

---

## 10 ON-CHIP EMULATOR (OnCE™)

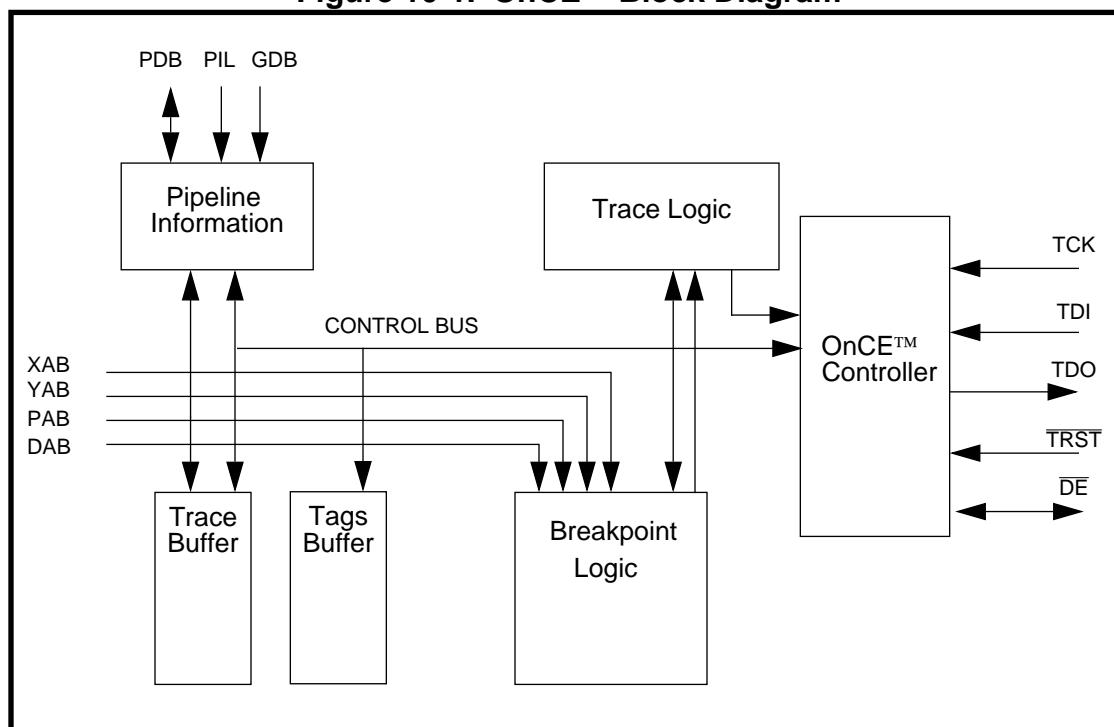
---

### 10.1 INTRODUCTION

---

The DSP56300 Core on-chip emulation (OnCE™) circuitry provides a means of interacting with the DSP56300 Core and its peripherals non-intrusively so that a user may examine registers, memory or on-chip peripherals facilitating hardware/software development on the DSP56300 Core processor. To achieve this, special circuits and dedicated pins on the DSP56300 Core are defined to avoid sacrificing any user-accessible on-chip resource. The OnCE™ resources can be accessed only after executing the JTAG instruction ENABLE\_ONCE (these resources are accessible even when the chip is operating in Normal Mode). See Chapter 11 for a description of the JTAG functionality and its relation to the OnCE. Figure 10-1 illustrates the block diagram of the OnCE™.

**Figure 10-1. OnCE™ Block Diagram**



### 10.2 ON-CHIP EMULATION (OnCE™) PINS

---

Since the OnCE controller functionality is accessed through the JTAG port, there are no

