

# Programming the DSP56300 OnCE and JTAG Ports

by Barbara Johnson

## 1 Introduction

This application note describes the DSP56300 OnCE and JTAG ports and explains how they interact. A series of examples demonstrates how to use the OnCE and JTAG ports to enter the Debug mode, save pipeline registers, read from the trace buffer, write to memory, and read from memory. For more information on the OnCE and JTAG ports, see the applicable sections of the *DSP56300 Family Manual*.

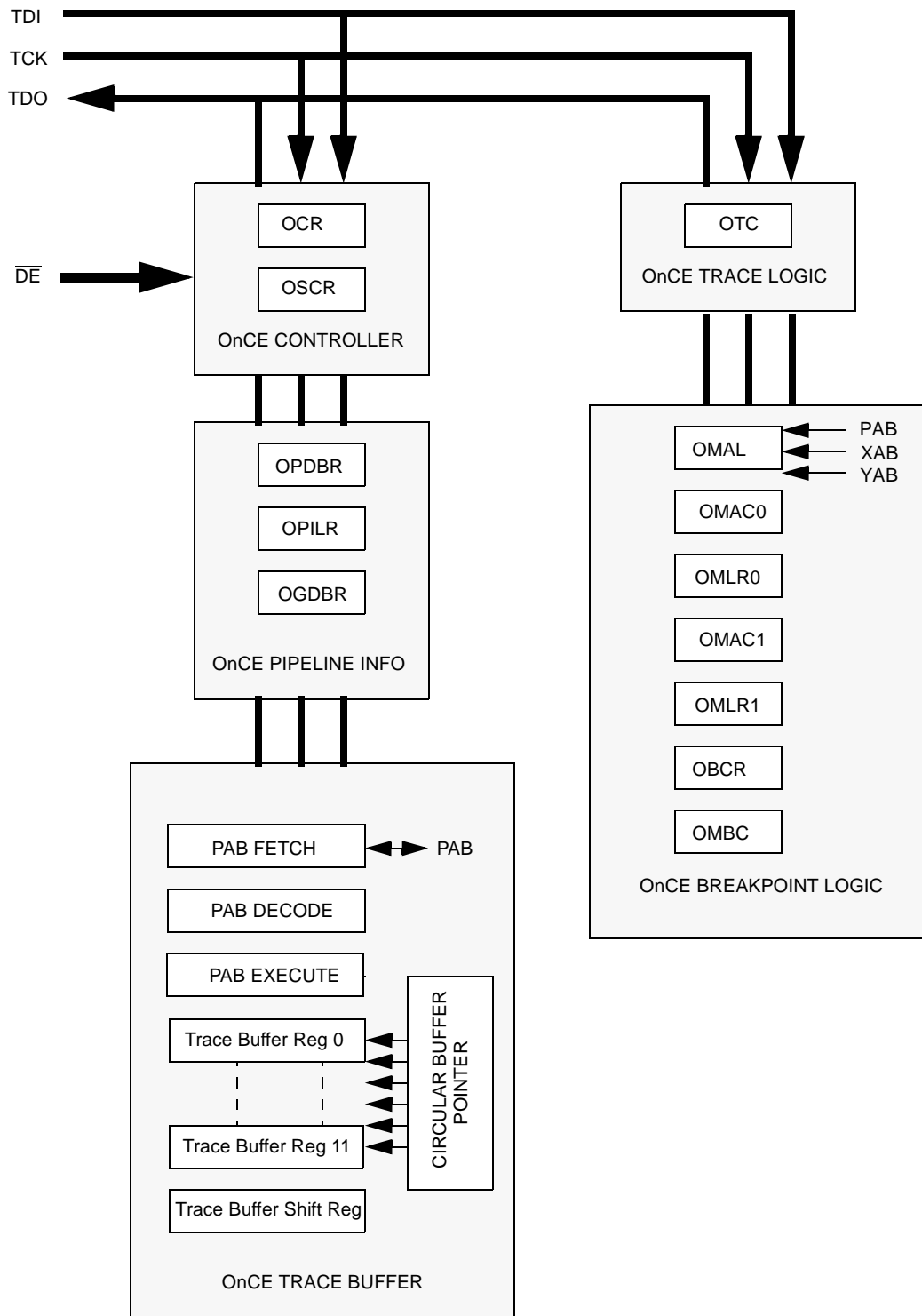
## 2 OnCE Module

The On-Chip Emulation (OnCE) module allows you to examine registers, memories, and on-chip peripherals, which facilitates hardware and software development on the DSP56300 core processor. The OnCE module interacts with the core and its peripherals non-intrusively and is accessed through the JTAG port (see *JTAG Port* on page 7). **Figure 2-1, OnCE Port Architecture**, on page 2 shows the OnCE port architecture.

## Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
<b>2</b>	<b>OnCE Module .....</b>	<b>1</b>
2.1	OnCE Controller .....	3
2.1.1	OnCE Command Register (OCR) ...	3
2.1.2	OnCE Status and Control Register (OSCR) .....	4
2.2	OnCE Memory Breakpoint Logic ...	5
2.3	OnCE Trace Logic .....	6
2.4	Pipeline Information and the OGDB Register .....	6
2.5	OnCE Trace Buffer .....	7
<b>3</b>	<b>JTAG Port .....</b>	<b>7</b>
3.1	TAP Controller .....	8
3.2	Signals .....	10
3.3	JTAG Instruction Register .....	10
3.4	JTAG ID Register .....	11
3.5	JTAG Boundary Scan Register .....	12
<b>4</b>	<b>Serial Protocol .....</b>	<b>12</b>
<b>5</b>	<b>Examples of OnCE/JTAG Interaction .....</b>	<b>13</b>
5.1	Entering the Test-Logic-Reset State .....	13
5.2	Executing the JTAG DEBUG_REQUEST Instruction .....	15
5.3	Executing the JTAG ENABLE_ONCE Instruction .....	15
5.4	Reading the Pipeline Registers .....	16
5.5	Reading the Trace Buffer .....	17
5.6	Writing to Memory .....	20
5.7	Reading from Memory .....	22





**Figure 2-1.** OnCE Port Architecture

## 2.1 OnCE Controller

To communicate with the OnCE controller, the DSP must be in Debug mode. To enter Debug mode from a hardware or software breakpoint, either single step through opcodes or assert the  $\overline{DE}$  pin.

A state machine decodes 8-bit commands specified in the OnCE Command Register (OCR) and controls the interaction with the OnCE registers. The OCR receives serial data from the TDI signal.

