SECTION 23
READI MODULE

23.1 Overview

The READI module provides real-time development capabilities for RCPU-based MCUs in compliance with the IEEE-ISTO 5001 - 1999. This module provides development support capabilities for MCUs in single chip mode, without requiring address and data pins for internal visibility.

The auxiliary port, along with RCPU development features (such as background debug mode and watchpoints) supports all software and hardware development in single chip mode. The auxiliary port, along with (on-chip) calibration RAM, allows calibration variable acquisition and calibration constant tuning in single chip mode, for automotive powertrain development systems.

NOTE

In this section the bit numbering in the register definitions of tool mapped registers follows the Nexus IEEE-ISTO 5001 - 1999 bit numbering convention of MSB = Bit 31 and LSB = Bit 0, unlike the PowerPC standard MSB = Bit 0 and LSB = Bit 31. The bit description tables list the PowerPC bit numbering and Nexus bit numbering.

23.1.1 General Description

The READI module interfaces to the RCPU processor and internal buses to provide development support as per the IEEE-ISTO 5001 - 1999. The development features supported are program trace, data trace, watchpoint trace, ownership trace, run-time access to the MCU's internal memory map, and access to RCPU internal registers during halt, via the auxiliary port.

23.1.2 Feature Summary List

The READI module is compliant with Class 3 of the IEEE-ISTO 5001 - 1999. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code may be traced.
- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to (selected) internal memory resources. Data trace also allows for calibration variable acquisition in automotive powertrain development systems.
  — Two data trace windows with programmable address range and access at-

tributes. Data trace windowing reduces the requirements on the auxiliary port bandwidth by constraining the number of trace locations.

- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted to indicate when a new process/task is activated, allowing development tools to trace process/task flow.
- Run-time access to on-chip memory map and PowerPC™ special purpose registers (SPRs) via the READI read/write access protocol. This feature supports accesses for runtime internal visibility, calibration constant acquisition and tuning, and external rapid prototyping for powertrain automotive development systems.
- Watchpoint messaging via the auxiliary port
- Nine or 16 full-duplex auxiliary pin interface for medium and high visibility throughput
  - — One of two modes selected during reset: full port mode (FPM) and reduced port mode (RPM).
    - FPM comprises 16 pins and RPM comprises nine pins
  - — Auxiliary output port
    - One MCKO (message clock out) pin
    - Two or eight MDO (message data out) pins
    - One $\overline{\text{MSEO}}$ (message start/end out) pin
  - — Auxiliary input port
    - One $\overline{\text{MCKI}}$ (message clock in) pin
    - One or two MDI (message data in) pins
    - One $\overline{\text{MSEI}}$ (message start/end in) pin
    - One $\overline{\text{EVTI}}$ (event in) pin
    - One $\overline{\text{RSTI}}$ (reset in) pin
- All features configurable and controllable via the auxiliary port
- Security features for production environment
- Support of existing RCPU development access protocol via the auxiliary port
- READI module can be reset independent of system reset
- Parametrics:
  - — Two bits are downloaded per clock in full port mode. For example, with input clock running at 28 MHz, this translates to a download rate of 56 Mbits/s.
  - — One bit is downloaded per clock in reduced port mode. For example, with input clock running at 28 MHz, this translates to a download rate of 28 Mbits/s.
  - — Eight bits are uploaded per clock in full port mode. For example, with system clock running at 56 MHz, this translates to a upload rate of 448 Mbits/s.
  - — Two bits are uploaded per clock in reduced port mode. For example, with system clock running at 56 MHz, this translates to a upload rate of 112 Mbits/s.

23.1.3 Functional Block Diagram

The functional block diagram of the READI module is shown in **Figure 23-1**.

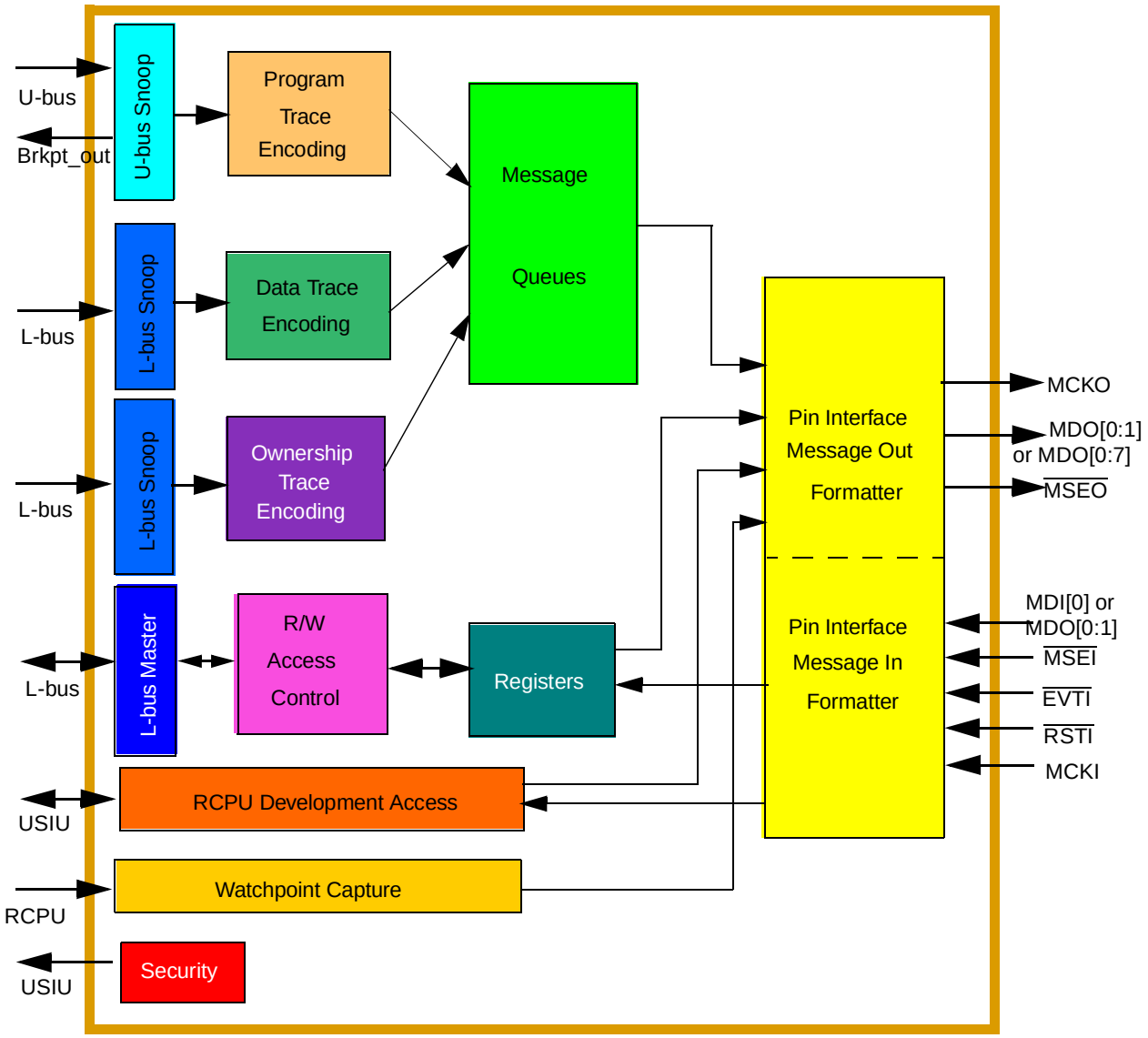


Figure 23-1 READI Functional Block Diagram

23.1.4 Modes of Operation

The various operating modes of the READI module are:

1. Reset
2. Secure
3. Normal
4. Disabled

23.1.4.1 READI Reset Configuration

The READI reset configuration information is received via $\overline{\text{EVTI}}$ and MDI[0] to enable or disable the READI module, and select the port size. $\overline{\text{EVTI}}$ and MDI[0] are sampled synchronously at the negation of $\overline{\text{RSTI}}$. Reset configuration information must be valid on $\overline{\text{EVTI}}$ and MDI[0] at least four clocks prior to the negation of $\overline{\text{RSTI}}$.

The following table describes the READI reset configuration options.

Table 23-1 READI Reset Configuration Options

$\overline{\text{EVTI}}$	MDI [0]	Configuration
1	X	Module Disabled. All outputs three-stated.
0	1	Module Enabled, Full Port Configuration 2 MDI, 8 MDO
0	0	Module Enabled, Reduced Port Configuration 1 MDI, 2 MDO

If $\overline{\text{EVTI}}$ is asserted at negation of $\overline{\text{RSTI}}$, the READI module will be enabled.

The READI module will three-state the auxiliary output port when $\overline{\text{RSTI}}$ is asserted.

23.1.4.2 Security

Security is provided via the UC3F censorship mechanism. If a UC3F array is in censored mode, reads or writes to the UC3F will not be allowed (RCPU will not be able to fetch instructions from the UC3F) once any of the following cases are detected:

- Program trace and/or data trace are enabled
- Read/write access is attempted (can be to any address location)
- RCPU development access is enabled.

23.1.4.3 Disabled

If $\overline{\text{EVTI}}$ is negated at negation of $\overline{\text{RSTI}}$, the READI module will be disabled. No trace output will be provided, and output auxiliary port will be three-stated. Any message sent by the tool is ignored.

23.1.5 Parametrics

With 16 message queues, throughput numbers were calculated for the following benchmark codes [assuming full port mode]:

- For an example benchmark which had 10.9% direct branches, 2.5% indirect branches, 10.4% data writes, and 19.3% data reads, approximately 20% of total data trace accesses will be traced.
- For another example benchmark which had 9.8% direct branches, 2.8% indirect branches, 6.6% data writes, and 18.3% data reads, approximately 27% of total data trace accesses will be traced.

For reduced port mode, the data trace feature should not be used, or used sparingly, so as not to cause queue overruns.

23.1.6 Programmer's Model

The READI registers do not follow the recommendations of the IEEE-ISTO 5001 - 1999, but are loosely based on the 0.9 release of the standard.

READI registers are classified into two categories: user mapped register and tool mapped registers.

User mapped register (memory mapped register):

- Ownership trace register

Tool mapped registers (registers which can be accessed only through the development tool and are not memory mapped):

- Device ID register
- Development control register
- User base address register
- Read/write access register
- Upload/download information register
- Data trace attributes register 1
- Data trace attributes register 2

23.1.7 Messages

The READI module implements messaging via the auxiliary port according to the IEEE-ISTO 5001 - 1999. Messaging will be implemented via transfer codes (TCODEs) on the auxiliary port. The TCODE number for the message identifies the transfer format (the number and/or size of packets to be transferred) and the purpose of each packet.

Public messages outlined in **Table 23-2** are supported by READI.

Table 23-2 Public Messages

TCODE Number	Message Name
1	Device ID
2	Ownership Trace Message
3	Program Trace — Direct Branch Message
4	Program Trace — Indirect Branch Message
5	Data Trace — Data Write Message
6	Data Trace — Data Read Message
8	Error Message
10	Program Trace Correction
11	Program Trace — Direct Branch Synchronization Message
12	Program Trace — Indirect Branch Synchronization Message

Table 23-2 Public Messages (Continued)

TCODE Number	Message Name
13	Data Trace — Data Write Synchronization Message
14	Data Trace — Data Read Synchronization Message
15	Watchpoint Message
16	Auxiliary Access — Device Ready for Upload/Download Message
17	Auxiliary Access — Upload Request (Tool Requests Information) Message
18	Auxiliary Access — Download Request (Tool Provides Information) Message
19	Auxiliary Access — Upload/Download Information (Device/Tool Provides Information) Message

Table 23-3 Vendor-Defined Messages

TCODE Number	Message Name
56	RCPU Development Access — DSDI Data (Tool Provides Information) Message
57	RCPU Development Access — DSDO Data (Device Provides Information) Message
58	RCPU Development Access — BDM Status (Device Provides Information) Message
59	Program Trace — Indirect Branch Message With Compressed Code
60	Program Trace — Direct Branch Synchronization Message With Compressed Code
61	Program Trace — Indirect Branch Synchronization Message With Compressed Code

Vendor-defined messages outlined in **Table 23-3** are also supported by READI.

23.1.8 Terms and Definitions

Table 23-4 Terms and Definitions

Term	Description
Auxiliary Port	Refers to IEEE-ISTO 5001 auxiliary port.
Branch Trace Messaging (BTM)	External visibility of addresses for taken branches and exceptions, and the number of sequential instructions executed between each taken branch.
BDM	Background Debug Mode.
Compressed Code Mode	Current instruction stream is fetching compressed code.
Calibration Constants	Performance related constants which must be tuned for automotive powertrain and disk drive applications.
Calibration Variables	Intermediate calculations which must be visible during the calibration or tuning process to enable accurate tuning of calibration constants.
Data Read Message (DRM)	External visibility of data reads to internal memory-mapped resources.
Data Write Message (DWM)	External visibility of data writes to internal memory-mapped resources.
Data Trace Messaging (DTM)	External visibility of how data flows through the embedded system. May include DRM and/or DWM.
Download	Tool sends information to the device

Table 23-4 Terms and Definitions (Continued)

Term	Description
Field	Number of bits representing single piece of information
FPM	Full Port Mode. This is the default port mode for READI.
IEEE-ISTO 5001	IEEE-ISTO 5001, formerly known as Global Embedded Processor Debug Interface Standard. World Wide Web documentation at ***http://www.nexus-standard.org***.
Halt	RCPU is in freeze state (typically in debug mode)
Instruction Fetch	The process of reading the instruction data received from the instruction memory.
Instruction Issue	The process of driving valid instruction bits inside the processor. The instruction is decoded by each execution unit, and the appropriate execution unit prepares to execute the instruction during the next clock cycle.
Instruction Taken	An instruction is taken after it has been issued and recognized by the appropriate execution unit. All resources to perform the instruction are ready, and the processor begins to execute it.
Instruction Retire	Completion of the instruction issue, execution and writeback stages. An instruction is ready to be retired if it completes without generating an exception and all instructions ahead of it in history buffer have completed without generating an exception.
ICTRL	Instruction bus Support Control Register.
Ownership Trace Message (OTM)	Visibility of process/function that is currently executing.
Public Messages	Messages on the auxiliary pins for accomplishing common visibility and controllability requirements e.g. DRM and DWM.
RCPU	PowerPC based processor used in the Motorola MPC500 family of microcontrollers.
READI	Real time Embedded Applications Development Interface.
READI pins	Refers to IEEE-ISTO 5001 auxiliary port.
RPM	Reduced Port Mode. This is the alternate port mode for READI.
run-time	RCPU is executing program code in normal mode
Sequential Instruction	Any instruction other than a flow-control instruction or **isync**.
Snooping	Monitoring addresses driven by a bus master to detect the need for coherency actions.
Standard	The phrase “according to the standard” implies according the IEEE-ISTO 5001 - 1999.
Superfield	One or more message “fields” delimited by $\overline{\text{MSEO}}$/$\overline{\text{MSEI}}$ assertion/negation. The information transmitted between “start-message” and “end-packet” states.
Show Cycle	An internal access (e.g., to an internal memory) reflected on the external bus using a special cycle (marked with a dedicated transfer code). For an internal memory “hit,” an address-only bus cycle is generated; for an internal memory “miss,” a complete bus cycle is generated.
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
TCK_DSCK	Multiplexed pin: JTAG Clock or Development Port Clock.
TDI_DSDI	Multiplexed pin: JTAG Data In or Development Port Serial Data In.
TDO_DSDO	Multiplexed pin: JTAG Data Out or Development Port Serial Data Out.
Upload	Device sends information to the tool.
VSYNC	Internal RCPU signal
VF	Internal RCPU signal which indicates instruction queue status.
VFLS	Internal RCPU signal which indicates history buffer flush status.

23.2 Programmer's Model

This section describes the READI programmer's model. READI resources can only be configured and accessed via the auxiliary port.

The READI registers do not follow the recommendations of the IEEE-ISTO 5001 - 1999, but are loosely based on the 0.9 release of the standard.

23.2.1 Register Map

READI registers are accessible via the *auxiliary* port. They can be classified into two categories: user mapped registers and tool mapped registers.

23.2.1.1 User Mapped Register

The operating system writes the ID for the current task/process in the OTR register. **Table 23-5** shows the location of the register bits. Their functions are explained below.

The current task/process (CTP) field is updated by the operating system software to provide task/process ID information. The OT register can only be accessed by supervisor data attributes. Only CPU writes to this register will be transmitted out. This register is not accessible via the auxiliary port download request message.

READI_OTR — READI Ownership Trace Register **0x38 002C**

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Current Task Process (CTP)															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	LSB 31
Current Task Process (CTP)															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 23-5 READI_OTR Bit Descriptions

Bit(s)	Name	Description
31:0	CTP	READI ownership trace register, write only.

23.2.1.2 Tool Mapped Registers

Table 23-6 defines READI registers which are not memory mapped and can only be accessed through the development tool. Their corresponding access opcodes are also defined

Table 23-6 Tool Mapped Register Space

Access Opcode	Register	Access Type
0x08 (8)	Device ID Register (DID)	Read Only
0x0A (10)	Development Control Register (DC)	Read/Write
0x0D (13)	User Base Address Register (UBA)	Read Only
0x0F (15)	Read/Write Access Register (RWA)	Read/Write
0x10 (16)	Upload/Download Information Register (UDI)	Read/Write
0x14 (20)	Data Trace Attributes Register 1 (DTA1)	Read/Write
0x15 (21)	Data Trace Attributes Register 2 (DTA2)	Read/Write

23.2.1.3 Device ID (DID) Register

Accessing the DID register provides key attributes to the development tool concerning the MCU. This information is also transmitted via the auxiliary output port upon exit of READI reset ($\overline{\text{RSTI}}$), if $\overline{\text{EVTI}}$ is asserted at $\overline{\text{RSTI}}$ negation. Table 23-7 gives the bit descriptions.

READI_DID — READI Device ID Register **0x08**

MSB 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV				MDC						PN					
HRESET:															
—	—	—	—	0	0	0	0	1	0	0	0	0	0	1	1

LSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
PN				MID											RE-SERVED
HRESET:															
0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1

Table 23-7 READI_DID Bit Descriptions

RCPU Bit(s)	Nexus Bit(s)	Name	Description
0:3	31:28	REV	READI version number. This field contains the revision level of the device.
4:9	27:22	MDC	READI Manufacturer Design Center. This field identifies the manufacturer's design center. The MPC565 has a value of 0x02.
10:19	21:12	PN	READI Part Number. This part number identification field. The MPC565 is 0x033.
20:30	11:1	MID	READI Manufacturer ID. This field identifies the manufacturer of the device, Motorola's ID is 0x1C.
31	0	—	Reserved

23.2.1.4 Development Control (DC) Register

The DC register is used for basic development control of the READI module. **Table 23-8** shows the location of register bits.

READI_DC — READI Deveopment Control (DC) Register **0x0A**

MSB 7	6	5	4	3	2	1	LSB 0
DOR	DME	DPA	TM			EC	
HARD RESET:							
0	0	0	0	0	0	0	0

Table 23-8 READI_DC Bit Descriptions

RCPU Bit(s)	Nexus Bit(s)	Name	Description
0	7	DOR[1]	READI Debug Mode Entry Out-of-reset Field can be configured to enable or disable debug mode entry out of reset. 0 =Debug Mode Not Entered Out-of-Reset 1 = Debug Mode Entered Out-of-Reset
1	6	DME[1]	READI Debug Mode Enable Field can be configured to enable or disable debug mode. 0 = Debug Mode Disabled 1 = Debug Mode Enabled
2	5	DPA[1]	READI RCPU Development Access Field can be configured to access the RCPU Development features via the READI module or via the debug pins (TCK_DSCK, TDI_DSDI and TDO_DSDO). The default DPA setting configures the RCPU development features to be accessed via debug pins. 0 = RCPU Development Access Disabled through READI, BDM pins enabled 1 = RCPU Development Access Enabled through READI

Table 23-8 READI_DC Bit Descriptions (Continued)

RCPU Bit(s)	Nexus Bit(s)	Name	Description
3:5	4:2	TM	READI Trace Mode Field can be configured to enable BTM, DTM and OTM. Any or all types of trace may be enabled. 000 = No Trace 1xx = BTM Branch Trace Messaging Enabled x1x = DTM Data Trace Messaging Enabled xx1 = OTM Ownership Trace Messaging Enabled
6:7	1:0	EC	READI $\overline{\text{EVTI}}$ Control Field can be configured for synchronization and breakpoint generation. If the EC is equal to 0b00, asserting $\overline{\text{EVTI}}$ will cause the next program and data trace message to be a synchronization message (providing program and data trace are enabled). If the EC field is equal to 0b01, a breakpoint will be generated. If the field is configured to one of the reserved states, its action reverts to that of the default state. **NOTE**: The $\overline{\text{EVTI}}$ pin is level sensitive when EC is configured for breakpoint generation. This implies that as long as $\overline{\text{EVTI}}$ assertion is continued (with EC set to 0b01), the READI module will continue requesting a breakpoint. The user must detect breakpoint generation and negate the $\overline{\text{EVTI}}$ pin appropriately. 00 = $\overline{\text{EVTI}}$ for program and data trace synchronization 01 = $\overline{\text{EVTI}}$ for breakpoint generation 1x = Reserved

NOTES:

1. The DOR, DME, and DPA fields in the DC register can only be modified when system reset is asserted, or reset (to default state) when the READI module is reset by the assertion of $\overline{\text{RSTI}}$.

23.2.1.5 RCPU Development Access Modes

Table 23-9 describes the allowed modes for RCPU development access.

Table 23-9 RCPU Development Access Modes

DPA	DME	DOR	RCPU Development Access through READI
0	x	x	No RCPU Development Access via READI. RCPU access is done through BDM pins.
1	0	x	Non-debug mode access of RCPU development through READI.
1	1	0	Debug mode is enabled through READI (RCPU is still in normal mode, out of reset)
1	1	1	Debug mode is enabled through READI and entered out-of-reset. Debug mode entry causes RCPU to halt.

23.2.1.6 User Base Address (UBA) Register

The UBA register defines the memory map address for the OT register. **Table 23-10** gives a description of the register bits.

READI_UBA — READI User Base Address Register **0x0D**

MSB 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UBA															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
UBA															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0

Table 23-10 READI_UBA Bit Descriptions

RCPU Bit(s)	Nexus Bit(s)	Name	Description
0:31	31:0	UBA	The user base address (UBA) field defines the memory map address for the OT register. The MPC565 user base address is 0x38002C. The UBA register is read-only by the development tool.

23.2.1.7 Read/Write Access (RWA) Register

The RWA register provides DMA-like access to memory mapped locations, PowerPC special purpose registers, and READI tool mapped registers. **Table 23-11** shows the location of register bits.

READI_RWA — READI Read/Write Access Register **0x0F**

MSB 79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
SC	RWAD														
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
RWAD										RW	SZ		WD		
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
WD															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WD													PRV		MAP
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
CNT															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 23-11 READI_RWA Read/Write Access Bit Descriptions

RCPU Bit(s)	Nexus Bit(s)	Name	Description
0	79	SC	The start complete (SC) field is set when a read or write access is initiated. The device will clear the SC bit once the read or write access completes. During a block access, if the SC bit is reset, the access will terminate. 0 = Access complete 1 = Start access
1:25	78:54	RWAD	Read/write address (RWAD) bits are used to identify the address of internal memory-mapped resources to be accessed, or the lowest address (i.e., lowest unsigned value) for a block move (CNT > 0). The address range for a block move is from RWAD to RWAD + CNT. **NOTE:** The RWD field of the UDI register is shared with the WD field of the RWA register.
26	53	RW	The read/write (RW) field can be configured to allow selection of a read or a write access. 0 = Read access 1 = Write access
27:28	52:51	SZ	The word size (SZ) field can be configured to allow 32-bit, 16-bit, or 8-bit read/write accesses. If the field is configured to one of the reserved states, its action reverts to that of the default state. 00 = 32-bit 01 = 16-bit 10 = 8=bit 11 = Reserved
29:60	50:19	WD	Write data (WD) bits contain the data to be written. For a read access, the data stored is a don't care.
61:62	18:17	PRV	The Privilege Attribute Field can be configured to select different read/write access attributes. 00 = User Data 01 = User Instruction 10 = Supervisor Data 11 = Supervisor Instruction
63	16	MAP	The Map Select Field can be configured to allow access to multiple memory maps. The primary processor memory map (MAP equal to 0b0) is designated as the default. The secondary memory map (MAP equal to 0b1) can be set to select the PowerPC special purpose registers. 0 = Primary memory map 1 = Secondary memory map (PPC Special Purpose Registers)
64:79	15:0	CNT	The Access Count Field can be configured to indicate the number of accesses of word size (defined in SZ field). The CNT value is used to increment the specified address in the RWAD field for block read/write accesses. For a single read/write access, the CNT value should equal to 0x0000. A 64-Kbyte block read/write access can be performed by configuring the CNT bits as 0xFFFF. If a user wants to terminate a block read or write access which has not completed, the CNT bits should be reset.

23.2.1.8 Upload/Download Information (UDI) Register

The UDI register is used to store the data to be written for block write access, and the data read for read (single and block) accesses. **Table 23-12** gives a description of the register bits.

READI_UDI — READI Upload/Download Information Register **0x10**

MSB 33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
RWD																
HARD RESET:																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
RWD															ERR	DV
HARD RESET:																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 23-12 READI_UDI Bit Descriptions

RCPU Bit(s)	Nexus Bit(s)	Name	Description
0:31	33:2	RWD	The Read/Write Data Field is used to store data for read accesses and block write accesses. It can contain three sizes of data. Refer to **Table 23-13**, **Table 23-14** and **Table 23-15** for details.
32	1	ERR	The Error Field is used to determine the status of the read or write access. Refer to **Table 23-13** and **Table 23-14** for details. 0 = Read or write access has not been completed.. 1 = Read or write access has completed. **NOTE**: The ERR field is read-only.
33	0	DV	The Data Valid Field is used to determine the status of the read or write access. Refer to **Table 23-13** and **Table 23-14** for details. 0 = No error has occurred. 1 = Access error occurred. **NOTE**: The DV field is read-only.

Table 23-13 Read Access Status

ERR	DV	Status
0	0	Read access has not yet completed
0	1	Read access has completed and no access error occurred
1	0	Access error occurred
1	1	Not allowed

Table 23-14 Write Access Status

ERR	DV	Status
0	0	Write access has completed and no access error occurred
1	0	Write access error occurred (Error Message sent out)
0	1	Write access has not yet completed
1	1	Not allowed

Table 23-15 RWD Field Configuration

						LSB
8 bit	Reserved – Read as Zeros			LS Byte	ERR	DV
16 bit	Reserved – Read as Zeros		MS Byte	LS Byte	ERR	DV
32 bit	MS Byte			LS Byte	ERR	DV

NOTE

The RWD field of the UDI register is shared with the WD field of the RWA register.

23.2.1.9 Data Trace Attributes 1 and 2 (DTA1 and DTA2) Registers

The DTA1 and DTA2 registers allow data trace messaging (DTM) to be restricted to reads, writes or both for a user programmable address range. Two DTA registers allow two address ranges to be selected for DTM. Refer to **Table 23-16** for register bit descriptions.

READI_DTA 1 — READI Data Trace Attributes 1 Register **0x14**
READI_DTA 2 — READI Data Trace Attributes 2 Register **0x15**

MSB 47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
DTEA															
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DTEA							DTSA								
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
DTSA														TA	
HARD RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 23-16 READI_DTA 1 AND 2 Bit Descriptions

RCPU Bit(s)	Nexus Bit(s)	Name	Description
0:22	47:25	DTEA[1]	The Read/Write End Field defines the end address for the address range. Refer to **Table 23-30**.
23:45	24:2	DTSA[1]	The Read/Write Start Field defines the starting address for the address range. Refer to **Table 23-30**.
46:47	1:0	TA	The Read/Write Trace Field can be configured to allow enabling or disabling data read and/or data write traces. 00 = Disable data read and data write trace x1 = Enable data read trace 1x = Enable data write trace

NOTES:
1. Data trace range start and end addresses must be word-aligned.

