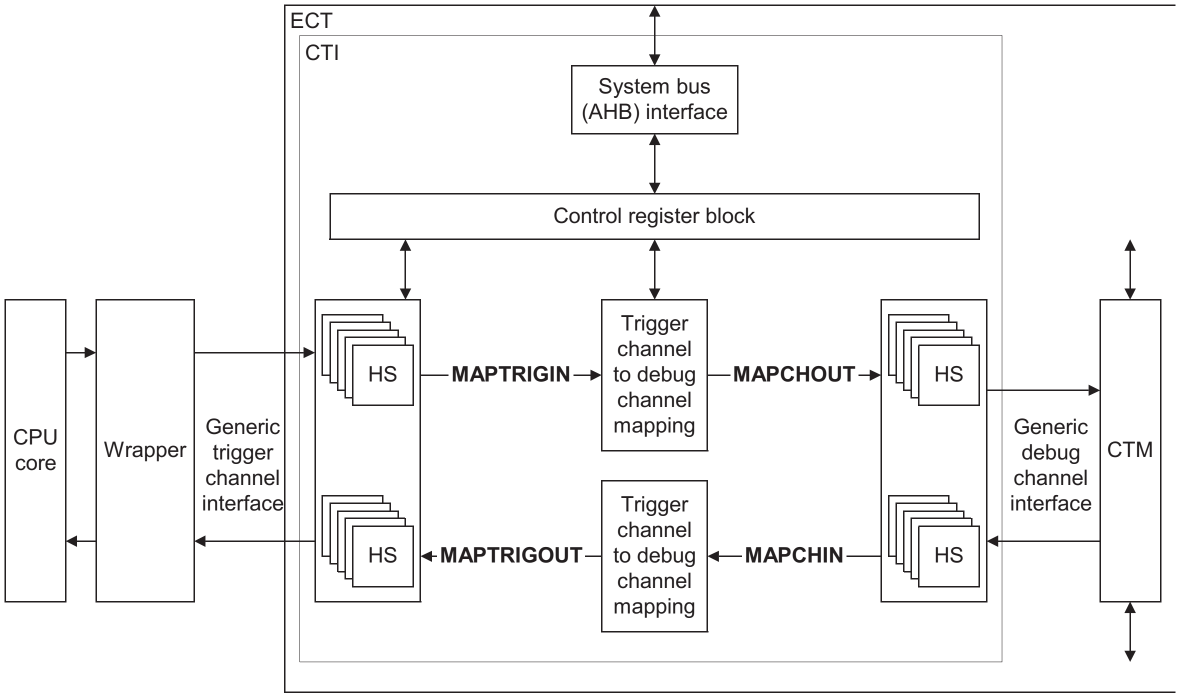
ECT
CTI
System bus (AHB) interface
Control register block
CPU core
Wrapper
Generic trigger channel interface
HS
MAPTRIGIN
Trigger channel to debug channel mapping
MAPCHOUT
HS
Generic debug channel interface
CTM
HS
MAPTRIGOUT
Trigger channel to debug channel mapping
MAPCHIN
HS

Figure 2-10 CTI structure overview

The CTI maps:

- trigger inputs onto channels
- channel inputs onto trigger outputs.

The CTI takes cross trigger event inputs from its associated subsystem, or configuration register (an application-derived trigger event), and generates outputs on the appropriate channels as defined by the configuration registers for the CTM.

The subsystem programs the configuration registers of the CTI it is connected to by using the AHB interface on the CTI.

ECTTRIGIN, as shown in Figure 2-3 on page 2-6, is the collective name for any event from the subsystem. If the **ECTTRIGIN** signals do not require synchronization the synchronization block can be bypassed using the **ECTTISBYPASSIN** inputs. Depending on the configuration registers, the resultant signal can be mapped to a channel.

ECTTRIGOUT is the collective name for any event from the TI. You can bypass the **ECTTRIGOUTACK** synchronization block using the **ECTTISBYPASSACK** inputs.

2.2.2 ECTTRIGIN handshake

Figure 2-11 shows the circuit required for the **ECTTRIGIN** handshaking. You can bypass the synchronization if all the subsystems are synchronous, by asserting the relevant bit of **ECTTISBYPASSIN[7:0]** HIGH.

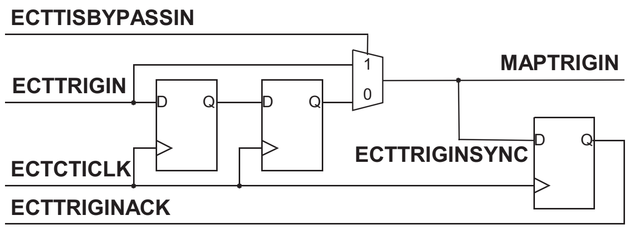

Figure 2-11 ECTTRIGIN handshaking

Figure 2-12 is an example of a timing diagram of the circuit shown in Figure 2-11, when **ECTTISBYPASSIN** is inactive so enabling the synchronization logic.

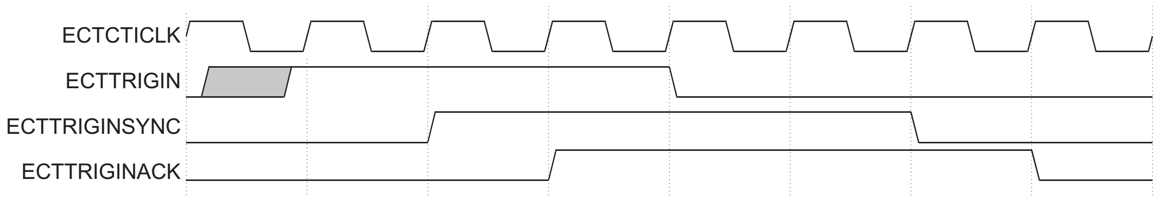

Figure 2-12 ECTTRIGIN handshaking timing

2.2.3 Trigger interface wrapper input

To interface the ECT into a system you might have to apply a wrapper to the TI. The implementation of the CTI only detects a level 1 as an event. Therefore the signal might have to pass through a wrapper to adapt its behavior (Inverter, convert for example). In most cases, where adaptation is not required and when the **ECTCTICLK** is synchronous or slower than the subsystem core clock, no wrapper is required. For example, **TIIN** is fed straight through to the **ECTTRIGIN**.

There might be a case where the trigger signals clock domain, **TIINCLK** and **ECTCTICLK**, are not synchronous. Figure 2-13 on page 2-18 shows that if the triggers are operating in a higher frequency clock domain (**TIINCLK**) the wrapper must have logic that captures and holds the trigger until it has been acknowledged by the ECT.

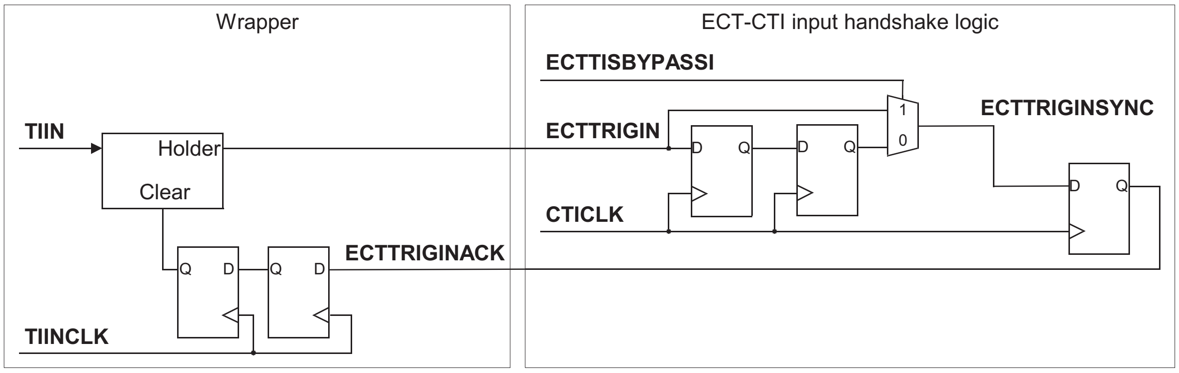

Figure 2-13 Input wrapper signal synchronization

If the input driver of **TIIN** requires an acknowledgement signal then the wrapper must ensure any necessary synchronization takes place.

Assuming the clock domains are synchronous, you can connect the **TIIN** inputs to the **ECTTRIGIN**. The example wrapper supplied with the ECT assumes this is the case. See *Cross trigger events* on page 2-20 and *Example wrapper* on page 2-23.

2.2.4 ECTTRIGOUT handshake

Figure 2-14 on page 2-19 shows the **ECTTRIGOUT** handshaking logic.

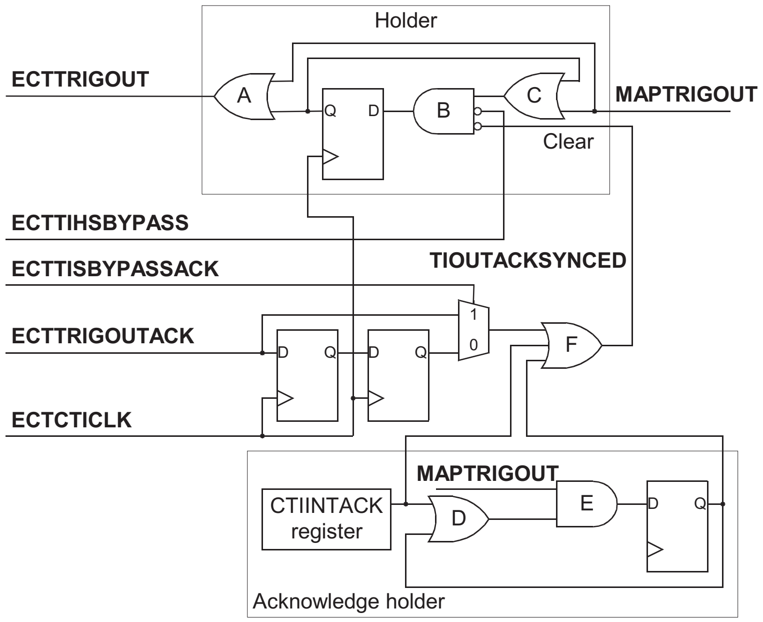

Figure 2-14 ECTTRIGOUT handshaking logic

When an event is detected on one bit of **MAPTRIGOUT**, it is propagated directly to the corresponding bit of **ECTTRIGOUT** through the OR gate A. If the **ECTTIHSBYPASS** signal is LOW, the signal **ECTTRIGOUT** is kept activated using the feedback loop though gates C and B, until the acknowledgement signal, **TIOUTACKSYNCED** (synchronized **ECTTRIGOUTACK**), is received into the not input of AND gate B. If you have to synchronize the **ECTTRIGOUT** signal for the subsystem you must implement the required logic in the wrapper.

Figure 2-15 on page 2-20 shows **ECTTRIGOUT** timing. If **MAPTRIGOUT** is HIGH while the acknowledgement signal has already been received, **ECTTRIGOUT** is also HIGH until **MAPTRIGOUT** is deactivated. If you are using a sticky class output, a write of HIGH to the **CTIINTACK** register is required to acknowledge the output trigger. The Acknowledge holder circuit is then used to hold the acknowledgement until the **MAPTRIGOUT** has been deactivated.

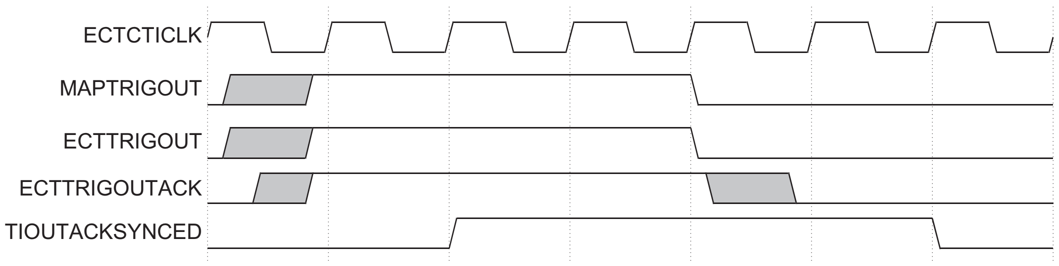

Figure 2-15 ECTTRIGOUT timing

In Figure 2-15 **ECTTIHSBYPASS** is LOW and **ECTTISBYPASSACK** is LOW (enabling acknowledgement synchronization logic). The acknowledgement signal **ECTTRIGOUTACK** is received at the same clock cycle as **ECTTRIGOUT** is propagated. This might be due to the subsystem running at a higher frequency or the **ECTTRIGOUT** being fed directly back into the **ECTTRIGOUTACK** port as in the case of level class trigger outputs as shown in Figure 2-16 on page 2-21.

If the **ECTTIHSBYPASS** signal is HIGH, handshake bypass is enabled and **ECTTRIGOUT** follows **MAPTRIGOUT** as gate B disables the holder and acknowledge logic so that only gate A is active.

2.2.5 Cross trigger events

Figure 2-16 on page 2-21 shows an example wrapper for all external classes of cross trigger output event.

Note

If timing constraints allow, you can remove the registers with (Timing) in Figure 2-16 on page 2-21. These registers are included to simplify the synthesis process for this example.

