

Chapter 7

Debugging Support

The DSP56300 modules and features for debugging applications during system development are as follows:

- *JTAG Test Access Port (TAP)*: Provides the TAP and Boundary Scan functionality based on the *IEEE Standard Test Access Port and Boundary-Scan Architecture* (IEEE 1149.1), which can test a circuit board containing a DSP56300 family chip including signal levels at the chip-to-board interface (that is, the boundary), but not the internal chip functions. The TAP also provides external access to the On-Chip Emulation (OnCE™) module
- *OnCE module*: Debugs software used with a DSP56300 family device and tests the hardware interface. The OnCE module has one dedicated external pin connection, the Debug Event (\overline{DE}) pin. All other communication with the module occurs through the TAP pins.
- *Address Trace Mode*: This feature, enabled by the ATE bit in the Operating Mode Register (OMR), allows tracing of internal accesses by monitoring the external address lines (A[23 – 0] or A[17–0]).

The debugging interface uses six interface signals. As described in the IEEE 1149.1 standard, the JTAG TAP requires a minimum of four pins to support the TDI, TDO, TCK, and TMS signals. The DSP56300 family also provides a pin for the optional \overline{TRST} signal. The OnCE module uses one pin for the \overline{DE} signal. **Table 7-1** describes the signals.

Table 7-1. Debugging Control Signals

Name	Abbrev.	Type	Module	Signal Description
Test Clock	TCK	Input	TAP	TCK is the external clock that synchronizes the test logic.
Test Mode Select	TMS	Input	TAP	TMS sequences the TAP controller state machine. TMS is sampled on the rising edge of TCK and has an internal pull-up resistor.
Test Data Input	TDI	Input	TAP	Receives serial test instruction and data, which is sampled on the rising edge of TCK and has an internal pull-up resistor. Register values are shifted in Least Significant Bit (LSB) first.

Table 7-1. Debugging Control Signals (Continued)

Name	Abbrev.	Type	Module	Signal Description
Test Data Output	TDO	Output	TAP	The serial output for test instructions and data. TDO is tri-stateable and is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK. Register values are shifted out LSB first.
Test Reset	$\overline{\text{TRST}}$	Input	TAP	Initializes the test controller asynchronously. $\overline{\text{TRST}}$ has an internal pull-up resistor. To reset the TAP controller synchronously, use TCK to clock five consecutive 1s into TMS. To reset the remaining parts of the DSP core and the peripherals (or in some cases, such as the HI32, only the internal portion of a peripheral), use the RESET input signal.
Debug Event	$\overline{\text{DE}}$	Input or Output	OnCE	<p>An open-drain signal providing, as an input, a means of entering the Debug mode of operation from an external command controller, and, as an output, a means of acknowledging that the chip has entered the Debug mode. This signal, when asserted as an input, causes the DSP56300 core to finish executing the current instruction, save the instruction pipeline information, enter Debug mode, and wait for commands to be entered from the debug serial input line. This signal is asserted as an output for three clock cycles when the chip enters Debug mode as a result of a debug request or as a result of meeting a breakpoint condition. The $\overline{\text{DE}}$ has an internal pull-up resistor.</p> <p>This is not a standard part of the JTAG Test Access Port (TAP) Controller. The signal connects directly to the OnCE module to initiate Debug mode directly or to provide a direct external indication that the chip has entered Debug mode. All other interaction with the OnCE module must occur through the JTAG port.</p>

7.1 JTAG Test Access Port

The DSP56300 core provides a dedicated user-accessible Test Access Port (TAP) based on the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1)*. Problems of testing high density circuit boards led to development of this standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The DSP56300 core implementation supports circuit-board test strategies based on this standard.

7.1.1 Boundary Scan Architecture Overview

The test logic includes a TAP consisting of four dedicated signal pins, a 16-state controller, and three test data registers. A Boundary Scan Register (BSR) links all device signal pins into a single shift register. The test logic, implemented with static logic design,

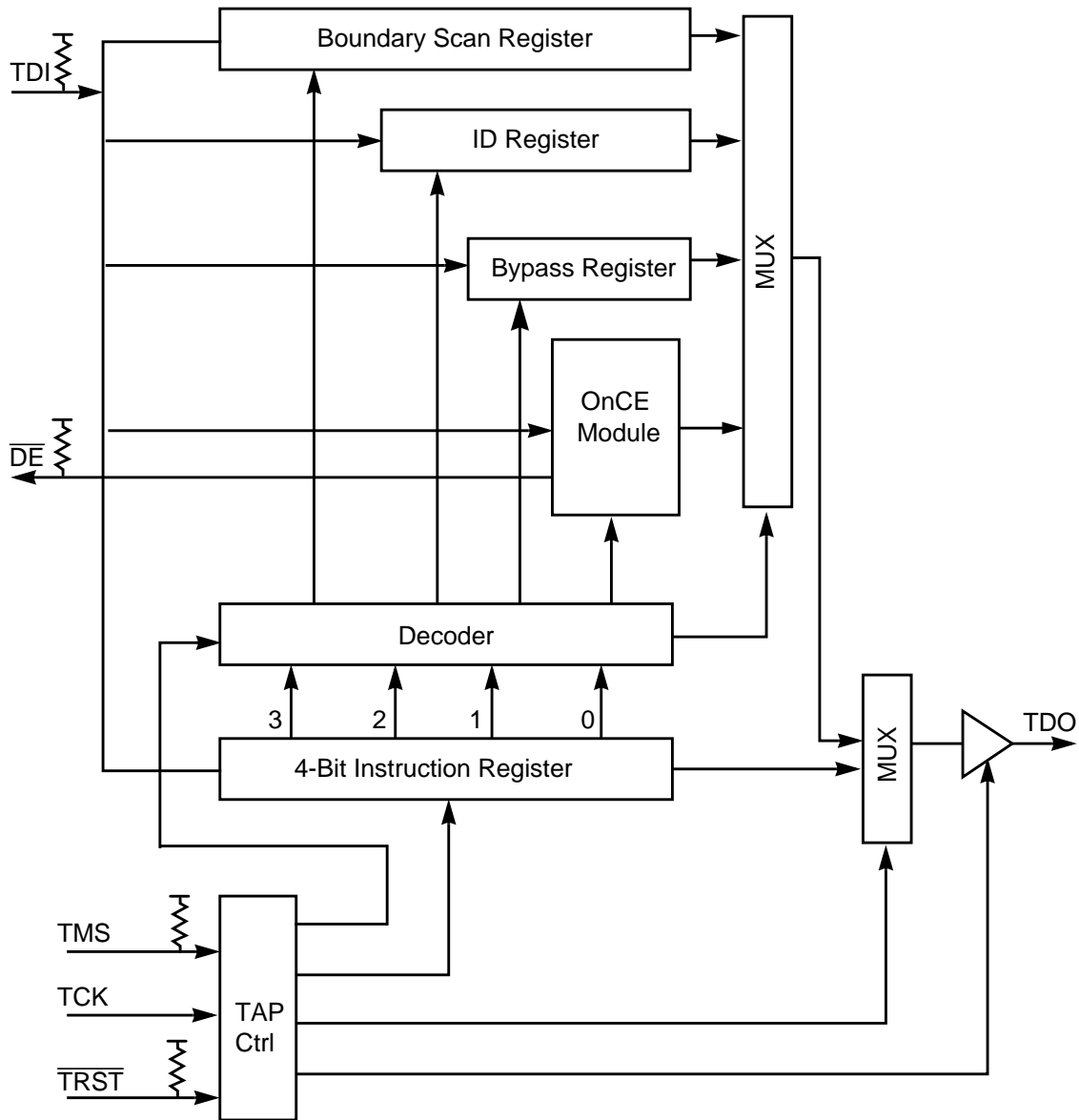
is independent of the device system logic. The DSP56300 core has the following capabilities initiated by the associated JTAG commands (listed in parentheses):

- Perform boundary scan operations to test circuit-board electrical continuity (EXTEST)
- Bypass the DSP56300 core for a given circuit board test by effectively reducing the BSR to a single cell (BYPASS)
- Sample the DSP56300 core-based device system pins during operation and transparently shift out the result in the BSR; preload values to output pins prior to invoking the EXTEST instruction (SAMPLE/PRELOAD)
- Disable the output drive to pins during circuit-board testing (HIGHZ)
- Access the OnCE controller and circuits to control a target system (ENABLE_ONCE)
- Enter the Debug mode of operation (DEBUG_REQUEST)
- Query identification information on manufacturer, part number, and version from a DSP56300 core-based device (IDCODE)
- Force test data onto the outputs of a DSP56300 core-based device while replacing its BSR in the serial data path with a single-bit register (CLAMP)

This section discusses aspects of the JTAG implementation that are specific to the DSP56300 core and is to be used with the supporting IEEE 1149.1 standards document. The discussion covers items the standard requires to be defined and includes additional information specific to the DSP56300 core implementation. **Figure 7-1** shows the block diagram of the DSP56300 core implementation of JTAG, which includes a 4-bit Instruction Register and three test registers: a 1-bit Bypass Register, a 32-bit Identification Register, and a Boundary Scan Register (BSR) whose size is chip-specific. This implementation includes a dedicated TAP and five pins.

7.1.2 TAP Controller

The TAP controller interprets the sequence of logical values on the TMS signal. It is a synchronous state machine that controls the operation of the JTAG logic. **Figure 7-2** shows the state machine. The value shown adjacent to each change-of-state arrow represents the value of the TMS signal sampled on the rising edge of the TCK signal. For a description of the TAP controller states, see the IEEE 1149.1 specification.



Note: All shown pull-up resistors are internal.