The OnCE decoder (ODEC) receives the 8-bit command from the OCR, a signal from the JTAG controller to indicate that it has received 8 or 24 bits and that an update of the selected data register must be performed, and a signal indicating that the core has halted.

The 24-bit OnCE Status and Control Register (OSC) enables trace mode and indicates the reason for entering Debug mode.

### 2.1.1 OnCE Command Register (OCR)

**Table 2-1** shows the OCR and **Table 2-2** shows the selected OCR registers.

**Table 2-1.** OnCE Command Register (OCR) Bits

Bit Name	Description	
7 Read/Write Command	R/W = 0	Writes the data associated with the command into the register specified by RS[4–0].
	R/W = 1	Reads the data in the register specified by RS[4–0].
6 Go Command	GO = 0	Inactive. No action taken.
	GO = 1	Executes the instruction in the OPILR register. See <i>Pipeline Information and the OGDB Register</i> on page 6.
5 Exit Command	EX = 0	Remains in Debug mode.
	EX = 1	Leaves Debug mode and resumes normal operation.
4-0 Register Select	RS[4–0]	Register select bits that define which register is the source or destination for read or write operations. Refer to <b>Table 2-2</b> .

**Table 2-2.** OCR Register Select Bits

RS[4–0]	Register Selected	See Also
00000	OnCE Status and Control Register (OSCR).	<i>OnCE Status and Control Register (OSCR)</i> on page 4.
00001	Memory Breakpoint Counter (OMBC).	<i>OnCE Memory Breakpoint Logic</i> on page 5.
00010	Breakpoint Control Register (OBCR).	<i>OnCE Memory Breakpoint Logic</i> on page 5
00101	Memory Limit Register 0 (OMLR0).	<i>OnCE Memory Breakpoint Logic</i> on page 5
00110	Memory Limit Register 1 (OMLR1).	<i>OnCE Memory Breakpoint Logic</i> on page 5
01001	GDB Register (OGDBR).	<i>Pipeline Information and the OGDB Register</i> on page 6.
01010	PDB Register (OPDBR).	<i>Pipeline Information and the OGDB Register</i> on page 6.

## OnCE Controller

**Table 2-2.** OCR Register Select Bits (Continued)

RS[4–0]	Register Selected	See Also
01011	PIL Register (OPILR).	<i>Pipeline Information and the OGDB Register on page 6.</i>
01100	PDB GO-TO Register (for GO TO command)	N/A
01101	Trace Counter (OTC).	<i>See OnCE Trace Logic on page 6.</i>
01110	Tags Buffer (TAGB).	N/A
01111	PAB Register for Fetch (OPABFR).	<i>OnCE Trace Buffer on page 7.</i>
10000	PAB Register for Decode (OPABDR).	<i>OnCE Trace Buffer on page 7.</i>
10001	PAB Register for Execute (OPABEX).	<i>OnCE Trace Buffer on page 7.</i>
10010	Trace Buffer and Increment Pointer.	<i>OnCE Trace Buffer on page 7.</i>
11111	No Register Selected	N/A

### 2.1.2 OnCE Status and Control Register (OSCR)

Table 2-3 shows the OSCR.

**Table 2-3.** OnCE Status and Control Register (OSCR) Bits

Bit Name	Description		
23–8	Reserved		
7–6 Core Status	OS1 = 0	OS0 = 0	DSP56300 core is executing instructions
	OS1 = 0	OS0 = 1	DSP56300 core is in wait or stop
	OS1 = 1	OS0 = 0	DSP56300 core is waiting for the bus
	OS1 = 1	OS0 = 1	DSP56300 core is in Debug mode
5 Cache Hit	HIT = 0		Cache hit has not occurred.
	HIT = 1		Cache hit has occurred in Cache mode and in Debug mode.
4 Trace Occurrence	TO = 0		DSP56300 core leaves Debug mode.
	TO = 1		Debug mode is entered because trace counter is 0 while trace mode is enabled.
3 Memory Breakpoint Occurrence	MBO = 0		DSP56300 core leaves Debug mode.
	MBO = 1		Debug mode is entered because a memory breakpoint has been encountered.
2 Software Debug Occurrence	SWO = 0		DSP56300 core leaves Debug mode.
	SWO = 1		Debug mode is entered because the execution of the DEBUG or DEBUGcc instruction with condition is true.
1 Interrupt Mode Enable	IME = 0		Interrupt mode is disabled.
	IME = 1		Interrupt mode is enabled.
0 Trace Mode Enable	TME = 0		Trace mode is disabled.
	TME = 1		Trace mode is enabled.

## 2.2 OnCE Memory Breakpoint Logic

The breakpoint logic contains a latch for addresses of registers that store the upper and lower address limits, address comparators, and a breakpoint counter.

The 24-bit OnCE Breakpoint Control Register (OBCR) defines memory breakpoint events. It enables breakpoints on P/X/Y access and breakpoints on read or write access. The OBCR, shown in **Table 2-4**, can be read or written through the JTAG port.

**Table 2-4.** OnCE Breakpoint Control Register (OBCR) Bits

Bit Name	Description		
23–12	Reserved		
11–10 Breakpoint 0 and 1 Event Select	BT1 = 0	BT0 = 0	Breakpoint 0 and Breakpoint 1
	BT1 = 0	BT0 = 1	Breakpoint 0 or Breakpoint 1
	BT1 = 1	BT0 = 0	Breakpoint 1 after Breakpoint 0
	BT1 = 1	BT0 = 1	Breakpoint 0 after Breakpoint 1
9–8 Breakpoint 1 Condition Select	CC11 = 0	CC10 = 0	Breakpoint on not equal
	CC11 = 0	CC10 = 1	Breakpoint on equal
	CC11 = 1	CC10 = 0	Breakpoint on less than
	CC11 = 1	CC10 = 1	Breakpoint on greater than
7–6 Breakpoint 1 Read/Write Select	RW11 = 0	RW10 = 0	Breakpoint disabled
	RW11 = 0	RW10 = 1	Breakpoint on write access
	RW11 = 1	RW10 = 0	Breakpoint on read access
	RW11 = 1	RW10 = 1	Breakpoint on read or write access
5–4 Breakpoint 0 Condition Select	CC01 = 0	CC00 = 0	Breakpoint on not equal
	CC01 = 0	CC00 = 1	Breakpoint on equal
	CC01 = 1	CC00 = 0	Breakpoint on less than
	CC01 = 1	CC00 = 1	Breakpoint on greater than
3–2 Breakpoint 0 Read/Write Select	RW01 = 0	RW00 = 0	Breakpoint disabled
	RW01 = 0	RW00 = 1	Breakpoint on write access
	RW01 = 1	RW00 = 0	Breakpoint on read access
	RW01 = 1	RW00 = 1	Breakpoint on read or write access
1–0 Memory Breakpoint 0 and 1 Select	MBS1 = 0	MBS0 = 0	Breakpoint on DMA access
	MBS1 = 0	MBS0 = 1	Breakpoint on P access
	MBS1 = 1	MBS0 = 0	Breakpoint on X access
	MBS1 = 1	MBS0 = 1	Breakpoint on Y access

The OnCE Memory Limit Register 0 (OMLR0) stores the memory breakpoint 0 limit. The OnCE Memory Limit Register 1 (OMLR1) stores the memory breakpoint 1 limit. Both registers are 24-bit and can be read or written through the JTAG port.

