Chapter 14
Debug Test Access Port

This chapter introduces the Debug Test Access Port built into processor. It contains the following sections:

14.1 Debug Test Access Port and Debug state

In Debug state, JTAG-based hardware provides access to the processor and debug unit. Access is through scan chains and the *Debug Test Access Port* (DBGTAP). The *DBGTAP state Machine* (DBGTAPSM) is illustrated in Figure 14-1.

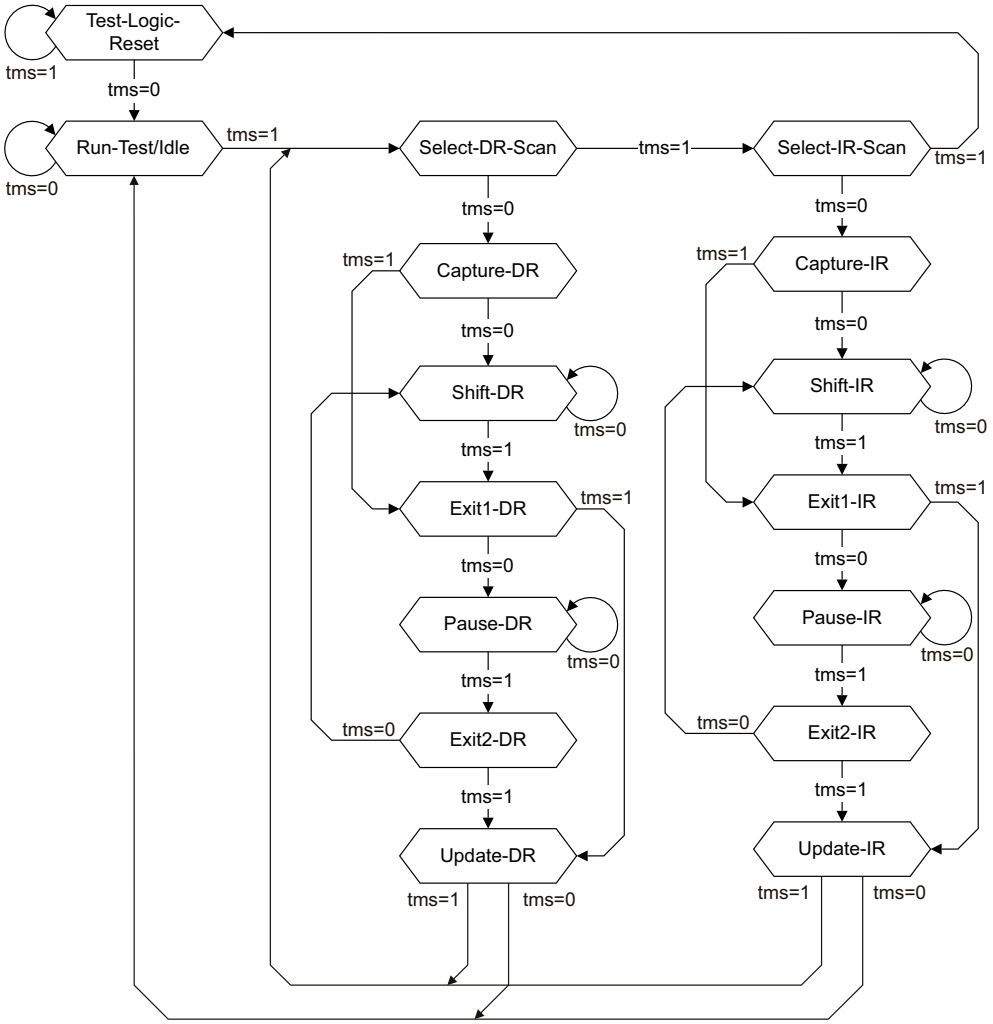

Figure 14-1 JTAG DBGTAP state machine diagram[1]

1. From IEEE Std 1149.1-2001. Copyright 2001 IEEE. All rights reserved.

14.2 Synchronizing RealView ICE

The system and test clocks are synchronized internally to the macrocell. The ARM RealView ICE debug agent directly supports one or more cores within an ASIC design. The off-chip device, for example, RealView ICE, issues a **TCK** signal and waits for the **RTCK**, Returned **TCK**, signal to come back. Synchronization is maintained because the off-chip device does not progress to the next **TCK** edge until after an **RTCK** edge is received. Figure 14-2 shows this synchronization.

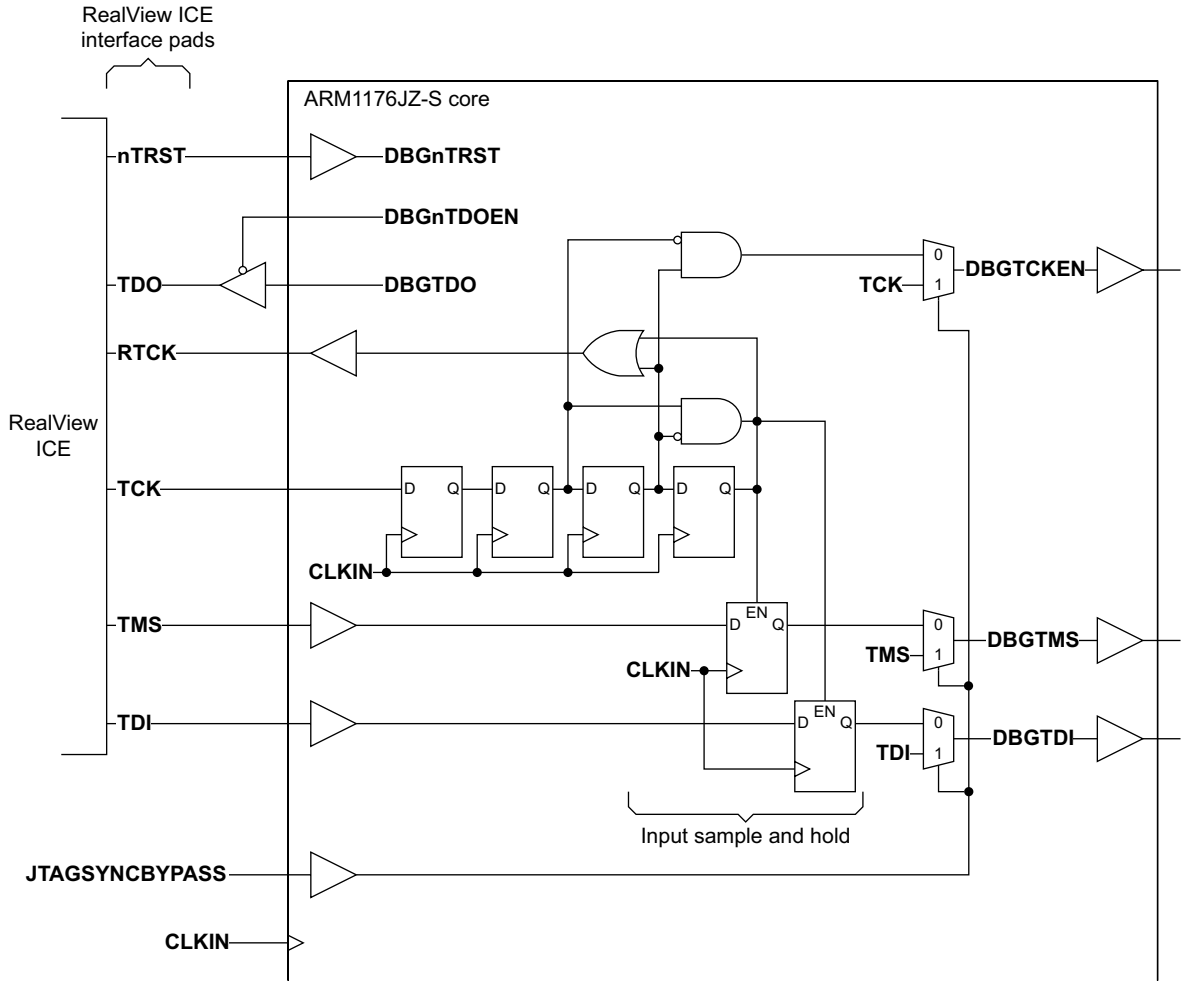

Figure 14-2 RealView ICE clock synchronization

Note

All of the D type flip-flops are reset by **DBGnTRST**.

14.3 Entering Debug state

Halting debug-mode is enabled by writing a 1 to bit 14 of the DSCR, see *CP14 c1, Debug Status and Control Register (DSCR)* on page 13-7. This can only be done by a DBGTAP debugger hardware such as RealView ICE. When this mode is enabled and the core is in a state where debug is permitted the processor halts, instead of taking an exception in software, if one of the following events occurs:

- vector catch occurs
- a breakpoint hits
- a watchpoint hits
- a BKPT instruction is executed.

The processor also enters Debug state, provided that its state permits debug, when:

- A Halt instruction has been scanned in through the DBGTAP. The DBGTAP controller must pass through Run-Test/Idle to issue the Halt command to the processor.
- **EDBGRQ** is asserted.

If debug is enabled by DBGEN, scanning a Halt instruction in through the DBGTAP, or asserting **EDBGRQ**, halts the processor and causes it to enter Debug state, regardless of the selection of a debug-state in DSCR[15:14]. This means that a debugger can halt the processor immediately after reset in a situation where it cannot first enable Halting debug-mode during reset.

The core halted bit in the DSCR is set when Debug state is entered. At this point, the debugger determines why the integer core was halted and preserves the processor state. The MSR instruction can be used to change modes permitted by the **SPIDEN** signal and SUIDEN bit and gain access to banked registers in the machine. While in Debug state:

- the PC is not incremented
- interrupts are ignored
- all instructions are read from the instruction transfer register, scan chain 4.

Debug state on page 13-37 describes the Debug state.

14.4 Exiting Debug state

To exit from Debug state, scan in the Restart instruction through the processor DBGTAP. You might want to adjust the PC before restarting, depending on the way the integer core entered Debug state. When the state machine enters the Run-Test/Idle state, normal operations resume. The delay, waiting until the state machine is in Run-Test/Idle, enables conditions to be set up in other devices in a multiprocessor system without taking immediate effect. When Run-Test/Idle state is entered, all the processors resume operation simultaneously. The core restarted bit is set when the Restart sequence is complete.

14.5 The DBGTAP port and debug registers

The processor DBGTAP controller is the part of the debug unit that enables access through the DBGTAP to the on-chip debug resources, such as breakpoint and watchpoint registers. The DBGTAP controller is based on the IEEE 1149.1 standard and supports:

- a device ID register
- a bypass register
- a five-bit instruction register
- a five-bit scan chain select register.