Figure 7-1. Test Access Port with OnCE Module Block Diagram

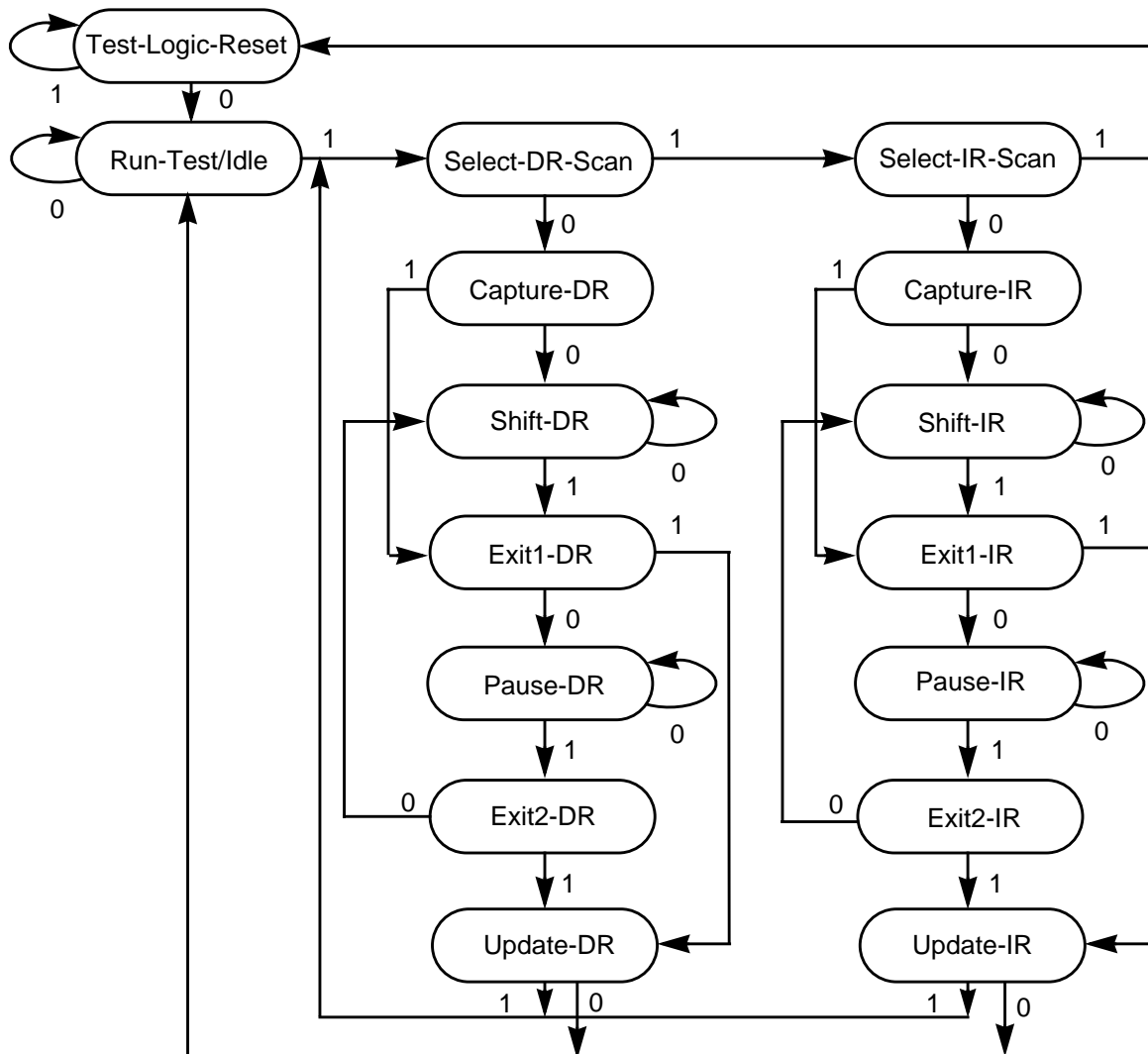


Figure 7-2. TAP Controller State Machine

7.1.3 Boundary Scan Register

The Boundary Scan Register (BSR) in the DSP56300 core JTAG implementation contains bits for all device signal and clock pins and associated control signals. All bidirectional pins are controlled by an associated control bit in the BSR. The boundary scan bit definitions vary according to specific chip implementations. See the device-specific user's manual for a complete description of the BSR contents.

7.1.4 Instruction Register

The DSP56300 core JTAG implementation includes the three mandatory public instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS) and supports the optional CLAMP instruction defined by IEEE 1149.1. The HI-Z public instruction can disable all

device output drivers. The ENABLE_ONCE public instruction enables the JTAG port to communicate with the OnCE circuitry. The DEBUG_REQUEST public instruction enables the JTAG port to force the DSP56300 core into Debug mode. The DSP56300 core includes a 4-bit instruction register without parity consisting of a shift register with four parallel outputs. Data is transferred from the shift register to the parallel outputs during the Update-IR controller state. **Figure 7-3** shows the Instruction Register configuration.

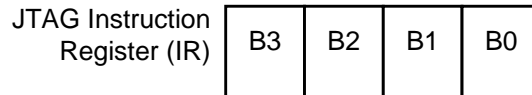


Figure 7-3. JTAG Instruction Register Format

The four bits decode the eight instructions shown in **Table 7-2**. The 0101 code is reserved for future enhancements. All other encodings (1000 – 1110) are decoded as BYPASS.

Table 7-2. JTAG Instructions

Code				Instruction
B3	B2	B1	B0	
0	0	0	0	EXTEST
0	0	0	1	SAMPLE/PRELOAD
0	0	1	0	IDCODE
0	0	1	1	RESERVED
0	1	0	0	HI-Z
0	1	0	1	CLAMP
0	1	1	0	ENABLE_ONCE ¹
0	1	1	1	DEBUG_REQUEST ¹
1	1	1	1	BYPASS
Notes: 1. The ENABLE_ONCE and DEBUG_REQUEST public instructions are not part of the IEEE 1149.1 standard. 2. x = either 1 or 0				

The parallel output of the instruction register is reset to 0010 in the Test-Logic-Reset controller state, which is equivalent to the IDCODE instruction. During the Capture-IR controller state, the parallel inputs to the instruction shift register are loaded with 01 in the Least Significant Bits (LSBs) as required by the standard. The Two Most Significant Bits (MSBs) are loaded with the values of the core status bits OS1 and OS0 from the OnCE controller.

7.1.4.1 EXTEST (B[3 – 0] = 0000)

The external test (EXTEST) instruction selects the BSR. EXTEST also asserts internal reset for the DSP56300 core system logic to force a predictable internal state while performing external boundary scan operations. Using the TAP, the BSR can:

- Scan user-defined values into the output buffers
- Capture values presented to input pins
- Control the direction of bidirectional pins
- Control the output drive of tri-stateable output pins

For details on the function and use of EXTEST, refer to the IEEE 1149.1 standards document.

7.1.4.2 SAMPLE/PRELOAD (B[3 – 0] = 0001)

The SAMPLE/PRELOAD instruction performs two separate functions. First, it obtains a snapshot of system data and control signals that occurs on the rising edge of TCK in the Capture-DR controller state. The data is observed by shifting it transparently through the BSR.

Note: Since no internal synchronization exists between the JTAG clock (TCK) and the system clock (CLK), you must provide some form of external synchronization to achieve meaningful results.

Secondly, SAMPLE/PRELOAD can initialize the BSR output cells prior to selection of EXTEST. This initialization ensures that known data appears on the outputs when the EXTEST instruction starts executing.

7.1.4.3 IDCODE (B[3 – 0] = 0010)

The IDCODE instruction selects the ID register. This public instruction allows identification of the manufacturer, part number, and version of a component through the TAP. **Figure 7-3** shows the ID register configuration.

31	28 27	22 21	17 16	12 11	1 0
Version Number	Manufacturer's Use	Sequence Number		Manufacturer Identity	IEEE 1149.1 Requirement
n n n n	Design Center Number 0 0 0 1 1 0	Core Number 0 0 0 0 0	Chip Derivative Number n n n n n	0 0 0 0 0 0 0 1 1 1 0	1

Figure 7-3. Identification Register Configuration

One application of the ID register is to distinguish the manufacturer(s) of components on a board when multiple sourcing is used. As more components that conform to the IEEE 1149.1 standard emerge, it is desirable for a system diagnostic controller unit to blindly interrogate a board design in order to determine the type of each component in each location. This information is also available for factory process monitoring and for failure mode analysis of assembled boards.

Version Number	<p>The major revision or mask set change of the device (for example, 0000 = Revision 0; 0001 = Revision A). This information is in the boundary-scan description language (BSDL) file for the device. The BSDL file for each device in the DSP56300 family is available for download from Motorola's World Wide Web site at:</p> <p>http://www.mot.com/pub/SPS/DSP/LIBRARY/</p> <p>Note that there are no revision changes for individual masks of a chip. Revision changes apply to groupings of masks (that is, mask sets). For example, for the DSP56301, a mask set of 0F92R and 1F92R has the revision number of \$1. A different mask set consisting of 0F48S, 1F48S, and 3F48S comprises Revision \$2.</p>
Manufacturer's Use	<p>The Motorola Design Center Number (bits 27 – 22). The Motorola Semiconductor Israel Ltd (MSIL) Design Center Number is 000110.</p>
Sequence Number	<p>Divided into two parts: Core Number (bits 21:17) and Chip Derivative Number (bits 16 – 12). the DSP56300 core number is 00000.</p>
Manufacturer Identity	<p>Motorola's Manufacturer Identity is 00000001110.</p>

Once the IDCODE instruction is decoded, it selects the ID register, which is a 32-bit data register. The Bypass register loads a logic 0 at the start of a scan cycle, whereas the ID register loads a logic 1 into its LSB. Examination of the first bit of data shifted out of a component during a test data scan sequence immediately following exit from Test-Logic-Reset controller state shows whether such a register is included in the design. When the IDCODE instruction is selected, the operation of the test logic has no effect on the operation of the on-chip system logic as required by the IEEE 1149.1 standard.

7.1.4.4 CLAMP (B[3 – 0] = 0011)

CLAMP is an optional instruction defined by the IEEE 1149.1 standard. It selects the 1-bit Bypass register as the serial path between TDI and TDO, while allowing signals driven from the component pins to be determined from the BSR. During testing of ICs on a PCB, it may be necessary to place static guarding values on signals that control operation of logic not involved in the test. The EXTEST instruction could be used for this purpose, but since it selects the BSR, the required guarding signals would be loaded as part of the complete serial data stream shifted in, both at the start of the test and each time a new test pattern is entered. Since the CLAMP instruction allows guarding values to be applied using the BSR of the appropriate ICs while selecting their Bypass registers, it allows much faster testing than EXTEST. Data in the boundary scan cell remains unchanged until a new instruction is shifted in or the JTAG state machine is set to its reset state. The CLAMP instruction also asserts internal reset for the DSP56300 core system logic to force a predictable internal state while performing external boundary scan operations.

7.1.4.5 HI-Z (B[3 – 0] = 0100)

HI-Z is a manufacturer's optional public instruction to prevent the need to backdrive the output pins during circuit-board testing. When HI-Z is invoked, all output drivers, including the two-state drivers, are turned off (that is, high impedance). The instruction selects the Bypass register. HI-Z also asserts internal reset for the DSP56300 core system logic to force a predictable internal state while performing external boundary scan operations.

7.1.4.6 ENABLE_ONCE(B[3:0] = 0110)

ENABLE_ONCE is not included in the IEEE 1149.1 standard. It is a public instruction that enables you to perform system debug functions. When ENABLE_ONCE is decoded, the TDI and TDO pins connect directly to the OnCE registers. The particular OnCE register connected between TDI and TDO at a given time is selected by the OnCE controller, depending on the OnCE instruction currently executing. All communication with the OnCE controller occurs through the Select-DR-Scan path of the JTAG TAP Controller.