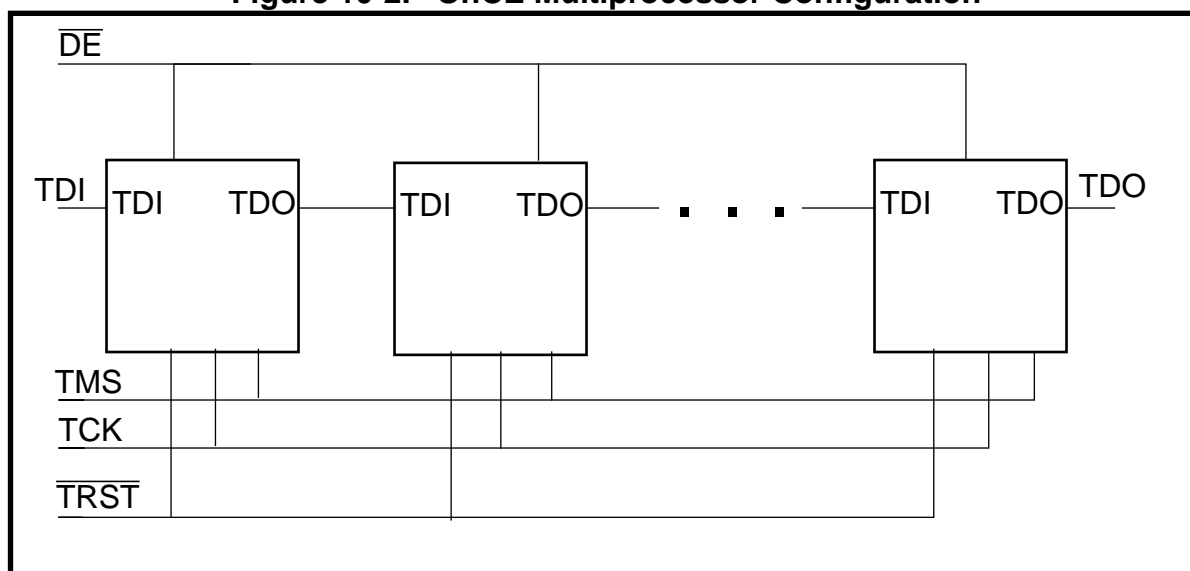
dedicated OnCE pins for clock, data-in and data-out. The JTAG pins TCK, TDI and TDO are used to shift in and out data and/or instructions. See Paragraph 11.2.1 for the description of the JTAG pins. In order to facilitate emulation specific functions, one additional pin, called  $\overline{DE}$ , may be used. This pin is described below.

### 10.2.1 Debug Event ( $\overline{DE}$ )

The bidirectional open drain debug event pin  $\overline{DE}$  provides a fast means of entering the Debug Mode of operation from an external command controller (when input) as well as a fast means of acknowledging the entering the Debug Mode of operation to an external command controller (when output). The assertion of this pin by a command controller causes the DSP56300 Core to finish the current instruction being executed, save the instruction pipeline information, enter the Debug Mode, and wait for commands to be entered from the TDI line. If  $\overline{DE}$  was used to enter the Debug Mode then  $\overline{DE}$  must be negated after the OnCE™ responds with an acknowledge and before sending the first OnCE™ command. The assertion of this pin by the DSP56300 Core indicates that the DSP has entered the Debug Mode and is waiting for commands to be entered from the TDI line.

The  $\overline{DE}$  pin also facilitates multiple processor connections as depicted in Figure 10-2.

**Figure 10-2. OnCE Multiprocessor Configuration**



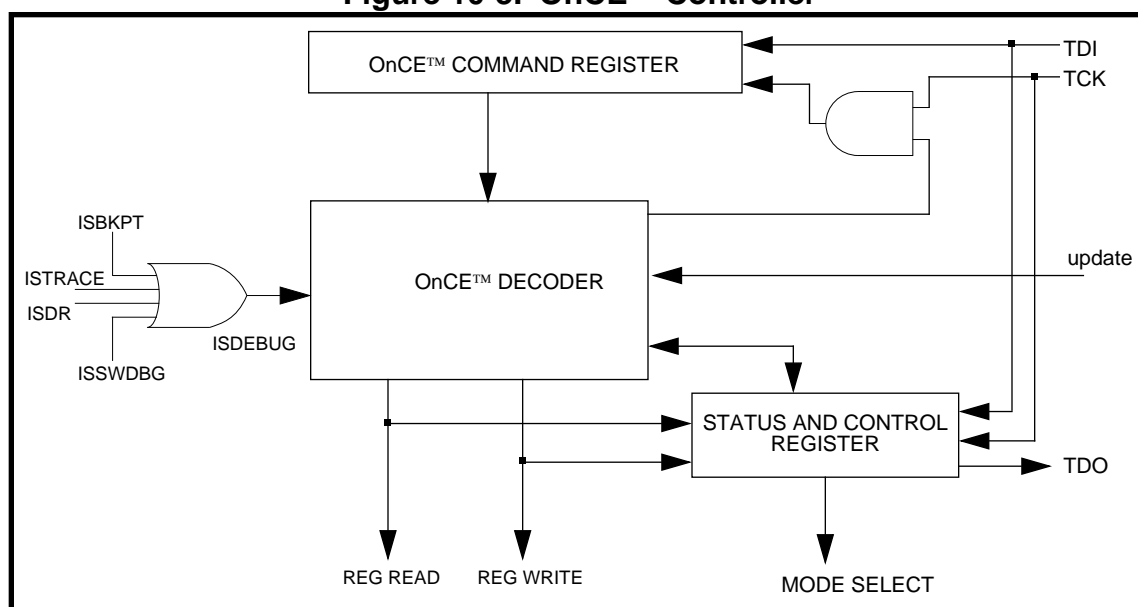
In this way the user is able to stop all the devices in the system when one of the devices has entered the Debug Mode. The user can also stop all the devices synchronously by asserting the  $\overline{DE}$  line.