Table 23-17 Data Trace Values

Programmed Values	Range Selected
DTSA < DTEA	DTSA → ← DTEA
DTSA > DTEA	Invalid Range
DTSA = DTEA	Word at DTSA

NOTE

There is no way to distinguish between off-core PowerPC special purpose register (SPR) map and normal memory map accesses via the defined address range control. If data trace ranges are set up such that the off-core PowerPC SPR map falls within active ranges, then accesses to these off-core PowerPC SPRs will be traced, and the messages will not be distinguishable from accesses to normal memory map space. Off-core PowerPC SPRs typically exist in the 8- to 16-Kbyte lowest memory block (0x2000 – 0x3FF0). If data or peripherals are mapped to this space, load/stores to PowerPC SPRs will be indistinguishable from data or peripheral accesses.

23.2.2 Accessing Memory Mapped Locations Via the Auxiliary Port

The control and status information is accessed via the four auxiliary access public messages: device ready for upload/download, upload request (tool requests information), download request (tool provides information), and upload/download information (device/tool provides information).

To write control or status to memory mapped locations the following sequence would be required.

1. The tool confirms that the device is ready (so as to not cancel an ongoing read

write access). The tool transmits the download request public message (TCODE 18) which contains write attributes, write data, and target address.

2. The tool waits for device ready for upload/download (TCODE 16) message before initiating next access.

To read control or status from memory mapped locations the following sequence would be required.

1. The tool confirms that the device is ready. The tool transmits the download request public message (TCODE 18) which contains read attributes and target address.
2. When device reads data it transmits upload/download information message (TCODE 19) containing read data. Device is now ready for next access.

For a block write to memory mapped locations the following sequence would be required.

1. The tool confirms that the device is ready. The tool transmits the download request public message (TCODE 18) which contains block write attributes, first write data, and target address.
2. The tool waits for device ready for upload/download message (TCODE 16). When it is transmitted by device, tool transmits upload/download information message (TCODE 19) containing next write data. This step is repeated until all data is written

For a block read from memory mapped locations the following sequence would be required.

1. The tool confirms that the device is ready. The tool transmits the download request public message (TCODE 18) which contains block read attributes and target address.
2. The tool waits for upload/download information message (TCODE 19) from device, which contains read data. This step is repeated until all data is read.

Refer to **23.6 Read/Write Access** for more details on read/write access protocol.

23.2.3 Accessing READI Tool Mapped Registers Via the Auxiliary Port

To write control or status data to READI tool mapped registers the following sequence would be required.

1. The tool confirms that the device is ready. The tool transmits the download request message (TCODE 18) which contains write data, and register opcode.
2. The tool waits for device ready for upload/download message (TCODE 16) before initiating next access.

To read control or status from READI tool mapped registers the following sequence would be required

1. The tool confirms that the device is ready. The tool transmits the upload request message (TCODE 17) which contains the target opcode.

2. When device reads data it transmits upload/download information message (TCODE 19) containing read data. Device is now ready for next access.

Refer to **23.6 Read/Write Access** for more details on read/write access protocol.

23.2.4 Partial Register Updates

Registers may be updated via the auxiliary port using the download request message with the message containing only **N** (where **N** is less than register width) most-significant bits of the register. In such cases the bits not transmitted will be reset to 0b0. The bits transmitted will be aligned such that the last bit transmitted will be the most significant bit of the register. Therefore a message size that is divisible by the input port size should be transmitted.

23.2.5 Programming Considerations

The following programing guidelines are recommended for users of the READI features.

23.2.5.1 Program Trace Guidelines

Program trace via BTM is not supported during BDM.

For program trace synchronization to work, the ICTRL (refer to **Table 23-21**) register must be programmed such that show cycle will be performed for all changes in the program flow (ISCTL field = 01).

NOTE

The user must program the ICTRL for change of flow show cycles early in the reset vector, before any branches, otherwise trace is not guaranteed.

If BDM is enabled, the ICTRL register cannot be modified through the program and can only be modified through RCPU development access.

It is also recommended that the USIU be programmed to ignore instruction show cycles (so as to not impact U-bus performance).

To correctly trace program execution using BTM, the READI module must be enabled prior to release of system reset. If the READI module is enabled ($\overline{\text{EVTI}}$ asserted, $\overline{\text{RSTI}}$ negated) after the RCPU has started execution of the program, the trace cannot be guaranteed.

Refer to **Figure 23-2** for further details.

23.2.5.2 Compressed Code Mode Guidelines

The ICTRL register must be programmed such that a show cycle will be performed for all changes in the program flow (ISCTL field = 01). (See **Table 22-21**.)

The BBC must be enabled to display data on instruction show cycles. BBCMCR[DECOMP_SC_EN] (refer to **4.8.2.1 BBC Module Configuration Register**

BBCMCR) must be set when decompression is enabled. This will allow READI to track the compressed code. BBCMCR[DECOMP_SC_EN] should not be set if there is no intention to use compressed code, as it will degrade U-bus performance.

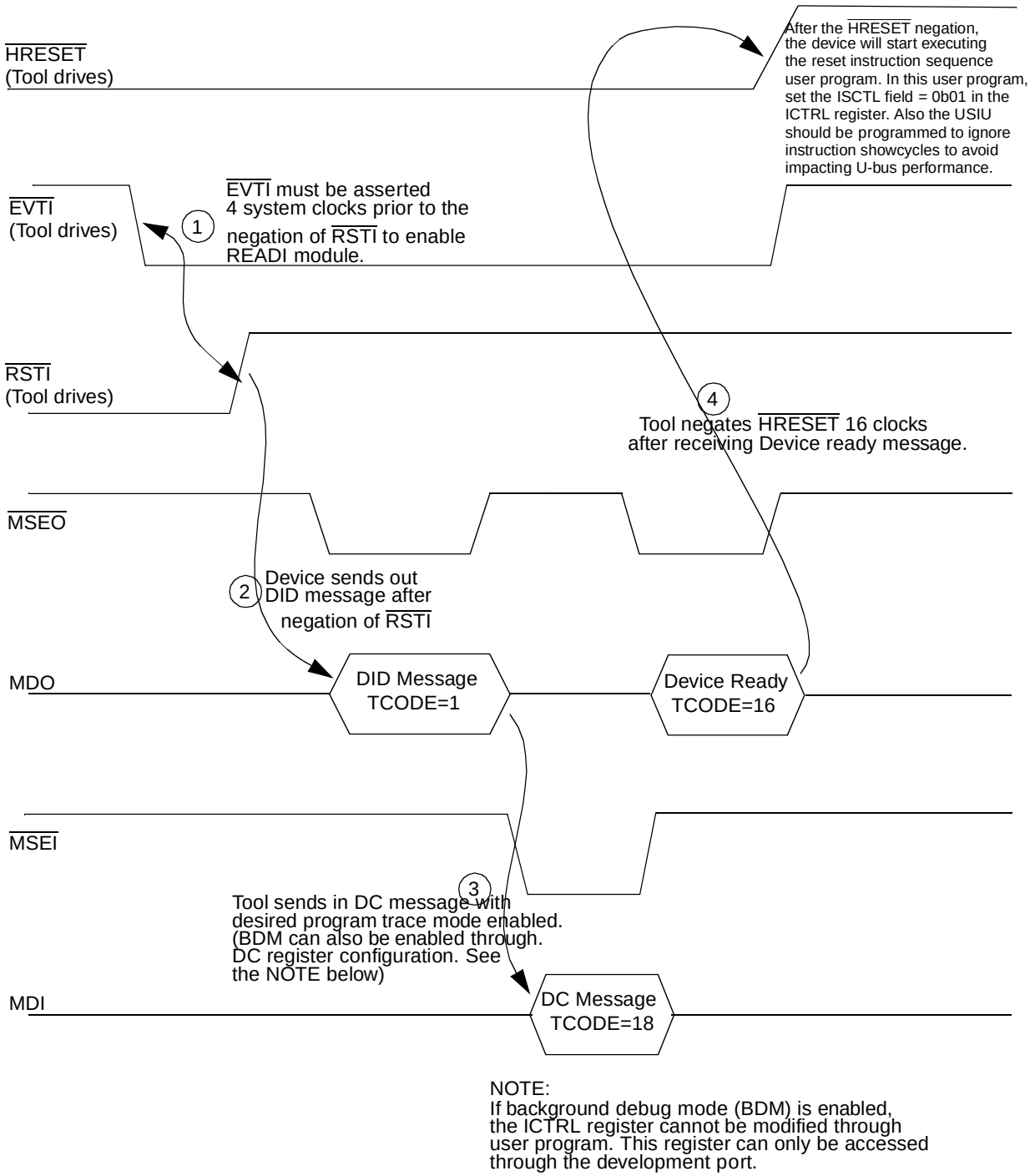


Figure 23-2 Enabling Program Trace out of System Reset

23.2.5.3 Reset Sequence Guidelines

Refer to **23.3.3 READI Reset Configuration** for further details.

23.2.5.4 BDM Guidelines

Refer to **23.3.3.1 Reset Configuration for Debug Mode** for further details.

23.3 Pin Interface

This section details information regarding the READI pins and pin protocol.

23.3.1 Functional Description

The READI pin interface provides the function of transmitting messages from the message queues to the external tools. The pin interface also provides the control for timing and logic for formatting the messages.

23.3.1.1 Pins Implemented

The READI module implements one MCKO, MCKI, $\overline{\text{EVTI}}$, $\overline{\text{RSTI}}$, $\overline{\text{MSEO}}$ and $\overline{\text{MSEI}}$ pin. It also implements 1 or 2 MDI and 2 or 8 MDO pins. The input pins are synchronized to MCKI input clock and the output pins are synchronized to free running MCKO output clock.The READI pin definition is outlined in **Table 23-18**.

NOTE

MCKI clock frequency has to be less than or equal to one half of MCKO clock frequency.

Table 23-18 Description of READI Pins

IEEE-ISTO 5001 Pin Name	Input/ Output	Description of Pin
MCKO	Output	Message Clock-Out (MCKO) is a free-running output clock to development tools for timing of MDO and $\overline{\text{MSEO}}$ pin functions. MCKO is the same as the MCU system clock.
MDO[7:0] or MDO[1:0]	Output	Message Data Out (MDO[7:0] or MDO[1:0]) are output pins used for uploading OTM, BTM, DTM, and Read/Write Accesses. External latching of MDO will occur on rising edge of MCKO. 8 pins are implemented.
$\overline{\text{MSEO}}$	Output	Message Start/End Out ($\overline{\text{MSEO}}$) is an output pin which indicates when a message on the MDO pins has started, when a variable length packet has ended, and when the message has ended. 1 $\overline{\text{MSEO}}$ pin is implemented. External latching of $\overline{\text{MSEO}}$ will occur on rising edge of MCKO.
MCKI	Input	Message Clock-In (MCKI) is a input clock from development tools for timing of MDI and $\overline{\text{MSEI}}$ pin functions. MCKI frequency has to be less than or equal to one half of MCKO frequency.
MDI[1:0] or MDI[0]	Input	Message Data In (MDI[1:0] or MDI[0]) are input pins used for downloading configuration information, writes to user resources, etc. Internal latching of MDI will occur on rising edge of MCKI. 2 pins are implemented on the MPC565.

Table 23-18 Description of READI Pins (Continued)

IEEE-ISTO 5001 Pin Name	Input/ Output	Description of Pin
$\overline{\text{MSEI}}$	Input	Message Start/End In ($\overline{\text{MSEI}}$) is an input pin which indicates when a message on the MDI pins has started, when a variable length packet has ended, and when the message has ended. 1 $\overline{\text{MSEI}}$ pin is implemented. Internal latching of $\overline{\text{MSEI}}$ will occur on rising edge of MCKI.
$\overline{\text{EVTI}}$	Input	Event In ($\overline{\text{EVTI}}$) — The $\overline{\text{EVTI}}$ pin is level sensitive when configured for breakpoint generation, otherwise it is edge sensitive.
$\overline{\text{RSTI}}$	Input	Reset In ($\overline{\text{RSTI}}$).

23.3.2 Functional Block Diagram

Figure 23-3 depicts the functional block diagram of the pin interface.

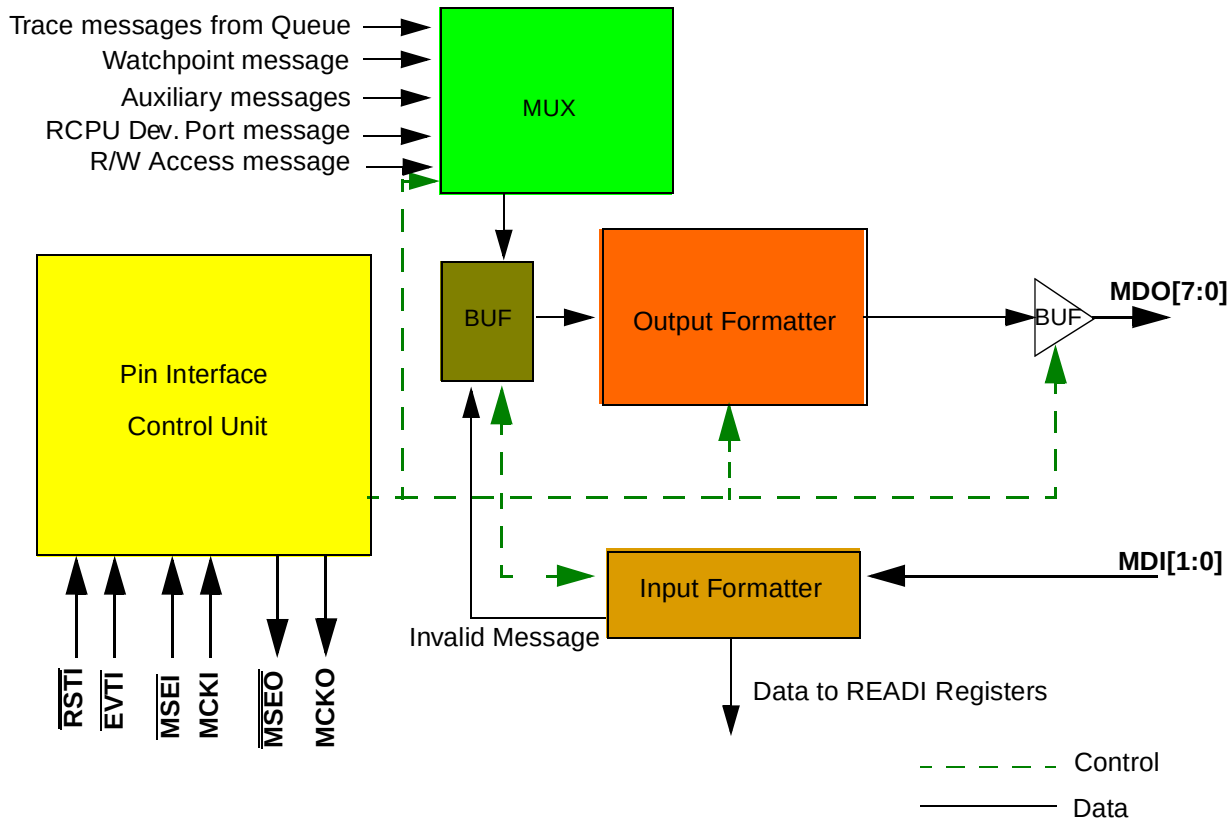


Figure 23-3 Functional Diagram of Pin Interface

The pin interface is responsible for handshaking with the message queue and registers. It is also responsible for requesting new messages from the message queue. A message is always requested from the message queue if the message queue is not empty, the message buffer is available and a higher priority message is not requesting to be transmitted. The rate at which data is removed from the queue depends on the average message length, the number of MDO pins, and the MCKO clocking rate.

23.3.2.1 Message Priority

Message formatting is performed in the pin interface block. The following priority scheme is implemented for messages sent to the pin output formatter block, with 1 being the highest priority and 5 being the lowest priority:

1. Invalid message
2. READI register access handshakes (device ready/download information)
3. Watchpoint messages
4. Read/write access message
5. RCPU development access message
6. Queued messages (program trace, data trace, and ownership trace)

23.3.2.2 Pin Protocol

The protocol for the MCU receiving and transmitting messages via the auxiliary pins will be accomplished with the $\overline{\text{MSEI}}$ and $\overline{\text{MSEO}}$ pin functions respectively. The $\overline{\text{MSEI}}$ pin will provide the protocol for the MCU receiving messages, and the $\overline{\text{MSEO}}$ pin will provide the protocol for the MCU transmitting messages.

The $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ protocol is illustrated in **Table 23-19**.

Table 23-19 $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ protocol

Operation	$\overline{\text{MSEO}}$/$\overline{\text{MSEI}}$ state
Idle	“1”s at all clocks
Start	Two “1”s followed by one “0”
Active	“0”s at all clocks during transmission of a message
End of variable length packet	“0” followed by “1”
End of packet and message	“0” followed by two or more “1”s

$\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ are used to signal the end of variable-length packets and messages. They are not required to indicate end of fixed length packets. $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ are sampled on the rising edge of MCKI and MCKO respectively.

Fixed width fields can be concatenated before variable length fields without regard to the individual fields starting or ending at message ***N*** bit boundaries. Variable width fields must end at message ***N*** bit boundaries (where ***N*** is MDI/MDO pins).

Figure 23-4 shows the basic relation between the MDO and $\overline{\text{MSEO}}$ pins, and packet structure. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

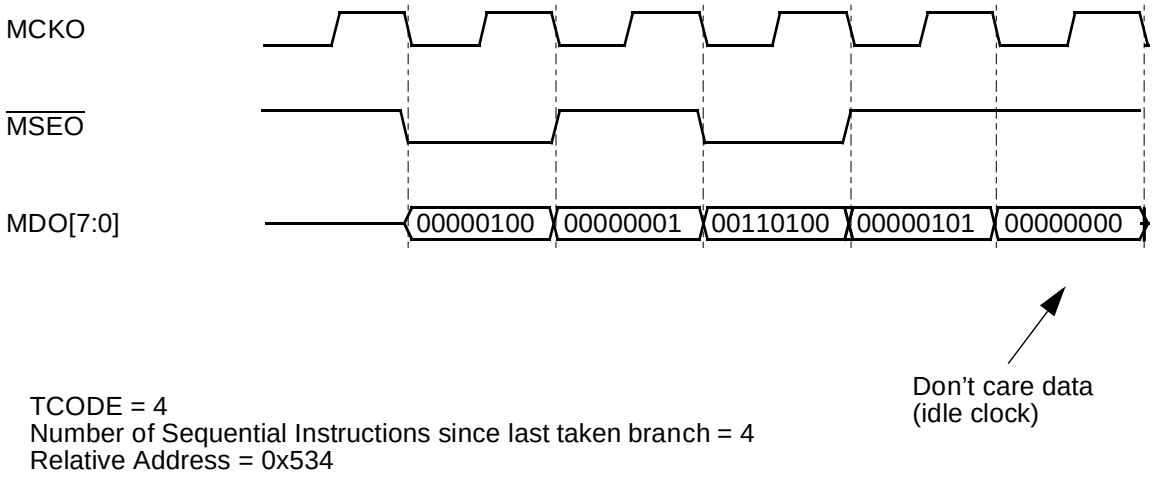


Figure 23-4 Auxiliary Pin Packet Structure for Program Trace Indirect Branch Message

Figure 23-5 illustrates the state diagram for $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ transfers.

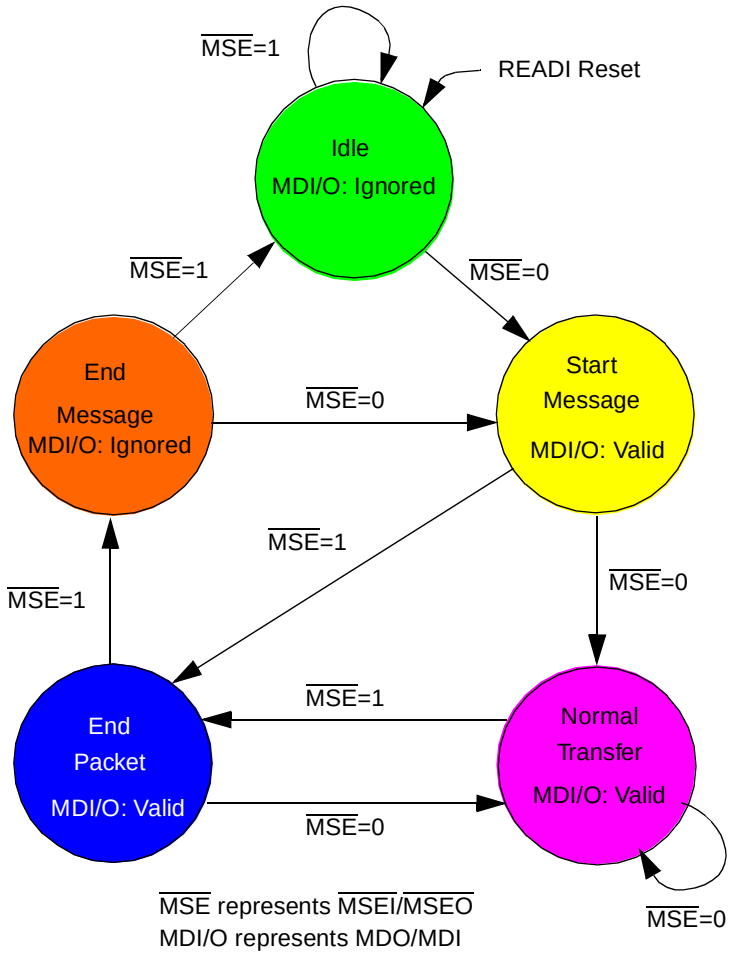


Figure 23-5 $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ Transfers

NOTE

In the “End Message” state data on the MDI/MDO pins is ignored.

23.3.2.3 Messages

Public messages outlined in **Table 23-2** are supported by READI.

Table 23-20 Public Messages Supported

Message Name	Minimum Packet Size (bits)	Maximum Packet Size (bits)	Packet Type	Packet Description	Direction
Device ID	6	6	Fixed	TCODE number = 1	From Device
	32	32	Fixed	Device ID information	
Ownership Trace Message	6	6	Fixed	TCODE number = 2	From Device
	32	32	Fixed	Task/Process ID tag	
Program Trace — Direct Branch Message	6	6	Fixed	TCODE number = 3	From Device
	1	8	Variable	number of sequential instructions executed since last taken branch	
Program Trace — Indirect Branch Message	6	6	Fixed	TCODE number = 4	From Device
	1	8	Variable	number of sequential instructions executed since last taken branch	
	1	23	Variable	unique portion of the target address for taken branches and exceptions	
Data Trace — Data Write Message	6	6	Fixed	TCODE number = 5	From Device
	1	25	Variable	unique portion of the data write address	
	8	32	Variable	data write value (8, 16, 32 bits)	
Data Trace — Data Read Message	6	6	Fixed	TCODE number = 6	From Device
	1	25	Variable	unique portion of the data read address	
	8	32	Variable	data read value (8, 16, 32 bits)	
Error Message	6	6	Fixed	TCODE number = 8	From Device
	5	5	Fixed	error code	
Program Trace Correction Message	6	6	Fixed	TCODE number = 10 (0xA)	From Device
	1	8	Variable	correcting the number of instructions in the trace	
Program Trace — Direct Branch Synchronization Message	6	6	Fixed	TCODE number = 11 (0xB)	From Device
	1	1	Variable	number of program trace messages cancelled	
	1	23	Variable	full target address	
Program Trace — Indirect Branch Synchronization Message	6	6	Fixed	TCODE number = 12 (0xC)	From Device
	1	1	Variable	number of program trace messages cancelled	
	1	23	Variable	full target address	
Data Trace — Data Write Synchronization Message	6	6	Fixed	TCODE number = 13 (0xD)	From Device
	1	1	Variable	number of messages canceled	
	1	25	Variable	full target address	
	8	32	Variable	data write value (8, 16, 32 bits)	

Table 23-20 Public Messages Supported (Continued)

Message Name	Minimum Packet Size (bits)	Maximum Packet Size (bits)	Packet Type	Packet Description	Direction
Data Trace — Data Read Synchronization Message	6	6	Fixed	TCODE number = 14 (0xE)	From Device
	1	1	Variable	number of messages canceled	
	1	25	Variable	full target address	
	8	32	Variable	data read value (8, 16, 32 bits)	
Watchpoint Message	6	6	Fixed	TCODE number = 15 (0xF)	From Device
	6	6	Fixed	number indicating watchpoint source	
Auxiliary Access — Device Ready for Upload/Download Message	6	6	Fixed	TCODE number = 16 (0x10)	From Device
Auxiliary Access — Upload Request Message	6	6	Fixed	TCODE number = 17 (0x11)	From Tool
	8	8	Fixed	opcode to enable selected configuration, status or data upload from MCU	
Auxiliary Access — Download Request Message	6	6	Fixed	TCODE number = 18 (0x12)	From Tool
	8	8	Fixed	opcode to enable selected configuration or data download to MCU	
	8	80	Variable	Depending upon opcode selected for download, information to be downloaded to device will vary.	
Auxiliary Access — Upload/Download Information Message	6	6	Fixed	TCODE number = 19 (0x13)	From Device / Tool
	8	80	Variable	1). For an access, depending on word size selected (SZ field in RWA register), variable-length packets of information (10, 18, or 34bits) will be uploaded/downloaded from/to device. 2). Depending upon opcode selected for upload from internal READI registers, information to be uploaded to the device will vary.	

Table 23-21 describes the error code encoding for error messages.

Table 23-21 Error Codes

Error Code	Description
00000	Ownership trace overrun (Not Used)
00001	Program trace overrun (Not Used)
00010	Data trace overrun (Not Used)
00011	Read/write access error
00100	Invalid message
00101	Invalid Access Opcode
00110	Watchpoint overrun
00111	Program/Data/Ownership trace overrun
01000-10111	Reserved
11000-11111	Vendor Defined

Vendor-defined messages outlined in **Table 23-22** are also supported by READI.

Table 23-22 Vendor-Defined Messages Supported

TCODE Name	Minimum Packet Size (bits)	Maximum Packet Size (bits)	Packet Type	Packet Description	Direction
Dev Port Access — DSDI Data Message	6	6	Fixed	TCODE number = 56 (0x38)	From Tool
	10	35	Variable	BDM Development Serial Data In (DSDI)	
Dev Port Access — DSDO Data Message	6	6	Fixed	TCODE number = 57 (0x39)	From Device
	10	35	Variable	BDM Development Serial Data Out (DSDO)	
Dev Port Access — BDM Status Message	6	6	Fixed	TCODE number = 58 (0x3A)	From Device
	1	1	Fixed	BDM status	
Program Trace — Indirect Branch Message With Compressed Code	6	6	Fixed	TCODE number = 59 (0x3B)	From Device
	1	8	Variable	Number of sequential instructions executed since last taken branch	
	6	6	Fixed	Bit address	
	1	23	Variable	Unique portion of the target address for taken branches and exceptions (compressed code)	
Program Trace — Direct Branch Synchronization Message With Compressed Code	6	6	Fixed	TCODE number = 60 (0x38C	From Device
	6	6	Fixed	Bit address	
	1	23	Variable	Current instruction address	
Program Trace — Indirect Branch Synchronization Message With Compressed Code	6	6	Fixed	TCODE number = 61 (0x3D)	From Device
	6	6	Fixed	Bit address	
	1	23	Variable	Current instruction address	

23.3.2.4 Message Formats

Message formatting is performed in the pin interface block. Raw messages read from the message queue are independent of the number of MDO pins implemented.

Table 23-23 shows the various message formats that the pin interface formatter has to encounter.

NOTE

For variable length fields, the transmitted size of the field is determined as the bits from the least significant bit to the most significant non-zero valued bit, (i.e., most significant 0 value bits are not transmitted).