7.1.4.7 DEBUG_REQUEST(B[3 – 0] = 0111)

DEBUG_REQUEST is not included in the IEEE 1149.1 standard. It is a public instruction that enables you to generate a debug request signal to the DSP56300 core. When DEBUG_REQUEST is decoded, the TDI and TDO pins connect to the instruction registers. In the Capture-IR state of the TAP, the OnCE status bits are captured in the Instruction shift register, so the external JTAG controller must continue to shift in the DEBUG_REQUEST while polling the status bits that are shifted out until the Debug mode of operation is entered (acknowledged by the combination 11 on OS[1 – 0]). After

acknowledgment of Debug mode is received, the external JTAG controller must issue the ENABLE_ONCE instruction so you can perform system debug functions.

7.1.4.8 BYPASS (B[3 – 0] = 1111)

BYPASS selects the single-bit Bypass register, as shown in **Figure 7-4**. This creates a shift-register path from TDI to the Bypass register, and finally to TDO, circumventing the BSR. This instruction enhances test efficiency when a component other than the DSP56300 core-based device becomes the device under test. When the current instruction selects the Bypass register, the shift-register stage is set to a logic 0 on the rising edge of TCK in the Capture-DR controller state. Therefore, the first bit shifted out after selection of the Bypass register is always a logic 0.

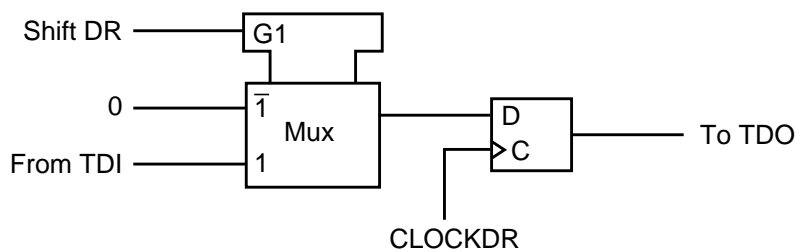


Figure 7-4. Bypass Register

7.1.5 DSP56300 JTAG Restrictions

The control afforded by the output enable signals using the BSR and the EXTEST instruction requires a compatible circuit-board test environment to avoid device-destructive configurations. You must avoid situations in which the DSP56300 core output drivers are enabled into actively driven networks. In addition, EXTEST can execute only after power-up or regular hardware reset while EXTAL is provided. While EXTEST executes, EXTAL can remain inactive.

Two constraints relate to the JTAG interface. First, the TCK input does not include an internal pull-up resistor and should not be left unconnected. The second constraint is to ensure that the JTAG test logic is kept transparent to the system logic by forcing the TAP into the Test-Logic-Reset controller state, using either of two methods. During power-up, $\overline{\text{TRST}}$ must be externally asserted to force the TAP controller into this state. After power-up finishes, TMS must be sampled as a logic 1 for five consecutive TCK rising edges. If TMS either remains unconnected or is connected to V_{CC} , then the TAP controller cannot leave the Test-Logic-Reset state, regardless of the state of TCK. The DSP56300 core features a low-power Stop mode, which is invoked using the STOP instruction. The interaction of the JTAG interface with low-power Stop mode is as follows:

1. The TAP controller must be in the Test-Logic-Reset state to either enter or remain in the low-power Stop mode. Leaving the TAP controller Test-Logic-Reset state negates the ability to achieve low power, but does not otherwise affect device functionality.
2. The TCK input is not blocked in low-power Stop mode. To consume minimal power, the TCK input should be externally pulled to V_{CC} or GND.
3. The TMS and TDI pins include on-chip pull-up resistors. In low-power Stop mode, these two pins should remain either unconnected or connected to V_{CC} to achieve minimal power consumption.

During Stop mode all DSP56300 core clocks are disabled, so the JTAG interface provides the means for polling the device status (sampled in the Capture-IR state). For a DSP56300 derivative that does not include the \overline{DE} pin, the JTAG interface provides the DEBUG_REQUEST instruction for entering Debug mode.

7.2 OnCE™ Module

The DSP56300 core On-Chip Emulation (OnCE™) module interacts with the DSP56300 core and its peripherals non-intrusively so that you can examine registers, memory, or on-chip peripherals, thus facilitating hardware and software development on the DSP56300 core processor. Special circuits and dedicated pins on the DSP56300 core are defined to avoid sacrificing any user-accessible on-chip resource.

The OnCE module controller functionality is accessed through the JTAG test access port (TAP). In addition to describing OnCE features and functionality, this section gives examples of debugging procedures using the OnCE module. The OnCE module resources can be accessed only after the JTAG ENABLE_ONCE executes instruction (these resources are accessible even when the chip operates in Normal mode). **Figure 7-5** shows the block diagram of the OnCE module.

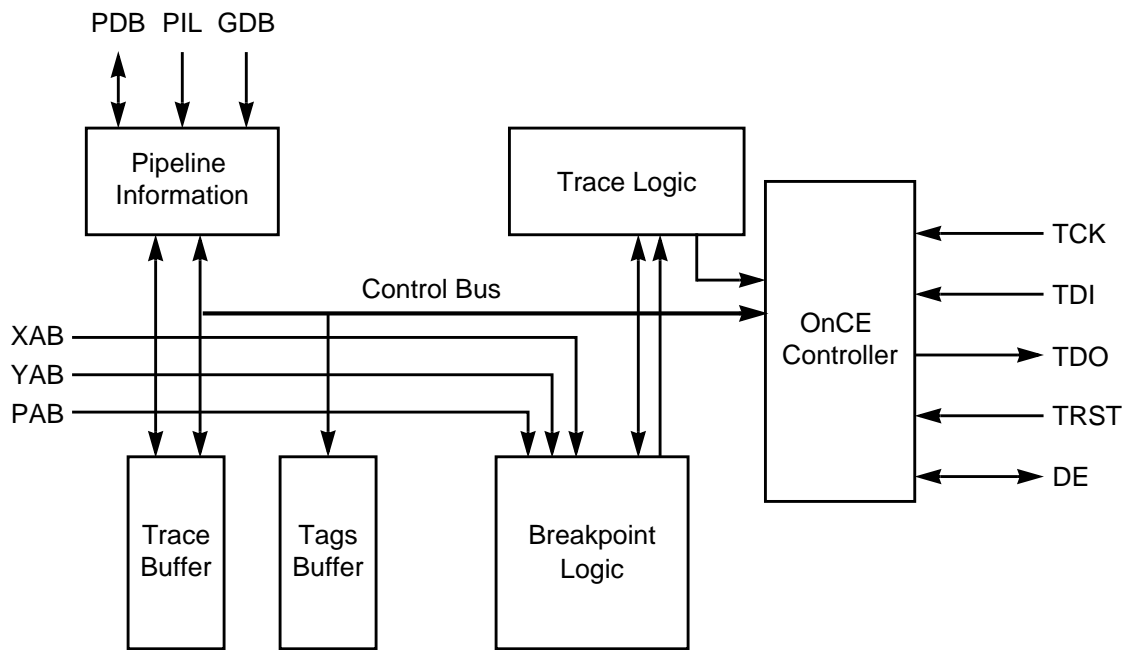


Figure 7-5. OnCE Block Diagram

The OnCE module controller functionality is accessed through the JTAG port. The JTAG TCK, TDI, and TDO pins shift data and instructions in and out.

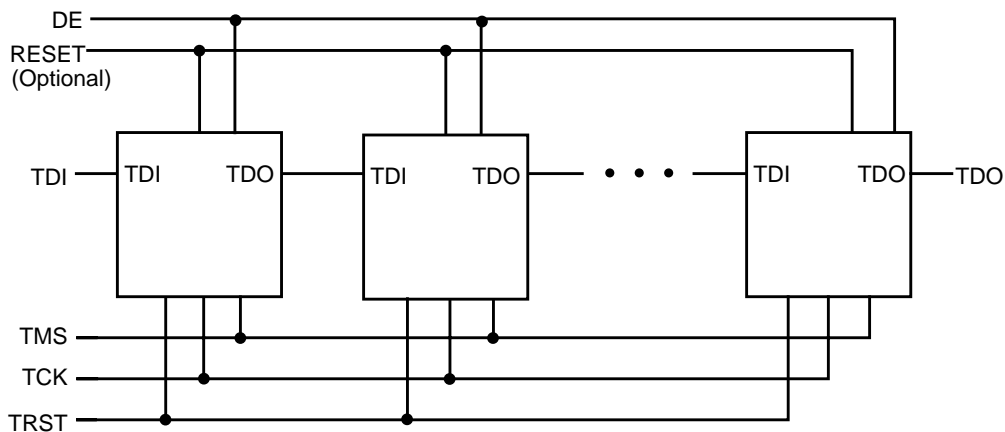


Figure 7-6. OnCE Multiprocessor Configuration

7.2.1 OnCE Controller

The OnCE Controller contains the following blocks: OnCE Command Register (OCR), OnCE Decoder, and the OnCE Status and Control Register (OSCR). **Figure 7-7** shows a block diagram of the OnCE controller.

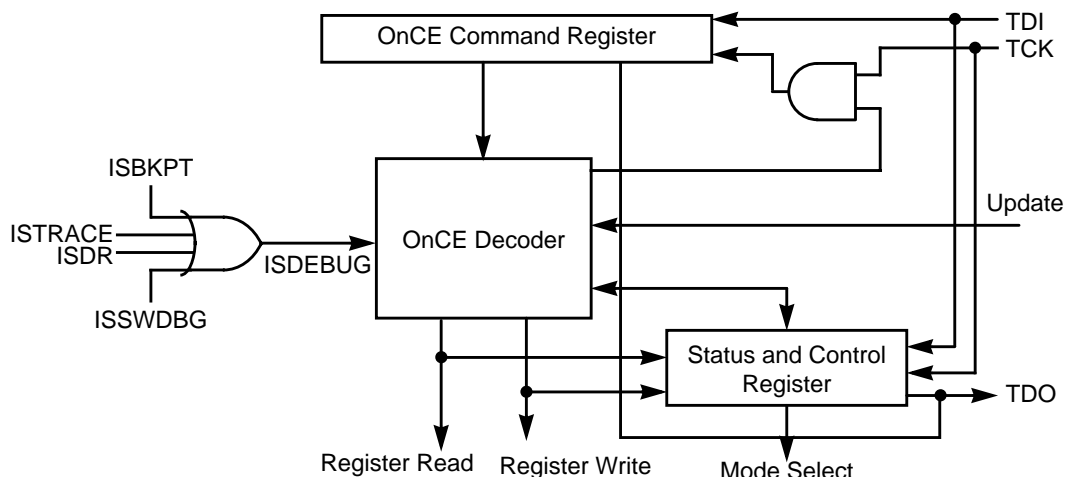


Figure 7-7. OnCE Controller

7.2.1.1 OnCE Command Register (OCR)

The OnCE Command Register (OCR) is a shift register that receives its serial data from the TDI pin. It holds the 8-bit commands to be used as input for the OnCE Decoder. The OCR is shown in **Figure 7-8**.