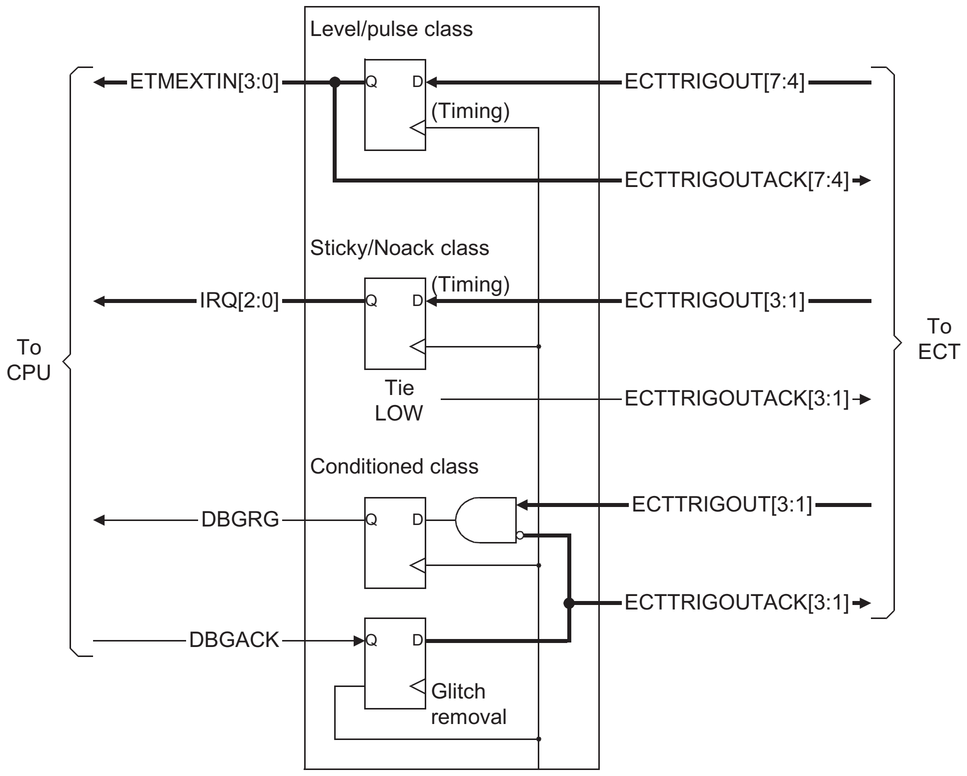

Figure 2-16 Example output wrapper configurations

The external classes of cross trigger output event are:

Sticky class A typical function for a sticky class signal is an interrupt. When the **ECTTRIGOUT** is used to drive an interrupt signal, the output is kept active until it is cleared by a write to the corresponding bit of the **CTIINTACK** register. The Acknowledge holder holds the acknowledgement until the **MAPTRIGOUT** is deactivated. The **ECTTRIGOUTACK** is tied LOW, and the corresponding **ECTTIHSBYPASS** is LOW.

Noack class The wrapper used is the same as that for the sticky class. If the output trigger does not require an acknowledgement and all clock domains are the same, **ECTTIHSBYPASS** is HIGH. This means that the **ECTTRIGOUT** signal is the same as the **MAPTRIGOUT** signal.

This class is typically used where the clock domains between the source and destination are the same and the source is guaranteed to be at least one clock cycle. If the source signal is active for one clock cycle, you must ensure that there is no clock skew between source and destination.

In this class you can design the wrapper so that multiple shot events are supported. The ECT can be set up so that all handshaking is bypassed. The destination wrapper receives the source signal and then acknowledges it through another channel. When the source wrapper receives the acknowledgement it can then send another event. The logic design for the wrappers can vary according to your requirements.

Level/pulse class

A typical function for a level class is an ETM **EXTIN** signal. The ETM **EXTIN** input is level-sensitive. The **ECTTRIGOUT** has to keep the signal active for at least one clock cycle. This is achieved in the wrapper by feeding the **ECTTRIGOUT** back in to the **ECTTRIGOUTACK**. The **ECTTISBYPASSACK** is HIGH, **ECTTIHSBYPASS** is LOW, and the corresponding bit of the **CTITRIGINCLEAR** register is LOW.

Note

To generate a single clock cycle pulse you must remove the timing D-type from the wrapper.

Conditioned class

A typical function for a level class is a **DBGRQ/DBGACK** to and from an ARM core. **DBGRQ** is activated until **DBGACK** is received. The processor can enter debug mode without the ECT requesting it, for example after a breakpoint.

Most of the ARM cores do not support receiving a debug request while in debug mode. For this reason, the wrapper is required to gate the **DBGRQ** whenever **DBGACK** is asserted. The **DBGACK** signal from the CPU is a combinatorial output. Therefore even if the CTI operates in the same clock domain as the CPU, this signal must be registered in the wrapper before being used. You can also use **DBGACK** as an input trigger as shown in Figure 2-16 on page 2-21. The trigger can then be propagated to the other cores in the system. **ECTTIHSBYPASS** is LOW and **ECTCTISBYPASS** is set according to the clock domains.

If the output trigger has to be a single clock length pulse then you have to add the necessary logic to ensure correct signal shaping.

Table 2-1 shows the use of the **ECTTIHSBYPASS** and **ECTCTISBYPASS**.

Table 2-1 Bypass modes

Bypass modes	ECTCTISBYPASS	ECTTIHSBYPASS
Handshaking and synchronization enabled. Used when clock domains are asynchronous.	0	0
Handshaking enabled and synchronization disabled. Used when clock domains are synchronous but clock skew is unknown. Level class triggers.	1	0
Handshaking disabled, synchronization enable/disable. Used when clock domains are synchronous and clock skew allows. Used when no acknowledge is required.	x	1

2.2.6 Example wrapper

The ECT is supplied with an example wrapper (in the `docs` directory) that supports an ARM9/ETM subsystem. Table 2-2 shows the configuration of the inputs from the subsystem to the wrapper.

Table 2-2 Example subsystem wrapper inputs

Subsystem wrapper inputs	Class	Type
TIIN[0]	Conditioned (glitch removal)	**DBGACK**
TIIN[1]	Level	**INTIN[0]**
TIIN[2]	Level	**INTIN[1]**
TIIN[3]	Level	**INTIN[2]**
TIIN[4]	Level	**ETMEXTOUT[0]**
TIIN[5]	Level	**ETMEXTOUT[1]**
TIIN[6]	Level	**ETMEXTOUT[2]**
TIIN[7]	Level	**ETMEXTOUT[3]**

The **INTIN** inputs are normally held until cleared by an interrupt service routine so in this case can be treated as a level class input.

Table 2-3 shows the configuration of the outputs from the example wrapper to the subsystem.

Table 2-3 Example subsystem wrapper outputs

Subsystem wrapper outputs	Class	Type
TIOUT[0]	Conditioned (**DBACK** conditioned)	**DBGRQ**
TIOUT[1]	Sticky	**IRQ[0]**
TIOUT[2]	Sticky	**IRQ[1]**
TIOUT[3]	Sticky	**IRQ[2]**
TIOUT[4]	Level	**ETMEXTIN[0]**
TIOUT[5]	Level	**ETMEXTIN[1]**
TIOUT[6]	Level	**ETMEXTIN[2]**
TIOUT[7]	Level	**ETMEXTIN[3]**

2.3 Channel interface

This section describes the *Channel Interface* (CI):

- *About the CI*
- *ECTCHIN handshake* on page 2-29
- *ECTCHOUT handshake* on page 2-30
- *Channel interface connections* on page 2-33.

2.3.1 About the CI

The CTM contains some sequential elements to complete the handshake signalling with the CTI.

The CI is controlled by the CTM which is a crossbar consisting of four channels and four CIs. For example **ECTCHOUT0[3:0]** represents four channels for CI0. Each channel provides a single bit output (for example **ECTCHOUT0[0]**) to each CI that is the logical OR of all the other CI input ports (for example **ECTCHINx[0]**).

You can use a channel to represent an occurrence or condition. A channel can have a dedicated meaning in a system, for example system error. Also, you can configure the use and meaning of a channel. The CI can be connected to another CI or a CTI. The ECT has four CIs, two internal and two external. The bypass controls for the internal CIs are brought to the top level of the ECT.

Figure 2-17 shows a block diagram for a single CI on the CTM with four channels.

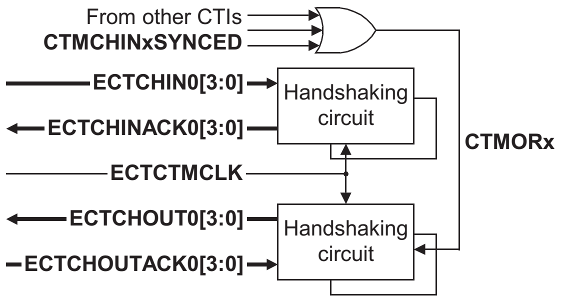

Figure 2-17 Single CI of CTM

Each **ECTCHIN** input is synchronized, if required, and acknowledged. This is to inform the sending ECT or CTI that the **ECTCHIN** has been received and therefore can be cleared.

Figure 2-18 on page 2-26 shows the circuit used for CTM for all four CIs (each CI has four channels).

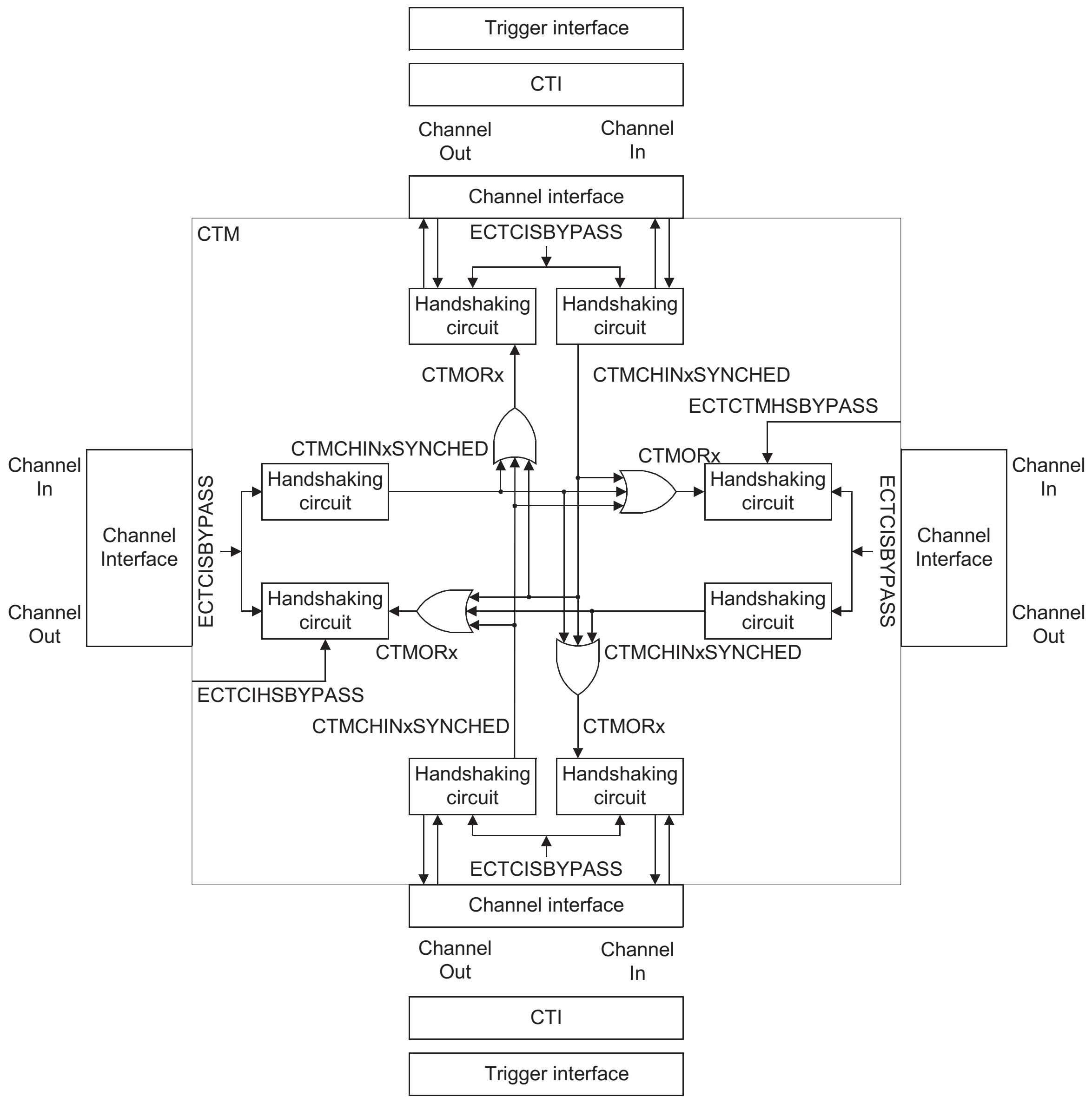

Figure 2-18 Handshaking signalling in the CTM

Note

The **ECTCHIN** is not routed back to its own **ECTCHOUT**. The CTI has logic to generate its own **ECTCHOUT** trigger.

The sequential elements for **ECTCHIN**, **ECTCHINACK**, **ECTCHOUT**, and **ECTCHOUTACK** signals are bused, one element for each channel, and an independent handshake is implemented for each bit.