Table 23-23 Message Field Sizes

Message	TCODE	Field # 1	Field # 2	Field # 3	Max Size[1]	Min. Size[2]
Device ID	1	Fixed = 32	NA	NA	38 bits	38 bits
Ownership Trace Message	2	Fixed = 32	NA	NA	38 bits	38 bits
Program Trace — Direct Branch Message	3	Variable Max = 8 Min = 1	NA	NA	14 bits	7 bits
Program Trace — Indirect Branch Message	4	Variable Max = 8 Min = 1	Variable Max = 23 Min = 1	NA	37 bits	8 bits
Data Trace — Data Write Message	5	Variable Max = 25 Min = 1	Variable Max = 32 Min = 8	NA	63 bits	15 bits
Data Trace — Data Read Message	6	Variable Max = 25 Min = 1	Variable Max = 32 Min = 8	NA	63 bits	15 bits
Error Message	8	Fixed = 5	NA	NA	11 bits	11 bits
Program Trace Correction Message	10	Variable Max = 8 Min = 1	NA	NA	14 bits	7 bits
Program Trace — Direct Branch Synchronization Message	11	Variable Max = 1 Min = 1	Variable Max = 23 Min = 1	NA	30 bits	8 bits
Program Trace — Indirect Branch Synchronization Message	12	Variable Max = 1 Min = 1	Variable Max = 23 Min = 1	NA	30 bits	8 bits
Data Trace — Data Write Synchronization Message	13	Variable Max = 1 Min = 1	Variable Max = 25 Min = 1	Variable Max = 32 Min = 8	64 bits	16 bits
Data Trace — Data Read Synchronization Message	14	Variable Max = 1 Min = 1	Variable Max = 25 Min = 1	Variable Max = 32 Min = 8	64 bits	16 bits

Table 23-23 Message Field Sizes

Message	TCODE	Field # 1	Field # 2	Field # 3	Max Size[1]	Min. Size[2]
Watchpoint Message	15	Fixed = 6	NA	NA	12 bits	12 bits
Auxiliary Access — Device Ready for Upload/Download Message	16	NA	NA	NA	6 bits	6 bits
Auxiliary Access — Upload Request (Tool requests information) Message	17	Fixed = 8	NA	NA	14 bits	14 bits
Auxiliary Access — Download Request (Tool provides Information) Message	18	Fixed = 8	Variable Max = 80 Min = 8	NA	94 bits	22 bits
Auxiliary Access — Upload/Download Information (Device/ Tool provides Information) Message	19	Variable Max = 80 Min = 8	NA	NA	86 bits	14 bits
Dev Port Access — DSDI Data (Tool Provides Information) Message	56	Variable Max = 35 Min = 10	NA	NA	41 bits	16 bits
Dev Port Access — DSDO Data (Device Provides Information) Message	57	Variable Max = 35 Min = 10	NA	NA	41 bits	16 bits
Dev Port Access — BDM Status (Device Provides Information) Message	58	Fixed = 1	NA	NA	7 bits	7 bits
Program Trace — Indirect Branch Message With Compressed Code	59	Variable Max = 8 Min = 1	Fixed = 6	NA	43 bits	14 bits
Program Trace — Direct Branch Synchronization Message With Compressed Code	60	Fixed = 6	Variable Max = 23 Min = 1	NA	35 bits	13 bits
Program Trace— Indirect Branch Synchronization Message With Compressed Code	61	Fixed = 6	Variable Max = 23 Min = 1	NA	35 bits	13 bits

NOTES:

1. Maximum information size. The actual number of bits transmitted is dependant on the number of MDO pins.
2. Minimum information size. The actual number of bits transmitted is dependent on the number of MDO pins.

The maximum message length is 94 bits. The maximum number of fields is 3, excluding the TCODE itself.

The double edges in **Table 23-23** indicate the required $\overline{\text{MSEO}}/\overline{\text{MSEI}}$ assertion or negation indicating the start of message or end of message respectively.

The shaded edges in **Table 23-23** indicate how information can be packed into super-fields delimited via $\overline{\text{MSEO}}/\overline{\text{MSEI}}$ assertion followed by $\overline{\text{MSEO}}/\overline{\text{MSEI}}$ negation.

23.3.2.5 Rules of Messages

- A variable sized field within a message must end on a port boundary.
- A variable sized field may start within a port boundary only when following a fixed length packet.
- Super-fields must end on a port boundary (2-, 4-, or 8-bit boundaries depending on whether the device receives or sends messages, and the port size configured).
- When a variable length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest-order bit so that it can end on a port boundary.
- A data field within a data trace message must be eight, 16, or 32 bits in length.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information. Within a field, the lowest significant bits are shifted out first. **Figure 23-6** shows the transmission sequence of a message which is made up of a TCODE, a variable length field, and finally a super-field consisting of a fixed length and a variable length field. The only exception to this rule is the dev port access messages. See **23.10.1 RCPU Development Access Messaging** for further details.

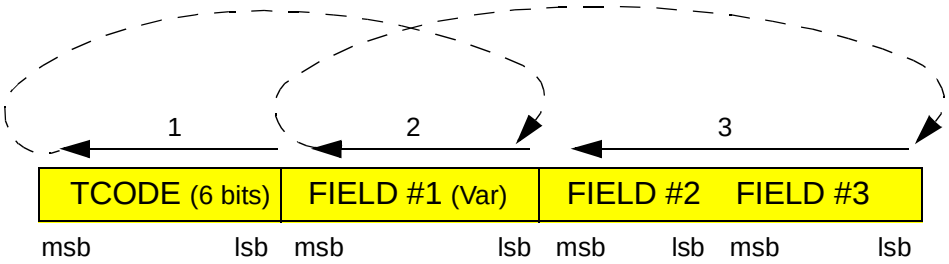


Figure 23-6 Transmission Sequence of messages.

23.3.2.6 Examples

The following are examples of branch trace messages.

Table 23-24 illustrates an example of how the indirect branch public message is transmitted.

NOTE 1

The example uses a four-bit output port. Note also that T0, I0, and A0 are the least significant bits where:

Tx = TCODE number (fixed)
Ix = Number of sequential instructions (variable)
Ax = Unique portion of the address (variable)

NOTE 2

During clock 7, the tool should ignore data on MDO pins.

Table 23-24 Indirect Branch Message

Clock	MDO[3:0]				$\overline{\text{MSEO}}$	
	3	2	1	0	1	Idle
0	X	X	X	X	1	Idle (or end of last message)
1	T3	T2	T1	T0	0	Start Message
2	I1	I0	T5	T4	0	Normal Transfer
3	I5	I4	I3	I2	0	Normal Transfer
4	0	0	I7	I6	1	End Packet
5	A3	A2	A1	A0	0	Normal Transfer
6	A7	A6	A5	A4	1	End Packet
7	0	0	0	0	1	End Message
8	T3	T2	T1	T0	0	Start Message

Table 23-24 is an example of the minimum transmission of any message containing a variable length field (three clocks).

NOTE 1

The example uses a 4-bit output port. Note also that T0, and I0 are the least significant bits where:

Tx = TCODE number (fixed)
Ix = Number of sequential instructions (variable)

NOTE 2

During clock 3, the tool should ignore data on MDO pins.

Table 23-25 Direct Branch Message

Clock	MDO[3:0]				$\overline{\text{MSEO}}$	
	3	2	1	0	1	Idle
1	T3	T2	T1	T0	0	Start Message
2	I1	I0	T5	T4	1	End Packet
3	0	0	0	0	1	End Message

23.3.2.7 Non-Temporal Ordering of Transmitted Messages

Trace messages sent out may not be in the sequence they actually occurred. The traces are monitored on the internal buses and these traces are captured as they occur and are sent out in the order they were captured and processed.

BTMs are in sequence and DTMs are in sequence, however, temporal order of DTMs interleaved with BTMs may not be accurate with regard to logical flow of code.

23.3.3 READI Reset Configuration

The READI reset configuration information is received via $\overline{\text{EVTI}}$ and MDI[0] to enable or disable the READI module and select the port size. $\overline{\text{EVTI}}$ and MDI[0] are sampled synchronously at the negation of $\overline{\text{RSTI}}$. Reset configuration information must be valid on $\overline{\text{EVTI}}$ and MDI[0] at least four clocks prior to the negation of $\overline{\text{RSTI}}$.

If $\overline{\text{EVTI}}$ is sampled asserted at negation of $\overline{\text{RSTI}}$, the READI module will be enabled. This is illustrated in **Figure 23-7**.

READI control and status information will be reset and the auxiliary output port will be three-stated, when $\overline{\text{RSTI}}$ is asserted. System reset will not reset the READI control and status information and not three-state the auxiliary output port.

Port size configuration is selected via the value of MDI[0] at the negation of $\overline{\text{RSTI}}$. **Table 23-26** describes the READI reset configuration options.

Table 23-26 READI Reset Configuration Options

$\overline{\text{EVTI}}$	MDI [0]	Configuration
1	X	Module Disabled. All outputs three-stated.
0	1	Module Enabled, Default Port Configuration 2 MDI, 8 MDO
0	0	Module Enabled, Alternate Port Configuration 1 MDI, 2 MDO

$\overline{\text{RSTI}}$ has a pull-down resistor in the pads. If the auxiliary port is not connected to a tool, READI module will be in reset state and not drive the auxiliary output port.

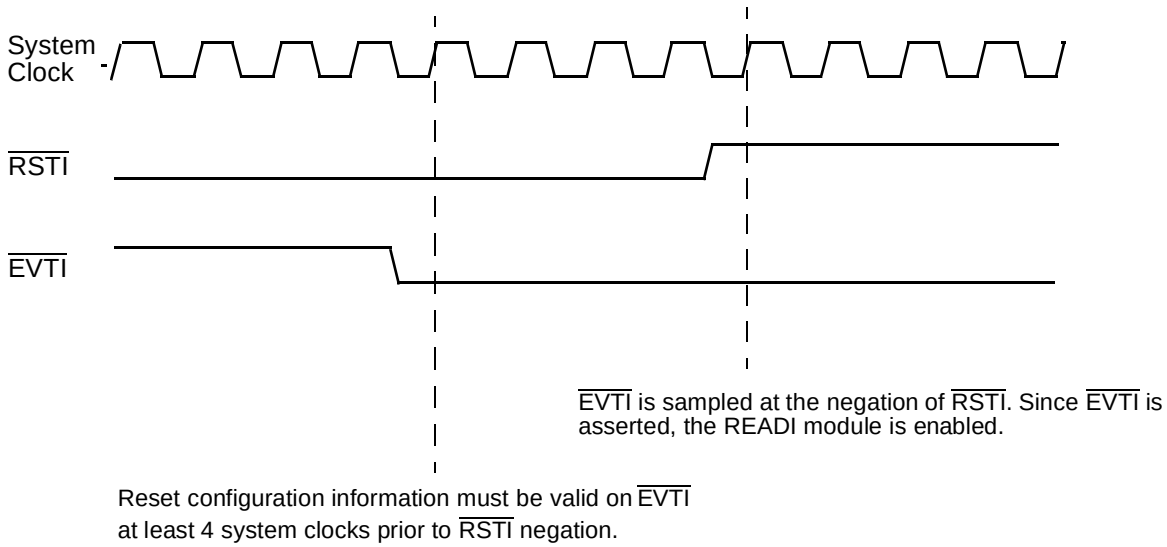


Figure 23-7 READI Module Enabled

23.3.3.1 Reset Configuration for Debug Mode

To enable RCPU development access via the READI pins, the reset sequence outlined below should be used:

- Assert READI reset (RSTI), event-in (EVTI) and system reset (HRESET)
- Negate RSTI
- Upon negation of RSTI, tool should configure the DOR, DME, and DPA fields in the DC register to desired setting.
- Tool negates HRESET at least 16 system clocks after receiving the device ready message

Refer to **Figure 23-68** for further details.

23.3.3.2 Reset Configuration for Non-Debug Mode

Refer to **23.3.3.1 Reset Configuration for Debug Mode** for details on reset configuration for non-debug mode. The only difference between non-debug mode reset configuration and debug mode reset configuration is the values of the DOR, DME, and DPA fields in the DC register.

23.3.3.3 Secure Mode

Refer to **23.1.4.2 Security** for further details.

23.3.3.4 Disabled Mode

If EVTI is negated at negation of RSTI, the READI module will be disabled. No trace output will be provided, and output auxiliary pins will be three-stated. This is illustrated in **Figure 23-8**.

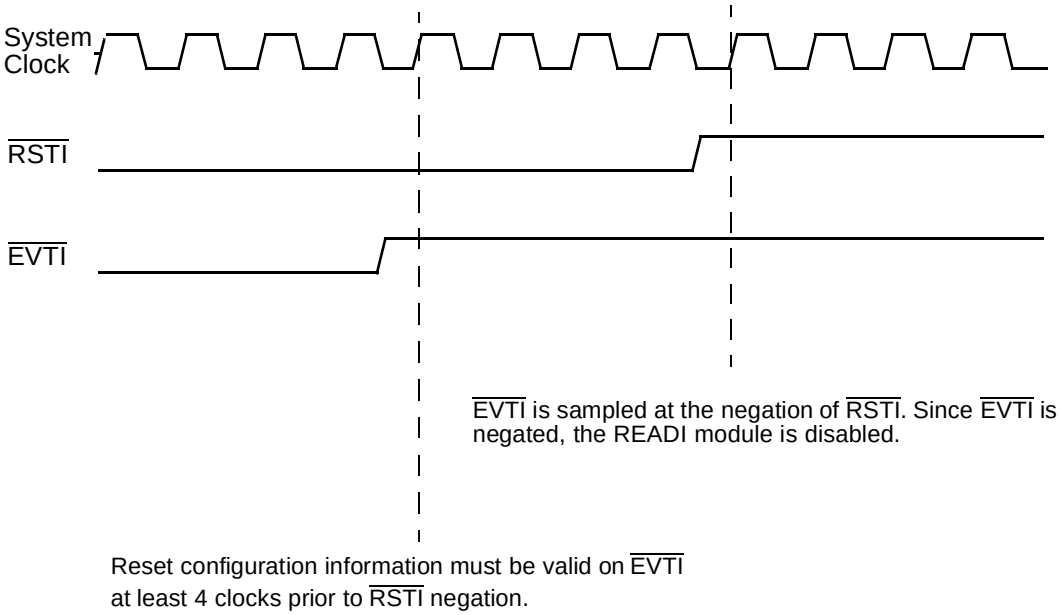


Figure 23-8 READI Module Disabled

23.3.3.5 Guidelines For Transmitting Input Messages

- An error message is sent out when an invalid TCODEis detected by the pin input formatter. Refer to **23.6.8.2 Invalid Message** for further details.
- An error message is sent out when an invalid access opcode is detected in auxiliary input messages by the pin input formatter. Refer to **23.6.8.3 Invalid Access Opcode** for further details.
- If the TCODE is valid, then READI will expect that the correct number of packets have been received and no further checking will be performed. If the number of packets received by READI is not correct, READI response is not defined, unless the message is a download request message (refer to **23.2.4 Partial Register Updates** for further details).

23.4 Program Trace

This section details the program trace mechanism supported by READI for the RCPU. Program trace is implemented via branch trace messaging (BTM) as per the IEEE-ISTO 5001 - 1999 definition.

23.4.1 Branch Trace Messaging

Branch trace messaging facilitates program trace by providing the following types of information:

- Messaging for taken direct branches includes how many sequential instructions were executed since the last taken branch or exception. Direct (or indirect) branches not taken are counted as sequential instructions.
- Messaging for taken indirect branches and exceptions includes how many se-

quential instructions were executed since the last taken branch or exception and the unique portion of the branch target address or exception vector address.
- For some mispredicted branches and exception occurrences, program trace correction messages correct the number of instructions since last taken branch as transmitted in prior BTM message.

23.4.1.1 RCPU Instructions that Cause BTM Messages

The following RCPU instructions, when executed, cause indirect branch messages to be encoded:

1. Taken branch relative to link or counter registers
2. Context switching sequential Instructions
3. Exception taken (error/interrupts)

The following RCPU instruction, when executed, causes direct branch messages to be encoded:

4. Taken direct branch instructions

23.4.2 Review of RCPU Instruction Execution

Branch trace messaging for the RCPU is accomplished by snooping the U-bus address, data and program trace signals, and RCPU VF and VFLS signals, and performing algorithms necessary to generate the trace messages.

23.4.2.1 RCPU Pipeline and Execution Model

The RCPU instruction sequencer provides the central control for data flow between execution units and data files. It also implements the instruction pipeline to fetches and issues instructions to the execution units, and to control the pipeline. **Figure 23-9** depicts the RCPU instruction flow.

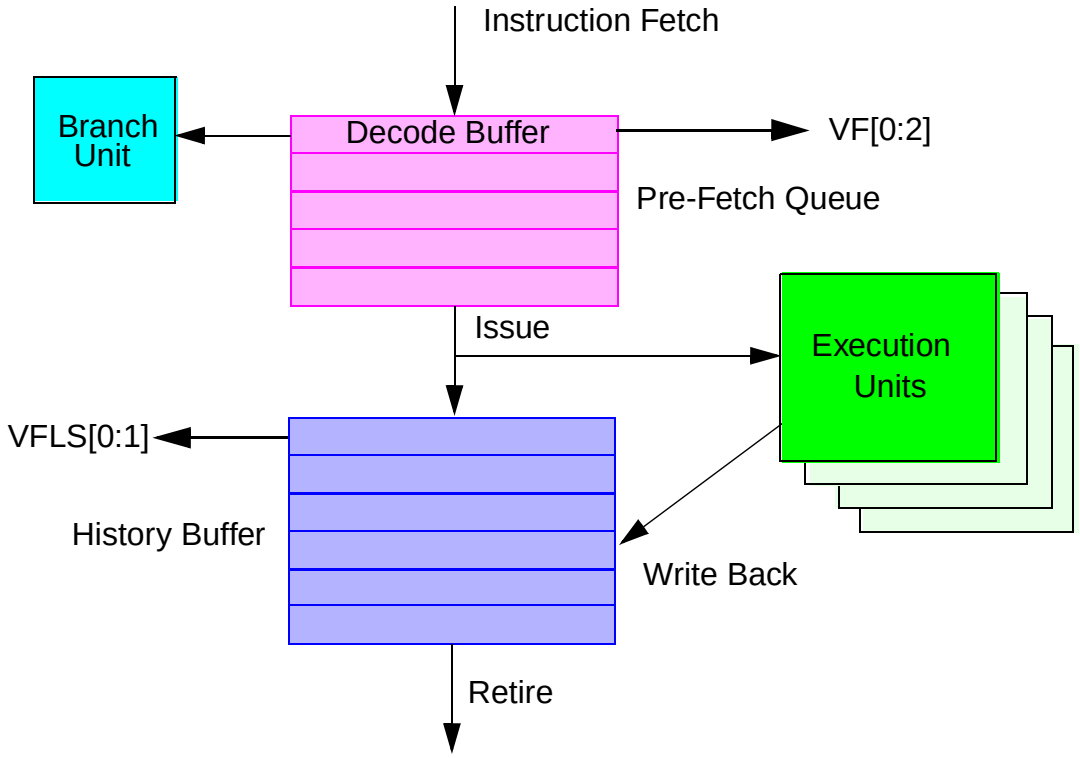


Figure 23-9 RCPU Instruction Flow

A generalized execution flow for the sequencer is as follows. For more details refer to the ***RCPU Reference Manual*** **(RCPURM/AD)**.

1. Fetch instructions into the prefetch queue [4+1 instructions deep].
2. Decode instructions fetched into the prefetch queue. Signal instruction type based on the decode through the VF signals.
   — If the decoded branch instruction is unconditional, instructions subsequent to the branch instruction in the prefetch queue are flushed. Fetching of instructions is started from the new target address.
   — If the decoded branch instruction is conditional and predicted not taken, the branch instruction is treated as a sequential instruction. Instructions are not flushed from the prefetch queue.
   — If the decoded branch instruction is conditional and predicted taken, instructions subsequent to the branch instruction in the prefetch queue are flushed. Fetching of instructions is started from the new target address.
3. Issue decoded instructions at the top of the prefetch queue to the execution units, and load processor status into history buffer [six instructions deep].
4. If the conditional branch instruction which was predicted taken, is subsequently discovered to be mispredicted (condition not met), fetched instructions (from the new target address specified by the branch instruction) are flushed. The

cancelled branch and the number of flushed instructions are signalled through the VF signals. Now fetching of instructions is started from the old address (before the branch was predicted as taken).

5. An instruction is ‘retired’ when the execution unit completes execution (without an exception) of the issued instruction (write-back completed) and the instruction is at the top of the history buffer queue; (i.e., all previously issued instructions completed without exception).
   — Issued instructions may complete execution out of order, but are always retired from the history buffer in order.
6. When an instruction-caused exception is recognized, instructions that appear earlier in the instruction stream are required to complete before the exception is taken. An instruction is said to have “completed” when the results of that instruction’s execution has been committed to the appropriate registers (i.e., following the writeback stage).
   — Exception conditions may be recognized out of order, but they are handled strictly in order.
7. Instructions that appear after the exception-causing instruction which have not been executed are flushed from the prefetch queue. The number of instructions flushed is signalled via the VF signals. Instructions that appear after the exception-causing instruction which have been executed are flushed from the history buffer. The number of instructions flushed from the history buffer are signaled via the VFLS signals.
8. When exceptions occur, information about the state of the processor is saved to certain registers, and the processor begins execution at an address predetermined for each exception.

As can be seen from the above flow, the following issues arise, when tracking the execution of the RCPU:

1. Not all fetched (decoded) instructions are executed.
2. Not all executed (issued and completed) instructions are retired.
3. Taken branches may be cancelled due to misprediction or exceptions.
4. Executed instructions (sequential or branches) which are subsequent to the exception-causing instruction will be flushed from the history buffer due to exception.

The trace information that is generated by RCPU is ideally suited for off-line software oriented processing, where there is the ability to trace back and recover from flushed queues.

23.4.2.2 RCPU Branch Trace Indicators

The RCPU supports program trace by providing information on the status of the instruction decode, instruction queue flush and history buffer flush. For more information, consult the section on development support in the ***RCPU Reference Manual (RCPURM/AD)***. The branch trace indicators from the RCPU are the VF and the VFLS signals.

The VF signals indicate the type for the current instruction decoded and the outcomes of change-of-flow instructions. They also indicate the number of instructions flushed from the instruction prefetch queue on change of flow. This indication is in the clock following the indication of change of flow. For details on VF instruction type encoding and VF queue flush Information refer to **Table 23-27** and **Table 23-28** respectively.

Table 23-27 VF Instruction Type Encoding

VF	Instruction Type	Next VF Encoding
000	None	Instruction Type
001	Sequential	
010	Branch (direct or indirect) NOT taken	
011	VSYNC was asserted/negated and therefore the next instruction will be marked with program trace cycle attribute	
100	Interrupt/Exception taken – the target instruction fetch will be marked with program trace cycle attribute.	Queue Flush Information
101	Branch Indirect taken, ***rfi***, ***mtmsr***, ***isync*** and in some cases ***mtspr*** – the target instruction fetch will be marked with program trace cycle attribute.	
110	Branch direct taken	
111	Branch (direct or indirect) NOT taken (misprediction)	

The RCPU VF signals indicate (instruction pre-fetch) queue flush information in the clock following a taken change-of-flow indication. This encoding is as follows.

Table 23-28 VF Queue Flush Information

VF	Instruction Type	Next VF Encoding
000	0 Instructions flushed from instruction queue	Instruction Type
001	1 Instructions flushed from instruction queue	
010	2 Instructions flushed from instruction queue	
011	3 Instructions flushed from instruction queue	
100	4 Instructions flushed from instruction queue	
101	5 Instructions flushed from instruction queue	
110	Reserved	
111	Branch (direct or indirect) NOT taken (mis-prediction)	Queue Flush Information

NOTE

Encoding VF equal to 0b111 for the queue flush information indicates that the predicted branch (indicated in the previous clock by appropriate VF encoding) is cancelled, and that further flush information follows in the next clock.

The VFLS signals track the history buffer flushes. This aids the READI module to determine how many instructions actually executed and retired. Refer to **Table 23-29** for details on VFLS encoding.

Table 23-29 VFLS History Buffer Flush Encoding

VFLS	History Buffer Flush Information
00	0 Instructions flushed from History Buffer
01	1 Instructions flushed from History Buffer
10	2 Instructions flushed from History Buffer
11	Ignored for Program Trace

NOTE

The history buffer has a queue depth of six instructions, but can flush (restore processor state) up to two instructions at a time.

Using the RCPU program trace information available via the VF and the VFLS signals, and snooping the U-bus for instruction fetches, the READI module generates program trace messages to track the change of flow for the RCPU.

23.4.3 BTM Message Formats

BTM messages are of five types — direct, indirect, correction, error, and synchronization.

23.4.3.1 Direct Branch Messages

Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode.

The program trace direct branch message has the following format:

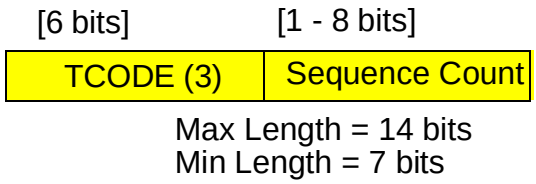


Figure 23-10 Direct Branch Message Format

23.4.3.2 Indirect Branch Messages

Indirect branches include interrupts, exceptions, and all taken branches whose destination is determined at run time. For the RCPU, certain sequential instructions are tagged with the indirect change-of-flow attribute because these instruction affect the machine in a similar manner to true indirect change-of-flow instructions. These instruc-

tions are the ***rfi***, ***isync***, ***mtmsr*** and certain ***mtspr*** (to CMPA – CMPF, ICTRL, ECR and DER)

The program trace indirect branch message has the following format:

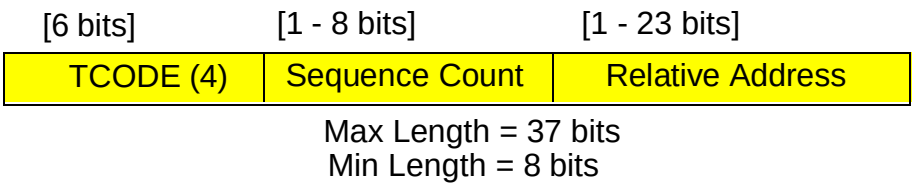


Figure 23-11 Indirect Branch Message Format

For compressed code support, six additional bits indicate the starting bit address within the word of the compressed instruction.

The program trace indirect branch with compressed code message has the following format.

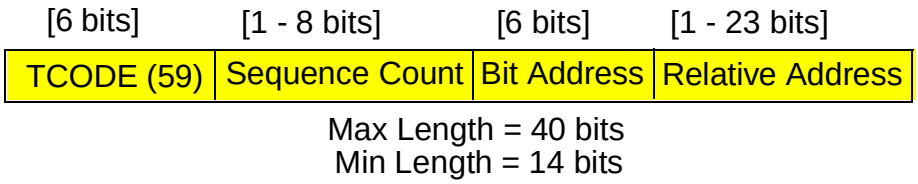


Figure 23-12 Indirect Branch Message Format with Compressed Code

23.4.3.3 Program Trace Correction Message

In case of a mispredicted branch or an exception, a program trace correction message may also be sent indicating a number which corrects the number of instructions (not messages) in the trace.

In the case of a synchronizing branch trace message getting corrected due to an exception or misprediction, the next branch trace message will be a synchronizing message.

Table 23-30 illustrates an example of a program trace correction message in case of a mispredicted branch. **Table 23-31** illustrates an example of a program trace correction message in case of an exception.

NOTE 1

In case of an exception, the sequential instruction count is reset to 0, after the program trace correction message is sent.

NOTE 2

In case of an mispredicted branch, the correction count is always 1 and the sequential instruction count is reset to 1 (to denote the not-taken branch as a sequential instruction), after the program trace correction message is sent. This is because a mispredicted branch is considered to be a sequential instruction.

Table 23-30 Program Trace Correction Due to a Mispredicted Branch

Time	Processor State	Message sent
1	Sequential Instruction	
2	Sequential Instruction	
3	Sequential Instruction	
4	Sequential Instruction	
5	Sequential Instruction	
6	Direct Branch Instruction	Direct Branch Message TCODE = 3 Number of sequential instructions executed since last taken branch = 5
7	Sequential Instruction	
8	Sequential Instruction	
9	Sequential Instruction	
10	Sequential Instruction	
11	Indirect Branch Instruction (mispredicted taken)	Indirect Branch Message TCODE = 4 Number of sequential instructions executed since last taken branch = 4 Unique portion of the target address
12	Sequential Instruction	
13	Sequential Instruction	
14	Sequential Instruction	
15	Branch Correction	Program Trace Correction Message TCODE = 10 Number of instructions to rewind from trace = 1
16	Sequential Instruction	
17	Sequential Instruction	

Table 23-30 Program Trace Correction Due to a Mispredicted Branch (Continued)

Time	Processor State	Message sent
18	Indirect Branch Instruction (predicted taken)	Indirect Branch Message TCODE = 4 Number of sequential instructions executed since last valid taken branch = 3 Unique portion of the target address
At Time 11, the Indirect Branch is mispredicted taken. At Time 15, branch correction occurs due to the mispredicted branch which was taken at Time 11. A program trace correction message is sent out correcting the number of instructions in the trace (1). Sequential instruction which occurred at Time 12, 13, and 14 respectively are not included in the correction count because the tool is not aware that they occurred (they were not transmitted out). At Time 18, the Indirect Branch Message indicates that 3 sequential instructions were executed since trace correction (this includes the mispredicted branch instruction which is considered to be a sequential instruction).		