Figure 7-8. OnCE Command Register (OCR) Format

Table 7-3. OnCE Command Register (OCR) Bit Definitions

Bit Number	Bit Name	Reset Value	Description		
7	R/W		Read/Write Command Specifies the direction of the data transfer.		
				R/W	Action
			0	Write the data associated with the command into the register specified by RS[4 – 0].	
			1	Read the data contained in the register specified by RS[4 – 0].	

Table 7-3. OnCE Command Register (OCR) Bit Definitions (Continued)

Bit Number	Bit Name	Reset Value	Description
6	GO		<p>Go Command</p> <p>If the GO bit is set, executes the instruction that resides in the OnCE PIL register. To execute the instruction, the core leaves Debug mode. The core returns to the Debug mode immediately after executing the instruction if the EX bit is cleared. The core continues normal operation if the EX bit is set. The GO command executes only if the operation is a write to the OnCE Program Data Bus Register (OPDBR) or a read/write to No Register Selected. Otherwise, the GO bit is ignored.</p>
5	EX		<p>Exit Command</p> <p>If the EX bit is set, the core exits Debug mode and resumes normal operation. The EXIT command executes only if the GO command is issued, and the operation writes to OPDBR or reads/writes to No Register Selected. Otherwise, the EX bit is ignored.</p>

Table 7-3. OnCE Command Register (OCR) Bit Definitions (Continued)

Bit Number	Bit Name	Reset Value	Description	
4 – 0	RS		Register Select Defines which register is the source/destination for the read/write operation. Following is the OnCe Register Select Encoding:	
			RS[4 – 0]	Register Selected
			00000	OnCE Status and Control Register (OSCR)
			00001	OnCE Memory Breakpoint Counter (OMBC)
			00010	OnCE Breakpoint Control Register (OBCR)
			00011	Reserved
			00100	Reserved
			00101	OnCE Memory Limit Register 0 (OMLR0)
			00110	OnCE Memory Limit Register 1 (OMLR1)
			00111	Reserved
			01000	Reserved
			01001	OnCE GDB Register (OGDBR)
			01010	OnCE PDB Register (OPDBR)
			01011	OnCE PIL Register (OPILR)
			01100	PDB GO-TO Register (for GO TO command)
			01101	OnCE Trace Counter (OTC)
			01110	Reserved
			01111	OnCE PAB Register for Fetch (OPABFR)
			10000	OnCE PAB Register for Decode (OPABDR)
			10001	OnCE PAB Register for Execute (OPABEX)
			10010	Trace Buffer and Increment Pointer
			10011	Reserved
			101xx	Reserved
			11xx0	Reserved
			11x0x	Reserved
110xx	Reserved			
11111	No Register Selected			

7.2.1.2 OnCE Decoder (ODEC)

The OnCE Decoder (ODEC) supervises the entire OnCE module activity. It receives as input the 8-bit command from the OCR, a signal from the JTAG Controller (indicating that 8/24 bits have been received and that the selected data register must be updated), and a signal indicating that the core halted. The ODEC generates all the strobes required for reading and writing the selected OnCE registers.

7.2.1.3 OnCE Status and Control Register (OSCR)

The OnCE Status and Control Register (OSCR) enables the Trace mode of operation and indicates the reason for entering Debug mode. The control bits are read/write, and the status bits are read-only. The OSCR bits are cleared by hardware reset. The OSCR is shown in **Figure 7-9**.

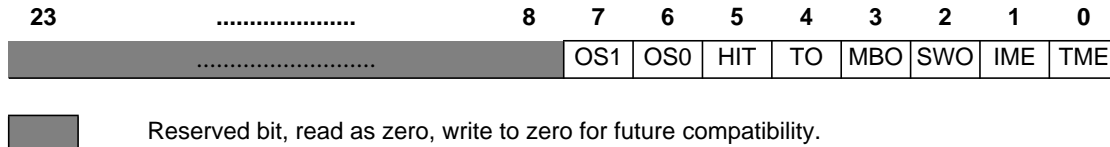


Figure 7-9. OnCE Status and Control Register (OSCR)

Table 7-4. OnCE Status and Control Register (OSCR) Bit Definitions

Bit Number	Bit Name	Reset Value	Description
23 – 0		0	Reserved. Write to zero for future compatibility.
7 – 6	OS	0	Core Status Read-only status bits that provide core status information. Examining the status bits, you can determine whether the chip has entered Debug mode. To find the reason for entering Debug mode, consult the OSCR SWO, MBO, and TO bits. You can also examine these bits to determine why the chip has not entered the Debug mode after debug event assertion (\overline{DE}) or execution of the JTAG Debug Request instruction (core waiting for the bus, STOP or WAIT instruction, etc.). The OS bits are also reflected in the JTAG instruction shift register, which allows the polling of the core status information at the JTAG level so that you can read the OSCR after the DSP56300 core executes the STOP instruction (and therefore there are no clocks).
			OS1 OS0 Description
			0 0 DSP56300 core is executing instructions
			0 1 DSP56300 core is in Wait or Stop mode
			1 0 DSP56300 core is waiting for bus
1 1 DSP56300 core is in Debug mode			
5	HIT	0	Cache Hit A read-only status bit that is set when a cache hit occurs in Cache mode in the Debug mode of operation. In PRAM mode, this bit reads as one.

Table 7-4. OnCE Status and Control Register (OSCR) Bit Definitions (Continued)

Bit Number	Bit Name	Reset Value	Description
4	TO	0	<p>Trace Occurrence The Trace Occurrence (TO) bit is a read-only status bit that is set when all the following occur:</p> <ul style="list-style-type: none"> ■ Trace Counter = 0 ■ Trace mode is enabled ■ Debug mode of operation is entered <p>This bit is cleared when the DSP leaves Debug mode.</p>
3	MBO	0	<p>Memory Breakpoint Occurrence A read-only status bit that is set when the DSP enters Debug mode because a memory breakpoint has been encountered. This bit is cleared when the DSP leaves Debug mode.</p>
2	SWO	0	<p>Software Debug Occurrence A read-only status bit that is set when the DSP enters Debug mode because of the execution of the DEBUG or DEBUGcc instruction with condition true. This bit is cleared when the DSP leaves Debug mode.</p>
1	IME	0	<p>Interrupt Mode Enable When this control bit is set, the chip executes a vectored interrupt to the address VBA:\$06 instead of entering Debug mode.</p>
0	TME	0	<p>Trace Mode Enable When set, this control bit enables the Trace mode of operation.</p>

7.2.2 OnCE Memory Breakpoint Logic

Memory breakpoints can be set on program memory or data memory locations. In addition, the breakpoint does not have to be in a specific memory address, but within an approximate address range of where the program may be executing. This significantly increases your ability to monitor what the program is doing in real-time. The breakpoint logic, shown in **Figure 7-10.**, contains a latch for the addresses, registers that store the upper and lower address limit, address comparators, and a breakpoint counter. Address comparators are useful in determining where a program may be getting lost or when data is written where it should not be written. They are also useful in halting a program at a specific point to examine/change registers or memory. Using address comparators to set breakpoints enables you to set breakpoints in RAM or ROM in any operating mode. Memory accesses are monitored according to the contents of the OBCR as specified in **Section 7.2.2.6**, "OnCE Breakpoint Control Register (OBCR)," on page 7-19.

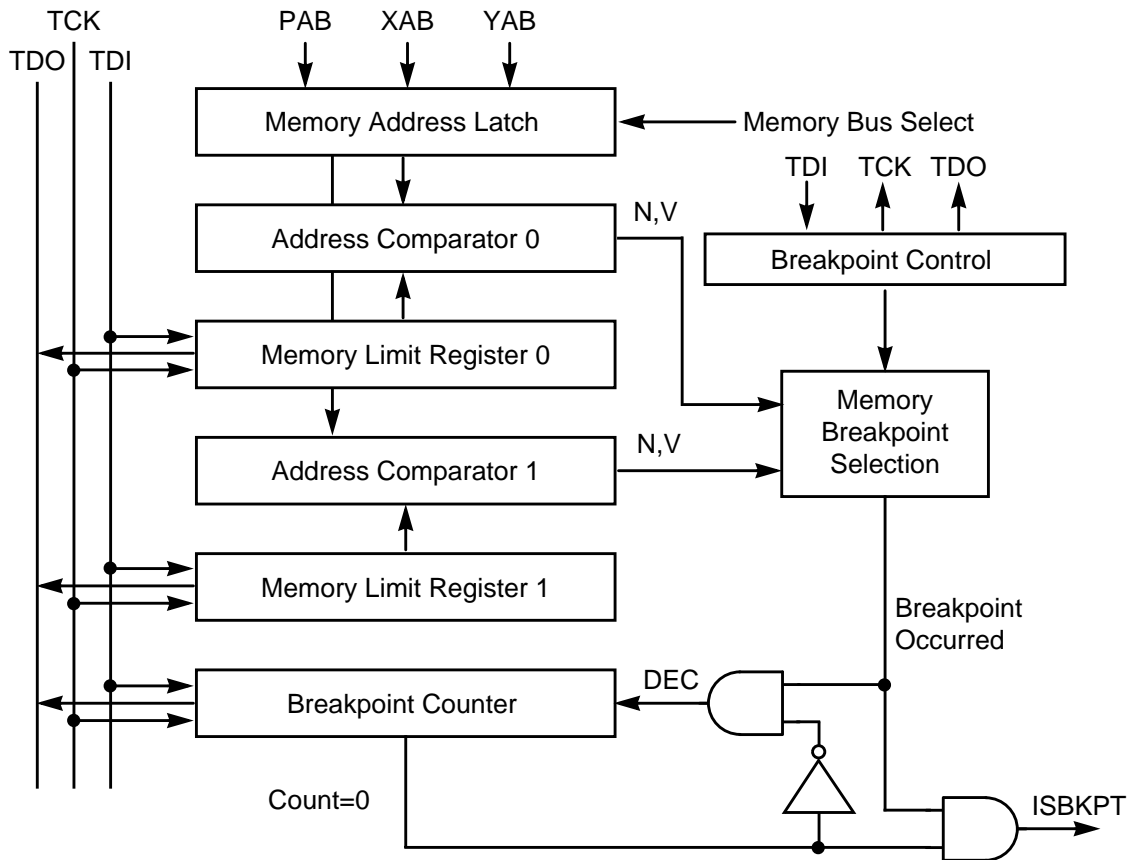


Figure 7-10. OnCE Memory Breakpoint Logic 0

7.2.2.1 OnCE Memory Address Latch (OMAL)

The OnCE Memory Address Latch (OMAL) is a 24-bit register that latches the PAB, XAB or YAB on every instruction cycle according to the MBS[1–0] bits in the OBCR.