You can synchronize the channel to capture the **ECTCHIN** if required:

1. The **ECTCHIN** is broadcast to each of the other CIs.
2. The blocks connected to these are then responsible for sending back an acknowledgement when required.
3. This causes the CTM to release the appropriate **ECTCHOUT** signal for that CI.

Figure 2-19 shows the timing diagram for this.

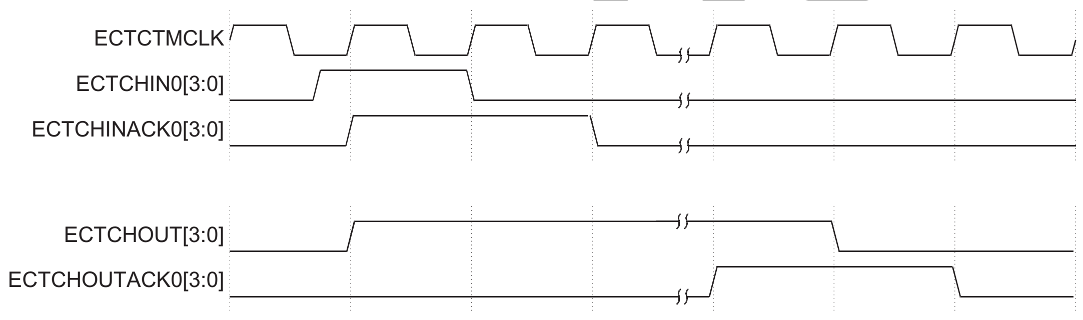

Figure 2-19 CTM timing

When the handshaking is enabled the trigger sources are decoupled from the trigger sinks, which can be in different clock domains. Also the trigger sinks are decoupled from each CI, so that a delayed **ECTCHOUTACK** signal from one of the blocks connected to the interface does not cause the system associated with another interface to stall. All signal synchronization and capture is achieved using **ECTCTMCLK**.

Handshake signalling is used because it is robust with asynchronous systems, and it automatically adapts to changes in clock frequency of components connected to the CTM. In this scheme, an event to the CTM must be held when asserted until an acknowledgement is received. The handshaking can be bypassed if required. However, the system clock domains must be the same and the source signal must be at least one clock cycle.

The CTM contains some sequential elements that generate the acknowledgement signals for each channel in the CI. The handshake is only dependent on the following clock domains:

- the CTM clock domain, **ECTCTMCLK**

- the clock domain of each block connected to the CI.

This removes the possibility of a protocol deadlock, which can occur when one core is powered off or its clock is disabled and the acknowledgement is dependent on an acknowledgement being received from all the devices subscribed to that channel.

If the handshaking is disabled **ECTCHOUT** does not require an acknowledgement. The signal follows the input **CTMORx** of the handshaking circuit. The handshaking can only be bypassed when the two clock domains are the same and clock skew is not a problem.

If the ECTs are stacked as shown in Figure 2-7 on page 2-9, the CTMs are connected as shown in Figure 2-20 on page 2-29. The Interface 0 generates a channel event. This event is propagated to Interface 1 as shown by the thick lines.

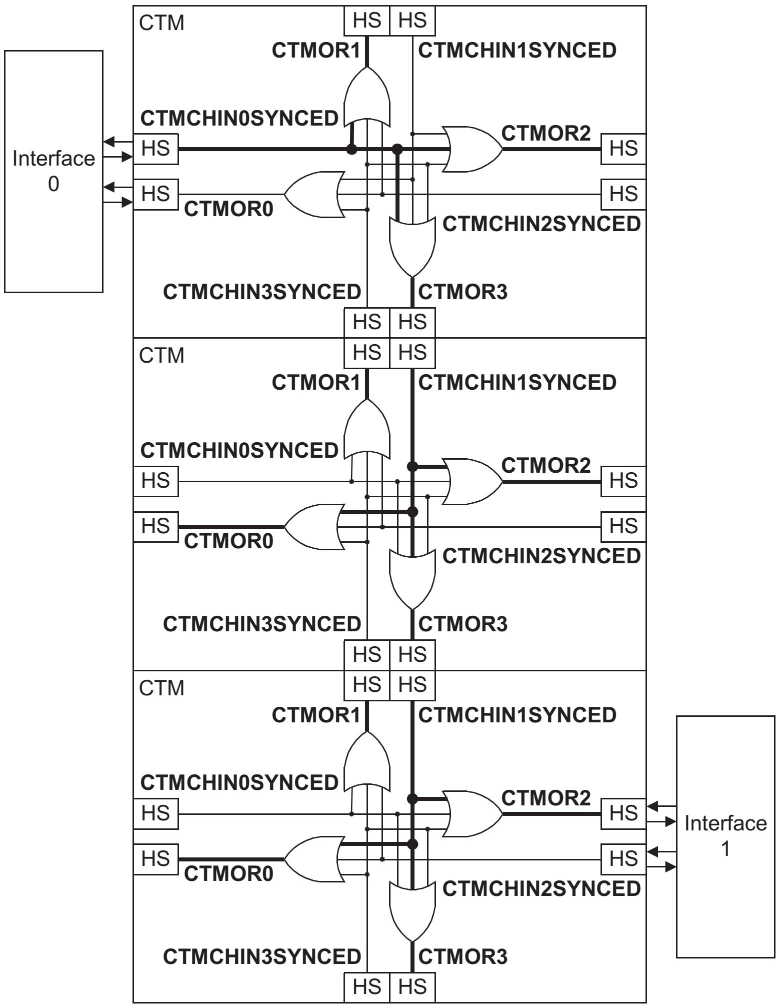

Figure 2-20 Stacked CTMs

2.3.2 ECTCHIN handshake

Figure 2-21 on page 2-30 shows the circuit required for the **ECTCHIN** handshaking.

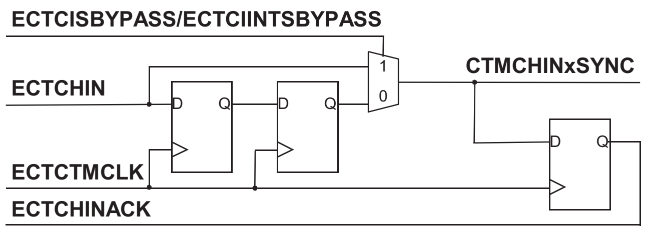

Figure 2-21 ECTCHIN handshake

You can bypass the synchronization when all the subsystems are synchronous, by tying the port **ECTCISBYPASS** HIGH. This is the same circuit as used in the Figure 2-11 on page 2-17. The ECT has four **ECTCHIN** handshaking circuits and therefore four bypass ports **ECTCISBYPASS0** and **ECTCISBYPASS1** for the external CIs, and **ECTCIINTSBYPASS0** and **ECTCIINTSBYPASS1** for the internal CIs.

Note

Both sets of bypass have the same function and **ECTCISBYPASS** is used in the rest of this document.

If **ECTCISBYPASS** is LOW the **CTMCHINxSYNCHED** signal is a synchronized version of the **ECTCHIN** signal. If HIGH, **CTMCHINxSYNCH** is **ECTCHIN**. The circuit uses **CTMCHINxSYNCH**, to generate the acknowledgement signal **ECTCHINACK** one clock cycle after **CTMCHINxSYNCH** is generated.

When a processor uses the same clock, you can tie the relevant bit of the **ECTCISBYPASS** HIGH so that the synchronization is not used.

Figure 2-22 is a timing diagram of the circuit shown in Figure 2-22 when **ECTCISBYPASS** is inactive enabling the synchronization logic.

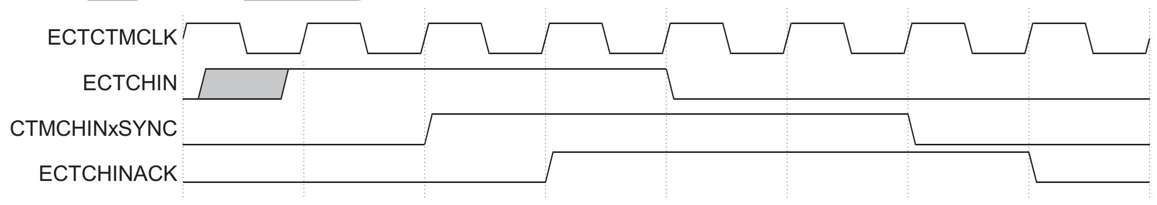

Figure 2-22 ECTCHIN handshake timing

2.3.3 ECTCHOUT handshake

Figure 2-23 on page 2-31 shows the circuit used for the **ECTCHOUT** handshaking.

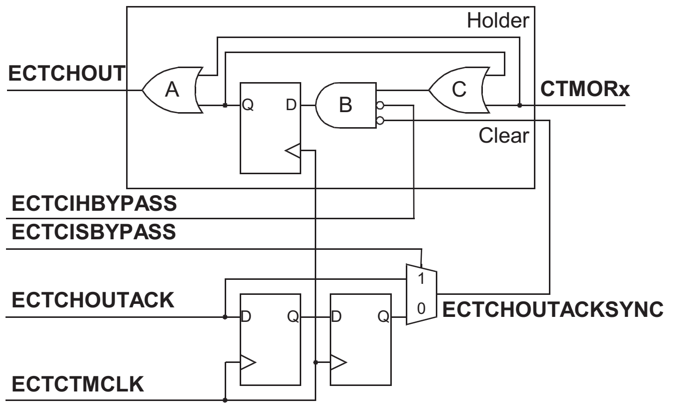

Figure 2-23 ECTCHOUT circuit

The circuit is clocked by the **ECTCTMCLK** signal, and is implemented once for each bit of the **ECTCHOUT** signal. The ECT has four **ECTCHOUT** handshaking circuits and therefore four handshaking bypass buses, **ECTCIHSBYPASS0** and **ECTCIHSBYPASS1** for the external CIs and **ECTCIINTHSBYPASS0** and **ECTCIINTHSBYPASS1** for the internal CIs.

Note

Both sets of bypass have the same function and **ECTCIHSBYPASS** is used in the rest of this document.

When an event is detected on one bit of the **CTMORx**, it is propagated directly to the corresponding bit of **ECTCHOUT** using OR gate A. If **ECTCIHSBYPASS** is LOW, the **ECTCHOUT** is kept activated using the feedback loop through gates C and B, until the acknowledgement signal **ECTCHOUTACKSYNC** is received into the Not input of AND gate B. If the **ECTCHOUT** signal has to be synchronized for the subsystem you must implement the required logic in either their CTI or CTM.

If **CTMORx** is HIGH while the acknowledgement signal has already been received, **ECTCHOUT** is also HIGH until **CMORx** is deactivated.

Figure 2-24 on page 2-32 shows the **ECTCHOUT** handshake timing diagram where **ECTCISBYPASS** = 0.

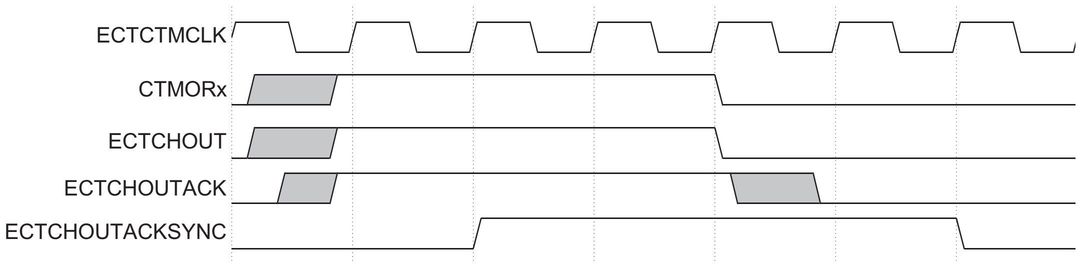

Figure 2-24 ECTCHOUT handshake timing (ECTCISBYPASS = 0)

In Figure 2-24 **ECTCISBYPASS** is inactive (enabling synchronization logic) and **ECTCIHSBYPASS** is inactive (enabling the handshake logic). The acknowledgement signal **ECTCHOUTACK** is received at the same clock cycle as **ECTCHOUT** is propagated. This might be because of the acknowledgement subsystem running at a higher frequency.

Figure 2-25 shows an example of a timing diagram for the **ECTCHOUT** handshake circuit when the frequency of the cross-trigger interface connected is the same as the cross-trigger matrix, and the acknowledgement signal **ECTCHOUTACK** is received a clock cycle after **CTMORx** is propagated.

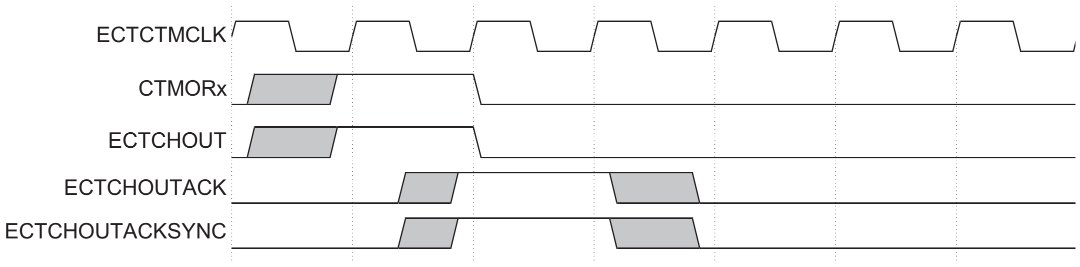

Figure 2-25 ECTCHOUT handshake timing (ECTCISBYPASS = 1)

ECTCISBYPASS is HIGH and **ECTCIHSBYPASS** is LOW. The acknowledgement **ECTCHOUTACK** is received early enough so that the holding circuit is not required to hold the **CTMORx** value.