In addition, the public instructions that Table 14-1 lists are supported.

Table 14-1 Supported public instructions

Binary code	Instruction	Description
b00000	EXTEST	This instruction connects the selected scan chain between **DBGTDI** and **DBGTDO**. When the instruction register is loaded with the EXTEST instruction, the debug scan chains can be written. See *Scan chains* on page 14-10.
b00001	-	Reserved.
b00010	Scan_N	Selects the *Scan Chain Select Register* (SCREG). This instruction connects SCREG between **DBGTDI** and **DBGTDO**. See *Scan chain select register (SCREG)* on page 14-9.
b00011	-	Reserved.
b00100	Restart	Forces the processor to leave Debug state. This instruction is used to exit from Debug state. The processor restarts when the Run-Test/Idle state is entered.
b00101	-	Reserved.
b00110	-	Reserved.
b00111	-	Reserved.
b01000	Halt	Forces the processor to enter Debug state. This instruction stops the processor and puts it into Debug state.
b01001	-	Reserved.
b01010-b01011	-	Reserved.
b01100	INTEST	This instruction connects the selected scan chain between **DBGTDI** and **DBGTDO**. When the instruction register is loaded with the INTEST instruction, the debug scan chains can be read. See *Scan chains* on page 14-10.
b01101-b11100	-	Reserved.

Table 14-1 Supported public instructions (continued)

Binary code	Instruction	Description
b11101	ITRsel	When this instruction is loaded into the IR, Update-DR state, the DBGTAP controller behaves as if IR=EXTEST and SCREG=4. The ITRsel instruction makes the DBGTAP controller behave as if EXTEST and scan chain 4 are selected. It can be used to speed up certain debug sequences. See *Using the ITRsel IR instruction* on page 14-22 for the effects of using this instruction.
b11110	IDcode	See IEEE 1149.1. Selects the DBGTAP controller device ID code register. The IDcode instruction connects the device identification register, or ID register, between **DBGTDI** and **DBGTDO**. The ID register is a 32-bit register that enables you to determine the manufacturer, part number, and version of a component using the DBGTAP. See *Device ID code register* on page 14-8 for details of selecting and interpreting the ID register value.
b11111	Bypass	See IEEE 1149.1. Selects the DBGTAP controller bypass register. The Bypass instruction connects a 1-bit shift register, the bypass register, between **DBGTDI** and **DBGTDO**. The first bit shifted out is a 0. All unused DBGTAP controller instruction codes default to the Bypass instruction. See *Bypass register* on page 14-8.

Note

Sample/Preload, Clamp, HighZ, and ClampZ instructions are not implemented because the processor DBGTAP controller does not support the attachment of external boundary scan chains.

All unused DBGTAP controller instructions default to the Bypass instruction.

14.6 Debug registers

You can connect the following debug registers between **DBGTDI** and **DBGTDO**:

- *Bypass register*
- *Device ID code register*
- *Instruction register* on page 14-9
- *Scan chain select register (SCREG)* on page 14-9
- *Scan chain 0, debug ID register (DIDR)* on page 14-11
- *Scan chain 1, Debug Status and Control Register (DSCR)* on page 14-11
- *Scan chain 4, instruction transfer register (ITR)* on page 14-13
- *Scan chain 5* on page 14-15.
- *Scan chain 6* on page 14-17.
- *Scan chain 7* on page 14-17.

14.6.1 Bypass register

Purpose Bypasses the device by providing a path between **DBGTDI** and **DBGTDO**.

Length 1 bit.

Operating mode When the bypass instruction is the current instruction in the instruction register, serial data is transferred from **DBGTDI** to **DBGTDO** in the Shift-DR state with a delay of one **TCK** cycle. There is no parallel output from the bypass register. A logic 0 is loaded from the parallel input of the bypass register in the Capture-DR state. Nothing happens at the Update-DR state.

Order Figure 14-3 shows the order of bits in the bypass register.

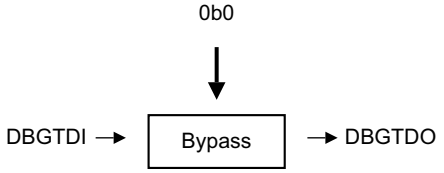

Figure 14-3 Bypass register bit order

14.6.2 Device ID code register

Purpose Device identification. To distinguish the ARM1176JZF-S processors from other processors, the DBGTAP controller ID is unique for each. This means that a DBGTAP debugger, such as RealView ICE, can easily see the processor that it is connected to. The Device ID register version and manufacturer ID fields are routed to the edge of the chip so that partners can create their own Device ID numbers by tying the pins to HIGH or LOW values.

The default manufacturer ID for the ARM1176JZF-S processor is b11110000111. The part number field is hard-wired inside the ARM1176JZF-S to `0x7B76`.

All ARM semiconductor partner-specific devices must be identified by manufacturer ID numbers of the form shown in *c0, Main ID Register* on page 3-20.

Length 32 bits.

Operating mode When the ID code instruction is current, the shift section of the device ID register is selected as the serial path between **DBGTDI** and **DBGTDO**. There is no parallel output from the ID register. The 32-bit device ID code is loaded into this shift section during the Capture-DR state. This is shifted out during Shift-DR, least significant bit first, while a *don't care* value is shifted in. The shifted-in data is ignored in the Update-DR state.

Order Figure 14-4 shows the order of bits in the ID code register.

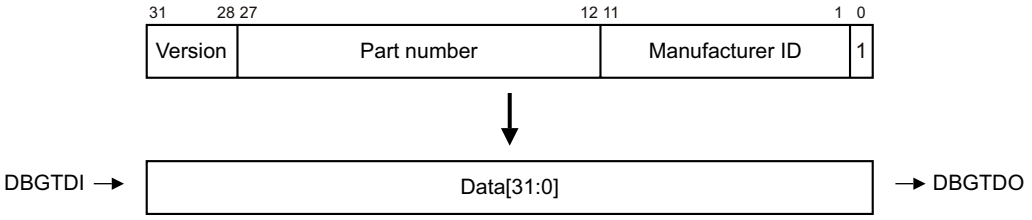

Figure 14-4 Device ID code register bit order

14.6.3 Instruction register

Purpose Holds the current DBGTAP controller instruction.

Length 5 bits.

Operating mode When in Shift-IR state, the shift section of the instruction register is selected as the serial path between **DBGTDI** and **DBGTDO**. At the Capture-IR state, the binary value b00001 is loaded into this shift section. This is shifted out during Shift-IR, least significant bit first, while a new instruction is shifted in, least significant bit first. At the Update-IR state, the value in the shift section is loaded into the instruction register so it becomes the current instruction. On DBGTAP reset, the IDcode becomes the current instruction.

Order Figure 14-5 shows the order of bits in the instruction register.

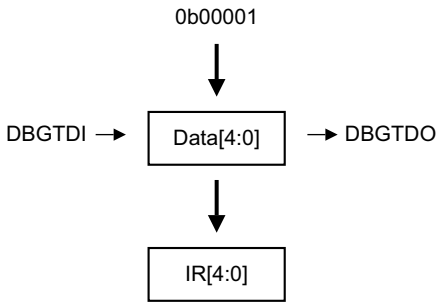

Figure 14-5 Instruction register bit order

14.6.4 Scan chain select register (SCREG)

Purpose Holds the currently active scan chain number.

Length 5 bits.

Operating mode After Scan_N has been selected as the current instruction, when in Shift-DR state, the shift section of the scan chain select register is selected as the serial path between **DBGTDI** and **DBGTDO**. At the Capture-DR state, the binary value b10000 is loaded into this shift section. This is shifted out during Shift-DR, least significant bit first, while a new value is shifted in, least significant bit first. At the Update-DR state, the value in the shift section is loaded into the Scan Chain Select Register to become the current active scan chain. All additional instructions such as INTEST then apply to that scan chain. The currently selected scan chain only changes when a Scan_N or ITRsel instruction is executed, or a DBGTAP reset occurs. On DBGTAP reset, scan chain 3 is selected as the active scan chain.

Order Figure 14-6 shows the order of bits in the scan chain select register.

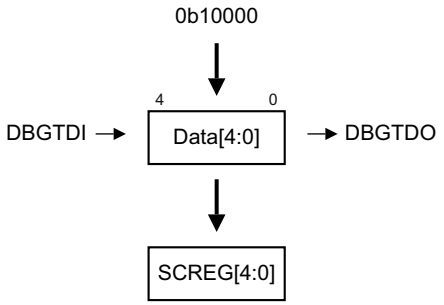

Figure 14-6 Scan chain select register bit order

14.6.5 Scan chains

To access the debug scan chains you must:

1. Load the Scan_N instruction into the IR. Now SCREG is selected between **DBGTDI** and **DBGTDO**.
2. Load the number of the required scan chain. For example, load b00101 to access scan chain 5.
3. Load either INTEST or EXTEST into the IR.
4. Go through the DR leg of the DBGTAPSM to access the scan chain.

INTEST and EXTEST are used as follows:

INTEST Use INTEST for reading the active scan chain. Data is captured into the shift register at the Capture-DR state. The previous value of the scan chain is shifted out during the Shift-DR state, while a new value is shifted in. The scan chain is not updated during Update-DR. Those bits or fields that are defined as cleared on read are only cleared if INTEST is selected, even when EXTEST also captures their values.

EXTEST Use EXTEST for writing the active scan chain. Data is captured into the shift register at the Capture-DR state. The previous value of the scan chain is shifted out during the Shift-DR state, while a new value is shifted in. The scan chain is updated with the new value during Update-DR.

Note

There are some exceptions to this use of INTEST and EXTEST to control reading and writing the scan chain. These are noted in the relevant scan chain descriptions.

Scan chain 0, debug ID register (DIDR)

Purpose Debug.

Length 8 + 32 = 40 bits.

Description Debug identification. This scan chain accesses CP14 debug register 0, the debug ID register. Additionally, the eight most significant bits of this scan chain contain an implementor code. This field is hardwired to `0x41`, the implementor code for ARM Limited, as specified in the *ARM Architecture Reference Manual*. This register is read-only. Therefore, EXTEST has the same effect as INTEST.

Order Figure 14-7 shows the order of bits in scan chain 0.