Table 23-31 Program Trace Correction Due to an Exception

Time	Processor State	Message sent
1	Sequential Instruction	
2	Sequential Instruction	
3	Sequential Instruction	
4	Direct Branch Instruction	Direct Branch Message TCODE = 3 Number of sequential instructions executed since last taken branch = 3
5	Sequential Instruction	
6	Sequential Instruction	
7	Sequential Instruction	
8	Sequential Instruction	
9	Indirect Branch Instruction	Indirect Branch Message TCODE = 4 Number of sequential instructions executed since last taken branch = 4 Unique portion of the target address
10	Sequential Instruction	
11	Sequential Instruction	
12	Indirect Branch Instruction	Indirect Branch Message TCODE = 4 Number of sequential instructions executed since last taken branch = 2 Unique portion of the target address

Table 23-31 Program Trace Correction Due to an Exception (Continued)

Time	Processor State	Message sent
13	Sequential Instruction	
14	Exception due to instruction at Time 8	Program Trace Correction Message TCODE = 10 Number of instructions to rewind from trace = 5
16	Indirect Branch Instruction	Indirect Branch Message TCODE = 4 Number of sequential instructions executed since last taken branch = 0 Unique portion of the target address
At Time 8, the Sequential instruction that causes an exception is issued. At Time 14, the instruction issued at Time 8 causes an exception. A Program Trace Correction Message is sent out correcting the number of instructions in the trace (5). The sequential instruction that occurred at Time 13 is not included in the correction count because the tool is not aware that it occurred (it was not transmitted out). Note: The sequential instruction at Time 8 did not retire and is included in the correction number.		

The program trace correction message has the following format:

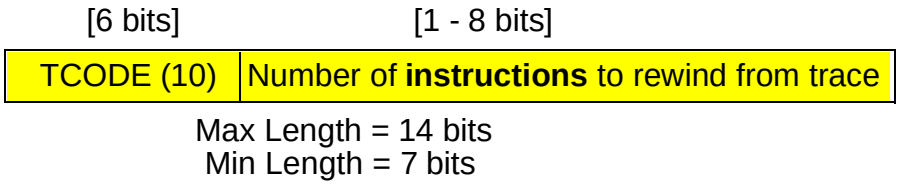


Figure 23-13 Program Trace Correction Message Format

23.4.3.4 Error Message (Queue Overflow)

A program/data/ownership trace overrun error occurs when a trace message cannot be queued due to the queue being full, provided program trace is enabled.

The overrun error causes the message queue to be flushed, and an error message to be queued. The error code within the error message indicates that a program/data/ownership trace overrun error has occurred. The next BTM will be a synchronization message.

The error message has the following format:

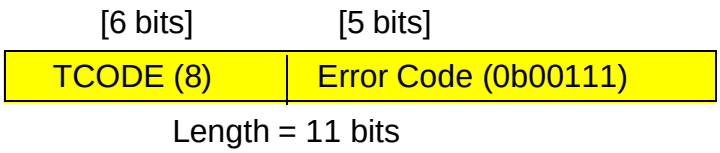


Figure 23-14 Error Message (Queue Overflow) Format

23.4.3.5 Program Trace Synchronization Messages

A program trace synchronization message is transmitted via the auxiliary port (provided program trace is enabled) for the following conditions:

- Initial program trace message upon exit of any system reset will be a synchronization message.
- Upon exit of sleep, deep-sleep and low power down mode, the first BTM will be a synchronization message.
- Initial program trace message upon exit of background debug mode. Upon exiting BDM, the next BTM will be a synchronization message.
- When BTM is enabled, the first BTM will be a synchronization message.
- After 255 program trace messages have been queued without synchronization, the next BTM will be a synchronization message.
- Upon assertion of an event In ($\overline{\text{EVTI}}$) pin. If the READI module is not disabled, an $\overline{\text{EVTI}}$ assertion will cause the next BTM to be a synchronization message (provided the EC field is 0b00 in the DC register).
- Upon occurrence of a watchpoint, the next BTM will be a synchronization message (provided program trace is enabled).
- Occurrence of queue overrun. A program trace overrun error occurs when a trace message cannot be queued due to the queue being full. This causes the message queue to be flushed, and an error message is placed as the first message in the queue. The error code within the Error message will indicate that program/data/ownership trace overrun has occurred. The next BTM will be a synchronization message.
- Sequential instruction count overflow. When the sequential instruction counter reaches its maximum count (up to 256 sequential instructions may be executed), the next BTM will be a program trace synchronization message.The sequential instruction counter is reset.
- Upon entering or exiting code compression mode, the next BTM will be a synchronization message.
- The next change-of-flow instruction fetch following VSYNC will be a synchronization message.

Program trace synchronization messages provide the full address (without leading zeros) and ensure that development tools fully synchronize with program trace regularly. Synchronization messages provide a reference address for subsequent BTMs, in which only the unique portion of the program trace address is transmitted.

NOTE

For program trace synchronization to work, the ICTRL register must be programmed such that show cycle will be performed for all changes in the program flow (ISCTL field = 01).

It is also recommended that the USIU be programmed to ignore instruction show cycles (so as to not impact U-bus performance). Synchronization will only occur at changes in program flow boundaries, and cannot be forced by the READI module. Synchronizations on errors, overflows, as well as periodic synchronizations will not be deterministic to the nearest instruction, but to the next taken change in program flow. The start of program trace (enabled via any means) will be also deferred to the next change in program flow.

Program trace synchronization messages are of the following types:

- Direct branch with compressed code
- Indirect branch with compressed code
- Direct branch
- Indirect branch

23.4.3.6 Direct Branch Synchronization Message

The program trace direct branch synchronization message has the following format:

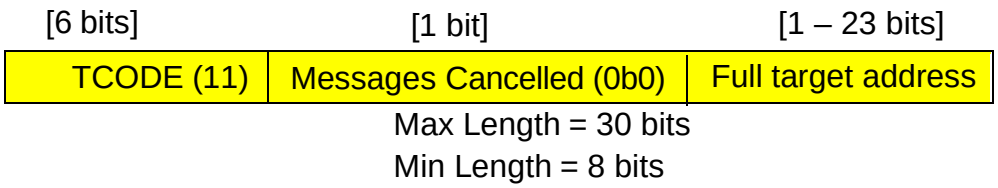


Figure 23-15 Direct Branch Synchronization Message Format

23.4.3.7 Indirect Branch Synchronization Message

The program trace indirect branch synchronization message has the following format:

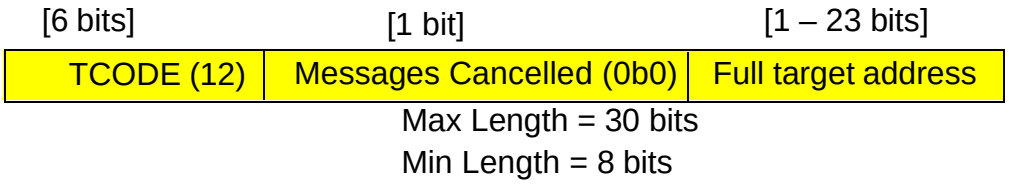


Figure 23-16 Indirect Branch Synchronization Message Format

23.4.3.8 Direct Branch Synchronization Message With Compressed Code

For compressed code support, six additional bits indicate the starting bit address within the word of the compressed instruction. The program trace direct branch synchronization with compressed code message has the following format:

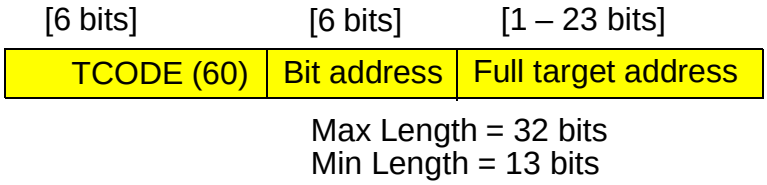


Figure 23-17 Direct Branch Synchronization Message Format With Compressed Code

23.4.3.9 Indirect Branch Synchronization Message with Compressed Code

For compressed code support, six additional bits indicate the starting bit address within the word of the compressed instruction. The program trace indirect branch synchronization with compressed code message has the following format:

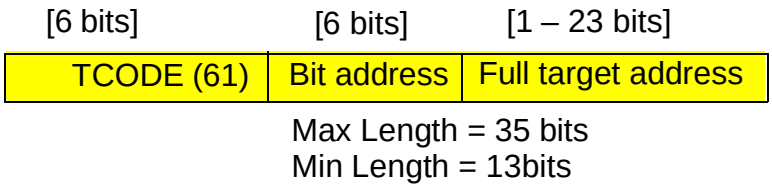


Figure 23-18 Indirect Branch Synchronization Message Format With Compressed Code

23.4.3.10 Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001 - 1999 recommendations, and is designed to reduce the number of bits transmitted for addresses of indirect branch messages.

The address transmitted is relative to the address of the previous branch trace message. It is generated by XORing the new address with the previous address, and then using only the results up to the most significant ‘1’ in the result. To recreate this address, an XOR of the (most-significant 0-padded) message address with the previously decoded address gives the current address. **Figure 23-19** shows how a relative address is generated and how it can be used to recreate the original address.

Previous Address (A1) = 0x0003 FC01, New Address (A2) = 0x0003 F365

Message Generation:

A1 = 0000 0000 0000 0011 1111 1100 0000 0001
A2 = 0000 0000 0000 0011 1111 0011 0110 0101

A1 ⊕ A2 = 0000 0000 0000 0000 0000 111 0110 0100

Address Message (M1) = 1111 0110 0100

Address Recreation:

A1 ⊕ M1 = A2

A1 = 0000 0000 0000 0011 1111 1100 0000 0001
⊕
M1 = 0000 0000 0000 0000 0000 1111 0110 0100

A2 = 0000 0000 0000 0011 1111 0011 0110 0101

Figure 23-19 Relative Address Generation and Re-creation

23.4.4 BTM Operation

23.4.4.1 BTM Capture and Encoding Algorithm

BTM is accomplished by capturing instruction fetch information from the U-bus and instruction execution information from the RCPU (VF and VFLS signals), and combining them to generate program trace messages.

23.4.4.2 Instruction Fetch Snooping

Instruction fetches are snooped on the U-bus. There is a one-to-one correspondence between instruction fetches marked with the U-bus program trace attribute and the indication of RCPU VF signal (only 3, 4, 5, and 6) between two synchronization events.

Since U-bus program trace attribute occurs after the indication of VF, it is latched and paired with the nearest (previous) unpaired VF (3, 4, 5, and 6) indication to determine the instruction address.

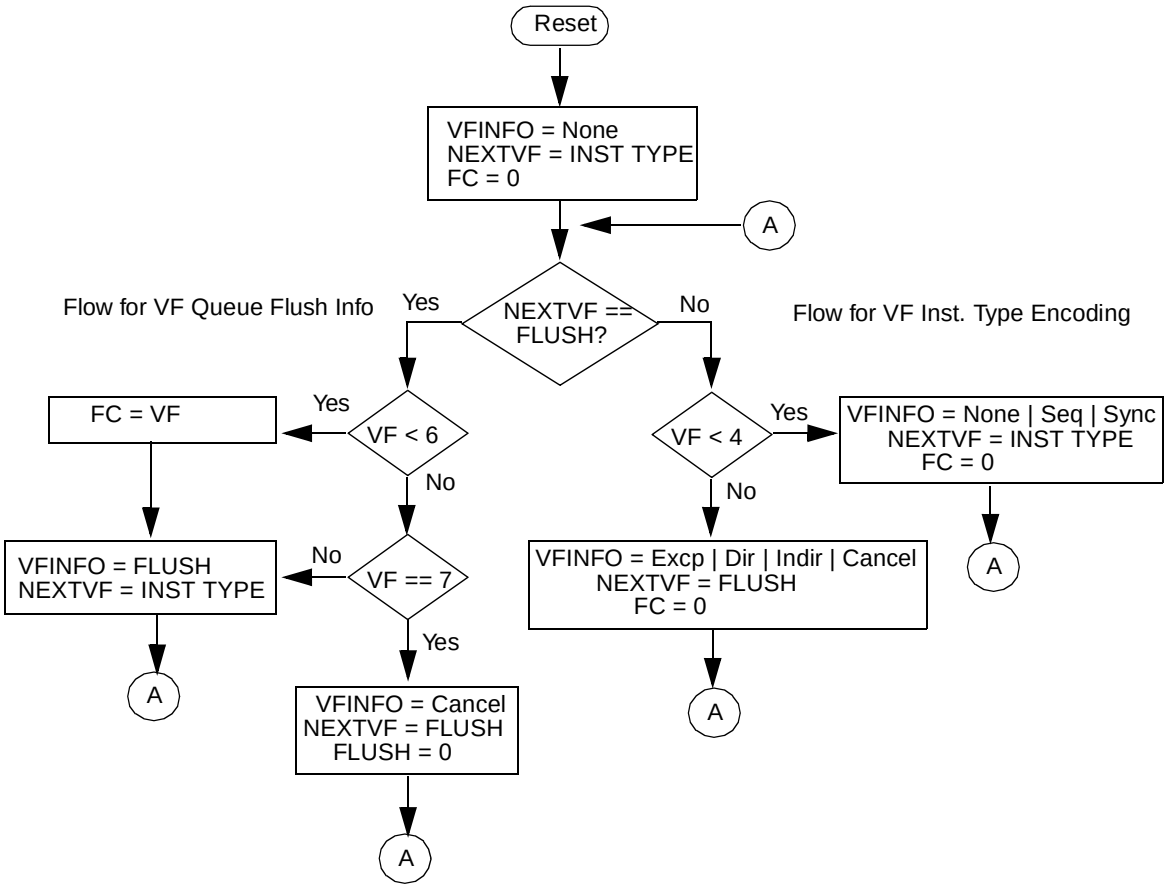
For all other VF indications, except 3, 4, 5, and 6, it is not possible to determine the instruction address.

23.4.4.3 Instruction Execution Tracking

Instruction execution tracking is performed by capturing the RCPU VF and VFLS signals, and decoding them to infer the state of the processor. The RCPU VF signals indicate two classifications of information:

- The current instruction type which is being loaded into the RCPU instruction queue. For further details refer to the ***RCPU Reference Manual* (RCPURM/AD)**.
- The number of instructions which are currently being flushed from the RCPU instruction queue. For further details refer to the ***RCPU Reference Manual* (RCPURM/AD)**.

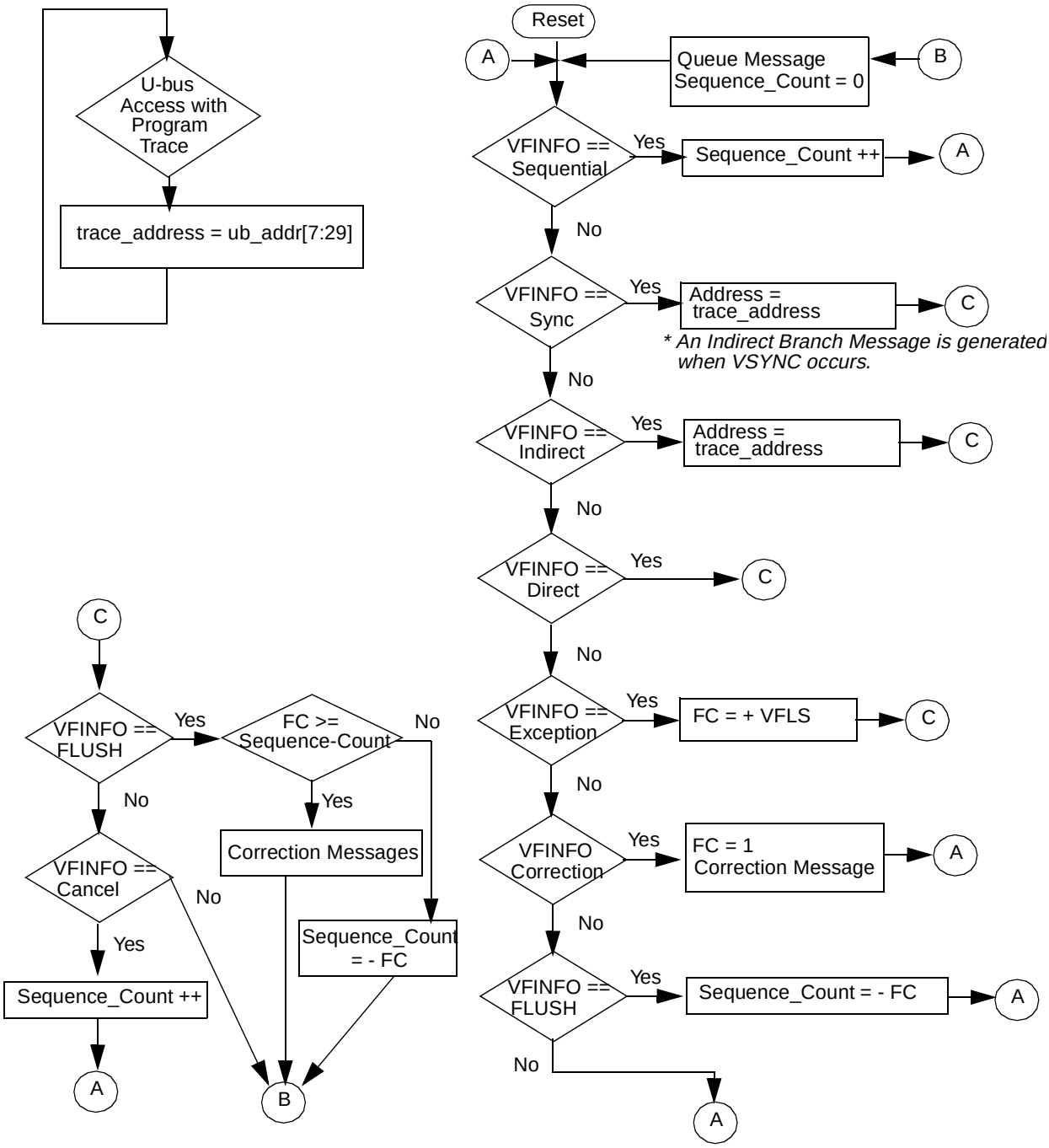
The algorithm to detect the information that is being indicated on the VF signals is depicted in **Figure 23-20**.



VFINFO is the type of information the RCPU VF signals indicate during this cycle
NEXTVF is the type of information the RCPU VF signals will indicate in the next cycle
FC is the number of instructions flushed from the prefetch queue

Figure 23-20 VF State Decoding

Based on the determination of the VF state decoding, **Figure 23-21** depicts the sequence for generating and queueing program trace messages.



VFINFO is the type of information the RCPU VF signals indicate during this cycle
NEXTVF is the type of information the RCPU VF signals will indicate in the next cycle
FLUSH is a state of RCPU VF signals.
VFLS indicates history buffer flush.
FC indicates the number of instructions to be flushed.

Figure 23-21 BTM Encoding Flow

NOTE

In **Figure 23-21** some operations require multiple cycles. The RCPU VFLS signals may take up to three clocks to indicate the full status of the history buffer flush (two instructions are flushed per clock).

For compressed code support, refer to **Figure 23-22** for further details.

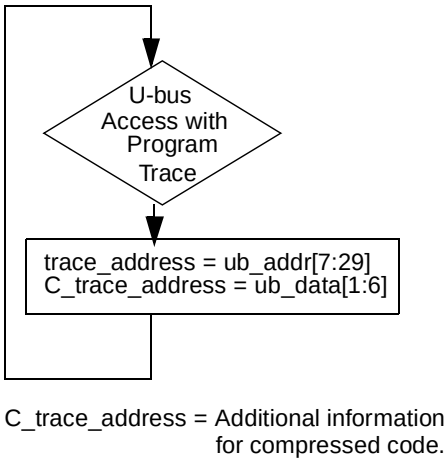


Figure 23-22 BTM Encoding Flow — Compressed Code Support

23.4.4.4 Instruction Flush Cases

The various conditions under which the RCPU may signal instruction flushes of the RCPU prefetch queue or RCPU history buffer are:

1. A taken branch (direct, indirect, interrupt or exception) will cause the instruction prefetch queue (which contains instructions from the now old stream) to be flushed, and fetching will start from the branch target stream. The sequential instruction count will be updated to reflect this.
2. A mispredicted branch will cause instructions fetched from the new stream to be flushed, and fetching will resume from the old stream. It will also require a program trace message to be cancelled and the trace to be corrected.
3. An exception can cause cancellation of multiple taken branches which may require cancelling multiple program trace messages.

23.4.5 BTM Queueing

READI implements a queue 16 message deep for program trace, data trace, and ownership trace messages. Messages that enter the queue are transmitted via the output auxiliary port in the order in which they are queued.

If multiple trace messages need to be queued at the same time, program trace messages will have the highest priority unless the data trace buffers are full, in which case the data trace messages are given temporary higher priority than the program trace messages.

23.4.6 Timing Diagram

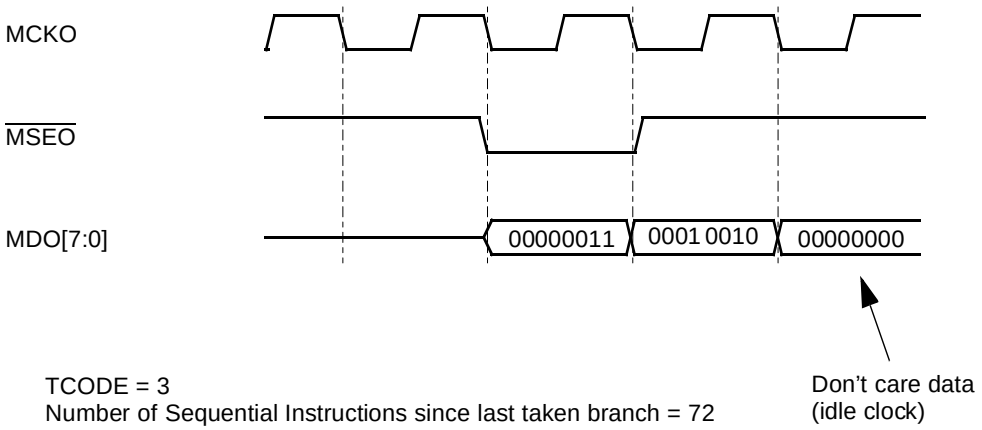


Figure 23-23 Direct Branch Message

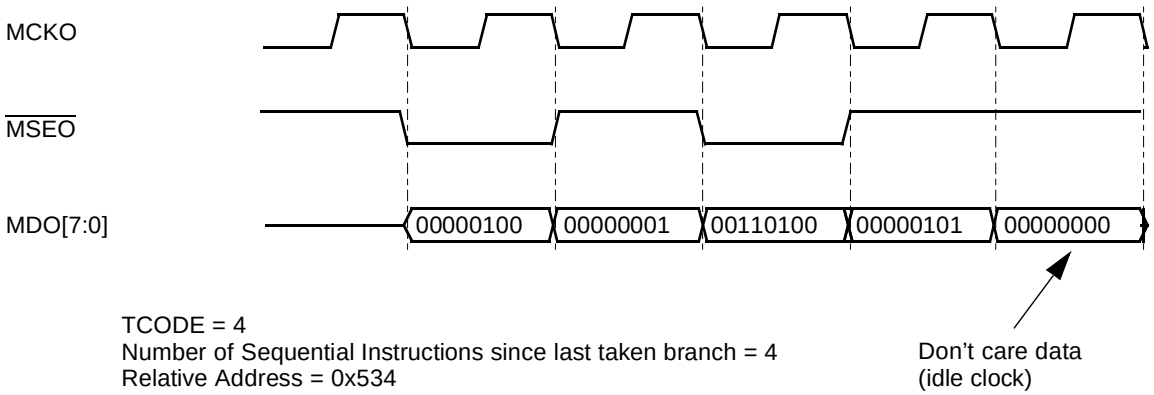


Figure 23-24 Indirect Branch Message

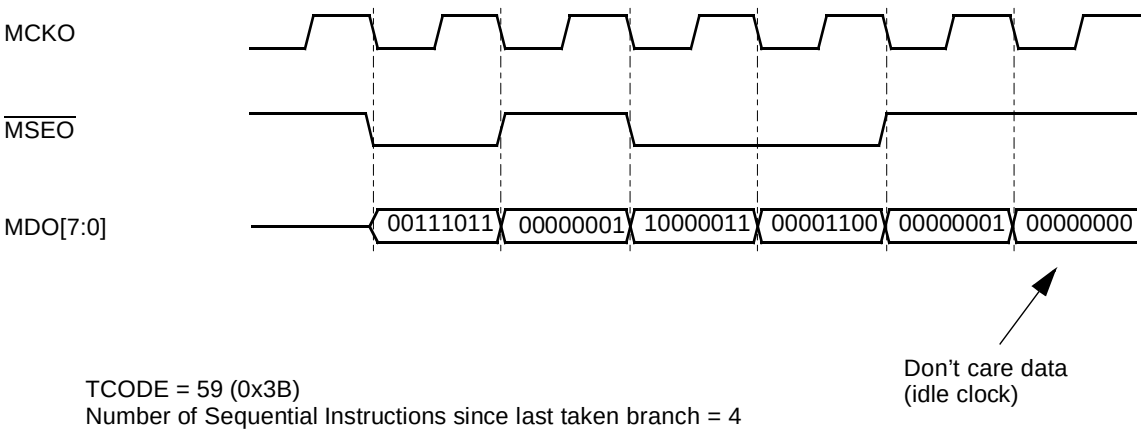


Figure 23-25 Indirect Branch Message With Compressed Code

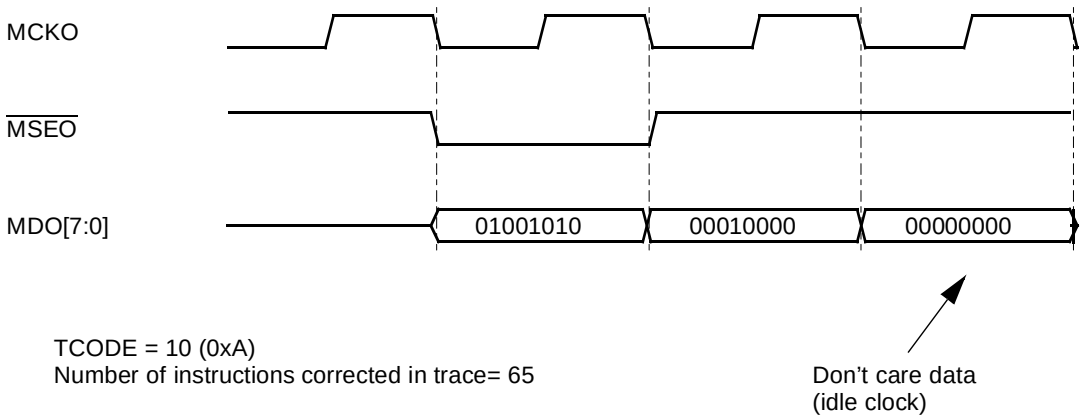


Figure 23-26 Program Trace Correction Message

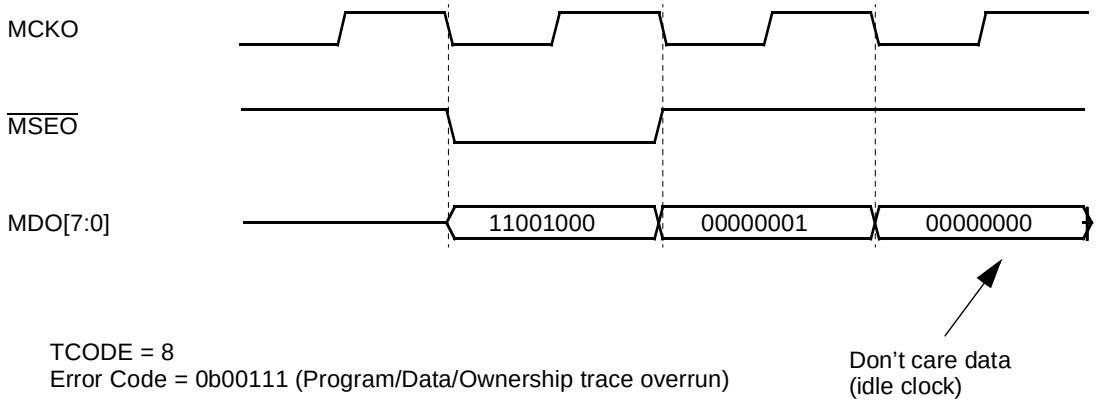


Figure 23-27 Error Message (Program/Data/Ownership Trace Overrun)