Figure 2-26 on page 2-33 shows the timing diagram for the **ECTCHOUT** handshake circuit when the frequency of the cross-trigger interface connected is slower than the cross-trigger matrix.

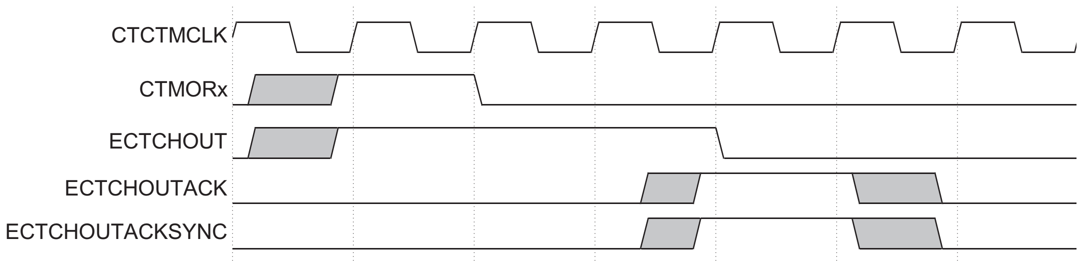

Figure 2-26 ECTCHOUT handshake timing ECTCHOUT held

The acknowledgement signal **ECTCHOUTACK** is received a number of clock cycles after **CTMORx** is propagated. **ECTCHOUT** is held through the feedback circuit gates B and C. The **ECTCISBYPASS** is HIGH and **ECTCIHSBYPASS** is LOW in this case. If **ECTCISBYPASS** is LOW the **ECTCHOUT** is held until **ECTCHOUTACKSYNC** is active on the Not input of gate B.

If **ECTCIHSBYPASS** is HIGH then the handshaking logic is disabled and the OR gate A effectively becomes the only active gate in the circuit, so **ECTCHOUT** is the same as **CTMORx**.

2.3.4 Channel interface connections

Figure 2-27 on page 2-34 shows two CTMs connected together. A CTI connected to the CTM uses the same circuitry but with different port names. If the block connected to the CTM is operating at the same frequency then the bypass circuitry can be enabled.

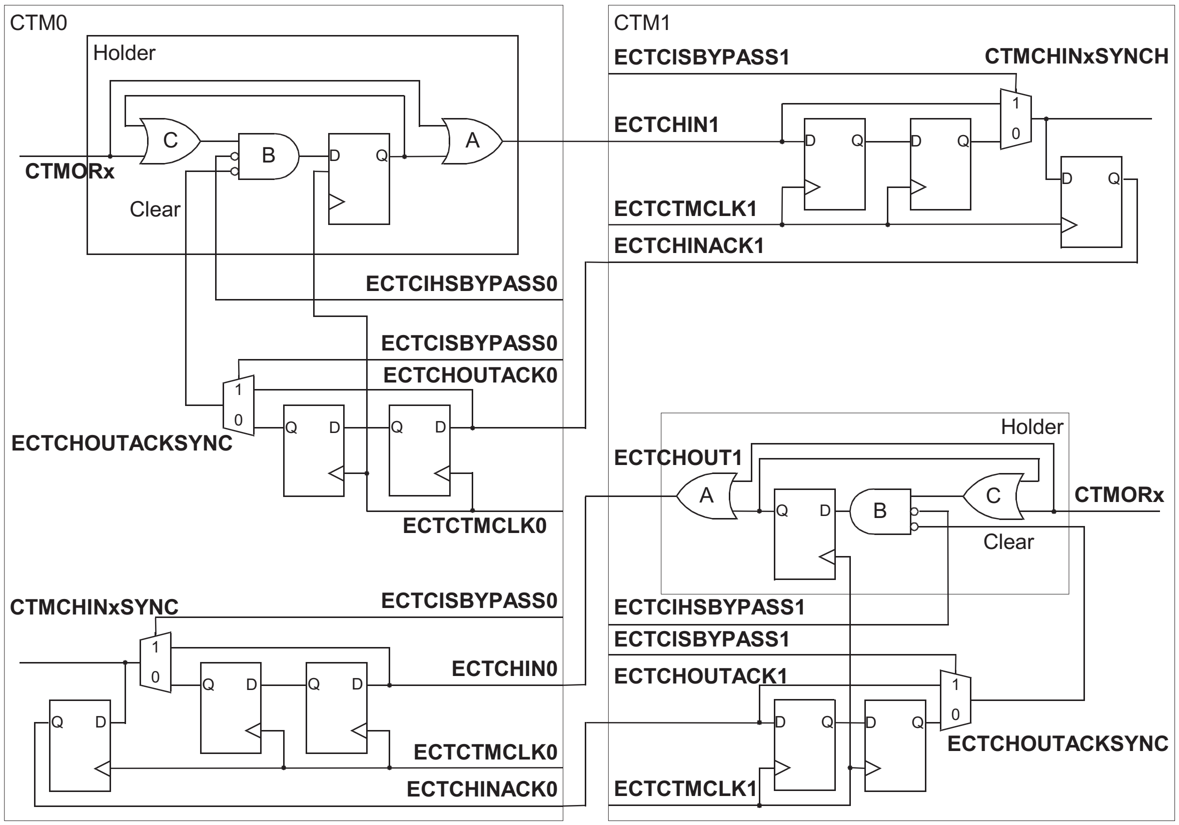

Figure 2-27 Channel handshake interface connections

The **ECTCISBYPASS** ports are effectively one port for the CI. In Figure 2-27 **ECTCISBYPASS0** and **ECTCISBYPASS1** are connected to the same signal. The CI is either synchronous or asynchronous. Therefore, the same synchronization bypass can be used on both sides of the interface. There is one **ECTCIHSBYPASS** signal for each bit of the **ECTCHOUT** bus. As long as both CTMs are running in the same clock domain and there is no requirement for an acknowledgement signal, the handshaking can be bypassed on a bit-by-bit basis.

2.4 AHB interface

The AHB interface is a standard AHB slave. Figure 2-28 shows the main inputs and outputs of the AHB interface.

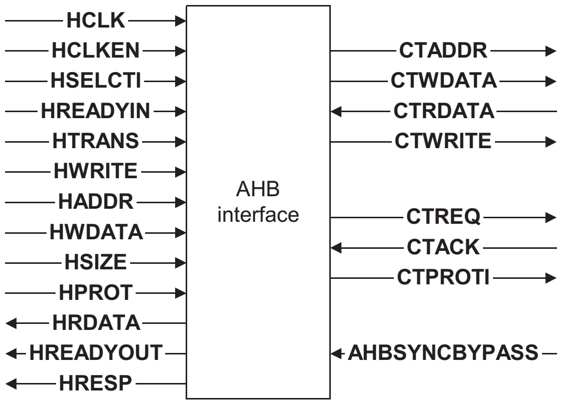

Figure 2-28 AHB interface signals

When **ASYNCHBYPASS** is LOW, synchronization circuits are inserted to cope with **HCLK** and **ECTCTICLK** being asynchronous.

2.5 CTI register access

The CTI is accessed as a memory-mapped device.

ARM recommend that each core is responsible for the configuration of the CTI registers relevant to it. Working like this, the software running on each core, or the debugger attached to each core, can control directly cross-triggering events pertinent to that core.

2.5.1 Access issues

Because you can use the ECT to generate intrusive debug events, it is important to ensure that these macrocells are only used during product development, and that their use (either inadvertently or maliciously) in a production system is prevented.

The following access mechanisms are implemented for protecting the CTI from unwanted configuration:

Debug enable

The DBGEN input enables a hardware signal to enable/disable the debug mapping in the CTI module. For example, this pin can be activated during development but tied LOW in production systems to eliminate the chance of the ECT generating debug activates in a finalized product. This signal can change during operation (though this is not recommended for ARM based processors) to prevent a CPU from generating/receiving debug triggers in certain situations, for example performing critical operations.

Access key, CTILOCK Register

This requires a suitable key to be presented to the CTI before it is enabled or configured.

Privileged access, CTIPROTECTION Register

This restricts access to the configuration registers so that only privileged (supervisor mode) code can access them. In Protection mode user access is effectively masked, so any reads from the ECT returns zeros. The ECT does not respond with an ERROR on the HRESP. This ensures that user code does not require an extra trap handler for DABORTS that occur if an ERROR condition is signalled.

2.6 ECT registers

The ECT registers control the behavior of the CTI attached to a specific core. The following features are implemented:

- the CTI is an AHB slave and all the registers (4kB) can be accessed through this interface
- the **ECTDBGEN** input disables the CTI mapping logic
- the access to registers is locked using a single register
- the registers can optionally be protected where they can only be accessed in supervisor mode
- the ECT has two CTIs.

The ECT registers are located in the two CTIs. Each CTI has its own AHB interface and it is recommended that each core is responsible for the configuration of the CTI registers relevant to it. Figure 2-29 shows the structure of the CTI.

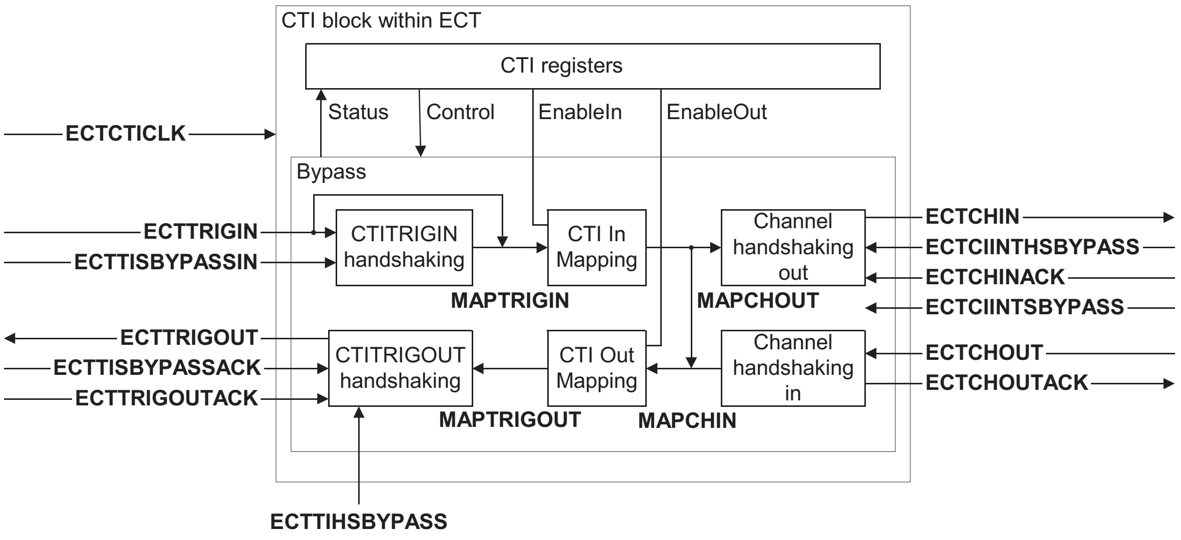

Figure 2-29 CTI structure

The handshaking blocks are described in *ECTTRIGIN handshake* on page 2-17 and *ECTTRIGOUT handshake* on page 2-18. The registers within each CTI are exactly the same.

The configuration registers control the driving of source signals onto and out of the channels to the CTM. Chapter 3 *Programmer's Model* describes the register specification. The bus **MAPTRIGIN[7:0]** is the output of the **ECTTRIGIN** handshaking block as shown in Figure 2-29 on page 2-37.

Figure 2-30 shows an example of ECT In Mapping block registers where part of the **MAPTRIGIN[7:0]** bus bits [2:0] are mapped onto channels 0 and 1 (**MAPCHOUT[3:0]**). In this case, the EnableIn bus comes from the CTIINEN registers, one for each **MAPTRIGIN** bit (8 in total). Each bit of the register masks the input signal for the corresponding channel.

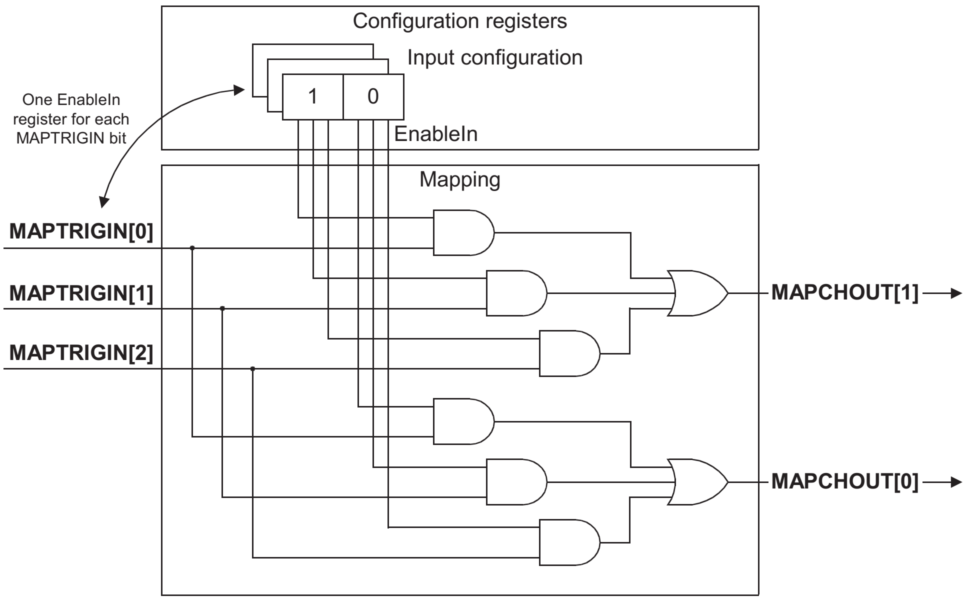

Figure 2-30 Registers controlling the events from the core

Note

You might want to ensure that only one input signal is mapped to a channel (**ECTCHOUT[x]**), and that only one channel (**ECTCHIN[x]**) drives an output trigger. This is fully supported by the ECT. However there is no mechanism to determine which signal raised the event if more than one input signal is enabled onto an output trigger.