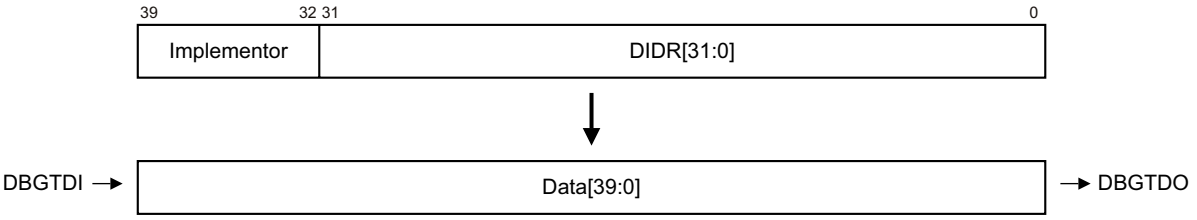

Figure 14-7 Scan chain 0 bit order

Scan chain 1, Debug Status and Control Register (DSCR)

Purpose Debug.

Length 32 bits.

Description This scan chain accesses CP14 register 1, the DSCR. This is mostly a read/write register, although certain bits are read-only for the Debug Test Access Port. See *CP14 c1, Debug Status and Control Register (DSCR)* on page 13-7 for details of DSCR bit definitions, and for read/write attributes for each bit. Those bits defined as cleared on read are only cleared if INTEST is selected.

Order Figure 14-8 shows the order of bits in scan chain 1.

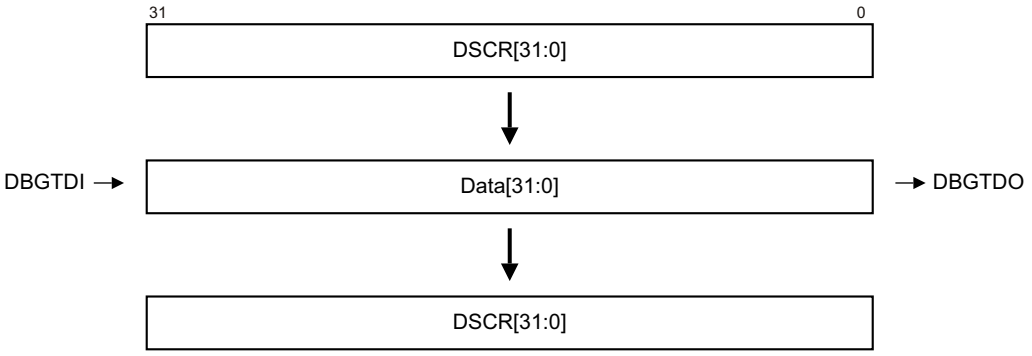

Figure 14-8 Scan chain 1 bit order

The following DSCR bits affect the operation of other scan chains:

DSCR[30:29] rDTRfull and wDTRfull flags. These indicate the status of the rDTR and wDTR registers. They are copies of the rDTRempty, NOT rDTRfull, and wDTRfull bits that the DBGTAP debugger sees in scan chain 5.

DSCR[13] Execute ARM instruction enable bit. This bit enables the mechanism used for executing instructions in Debug state. It changes the behavior of the rDTR and wDTR registers, the sticky precise Data Abort bit, rDTRempty, wDTRfull, and InstCompl flags. See *Scan chain 5* on page 14-15.

DSCR[6] Sticky precise Data Abort flag. If the core is in Debug state and the DSCR[13] execute ARM instruction enable bit is HIGH, then this flag is set on precise Data Aborts. See *CP14 c1, Debug Status and Control Register (DSCR)* on page 13-7.

Note

Unlike DSCR[6], DSCR [7] sticky imprecise Data Aborts flag and DSCR[8] sticky Undefined bits do not affect the operation of the other scan chains.

Scan chain 4, instruction transfer register (ITR)

Purpose Debug

Length 1 + 32 = 33 bits

Description This scan chain accesses the *Instruction Transfer Register* (ITR), used to send instructions to the core through the *Prefetch Unit* (PU). It consists of 32 bits of information, plus an additional bit to indicate the completion of the instruction sent to the core, InstCompl. The InstCompl bit is read-only.

While in Debug state, an instruction loaded into the ITR can be issued to the core by making the DBGTAPSM go through the Run-Test/Idle state. The InstCompl flag is cleared when the instruction is issued to the core and set when the instruction completes.

For an instruction to be issued when going through Run-Test/Idle state, you must ensure the following conditions are met:

- The processor must be in Debug state.
- The DSCR[13] execute ARM instruction enable bit must be set. For details of the DSCR see *CP14 c1, Debug Status and Control Register (DSCR)* on page 13-7.
- Scan chain 4 or 5 must be selected.
- INTEST or EXTEST must be selected.
- Ready flag must be captured set. That is, the last time the DBGTAPSM went through Capture-DR the InstCompl flag must have been set.
- The DSCR[6] sticky precise Data Abort flag must be clear. This flag is set on precise Data Aborts.

For an instruction to be loaded into the ITR when going through Update-DR, you must ensure the following conditions are met:

- The processor can be in any state.
- The value of DSCR[13] execute ARM instruction enable bit does not matter.
- Scan chain 4 must be selected.
- EXTEST must be selected.
- Ready flag must be captured set. That is, the last time the DBGTAPSM went through Capture-DR the InstCompl flag must have been set.
- The value of DSCR[6] sticky precise Data Abort flag does not matter.

Order Figure 14-9 shows the order of bits in scan chain 4.

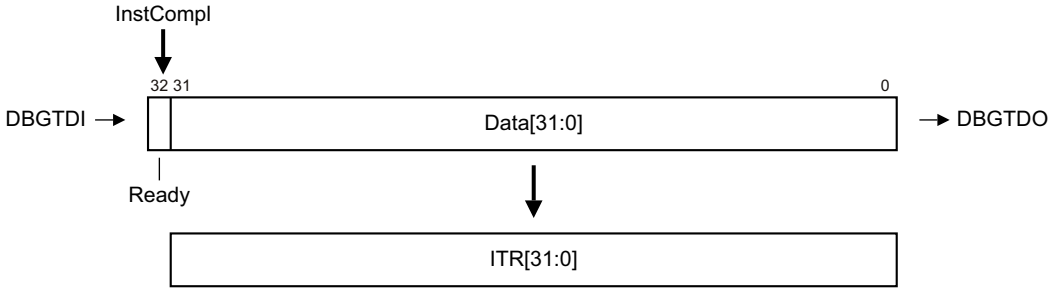

Figure 14-9 Scan chain 4 bit order

It is important to distinguish between the InstCompl flag and the Ready flag:

- The InstCompl flag signals the completion of an instruction.
- The Ready flag is the captured version of the InstCompl flag, captured at the Capture-DR state. The Ready flag conditions the execution of instructions and the update of the ITR.

The following points apply to the use of scan chain 4:

- When an instruction is issued to the core in Debug state, the PC is not incremented. It is only changed if the instruction being executed explicitly writes to the PC. For example, branch instructions and move to PC instructions.
- If CP14 debug register c5 is a source register for the instruction to be executed, the DBGTAP debugger must set up the data in the rDTR before issuing the coprocessor instruction to the core. See *Scan chain 5* on page 14-15.
- Setting DSCR[13] the execute ARM instruction enable bit when the core is not in Debug state leads to Unpredictable behavior.
- The ITR is write-only. When going through the Capture-DR state, an Unpredictable value is loaded into the shift register.

Scan chain 5

Purpose Debug.

Length 1 + 1 + 32 = 34 bits.

Description This scan chain accesses CP14 register c5, the data transfer registers, rDTR and wDTR. The rDTR is used to transfer words from the DBGTAP debugger to the core, and is read-only to the core and write-only to the DBGTAP debugger. The wDTR is used to transfer words from the core to the DBGTAP debugger, and is read-only to the DBGTAP debugger and write-only to the core.

The DBGTAP controller only sees one, read/write, register through scan chain 5, and the appropriate register is chosen depending on the instruction used. INTEST selects the wDTR, and EXTEST selects the rDTR.

Additionally, scan chain 5 contains some status flags. These are nRetry, Valid, and Ready. They are the captured versions of the rDTRempty, wDTRfull, and InstCompl flags respectively. All are captured at the Capture-DR state.

Order Figure 14-10 shows the order of bits in scan chain 5 with EXTEST selected. Figure 14-11 shows the order of bits in scan chain 5 with INTEST selected.

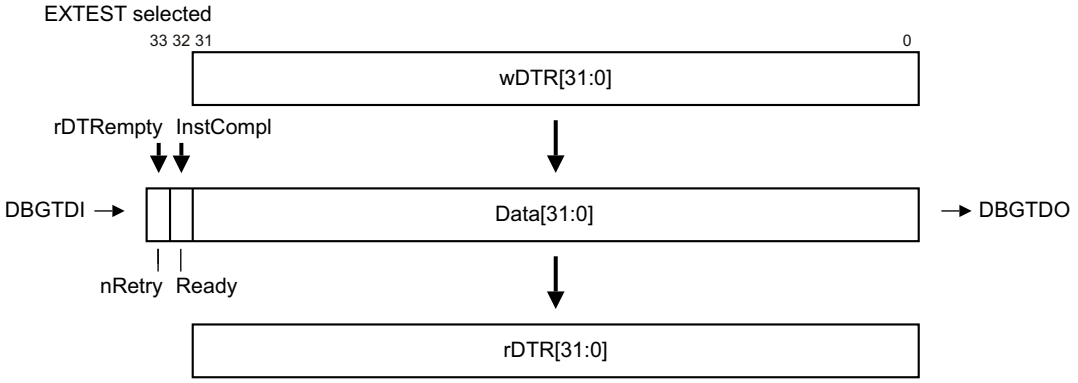

Figure 14-10 Scan chain 5 bit order, EXTEST selected

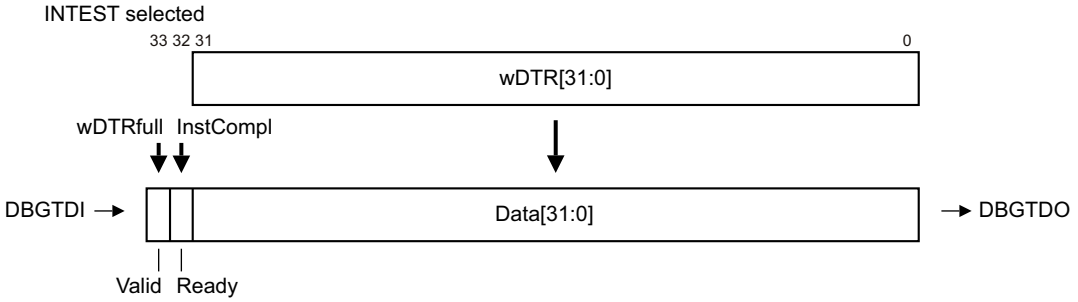

Figure 14-11 Scan chain 5 bit order, INTEST selected

You can use scan chain 5 for two purposes:

- As part of the *Debug Communications Channel* (DCC). The DBGTAP debugger uses scan chain 5 to exchange data with software running on the core. The software accesses the rDTR and wDTR using coprocessor instructions.