7.2.2.2 OnCE Memory Limit Register 0 (OMLR0)

The OnCE Memory Limit Register 0 (OMLR0) is a 24-bit register that stores the memory breakpoint limit. OMLR0 can be read or written through the JTAG port. Before enabling breakpoints, OMLR0 must be loaded by the external command controller.

7.2.2.3 OnCE Memory Address Comparator 0 (OMAC0)

The OnCE Memory Address Comparator 0 (OMAC0) compares the current memory address (stored in OMAL) with the OMLR0 contents.

7.2.2.4 OnCE Memory Limit Register 1 (OMLR1)

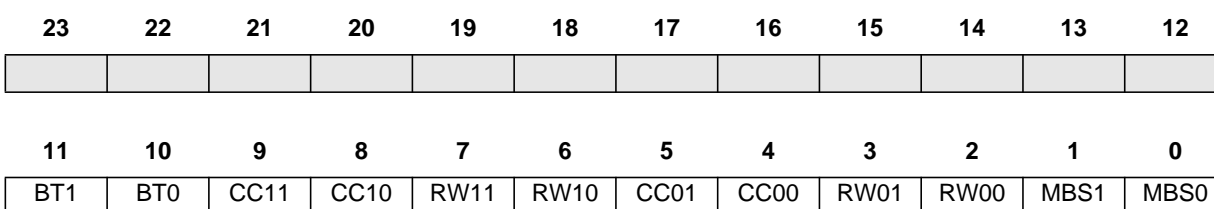
The OnCE Memory Limit Register 1 (OMLR1) is a 24-bit register that stores the memory breakpoint limit. OMLR1 can be read or written through the JTAG port. Before enabling breakpoints, OMLR1 must be loaded by the external command controller.

7.2.2.5 OnCE Memory Address Comparator 1 (OMAC1)

The OnCE Memory Address Comparator 1 (OMAC1) compares the current memory address (stored in OMAL) with the OMLR1 contents.

7.2.2.6 OnCE Breakpoint Control Register (OBCR)

The OnCE Breakpoint Control Register (OBCR) defines the memory breakpoint events. The OBCR can be read or written through the JTAG port. All OBCR bits are cleared on hardware reset.



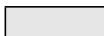
 Reserved, read as zero, should be written with zero for future compatibility.

Figure 7-11. OnCE Breakpoint Control Register (OBCR)

Table 7-5. OnCE Breakpoint Control Register (OBCR) Bit Definitions

Bit Number	Bit Name	Reset Value	Description	
23 – 12		0	Reserved. Write to zero for future compatibility.	
11 – 10	BT	0	Breakpoint Event Bits Define the sequence between breakpoints 0 and 1. If the condition defined by BT[1 – 0] is met, then the Breakpoint Counter (OMBC) is decremented.	
			BT[1 – 0]	Description
			00	Breakpoint 0 and Breakpoint 1
			01	Breakpoint 0 or Breakpoint 1
			10	Breakpoint 1 after Breakpoint 0
			11	Breakpoint 0 after Breakpoint 1

Table 7-5. OnCE Breakpoint Control Register (OBCR) Bit Definitions (Continued)

Bit Number	Bit Name	Reset Value	Description	
9 – 8	CC1	0	Breakpoint1 Condition Code Define the condition of the comparison between the current memory address (OMAL) and the OnCE Memory Limit Register 1 (OMLR1).	
			CC1[1 – 0]	Description
			00	Breakpoint on not equal
			01	Breakpoint on equal
			10	Breakpoint on less than
7 – 6	RW1	0	Breakpoint 1 Read/Write Define memory breakpoint 1 to occur when a memory address access is performed for read, write or both.	
			RW1[1 – 0]	Description
			00	Breakpoint disabled
			01	Breakpoint on write access
			10	Breakpoint on read access
4 – 5	CC0	0	Breakpoint 0 Condition Code Define the condition of the comparison between the current Memory Address (OMAL) and the Memory Limit Register 0 (OMLR0).	
			CC0[1 – 0]	Description
			00	Breakpoint on not equal
			01	Breakpoint on equal
			10	Breakpoint on less than
3 – 2	RW0	0	Breakpoint 0 Read/Write Define the memory breakpoint 0 to occur when a memory address access is performed for read, write, or both.	
			RW0[1 – 0]	Description
			00	Breakpoint disabled
			01	Breakpoint on write access
			10	Breakpoint on read access
			11	Breakpoint on read or write access

Table 7-5. OnCE Breakpoint Control Register (OBCR) Bit Definitions (Continued)

Bit Number	Bit Name	Reset Value	Description	
1 – 0	MBS	0	Memory Breakpoint Enable memory breakpoints 0 and 1, allowing them to occur when a memory access is performed on P, X, or Y memory.	
			MBS[1 – 0]	Description
			00	Reserved
			01	Breakpoint on P access
			10	Breakpoint on X access
			11	Breakpoint on Y access

7.2.2.7 OnCE Memory Breakpoint Counter (OMBC)

The OnCE Memory Breakpoint Counter is a 24-bit counter that is loaded with a value equal to the number of times minus one that a memory access event should occur before a memory breakpoint is declared. The memory access event is specified by the OBCR and by the memory limit registers. On each occurrence of the memory access event, the breakpoint counter decrements. When the counter reaches 0 and a new event occurs, the chip enters Debug mode. The OMBC can be read or written through the JTAG port. Each time the limit register changes or a different breakpoint event is selected in the OBCR, the breakpoint counter must be written afterwards. This ensures that the OnCE breakpoint logic is reset and that no previous events can affect the new breakpoint event selected. The breakpoint counter is cleared by hardware reset.

7.2.3 Cache Support

To keep track of the cache contents and status, the eight Tag values, Tag lock/unlock status, and LRU status can be read via the OnCE module. Nine 24-bit registers are implemented as a circular buffer with a 4-bit counter. All registers have the same address, but any access to the Tag buffer increments the counter, thus pointing to the next register in the circular buffer. When Debug mode is exited, the counter is cleared, so when Debug mode is re-entered, the first read from the Tag buffer address always starts from the first register of the nine (Tag number 0) and circles continuously among these nine registers. The register mapping in the circular Tag buffer is shown in **Figure 7-12**, "Circular Tags Buffer (TAGB)," on page 7-22.

At any time, at least one LRU bit in the LRU/Lock Status Register is set, but multiple LRU bits can be set at the same time because locked sectors can be the Least Recently Used sector even though they cannot be replaced. Therefore, the next sector to be replaced is the only sector whose LRU bit is set and whose lock bit is cleared. The one exception to this rule occurs when all eight sectors are locked and LRU, in which case there is no next

sector to be replaced, because no sector can be replaced until at least one sector is unlocked.

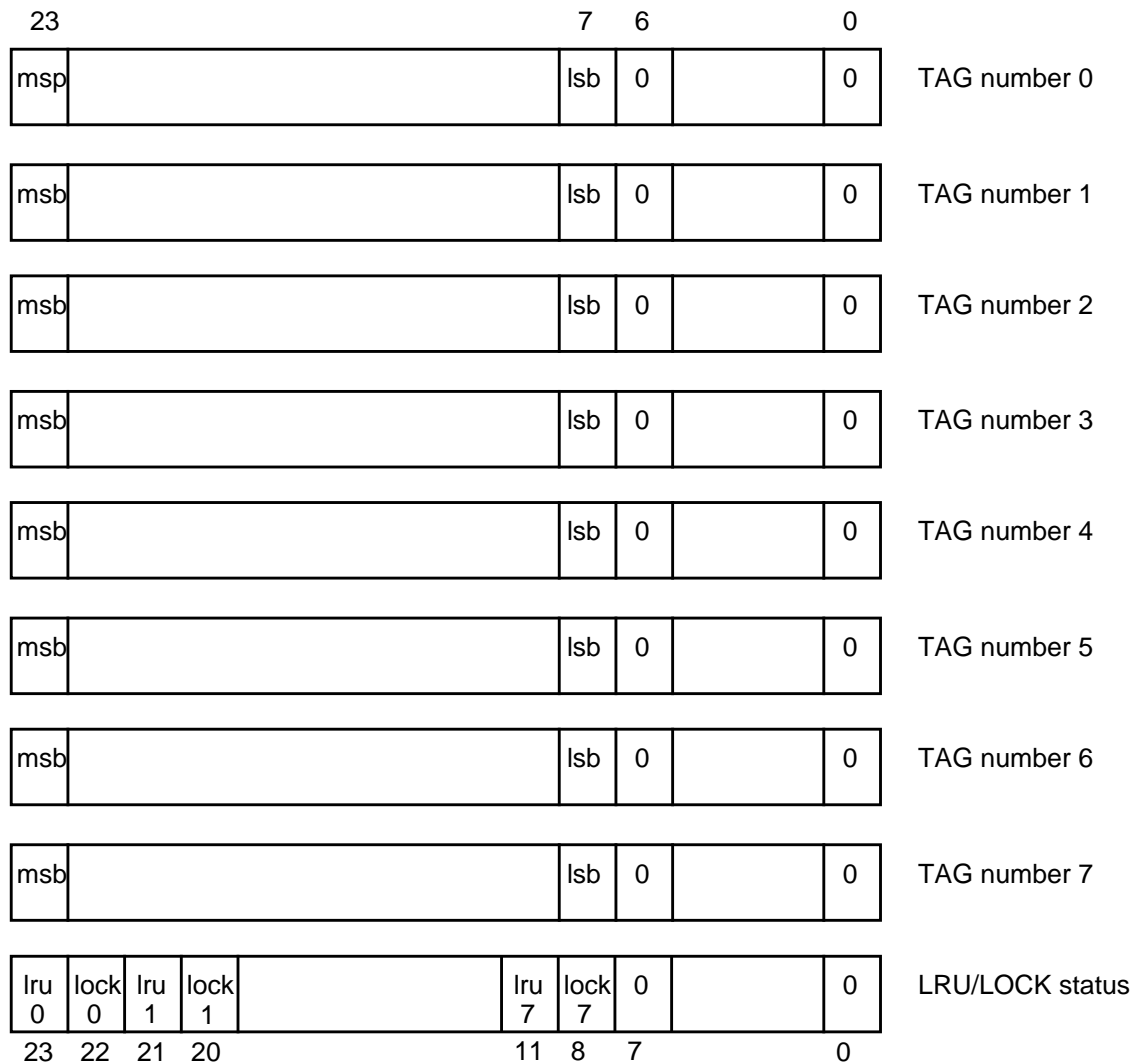


Figure 7-12. Circular Tags Buffer (TAGB)

7.2.3.1 OnCE Trace Logic

The 24-bit OnCE Trace Counter (OTC) can be read or written through the JTAG port. If N instructions are to be executed before Debug mode is entered, the Trace Counter should be loaded with N – 1. The Trace Counter is cleared by hardware reset. When the OnCE Trace Logic is used, instructions can execute in single or multiple steps. The OnCE Trace Logic causes the chip to enter Debug mode after one or more instructions execute and to wait for OnCE commands from the debug serial port. The OnCE Trace Logic block diagram is shown in **Figure 7-13**.