## 10.3 OnCE™ CONTROLLER

The OnCE™ Controller contains the following blocks: OnCE™ command register, OnCE™

decoder, and the status/control register. Figure 10-3 illustrates a block diagram of the OnCE™ controller.

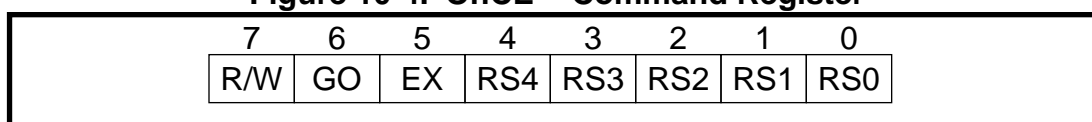
**Figure 10-3. OnCE™ Controller**



### 10.3.1 OnCE™ Command Register (OCR)

The OnCE™ Command Register is an 8-bit 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 Command Register is shown in Figure 10-4.

**Figure 10-4. OnCE™ Command Register**



#### 10.3.1.1 Register Select (RS4-RS0) Bits 0-4

The Register Select bits define which register is source/destination for the read/write operation. See Table 10-1 for the OnCE™ register addresses.

#### 10.3.1.2 Exit Command (EX) Bit 5

If the EX bit is set, leave Debug Mode and resume normal operation. The Exit command is executed only if the Go command is issued, and the operation is write to OPDBR or read/write to “No Register Selected”. Otherwise the EX bit is ignored.

EX	Action
0	remain in Debug Mode
1	leave Debug Mode

---

**Table 10-1. OnCE™ Register Addressing.**

---

RS4-RS0	Register Selected
00000	OnCE™ Status and Control Register (OSCR)
00001	Memory Breakpoint Counter (OMBC)
00010	Breakpoint Control Register (OBCR)
00011	Reserved
00100	Reserved
00101	Memory Limit Register0 (OMLR0)
00110	Memory Limit Register1 (OMLR1)
00111	Reserved
01000	Reserved
01001	GDB Register (OGDBR)
01010	PDB Register (OPDBR)
01011	PIL Register (OPILR)
01100	PDB GO-TO Register (for GO TO command)
01101	Trace Counter (OTC)
01110	Tags Buffer (TAGB)
01111	PAB Register for Fetch (OPABFR)
10000	PAB Register for Decode (OPABDR)
10001	PAB Register for Execute (OPABEX)
10010	Trace Buffer and Increment Pointer
10011	Reserved Address
101xx	Reserved Address
11xx0	Reserved Address
11x0x	Reserved Address
110xx	Reserved Address
11111	No Register Selected

---

### 10.3.1.3 Go Command (GO) Bit 6

If the GO bit is set, execute instruction which resides in the PIL register. To execute the instruction, the core leaves the Debug Mode. The core will return to the Debug Mode immediately after executing the instruction if the EX bit is cleared. The core goes on to normal operation if the EX bit is set. The GO command is executed only if the operation is write to OPDBR or read/write to “No Register Selected”. Otherwise the GO bit is ignored.

GO	Action
0	inactive (no action taken)
1	execute instruction in PIL

### 10.3.1.4 Read/Write Command (R/W) Bit 7

The R/W bit specifies the direction of data transfer.