- For examining and modifying the processor state while the core is halted. For example, to read the value of an ARM register:
  1. Issue a `MCR cp14, 0, Rd, c0, c5, 0` instruction to the core to transfer the register contents to the CP14 debug c5 register.
  2. Scan out the wDTR.

The DBGTAP debugger can use the DSCR[13] execute ARM instruction enable bit to indicate to the core that it is going to use scan chain 5 as part of the DCC or for examining and modifying the processor state. DSCR[13] = 0 indicates DCC use. The behavior of the rDTR and wDTR registers, the sticky precise Data Abort, rDTRempty, wDTRfull, and InstCompl flags changes accordingly:

- DSCR[13] = 0:
  - The wDTRfull flag is set when the core writes a word of data to the DTR and cleared when the DBGTAP debugger goes through the Capture-DR state with INTEST selected. Valid indicates the state of the wDTR register, and is the captured version of wDTRfull. Although the value of wDTR is captured into the shift register, regardless of INTEST or EXTEST, wDTRfull is only cleared if INTEST is selected.
  - The rDTR empty flag is cleared when the DBGTAP debugger writes a word of data to the rDTR, and set when the core reads it. nRetry is the captured version of rDTRempty.
  - rDTR overwrite protection is controlled by the nRetry flag. If the nRetry flag is sampled clear, meaning that the rDTR is full, when going through the Capture-DR state, then the rDTR is not updated at the Update-DR state.
  - The InstCompl flag is always set.
  - The sticky precise Data Abort flag is Unpredictable. See *CP14 c1, Debug Status and Control Register (DSCR)* on page 13-7.
- DSCR[13] = 1:
  - The wDTR Full flag behaves as if DSCR[13] is clear. However, the Ready flag can be used for handshaking in this mode.
  - The rDTR Empty flag status behaves as if DSCR[13] is clear. However, the Ready flag can be used for handshaking in this mode.
  - rDTR overwrite protection is controlled by the Ready flag. If the InstCompl flag is sampled clear when going through Capture-DR, then the rDTR is not updated at the Update-DR state. This prevents an instruction that uses the rDTR as a source operand from having it modified before it has time to complete.
  - The InstCompl flag changes from 1 to 0 when an instruction is issued to the core, and from 0 to 1 when the instruction completes execution.
  - The sticky precise Data Abort flag is set on precise Data Aborts.

The behavior of the rDTR and wDTR registers, the sticky precise Data Abort, rDTRempty, wDTRfull, and InstCompl flags when the core changes state is as follows:

- The DSCR[13] execute ARM instruction enable bit must be clear when the core is not in Debug state. Otherwise, the behavior of the rDTR and wDTR registers, and the flags, is Unpredictable.
- When the core enters Debug state, none of the registers and flags are altered.
- When the DSCR[13] execute ARM instruction enable bit is changed from 0 to 1:
  1. None of the registers and flags are altered.
  2. Ready flag can be used for handshaking.

- The InstCompl flag must be set when the DSCR[13] execute ARM instruction enable bit is changed from 1 to 0. Otherwise, the behavior of the core is Unpredictable. If the DSCR[13] flag is cleared correctly, none of the registers and flags are altered.
- When the core leaves Debug state, none of the registers and flags are altered.

Scan chain 6

Purpose Embedded Trace Macrocell.

Length 1 + 7 + 32 = 40 bits.

Description This scan chain accesses the register map of the Embedded Trace Macrocell. See the description in the programmer's model chapter in the *Embedded Trace Macrocell Architecture Specification* for details of register allocation.

To access this scan chain you must select INTEST. Accesses to scan chain 6 with EXTEST selected are ignored. In scan chain 6 you must use the nRW bit, bit[39], to distinguish between reads and writes, as the *Embedded Trace Macrocell Architecture Specification describes*.

Note

For scan chain 6, the use of INTEST and EXTEST differs from their standard use that the start of this section describes.

Order Figure 14-12 shows the order of bits in scan chain 6.

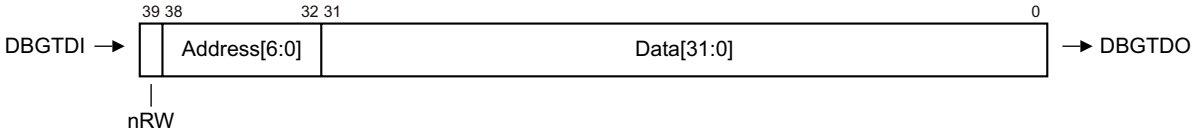

Figure 14-12 Scan chain 6 bit order

Scan chain 7

Purpose Debug.

Length 7 + 32 + 1 = 40 bits.

Description Scan chain 7 accesses the VCR, PC, BRPs, and WRPs. The accesses are performed with the help of read or write request commands. A read request copies the data held by the addressed register into scan chain 7. A write request copies the data held by the scan chain into the addressed register. When a request is finished the ReqCompl flag is set. The DBGTAP debugger must poll it and check it is set before another request can be issued. The exact behavior of the scan chain is as follows:

- Either INTEST or EXTEST must be selected. INTEST and EXTEST have the same meaning in this scan chain.

  Note

  For scan chain 7, the use of INTEST and EXTEST differs from the standard use that the start of this section describes.

- If the value captured by the Ready/nRW bit at the Capture-DR state is 1, the data that is being shifted in generates a request at the Update-DR state. The Address field indicates the register being accessed, see Table 14-2 on

page 14-19, the Data field contains the data to be written and the Ready/nRW bit holds the read/write information, 0=read and 1=write. If the request is a read, the Data field is ignored.

- When a request is placed, the Address and Data sections of the scan chain are frozen. That is, their contents are not shifted until the request is completed. This means that, if the value captured in the Ready/nRW field at the Capture-DR state is 0, the shifted-in data is ignored and the shifted-out value is all 0s.
- After a read request has been placed, if the DBGTAPSM goes through the Capture-DR state and a logic 1 is captured in the Ready/nRW field, this means that the shift register has also captured the requested register contents. Therefore, they are shifted out at the same time as the Ready/nRW bit. The Data field is corrupted as new data is shifted in.
- After a write request has been placed, if the DBGTAPSM goes through the Capture-DR state and a logic 1 is captured in the Ready/nRW field, this means that the requested write has completed successfully.
- If the Address field is all 0s, address of the NULL register, at the Update-DR state, then no request is generated.
- A request to a reserved register generates Unpredictable behavior.

Order Figure 14-13 shows the order of bits in scan chain 7.

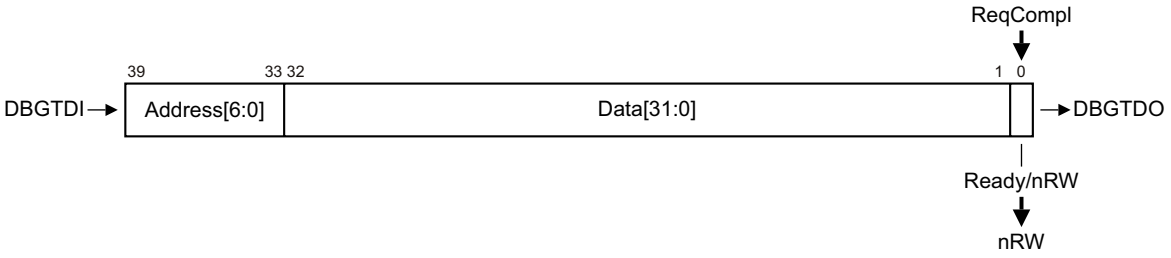

Figure 14-13 Scan chain 7 bit order

A typical sequence for writing registers is as follows:

1. Scan in the address of a first register, the data to write, and a 1 to indicate that this is a write request.
2. Scan in the address of a second register, the data to write, and a 1 to indicate that this is a write request.

   Scan out 40 bits. If Ready/nRW is 0, repeat this step. If Ready/nRW is 1, the first write request has completed successfully and the second has been placed.
3. Scan in the address 0. The rest of the fields are not important.

   Scan out 40 bits. If Ready/nRW is 0, repeat this step. If Ready/nRW is 1, the second write request has completed successfully. The scanned-in null request has avoided the generation of another request.

A typical sequence for reading registers is as follows:

1. Scan in the address of a first register and a 0 to indicate that this is a read request. The Data field is not important.
2. Scan in the address of a second register and a 0 to indicate that this is a read request.

Scan out 40 bits. If Ready/nRW is 0, then repeat this step. If Ready/nRW is 1, the first read request has completed successfully and the next scanned-out 32 bits are the requested value. The second read request was placed at the Update-DR state.

3. Scan in the address 0, the rest of the fields are not important.

   Scan out 40 bits. If Ready/nRW is 0, then repeat this step. If Ready/nRW is 1, the second read request has completed successfully and the next scanned-out 32 bits are the requested value. The scanned-in null request has avoided the generation of another request.

The register map is similar to the one of CP14 debug, and Table 14-2 lists it.

Table 14-2 Scan chain 7 register map

Address[6:0]	Register number	Abbreviation	Register name
b0000000	0	NULL	No request register
b0000001-b0000110	1-6	-	Reserved
b0000111	7	VCR	Vector catch register
b0001000	8	PC	Program counter
b0010011-b0111111	19-63	-	Reserved
b1000000-b1000101	64-69	BVRy[a]	Breakpoint value registers
b1000110-b1001111	70-79	-	Reserved
b1010000-b1010101	80-85	BCRy[a]	Breakpoint control registers
b1010110-b1011111	86-95	-	Reserved
b1100000-b1100001	96-97	WVRy[a]	Watchpoint value registers
b1100010-1b101111	98-111	-	Reserved
b1110000-b1110001	112-113	WCRy[a]	Watchpoint control registers
b1110010-b1111111	114-127	-	Reserved

a. y is the decimal representation for the binary number Address[3:0]

The following points apply to the use of scan chain 7:

- Every time there is a request to read the PC, a sample of its value is copied into scan chain 7. Writes are ignored. The sampled value can be used for profiling of the code. See *Interpreting the PC samples* on page 14-20 for details of how to interpret the sampled value.
- The external program counter sample register always reads `0xFFFFFFFF` in Debug state or when the core is in a mode when Non-invasive debug is not permitted.
- When accessing registers using scan chain 7, the processor can be either in Debug state or in normal state. This implies that breakpoints, watchpoints, and vector traps can be programmed through the Debug Test Access Port even if the processor is running.