## Pipeline Information and the OGDB Register

The 24-bit OnCE Memory Address Latch (OMAL) register latches the PAB, XAB, YAB, or DAB on every instruction cycle according to the Memory Breakpoint 0 and 1 Select MBS[1–0] bits in the OBCR. The OnCE Memory Address Comparator 0 (OMAC0) compares the current memory address stored in OMAL with the OMLR0 contents. The OnCE Memory Address Comparator 1 (OMAC1) compares the current memory address stored in OMAL with the OMLR1 contents.

The 24-bit OnCE Memory Breakpoint Counter (OMBC) 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 OBCR and the memory limit registers specify the memory access event. The breakpoint counter decrements on each occurrence of the memory access event. The DSP enters Debug mode when the counter reaches 0 and a new event occurs. The OMBC can be read or written through the JTAG port.

## 2.3 OnCE Trace Logic

The OnCE trace logic allows execution of instructions in single or multiple steps. It causes the DSP to enter Debug mode after one or more instructions execute and to wait for OnCE commands from the debug serial port.

The 24-bit OnCE Trace Counter (OTC) allows multiple instruction steps to be taken in real time before Debug mode is entered. It can be read or written through the JTAG port. If  $N$  instructions are to execute before Debug mode, the OTC should be loaded with  $N-1$ .

To enable trace mode, the OTC is loaded with a value, the program counter is set to the start location of the instruction(s) to be executed in real time, the trace mode that enables the TME bit in the OSCR is set, and the core exits Debug mode.

When the core exits Debug mode, the counter decrements after each execution of an instruction. All executed instructions, including fast interrupt services and repeated instructions, cause the OTC to decrement. When the OTC reaches 0, the core reenters Debug mode, the trace occurrence TO bit in the OSCR is set, the core status bits OS[1–0] in the OSCR are set, and the  $\overline{DE}$  signal is asserted to indicate the core has entered Debug mode and is requesting service.

## 2.4 Pipeline Information and the OGDB Register

The OnCE program data bus PDB Register (OPDBR) is a 24-bit latch that stores the value of the PDB generated by the last program memory access of the core before Debug mode is entered. It can be read or written through the JTAG port. This register is affected by the operations performed during Debug mode and must be restored by the external command controller upon return to Normal mode.

The OnCE Program Instruction Latch Register (OPILR) is a 24-bit latch that stores the value of the instruction latch before Debug mode is entered. It can be read only through the JTAG port. The host computer should immediately save this register when the Debug mode of operation is entered, so the exact state of the program controller pipeline can be restored upon return to User mode.

The OnCE GDB Register (OGDBR) is a 24-bit latch that can be read only through the JTAG port. The OGDBR is not actually required for restoring pipeline status, but is required for passing information between the DSP and the external command controller. This register is mapped to the X internal I/O space address \$FFFFFC. When the external command controller needs the contents of a register or memory location, it forces the DSP to execute an instruction that brings that information to the OGDBR. The contents of the OGDBR are then delivered serially to the external command converter by the command READ GDB REGISTER.

## 2.5 OnCE Trace Buffer

Three 24-bit program address bus (PAB) registers provide pipeline information when Debug mode is entered. A trace buffer stores the address of the last-executed instruction and the address of the last 12 change-of-flow instructions:

- The OnCE PAB Register for Fetch (OPABFR) stores the address of the last instruction whose fetch was started before Debug mode was entered. It can be read only through the JTAG port.
- The OnCE PAB Register for Decode (OPABDR) stores the address of the instruction currently in the program data bus. The fetch for this instruction was completed before the DSP entered Debug mode. It can be read only through the JTAG port.
- The OnCE PAB Register for Execute (OPABEX) stores the address of the instruction currently in the instruction latch. This instruction would have been decoded and executed if the DSP had not entered Debug mode. The OPABEX register can be read only through the JTAG port.

The circular trace buffer contains 25-bit registers and a 4-bit counter. Although all 12 of the trace buffer registers have the same address, any read access to the trace buffer address increments the counter, thus pointing to the next trace buffer register.

When the DSP enters Debug mode, 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 obtain the other addresses from the oldest to newest, in order of execution.

A complete set of 12 reads of the trace buffer must be performed, since each read increments the trace buffer pointer, thus pointing to the next location. After 12 reads, the pointer indicates the same location as it did before the read procedure started.

## 3 JTAG Port

The Joint Test Action Group (JTAG) port is a user-accessible test access port (TAP) that provides a way to enter Debug mode and access the OnCE controller. **Figure 3-1** shows the TAP block diagram.