R/W	Action
0	write the data associated with the command into the register specified by RS4-RS0
1	read the data contained in the register specified by RS4-RS0

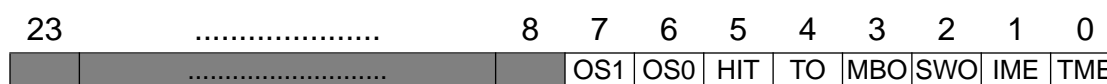
## 10.3.2 OnCE™ Decoder (ODEC)

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

## 10.3.3 OnCE™ Status and Control Register (OSCR)

The Status and Control Register is a 24-bit register used to enable the trace mode of operation and to indicate the cause of entering the Debug Mode. The control bits are read/write while the status bits are read only. The OSCR bits are cleared on hardware reset. The OSCR is shown in Figure 10-5.

**Figure 10-5. OnCE™ Status and Control Register (OSCR)**



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

---

#### 10.3.3.1 Trace Mode Enable (TME) Bit 0

The TME control bit, when set, enables the Trace Mode of operation.

#### 10.3.3.2 Interrupt Mode Enable (IME) Bit 1

The IME control bit, when set, will cause the chip to execute a vectored interrupt to the address VBA:\$06 instead of entering the Debug Mode.

#### 10.3.3.3 Software Debug Occurrence (SWO) Bit 2

This read-only status bit is set when the Debug Mode of operation is entered due to the execution of the DEBUG or DEBUGcc instruction with condition true. This bit is cleared when leaving the Debug Mode.

#### 10.3.3.4 Memory Breakpoint Occurrence (MBO) Bit 3

This read-only status bit is set when the Debug Mode of operation is entered due to the occurrence of a memory breakpoint. This bit is cleared when leaving the Debug Mode.

#### 10.3.3.5 Trace Occurrence (TO) Bit 4

This read-only status bit is set when the Debug Mode of operation is entered when the trace counter is zero while trace mode is enabled. This bit is cleared when leaving the Debug Mode.

#### 10.3.3.6 Cache Hit (HIT) Bit 5

This read only status bit is set when a cache hit has occurred when in cache mode and in Debug Mode of operation. When in PRAM mode this bit will read as one.

#### 10.3.3.7 Core Status (OS0,OS1) Bits 6-7

These read only status bits provide core status information. By examining the status bits the user can determine whether the chip has entered the Debug Mode (examining SWO, MBO and TO will identify the cause of entering the Debug Mode). The user can also examine these bits and determine the cause why the chip has not entered the Debug Mode after debug event assertion ( $\overline{DE}$ ) or as a result of the execution of the JTAG Debug Request instruction (core waiting for the bus, STOP or WAIT instruction etc.). These bits are also reflected in the JTAG instruction shift register which allows the polling of the core status information at the JTAG level. This is useful for the case in which the DSP56300 Core executes the STOP instruction (and therefore there are no clocks) to allow the reading of OSCR. See Table 10-4 for the definition of the OS0-OS1 bits.

---

**Table 10-2. Core Status Bits Description.**

---

OS1	OS0	DESCRIPTION
0	0	DSP56300 Core is executing instructions
0	1	DSP56300 Core is in Wait or STOP
1	0	DSP56300 Core is waiting for bus
1	1	DSP56300 Core is in Debug Mode

#### 10.3.3.8 Reserved Bits 8-23

These bits are reserved for future use. They read as zero and should be written with zero for future compatibility.

## 10.4 OnCE™ MEMORY BREAKPOINT LOGIC

---

Memory breakpoints may be set on program memory or data memory locations. Also, 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 the programmer's ability to monitor what the program is doing in real-time.

The breakpoint logic, described in Figure 10-6., 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 being written to areas that should not be written to. 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 the user to set breakpoints in RAM or ROM and while in any operating mode. Memory accesses are monitored according to the contents of the OBCR as specified in Section 10.4.6.