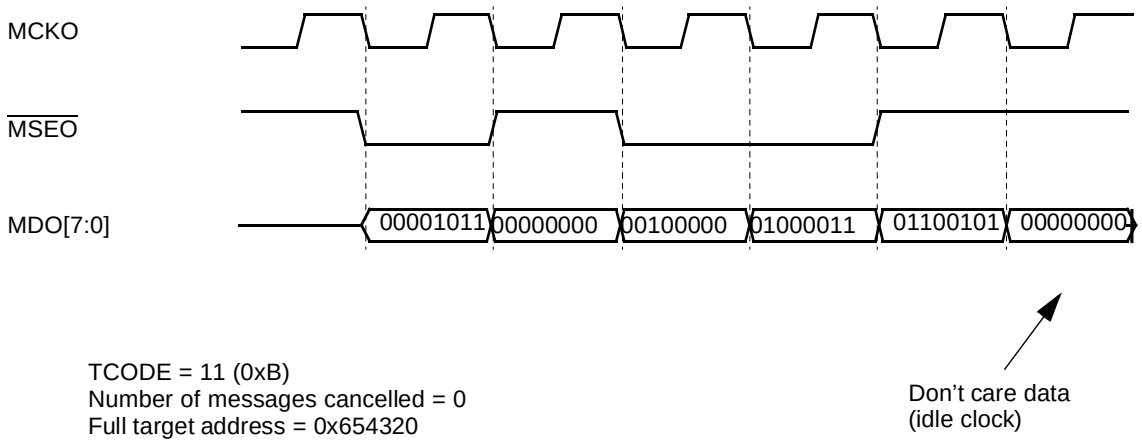


Figure 23-28 Direct Branch Synchronization Message

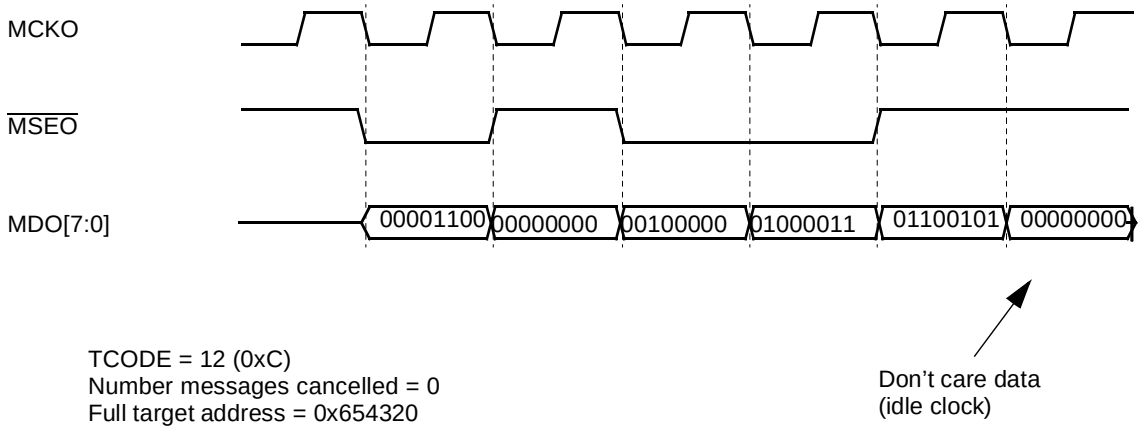


Figure 23-29 Indirect Branch Synchronization Message

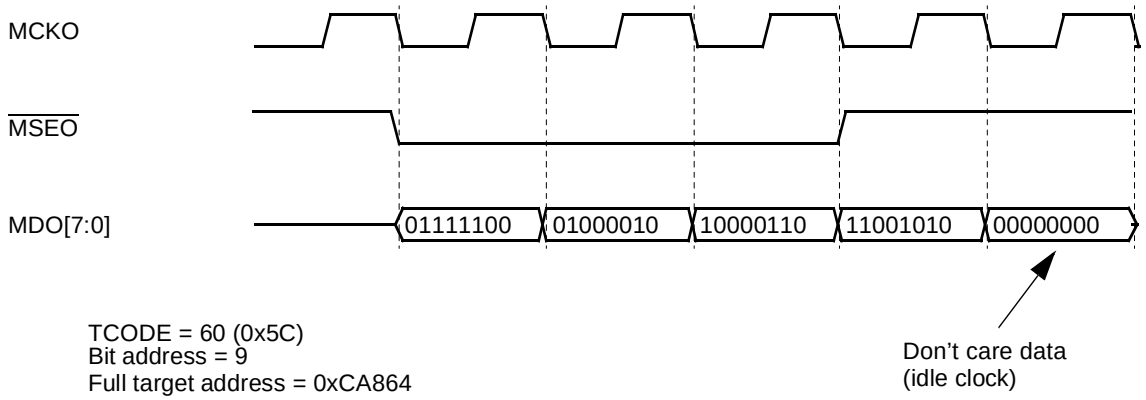


Figure 23-30 Direct Branch Synchronization Message With Compressed Code

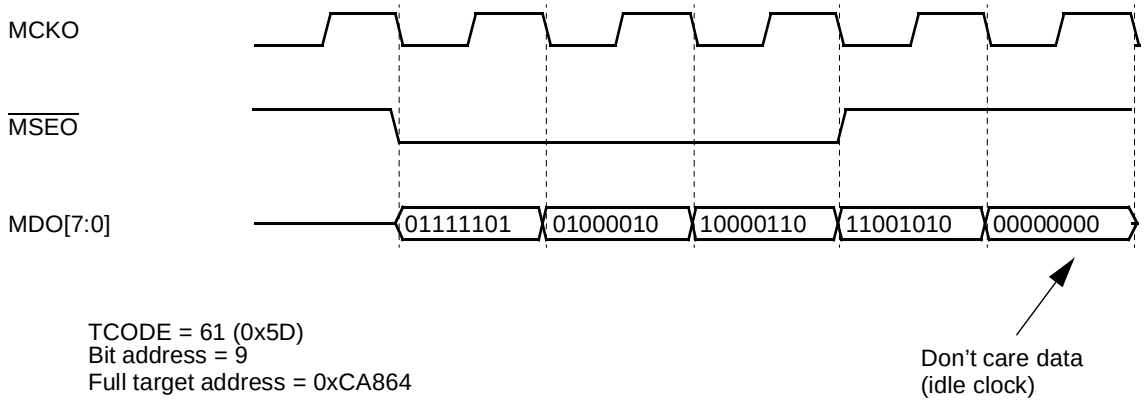


Figure 23-31 Indirect Branch Synchronization Message With Compressed Code

23.4.7 Program Trace Guidelines

Refer to **23.2.5.1 Program Trace Guidelines** for further details.

23.5 Data Trace

This section details the data trace mechanism supported by READI. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM), as per the IEEE-ISTO 5001 - 1999.

23.5.1 Data Trace for the Load/Store Bus (L-bus)

The L-bus allows the RCPU to perform loads and stores, and the L2U to read and write the L-bus resources. Snooping for data trace on the L-bus requires the READI module to handle the full range of L-bus cycles. This includes various cases of pipelining and aborted cycles.

Data trace requires snooping the L-bus cycles, and storing the information for qualifying accesses (based on enabled features and matching target addresses). The READI module traces all data accesses that meet the selected range and attributes. This includes all RCPU initiated accesses and all L-bus accesses.

L-bus data cycles can have data sizes of 8, 16 or 32 bits.The READI module supports all three data sizes.

23.5.2 Data Trace Message Formats

Data trace messages are of five types:

- Data write
- Data read
- Data write synchronization

- Data read synchronization
- Error message

23.5.2.1 Data Write Message

The data write message contains the data write value and the address of the target location, relative to the previous data trace message.

The data write message has the following format:

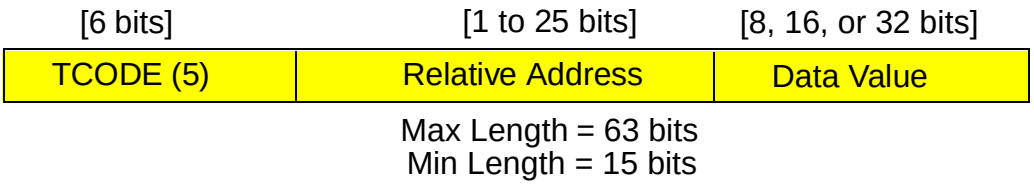


Figure 23-32 Data Write Message Format

23.5.2.2 Data Read Message

The data read message contains the data read value and the address of the target location, relative to the previous data trace message.

The data read message has the following format:

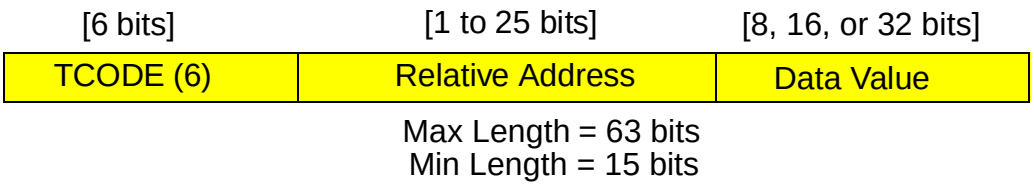


Figure 23-33 Data Read Message Format

23.5.2.3 Data Trace Synchronization Messages

A data trace synchronization message shall be transmitted via the auxiliary port (provided data trace is enabled) for the following conditions:

- Initial data trace message upon exit of any system reset will be a synchronization message.
- Upon exit of sleep, deep-sleep and low power down mode, the first data trace message will be a synchronization message.
- Initial data trace message upon exit of background debug mode. Upon exiting BDM, the next data trace message will be a synchronization message.
- When data trace is enabled, the first data trace message will be a synchronization message.

- After 255 data trace messages have been queued without synchronization, the next data trace message will be a synchronization message.
- Upon assertion of an event In ($\overline{\text{EVTI}}$) pin. If the READI module is not disabled at reset, when $\overline{\text{EVTI}}$ asserts, if the EC field is 0b00 in the DC register, the next data trace message will be a synchronization message.
- Upon occurrence of a watchpoint, the next data trace message will be a synchronization message.
- Occurrence of queue overrun. A data trace overrun error occurs when a trace message cannot be queued due to the queue being full (provided data trace is enabled). This causes the message queue to be flushed, and an error message is placed as the first message in the queue. The error code within the Error message indicates that program/data/ownership trace overrun has occurred. The next data trace message will be a synchronization message.

Data trace synchronization messages provide the full address (without leading zeros) and ensure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent DTMs, in which only the unique portion of the data trace address is transmitted.

Data trace synchronization messages are of two types:

- Data write
- Data read

23.5.2.4 Data Write Synchronization Message

The data write synchronization message has the following format:

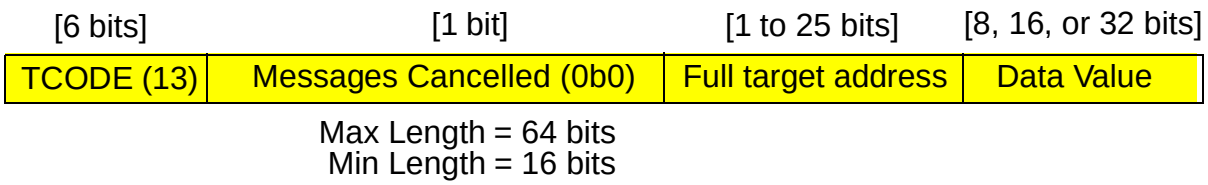


Figure 23-34 Data Write Synchronization Message Format

23.5.2.5 Data Read Synchronization Messaging

The data read synchronization message has the following format:

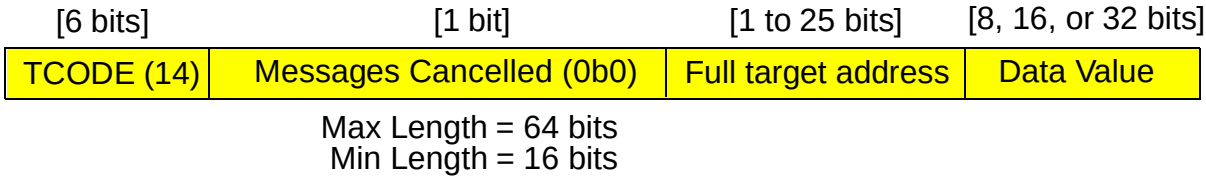


Figure 23-35 Data Read Synchronization Message Format

23.5.2.6 Relative Addressing

Refer to **23.5.2.6 Relative Addressing** for further details.

23.5.3 Error Message (Queue Overflow)

A program/data/ownership trace overrun error occurs when a trace message cannot be queued due to the queue being full, provided data trace is enabled.

The overrun error causes the message queue to be flushed, and a error message to be queued. The error code within the error message indicates that a program/data/ownership trace overrun error has occurred. The next DTM will be a synchronization message.

The error message has the following format:

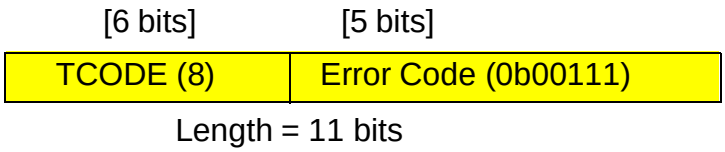


Figure 23-36 Error Message (Queue Overflow) Format

23.5.4 Data Trace Operation

Data trace is performed by snooping the L-bus for read or write cycles. Data trace functions are enabled by setting the appropriate fields in the DC register and the DTA registers. For details on field configuration, refer to **23.2.1.4 Development Control (DC) Register** and **23.2.1.9 Data Trace Attributes 1 and 2 (DTA1 and DTA2) Registers** respectively.

Data trace flow is depicted in **Figure 23-37**.

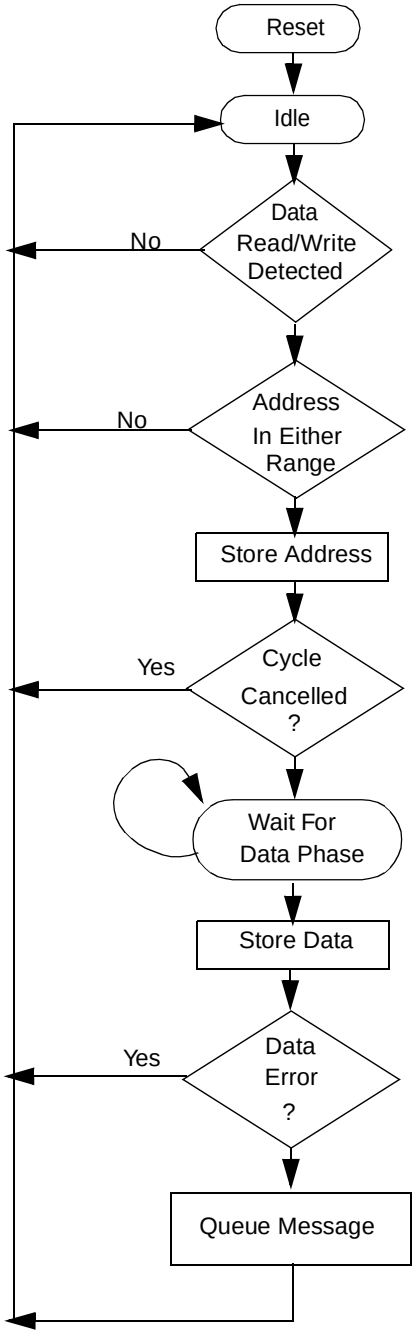


Figure 23-37 Data Trace Flow Diagram for Non-Pipelined Access

23.5.5 Data Trace Windowing

Data trace windowing is achieved via the address range within the DTEA and the DTSA fields of the DTA registers. All L-bus accesses which fall within these two address ranges, provided the address ranges are enabled in either DTA register, are candidates to be transmitted. Data read and/or data write trace may be enabled via the TA field of the data trace attributes registers (DTA).

NOTE 1

Data trace ranges are word aligned. Therefore, the address range fields (DTEA and DTSA) of the DTA registers are only 23 bits wide and, as such, should be assigned by the tool with the 23 most significant bits of the intended 25-bit range address.

NOTE 2

The off-core PowerPC special purpose register (SPR) map cannot be distinguished from the normal memory map accesses via the defined address range control. If data trace ranges are set up such that the off-core PowerPC SPR map falls within active ranges, then accesses to these off-core PowerPC SPRs will be traced, and the messages will not be distinguishable from accesses to normal memory map space. Off-core PowerPC SPRs typically exist in the 8-Kbyte – 16-Kbyte lowest memory block (0x2000 - 0x3FF0). If data or peripherals are mapped to this space, load/stores to PowerPC SPRs will be indistinguishable from data or peripheral accesses.

23.5.6 Special L-bus Cases

Special L-bus cases are handled as described in **Table 23-32**.

Table 23-32 Special L-bus Case Handling

Special Case	Action
L-bus Cycle Aborted	Cycle ignored
L-bus Cycle with data error	Message discarded
L-bus Cycle terminated due to address error	Cycle ignored
L-bus Cycle completed without error	Cycle captured and transmitted
L-bus Cycle initiated by READI (Read/Write Access)	Cycle ignored
L-bus Cycle is an instruction fetch	Cycle ignored
Data Storage Interrupt	Cycle ignored
System Rese	Cycle ignored

23.5.7 Data Trace Queuing

READI implements a queue 16 messages deep for queuing program trace, data trace, and ownership trace messages. Messages that enter the queue are transmitted via the output auxiliary port in the order in which they are queued.

NOTE

If multiple trace messages need to be queued at the same time, program trace messages have a higher priority for queue entry than data trace messages, unless the data trace buffers are full, in which case the data trace messages are given temporary higher priority than the program trace messages.

23.5.8 Throughput and Latency

23.5.8.1 Assumptions for Throughput Analysis

- All accesses are data trace only
- 56-MHz operation
- Output pins are always free (not in middle of transmission) when requested
- Relative Address field for data trace messages is 20 bits
- Data field for data trace messages is 32 bits
- One idle clock between data trace messages

23.5.8.2 Throughput Calculations

The data (read or write) trace message is 58 bits (6 [TCODE] + 20 [Relative address] + 32 [Data]).

Data trace messages are transmitted out via the MDO pins. Hence it will take eight clocks (58 bits/8 MDO pins) to send a message. There will be one idle clock before the next data trace message can be sent.

At 56 MHz, it will take 161ns ((8+1) x 17.8) to transmit the message.

Therefore, the average number of data trace messages that can be transmitted out is 6.2 million (1/161ns) per second, or 24.8 million bytes of read/write data per second.

23.5.9 Timing Diagrams

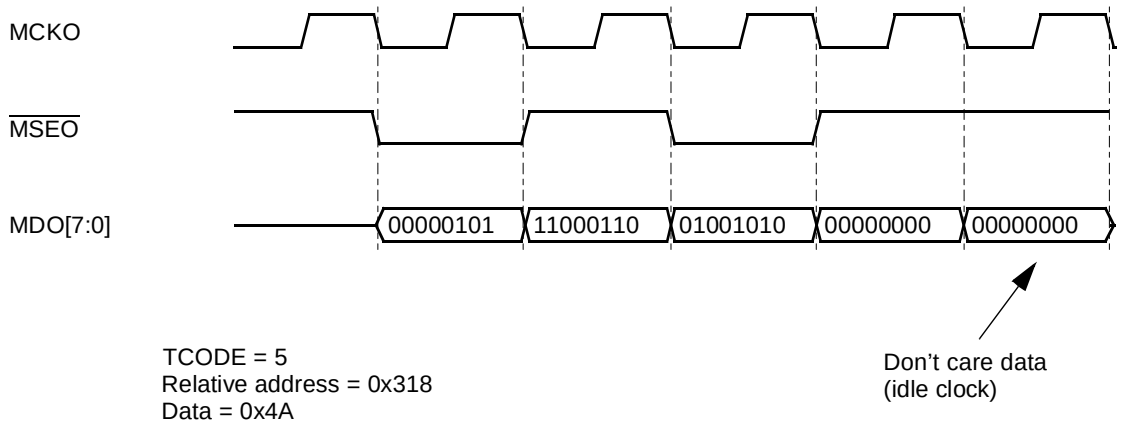


Figure 23-38 Date Write Message

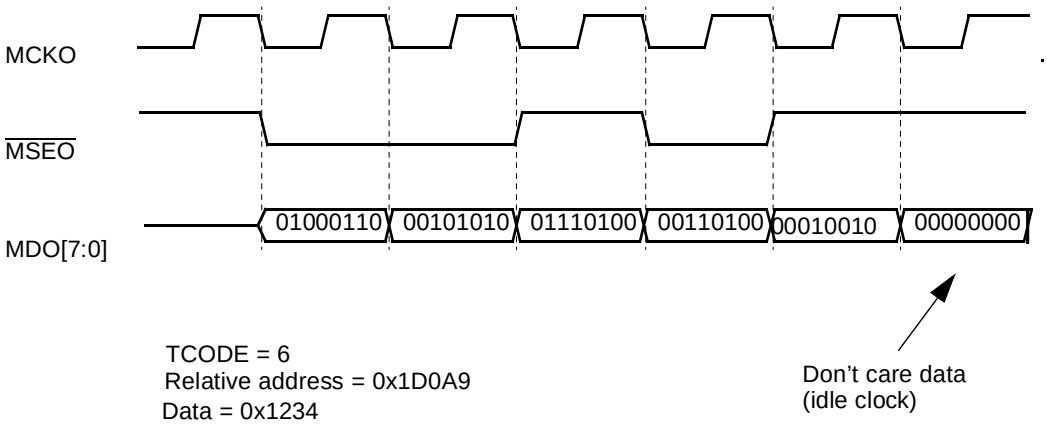


Figure 23-39 Data Read Message

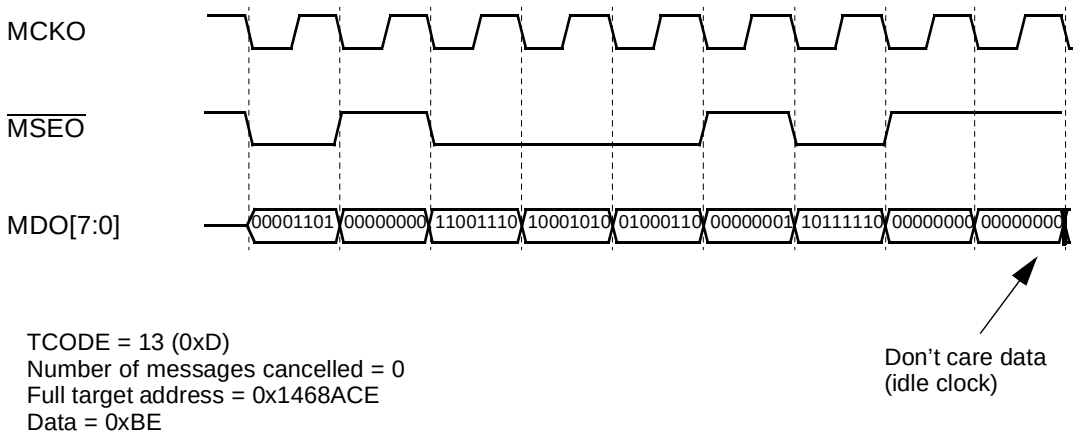


Figure 23-40 Data Write Synchronization Message

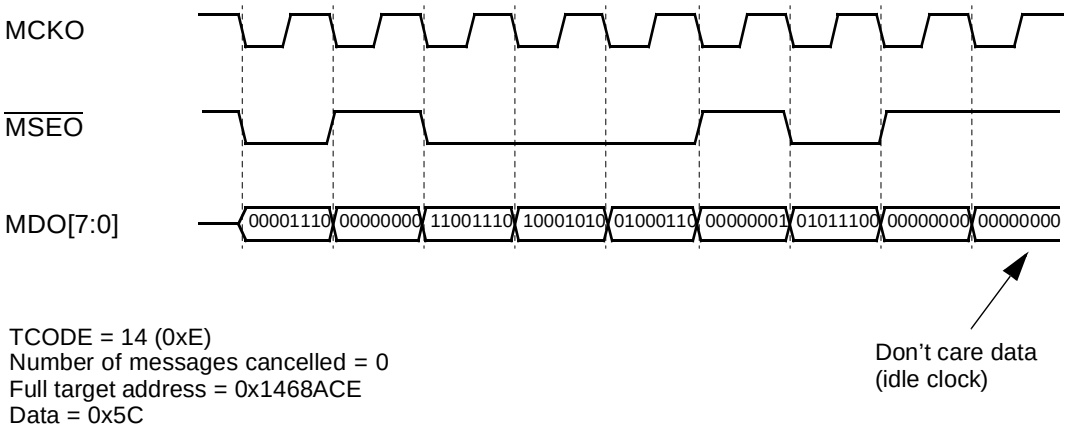


Figure 23-41 Data Read Synchronization Message

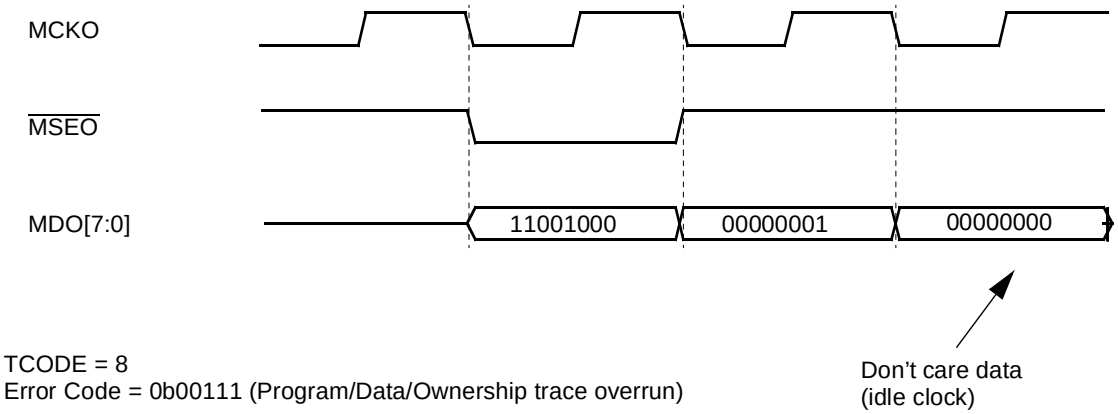


Figure 23-42 Error Message (Program/Data/Ownership Trace Overrun)

23.6 Read/Write Access

The read/write access feature allows access to internal memory mapped space via the auxiliary port. Read/write mechanism supports single and block, reads and writes.

23.6.1 Functional Description

The READI module is capable of bus mastership on the L-bus and for setting up and reading data and status.

All accesses are setup and initiated to the read/write access register (RWA) and upload/download information register (UDI) via the four auxiliary access public messages: device ready for upload/download, upload request (tool requests information),

download request (tool provides information), upload/download information (device/ tool provides information).

Read/write access features are enabled by setting the appropriate fields in the RWA register. For details on field configuration, refer to **23.2.1.7 Read/Write Access (RWA) Register**.

The functional flow for read/write access to memory mapped locations and PowerPC registers is depicted in **Figure 23-43**.