## TAP Controller

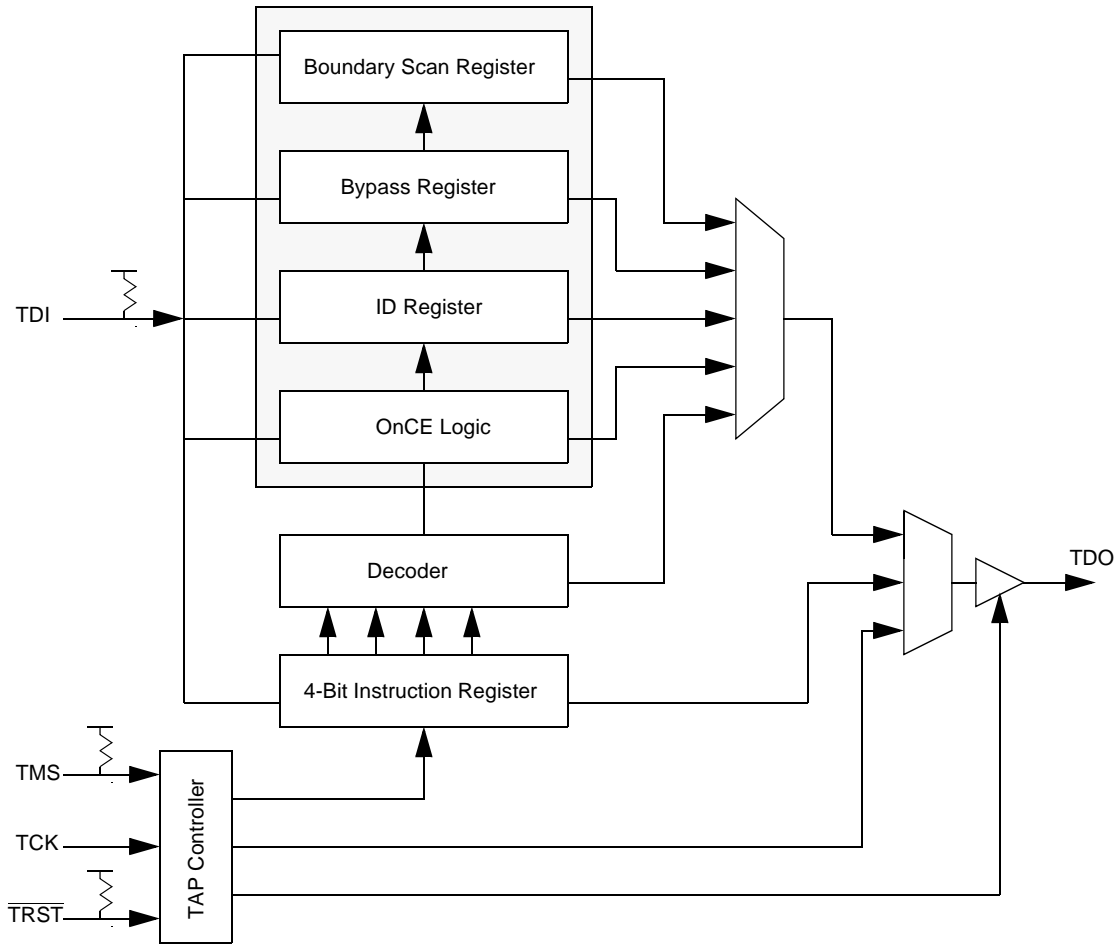


Figure 3-1. TAP Block Diagram

### 3.1 TAP Controller

The JTAG port consists of a TAP controller state machine that controls the operation of the JTAG logic (see **Figure 3-2**). The value on the TMS signal changes the TAP controller's state on the rising edge of the TCK signal.

There are two paths to the TAP controller state machine. The first path consists of Select\_DR\_Scan, Capture\_DR, Shift\_DR, Exit\_DR, and Update\_DR states, which capture and load data into the test data register. For example, reading the pipeline registers requires the use of this sequence. The second path consists of Select\_IR\_Scan, Capture\_IR, Shift\_IR, Exit\_IR, and Update\_IR states, which capture and load instructions into the instruction register. For example, shifting in the JTAG ENABLE\_ONCE instruction requires the use of this sequence.



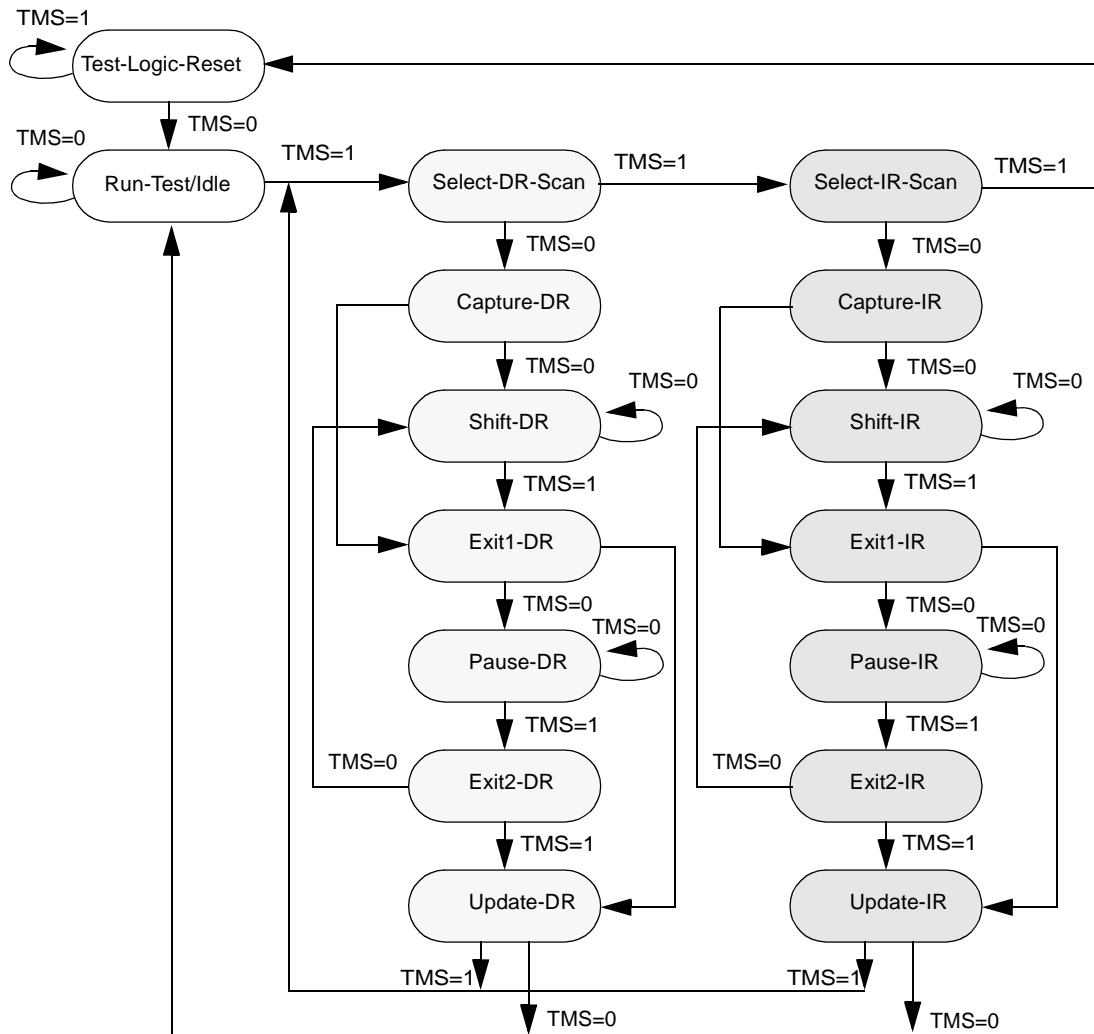


Figure 3-2. TAP Controller State Machine

## JTAG Instruction Register

### 3.2 Signals

The JTAG port has five signals: TCK, TMS, TDI, TDO, and  $\overline{\text{TRST}}$ . The OnCE port has one signal,  $\overline{\text{DE}}$ , for use by the OnCE module. **Table 3-1** describes the OnCE/JTAG signals.