Figure 2-31 on page 2-39 shows an example of ECT Out Mapping block.

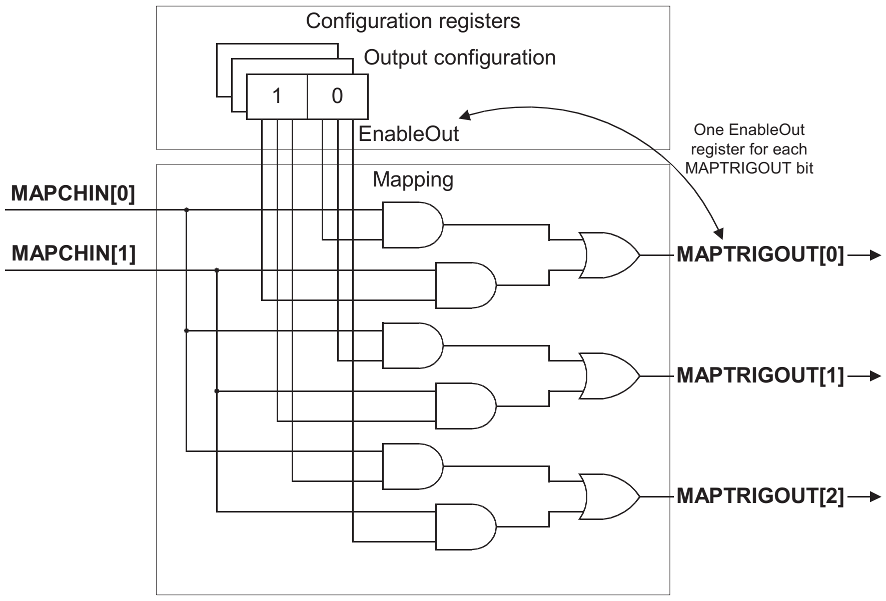

Figure 2-31 Registers controlling the events to the core

Figure 2-31 shows the enable output registers where two bits of the **MAPCHIN[3:0]** bus are mapped onto three bits of the **MAPTRIGOUT[7:0]** bus. There are eight Output Configuration registers, CTIOUTEN0-7, one for each trigger. The registers have enable bits for each of the four channels.

2.6.1 Application trigger

The CTI enables the application to raise an event on one of the channels. Do this by writing a 1 to the CTIAPPSET register. To clear the application trigger you have to write a 1 to the corresponding bit in the CTIAPPCLEAR. This set and clear mechanism enables you to modify the application triggers without requiring a read/modify/write sequence in the code. Each register has one bit corresponding to each of the four channels.

You can also generate application triggers by writing a 1 to the CTIAPPPULSE register. The signal generated is only one cycle and you do not have to clear it by writing to another register. However, you can extend the width of the pulse to multi-cycle through the handshaking circuits.

Figure 2-32 on page 2-40 shows an example of the circuit used to implement the application trigger.

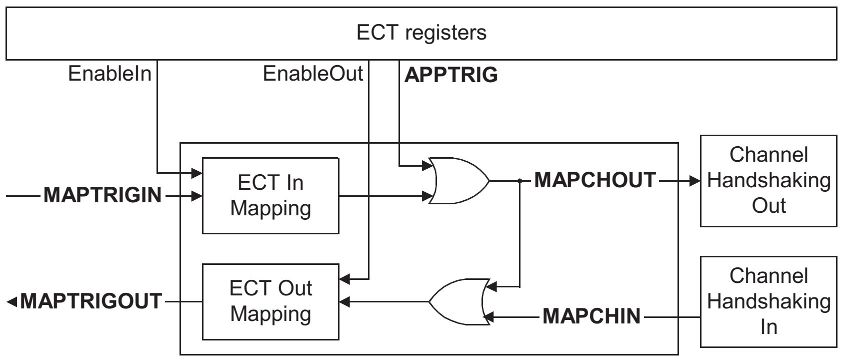

Figure 2-32 Application-driven trigger

This enables the application to raise a special event.

2.6.2 Possible system deadlock situation to avoid

You must consider carefully the use of the channels so that a possible system deadlock cannot occur. See Figure 2-33.

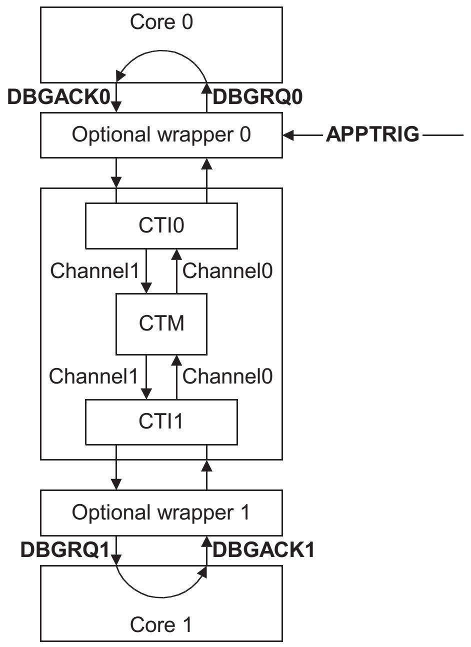

Figure 2-33 System deadlock

Figure 2-33 on page 2-40 shows the ECT mapping logic set up as follows:

1. A **DBGACK** from one core generates a **DBGRQ** on the other core.
2. **DBGRQ0** is generated by an **APPTRIG** signal.
3. Core0 responds with a **DBGACK0** signal when the core is in debug mode. This is propagated to **DBGRQ1** by the ECT Channel1.
4. Core1 responds with a **DBGACK1** that in turn is then propagated to Core0 through Channel0, so generating another **DBGRQ0** that starts the whole process again.

The deadlock situation occurs as both cores trigger the other back into debug mode as soon as it enters execution mode (**DBGACK** going LOW). You must avoid this situation in one of the following ways:

- Set up the system so that only one core can trigger the **DBGRQ** on the other core/s.
- Disable the mapping logic in Debug mode and then re-enable when all cores are out of debug.
- Include conditioning logic in the wrapper so that the **DBGRQ** signal is dependent on another channel as shown in Figure 2-34. An application trigger in the debug routine generates Channel2. The debug routine determines that all the cores mapped to the **DBGRQ** are out of debug (**DBGACK**s are LOW) before allowing the **DBGRQ** signal to be generated to the core.

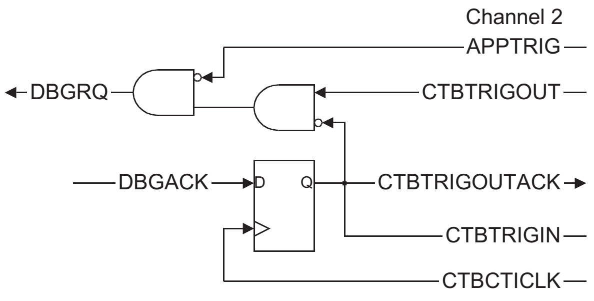

Figure 2-34 Wrapper to condition DBGRQ signal

Chapter 3
Programmer's Model

This chapter describes the ECT registers and provides information for programming the microcontroller. It contains the following sections:

3.1 About the programmer's model

The base addresses of the CTIs are not fixed, and can be different for any particular system implementation. However, the offset of any particular register from the base address is fixed.

The following locations are reserved for future possible expansion and must not be used during normal operation:

- `0x040-0x09C`
- `0x0C0-0x12C`.

The following applies to all registers:

- reserved or unused bits of registers must be written as 0, and ignored on a read unless otherwise stated in the text
- all register bits are reset to 0 unless otherwise stated in the text
- all registers must be accessed as words and so are compatible with big and little endian systems.

3.2 Summary of CTI registers

Table 3-1 shows the CTI registers.

Table 3-1 CTI register summary

Address offset	Register	Type	Width	Reset value	Description
`0x000`	CTICONTROL	Read/ write	1	`0x0`	See *CTI Control Register, CTICONTROL* on page 3-5
`0x004`	CTISTATUS	Read	2	`0x1` or `0x3`	See *CTI Status Register, CTISTATUS* on page 3-5
`0x008`	CTILOCK	Write	32	`0x00000000`	See *CTI Lock Enable Register, CTILOCK* on page 3-6
`0x00C`	CTIPROTECTION	Read	1	`0x0`	See *CTI Protection Enable Register, CTIPROTECTION* on page 3-6
`0x010`	CTIINTACK	Write	8	-	See *CTI Interrupt Acknowledge Register, CTIINTACK* on page 3-7
`0x014`	CTIAPPSET	Read/ write	4	`0x0`	See *CTI Application Trigger Set Register, CTIAPPSET* on page 3-7
`0x018`	CTIAPPCLEAR	Read/ write	4	`0x0`	See *CTI Application Trigger Clear Register, CTIAPPCLEAR* on page 3-8
`0x01C`	CTIAPPPULSE	Write	4	`0x0`	See *CTI Application Pulse Register, CTIAPPPULSE* on page 3-8
`0x020-0x03C`	CTIINEN	Read/ write	4	`0x00`	See *CTI Trigger to Channel Enable Registers, CTIINEN0-7* on page 3-9
`0x0A0-0x0BC`	CTIOUTEN	Read/ write	4	`0x00`	See *CTI Channel to Trigger Enable Registers, CTIOUTEN0-7* on page 3-9
`0x130`	CTITRIGINSTATUS	Read	8	-	See *CTI Trigger In Status Register, CTITRIGINSTATUS* on page 3-10
`0x134`	CTITRIGOUTSTATUS	Read	8	`0x00`	See *CTI Trigger Out Status Register, CTITRIGOUTSTATUS* on page 3-10
`0x138`	CTICHINSTATUS	Read	4	-	See *CTI Channel In Status Register, CTICHINSTATUS* on page 3-11
`0x13C`	CTICHOUTSTATUS	Read	4	`0x0`	See *CTI Channel Out Status Register, CTICHOUTSTATUS* on page 3-11

Table 3-1 CTI register summary (continued)

Address offset	Register	Type	Width	Reset value	Description
`0xFE0`	CTIPERIPHID0	Read	8	`0x00`	See *CTIPERIPHID0 Register* on page 3-13
`0xFE4`	CTIPERIPHID1	Read	8	`0x19`	See *CTIPERIPHID1 Register* on page 3-13
`0xFE8`	CTIPERIPHID2	Read	8	`0x04`	See *CTIPERIPHID2 Register* on page 3-14
`0xFEC`	CTIPERIPHID3	Read	8	`0x00`	See *CTIPERIPHID3 Register* on page 3-14
`0xFF0`	CTIPCELLID0	Read	8	`0x0D`	See *CTIPCELLID0 Register* on page 3-15
`0xFF4`	CTIPCELLID1	Read	8	`0xF0`	See *CTIPCELLID1 Register* on page 3-15
`0xFF8`	CTIPCELLID2	Read	8	`0x05`	See *CTIPCELLID2 Register* on page 3-16
`0xFFC`	CTIPCELLID3	Read	8	`0xB1`	See *CTIPCELLID3 Register* on page 3-16

3.3 Register descriptions

This section describes the ECT CTI registers.

Note

All register wait states are a minimum of three cycles for reads and four cycles for writes if **HCLK** and **ECTCTICLK** are the same. Otherwise wait states depend on the ratio between the two clocks.

3.3.1 CTI Control Register, CTICONTROL

CTICONTROL is a read/write register. This register enables the CTI.

Table 3-2 shows the bit assignments for the CTI Control Register.

Table 3-2 CTI Control Register bit assignments

Bits	Name	Description
[31:1]	Reserved	Reserved. read as zero, do not modify.
[0]	GLBEN	Enables or disables the ECT: 0 = disabled (reset) 1 = enabled.When disabled, all cross triggering mapping logic functionality is disabled for this processor.

3.3.2 CTI Status Register, CTISTATUS

CTISTATUS is a read-only register. This register provides the locked and **ECTDBGEN** status of the CTI.

Table 3-3 shows the bit assignments for the CTI Status Register.

Table 3-3 CTI Status Register bit assignments

Bits	Name	Description
[31:2]	Reserved	Reserved. read as zero, do not modify.
[1]	DGBEN	Shows the status of the **ECTDBGEN** port. The **ECTDBGEN** is used to enable the Trigger and CIs to the ECT: 0 = Interfaces are disabled 1 = Interfaces are enabled. Because the AHB interface is not controlled by the **ECTDBGEN** port this register can be read while **ECTDBGEN** is LOW. The reset value depends on the **ECTDGEN** port.
[0]	LOCKED	Shows the locked status of the CTI:0 = Access to the CTI is not locked1 = Access to the CTI is locked (reset). The access can be unlocked by using the CTILock register.