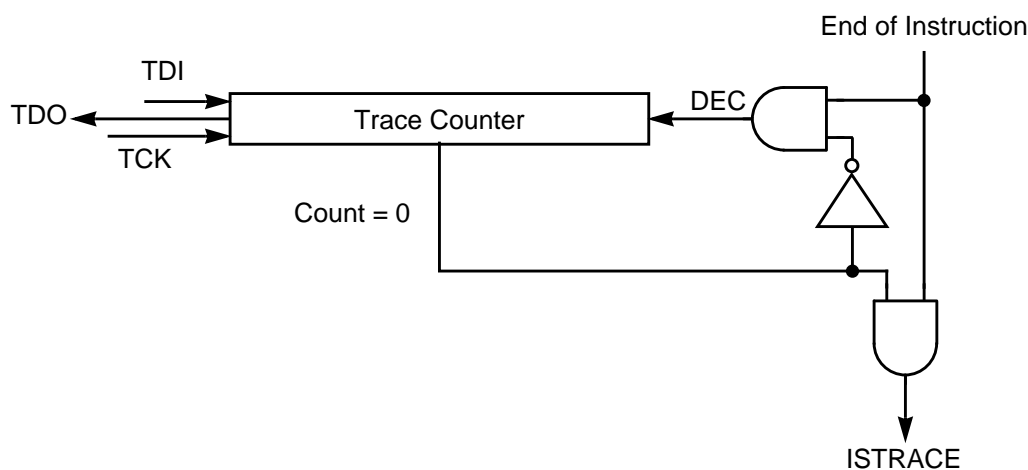


Figure 7-13. OnCE Trace Logic Block Diagram

Trace mode has an associated counter so that more than one instruction can be executed before returning to Debug mode. The counter allows you to take multiple real-time instruction steps before entering Debug mode. This feature helps you to debug sections of code that do not have a normal flow or are hanging up in infinite loops. The Trace Counter also enables you to count the number of instructions executed in a code segment.

To enable Trace mode, the counter is loaded with a value, the program counter is set to the start location of the instruction(s) to be executed real-time, the TME bit is set in the OSCR and the DSP56300 core exits Debug mode by executing the appropriate command issued by the external command controller.

When Debug mode is exited, the counter decrements after each execution of an instruction. Interrupts are serviceable and all instructions executed—including fast interrupt services and repeated instructions—decrement the Trace Counter. When it decrements to 0, the DSP56300 core re-enters Debug mode, the Trace Occurrence bit (TO) in the OSCR is set, the core Status bits OS[1 – 0] are set to 11, and the \overline{DE} pin (if provided) is asserted to indicate that the DSP56300 core has entered Debug mode and is requesting service.

7.2.4 Methods of Entering Debug Mode

The chip acknowledges entering Debug mode by setting the Core Status bits OS1 and OS0 and asserting the \overline{DE} line. This informs the external command controller that the chip is in Debug mode and awaiting commands. The DSP56300 core can disable the OnCE module if the ROM Security option is implemented. If the ROM Security is implemented, the OnCE module remains inactive until the DSP56300 core executes a write operation to the OGDBR.

Following is a list of ways to enter Debug mode:

- **External Debug Request During $\overline{\text{RESET}}$ Assertion:** Holding the $\overline{\text{DE}}$ line asserted during the assertion of $\overline{\text{RESET}}$ causes the chip to enter the Debug mode. After receiving the acknowledge, the external command controller must negate the $\overline{\text{DE}}$ line before sending the first command. In this case, the chip does not execute any instruction before entering the Debug mode.
- **External Debug Request During Normal Activity:** Holding the $\overline{\text{DE}}$ line asserted during normal chip activity causes the chip to finish executing the current instruction and then enter Debug mode. After receiving the acknowledge, the external command controller must negate the $\overline{\text{DE}}$ line before sending the first command. This process is the same for any newly fetched instruction, including instructions fetched by the interrupt processing or instructions that are aborted by the interrupt processing. In this case the chip finishes executing the current instruction and stops after the newly fetched instruction enters the instruction latch.
- **Executing the JTAG DEBUG_REQUEST Instruction:** Executing the JTAG instruction DEBUG_REQUEST asserts an internal debug request signal. The chip finishes executing the current instruction and stops after the newly fetched instruction enters the instruction latch. After entering the Debug mode, the Core Status bits OS1 and OS0 are set and the $\overline{\text{DE}}$ line is asserted, thus acknowledging the external command controller that the Debug mode of operation has been entered.
- **External Debug Request During Stop:** Executing the JTAG instruction DEBUG_REQUEST (or asserting $\overline{\text{DE}}$) while the chip is in Stop state (i. e., has executed a STOP instruction) causes the chip to exit the Stop state and enter Debug mode. After receiving the acknowledge, the external command controller must negate $\overline{\text{DE}}$ before sending the first command. In this case, the chip finishes executing the STOP instruction and halts after the next instruction enters the instruction latch.
- **External Debug Request During Wait:** Executing the JTAG instruction DEBUG_REQUEST (or asserting $\overline{\text{DE}}$) while the chip is in the Wait state (i. e., has executed a WAIT instruction) causes the chip to exit the Wait state and enter Debug mode. After receiving the acknowledge, the external command controller must negate $\overline{\text{DE}}$ before sending the first command. In this case, the chip completes the execution of the WAIT instruction and halts after the next instruction enters the instruction latch.
- **Software Request During Normal Activity:** Upon executing the DSP56300 core instruction DEBUG (or DEBUGcc when the specified condition is true), the chip enters Debug mode after the instruction following the DEBUG instruction enters the instruction latch.

- **Enabling Trace Mode:** When the Trace mode mechanism is enabled and the Trace Counter is greater than 0, the Trace Counter decrements after each instruction executes. Execution of an instruction when the Trace Counter = 0 causes the chip to enter the Debug mode after completing the execution of the instruction. Only instructions actually executed cause the Trace Counter to decrement. An aborted instruction does not decrement the Trace Counter and does not cause the chip to enter Debug mode.
- **Enabling Memory Breakpoints:** When the memory breakpoint mechanism is enabled with a Breakpoint Counter value of 0, the chip enters Debug mode after executing the instruction that caused the memory breakpoint to occur. For breakpoints on executed Program memory fetches, the breakpoint is acknowledged immediately after the fetched instruction executes. For breakpoints on accesses to X, Y or P memory spaces by MOVE instructions, the breakpoint is acknowledged after execution of the instruction following the instruction that accessed the specified address.

To restore the pipeline and to resume normal chip activity upon returning from the Debug mode, a number of on-chip registers store the chip pipeline status. **Figure 7-14** shows the block diagram of the Pipeline Information Registers with the exception of the PAB registers, which are shown in **Figure 7-15**, "OnCE Trace Buffer Block Diagram," on page 7-28.

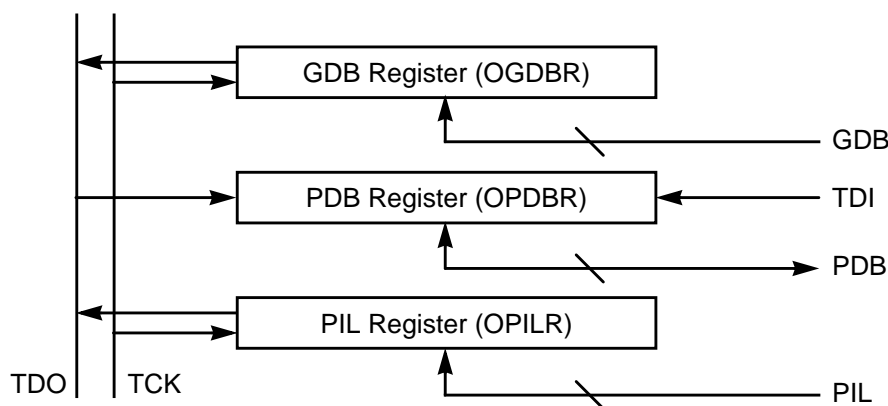


Figure 7-14. OnCE Pipeline Information and GDB Registers

- **OnCE PDB Register (OPDBR):** A 24-bit latch that stores the value of the Program Data Bus generated by the last program memory access of the core before Debug mode is entered. The OPDBR is read or written through the JTAG port. This register is affected by the operations performed during the Debug mode and must be restored by the external command controller when returning to Normal mode.

- **OnCE PIL Register (OPILR):** A 24-bit latch that stores the value of the Instruction Latch before Debug mode is entered. OPILR can only be read through the JTAG port. Since the Instruction Latch is affected by the operations performed during Debug mode, it must be restored by the external command controller when returning to Normal mode. Since there is no direct write access to the Instruction Latch, restoration is accomplished by writing to the OPDBR with no-GO and no-EX. The data written on PDB is transferred into the Instruction Latch.
- **OnCE GDB Register (OGDBR):** A 24-bit latch that can only be read through the JTAG port. The OGDBR is not actually required for a pipeline status restore, but is required for passing information between the chip and the external command controller. The OGDBR is mapped on the X internal I/O space at address \$FFFFFC. When the external command controller needs the contents of a register or memory location, it forces the chip to execute an instruction that brings this information to the OGDBR. Then the contents of the OGDBR are delivered serially to the external command controller by the command READ GDB REGISTER.

7.2.5 Trace Buffer

To ease debugging activity and keep track of program flow, the DSP56300 core provides a number of on-chip dedicated resources. Three read-only PAB registers give pipeline information when Debug mode is entered, and a Trace Buffer stores the address of the last instruction executed, as well as the addresses of the last eight change of flow instructions.

- **OnCE PAB Register for Fetch (OPABFR):** A 24-bit register that stores the address of the last instruction whose fetch started before Debug mode was entered. The OPABFR can only be read through the JTAG port. This register is not affected by the operations performed during Debug mode.
- **PAB Register for Decode (OPABDR):** A 24-bit register that stores the address of the instruction currently on the PDB. This is the instruction whose fetch completed before the chip entered Debug mode. The OPABDR can only be read through the JTAG port. This register is not affected by the operations performed during the Debug mode.
- **PAB Register for Execute (OPABEX):** A 24-bit register that stores the address of the instruction currently in the Instruction Latch. This is the instruction that would have decoded and executed if the chip had not entered Debug mode. The OPABEX register can only be read through the JTAG port. This register is not affected by the operations performed during Debug mode.