**Table 3-1.** OnCE/JTAG Signals

Signal		Description
$\overline{\text{DE}}$	Debug Event	<ul style="list-style-type: none"> <li>Bidirectional.</li> <li>As input, allows an external command converter to initiate Debug mode.</li> <li>As output, acknowledges to an external command converter that the chip has entered Debug mode.</li> </ul>
TCK	Test Clock	<ul style="list-style-type: none"> <li>Input.</li> <li>Synchronizes the test logic.</li> </ul>
TMS	Test Mode Select	<ul style="list-style-type: none"> <li>Input.</li> <li>Sequences the state machine of the test controller.</li> <li>Sampled on the rising edge of TCK.</li> </ul>
TDI	Test Data Input	<ul style="list-style-type: none"> <li>Input.</li> <li>Receives serial test instructions and data.</li> <li>Sampled on the rising edge of TCK.</li> </ul>
TDO	Test Data Output	<ul style="list-style-type: none"> <li>Output.</li> <li>Sends test instructions and data.</li> <li>Changes on the falling edge of TCK.</li> </ul>
$\overline{\text{TRST}}$	Test Reset	<ul style="list-style-type: none"> <li>Input.</li> <li>Initializes the test controller.</li> </ul>

### 3.3 JTAG Instruction Register

The DSP56300 core includes a 4-bit instruction register consisting of a shift register with four parallel outputs. Data transfers from the shift register to the parallel outputs during the Update-IR controller state. The four bits decode the eight unique instructions shown in **Table 3-2**. The parallel inputs to the instruction shift register are loaded with 01 in the least significant bits, as required by the IEEE 1149.1 standard. The two most significant bits are loaded with the values of the core status bits OS[1–0] from the OSCR. **Table 3-3** shows the possible values of the parallel inputs to the instruction register.

**Table 3-2.** JTAG Instructions

B3	B2	B1	B0	INSTRUCTION	
0	0	0	0	EXTEST	<ul style="list-style-type: none"> <li>Selects the boundary scan register.</li> <li>Asserts internal reset for the core system logic to force a predictable internal state during external boundary scan operations.</li> </ul>
0	0	0	1	SAMPLE/PRELOAD	<ul style="list-style-type: none"> <li>Makes a snapshot of system data and control signals.</li> <li>Initializes the boundary scan register output cells prior to selection of EXTEST, which ensures that known data appears on the outputs when the EXTEST instruction is entered.</li> </ul>

**Table 3-2.** JTAG Instructions (Continued)

B3	B2	B1	B0	INSTRUCTION	
0	0	1	0	IDCODE	<ul style="list-style-type: none"> <li>Selects the identification register that allows the manufacturer, part number, and component version to be determined through the test access port.</li> </ul>
0	0	1	1	CLAMP	<ul style="list-style-type: none"> <li>Allows guarding values to be applied using the boundary scan register of the appropriate ICs while selecting their bypass registers.</li> </ul>
0	1	0	0	HI-Z	<ul style="list-style-type: none"> <li>Disables all output drivers.</li> </ul>
0	1	1	0	ENABLE_ONCE	<ul style="list-style-type: none"> <li>Allows the user to perform system debug functions.</li> </ul>
0	1	1	1	DEBUG_REQUEST	<ul style="list-style-type: none"> <li>Allows the user to generate a debug request signal to the DSP56300 core.</li> </ul>
1	1	1	1	BYPASS	<ul style="list-style-type: none"> <li>Selects the 1-bit BYPASS register, which enhances efficiency when the device under test is not a DSP56300 core-based device.</li> </ul>

**Table 3-3.** Parallel Inputs to the Instruction Shift Register

VALUE	STATUS
0001	User mode
0101	External Access/Bus Wait mode
1001	Step mode
1101	Debug mode

### 3.4 JTAG ID Register

The IDCODE instruction selects the 32-bit JTAG ID Register to allow the manufacturer, part number, and component version to be determined through the TAP. **Figure 3-3** 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 3-3.** Identification Register Configuration

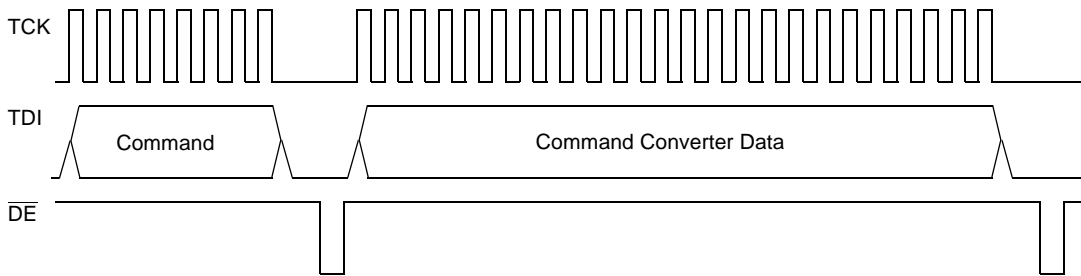
### 3.5 JTAG Boundary Scan Register

The JTAG Boundary Scan Register (BSR) contains bits for all device signals, clock signals, and associated control signals. All bidirectional signals have a single register bit in the BSR for signal data and are controlled by an associated control bit in the BSR.