3.3.3 CTI Lock Enable Register, CTILOCK

CTILOCK is a write-only register. This register enables or disables all other register write accesses by providing a LOCKED mode for the AHB interface. When the AHB interface is locked, write accesses to the CTI registers except CTILOCK are ignored. To unlocked the AHB interface, a LOCKKEY value, `0x0ACCE550`, must be written to the CTILOCK register. When unlocked, the AHB interface accepts write accesses. To lock the AHB interface data that is not equal to LOCKKEY value, `0x0ACCE550`, is written into CTILOCK. The LOCK mode protects the CTI from spurious writes. The LOCKED bit in the CTISTATUS register indicates the status of the lock. The write access to CTILOCK is affected by the protection status. Therefore if protection mode, CTIPROTECTION, is enabled, accesses to the CTILOCK must be handled in privilege modes.

Table 3-4 shows the bit assignments for the CTI Lock Enable Register.

Table 3-4 CTI Lock Enable Register bit assignments

Bits	Name	Description
[31:0]	LOCKKEY	To enable the other registers in the CTI to be accessible, the correct access code of `0x0ACCE550` must be written to this register. To disable access to the other CTI registers, any value other than `0x0ACCE550` must be written to this register. In reset the lock is set.

3.3.4 CTI Protection Enable Register, CTIPROTECTION

CTIPROTECTION is a read-only register. This register enables or disables protected register access, stopping register accesses when the processor is in user mode. If the CTIPROTECTION Register is enabled 1 then all register access must be done in privileged mode.

Table 3-5 shows the bit assignments for the CTI Protection Enable Register.

Table 3-5 CTI Protection Enable Register bit assignments

Bits	Name	Description
[31:0]	Reserved	Reserved. read as zero, do not modify.
[0]	PROTECTION	Enables or disables protected register access: 0 = protection mode disabled (reset) 1 = protection mode enabled. When enabled, only privileged mode accesses (reads and writes) can access the CTI registers, that is when **HPROT[1]** is set HIGH for the current transfer. If an access is made in user mode all registers return 0. **HRESP[1:0]** does not return an error because operating systems might not be able to handle this type of error response. When disabled, both user mode and privileged mode can access the registers in the CTI, except for the CTIPROTECTION that can only be accessed in privileged mode, even when the protection mode is disabled.

Note

If a system bus master accessing the ECT cannot generate accurate protection information, the register stays in its reset state to enable user mode access.

3.3.5 CTI Interrupt Acknowledge Register, CTIINTACK

CTIINTACK is a write-only register. Any bits written as a 1 cause the **CTITRIGOUT** output signal to be acknowledged. The acknowledgement is cleared when **MAPTRIGOUT** is deactivated. This register is used when the **CTITRIGOUT** is used as a sticky class output.

Table 3-6 shows the bit assignments for the CTI Interrupt Acknowledge Register.

Table 3-6 CTI Interrupt Acknowledge Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	INTACK	Acknowledges the corresponding **CTITRIGOUT** output: 0 = nothing happens 1 = **CTITRIGOUT** is acknowledged and will be cleared when **MAPTRIGOUT** is low. There is one bit of the register for each **CTITRIGOUT** output.

3.3.6 CTI Application Trigger Set Register, CTIAPPSET

CTIAPPSET is a read/write register. A write to this register causes a channel event to be raised, corresponding to the bit written to.

Table 3-7 shows the bit assignments for the CTI Application Trigger Set Register.

Table 3-7 CTI Application Trigger Set Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	APPSET	Setting a bit HIGH generates a channel event for the selected channel. Read: 0 = application trigger inactive (reset) 1 = application trigger active. Write: 0 = no effect 1 = generate channel event. There is one bit of the register for each channel.

Note

The CTIINEN registers do not affect the CTIAPPSET operation.

3.3.7 CTI Application Trigger Clear Register, CTIAPPCLEAR

CTIAPPCLEAR is a read/write register. A write to this register causes a channel event to be cleared, corresponding to the bit written to.

Table 3-8 shows the bit assignments for the CTI Application Trigger Clear Register.

Table 3-8 CTI Application Trigger Clear Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	APPCLEAR	Clears corresponding bits in the CTIAPPSET register: 0 = no effect 1 = application trigger disabled in the CTIAPPSET register. There is one bit of the register for each channel.

3.3.8 CTI Application Pulse Register, CTIAPPPULSE

CTIAPPPULSE is a write-only register. A write to this register causes a channel event pulse (one **ECTCTICLK** period) to be generated, corresponding to the bit written to. The pulse external to the ECT can be extended to multi-cycle by the handshaking interface circuits. This register clears itself immediately, so it can be repeatedly written without software having to clear it.

Table 3-9 shows the bit assignments for the CTI Application Pulse Register.

Table 3-9 CTI Application Pulse Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	APPULSE	Setting a bit HIGH generates a channel event pulse for the selected channel. Write: 0 = no effect 1 = channel event pulse generated for one **ECTCTICLK** period. There is one bit of the register for each channel.

Note

The CTIINEN registers do not affect the CTIAPPPULSE operation.

3.3.9 CTI Trigger to Channel Enable Registers, CTIINEN0-7

These registers are read/write registers that enable the signalling of an event on a CTM channel/s when the core issues a trigger (**ECTTRIGIN**) to the CTI. There is one register for each of the eight **ECTTRIGIN** inputs. Within each register there is one bit for each of the four channels implemented. These registers do not affect the application trigger operations.

Table 3-10 shows the bit assignments for the CTI Trigger to Channel Enable Registers.

Table 3-10 CTI Trigger to Channel Enable Registers bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	TRIGINEN	Enables a cross trigger event to the corresponding channel when an **ECTTRIGIN** is activated: 0 = disables the **ECTTRIGIN** signal from generating an event on the respective channel of the CTM 1 = enables the **ECTTRIGIN** signal to generate an event on the respective channel of the CTM. There is one bit of the register for each of the 4 channels. For example in register CTIINEN0, **TRIGINEN[0]** set to 1 enables **ECTTRIGIN** onto channel 0.

3.3.10 CTI Channel to Trigger Enable Registers, CTIOUTEN0-7

CTIOUTEN0-7 is a read/write register. These registers are read/write registers that define which channel(s) can generate a **ECTTRIGOUT** output. There is one register for each of the eight **ECTTRIGOUT** outputs. Within each register there is one bit for each of the four channels implemented. These registers affect the mapping from application trigger to trigger outputs.

Table 3-11 shows the bit assignments for the CTI Channel to Trigger Enable Registers.

Table 3-11 CTI Channel to Trigger Enable Registers bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	TRIGOUTEN	Changing the value of this bit from a 0 to a 1 will enable a channel event for the corresponding channel to generate a **ECTTRIGOUT** output: 0 = the channel input (**CTICHIN**) from the CTM is not routed to the **ECTTRIGOUT** output 1 = the channel input (**CTICHIN**) from the CTM is routed to the **ECTTRIGOUT** output. There is one bit for each of the four channels. For example in register CTIOUTEN0, enabling bit 0 enables **CTICHIN[0]** to cause a trigger event on the **ECTTRIGOUT[0]** output.

3.3.11 CTI Trigger In Status Register, CTITRIGINSTATUS

CTITRIGINSTATUS is a read-only register. This register provides the status of the **ECTTRIGIN** inputs.

Table 3-12 shows the bit assignments for the CTI Trigger In Status Register.

Table 3-12 CTI Trigger In Status Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	TRIGINSTATUS	Shows the status of the **ECTTRIGIN** inputs: 0 = **ECTTRIGIN** is inactive 1 = **ECTTRIGIN** is active. Because the register provides a view of the raw **ECTTRIGIN** inputs, the reset value is unknown. There is one bit of the register for each trigger input.

3.3.12 CTI Trigger Out Status Register, CTITRIGOUTSTATUS

CTITRIGOUTSTATUS is a read-only register. This register provides the status of the **ECTTRIGOUT** outputs.

Table 3-13 shows the bit assignments for the CTI Trigger Out Status Register.

Table 3-13 CTI Trigger Out Status Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	TRIGOUTSTATUS	Shows the status of the **ECTTRIGOUT** outputs: 0 = **ECTTRIGOUT** is inactive (reset) 1 = **ECTTRIGOUT** is active. There is one bit of the register for each trigger output.

3.3.13 CTI Channel In Status Register, CTICHINSTATUS

CTICHINSTATUS is a read-only register. This register provides the status of the CTI **CTICHIN** inputs.

Note

This is the CTI channel input **CTICHIN** and not the ECT **ECTCHIN** input.

Table 3-14 shows the bit assignments for the CTI Channel In Status Register.

Table 3-14 CTI Channel In Status Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	ECTCHINSTATUS	Shows the status of the **CTICHIN** inputs: 0 = **CTICHIN** is inactive 1 = **CTICHIN** is active. Because the register provides a view of the raw **CTICHIN** inputs from the CTM, the reset value is unknown. There is one bit of the register for each channel input.

3.3.14 CTI Channel Out Status Register, CTICHOUTSTATUS

CTICHOUTSTATUS is a read-only register. This register provides the status of the CTI **CTICHOUT** outputs.

Note

This is the CTI channel output **CTICHOUT** and not the ECT **ECTCHOUT** output.

Table 3-15 shows the bit assignments for the CTI Channel Out Status Register.

Table 3-15 CTI Channel Out Status Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	ECTCHOUTSTATUS	Shows the status of the **CTICHOUT** outputs: 0 = **CTICHOUT** is inactive (reset) 1 = **CTICHOUT** is active. There is one bit of the register for each channel output.

3.3.15 Peripheral Identification Registers, CTIPERIPHID0-3

CTIPERIPHID0-3 are read-only registers. These registers are four 8-bit registers that span address locations `0xFE0`-`0xFEC`. The registers can conceptually be treated as a single 32-bit register. The read-only registers provide the following options for the peripheral:

Part number [11:0] This identifies the peripheral. The three digit product code `0x900` is used for the CTI.

Designer [19:12] This is the identification of the designer. ARM Ltd is `0x41` (ASCII A).

Revision number [23:20]

This is the revision number of the peripheral. The revision number starts from 0, and the value is revision dependent.

Configuration [31:24]

This is the configuration option of the peripheral. The configuration value is 0.

Figure 3-1 shows the bit assignments for the Peripheral Identification Registers.

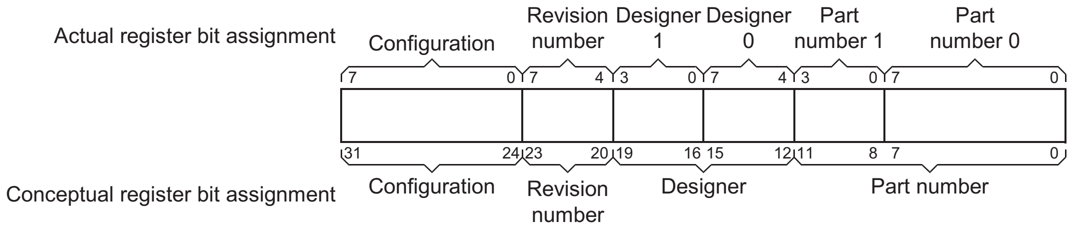

Figure 3-1 Peripheral Identification Registers bit assignments

The four 8-bit peripheral identification registers are described in the following sections:

CTIPERIPHID0 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-16shows the bit assignments for the CTIPERIPHID0 Register.

Table 3-16 CTIPERIPHID0 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	Partnumber0	These bits read back as `0x00`.

CTIPERIPHID1 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-17 shows the bit assignments for the CTIPERIPHID1 Register.

Table 3-17 CTIPERIPHID1 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:4]	Designer0	These bits read back as `0x1`
[3:0]	Partnumber1	These bits read back as `0x9`

CTIPERIPHID2 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-18 shows the bit assignments for the CTIPERIPHID2 Register.

Table 3-18 CTIPERIPHID2 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:4]	Revision	These bits read back as the revision number, which can be between 0 and 15
[3:0]	Designer1	These bits read back as `0x4`

CTIPERIPHID3 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-19 shows the bit assignments for the CTIPERIPHID3 Register.

Table 3-19 CTIPERIPHID3 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:2]	Configuration	These bits read back as `0x0`
[1:0]	Configuration	Indicates the number of interrupts supported: 00 = 32 (default) 01 = 64 10 = 256

3.3.16 Identification Registers, CTIPCELLID0-3

CTIPCELLID0-3 are read-only registers. The CTIPCELLID0-3 Registers are four 8-bit registers that span address locations `0xFF0`-`0xFFC`. These read-only registers can conceptually be treated as a single 32-bit register. The register is used as a standard cross-peripheral identification system. Figure 3-2 on page 3-15shows the bit assignment for the CTIPCELLID0-3 Registers.