Interpreting the PC samples

The PC values read correspond to instructions committed for execution, including those that failed their condition code. However, these values are offset as Table 13-22 on page 13-33 lists. These offsets are different for different processor states, so additional information is required:

- If a read request to the PC completes and Data[1:0] equals b00, the read value corresponds to an ARM state instruction whose 30 most significant bits of the offset address, instruction address + 8, are given in Data[31:2].
- If a read request to the PC completes and Data[0] equals b1, the read value corresponds to a Thumb state instruction whose 31 most significant bits of the offset address, instruction address + 4, are given in Data[31:1].
- If a read request to the PC completes and Data[1:0] equals b10, the read value corresponds to a Jazelle state instruction whose 30 most significant bits of its address are given in Data[31:2], the offset is 0. Because of the state encoding, the lower two bits of the Java address are not sampled. However, the information provided is enough for profiling the code.
- If the PC is read while the processor is in Debug state, the result is `0xFFFFFFFF`.

Scan chains 8-15

These scan chains are reserved.

Scan chains 16-31

These scan chains are unassigned.

14.6.6 Reset

The DBGTAP is reset either by asserting **DBGnTRST**, or by clocking it while DBGTAPSM is in the Test-Logic-Reset state. The processor, including CP14 debug logic, is not affected by these events. See *Reset modes* on page 9-10 and *CP14 registers reset* on page 13-25 for details.

14.7 Using the Debug Test Access Port

This section contains the following subsections:

- *Entering and leaving Debug state*
- *Executing instructions in Debug state*
- *Using the ITRsel IR instruction* on page 14-22
- *Transferring data between the host and the core* on page 14-23
- *Using the debug communications channel* on page 14-23
- *Target to host debug communications channel sequence* on page 14-24
- *Host to target debug communications channel* on page 14-24
- *Transferring data in Debug state* on page 14-25
- *Example sequences* on page 14-26.

14.7.1 Entering and leaving Debug state

Debug sequences on page 14-29 describes these debug sequences in detail.

14.7.2 Executing instructions in Debug state

When the processor is in Debug state, it can be forced to execute ARM state instructions using the DBGTAP. Two registers are used for this purpose, the *Instruction Transfer Register* (ITR) and the *Data Transfer Register* (DTR). The ITR is used to insert an instruction into the processor pipeline. An ARM state instruction can be loaded into this register using scan chain number 4. When the instruction is loaded, and INTEST or EXTEST is selected, and scan chain 4 or 5 is selected, the instruction can be issued to the core by making the DBGTAPSM go through the Run-Test/Idle state, provided certain conditions, that this section describes, are met. This mechanism enables re-executing the same instruction over and over without having to reload it. The DTR can be used in conjunction with the ITR to transfer data in and out of the core. For example, to read out the value of an ARM register:

1. issue an `MCR p14,0,Rd,c0,c5,0` instruction to the core to transfer the `<Rd>` contents to the c5 register
2. scan out the wDTR.

The DSCR[13] execute ARM instruction enable bit controls the activation of the ARM instruction execution mechanism. If this bit is cleared, no instruction is issued to the core when the DBGTAPSM goes through Run-Test/Idle. Setting this bit while the core is not in Debug state leads to Unpredictable behavior. If the core is in Debug state and this bit is set, the Ready and the sticky precise Data Abort flags condition the updates of the ITR and the instruction issuing, as *Scan chain 4, instruction transfer register (ITR)* on page 14-13 describes. As an example, this sequence stores out the contents of the ARM register R0:

1. Scan_N into the IR.
2. 1 into the SCREG.
3. INTEST into the IR.
4. Scan out the contents of the DSCR. This action clears the sticky precise Data Abort and sticky imprecise Data Abort flags and sticky Undefined bit.
5. EXTEST into the IR.
6. Scan in the previously read value with the DSCR[13] execute ARM instruction enable bit set.

7. Scan_N into the IR.
8. 4 into the SCREG.
9. EXTEST into the IR.
10. Scan the `MCR p14,0,R0,c0,c5,0` instruction into the ITR.
11. Go through the Run-Test/Idle state of the DBGTAPSM.
12. Scan_N into the IR.
13. 5 into the SCREG.
14. INTEST into the IR.
15. Scan out 34 bits. The 33rd bit indicates if the instruction has completed. If the bit is clear, repeat this step again.
16. The least significant 32 bits hold the contents of R0.

14.7.3 Using the ITRsel IR instruction

When the ITRsel instruction is loaded into the IR, at the Update-IR state, the DBGTAP controller behaves as if EXTEST and scan chain 4 are selected, but SCREG retains its value. It can be used to speed up certain debug sequences.

Figure 14-14 shows the effect of the ITRsel IR instruction.

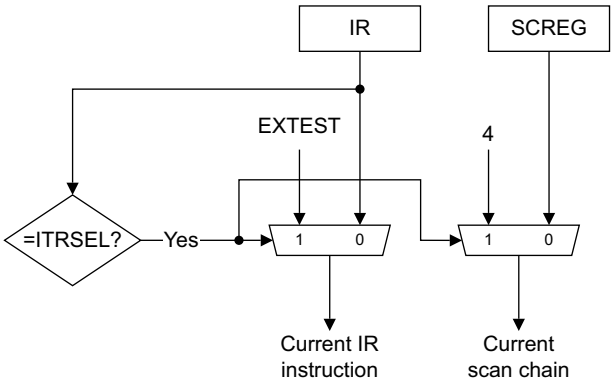

Figure 14-14 Behavior of the ITRsel IR instruction

Consider for example the preceding sequence to store out the contents of ARM register R0. This is the same sequence using the ITRsel instruction:

1. Scan_N into the IR.
2. 1 into the SCREG.
3. INTEST into the IR.
4. Scan out the contents of the DSCR. This action clears the sticky precise Data Abort and sticky imprecise Data Abort flags.
5. EXTEST into the IR.
6. Scan in the previously read value with the DSCR[13] execute ARM instruction enable bit set.
7. Scan_N into the IR.

8. 5 into the SCREG.
9. ITRsel into the IR. Now the DBGTAP controller works as if EXTEST and scan chain 4 is selected.
10. Scan the `MCR p14,0,R0,c0,c5,0` instruction into the ITR.
11. Go through the Run-Test/Idle state of the DBGTAPSM.
12. INTEST into the IR. Now INTEST and scan chain 5 are selected.
13. Scan out 34 bits. The 33rd bit indicates if the instruction has completed. If the bit is clear, repeat this step again.
14. The least significant 32 bits hold the contents of R0.

The number of steps has been reduced from 16 to 14. However, the bigger reduction comes when reading additional registers. Using the ITRsel instruction there are 6 extra steps, 9 to 14, compared with 10 extra steps, 7 to 16, in the first sequence.

14.7.4 Transferring data between the host and the core

There are two ways that a DBGTAP debugger can send or receive data from the core:

- using the DCC, when the processor is not in Debug state
- using the instruction execution mechanism that *Executing instructions in Debug state* on page 14-21 describes, when the core is in Debug state.

The following sections describe this:

- *Using the debug communications channel*.
- *Target to host debug communications channel sequence* on page 14-24
- *Host to target debug communications channel* on page 14-24
- *Transferring data in Debug state* on page 14-25
- *Example sequences* on page 14-26.

14.7.5 Using the debug communications channel

The DCC is defined as the set of resources that the external DBGTAP debugger uses to communicate with a piece of software running on the core.

The DCC in the processor is implemented using the two physically separate DTRs and a full/empty bit pair to augment each register, creating a bidirectional data port. One register can be read from the DBGTAP and is written from the processor. The other register is written from the DBGTAP and read by the processor. The full/empty bit pair for each register is automatically updated by the debug unit hardware, and is accessible to both the DBGTAP and to software running on the processor.

At the core side, the DCC resources are the following:

- CP14 debug register c5, DTR. Data coming from a DBGTAP debugger can be read by an MRC or STC instruction addressed to this register. The core can write to this register any data intended for the DBGTAP debugger, using an MCR or LDC instruction. Because the DTR comprises both a read, rDTR, and a write portion, wDTR, a piece of data written by the core and another coming from the DBGTAP debugger can be held in this register at the same time.

- Some flags and control bits at CP14 debug register c1, DSCR:

  DSCR[12] User mode access to DCC disable bit. If this bit is set, only privileged software can access the DCC. That is, access the DSCR and the DTR.

  DSCR[29] The wDTRfull flag. When clear, this flag indicates to the core that the wDTR is ready to receive data from the core.

  DSCR[30] The rDTRfull flag. When set, this flag indicates to the core that there is data available to read at the DTR.

At the DBGTAP side, the resources are the following:

- Scan chain 5. See *Scan chain 5* on page 14-15. The only part of this scan chain that it is not used for the DCC is the Ready flag. The rest of the scan chain is to be used in the following way:

  rDTR When the DBGTAPSM goes through the Update-DR state with EXTEST and scan chain 5 selected, and the nRetry flag set, the contents of the Data field are loaded into the rDTR. This is how the DBGTAP debugger sends data to the software running on the core.

  wDTR When the DBGTAPSM goes through the Capture-DR state with INTEST and scan chain 5 selected, the contents of the wDTR are loaded into the Data field of the scan chain. This is how the DBGTAP debugger reads the data sent by the software running on the core.

  Valid flag When set, this flag indicates to the DBGTAP debugger that the contents of the wDTR that it captured a short time ago are valid.

  nRetry flag When set, this flag indicates to the DBGTAP debugger that the scanned-in Data field has been successfully written into the rDTR at the Update-DR state.

14.7.6 Target to host debug communications channel sequence

The DBGTAP debugger can use the following sequence for receiving data from the core:

1. Scan_N into the IR.
2. 5 into the SCREG.
3. INTEST into the IR.
4. Scan out 34 bits of data. If the Valid flag is clear, repeat this step again.
5. The least significant 32 bits hold valid data.
6. Go to step 4 again to read out more data.