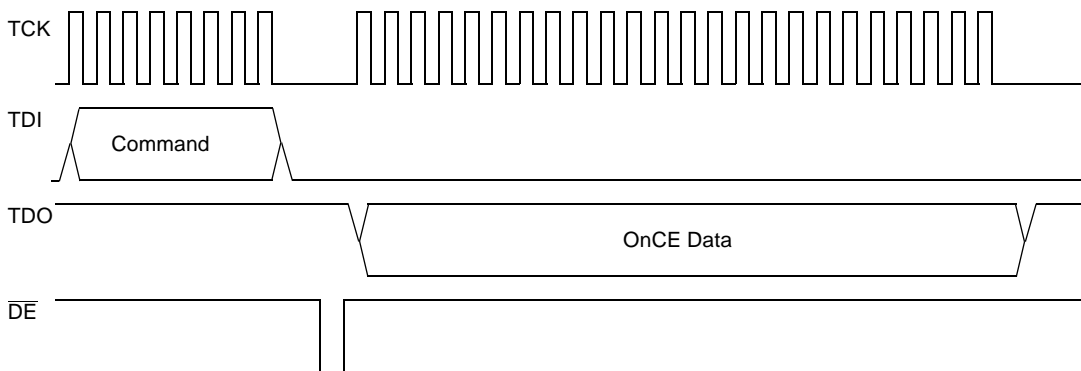
## 4 Serial Protocol

Before the DSP starts any debugging activity, the external command controller must wait for acknowledgment on  $\overline{DE}$  or it must poll the OS[1–0] bits in the JTAG instruction shift register. It communicates with the DSP by sending 8-bit commands that can be followed by 24 bits of data. Both command and data are sent least significant bit (LSB) first. An external command controller can send a new command only after DSP has acknowledges execution of the previous command.

**Figure 4-1** shows the relative timing of a write command and **Figure 4-2** shows the relative timing of a read command.



**Figure 4-1.** Write Command Timing



**Figure 4-2.** Read Command Timing

# 5 Examples of OnCE/JTAG Interaction

This section contains several OnCE/JTAG software routine examples based on the following system setup:

- DSP56300 device for receiving OnCE/JTAG signals (EVM 1)
- DSP56300 device for transmitting OnCE/JTAG signals (EVM 2)
- PC with Motorola DSP56300 tools
- Logic analyzer

The OnCE/JTAG emulation software is downloaded from the PC to the EVM 2 DSP via the OnCE port. EVM 2 acts as an external command controller, controlling the OnCE/JTAG signals of EVM 1. The logic analyzer examines relative signal timing. **Figure 5-1** shows the relationship among these components.

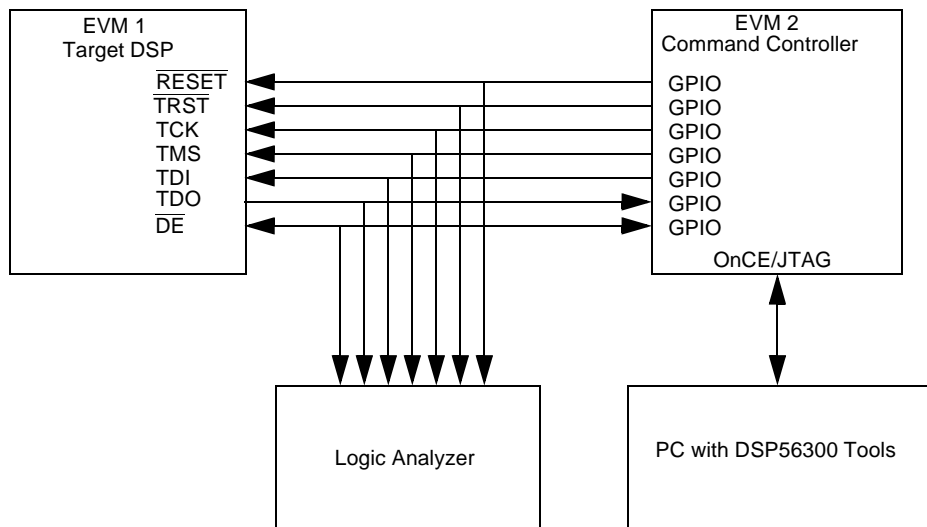


Figure 5-1. System Setup

## 5.1 Entering the Test-Logic-Reset State

To keep JTAG test logic transparent to system logic, the TAP controller must be forced into the test-logic-reset controller state, which requires:

- Asserting  $\overline{\text{TRST}}$  externally during power-up
- Sampling TMS as a logical 1 for five consecutive TCK rising edges after power-up completes.

Once the test-logic-reset state is entered, the run-test/idle state can be entered by calling the subroutine JTAG\_RTI, as **Example 1** shows.

Example 1. JTAG Run-Test/Idle Mode Sequence

```

org      x:
JTAG_RTI_SEQ:
dc      $30      ; go to next state
dc      $30      ; go to next state
dc      $30      ; go to next state
dc      $30      ; go to next state
    
```

## Entering the Test-Logic-Reset State

```

dc      $30    ; go to next state
dc      $10    ; go to Run-Test-Idle
dc      $00    ; EXIT

org     p:
main:
...
jsr     JTAG_RTI
...

JTAG_RTI:
move    #JTAG_RTI_SEQ,r0
jsr     JTAG_EXECUTE
rts

```

The JTAG\_RTI subroutine sends a sequence of 8-bit data to the JTAG\_EXECUTE subroutine. **Table 5-1** lists the JTAG\_RTI subroutine bit definitions.

**Table 5-1.** JTAG\_RTI Subroutine Bit Definitions

7-6	5	4	3	2	1-0
Reserved	TMS to send	TDI to send	Reserved	Read TDO	Reserved

For example, a value of \$30 indicates that TMS = 1, TDI = 1 are sent on the rising edge of TCK and TDO is not read on the falling edge of TCK. Thus, JTAG\_RTI sends a value \$30 five times to enter the test-reset-logic state and then sends a value of \$10 (TMS = 0, TDI =1, TDO is not read) to enter the run-test/idle state.

The JTAG\_EXECUTE subroutine shown in **Example 2** emulates OnCE/JTAG operation. The JTAG\_EXECUTE routine is repeated until an exit value of \$00 is encountered. For example, when the JTAG\_RTI sends an 8-bit data, the JTAG\_EXECUTE first determines if bit 2 is set to indicate that TDO needs to be read. Next, the JTAG\_EXECUTE subroutine determines the value of bit 5, then sets or clears the TMS value accordingly. Next, the subroutine determines the value of bit 4 and sets or clears the TDI value accordingly. The TCK signal is then toggled to send the values on the JTAG signals.