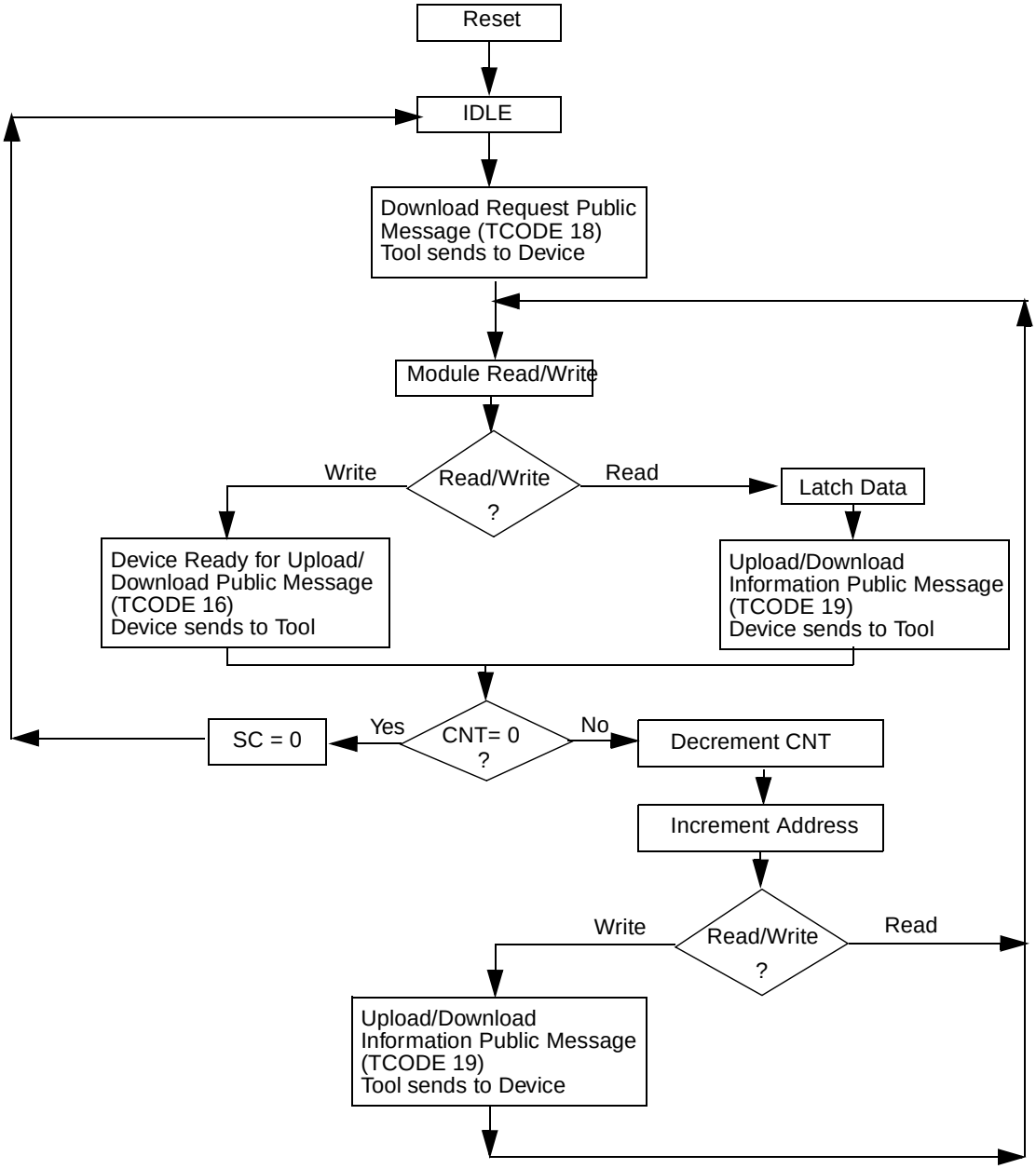


Figure 23-43 Read/Write Access Flow Diagram

23.6.2 Write Operation to Memory Mapped Locations and PowerPC Registers

23.6.2.1 Single Write Operation

For a single write access to memory mapped locations and PowerPC registers, the following sequence of operations need to be performed via the auxiliary port:

1. The tool confirms that the device is ready before transmitting download request public message (TCODE=18).
2. The download request public message contains:
   a. TCODE(18)
   b. Access opcode 0xF which signals that subsequent data needs to be stored in the RWA register.
   c. Configure the RWA register fields as follows:
      • Start/complete (1 to indicate start access) -> SC
      • Read/write address (write address) -> RWAD
      • Read/write (1 to indicate a write access) -> RW
      • Word size (32 bits, 16 bits, 8 bits) -> SZ
      • Write data (write data) -> WD
      • Privilege (user data/instruction, supervisor data/instruction) -> PRV
      • Map select (select memory map, 00 or 01) -> MAP
      • Access Count (0 to indicate single access) -> CNT
3. After completion of the write operation, the device ready for upload/download public message (TCODE=16) is transmitted to the tool indicating that the device is ready for next access.
4. The SC bit is cleared to indicate that the write access is complete.

23.6.2.2 Block Write Operation

For a block write access to memory mapped locations and PowerPC registers, the following sequence of operations need to be performed via the auxiliary port:

1. The tool confirms that the device is ready before transmitting download request public message (TCODE = 18).
2. The download request public message contains:
   a. TCODE(18)
   b. Access opcode 0xF which signals that subsequent data needs to be stored in the RWA register.
   c. Configure the RWA register fields as follows
      • Start/complete (1 to indicate start access) -> SC
      • Read/write address (starting write address of block) -> RWAD
      • Read/write (1 to indicate a write access) -> RW
      • Word size (32 bits, 16 bits, 8 bits) -> SZ
      • Write data (write data) -> WD
      • Privilege (user data/instruction, supervisor data/instruction) -> PRV
      • Map select (select memory map, 00 or 01) -> MAP

- Access count (non zero number to indicate size of block access) -> CNT

3. After completion of this write operation, the device ready for upload/download public message (TCODE = 16) is transmitted to the tool indicating that the device is ready for next access.
4. The specified address (stored in RWAD field) is incremented to the next word size and the number in the CNT field is decremented. The SC field is not cleared.
5. The tool transmits the next upload/download information public message (TCODE = 19).
6. The upload/download information public message contains:

   a. TCODE(19)
   b. Write data (write data -> UDI)

7. After the completion of this write operation, the device ready for upload/download public message (TCODE = 16) is transmitted to the tool indicating that the device is ready for next access.
8. The specified address (in RWAD field) is incremented to the next word size and the number in the CNT field is decremented. The SC field is not cleared.
9. Steps 5 through 8 are repeated until the count value in the CNT field of RWA register equals zero. The SC bit is cleared to indicate end of the block write access.

NOTE

For downloading write data to the device for block write operation, the download request public message (TCODE = 18) should not be used to write subsequent data to the UDI register. Data written to the UDI register (via download request message, TCODE 18) is not used by the device for any read/write operation.

23.6.3 Read Operation to Memory Mapped Locations and PowerPC Registers

23.6.3.1 Single Read Operation

For a single read access to memory mapped locations and PowerPC registers, the following sequence of operations need to be performed via the auxiliary port:

1. The tool confirms that the device is ready before transmitting download request public message (TCODE = 18).
2. The download request public message contains:

   a. TCODE(18)
   b. Access opcode 0xF which signals that subsequent data needs to be stored in the RWA register.
   c. Configure the RWA fields as follows:
   - Start/complete (1 to indicate start access) -> SC
   - Read/write address (read address) -> RWAD

- Read/write (0 to indicate a read access) -> RW
- Word size (32 bits, 16 bits, 8 bits) -> SZ
- Write data (0xXXXXXXXX-> WD [don't care])
- Privilege (user data/instruction, supervisor data/instruction) > PRV
- Map select (select memory map, 00 or 01) -> MAP
- Access count (0 to indicate single access) -> CNT

3. Data read from the specified address is stored in the UDI register.
4. Once the read access is completed, the upload/download information public message (TCODE = 19) is transmitted to the tool along with the data read from the UDI register. This message also indicates that the device is ready for next access.
5. The SC field in the RWA register is cleared.

23.6.3.2 Block Read Operation

For a block read access to memory mapped locations and PowerPC registers, the following sequence of operations need to be performed via the auxiliary port:

1. The tool confirms that the device is ready before transmitting download request public message (TCODE = 18).
2. The download request public message contains:
   a. TCODE(18)
   b. Access opcode 0xF which signals that subsequent data needs to be stored in the RWA register.
   c. Configure the RWA fields as follows:
      - Start/complete (1 to indicate start access) -> SC
      - Read/write address (starting read address of block) -> RWAD
      - Read/write (0 to indicate a read access) -> RW
      - Word size (32 bits, 16 bits, 8 bits) -> SZ
      - Write data (0xXXXXXXXX-> WD [don't care])
      - Privilege (user data/instruction, supervisor data/instruction) > PRV
      - Map select (select memory map, 00 or 01) -> MAP
      - Access count (non-zero number to indicate block access) -> CNT
3. Data read from the specified address is stored in the UDI register.
4. After the completion of this read operation, the upload/download information public message (TCODE=19) is transmitted to the tool along with the data read from the UDI register. This message also indicates that the device is ready to perform the next read operation.
5. The specified address (in RWAD field) is incremented to the next word size and the number in the CNT field is decremented. The SC field is not cleared.
6. The data read from the new address is stored in the UDI register.
7. Steps 4 through 7 are repeated until the count value in the CNT field of RWA register equals zero. The SC bit is cleared to indicate end of the block read access.

23.6.4 Read/Write Access to Internal READI Registers

23.6.4.1 Write Operation

For a write access to internal READI registers, the following sequence of operations need to be performed via the auxiliary port:

1. The tool confirms that the device is ready before transmitting download request public message (TCODE = 18).
2. The download request public message contains:
   a. TCODE(18)
   b. Access opcode, which specifies the register where data needs to be written, (e.g., access opcode 0x14 indicates that DTA1 register is the target register).
   c. Data to be written to the register.
3. After the data has been written to the targeted register, the device ready for upload/download public message (TCODE = 16) is transmitted to the tool indicating that the device is ready for next access.

23.6.4.2 Read Operation

For a read access to internal READI registers, the following sequence of operations need to be performed via the auxiliary port:

1. The tool confirms that the device is ready before transmitting upload request public message (TCODE = 17).
2. The upload request public message contains:
   a. TCODE(17)
   b. Access opcode, which specifies the register where data needs to be read from, (e.g., access opcode 0x14 indicates that DTA1 register is the target register).
3. The upload/download information public message (TCODE =19) is transmitted to the tool along with the data read from the targeted register indicating that the device is ready for next access.

23.6.5 Error Handling

The READI module handles the various error conditions in the manner shown in the following sections.

23.6.5.1 Access Alignment

The READI module will force address alignment based on the word size field (SZ) value. If the SZ field indicates word (32-bit) access, the least significant two bits of the read/write address field (RWAD) are ignored. If the SZ field indicates half-word (16-bit) access, the least significant bit of the read/write address field (RWAD) is ignored.

23.6.5.2 L-bus Address Error

An address error occurs on the L-bus when the address phase of a cycle is not completed normally. This could occur because of address not being valid or the address map not being valid. In such cases:

1. The access is terminated without retrying.
2. The SC bit of the RWA is reset. Block accesses do not continue.
3. The error message (TCODE = 8) is transmitted (error code 0b00011).

23.6.5.3 L-bus Data Error

L-bus data error is signalled due to:

- L-bus data phase error.
- U-bus address phase error (for a L-bus to U-bus cycle).
- U-bus data phase error (for a L-bus to U-bus cycle).

L-bus data error conditions are signalled along with the transfer acknowledge for the access. L-bus data error conditions may occur because of privilege violations, access to protected memory, etc. In such cases, for a read access, the ERR bit of the UDI is set, and the DV bit in the UDI is reset at the termination of the access. For a write access, an error public message (TCODE = 8) is transmitted (error code 0b00011).

23.6.6 Exception Sequences

The following cases are defined for sequences of the read/write protocol that differ from those described in the above sections:

1. If the SC bit is set to start READI read/write accesses, without valid values in the RWAD, then an L-bus address error may occur, which is handled as described above.
2. If a block access is in progress with all the cycles not yet completed, and the RWA is written to again, (with or without modifications), then the block access is terminated at the boundary of the nearest completed access. The resulting data is discarded and not written to the UDI. If a new access has been programmed in the RWA register, then that access will start once the controller has recovered.
3. When a block access is in progress with all the cycles not yet completed, writing the SC bit to 0 in RWA register will terminate the block access and device will send out device ready for upload/download message.
4. If a any type (single/block) of access is in progress with the cycles not yet completed, and system reset occurs, the device will send out an error message. The access will be terminated and the SC bit will be reset.
5. If any type of (single/block) of access is requested while system is in reset, the device will send out an error message. The access will not be started and the SC bit will be reset.

23.6.7 Secure Mode

For details refer to **23.1.4.2 Security**.

23.6.8 Error Messages

23.6.8.1 Read/Write Access Error

An error message is sent out when an L-bus access error or data error on a write access occurs. The error code within the error message indicates that an L-bus address or L-bus data error occurred. For other error handling, see **23.6.5 Error Handling**.

The error message has the following format:

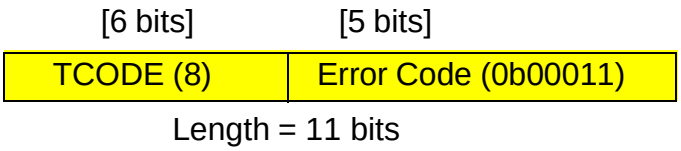


Figure 23-44 Error Message (Read/Write Access Error) Format

23.6.8.2 Invalid Message

An error message is sent out when an invalid message is received by READI. The error code within the error message indicates that an invalid TCODE was detected in the auxiliary input messages by the pin input formatter.

The error message has the following format:

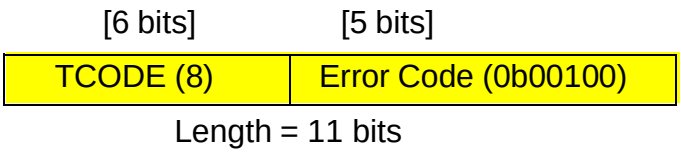


Figure 23-45 Error Message (Invalid Message) Format

NOTE

If the TCODE is valid, then READI will expect that the correct number of packets have been received and no further checking will be performed. If the number of packets received by READI is not correct, READI response is not deterministic.

23.6.8.3 Invalid Access Opcode

An error message is sent out when an invalid access opcode is received by READI. The error code within the error message indicates that an invalid access opcode was detected in the auxiliary input messages by the pin input formatter.

The error message has the following format:

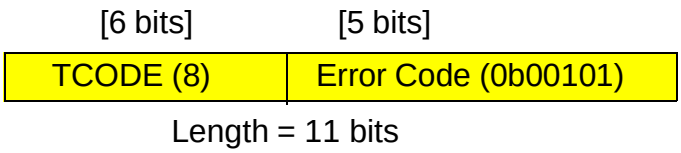


Figure 23-46 Error Message (Invalid Access Opcode) Format

23.6.9 Faster Read/Write Accesses With Default Attributes

Read/write access throughput may be increased by taking advantage of the default settings of the RWA register, and truncating the least significant zero bits of the download request message. For example, to read a word from the default memory map, with default attributes, a download request message that selects the RWA register, and transmits the SC, RWAD, RW fields only is sufficient. This message will contain 41 bits instead of the 94 bits for writing the full contents of the RWA register. See **Table 23-11** and **23.2.4 Partial Register Updates** for RWAR and partial register update details respectively.

NOTE

The last data bit transmitted in the download request message (TCODE 18) will always be the MSB of the register referenced by the opcode (SC field in the case of the RWA register).

23.6.10 Throughput and Latency

Throughput analysis has been performed for various read/write access cases such as single write, block write, single byte read, single word read, block byte read, block word read accesses to memory mapped locations. Data is presented for the two cases when the RWA register is written partially and completely.

23.6.10.1 Assumptions for Throughput Analysis

- All accesses are single read accesses only.
- MCKI running at 28 MHz.
- MCKO running at 56 MHz.
- 56 MHz internal operation.
- Five clock internal L-bus access (read)
- Output pins always free (not in middle of transmission) when requested.
- One idle clock between read messages.
- No delay from tool in responding — tool keeps up with READI port.

Table 23-33 Throughput Comparison for FPM and RPM MDO/MDI Configurations

Access Type	Reduced Port Mode 2 MDO / 1 MDI pins		Full Port Mode 8 MDO / 2 MDI pins	
	Full RWAR Update	Partial RWAR Update	Full RWAR Update	Partial RWAR Update
Single Write Access to memory mapped location – Word and Byte access (In Million Messages Per Second)	0.28	0.35	0.53	0.65
Single Read Access to memory mapped location – Word access (In Million Messages Per Second)	0.25	0.51	0.52	1.05
Single Read Access to memory mapped location – Byte access (In Million Messages Per Second)	0.27	0.56	0.53	1.05
Block Write Access to memory mapped locations – 64-Kbyte block (Word and Byte) write access (In 64-Kbyte Block Writes Per Second)	9	9	17	17
Block Read Access to memory mapped locations – 64-Kbyte block (Word) read access (In 64-Kbyte Block Writes Per Second)	32	32	77	77
Block Read Access to memory mapped locations – 64-Kbyte block (Byte) read access (In 64-Kbyte Block Writes Per Second)	61	61	95	95

23.7 Timing Diagrams

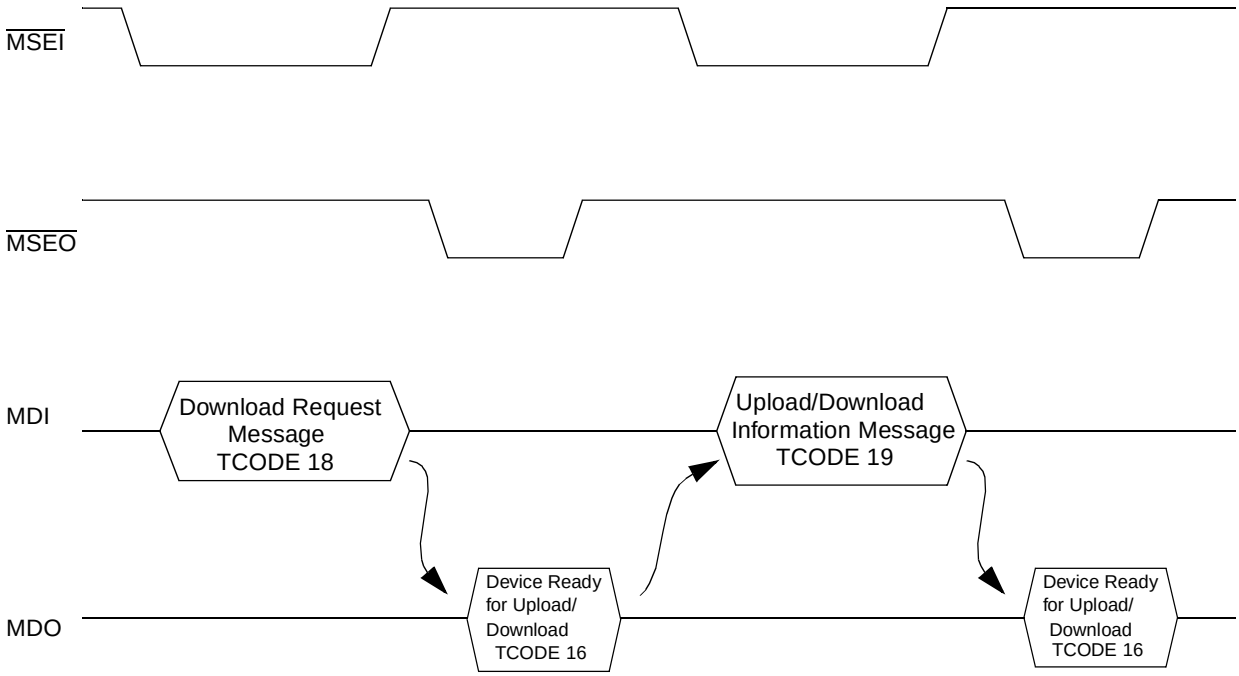


Figure 23-47 Block Write Access

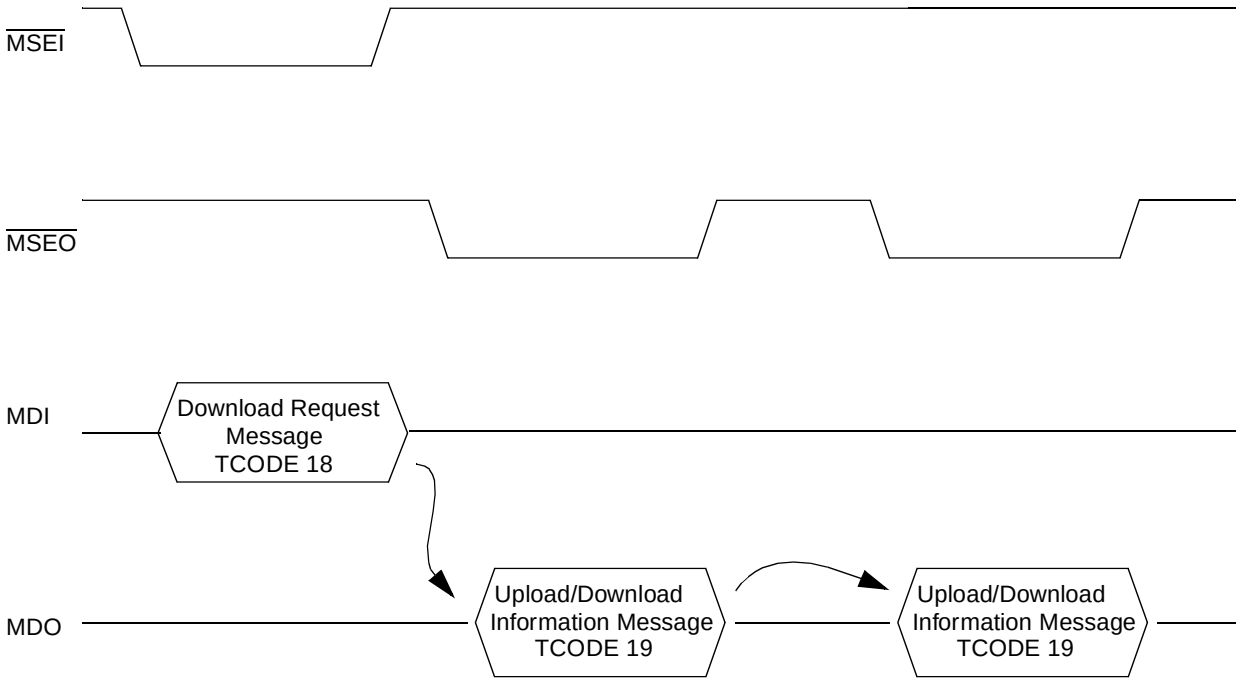


Figure 23-48 Block Read Access

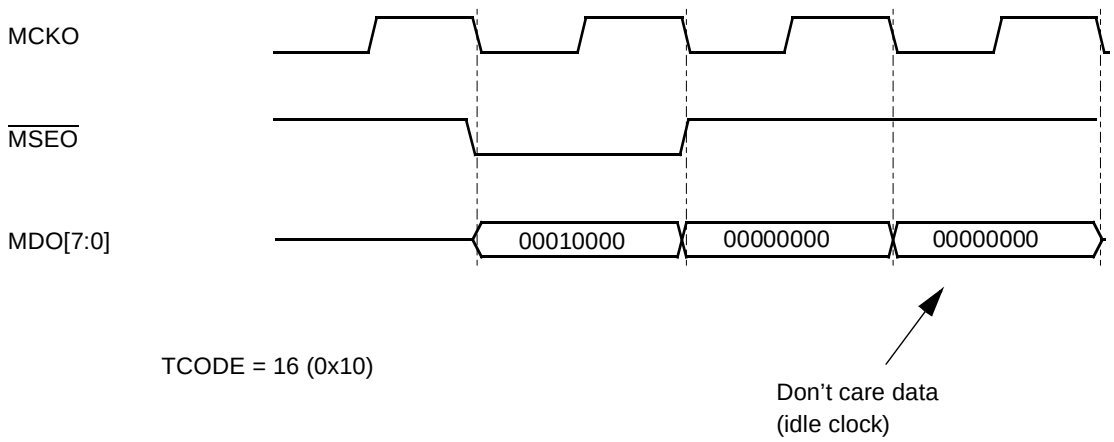


Figure 23-49 Device Ready for Upload/Download Request Message

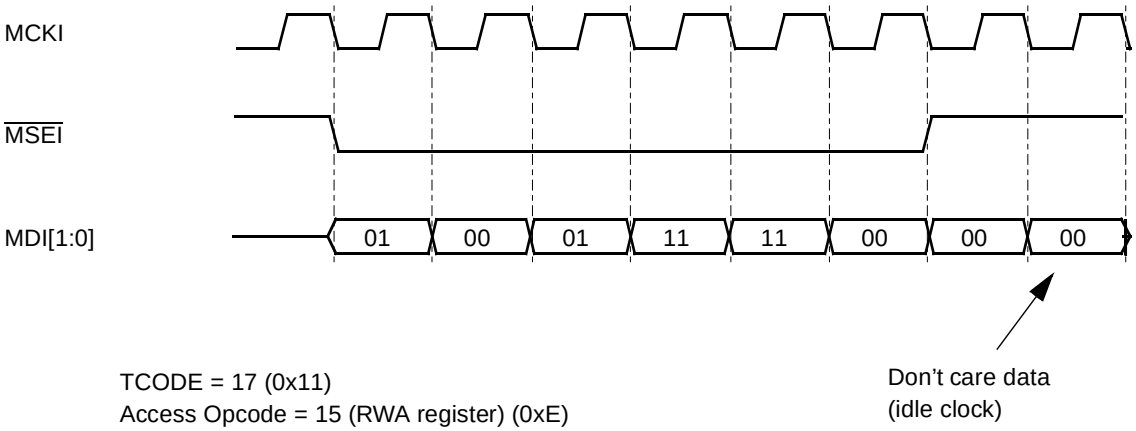


Figure 23-50 Upload Request Message

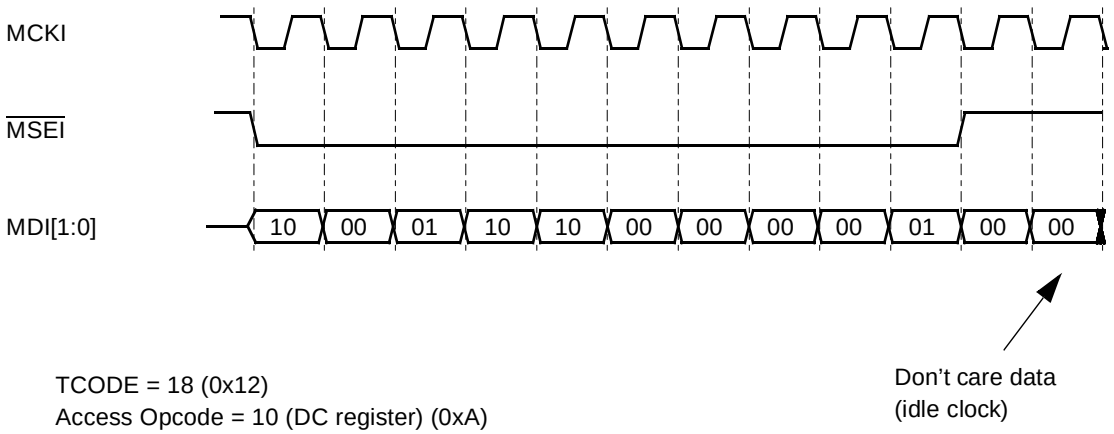


Figure 23-51 Download Request Message

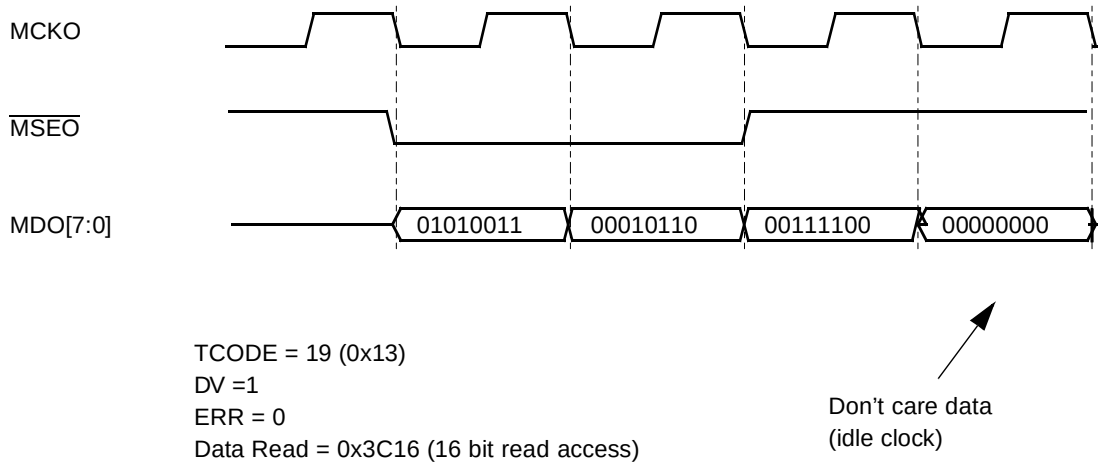


Figure 23-52 Upload/Download Information Message

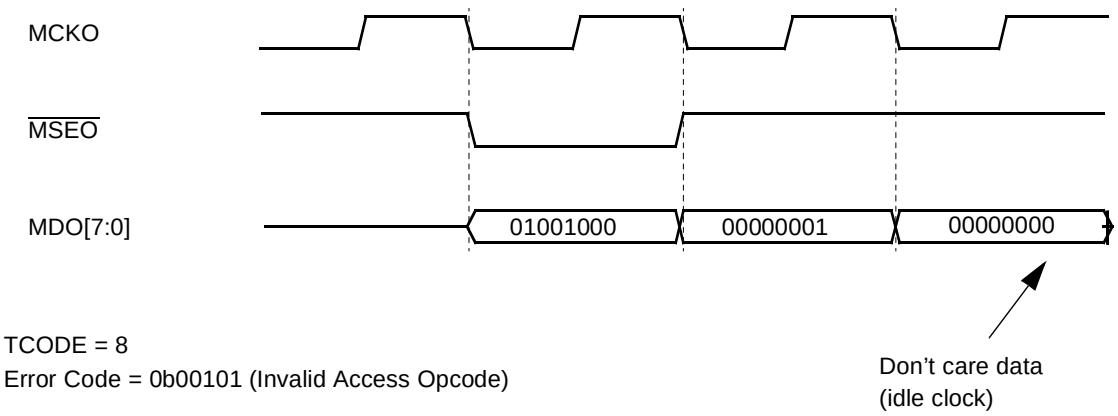


Figure 23-53 Error Message (Invalid Access Opcode)

23.8 Watchpoint Support

This section details the watchpoint support features of the READI module.