14.7.7 Host to target debug communications channel

The DBGTAP debugger can use the following sequence for sending data to the core:

1. Scan_N into the IR.
2. 5 into the SCREG.
3. EXTEST into the IR.
4. Scan in 34 bits, the least significant 32 holding the word to be sent. At the same time, 34 bits were scanned out. If the nRetry flag is clear, repeat this step again.

5. Now the data has been written into the rDTR. Go to step 4 again to send in more data.

14.7.8 Transferring data in Debug state

When the core is in Debug state, the DBGTAP debugger can transfer data in and out of the core using the instruction execution facilities that *Executing instructions in Debug state* on page 14-21 describes in addition to scan chain 5. You must ensure that the DSCR[13] execute ARM instruction enable bit is set for the instruction execution mechanism to work. When it is set, the interface for the DBGTAP debugger consists of the following:

- Scan chain 4. See *Scan chain 4, instruction transfer register (ITR)* on page 14-13. It is used for loading an instruction and for monitoring the status of the execution:

  ITR When the DBGTAPSM goes through the Update-DR state with EXTEST and scan chain 4 selected, and the Ready flag set, the ITR is loaded with the least significant 32 bits of the scan chain.

  InstCompl flag When clear, this flag indicates to the DBGTAP debugger that the last issued instruction has not yet completed execution. While Ready, captured version of InstCompl, is clear, no updates of the ITR and the rDTR occur and the instruction execution mechanism is disabled. No instruction is issued when going through Run-Test/Idle.

- Scan chain 5. See *Scan chain 5* on page 14-15. It is used for writing in or reading out the data and for monitoring the state of the execution:

  rDTR When the DBGTAPSM goes through the Update-DR state with EXTEST and scan chain 5 selected, and the Ready flag set, the contents of the Data field are loaded into the rDTR.

  wDTR When the DBGTAPSM goes through the Capture-DR state with INTEST or EXTEST selected, the contents of the wDTR are loaded into the Data field of the scan chain.

  InstCompl flag When clear, this flag indicates to the DBGTAP debugger that the last issued instruction has not yet completed execution. While Ready, captured version of InstCompl, is clear, no updates of the ITR and the rDTR occur and the instruction execution mechanism is disabled. No instruction is issued when going through Run-Test/Idle.

- Some flags and control bits at CP14 debug register c1, DSCR:

  DSCR[13] Execute ARM instruction enable bit. This bit must be set for the instruction execution mechanism to work.

  Sticky precise Data Abort flag

  DSCR[6]. When set, this flag indicates to the DBGTAP debugger that a precise Data Abort occurred while executing an instruction in Debug state. While this bit is set, the instruction execution mechanism is disabled. When this flag is set InstCompl stays HIGH, and additional attempts to execute an instruction appear to succeed but do not execute.

  Sticky imprecise Data Abort flag

  DSCR[7]. When set, this flag indicates to the DBGTAP debugger that an imprecise Data Abort occurred while executing an instruction in Debug state. This flag does not disable the Debug state instruction execution.

Sticky Undefined flag

DSCR[8]. When set, this flag indicates to the DBGTAP debugger that an Undefined exception occurred while executing an instruction in Debug state. This flag does not disable the Debug state instruction execution.

14.7.9 Example sequences

This section includes some example sequences to illustrate how to transfer data between the DBGTAP debugger and the core when it is in Debug state. The examples are related to accessing the processor memory.

Target to host transfer

The DBGTAP debugger can use the following sequence for reading data from the processor memory system. The sequence assumes that the ARM register R0 contains a pointer to the address of memory where the read has to start:

1. Scan_N into the IR.
2. 1 into the SCREG.
3. INTEST into the IR.
4. Scan out the contents of the DSCR. This clears the sticky precise Data Abort, sticky imprecise Data Abort flags, and sticky Undefined flags.
5. Scan_N into the IR.
6. 4 into the SCREG.
7. EXTEST into the IR.
8. Scan in the `LDC p14,c5,[R0],#4` instruction into the ITR.
9. Scan_N into the IR.
10. 5 into the SCREG.
11. INTEST into the IR.
12. Go through Run-Test/Idle state. The instruction loaded into the ITR is issued to the processor pipeline.
13. Scan out 34 bits of data. If the Ready flag is clear, repeat this step again.
14. The instruction has completed execution. Store the least significant 32 bits.
15. Go to step 13 again for reading out more data.
16. Scan_N into the IR.
17. 1 into the SCREG.
18. INTEST into the IR.
19. Scan out the contents of the DSCR. This clears the sticky precise Data Abort and sticky imprecise Data Abort and sticky Undefined flags. If the sticky precise Data Abort is set, this means that during the sequence one of the instructions caused a precise Data Abort. Not all the instructions that follow are executed. Register R0 points to the next word to be read, and after the cause for the abort has been fixed the sequence resumes at step 5.

Note

If the sticky imprecise Data Aborts flag is set, an imprecise Data Abort has occurred and the sequence restarts at step 1 after the cause of the abort is fixed and R0 is reloaded.

Host to target transfer

The DBGTAP debugger can use the following sequence for writing data to the processor memory system. The sequence assumes that the ARM register R0 contains a pointer to the address of memory where the write has to start:

1. Scan_N into the IR.
2. 1 into the SCREG.
3. INTEST into the IR.
4. Scan out the contents of the DSCR. This clears the sticky precise Data Abort, sticky imprecise Data Abort, and sticky Undefined flags.
5. Scan_N into the IR.
6. 4 into the SCREG.
7. EXTEST into the IR.
8. Scan in the `STC p14,c5,[R0],#4` instruction into the ITR.
9. Scan_N into the IR.
10. 5 into the SCREG.
11. EXTEST into the IR.
12. Scan in 34 bits, the least significant 32 holding the word to be sent. At the same time, 34 bits are scanned out. If the Ready flag is clear, repeat this step.
13. Go through Run-Test/Idle state.
14. Go to step 12 again for writing in more data.
15. Scan in 34 bits. All the values are don't care. At the same time, 34 bits are scanned out. If the Ready flag is clear, repeat this step. The don't care value is written into the rDTR, Update-DR state, immediately after Ready is seen set, Capture-DR state. However, the STC instruction is not re-issued because the DBGTAPSM does not go through Run-Test/Idle.
16. Scan_N into the IR.
17. 1 into the SCREG.
18. INTEST into the IR.
19. Scan out the contents of the DSCR. This clears the sticky precise Data Abort and sticky imprecise Data Abort flags. If the sticky precise Data Abort is set, this means that during the sequence one of the instructions caused a precise Data Abort. All the instructions that follow are not executed. Register R0 points to the next word to be written, and after the cause for the abort has been fixed, the sequences resumes at step 5.

Note

If the sticky imprecise Data Abort flag is set, an imprecise Data Abort has occurred and the sequence restarts at step 1 after the cause of the abort is fixed and c0 is reloaded.

14.8 Debug sequences

This section describes how to debug a program running on the processor using a DBGTAP debugger device such as RealView ICE. In Halting debug-mode, the processor stops when a debug event occurs enabling the DBGTAP debugger to do the following:

1. Perform a Data Synchronization Barrier operation to ensure imprecise data aborts are recognized and DSCR[19] is set.
2. Determine and modify the current state of the processor and memory.
3. Set up breakpoints, watchpoints, and vector traps.
4. Restart the processor.

You enable this mode by setting CP14 debug DSCR[14] bit. Only the DBGTAP debugger can do this. From here, it is assumed that the debug unit is in Halting debug-mode. *Monitor debug-mode debugging* on page 14-42 describes the monitor debug-mode debugging.

14.8.1 Debug macros

The debug code sequences in this section are written using a fixed set of macros. The mapping of each macro into a debug scan chain sequence is given in this section.

Scan_N <n>

Select scan chain register number <n>:

1. Scan the Scan_N instruction into the IR.
2. Scan the number <n> into the DR.

INTEST

1. Scan the INTEST instruction into the IR.

EXTEST

1. Scan the EXTEST instruction into the IR.

ITRsel

1. Scan the ITRsel instruction into the IR.

Restart

1. Scan the Restart instruction into the IR.
2. Go to the DBGTAP controller Run-Test/Idle state so that the processor exits Debug state.

INST <instr> [stateout]

Go through Capture-DR, go to Shift-DR, scan in an ARM instruction to be read and executed by the core and scan out the Ready flag, go through Update-DR. The ITR, scan chain 4, and EXTEST must be selected when using this macro.

1. Scan in:
   - Any value for the InstCompl flag. This bit is read-only.

- 32-bit assembled code of the instruction, instr, to be executed, for ITR[31:0].

2. The following data is scanned out:
   - The value of the Ready flag, to be stored in stateout.
   - 32 bits to be ignored. The ITR is write-only.

DATA <datain> [<stateout> [dataout]]

Go through Capture-DR, go to Shift-DR. Scan in a data item and scan out another one, go through Update-DR. Either the DTR, scan chain 5, or the DSCR, scan chain 1, must be selected when using this macro.

1. If scan chain 5 is selected, scan in:
   - Any value for the nRetry or Valid flag. These bits are read-only.
   - Any value for the InstCompl flag. This bit is read-only.
   - 32-bit datain value for rDTR[31:0].
2. The following data is scanned out:
   - The contents of wDTR[31:0], to be stored in dataout.
   - If the DSCR[13] execute ARM instruction enable bit is set, the value of the Ready flag is stored in stateout.
   - If the DSCR[13] execute ARM instruction enable bit is clear, the nRetry or Valid flag, depending on whether EXTEST or INTEST is selected, is stored in stateout.
3. If scan chain 1 is selected, scan in:
   - 32-bit datain value for DSCR[31:0].

   Stateout and dataout fields are not used in this case.

DATAOUT <dataout>

1. Scan out a data value. DSCR, scan chain 1, and INTEST must be selected when using this macro.
2. If scan chain 1 is selected, scan out the contents of the DSCR, to be stored in dataout.
3. The scanned-in value is discarded, because INTEST is selected.

REQ <address> <data> <nR/W> [<stateout> [dataout]]

Go through Capture-DR, go to Shift-DR, scan in a request and scan out the result of the former one, go through Update-DR. Scan chain 7, and either INTEST or EXTEST, must be selected when using this macro.

1. Scan in:
   - 7-bit address value for Address[6:0]
   - 32-bit data value for Data[31:0]
   - 1-bit nR/W value, 0 for read and 1 for write, for the Ready/nRW field.
2. Scan out:
   - the value of the Ready/nRW bit, to be stored in stateout
   - the contents of the Data field, to be stored in dataout.