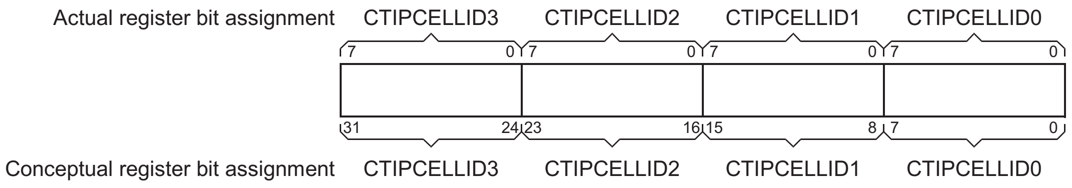

Figure 3-2 Identification Registers bit assignments

The four 8-bit registers are described in the following subsections:

- *CTIPCELLID0 Register*
- *CTIPCELLID1 Register*
- *CTIPCELLID2 Register* on page 3-16
- *CTIPCELLID3 Register* on page 3-16.

CTIPCELLID0 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-20 shows the bit assignments for the CTIPCELLID0 Register.

Table 3-20 CTIPCELLID0 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:0]	CTIPCELLID0	These bits read back as `0x0D`

CTIPCELLID1 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-21 shows the bit assignments for the CTIPCELLID1 Register.

Table 3-21 CTIPCELLID1 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:0]	CTIPCELLID1	These bits read back as `0xF0`

CTIPCELLID2 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-22 shows the bit assignments for the CTIPCELLID2 Register.

Table 3-22 CTIPCELLID2 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:0]	CTIPCELLID2	These bits read back as `0x05`

CTIPCELLID3 Register

This register is hard-coded and the fields within the register determine the reset value. This register can be accessed with three wait states. Table 3-23 shows the bit assignments for the CTIPCELLID3 Register.

Table 3-23 CTIPCELLID3 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:0]	CTIPCELLID3	These bits read back as `0xB1`

Chapter 4
Programmer's Model for Test

This chapter describes the additional logic for functional verification and provisions made for production testing. It contains the following sections:

4.1 ECT/CTI test harness overview

The additional logic for functional verification and production testing enables:

- capture of CTI input signals to the block
- stimulation of the CTI output signals.

The integration vectors provide a way of verifying that the TI of the ECT is correctly wired into a system. This is done by separately testing two groups of signals:

AMBA signals	These are tested by checking the connections of all the address and data bits.
Intra-chip signals	The tests for these signals are system-specific and enable you to write the necessary tests. Additional logic is implemented enabling you to read and write to each intra-chip input/output signal.

Test registers control these test features. This enables you to test the TI of the ECT in isolation from the rest of the system using only transfers from the AMBA AHB. As the CTM does not have an AHB interface only the signals into and out of the CTI can be tested with the means described. The CI signals into the CTM are tested implicitly because another ECT CI is connected to it. The integration tests then have to test that one ECT can raise a trigger event on the other ECT. The bypass signals are tied either HIGH or LOW and, as such, are not required in the integration test registers.

See Figure 2-5 on page 2-7 which shows the ECT ports.

A global bit called ITEN must be activated before any integration tests can be performed.

Figure 4-1 shows integration logic.

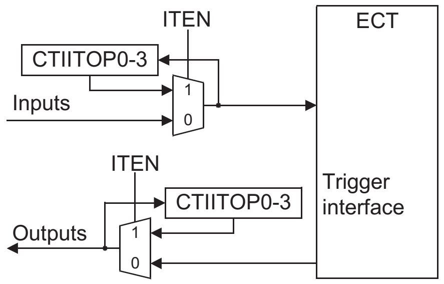

Figure 4-1 Integration logic

The registers CTIITIP0-3 and CTIITOP0-3 behave as follows:

- When the corresponding enable bit it active, a write to the register specifies the value to be driven for the corresponding signal
- Read the register returns the value after the multiplexor. When the enable bit is active, returns the value that was written into the register.

4.2 Scan testing

The ECT is designed to simplify:

- insertion of scan test cells
- use of *Automatic Test Pattern Generation* (ATPG).

This is the recommended method of manufacturing test.

4.3 Test registers

Table 4-1 shows how the CTI test registers are memory-mapped.

Table 4-1 Test registers memory map

Address offset	Register	Type	Width	Reset value	Description
`0x200`	CTIITCR	Read/write	1	`0x0`	See *CTIITCR Register*
`0x204`	CTIITIP0	Read/write	8	-	See *CTIITIP0 Register* on page 4-6
`0x208`	CTIITIP1	Read/write	4	-	See *CTIITIP1 Register* on page 4-6
`0x20C`	CTIITIP2	Read/write	8	-	See *CTIITIP2 Register* on page 4-7
`0x210`	CTIITIP3	Read/write	4	-	See *CTIITIP3 Register* on page 4-7
`0x214`	CTIITOP0	Read/write	8	`0x00`	See *CTIITOP0 Register* on page 4-7
`0x218`	CTIITOP1	Read/write	4	`0x00`	See *CTIITOP1 Register* on page 4-8
`0x21C`	CTIITOP2	Read/write	8	`0x00`	See *CTIITOP2 Register* on page 4-8
`0x220`	CTIITOP3	Read/write	4	`0x00`	See *CTIITOP3 Register* on page 4-9

All registers are only accessible when the CTIITCR Register has been configured correctly (see *CTIITCR Register*).

The registers in Table 4-1 are described in the following sections:

For all registers, unimplemented bits return 0 when read and ignore writes. The test registers are included to aid test, debug, and integration of the ECT.

4.3.1 CTIITCR Register

CTIITCR is a read/write register. This register is a one-bit read/write test control register. The ITEN bit in this register controls the input and output test control registers. This register must only be used in test mode.

This register can be accessed with three wait states for read and four for write.

Table 4-2 shows the bit assignments for the CTIITCR Register.

Table 4-2 CTIITCR Register bit assignments

Bits	Name	Description
[31:1]	Reserved	Reserved. read as zero, do not modify
[0]	ITEN	Integration test enable: 0 = normal mode (reset) 1 = test mode enabled.

4.3.2 CTIITIP0 Register

CTIITP0 is a read/write register. This register is an 8-bit register used to control and read the values of the **ECTTRIGIN** inputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-3 shows the bit assignments for the CTIITIP0 Register.

Table 4-3 CTIITIP0 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	ECTTRIGIN	Read the value of the **ECTTRIGIN** input when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.3 CTIITIP1 Register

CTIITP1 is a read/write register. This register is a 4-bit register used to control and read the values of the **CTICHIN** inputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-4 shows the bit assignments for the CTIITIP1 Register.

Table 4-4 CTIITIP1 Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	CTICHIN	Read the value of the **CTICHIN** input when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.4 CTIITIP2 Register

CTIITP2 is a read/write register. This register is an 8-bit register used to control and read the values of the **ECTTRIGOUTACK** inputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-5 shows the bit assignments for the CTIITIP2 Register.

Table 4-5 CTIITIP2 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify
[7:0]	ECTTRIGOUTACK	Read the value of the **ECTTRIGOUTACK** inputs when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.5 CTIITIP3 Register

CTIITP3 is a read/write register. This register is a 4-bit register used to control and read the values of the **CTICHOUTACK** inputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-6 shows the bit assignments for the CTIITIP3 Register.

Table 4-6 CTIITIP3 Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	ECTCHOUTACK	Read the value of the **CTICHOUTACK** inputs when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.6 CTIITOP0 Register

CTIITOP0 is a read/write register. This register is an 8-bit register used to control and read the value of the **ECTTRIGOUT** outputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-7 shows the bit assignment of the CTIITOP0 Register.

Table 4-7 CTIITOP0 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	ECTTRIGOUT	Read the value of the **ECTTRIGOUT** output when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.7 CTIITOP1 Register

CTIITOP1 is a read/write register. This register is a 4-bit register used to control and read the value of the **CTITCHOUT** outputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-8 shows the bit assignments for the CTIITOP1 Register.

Table 4-8 CTIITOP1 Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	CTITCHOUT	Read the value of the **CTICHOUT** output when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.8 CTIITOP2 Register

CTIITOP2 is a read/write register. This register is an 8-bit register used to control and read the value of the **ECTTRIGINACK** outputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-9 shows the bit assignments for the CTIITOP2 Register.

Table 4-9 CTIITOP2 Register bit assignments

Bits	Name	Description
[31:8]	Reserved	Reserved. read as zero, do not modify.
[7:0]	ECTTRIGINACK	Read the value of the **ECTTRIGINACK** output when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

4.3.9 CTIITOP3 Register

CTIITOP3 is a read/write register. This register is a 4-bit register used to control and read the value of the **ECTCHINACK** outputs. This register must only be used in test mode. This register can be accessed with three wait states for read and four for write.

Table 4-10 shows the bit assignments for the CTIITOP3 Register.

Table 4-10 CTIITOP3 Register bit assignments

Bits	Name	Description
[31:4]	Reserved	Reserved. read as zero, do not modify.
[3:0]	CTICHINACK	Read the value of the **CTICHINACK** output when the CTIITCR ITEN bit is LOW. Read the value of this field when the CTIITCR ITEN bit is HIGH. Write sets input to written value when the CTIITCR ITEN bit is HIGH.

Appendix A
Signal Descriptions

This appendix describes the signals that interface with the ARM ECT. It contains the following sections:

A.1 ECT CTI AMBA AHB signals

The CTI module is connected to the AMBA AHB as a bus slave. Table A-1 lists the AHB signals that are used and produced. The ECT has two AHB interfaces one for each CTI.

Note

x in a signal name denotes 0 or 1.

Table A-1 ECT CTI AMBA AHB signals

Name	Type	Source/ destination	Description
HADDRx[11:2]	Input	Master	AHB system address bus.
HCLKx	Input	Clock source	AMBA AHB bus clock, used to time all bus transfers. All signal timings are related to the rising edge of **HCLK**.
HPROTx[3:0]	Input	Master	Memory access protection type:**HPROT[1]** can be User mode (0) or privileged mode (1). **HPROT[3]**, **HPROT[2]** and **HPROT[0]** are not used.
HRDATAx[31:0]	Output	Slave	Read data bus, used to transfer data from bus slaves to bus master during read operations.
HREADYINx	Input	External slave	Transfer done signal, generated by an alternate slave. When HIGH, indicates that a transfer is complete. Can be driven LOW to extend a transfer.
HREADYOUTx	Output	Slave	Transfer done signal, generated by the ECT. When HIGH, indicates that a transfer is complete. Can be driven LOW to extend a transfer from the ECT.
HRESETnx	Input	Reset controller	AHB bus reset, active LOW.
HRESPx[1:0]	Output	Slave	Transfer response, which provides additional transfer status information. The response can be OKAY, ERROR, RETRY or SPLIT. The ECT responds with either OKAY or ERROR.
HSELCTIx	Input	Decoder	Slave select signal, which is a combinatorial decode of the address bus. It indicates that the current transfer is intended for the selected slave.
HSIZEx[2:0]	Input	Master	Size of the transfer, which must be word (32-bit) for the CTI (**HSIZE[2:0]** = 0b010).

Table A-1 ECT CTI AMBA AHB signals (continued)

Name	Type	Source/ destination	Description
HTRANSx[1:0]	Input	Master	Transfer type, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE, or BUSY.
HWDATAx[31:0]	Input	Master	Write data bus, used to transfer data from bus master to bus slaves during write operations.
HWRITEx	Input	Master	Transfer direction. Indicates a write transfer when HIGH, and a read transfer when LOW.

A.2 ECT clock, enable, and reset signals

Table A-2 lists the clock, enable, and reset signals for the CTI.

Table A-2 Clock, enable, and reset signals

Name	Type	Source/ destination	Description
ECTCTICLKx	Input	System	Clock used for the triggers and register control
nECTCTIRESETx	Input	System	Low-level reset
ECTCTMCLK	Input	System	Clock
nECTCTMRESET	Input	System	Asynchronous reset

A.3 ECT synchronization bypass signals

Table A-3 lists the CTI synchronization bypass signals for the CTI.

Table A-3 CTI synchronization bypass signals

Name	Type	Source/ destination	Description
ECTCIHSBYPASSx[3:0]	Input	Tied 0 or 1	Bypass handshaking between on the external the Channel Interfaces. Tie HIGH when all the cross trigger blocks are in the same clock domain, clock skew is not a problem, and the source trigger is a least one clock cycle, to bypass the handshaking circuits.
ECTCIINTHSBYPASSx[3:0]	Input	Tied 0 or 1	Bypass handshaking between the internal Channel Interfaces. Tie HIGH when all the cross trigger blocks are in the same clock domain, clock skew is not a problem, and the source trigger is a least one clock cycle, to bypass the handshaking circuits.
ECTCIINTSBYPASSx	Input	Tied 0 or 1	Bypass synchronization on the internal Channel Interfaces. Tie HIGH when both sides of the Channel Interface are synchronous to bypass the synchronization circuits.
ECTCISBYPASSx	Input	Tied 0 or 1	Bypass synchronization on the external Channel Interfaces. Tie HIGH when both sides of the Channel Interface are synchronous to bypass the synchronization circuits.
ECTCTIAHBSBYPASSx	Input	Tied 0 or 1	Bypass synchronization between **HCLK** and **ECTCTICLK**. Tie HIGH when **HCLK** and **ECTCTICLK** are synchronous.
ECTTIHSBYPASSx[7:0]	Input	Tied 0 or 1	Bypass handshaking between the CTI and subsystem (Trigger Interface). Tie HIGH when all the cross trigger blocks are in the same clock domain, clock skew is not a problem, and the source trigger is a least one clock cycle, to bypass the handshaking circuits.
ECTTISBYPASSACKx[7:0]	Input	Tied 0 or 1	Bypass synchronization with **ECTTRIGOUTACK** signals. Tie HIGH when all the CTI and subsystem are synchronous to bypass the synchronization circuits.
ECTTISBYPASSINx[7:0]	Input	Tied 0 or 1	Bypass synchronization on the **ECTTRIGIN** signals. Tie HIGH when the CTI and subsystem are synchronous to bypass the synchronization circuits.