The Trace Buffer stores the addresses of the last twelve change of flow instructions that executed, as well as the address of the last executed instruction. It is implemented as a circular buffer containing twelve 25-bit registers and one 4-bit counter. All the registers

have the same address, but any read access to the Trace Buffer address causes the counter to increment, thus pointing to the next Trace Buffer register. The registers are serially available to the external command controller through their common Trace Buffer address. **Figure 7-15** shows the block diagram of the Trace Buffer. The Trace Buffer is not affected by the operations performed during Debug mode except for the Trace Buffer pointer increment when reading the Trace Buffer. When Debug mode is entered, the Trace Buffer counter points to the Trace Buffer register containing the address of the last executed instructions. The first Trace Buffer read obtains the oldest address and the following Trace Buffer reads get the other addresses from the oldest to the newest, in order of execution.

Note: To ensure Trace Buffer coherence, a complete set of twelve reads of the Trace Buffer must be performed because each read increments the Trace Buffer pointer, thus pointing to the next location. After twelve reads, the pointer indicates the same location as before starting the read procedure.

Note: On any change of flow instruction, the Trace Buffer stores both the address of the change of flow instruction, as well as the address of the target of the change of flow instruction. In the case of conditional change of flows, the address of the change of flow instruction is always stored (regardless of the fact that the change of flow is true or false), but if the conditional change of flow is false (that is, not taken) the address of the target is not stored. In order to facilitate the program trace reconstruction, every Trace Buffer location has an additional invalid bit (the 25th bit). If a conditional change of flow instruction has a condition false, the invalid bit is set, thus marking this instruction as not taken. Therefore, it is imperative to read twenty-five bits of data when reading the twelve Trace Buffer registers. Since data is read LSB first, the invalid bit is the first bit to be read.

7.2.6 OnCE Commands and Serial Protocol

To permit an efficient means of communication between the external command controller and the DSP56300 core chip, the following protocol is adopted. Before starting any debugging activity, the external command controller must wait for an acknowledge on the \overline{DE} line indicating that the chip has entered Debug mode (optionally the external command controller can poll the OS1 and OS0 bits in the JTAG instruction shift register). The external command controller communicates with the chip by sending 8-bit commands that can be accompanied by 24 bits of data. Both commands and data are sent or received Least Significant Bit first. After sending a command, the external command controller should wait for the DSP56300 core chip to acknowledge execution of the command. The external command controller can send a new command only after the chip acknowledges execution of the previous command.

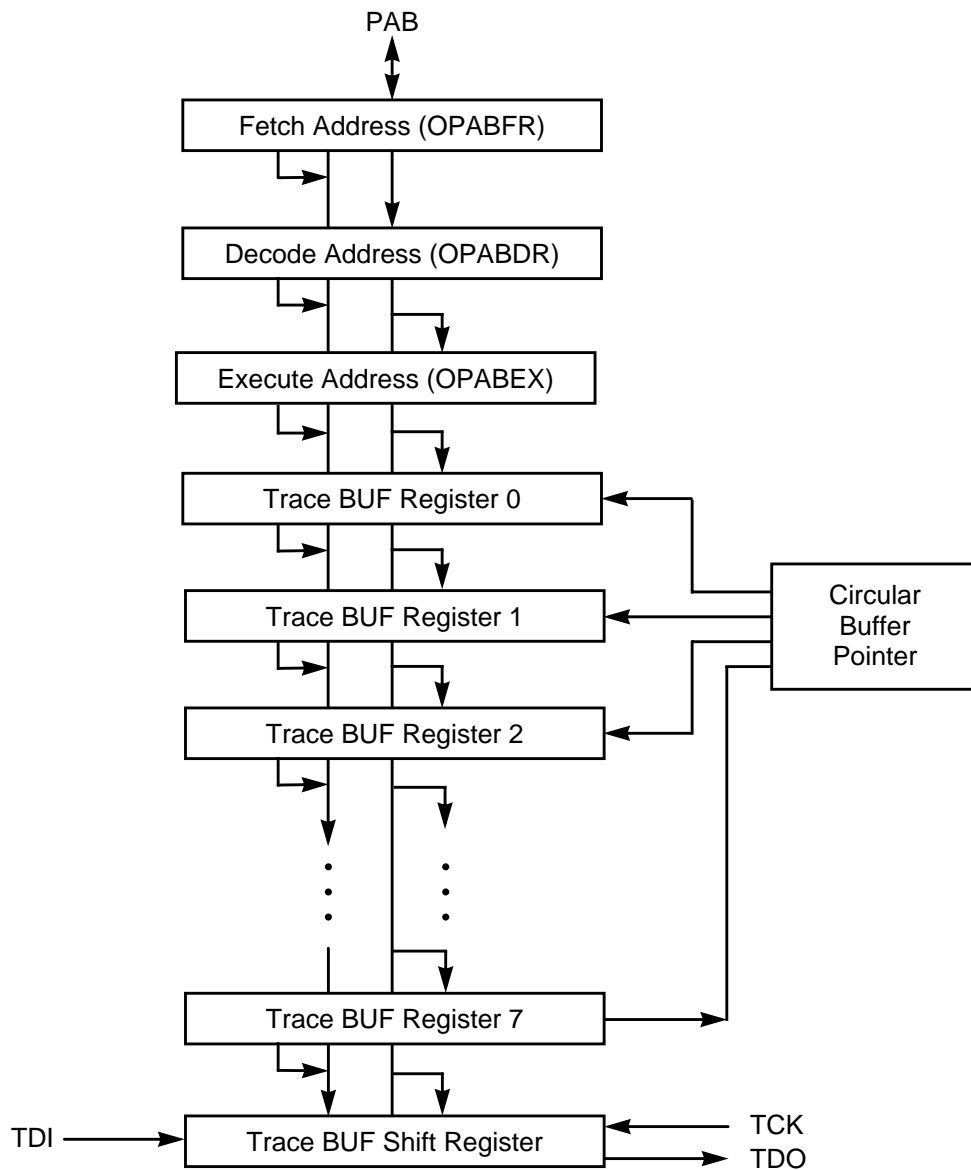


Figure 7-15. OnCE Trace Buffer Block Diagram

The OnCE commands are classified as follows:

- Read commands (when the chip delivers the required data)
- Write commands (when the chip receives data and writes the data in one of the OnCE registers)
- Commands that do not have data transfers associated with them

The commands are 8 bits long and have the format shown in **Figure 7-8**, "OnCE Command Register (OCR) Format," on page 7-13.

7.2.7 OnCE Module Examples

The following examples of debugging procedures using the OnCE module assume that the DSP is the only device in the JTAG chain. If more than one device in the chain exists (other DSPs or even other devices), the other devices can be forced to execute the JTAG BYPASS instruction so that their effect in the serial stream is one bit per additional device. The events select-DR, select-IR, update-DR, shift-DR etc. refer to bringing the JTAG TAP in the corresponding state.

7.2.7.1 Checking Whether the Chip Has Entered Debug Mode

There are two methods of verifying that the chip has entered Debug mode:

- Every time the chip enters Debug mode, a pulse is generated on the \overline{DE} line. A pulse is also generated every time the chip acknowledges the execution of an instruction in Debug mode. An external command controller can connect the \overline{DE} line to an interrupt pin to sense the acknowledge.
- An external command controller can poll the JTAG instruction shift register for the status bits OS1-OS0. When the chip is in Debug mode these bits are set to the value 11.

In the following paragraphs, the ACK notation denotes the operation performed by the command controller to check whether the chip has entered Debug mode (either by sensing \overline{DE} or by polling JTAG instruction shift register).

7.2.7.2 Polling the JTAG Instruction Register

To poll the core status bits in the JTAG Instruction Register, the following sequence must be performed:

1. Select shift-IR. Passing through capture-IR loads the core status bits into the instruction shift register.
2. Shift in ENABLE_ONCE. While shifting-in the new instruction the captured status information is shifted out. Pass through update-IR.
3. Return to Run-Test/Idle.

The external command controller can analyze the information shifted out and detect whether the chip has entered Debug mode.

7.2.7.3 Saving Pipeline Information

The debugging activity is accomplished by DSP56300 core instructions supplied from the external command controller. Therefore the current state of the DSP56300 core pipeline

must be saved before the debug activity starts and the state must be restored before returning to the Normal Mode of operation. The following description of the saving procedure assumes that ENABLE_ONCE has executed and Debug mode has been entered and verified as described in **Section 7.2.7.1**, "Checking Whether the Chip Has Entered Debug Mode," on page 7-29:

1. Select shift-DR. Shift in the Read PDB. Pass through update-DR.
2. Select shift-DR. Shift out the 24-bit OPDB register. Pass through update-DR.
3. Select shift-DR. Shift in the Read PIL. Pass through update-DR.
4. Select shift-DR. Shift out the 24-bit OPILR register. Pass through update-DR.

You do not need to verify acknowledge between Steps 1 and 2 or between Steps 3 and 4, because completion is guaranteed by design.

7.2.7.4 Reading the Trace Buffer

An optional step during debugging activity is reading the information associated with the Trace Buffer in order to enable an external program to reconstruct the full trace of the executed program. In the following description of the read Trace Buffer procedure, assume that all actions described in **Section 7.2.7.3** have executed:

1. Select shift-DR. Shift in the Read PABFR. Pass through update-DR.
2. Select shift-DR. Shift out the 24-bit OPABFR register. Pass through update-DR.
3. Select shift-DR. Shift in the Read PABDR. Pass through update-DR.
4. Select shift-DR. Shift out the 24-bit OPABDR register. Pass through update-DR.
5. Select shift-DR. Shift in the Read PABEX. Pass through update-DR.
6. Select shift-DR. Shift out the 24-bit OPABEX register. Pass through update-DR.
7. Select shift-DR. Shift in the Read FIFO. Pass through update-DR.
8. Select shift-DR. Shift out the 25 bit FIFO register. Pass through update-DR.
9. Repeat Steps 7 and 8 for the entire FIFO (12 times).

You must read the entire FIFO since each read increments the FIFO pointer thus pointing to the next FIFO location. At the end of this procedure the FIFO pointer points back to the beginning of the FIFO. The information read by the external command controller contains the address of the newly fetched instruction, the address of the instruction currently on the PDB, the address of the instruction currently on the instruction latch, and the addresses of the last twelve instructions that have been executed. A user program can now reconstruct the flow of a full trace based on this information and on the original source code of the currently running program.