**Example 2.** JTAG\_EXECUTE Subroutine

---

```

JTAG_EXECUTE:
move    x:(r0)+,a1
tst     a
beq     done
move    a1,x:JTAG_CMD
brclr   #DATA_RD,x:JTAG_CMD,no_read
read_TDO
brclr   #TDO_BIT,x:M_PDRD,TDO_CLR
TDO_SET
move    #>1,a
bra     no_read
TDO_CLR
move    #>0,a
no_read
brclr   #DATA_TMS,x:JTAG_CMD,TMS_CLR
TMS_SET
bset    #TMS_BIT,x:M_PDRD
bra     >cont1
TMS_CLR
bclr   #TMS_BIT,x:M_PDRD
cont1
brclr   #DATA_TDI,x:JTAG_CMD,TDI_CLR
TDI_SET
bset    #TDI_BIT,x:M_PDRD
bra     >cont2
TDI_CLR
bclr   #TDI_BIT,x:M_PDRD
cont2
bset    #TCK_BIT,x:M_PDRD
rep     #3

```

```

nop
bclr  #TCK_BIT,x:M_PDRD
bra   >JTAG_EXECUTE
done
rts

```

## 5.2 Executing the JTAG DEBUG\_REQUEST Instruction

Once the TAP controller is in the run-test/idle state, the JTAG DEBUG\_REQUEST instruction can be executed to assert an internal debug request signal. As **Table 3-2** on page 10 shows, to send the DEBUG\_REQUEST instruction, the JTAG Instruction Register requires a value of 0111. Note that the most significant bit of the JTAG Instruction Register is sent first. **Example 3** shows the JTAG DEBUG\_REQUEST sequence.

**Example 3.** JTAG DEBUG\_REQUEST Sequence

---

```

org     x:
JTAG_DR_SEQ:
dc     $30      ; go to Select DR
dc     $30      ; go to Select IR
dc     $10      ; go to Capture IR
dc     $10      ; go to Shift IR
dc     $14      ; go to Shift IR - 1
dc     $14      ; go to Shift IR - 1
dc     $14      ; go to Shift IR - 1
dc     $24      ; go to Exit IR - 0
dc     $30      ; go to Update IR
dc     $10      ; go to Run-Test-Idle
dc     $00      ; EXIT

org     p:
main:
...
jsr    JTAG_DR
...

JTAG_DR:
move   #JTAG_DR_SEQ,r0
jsr    JTAG_EXECUTE
rts

```

## 5.3 Executing the JTAG ENABLE\_ONCE Instruction

To determine whether the DSP has entered Debug mode, the status of the JTAG Instruction Shift Register can be polled by shifting in the JTAG ENABLE\_ONCE instruction and reading the status information that is shifted out. The JTAG Instruction Register requires a value of 0110 (see **Table 3-2** on page 10). At the same time that the ENABLE\_ONCE instruction is shifted in, the parallel inputs to the instruction shift register should have a value of 1101 (see **Table 3-3** on page 11) to indicate that the DSP has entered Debug mode. **Example 4** shows the JTAG ENABLE\_ONCE sequence.

**Example 4.** JTAG ENABLE\_ONCE Instruction

---

```

org     x:
JTAG_ENBL_ONCE_SEQ:
dc     $30      ; go to Select DR
dc     $30      ; go to Select IR
dc     $10      ; go to Capture IR
dc     $10      ; go to Shift IR
dc     $04      ; go to Shift IR - 0
dc     $14      ; go to Shift IR - 1
dc     $14      ; go to Shift IR - 1
dc     $24      ; go to Exit IR - 0
dc     $30      ; go to Update IR
dc     $10      ; go to Run-Test-Idle
dc     $00      ; EXIT

```

## Reading the Pipeline Registers

```

org          p:
main:
...
jsr          JTAG_ENBL_ONCE
...

JTAG_ENBL_ONCE:
move        #JTAG_ENBL_ONCE_SEQ,r0
jsr         JTAG_EXECUTE
rts

```

Another way to determine whether the DSP has entered Debug mode is to monitor the  $\overline{DE}$  signal. Every time the DSP acknowledges the execution of an instruction in Debug mode, it generates a pulse.

## 5.4 Reading the Pipeline Registers

After the JTAG instructions `DEBUG_REQUEST` and `ENABLE_ONCE` complete, and after the core status has been polled to verify that the DSP is in Debug mode, the pipeline must be saved before debugging starts.

As **Example 5** shows, reading the pipeline registers requires the following steps:

1. Send the 8-bit command `$8B` to read the `OPILR`, no `GO`, no `EXIT` and read the 24-bit data.
2. Send the 8-bit command `$8A` to read the `OPDBR`, no `GO`, no `EXIT` and read the 24-bit data.

### Example 5. Pipeline Register Read Sequence

---

```

org          x:
JTAG_PIPELN_RD_SEQ:
dc          $30      ; go to Select DR
dc          $10      ; go to Capture DR
                                   ;command: $8B
dc          $10      ; go to Shift DR
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
                                   ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                   ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $24      ; go to Exit DR - 0
                                   ;read data
dc          $30      ; go to Update DR
dc          $30      ; go to Select DR

```



```

dc          $10      ; go to Capture DR
;command: $8A
dc          $10      ; go to Shift DR
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $24      ; go to Exit DR - 0

dc          $30      ; go to Update DR
dc          $10      ; go to Run-Test-Idle
dc          $00      ; Exit

org         p:
main:
...
jsr        JTAG_PIPELN_RD
...

JTAG_XRAM_WR:
move      #JTAG_PIPELN_RD_SEQ,r0
jsr      JTAG_EXECUTE
rts

```

## 5.5 Reading the Trace Buffer

Reading the information associated with the trace buffer enables an external program to reconstruct the full trace of the executed program.

As **Example 6** shows, reading the trace buffer requires the following steps:

1. Send the 8-bit command \$8F to read the OPABFR, no GO, no EXIT and read the 24-bit data.
2. Send the 8-bit command \$90 to read the OPABDR, no GO, no EXIT and read the 24-bit data.
3. Send the 8-bit command \$91 to read the OPABEX, no GO, no EXIT and read the 24-bit data.
4. Send the 8-bit command \$92 to read the Trace Buffer and Increment Pointer, no GO, no EXIT and read the 25-bit data.
5. Repeat the last step for the entire FIFO (12 times).