RTI

1. Go through Run-Test/Idle DBGTAPSM state. This forces the execution of the instruction currently loaded into the ITR, provided the execute ARM instruction enable bit, DSCR[13], is set, the Ready flag was captured as set, and the sticky precise Data Abort flag is cleared.

14.8.2 General setup

You must setup the following control bits before DBGTAP debugging can take place:

- DSCR[14] Debug-mode select bit must be set to 1.
- DSCR[6] sticky precise Data Abort flag must be cleared down, so that aborts are not detected incorrectly immediately after startup.

The DSCR must be read, the DSCR[14] bit set, and the new value written back. The action of reading the DSCR automatically clears the DSCR[6] sticky precise Data Abort flag. All individual breakpoints, watchpoints, and vector catches reset disabled on power-up.

14.8.3 Forcing the processor to halt

Scan the Halt instruction into the DBGTAP controller IR and go through Run-Test/Idle.

14.8.4 Entering Debug state

To enter Debug state you must:

1. Check whether the core has entered Debug state, as follows:

```
SCAN_N  1                       ; select DSCR
INTEST
LOOP
    DATAOUT readDSCR
UNTIL    readDSCR[0]==1             ; until Core Halted bit is set
```

2. Save DSCR, as follows:

```
DATAOUT  readDSCR
Save value in readDSCR
```

3. Save wDTR, in case it contains some data, as follows:

```
SCAN_N  5                       ; select DTR
INTEST
DATA    0x00000000  Valid wDTR
If Valid==1 then Save value in wDTR
```

4. Set the DSCR[13] execute ARM instruction enable bit, so instructions can be issued to the core from now:

```
SCAN_N  1                       ; select DSCR
EXTEST
DATA modifiedDSCR               ; modifiedDSCR equals readDSCR with bit
                                ; DSCR[13] set
```

5. Before executing any instruction in Debug state you have to drain the write buffer. This ensures that no imprecise Data Aborts can return at a later point:

```
SCAN_N  4                       ; select ITR
INST    MCR p15,0,Rd,c7,c10,4   ; Data Synchronization Barrier
LOOP
    LOOP
        SCAN_N  4               ; select DTR
```

```
        RTI
        INST 0x0 Ready
    Until Ready == 1
    SCAN_N 1
    DATAOUT readDSCR
Until readDSCR[7]==1
SCAN_N 4
INST NOP                        ; NOP takes the
RTI                             ; imprecise Data Aborts
LOOP
    INST 0 Ready
Until Ready == 1
SCAN_N 1
DATAOUT readDSCR                ; clears DSCR[7]
```

6. Store out R0. It is going to be used to save the rDTR. Use the standard sequence of *Reading a current mode ARM register in the range R0-R14* on page 14-34. Scan chain 5 and INTEST are now selected.

7. Save the rDTR and the rDTRempty bit in three steps:

   a. The rDTRempty bit is the inverted version of DSCR[30], saved in step 2. If DSCR[30] is clear, register empty, there is no requirement to read the rDTR, go to 7.

   b. Transfer the contents of rDTR to R0:

```
ITRSEL                          ; select the ITR and EXTEST
INST   MRC p14,0,R0,c0,c5,0     ; instruction to copy CP14's debug
                                ; register c5 into R0
RTI
LOOP
    INST 0x00000000 Ready
UNTIL  Ready==1                 ; wait until the instruction ends
```

   c. Read R0 using the standard sequence of *Reading a current mode ARM register in the range R0-R14* on page 14-34.

8. Store out CPSR using the standard sequence of *Reading the CPSR/SPSR* on page 14-35.

9. Store out PC using the standard sequence of *Reading the PC* on page 14-36.

10. Adjust the PC to enable you to resume execution later:

    - subtract `0x8` from the stored value if the processor was in ARM state when entering Debug state
    - subtract `0x4` from the stored value if the processor was in Thumb state when entering Debug state
    - subtract `0x0` from the stored value if the processor was in Jazelle state when entering Debug state.

    These values are not dependent on the Debug state entry method. See *Behavior of the PC in Debug state* on page 13-38. The entry state can be determined by examining the T and J bits of the CPSR.

11. Cache and MMU preservation measures must also be taken here. This includes saving all the relevant CP15 registers using the standard coprocessor register reading sequence that *Coprocessor register reads and writes* on page 14-38 describes.

14.8.5 Leaving Debug state

To leave Debug state:

1. Restore standard ARM registers for all modes, except R0, PC, and CPSR.

2. Cache and MMU restoration must be done here. This includes writing the saved registers back to CP15.

3. Ensure that rDTR and wDTR are empty:

```
ITRSE                           ; select the ITR and EXTEST
INST    MCR p14,0,R0,c0,c5,0    ; instruction to copy R0 into
                                ; CP14 debug register c5
RTI
LOOP
    INST 0x00000000 Ready
UNTIL   Ready==1                ; wait until the instruction ends
SCAN_N 5
INTEST
DATA    0x0 Valid wDTR
```

4. If the wDTR did not contain any valid data on Debug state entry go to step 5. Otherwise, restore wDTRfull and wDTR, uses R0 as a temporary register, in two steps.

   a. Load the saved wDTR contents into R0 using the standard sequence of *Writing a current mode ARM register in the range R0-R14* on page 14-34. Now scan chain 5 and EXTEST are selected

   b. Transfer R0 into wDTR:

```
ITRSEL                          ; select the ITR and EXTEST
INST    MCR p14,0,R0,c0,c5,0    ; instruction to copy R0 into
                                ; CP14 debug register c5
RTI
LOOP
    INST 0x00000000 Ready
UNTIL   Ready==1                ; wait until the instruction ends
```

5. Restore CPSR using the standard CPSR writing sequence that *Writing the CPSR/SPSR* on page 14-35 describes.

6. Restore the PC using the standard sequence of *Writing the PC* on page 14-36.

7. Restore R0 using the standard sequence of *Writing a current mode ARM register in the range R0-R14* on page 14-34. Now scan chain 5 and EXTEST are selected.

8. Restore the DSCR with the DSCR[13] execute ARM instruction enable bit clear, so no more instructions can be issued to the core:

```
SCAN_N   1                      ; select DSCR
EXTEST
DATA modifiedDSCR               ; modifiedDSCR equals the saved contents
                                ; of the DSCR with bit DSCR[13] clear
```

9. If the rDTR did not contain any valid data on Debug state entry, go to step 10. Otherwise, restore the rDTR and rDTRempty flag:

```
SCAN_N   5                      ; select DTR
EXTEST
DATA    Saved_rDTR              ; rDTRempty bit is automatically cleared
                                ; as a result of this action
```

10. Restart processor:

```
RESTART
```

11. Wait until the core is restarted:

```
SCAN_N   1                      ; select DSCR
INTEST
LOOP
    DATAOUT readDSCR
UNTIL   readDSCR[1]==1          ; until Core Restarted bit is set
```

14.8.6 Reading a current mode ARM register in the range R0-R14

Use the following sequence to read a current mode ARM register in the range R0-R14:

```
SCAN_N   5                      ; select DTR
ITRSEL                          ; select the ITR and EXTEST
INST    MCR p14,0,Rd,c0,c5,0    ; instruction to copy Rd into CP14 debug
                                ; register c5
RTI
INTEST                          ; select the DTR and INTEST
LOOP
    DATA  0x00000000 Ready readData
UNTIL   Ready==1                ; wait until the instruction ends
Save value in readData
```

Note

Register R15 cannot be read in this way because the effect of the required MCR is to take an Undefined exception.

14.8.7 Writing a current mode ARM register in the range R0-R14

Use the following sequence to write a current mode ARM register in the range R0-R14:

```
SCAN_N  5                           ; select DTR
ITRSEL                              ; select the ITR and EXTEST
INST    MRC p14,0,Rd,c0,c5,0        ; instruction to copy CP14 debug
                                    ; register c5 into Rd
EXTEST                              ; select the DTR and EXTEST
DATA    Data2Write
RTI
LOOP
    DATA  0x00000000 Ready
UNTIL   Ready==1                    ; wait until the instruction ends
```

Note

Register R15 cannot be written in this way because the MRC instruction used updates the CPSR flags rather than the PC.

14.8.8 Reading the CPSR/SPSR

Here R0 is used as a temporary register:

1. Move the contents of CPSR/SPSR to R0.

```
SCAN_N  5                       ; select DTR
ITRSEL                          ; select the ITR and EXTEST
INST    MRS R0,CPSR             ; or SPSR
RTI
LOOP
    INST 0x00000000 Ready
UNTIL   Ready==1                ; wait until the instruction ends
```

2. Perform the read of R0 using the standard sequence that *Reading a current mode ARM register in the range R0-R14* on page 14-34 describes. Scan chain 5 and ITRsel are already selected.

14.8.9 Writing the CPSR/SPSR

Here R0 is used as a temporary register:

1. Load the required value into R0 using the standard sequence that *Writing a current mode ARM register in the range R0-R14* on page 14-34 describes. Now scan chain 5 and EXTEST are selected.

2. Move the contents of R0 to CPRS/SPRS:

```
ITRSEL                          ; select the ITR and EXTEST
INST    MSR CPSR,R0             ; or SPSR
RTI
LOOP
    INST 0x00000000 Ready
UNTIL   Ready==1                ; wait until the instruction ends
```

This instruction can modify the T and J bits. They have no effect in the execution of instructions while in Debug state but take effect when the core leaves Debug state.

The CPSR mode and control bits can be written in User mode when the core is in Debug state and the core is in a Non-secure world or the **SPIDEN** signal is asserted. This is essential so that the debugger can change mode and then get at the other banked registers.

14.8.10 Reading the PC

Here R0 is used as a temporary register:

1. Move the contents of the PC to R0:

```
ITRSEL                          ; select the ITR and EXTEST
INST    MOV R0,PC
RTI
LOOP
    INST 0x00000000 Ready
UNTIL   Ready==1                ; wait until the instruction ends
```

2. Read the contents of R0 using the standard sequence that *Reading a current mode ARM register in the range R0-R14* on page 14-34 describes.

14.8.11 Writing the PC

Here R0 is used as a temporary register:

1. Load R0 with the address to resume using the standard sequence that *Writing a current mode ARM register in the range R0-R14* on page 14-34 describes. Now scan chain 5 and EXTEST are selected.