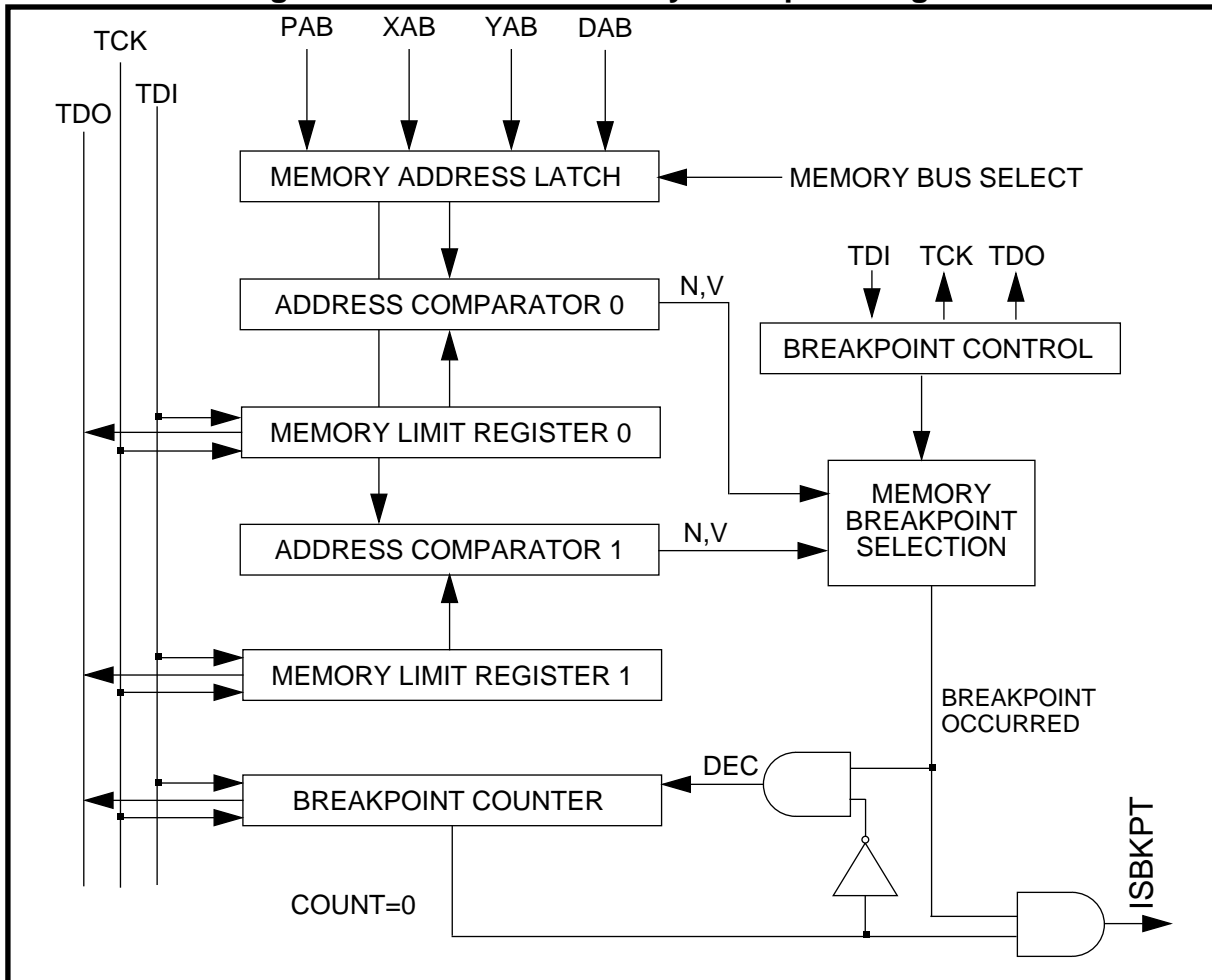
### 10.4.1 Memory Address Latch (OMAL)

The Memory Address Latch is a 24-bit register that latches the PAB, XAB, YAB or DAB on every instruction cycle according to the MBS1-MBS0 bits in OBCR.

### 10.4.2 Memory Limit Register 0 (OMLR0)

The Memory Limit Register 0 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.

**Figure 10-6. OnCE™ Memory Breakpoint Logic 0**



#### 10.4.3 Memory Address Comparator 0 (OMAC0)

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

#### 10.4.4 Memory Limit Register 1 (OMLR1)

The Memory Limit Register1 is a 24-bit register that stores the memory breakpoint limit. OMLR1 can be read or written through JTAG port. Before enabling breakpoints, OMLR1 must be loaded by the external command controller.