## Reading the Trace Buffer

### Example 6. Trace Buffer Read Sequence

```

org          x:
JTAG_TRBUFF_RD_SEQ:
dc          $30      ; go to Select DR
dc          $10      ; go to Capture DR
                                     ;command: $8F
dc          $10      ; go to Shift DR
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
                                     ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $24      ; go to Exit DR - 0
dc          $30      ; go to Update DR
dc          $30      ; go to Select DR
dc          $10      ; go to Capture DR
                                     ;command: $90
dc          $10      ; go to Shift DR
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read data
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0

```

## Reading the Trace Buffer

```

dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $24      ; go to Exit DR  - 0

dc      $30      ; go to Update DR
dc      $30      ; go to Select DR
dc      $10      ; go to Capture DR

;command: $91
dc      $10      ; go to Shift DR
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1

;read data
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0

;read data
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0

;read data
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $24      ; go to Exit DR  - 0

dc      $30      ; go to Update DR

;REPEAT THIS SEQUENCE 12X.
dc      $30      ; go to Select DR
dc      $10      ; go to Capture DR

;command: $92
dc      $10      ; go to Shift DR
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1

;read data
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0

;read data
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0

;read data
dc      $04      ; go to Shift DR - 0

```

## Writing to Memory

```
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $24      ; go to Exit DR - 0

dc      $30      ; go to Update DR
dc      $10      ; go to Run-Test-Idle
dc      $00      ; Exit

org      p:
main:
...
jsr      JTAG_TRBUFF_RD
...

JTAG_XRAM_WR:
move     #JTAG_TRBUFF_RD_SEQ,r0
jsr      JTAG_EXECUTE
rts
```

## 5.6 Writing to Memory

This example shows how to write a value to X memory. The assembly code to perform this write to memory is:

```
move #c0ffee,x0 ;opcode:      $44f400
;data:          $c0ffee
move x0,x:(r0)+ ;opcode:      $448500
```

As **Example 7** shows, writing to memory requires the following steps:

1. Send the 8-bit command \$0A to write to the OPDBR, no GO, no EXIT and send the 24-bit opcode \$44f400.
2. Send the 8-bit command \$4A to write to the OPDBR, GO, no EXIT and send the 24-bit data \$c0ffee.
3. Send the 8-bit command \$4A to write to the OPDBR, GO, no EXIT and send the 24-bit opcode \$448500.

### Example 7. Write-to-Memory Sequence

---

```
org      x:
JTAG_XRAM_WR_SEQ:
dc      $30      ; go to Select DR
dc      $10      ; go to Capture DR
;command: $0A

dc      $10      ; go to Shift DR
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
;data: $00

dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
;data: $F4

dc      $04      ; go to Shift DR - 0
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
dc      $04      ; go to Shift DR - 0
dc      $14      ; go to Shift DR - 1
```

```

dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
;data: $44
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $24      ; go to Exit DR - 0

dc          $30      ; go to Update DR
dc          $30      ; go to Select DR
dc          $10      ; go to Capture DR
;command: $4A
dc          $10      ; go to Shift DR
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
;data: $EE
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
;data: $FF
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
;data: $C0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $34      ; go to Exit DR - 1
;
dc          $30      ; go to Update DR
dc          $30      ; go to Select DR
dc          $10      ; go to Capture DR
;command: $4A
dc          $10      ; go to Shift DR
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
;data: $00
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
;data: $85
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0

```

## Reading from Memory

```

dc          $14    ; go to Shift DR - 1
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $24    ; go to Exit DR - 0

dc          $30    ; go to Update DR
dc          $10    ; go to Run-Test-Idle
dc          $00    ; Exit

org          p:
main:
...
jsr          JTAG_XRAM_WR
...

JTAG_XRAM_WR:
move        #JTAG_XRAM_WR_SEQ,r0
jsr          JTAG_EXECUTE
rts

```

## 5.7 Reading from Memory

This example shows how to read a value from P memory. The assembly code to perform this read from memory is:

```
movep (r0)+,x:OGDB          ;opcode: $08d87c
```

As **Example 8** shows, reading from memory requires the following steps:

1. Send the 8-bit command \$4A to write to the OPDBR, GO, no EXIT and send the 24-bit opcode \$08d87c.
2. Send the 8-bit command \$89 to read the OGDBR no GO, no EXIT and read the 24-bit data.

### Example 8. Read-from-Memory Sequence

---

```

org          x:
JTAG_PRAM_RD_SEQ:
dc          $30    ; go to Select DR
dc          $10    ; go to Capture DR

dc          $10    ; go to Shift DR
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $04    ; go to Shift DR - 0

dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $14    ; go to Shift DR - 1
dc          $14    ; go to Shift DR - 1
dc          $14    ; go to Shift DR - 1
dc          $14    ; go to Shift DR - 1
dc          $04    ; go to Shift DR - 0

dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $04    ; go to Shift DR - 0
dc          $14    ; go to Shift DR - 1
dc          $14    ; go to Shift DR - 1

```

```


dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $14      ; go to Shift DR - 1
                                     ;data: $08
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $24      ; go to Exit DR - 0
                                     ;
dc          $30      ; go to Update DR
dc          $30      ; go to Select DR
dc          $10      ; go to Capture DR
                                     ;command: $89
dc          $10      ; go to Shift DR
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $14      ; go to Shift DR - 1
                                     ;read byte
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read byte
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
                                     ;read byte
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $04      ; go to Shift DR - 0
dc          $24      ; go to Exit DR - 0
                                     ;
dc          $30      ; go to Update DR
dc          $10      ; go to Run-Test-Idle
dc          $00      ; Exit

org         p:
main:
...
    jsr     JTAG_PRAM_RD
...

JTAG_XRAM_WR:
    move   #JTAG_PRAM_RD_SEQ,r0
    jsr     JTAG_EXECUTE
    rts

```

OnCE and Mfax are trademarks of Motorola, Inc.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

#### How to reach us:

**USA/EUROPE  
Motorola Literature Distribution**  
P.O. Box 5405  
Denver, Colorado 80217  
1-303-675-2140  
1-800-441-2447

**Customer Focus Center**  
1-800-521-6274

**JAPAN**  
Motorola Japan Ltd.  
SPS, Technical Information Center  
3-20-1, Minami-Azabu, Minato-ku  
Tokyo 106-8573 Japan  
81-3-3440-3569

**ASIA/PACIFIC**  
Motorola Semiconductors H.K. Ltd.  
Silicon Harbour Centre  
2 Dai King Street  
Tai Po Industrial Estate  
Tai Po, N.T., Hong Kong  
852-26668334

**Mfax™, Motorola Fax Back System**  
RMFAX0@email.sps.mot.com  
<http://sps.motorola.com/mfax/>  
TOUCHTONE: 1-602-244-6609  
USA and Canada ONLY:  
1-800-774-1848

**Home Page**  
<http://www.mot.com/SPS/DSP/>

**DSP Helpline**  
[dsphelp@dsp.sps.mot.com](mailto:dsphelp@dsp.sps.mot.com)