2. Move the contents of R0 to the PC:

```
ITRSEL                          ; select the ITR and EXTEST
INST    MOV PC,R0
RTI
LOOP
    INST 0x00000000 Ready
UNTIL   Ready==1                ; wait until the instruction ends
```

14.8.12 General notes about reading and writing memory

The word-based read and write sequences are substantially more efficient than the halfword and byte sequences. This is because the ARM LDC and STC instructions always perform word accesses, and this can be used for efficient access to word width memory. Halfword and byte accesses must be done with a combination of loads or stores, and coprocessor register transfers. This is much less efficient. When writing data, the Instruction Cache might become incoherent. In those cases, the appropriate part of the Instruction Cache must be invalidated. In particular, the Instruction Cache must be invalidated before setting a software breakpoint or downloading code.

14.8.13 Reading memory as words

This sequence is optimized for a long sequential read. This sequence assumes that R0 has been set to the address to load data from prior to running this sequence. R0 is post-incremented so that it can be used by successive reads of memory.

1. Load and issue the LDC instruction:

```
SCAN_N  5                       ; select DTR
ITRSEL                          ; select the ITR and EXTEST
INST    LDC p14,c5,[R0],#4      ; load the content of the position of
                                ; memory pointed by R0 into wDTR and
                                ; increment R0 by 4
RTI
```

2. The DTR is selected to read the data:

```
INTEST                          ; select the DTR and INTEST
```

3. This loop keeps on reading words, but it stops before the latest read. It is skipped if there is only one word to read:

```
FOR(i=1; i <= (Words2Read-1); i++) DO
    LOOP
        DATA 0x00000000 Ready readData   ; gets the result of
                                         ; the previous read
        RTI                              ; issues the next read
    UNTIL Ready==1                       ; wait until the instruction ends
    Save value in readData
ENDFOR
```

4. Wait for the last read to finish:

```
LOOP
    DATA 0x00000000 Ready readData
UNTIL Ready==1                           ; wait until instruction ends
Save value in readData
```

5. Now check whether an abort occurred:

```
SCAN_N  1                        ; select DSCR
INTEST
DATAOUT DSCR                     ; this action clears the DSCR[6] flag
```

6. Scan out the contents of the DSCR. This clears the sticky precise Data Abort and sticky imprecise Data Abort flags. If the sticky precise Data Abort is set, this means that during the sequence one of the instructions caused a precise Data Abort. All the instructions that follow are not executed. Register R0 points to the next word to be written, and after the cause for the abort has been fixed the sequences resumes at step 1.

Note

If the sticky imprecise Data Aborts flag is set, an imprecise Data Abort has occurred and the sequence restarts at step 1 after the cause of the abort is fixed and R0 is reloaded.

14.8.14 Writing memory as words

This sequence is optimized for a long sequential write. This sequence assumes that R0 has been set to the address to store data to prior to running this sequence. Register R0 is post-incremented so that it can be used by successive writes to memory:

1. The instruction is loaded:

```
SCAN_N  5                        ; select DTR
ITRSEL                           ; select the ITR and EXTEST
INST    STC p14,c5,[R0],#4       ; store the contents of rDTR into the
                                 ; position of memory pointed by R0 and
                                 ; increment it by 4
EXTEST                           ; select the DTR and EXTEST
```

2. This loop writes all the words:

```
FOR (i=1; i <= Words2Write; i++) DO
    LOOP
        DATA Data2Write Ready
        RTI
    UNTIL Ready==1               ; wait until instruction ends
ENDFOR
INTEST                           ; deselect the DTR
```

3. Wait for the last write to finish:

```
LOOP
    DATA 0x00000000 Ready
```

```
UNTIL Ready==1                  ; wait until instruction ends
```

4. Check for aborts, as *Reading memory as words* on page 14-36 describes.

14.8.15 Reading memory as halfwords or bytes

The above sequences cannot be used to transfer halfwords or bytes because LDC and STC instructions always transfer whole words. Two operations are required to complete a halfword or byte transfer, from memory to ARM register and from ARM register to CP14 debug register. Therefore, performance is decreased because the load instruction cannot be kept in the ITR. This sequence assumes that R0 has been set to the address to load data from prior to running the sequence. Register R0 is post-incremented so that it can be used by successive reads of memory. Register R1 is used as a temporary register:

1. Load and issue the LDRH or LDRB instruction:

```
ITRSEL                          ; select the ITR and EXTEST
INST    LDRH R1,[R0],#2         ; LDRB R1,[R0],#1 for byte reads
RTI
LOOP
    INST 0x00000000 Ready
UNTIL Ready==1                  ; wait until instruction ends
```

2. Use the standard sequence that *Reading a current mode ARM register in the range R0-R14* on page 14-34 describes on register R1. Now scan chain 5 and INTEST are selected.

3. If there are more halfwords or bytes to be read go to 1.

4. Check for aborts, as *Reading memory as words* on page 14-36 describes.

14.8.16 Writing memory as halfwords/bytes

This sequence assumes that R0 has been set to the address to store data to prior to running this sequence. Register R0 is post-incremented so that it can be used by successive writes to memory. Register R1 is used as a temporary register:

1. Write the halfword/byte onto R1 using the standard sequence that *Writing a current mode ARM register in the range R0-R14* on page 14-34 describes. Scan chain 5 and EXTEST are selected.

2. Write the contents of R1 to memory:

```
ITRSEL                          ; select the ITR and EXTEST
INST    STRH R1,[R0],#2         ; STRB R1,[R0],#1 for byte writes
RTI
LOOP
    INST 0x00000000 Ready
UNTIL Ready==1                  ; wait until instruction ends
```

3. If there are more halfwords or bytes to be read go to 1.

4. Now check for aborts as *Reading memory as words* on page 14-36 describes.

14.8.17 Coprocessor register reads and writes

The processor can execute coprocessor instructions while in Debug state. Therefore, the straightforward method to transfer data between a coprocessor and the DBGTAP debugger is using an ARM register temporarily. For this method to work, the coprocessor must be able to transfer all its registers to the core using coprocessor transfer instructions.

14.8.18 Reading coprocessor registers

1. Load the value into ARM register R0:

```
ITRSEL                          ; select the ITR and EXTEST
INST    MRC px,y,R0,ca,cb,z
RTI
LOOP
    INST 0x00000000 Ready
UNTIL Ready==1                  ; wait until instruction ends
```

2. Use the standard sequence that *Reading a current mode ARM register in the range R0-R14* on page 14-34 describes.

14.8.19 Writing coprocessor registers

1. Write the value onto R0, using the standard sequence. See *Writing a current mode ARM register in the range R0-R14* on page 14-34 for more details. Scan chain 5 and EXTEST are selected.

2. Transfer the contents of R0 to a coprocessor register:

```
ITRSEL                          ; select the ITR and EXTEST
INST    MCR px,y,R0,ca,cb,z
RTI
LOOP
    INST 0x00000000  Ready
UNTIL Ready==1                  ; wait until instruction ends
```

14.9 Programming debug events

This section describes the following operations:

- *Reading registers using scan chain 7*
- *Writing registers using scan chain 7*
- *Setting breakpoints, watchpoints and vector traps*
- *Setting software breakpoints* on page 14-41.

14.9.1 Reading registers using scan chain 7

A typical sequence for reading registers using scan chain 7 is as follows:

```
SCAN_N  7                       ; select ITR
EXTEST
REQ 1stAddr2Rd0 0               ;read request for register 1stAddr2read
FOR(i=2; i <= Words2Read; i++) DO
    LOOP
        REQ ithAddr2Rd 0 0 Ready readData
                                ; ith read request while waiting
    UNTIL Ready==1              ; wait until the previous request completes
    Save value in readData
ENDFOR
LOOP
    REQ 0 0 0 Ready readData    ; null request while waiting
UNTIL Ready==1                  ; wait until last request completes
Save value in readData
```

14.9.2 Writing registers using scan chain 7

A typical sequence for writing to a register using scan chain 7 is as follows:

```
SCAN_N  7                       ; select ITR
EXTEST
REQ 1stAddr2Wr 1stData2Wr 0b1   ; write request for register 1stAddr2write
FOR(i=2; i <= Words2Write; i++) DO
    LOOP
        REQ ithAddr2Wr ithData2Wr 1 Ready
                                ; ith write request while waiting
    UNTIL Ready==1              ; wait until the previous request completes
ENDFOR
LOOP
    REQ 0 0 0 Ready             ; null request while waiting
UNTIL Ready==1                  ; wait until last request completes
```

14.9.3 Setting breakpoints, watchpoints and vector traps

You can program a vector catch debug event by writing to CP14 debug vector catch register.

You can program a breakpoint debug event by writing to CP14 debug 64-69 breakpoint value registers and CP14 debug 80-84 breakpoint control registers.

You can program a watchpoint debug event by writing to CP14 debug 96-97 watchpoint value registers and CP14 debug 112-113 watchpoint control registers.

Note

An External Debugger can access the CP14 debug registers whether the processor is in Debug state or not, so these debug events can be programmed on-the-fly, while the processor is in ARM/Thumb/Jazelle state.

See *Setting breakpoints, watchpoints, and vector catch debug events* on page 13-45 for the sequences of register accesses required to program these software debug events. See *Writing registers using scan chain 7* on page 14-40 to learn how to access CP14 debug registers using scan chain 7.

14.9.4 Setting software breakpoints

To set a software breakpoint on a certain Virtual Address, a debugger must go through the following steps:

1. Read memory location and save actual instruction.
2. Write the BKPT instruction to the memory location.
3. Read memory location again to check that the BKPT instruction got written.
4. If it is not written, determine the reason.

All of these can be done using the previously described sequences.

Note

Cache coherency issues might arise when writing a BKPT instruction. See *Debugging in a cached system* on page 13-43.

14.10 Monitor debug-mode debugging

If DSCR[15:14] b10 selecting Monitor debug-mode, then the processor takes an exception, rather than halting, when a software debug event occurs. See *Halting debug-mode debugging* on page 13-50 for details. When the exception is taken, the handler uses the DCC to transmit status information to, and receive commands from the host using a DBGTAP debugger. Monitor debug-mode is essential in real-time systems when the core cannot be halted to collect information.

14.10.1 Receiving data from the core

```
SCAN_N  5                       ; select DTR
INTEST
FOREACH Data2Read
    LOOP
        DATA 0x00000000 Valid readData
    UNTIL Valid==1              ; wait until instruction ends
    Save value in readData
END
```

14.10.2 Sending data to the core

```
SCAN_N  5                       ; select DTR
EXTEST
FOREACH Data2Write
    LOOP
        DATA Data2Write nRetry
    UNTIL nRetry==1             ; wait until instruction ends
END
```