A.4 ECT CTI subsystem signals

Table A-4 lists the signals for the CTI connected to the subsystem (Trigger Interface).

Table A-4 ECT CTI subsystem signals

Name	Type	Source/ destination	Description
ECTDBGENx	Input	System	Debug enable signal. When 0, the Cross Trigger Interface CI and TI are disabled (no trigger can be generated or received).
ECTTRIGINx[7:0]	Input	System	Input trigger.
ECTTRIGINACKx[7:0]	Output	System	Input trigger acknowledge.
ECTTRIGOUTx[7:0]	Output	System	Output trigger.
ECTTRIGOUTACKx[7:0]	Input	System	Output trigger acknowledge.

A.5 ECT channel interface signals

Table A-5 lists the signals for the ECT CTM connected to other ECTs (Channel Interface).

Table A-5 ECT channel interface signals

Name	Type	Source/ destination	Description
ECTCHINx[3:0]	Input	CTM	Trigger from the matrix
ECTCHINACKx[3:0]	Output	CTM	Trigger acknowledgement
ECTCHOUTx[3:0]	Output	CTM	Trigger request to the matrix
ECTCHOUTACKx[3:0]	Input	CTM	Trigger request acknowledgement

A.6 Scan test control signals

Table A-6 lists the internal scan test control signals.

Table A-6 Scan test control signals

Name	Type	Source/ destination	Description
SCANENABLE	Input	Scan controller	Scan enable, for all clock domains.
SCANINCTICLKx	Input	Scan controller	Scan data input for **ECTCTICLK** domain
SCANINCTMCLK	Input	Scan controller	Scan data input for **ECTCTMCLK** domain
SCANINHCLKx	Input	Scan controller	Scan data input for **HCLK** domain
SCANOUTCTICLKx	Output	Scan controller	Scan data output for **ECTCTICLK** domain
SCANOUTCTMCLK	Output	Scan controller	Scan data output for **ECTCTMCLK** domain
SCANOUTHCLKx	Output	Scan controller	Scan data output for **HCLK** domain

Appendix B
Timing Requirements

The timing requirements for the ECT interfaces are defined in this chapter. It contains the following sections:

B.1 AHB interface timing

Figure B-1 shows the AHB interface signals.

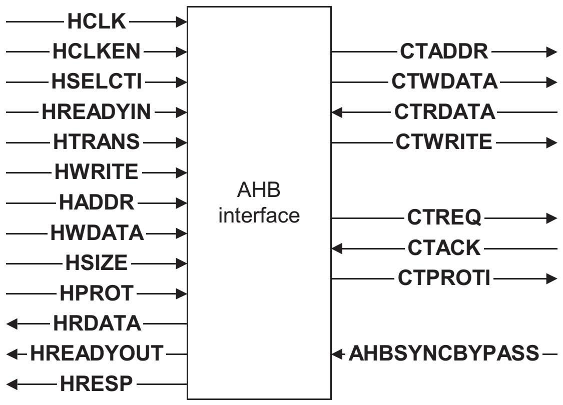

Figure B-1 AHB interface signals

Table B-1 lists the timing requirements for the AHB interface. All figures are expressed as a percentage of the **HCLK** period at maximum operating frequency.

Note

A 0% figure in the table indicates the hold times to clock edge plus the maximum clock skew for internal clock buffering.

Table B-1 AHB interface timing requirements

Parameter	Description	Max (ns)	Min (ns)
$T_{ovhdata}$	Rising **HCLK** to **HRDATA** valid	40%	-
$T_{ohhdata}$	**HRDATA** hold time from **HCLK** rising	-	>0%
T_{ovhcon}	Rising **HCLK** to AHB control outputs valid	40%	-
T_{ohhcon}	AHB control outputs hold time from **HCLK** rising	-	>0%
$T_{ishdata}$	AHB data inputs setup to rising **HCLK**	-	30%
$T_{ihhdata}$	AHB data inputs hold from rising **HCLK**	-	0%

Table B-1 AHB interface timing requirements (continued)

Parameter	Description	Max (ns)	Min (ns)
T_{ishcon}	AHB control inputs setup to rising **HCLK**	-	30%
T_{ihhcon}	AHB control inputs hold from rising **HCLK**	-	0%
$T_{ishresetn}$	**HRESETn** input setup to rising **HCLK**	-	50%
$T_{ihhresetn}$	**HRESETn** input hold from rising **HCLK**	-	5%

B.2 ECTCTMCLK domain timing

Figure B-2 shows **ECTCTMCLK** domain signals.

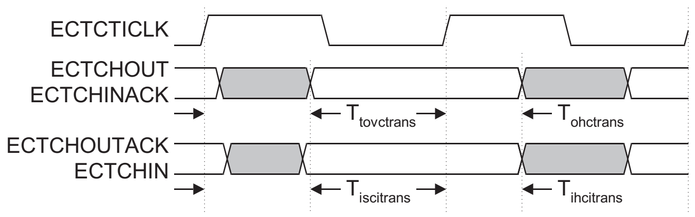

Figure B-2 ECTCTMCLK domain signals

Table B-2 lists the timing requirements for the **ECTCTMCLK** interface. All figures are expressed as a percentage of the **ECTCTMCLK** period at maximum operating frequency.

Note

A 0% figure in the table indicates the hold times to clock edge plus the maximum clock skew for internal clock buffering.

Bypass control signals for CTM must be static (tied HIGH or LOW).

Table B-2 ECTCTMCLK domain timing requirements

Parameter	Description	Max (ns)	Min (ns)
$T_{ovctrans}$	Rising **ECTCTMCLK** to **ECTCHINACK**	20%	-
$T_{ohctrans}$	**CLK** domain outputs hold time from **ECTCTMCLK** rising	-	>0%
$T_{iscitrans}$	Channel interface inputs setup to rising **ECTCTMCLK**	-	20%
$T_{ihcitrans}$	Channel interface inputs hold from rising **ECTCTMCLK**	-	0%
T_{oci2co}	Channel interface inputs to CI output	10%	-
T_{oci2to}	Channel interface inputs to TI output	40%	-
$T_{isctmresetn}$	**nCTMRESET** input setup to rising **HCLK**	-	50%
$T_{ihctmresetn}$	**nCTMRESET** input hold from rising **HCLK**	-	5%

B.3 ECTCTICLK domain timing

Figure B-3 shows **ECTCTICLK** domain signals.

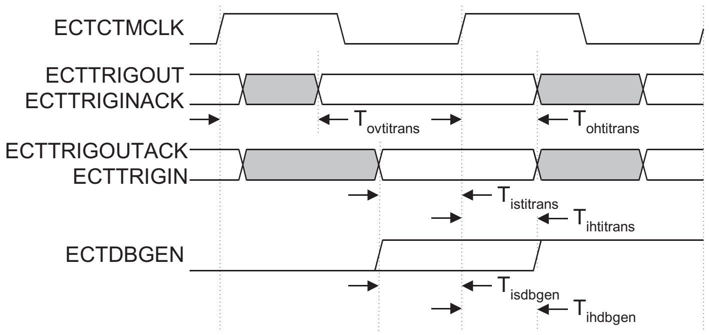

Figure B-3 ECTCTICLK domain signals

Table B-3 lists the timing requirements for the **ECTCTICLK** interface. All figures are expressed as a percentage of the **ECTCTICLK** period at maximum operating frequency.

Note

A 0% figure in the table indicates the hold times to clock edge plus the maximum clock skew for internal clock buffering.

Bypass control signals for CTI must be static (tied HIGH or LOW)

Table B-3 ECTCTICLK domain timing requirements

Parameter	Description	Max (ns)	Min (ns)
$T_{ivctrans}$	**ECTDBGEN** valid to rising **ECTCTICLK**	-	20%
$T_{ovctrans}$	Rising **ECTCTICLK** to **ECTTRIGINACK**	30%	-
$T_{ohctrans}$	**CLK** domain outputs hold time from **ECTCTICLK** rising	-	>0%
$T_{istitrans}$	Trigger interface inputs setup to rising **ECTCTICLK**	-	20%
$T_{ihtitrans}$	Trigger interface inputs hold from rising **ECTCTICLK**	-	0%
T_{oti2co}	Trigger interface inputs to channel interface output	40%	-

Table B-3 ECTCTICLK domain timing requirements (continued)

Parameter	Description	Max (ns)	Min (ns)
T_{oti2to}	Trigger interface inputs to Trigger interface output	75%	-
$T_{isctiresetn}$	**nCTIRESET** input setup to rising **HCLK**	-	50%
$T_{ihctiresetn}$	**nCTIRESET** input hold from rising **HCLK**	-	5%

Glossary

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

Advanced High-performance Bus (AHB)

The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture(AMBA)

AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB conforms to this standard.

See also Advanced High-performance Bus and AHB-Lite.

AHB *See* Advanced High-performance Bus.

AHB-Lite	AHB-Lite is a subset of the full AHB specification. It is intended for use in designs where only a single AHB master is used. This can be a simple single AHB master system or a multi-layer AHB system where there is only one AHB master on a layer.
Aligned	Refers to data items stored so that their address is divisible by the highest power of two that divides their size. Aligned words and halfwords therefore have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore refer to addresses that are divisible by four and two respectively. The terms byte-aligned and doubleword-aligned are defined similarly.
AMBA	*See* Advanced Microcontroller Bus Architecture.
Big-endian	Memory organization in which the least significant byte of a word is at a higher address than the most significant byte. *See also* Little-endian and Endianness.
Breakpoint	A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is halted unconditionally. Breakpoints are inserted by programmers to allow inspection of register contents, memory locations, and/or variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested. *See also* Watchpoint.
Burst	A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AHB buses are controlled using the **HBURST** signals to specify if transfers are single, four-beat, eight-beat, or 16-beat bursts, and to specify how the addresses are incremented.
Central Processing Unit (CPU)	The part of a processor that contains the ALU, the registers, and the instruction decode logic and control circuitry. Also commonly known as the processor core.
Clock gating	Gating a clock signal for a macrocell with a control signal, and using the modified clock that results to control the operating state of the macrocell.
CPU	*See* Central Processing Unit.
DBGTAP	*See* Debug Test Access Port.
Debugger	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging. An application that monitors and controls the operation of a second application. Usually used to find errors in the application program flow.

Debug Test Access Port (DBGTAP)	The collection of four mandatory terminals and one optional terminal that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **DBGTDI** (**TDI**), **DBGTDO** (**TDO**), **DBGTMS** (**TMS**), and **TCK**. The optional terminal is **DBGnTRST** (**TRST**).
EmbeddedICE logic	An on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface.
EmbeddedICE-RT	The JTAG-based hardware provided by debuggable ARM processors to aid debugging in real-time.
Embedded Trace Macrocell (ETM)	A hardware macrocell that outputs instruction and data trace information on a trace port.
Endianness	Byte ordering. The scheme that determines the order in which successive bytes of a data word are stored in memory. *See also* Little-endian and Big-endian.
ETM	*See* Embedded Trace Macrocell.
Exception	An event that occurs during program operation that makes continued normal operation inadvisable or impossible, and so makes it necessary to change the flow of control in a program. Exceptions can be caused by error conditions in hardware or software. The processor can respond to exceptions by running appropriate exception handler code that attempts to remedy the error condition, and either restarts normal execution or ends the program in a controlled way.
Halt mode	One of two mutually exclusive debug modes. In halt mode all processor execution halts when a breakpoint or watchpoint is encountered. All processor state, coprocessor state, memory and input/output locations can be examined and altered by the JTAG interface. *See also* Monitor mode.
Little-endian	Memory organization where the least significant byte of a word is at a lower address than the most significant byte. *See also* Big-endian and Endianness.
Macrocell	A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as an ARM processor, an Embedded Trace Macrocell, and a memory block) plus application-specific logic.
Processor	A contraction of microprocessor. A processor includes the CPU or core, plus additional components such as memory, and interfaces. These are combined as a single macrocell, that can be fabricated on an integrated circuit.

Register	A temporary storage location used to hold binary data until it is ready to be used.
Reserved	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as zero and are read as zero.
SBO	*See* Should Be One.
SBZ	*See* Should Be Zero.
SBZP	*See* Should Be Zero or Preserved.
Scan chain	A scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.
Should Be One (SBO)	Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.
Should Be Zero (SBZ)	Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.
Should Be Zero or Preserved (SBZP)	Should be written as 0 (or all 0s for bit fields) by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.
TAP	*See* Test Access Port.
Test Access Port (TAP)	The collection of four mandatory terminals and one optional terminal that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**.
Unaligned	Memory accesses that are not appropriately word-aligned or halfword-aligned. *See also* Aligned.
Undefined	Indicates an instruction that generates an Undefined instruction trap. See the *ARM Architectural Reference Manual* for more information on ARM exceptions.

Unpredictable	For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.
Watchpoint	A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to enable inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

Index