The READI module provides watchpoint messaging via the auxiliary port, as defined by the IEEE-ISTO 5001 - 1999.

READI is not compliant with all the breakpoint/watchpoint requirements defined in the IEEE-ISTO 5001 standard. Watchpoint trigger and breakpoint/watchpoint control registers are not implemented.

Watchpoint setting via READI can only be done using the BDM protocol.

23.8.1 Watchpoint Messaging

The READI module provides watchpoint messaging using IEEE-ISTO 5001 - 1999 defined public messages. The watchpoint status signals from the RCPU are snooped, and when watchpoints occur, a message is sent to the pin output formatter to be messaged out (the general message queue is bypassed to prevent watchpoint messages from being cancelled in the event of a queue overflow). The watchpoint message has the second highest priority. Refer to **23.3.2.1 Message Priority** for further details on message priorities. The watchpoint message contains the watchpoint code which indicates all the unique watchpoints have occurred since the last watchpoint message. If duplicate watchpoints occur before the watchpoint message is sent out, a watchpoint overrun message is generated. The watchpoint source field will indicate which watchpoints occurred.

The watchpoint message has the following format:

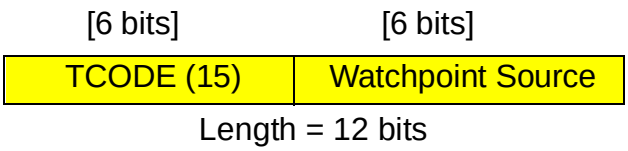


Figure 23-54 Watchpoint Message Format

23.8.1.1 Watchpoint Source Field

The Watchpoint Source field is outlined in **Table 23-34**.

Table 23-34 Watchpoint Source

Watchpoint Source	Description
0bXXXXX1	First L-bus watchpoint (LW0)
0bXXXX1X	Second L-bus watchpoint (LW1)
0bXXX1XX	First I-bus watchpoint (IW0)
0bXX1XXX	Second I-bus watchpoint (IW1)
0bX1XXXX	Third I-bus watchpoint (IW2)
0b1XXXXX	Fourth I-bus watchpoint (IW3)

23.8.2 Error Message (Watchpoint Overrun)

A watchpoint overrun error occurs when the same watchpoint occurs multiple times before the first occurrence of that watchpoint has been messaged out. The watchpoint message (which has information of all the watchpoints that occurred prior to the detection of the same watchpoint occurring multiple times) will be sent before the error message can be sent.

The overrun error causes further watchpoint occurrences to be ignored, until the error message has been sent. The error code within the error message indicates that a watchpoint overrun error has occurred.

The error message has the following format:

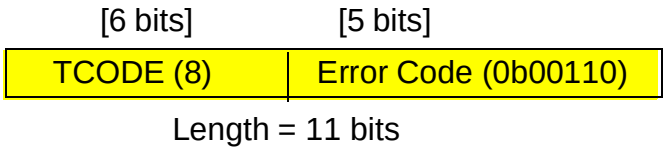


Figure 23-55 Error Message (Watchpoint Overrun) Format

23.8.3 Synchronization

Upon occurrence of a watchpoint, the next program and data trace message will be a synchronization message (provided program and data trace are enabled).

23.8.4 Timing Diagrams

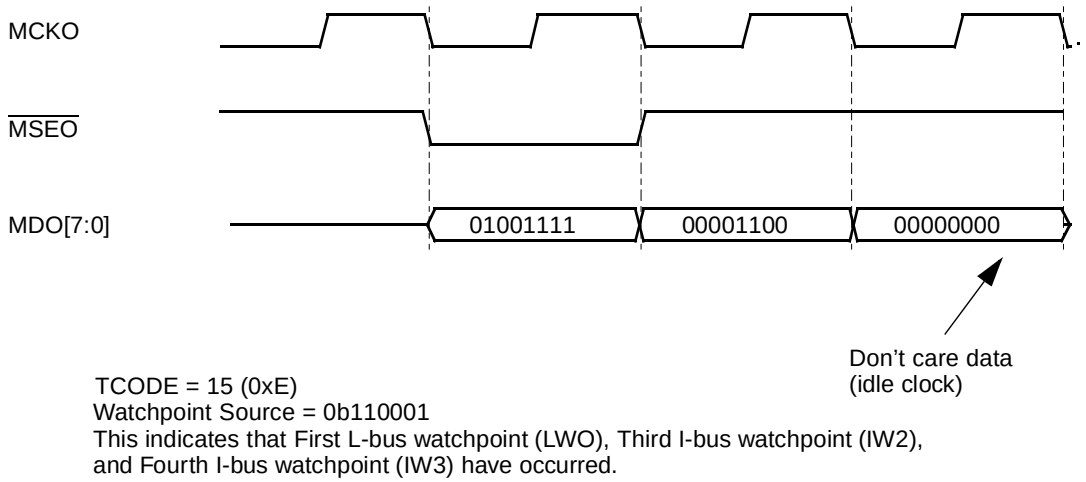


Figure 23-56 Watchpoint Message

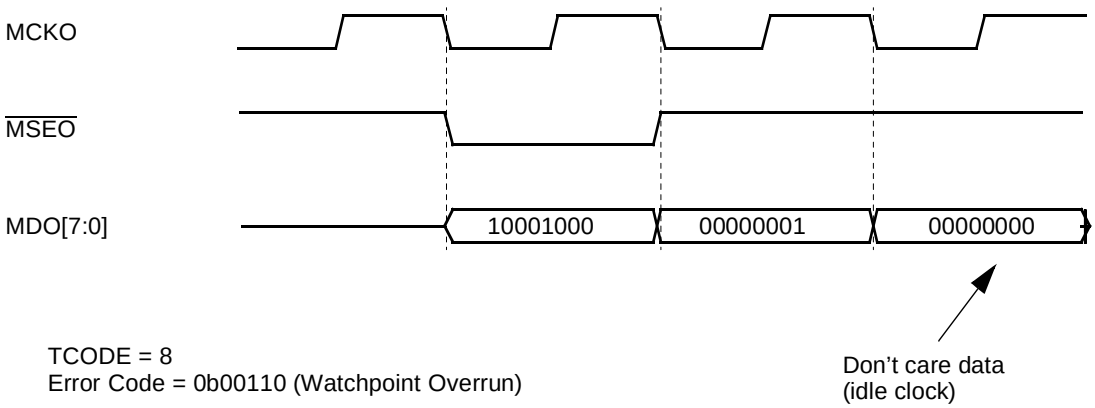


Figure 23-57 Error Message (Watchpoint Overrun)

23.9 Ownership Trace

This section details the ownership trace support features of the READI module.

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the

highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

23.9.1 Ownership Trace Messaging

Ownership trace information is messaged via the auxiliary port using an ownership trace message (OTM). The ownership trace register (OT), which can be accessed via auxiliary port, is updated by the operating system software to provide task/process ID information. When new information is updated in the register by the embedded processor, it is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

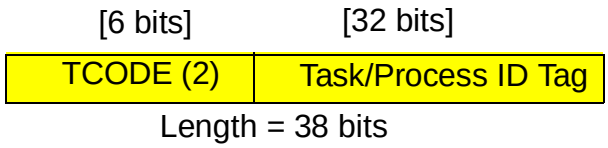


Figure 23-58 Ownership Trace Message Format

23.9.2 Error Message (Queue Overflow)

A program/data/ownership trace overrun error occurs when a trace message cannot be queued due to the queue being full, provided ownership trace is enabled.

The overrun error causes the message queue to be flushed, and a error message to be queued. The error code within the error message indicates that a program/data/ownership trace overrun error has occurred.

The error message has the following format:

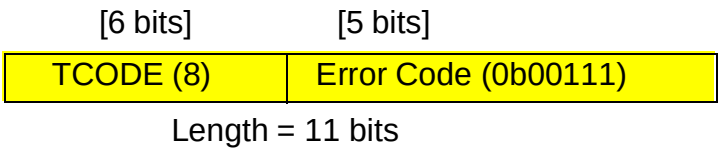


Figure 23-59 Error Message Format

23.9.2.1 OTM Flow

Ownership trace messages are generated when the operating system (privileged supervisor task) writes to the memory mapped ownership trace register.

The following flow describes the OTM process.

1. The OT register is a memory mapped register, whose address is located in the UBA. The OT register address can be read from the UBA register by the IEEE-ISTO 5001 tool.
2. Only privileged writes (byte/half word or word) initiated by the RCPU to the OT register that terminate normally are valid. The data value (word) written into the register is formed into the ownership trace message that is queued to be transmitted.
3. OT register reads and non-privileged OT register writes, or writes initiated by any source other than the RCPU, do not cause ownership trace messages to be transmitted by the READI module.

23.9.2.2 OTM Queueing

READI implements a 16-message-deep queue for program trace, data trace, and ownership trace messages. Messages that enter the queue are transmitted via the output auxiliary port in the order in which they are queued.

NOTE

If multiple trace messages need to be queued at the same time, ownership trace messages will have the lowest priority.

23.9.3 Timing Diagram

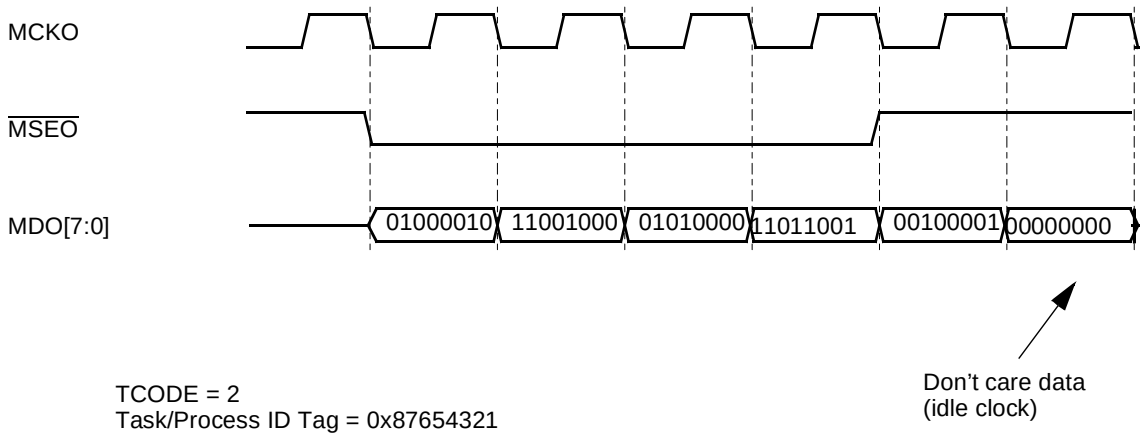


Figure 23-60 Ownership Trace Message

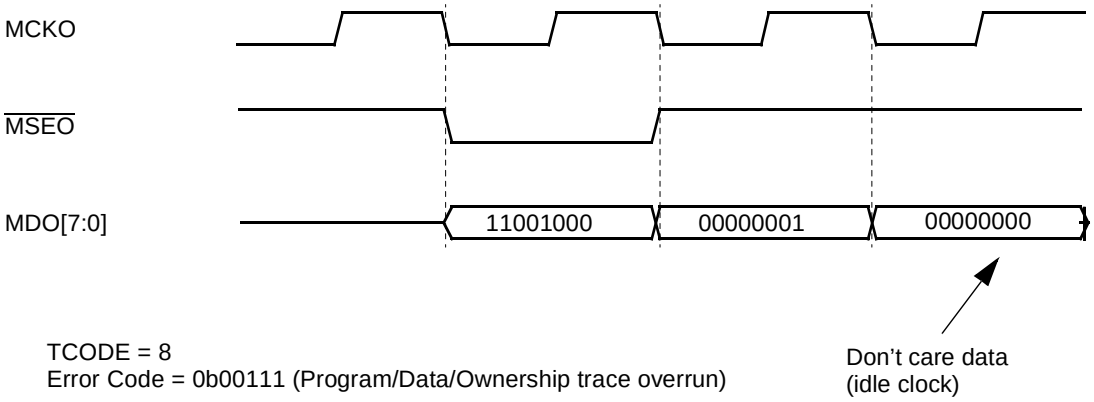


Figure 23-61 Error Message (Program/Data/Ownership Trace Overrun)

23.10 RCPU Development Access

This section details RCPU development access support features of the READI module.

The READI development port provides a full duplex serial interface for accessing existing RCPU user register and development features including BDM (background debug mode).

For further details on RCPU development function and usage, consult **SECTION 22 DEVELOPMENT SUPPORT** or the Development Port chapter of the ***RCPU Reference Manual*** **(RCPURM/AD)**.

RCPU development access can be achieved either via the READI pins or the BDM pins on the MCU. The access method is determined during READI module configuration. **Figure 23-62** shows how READI and BDM pins are multiplexed for RCPU development access.

When the READI module is configured for RCPU development access, IEEE-ISTO 5001 compliant vendor-defined messages are used for transmission of data in and out of the MCU.

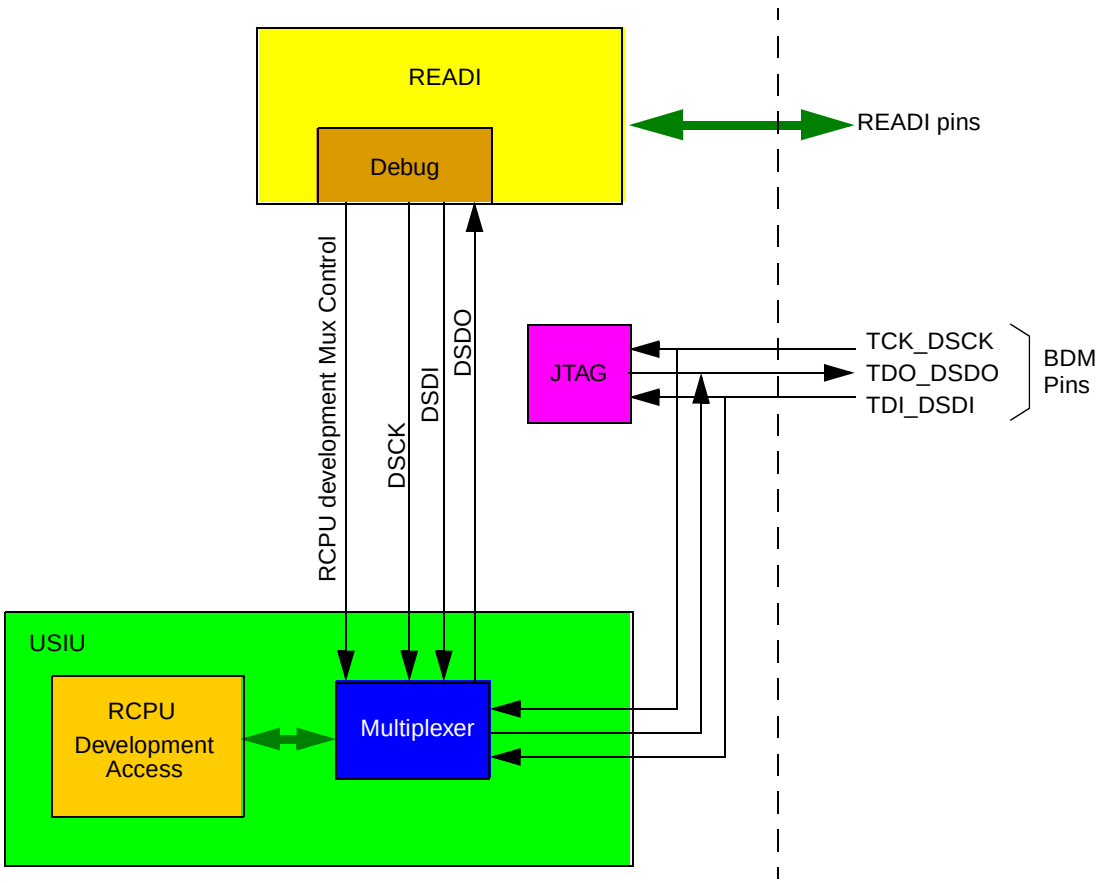


Figure 23-62 RCPU Development Access Multiplexing Between READI and BDM Pins

23.10.1 RCPU Development Access Messaging

The following RCPU development access messages are used for handshaking between the device and the tool — DSDI data message, DSDO data message and BDM status message.

23.10.1.1 DSDI Message

The DSDI message is used by the tool to download information to the RCPU.

The DSDI data field has a 3-bit status header followed by 7 or 32 bits of data/instruction, depending on the RCPU development port mode.

The DSDI message has the following format:

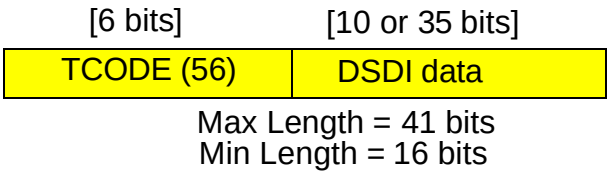


Figure 23-63 DSDI Message Format

NOTE

When sending in a DSDI data message, the DSDI data should contain the control and status bits (*start, mode, control*), followed by the 7 or 32-bit CPU instruction/data or trap enable, **MSBit first**. See **Figure 23-69** for DSDI data message transmission sequence.

23.10.1.2 DSDO Message

The DSDO message is used by the device to upload information from the RCPU.

The DSDO data field has a 3-bit status header followed by 7 or 32 bits of data/instruction, depending on the RCPU development port mode.

The three status bits in the DSDO data indicates if the device is ready to receive the next message from the tool.

The DSDO message has the following format:

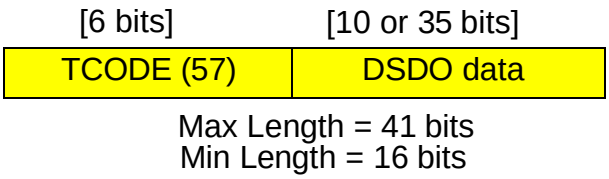


Figure 23-64 DSDO Message Format

NOTE

The DSDO data received will contain the control and status bits and data from the CPU, **MSbit first**. See **Figure 23-69** for DSDO data message transmission sequence.

23.10.1.3 BDM Status Message

BDM status message is generated by the device to let the tool know about the status of debug mode.

BDM status message (with BDM status field equal to 0b1) is sent when the RCPU is in debug mode and the device is ready to receive debug mode messages.

BDM status message (with BDM status field equal to 0b0) is sent out when the device exits BDM mode and RCPU is in normal operating mode.

The BDM status message has the following format:

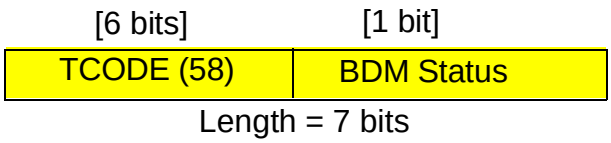


Figure 23-65 BDM Status Message Format

23.10.1.4 Error Message (Invalid Message)

An error message is sent out when an invalid message is received by READI. The error code within the error message indicates that an invalid TCODE was detected in the auxiliary input messages by the pin input formatter.

The error message has the following format:

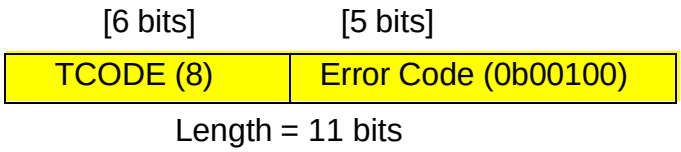


Figure 23-66 Error Message (Invalid Message) Format

23.10.2 RCPU Development Access Operation

The RCPU development access can be achieved either via the READI pins or the BDM pins. The default access is via the BDM pins.

To enable RCPU development access via the READI pins, the tool has to configure the DC register during the READI reset ($\overline{\text{RSTI}}$). Once the READI module takes the control of RCPU development access, the protocol for transmission of development serial data in (DSDI) and out (DSDO) is performed through the IEEE-ISTO 5001-1999 compliant vendor-defined messages.

After enabling RCPU development access via the READI pins, the READI module can enable debug mode and enter debug mode. When debug mode is enabled and entered, READI sends a BDM status message (BDM status field equal to 0b1) to the development tool indicating that the RCPU has entered debug mode and is now expecting instructions from the READI pins.

The development tool then uses the DSDI Data Message to send in the serial transmission data to READI. Data is transmitted to the tool using the DSDO data message.

This process continues until the RCPU exits debug mode and READI sends the BDM status message (BDM status field equal to 0b0) indicating debug mode exit.

NOTE

Only after the DSDO data message is sent out should another DSDI data message be sent in.

Synchronous self-clocked mode is selected by READI for RCPU development access. In this mode, the internal transmission between READI and the USIU is performed at system frequency.

When the RCPU is in debug mode, program trace is not allowed. If program trace is enabled, a program trace synchronization message is generated when debug mode exits.

When the RCPU is in debug mode, data trace and R/W access are allowed.

The flow chart in **Figure 23-67** shows RCPU development access configuration via READI. The modes of RCPU development access via READI are described below. Allowed modes are also summarized in **Table 23-9** of **23.2.1.5 RCPU Development Access Modes**.

23.10.2.1 Enabling RCPU Development Access Via READI Pins

Reset sequencing is done by the tool to initialize the READI pins and registers by asserting $\overline{\text{RSTI}}$ (the device sends out the device ID message after the $\overline{\text{RSTI}}$ negation). System reset is held by the tool until the READI module is reset and initialized with desired RCPU development access setting.

For RCPU development access to be enabled via the READI pins, the tool has to configure the DC register (DPA field equal to 0b1) after the negation of $\overline{\text{RSTI}}$, but before the negation of system reset. System reset should only be negated at least 16 system clocks after the DC register has been configured.

If the DC register is not configured such that READI module has control of the RCPU development access signals before the negation of the system reset, then RCPU development access is via debug mode pins. This is the default setting (DPA=0b0, DME=0bX, DOR=0bX in DC register).

NOTE

The READI module will ignore any incoming DSDI data messages when the module is not configured for RCPU development access.

23.10.2.2 Enabling Background Debug Mode (BDM) via READI Pins

After RCPU development access has been enabled via the READI pins, debug mode is enabled by setting bits in the DC register (DPA=0b1, DME=0b1, DOR=0b0).

23.10.2.3 Entering Background Debug Mode (BDM) Via READI Pins

There are three ways to enter debug mode (provided debug mode has been enabled):

1. Enter debug mode (halted state) out-of-system reset through READI module configuration. This is displayed in **Figure 23-68**.
2. Enter debug mode by downloading breakpoint instructions through RCPU development access when in non-debug (running) mode.
3. Enter debug mode if an exception or interrupt occurs.

When entering debug mode following an exception/breakpoint, the RCPU signals VFLS[0:1] are equal to 0b11. This causes READI to send a BDM status message to the tool indicating that the RCPU has entered debug mode and is now expecting instructions from the READI pins.

Debug mode enabling through READI and entering debug mode out of system reset is done by setting the following bits in the DC register (DPA=0b1, DME=0b1, DOR=0b1) during system reset. Debug mode entry causes RCPU to halt.

23.10.2.4 Non-Debug Mode Access of RCPU Development Access

The RCPU development access can be also be used while the RCPU is not halted (in debug mode). This feature is used to send in breakpoints or synchronization events to the RCPU. Please refer to the ***RCPU Reference Manual* (RCPURM/AD)** for further details.

Non-debug mode access of RCPU development can be achieved by configuring the READI module to take control of RCPU development access during module configuration of the DC register (DPA=0b1, DME=0b0, DOR=0bx).

23.10.2.5 RCPU Development Access Flow Diagram

Figure 23-67 has flow diagram describing how the RCPU development access can be achieved via READI pins.

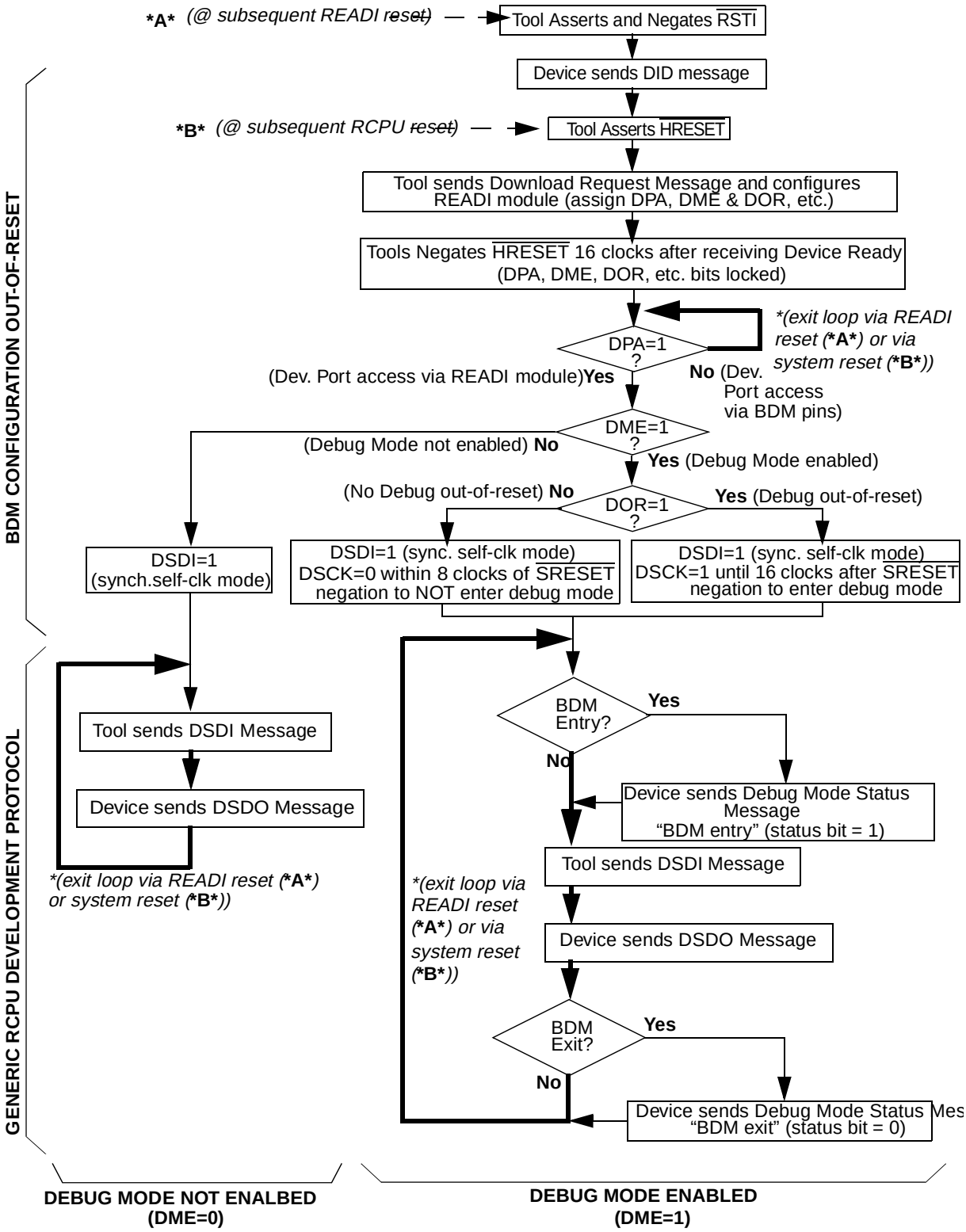


Figure 23-67 RCPU Development Access Flow Diagram

23.10.3 Throughput

The tool can send a DSDI data message into device upon the receipt of a DSDO data message as soon as the tool decodes the first two status bits of the DSDO data message just received and confirms valid data from the RCPU.