7.2.7.5 Displaying a Specified Register

The DSP56300 must be in Debug mode and all actions described in **Section 7.2.7.3** must have been executed:

1. Select shift-DR. Shift in the Write PDB with GO no-EX. Pass through update-DR.
2. Select shift-DR. Shift in the 24-bit opcode: MOVE reg, X:OGDB. Pass through update-DR to actually write OPDBR and thus begin executing the MOVE instruction.
3. Wait for DSP to reenter Debug mode (wait for \overline{DE} or poll core status).
4. Select shift-DR and shift in READ GDB REGISTER. Pass through update-DR (this selects OGDBR as the data register for read).
5. Select shift-DR. Shift out the OGDBR contents. Pass through update-DR. Wait for next command.

7.2.7.6 Displaying X Memory Area Starting at Address \$xxxxxx

The DSP56300 must be in Debug mode and all actions described in **Section 7.2.7.3** must have been executed. Since R0 is used as pointer for the memory, R0 is saved first. The sequence of actions is:

1. Select shift-DR. Shift in the Write PDB with GO no-EX. Pass through update-DR.
2. Select shift-DR. Shift in the 24-bit opcode: MOVE R0, X:OGDB. Pass through update-DR to actually write OPDBR and thus begin executing the MOVE instruction.
3. Wait for DSP to reenter Debug mode (wait for \overline{DE} or poll core status).
4. Select shift-DR and shift in READ GDB REGISTER. Pass through update-DR (this selects OGDBR as the data register for read).
5. Select shift-DR. Shift out the OGDBR contents. Pass through update-DR. R0 is now saved.
6. Select shift-DR. Shift in the Write PDB with no-GO no-EX. Pass through update-DR.
7. Select shift-DR. Shift in the 24-bit opcode: MOVE #\$xxxxxx,R0. Pass through update-DR to actually write OPDBR.
8. Select shift-DR. Shift in the Write PDB with GO no-EX. Pass through update-DR.
9. Select shift-DR. Shift in the second word of the 24-bit opcode: MOVE #\$xxxxxx,R0 (the \$xxxxxx field). Pass through update-DR to actually write

OPDBR and execute the instruction. R0 is loaded with the base address of the memory block to be read.

10. Wait for DSP to reenter Debug mode (wait for \overline{DE} or poll core status).
11. Select shift-DR. Shift in the Write PDB with GO no-EX. Pass through update-DR.
12. Select shift-DR. Shift in the 24-bit opcode: MOVE X:(R0)+, X:OGDB. Pass through update-DR to actually write OPDBR and thus begin executing the MOVE instruction.
13. Wait for DSP to reenter Debug mode (wait for \overline{DE} or poll core status).
14. Select shift-DR and shift in READ GDB REGISTER. Pass through update-DR (this selects OGDBR as the data register for read).
15. Select shift-DR. Shift out the OGDBR contents. Pass through update-DR. The memory contents of address \$xxxxxx has been read.
16. Select shift-DR. Shift in the NO SELECT with GO no-EX. Pass through update-DR. This re-executes the same MOVE X:(R0)+, X:OGDB instruction.
17. Repeat from Step 14 to complete the reading of the entire block. When finished, restore the original value of R0.

7.2.7.7 Returning From Debug Mode to Normal Mode to Current Program

When you have finished examining the current state of the machine, changed some of the registers, and wish to return and continue execution of its program from the point where it stopped, you must restore the machine pipeline and enable normal instruction execution, as follows:

1. Select shift-DR. Shift in the Write PDB with no-GO no-EX. Pass through update-DR.
2. Select shift-DR. Shift in the 24 bits of saved PIL (instruction latch value). Pass through update-DR to actually write the Instruction Latch.
3. Select shift-DR. Shift in the Write PDB with GO and EX. Pass through update-DR.
4. Select shift-DR. Shift in the 24 bits of saved PDB. Pass through update-DR to actually write the PDB. At the same time the internally saved value of the PAB is driven back from the PABFR register onto the PAB, the ODEC releases the chip from Debug mode and the normal flow of execution is continued.

7.2.7.8 Returning from Debug Mode to Normal Mode to a New Program

When you have finished examining the current state of the machine, changed some of the registers and wish to start the execution of a new program (the GOTO command), you

must force a change-of-flow to the starting address of the new program (\$xxxxxx), as follows:

1. Select shift-DR. Shift in the Write PDB with no-GO no-EX. Pass through update-DR.
2. Select shift-DR. Shift in the 24 bits of \$0AF080 which is the opcode of the JUMP instruction. Pass through update-DR to actually write the Instruction Latch.
3. Select shift-DR. Shift in the Write PDB-GO-TO with GO and EX. Pass through update-DR.
4. Select shift-DR. Shift in the 24 bits of \$xxxxxx. Pass through update-DR to actually write the PDB. At this time the ODEC releases the chip from Debug mode and the execution is started from the address \$xxxxxx.

If Debug mode entry occurred during a DO LOOP, REP instruction, or other special case (that is, interrupt processing, STOP, WAIT, conditional branching, etc.), you *must* reset the DSP56300 before executing the new program.

7.3 Examples of JTAG-OnCE Interaction

This section presents the details of the JTAG-OnCE interaction by describing the TMS sequencing required to achieve the communication described in **Section 7.2.7**. The external command controller can force the DSP56300 into Debug mode by executing the JTAG DEBUG_REQUEST instruction. To verify that the DSP56300 has entered Debug mode, the external command controller must poll the status by reading the OS[1 – 0] bits in the JTAG Instruction Shift Register. The TMS sequencing is listed in **Table 7-6**. The sequencing for enabling the OnCE module is described in **Table 7-7**. After executing the JTAG instructions DEBUG_REQUEST and ENABLE_ONCE and after the core status is polled to verify that the chip is in Debug mode, the pipeline saving procedure must occur. The TMS sequencing for this procedure is listed in **Table 7-8**.

Table 7-6. TMS Sequencing for DEBUG_REQUEST and Poll the Status

Step	TMS	JTAG	OnCE™	Note
a	0	Run-Test/Idle	Idle	
b	1	Select-DR-Scan	Idle	
c	1	Select-IR-Scan	Idle	
d	0	Capture-IR	Idle	status is sampled in shifter

Table 7-6. TMS Sequencing for DEBUG_REQUEST and Poll the Status (Continued)

Step	TMS	JTAG	OnCE™	Note
e	0	Shift-IR	Idle	the 4 bits of the JTAG DEBUG_REQUEST (0111) are shifted in while status is shifted out
			
e	0	Shift-IR	Idle	
f	1	Exit1-IR	Idle	
g	1	Update-IR	Idle	debug req is generated
h	1	Select-DR-Scan	Idle	
i	1	Select-IR-Scan	Idle	
j	0	Capture-IR	Idle	status is sampled in shifter
k	0	Shift-IR	Idle	the 4 bits of the JTAG DEBUG_REQUEST (0111) are shifted in while status is shifted out
			
k	0	Shift-IR	Idle	
l	1	Exit1-IR	Idle	
m	1	Update-IR	Idle	
n	0	Run-Test/Idle	Idle	This step is repeated enabling an external command controller to poll the status
			
n	0	Run-Test/Idle	Idle	

In Step n the external command controller verifies that OS[1 – 0] = 11, indicating that the chip has entered the Debug mode. If the chip has not yet entered the Debug mode, the external command controller goes to Step b, Step c, and so forth, until the Debug mode is acknowledged.

Table 7-7. TMS Sequencing for ENABLE_ONCE

Step	TMS	JTAG	OnCE™	Note
a	1	Test-Logic-Reset	Idle	
b	0	Run-Test/Idle	Idle	
c	1	Select-DR-Scan	Idle	
d	1	Select-IR-Scan	Idle	
e	0	Capture-IR	Idle	Capture core status bits

Table 7-7. TMS Sequencing for ENABLE_ONCE (Continued)

Step	TMS	JTAG	OnCE™	Note
f	0	Shift-IR	Idle	the 4 bits of the JTAG ENABLE_ONCE instruction (0110) are shifted into the JTAG instruction register while status is shifted out
g	0	Shift-IR	Idle	
h	0	Shift-IR	Idle	
i	0	Shift-IR	Idle	
j	1	Exit1-IR	Idle	
k	1	Update-IR	Idle	OnCE is enabled
l	0	Run-Test/Idle	Idle	This step can be repeated enabling an external command controller to poll the status
.....				
l	0	Run-Test/Idle	Idle	

Table 7-8. TMS Sequencing for Reading Pipeline Register

Step	TMS	JTAG	OnCE™	Note
a	0	Run-Test/Idle	Idle	
b	1	Select-DR-Scan	Idle	
c	0	Capture-DR	Idle	
d	0	Shift-DR	Idle	the 8 bits of the OnCE "Read PIL" (10001011) are shifted in
.....				
d	0	Shift-DR	Idle	
e	1	Exit1-DR	Idle	
f	1	Update-DR	Execute "Read PIL"	PIL value is loaded in shifter
g	1	Select-DR-Scan	Idle	
h	0	Capture-DR	Idle	
i	0	Shift-DR	Idle	the 24 bits of the PIL are shifted out (24 steps)
.....				
i	0	Shift-DR	Idle	
j	1	Exit1-DR	Idle	
k	1	Update-DR	Idle	
l	1	Select-DR-Scan	Idle	

Table 7-8. TMS Sequencing for Reading Pipeline Register (Continued)

Step	TMS	JTAG	OnCE™	Note
m	0	Capture-DR	Idle	
n	0	Shift-DR	Idle	the 8 bit of the OnCE "Read PDB" (10001010)are shifted in
.....				
n	0	Shift-DR	Idle	
o	1	Exit1-DR	Idle	
p	1	Update-DR	Execute "Read PDB"	PDB value is loaded in shifter
q	1	Select-DR-Scan	Idle	
r	0	Capture-DR	Idle	
s	0	Shift-DR	Idle	The 24 bits of the PDB are shifted out (24 steps)
.....				
s	0	Shift-DR	Idle	
t	1	Exit1-DR	Idle	
u	1	Update-DR	Idle	
v	0	Run-Test/Idle	Idle	This step can be repeated enabling an external command controller to analyze the information.
.....				
v	0	Run-Test/Idle	Idle	

During Step v, the external command controller stores the pipeline information and afterwards it can proceed with the debug activities, as requested by the user.

7.3.1 Address Trace Mode

Address Trace mode allows you to determine the address of internal accesses. The mode is disabled after reset and enabled by setting the ATE bit in the Operating Mode Register (OMR). When the mode is enabled and there is no simultaneous external access, the internal access is reflected on the external address lines. Use the status of \overline{BR} to determine whether the access referenced by A[0 – 23]/A[0 – 17] is internal or external, when this mode is enabled. \overline{BR} is deasserted for internal accesses and asserted for external accesses.