An example throughput analysis is performed with the following assumptions:

- READI configuration of RCPU development access and debug mode is already entered through READI
- The module is configured for reduced port mode
- MCKI running at 28 MHz
- MCKO running at 56 MHz
- 56-MHz internal operation
- READI auxiliary input and output pins are free (not in middle of transmission)
- No delay from tool in responding — tool keeps up with READI port
- Tool reads the complete DSDO data message before shifting in DSDI data message
- 10 clocks estimated to format and encode/decode DSDI data and DSDO data messages within READI

The DSDI Data message is 41 bits (six bits of TCODE and 35 bits of DSDI data data). It takes 41 clocks (41 bits / 1 MDI pins) to shift in the DSDI data message. It is estimated that READI will take approximately 10 clocks to decode the DSDI data message. After the message has been decoded, READI will take 35 clocks to serially shift in the 35 bits of DSDI data to the RCPU development port. Hence, it takes a total of 86 clocks (41 + 10 + 35) to decode and shift in DSDI data from the tool to the RCPU development port.

At 28 MHz, it translates to 3079 ns (35.8 x 81) to decode and shift in DSDI data to RCPU development port

As DSDI bits are shifted into the RCPU development register, DSDO bits are shifted out from the same RCPU development register (DPDR) and these are captured by READI.

It is estimated that READI will take approximately 10 clocks to encode the DSDO data. The DSDO message is 41 bits (6 bits of TCODE and 35 bits of DSDO data). It will take 21 clocks (41 bits / 2 MDO pins) for READI to transmit this message. Hence, it will take a total of 31 clocks (10 + 21) to encode the DSDO data message and shift out the DSDO data message to the tool.

At 56 MHz, it will take 552 ns (17.8 x 31) to encode and shift out DSDO data to the tool.

Thus, it will take 3631 ns (3079 + 552) for one complete DSDI data and DSDO data messaging cycle.

23.10.4 Timing Diagrams

Figure 23-68 shows the timing diagram of RCPU development access and entering debug mode out-of-system reset through READI.

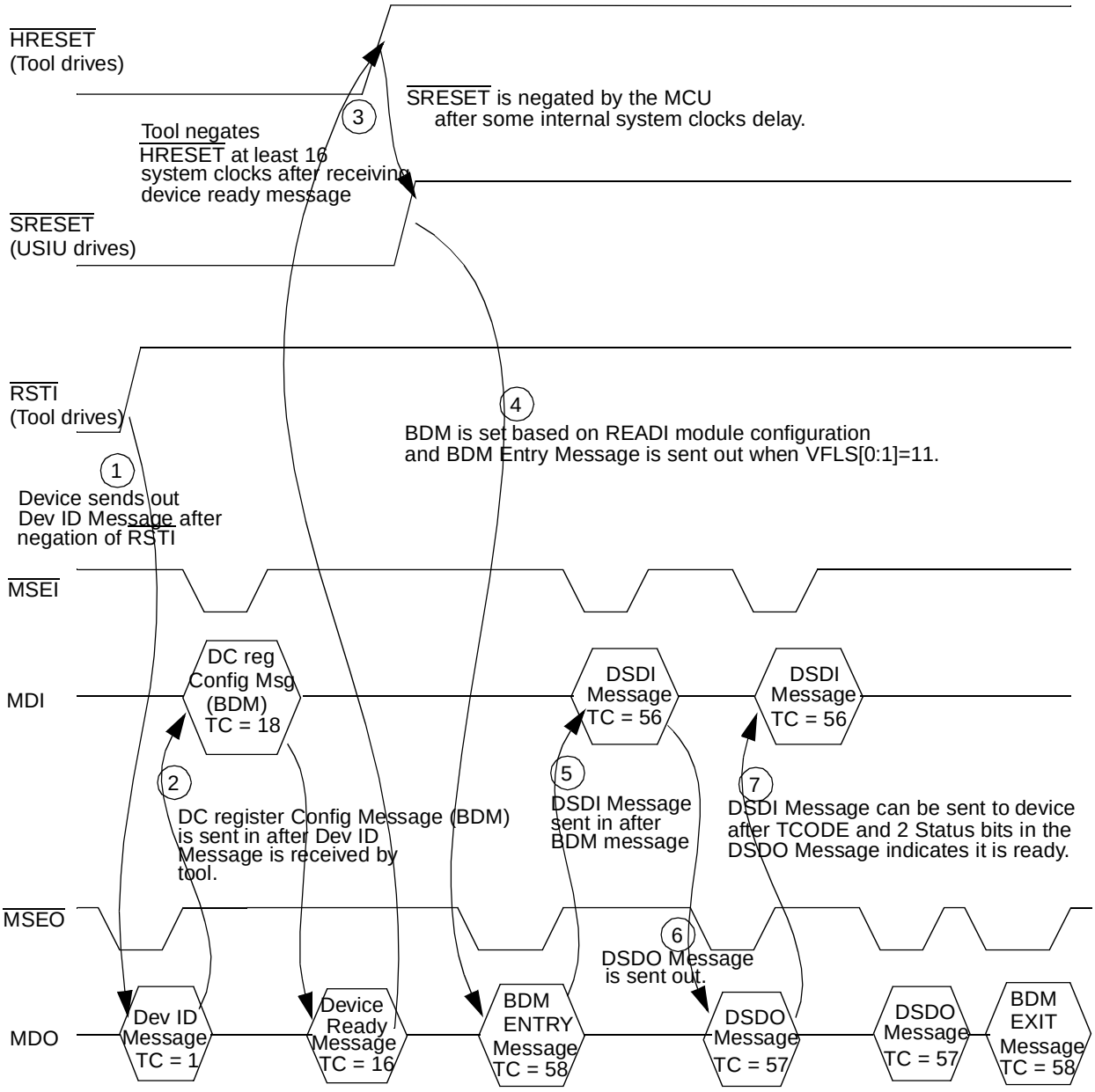


Figure 23-68 RCPU Development Access Timing Diagram — Debug Mode Entry Out-of-Reset

Figure 23-69 shows the transmission sequence of DSDI/DSDO data messages.

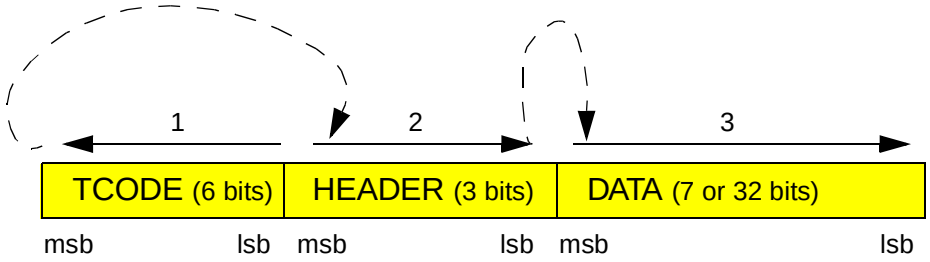


Figure 23-69 Transmission Sequence of DSD*x* Data Messages

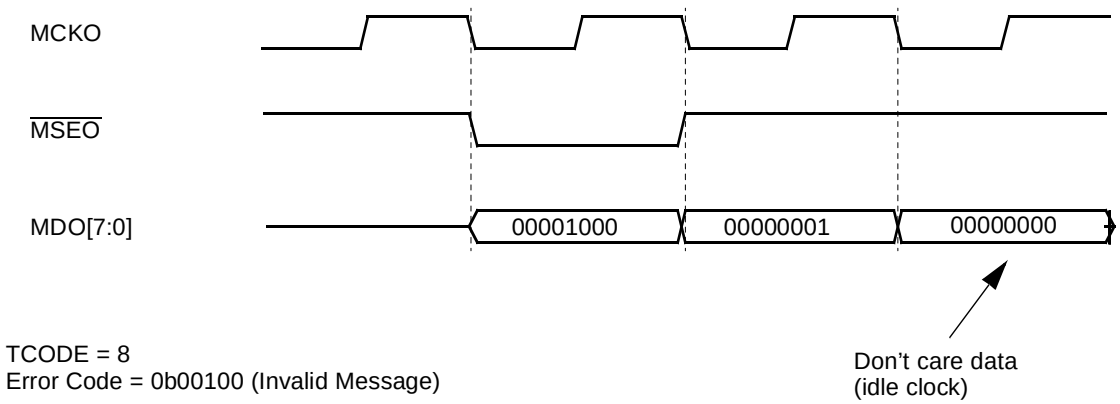


Figure 23-70 Error Message (Invalid Message)

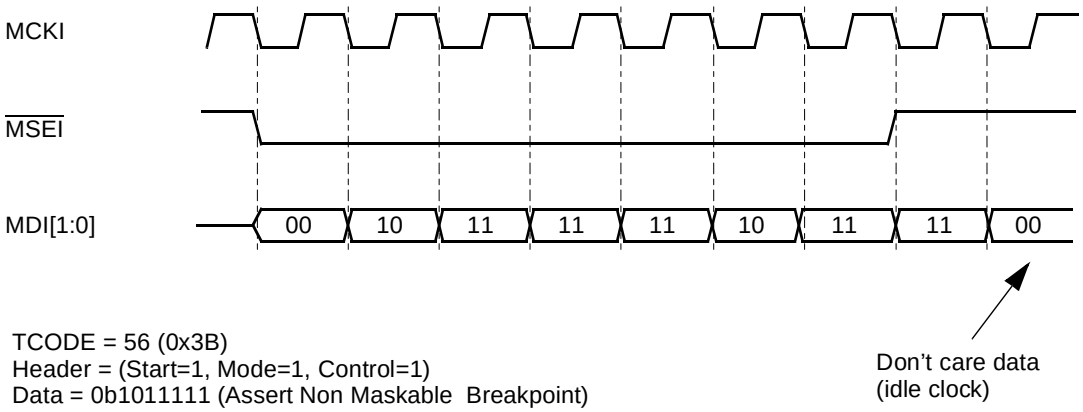


Figure 23-71 DSDI Data Message (Assert Non-Maskable Breakpoint)

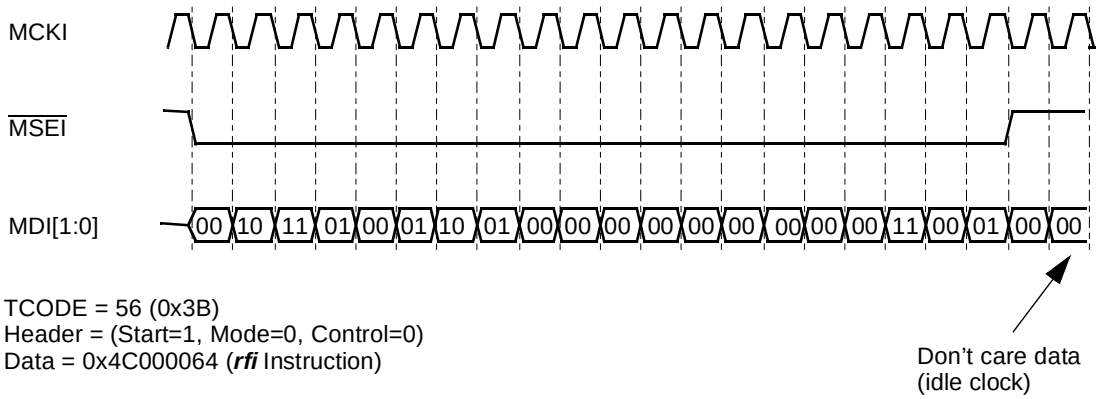


Figure 23-72 DSDI Data Message (CPU Instruction — *rfi*)

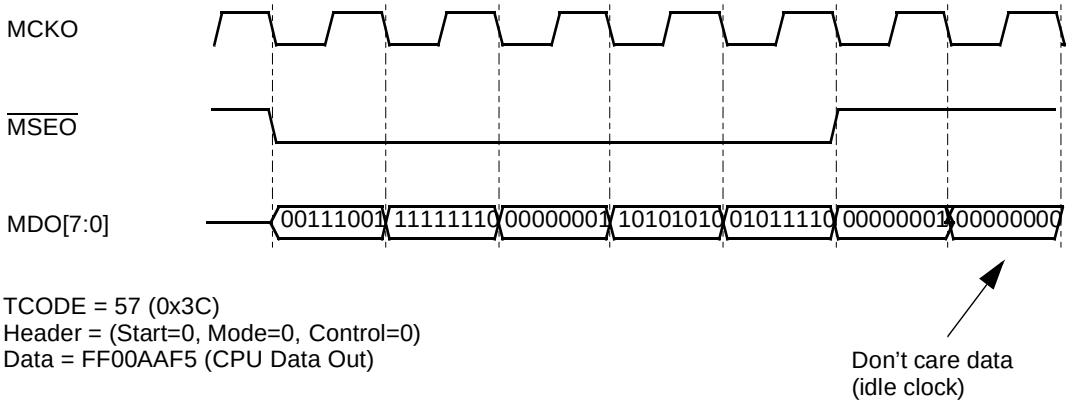


Figure 23-73 DSDO Data Message (CPU Data Out)

23.11 Power Management

This section details the power management features of the READI module.

The READI module is a development interface, and is not expected to function under normal (non-development) conditions. Therefore power management is required to reduce and minimize power consumption during normal operation of the part.

23.11.1 Functional Description

The following are the candidates for power management:

Table 23-35 Power Management Mechanism Overview

Feature	Power Saving Mechanism
Disabled Mode	If $\overline{\text{EVTI}}$ is negated at negation of $\overline{\text{RSTI}}$, the READI module will be disabled. No trace output will be provided, and output auxiliary port will be three-stated.
Sleep, Deep-Sleep and Low Power-Down Mode	All outputs will be held static.
READI Reset ($\overline{\text{RSTI}}$)	Output auxiliary pins will be three-stated.

23.11.2 Low Power Modes

When the MCU is in sleep, deep-sleep or low power-down mode, all internal clocks on the MCU are shut down, including the MCKO. The $\overline{\text{MSEO}}$ pin will be held negated.

Low power mode entry for the MCU will be held off until the READI module has transmitted all existing messages (in the queues and transmit buffers). During this time, input messages from the development tool are ignored.

Upon restoration of clocks in normal mode, program and data traces will be synchronized, if enabled.

23.12 Application Notes

This section describes application notes for READI.

23.12.1 Automotive Calibration

READI has features that can be used for calibration tasks during automotive system development. The basic needs for automotive development tools are

1. To acquire, while running an engine or vehicle, crank shaft synchronous data relating to calibration factors as they are being used or modified during high speed transient events, with acceptable impact to the system under development.
2. To acquire, while running an engine or vehicle, time synchronous data relating to calibration factors as they are being used or modified during high speed transient events, with acceptable impact to the system under development.
3. To coherently modify table(s) of calibration constant with the engine control unit is running an engine or vehicle.

23.12.1.1 Calibration Variable Acquisition

Calibration variable acquisition refers to the first two points listed in the basic needs for automotive development tools. READI supports calibration variable acquisition via data trace (read and write) messaging and read/write access.

Requirements for measurement:

- Time synchronous as well as engine synchronous
- Selection of values must be flexible

- Data to be measured (minimum)
  - 29 values engine speed synchronous
  - 45 values 10 ms synchronous
  - 45 values 100 ms synchronous
- Minimal interrupt load
- Minimal impact to system

23.12.2 Calibration Variables Located In Contiguous Memory Locations

23.12.2.1 Data Read Messaging

Calibration variables located in two contiguous memory (RAM) locations can be acquired using data read messaging. The software routines must read these variables for them to be traced and messaged via the READI port.

READI queue sizes and READI port throughput will require few clocks between the individual ‘reads’ of the variables, otherwise information will be lost.

DRM is accomplished by snooping the L-bus hence there is no impact to the RCPU performance.

23.12.2.2 Read/Write Access

In case the throughput requirements for data traces are not acceptable, block read/write access should be used to load the variables from the calibration memory.

This method may be more preferable because it is more likely to be synchronized and coherent data for multiple task scenarios.

The disadvantage of using read/write access is the impact it has on RCPU performance. Block word read accesses can be done on the L-bus every 13 clocks. If the block word read accesses happen to be overlapped with a block access via the RCPU, the maximum intrusion will be 7.7%.

23.12.3 Calibration Variables Not Located in Contiguous Memory Locations

23.12.3.1 Data Write Messaging

Option A:

If calibration variables are not located in contiguous memory locations, then a contiguous section of the RAM can be reserved for variable acquisition tracing. At engine/time synchronous points, the software can write these contiguous locations with the values of the variables. This activity will be traced and messaged out data write messaging.

This method requires a ‘shadow’ location for each word being monitored.

A load and a store will be required for each variable to update the shadow locations, which are traced.

READI queue sizes and IEEE-ISTO 5001 port throughput will require few clocks between the individual ‘writes’ of the variables. This could be achieved by adding a few ‘nop’ instructions.

Option B:

Another possibility is to allocate a (single/few) word(s) in the RAM that are configured to be traced by the READI module. Then at engine/time synchronous points, the software will write the values of a SERIES of variables to a corresponding location. This activity will be traced and messaged out data write messaging.

This method requires a ‘shadow’ location, for each SERIES (of related) variables or channels, in the RAM.

A load and a store will be required for each variable to update the shadow locations, which are traced. The 1st write to the location could be a byte which indicates the channel number. The calibration tool then will not be required to know the address translation for channel information.

The advantage of using this option is that fewer locations need to be reserved in memory for measurement and the data trace messages are smaller, since a whole table (45 values) may be written to the same location.

The disadvantage of using this option is that the calibration tool needs to be aware of which series a particular location represents, and the order in which the values are written.

READI queue sizes and READI port throughput will require few clocks between the individual ‘writes’ of the variables. This could be achieved by adding a few ‘nop’ instructions.

23.12.3.2 Read/Write Access

In case the throughput requirements for block data traces are not acceptable, read/write access should be used to load the variables from the calibration memory.

Option A: Block Read Access

This method calls for setting aside some memory (~ 1-Kbyte RAM locations) for calibration variable measurement. When the software tasks that modify the variables are completed, they will copy the variables into contiguous locations in this reserved space. Then the tasks will indicate to the external tool that the variables are valid by any of various means (write keyword to special address, watchpoint etc.). The calibration tool will then proceed to read the variables corresponding to the channel completed using the block read access feature.

This method may be more preferable because it is more likely to be synchronized and coherent data for multiple task scenarios.

The disadvantage of using block read access is the impact it has on RCPU performance. Block word read accesses can be done on the L-bus every 13 clocks. If the

block word read accesses happen to be overlapped with a block access via the RCPU, the maximum intrusion will be 7.7%.

Option B: Single Read Access

Single read accesses can also be used to acquire calibration variables which are not located in contiguous memory locations.

Single read accesses also have an the impact on RCPU performance. Single word read accesses can be done on the L-bus every 74 clocks. If single word read accesses happen to be overlapped with a block access via the RCPU, the maximum intrusion will be 1.3%. This is significantly better when compared to block read accesses (7.7%).

The disadvantage of using single read access the data trace throughput. For single read accesses, 0.41 million words can be messaged out via the READI port, whereas for block read accesses, approximately 4 million words can be messaged out via the READI port.

23.12.3.3 Calibration Constant Tuning

Calibration constant tuning refers to the third basic requirement for automotive development tools. READI supports calibration constant tuning via the read/write access feature.

The read/write access protocol provides run-time access to MCU registers and memory map. READI implements the run-time read/write access feature via two MDI pins and four MDO pins.

Read/write accesses can be either single or block write accesses. If the calibration constants that need to be modified are located in contiguous memory locations, the block write access feature should be used.

The disadvantage of using read/write access is the impact it has on RCPU performance. Block write accesses can be done on the L-bus every 26 clocks. If the block write happens to be overlapped with a block access via the RCPU, the maximum intrusion will be 3.9%.

23.12.4 UC3F Programming Guideline Flowchart via READI Read/Write Access

The user can use READI features such as read/write access or RCPU development access to program the UC3F. For details on how the UC3F can be programmed via READI read/write access, refer to **Figure 23-74** and **Figure 23-75**.

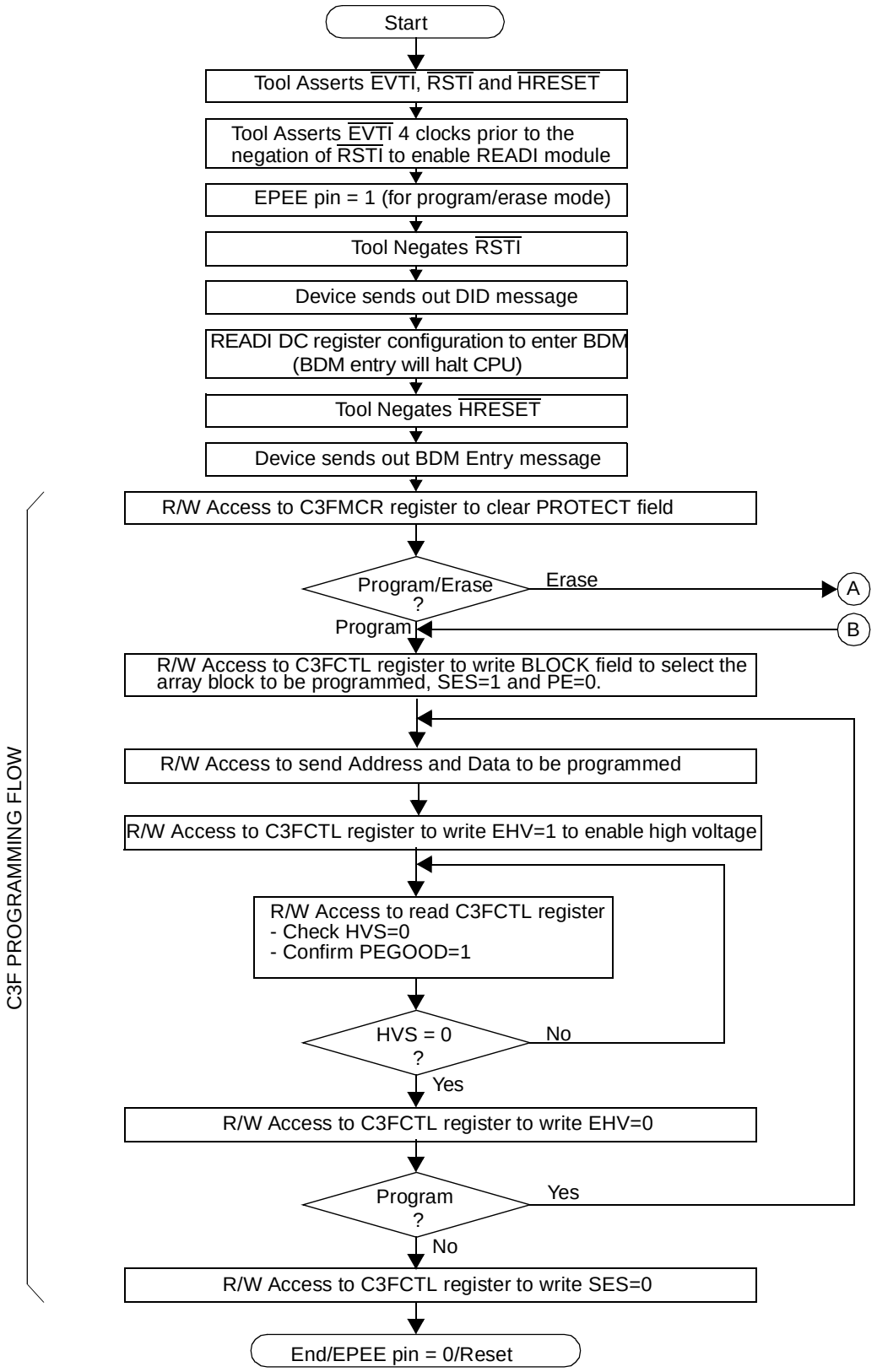


Figure 23-74 Programming UC3F Flash via READI R/W Access (Sheet 1 of 2)

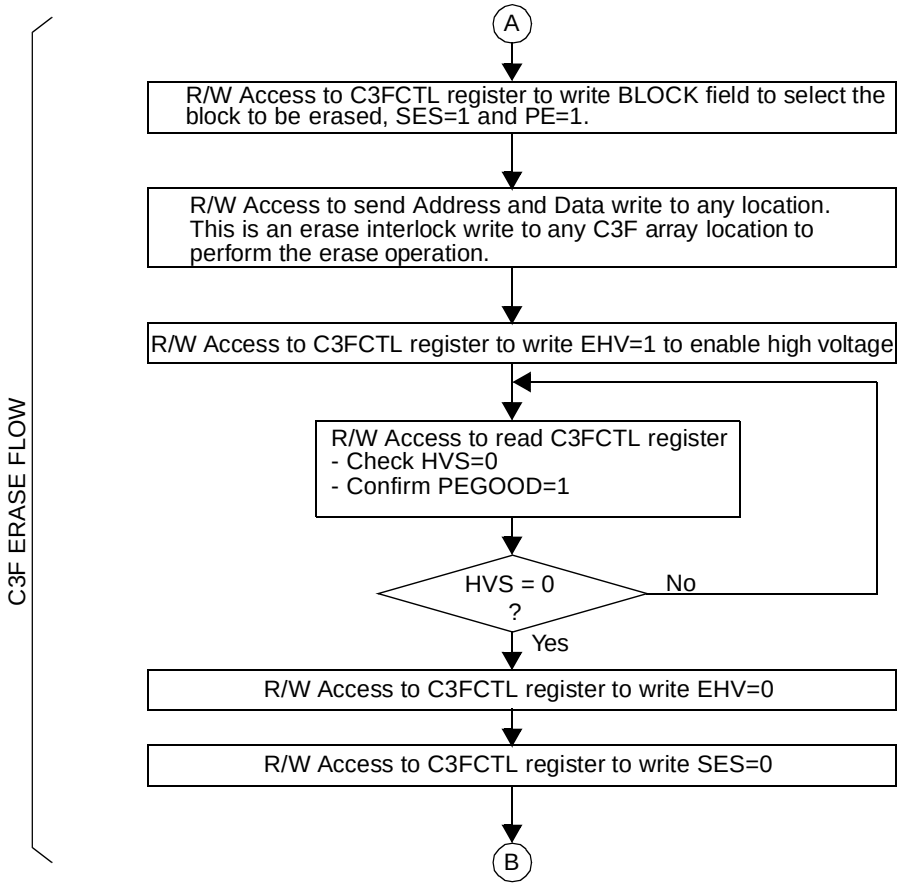


Figure 23-75 Programming UC3F Flash via READI R/W Access (Sheet 2 of 2)