#### 10.4.5 Memory Address Comparator 1 (OMAC1)

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

#### 10.4.6 Breakpoint Control Register (OBCR)


The Breakpoint Control Register is a 24-bit register used to define the memory breakpoint events. OBCR can be read or written through the JTAG port. All the bits of the OBCR are



cleared on hardware reset. The OBCR is described in Figure 10-7.

**Figure 10-7. Breakpoint Control Register**

23	22	21	20	19	18	17	16	15	14	13	12
11	10	9	8	7	6	5	4	3	2	1	0
BT1	BT0	CC11	CC10	RW11	RW10	CC01	CC00	RW01	RW00	MBS1	MBS0

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

#### 10.4.6.1 Memory Breakpoint Select (MBS0-MBS1) Bits 0-1

These control bits enable memory breakpoints 0 and 1, allowing them to occur when a memory access is performed on P, X or Y space or when a DMA access is performed. See the following table for the definition of the MBS0-MBS1 bits.

**Table 10-3. Memory Breakpoint 0 and 1 Select Table.**

MBS1	MBS0	DESCRIPTION
0	0	Breakpoint on DMA access
0	1	Breakpoint on P access
1	0	Breakpoint on X access
1	1	Breakpoint on Y access

#### 10.4.6.2 Breakpoint 0 Read/Write Select (RW00-RW01) Bits 2-3

These control bits define the memory breakpoints 0 to occur when a memory address accesses is performed for read, write or both. See the following table for the definition of the RW00-RW01 bits.

**Table 10-4. Breakpoint 0 Read/Write Select Table**

RW01	RW00	DESCRIPTION
0	0	Breakpoint disabled
0	1	Breakpoint on write access
1	0	Breakpoint on read access
1	1	Breakpoint on read or write access

---

#### 10.4.6.3 Breakpoint 0 Condition Code Select (CC00-CC01) Bits4-5

These control bits define the condition of the comparison between the current memory address (OMAL) and the memory limit register 0 (OMLR0). See the following table for the definition of the CC00-CC01 bits.

**Table 10-5. Breakpoint 0 Condition Select Table**

CC01	CC00	DESCRIPTION
0	0	Breakpoint on not equal
0	1	Breakpoint on equal
1	0	Breakpoint on less than
1	1	Breakpoint on greater than

#### 10.4.6.4 Breakpoint1 Read/Write Select (RW10-RW11) Bits 6-7

These control bits define memory breakpoints 1 to occur when a memory address accesses is performed for read, write or both. See the following table for the definition of the RW10-RW11 bits.

**Table 10-6. Breakpoint 1 Read/Write Select Table**

RW11	RW10	DESCRIPTION
0	0	Breakpoint disabled
0	1	Breakpoint on write access
1	0	Breakpoint on read access
1	1	Breakpoint read or write access

#### 10.4.6.5 Breakpoint1 Condition Code Select (CC10-CC11) Bits8-9

These control bits define the condition of the comparison between the current memory address (OMAL) and the memory limit register 1 (OMLR1). See the following table for the definition of the CC10-CC11 bits.

---

**Table 10-7. Breakpoint 1 Condition Select Table**

---

CC11	CC10	DESCRIPTION
0	0	Breakpoint on not equal
0	1	Breakpoint on equal
1	0	Breakpoint on less than
1	1	Breakpoint on greater than

#### 10.4.6.6 Breakpoint 0 and 1 Event Select (BT1-BT0) Bits10-11

These control bits define the sequence between breakpoint 0 and 1. If the condition defined by BT1-BT0 is met, then the Breakpoint Counter (OMBC) is decremented. See the following table for the definition of the BT1-BT0 bits.

---

**Table 10-8. Breakpoint 0 and 1 Event Select Table**

---

BT1	BT0	DESCRIPTION
0	0	Breakpoint 0 and Breakpoint 1
0	1	Breakpoint 0 or Breakpoint 1
1	0	Breakpoint 1 after Breakpoint 0
1	1	Breakpoint 0 after Breakpoint 1

#### 10.4.7 Memory Breakpoint Counter (OMBC)

The Memory Breakpoint Counter is a 24-bit counter which 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 register and by the memory limit registers. On each occurrence of the memory access event, the breakpoint counter is decremented. When the counter has reached the value of zero and a new occurrence takes place, the chip will enter the Debug Mode. The OMBC can be read or written through the JTAG port. Every time that the limit register is changed, 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 will affect the new breakpoint event selected. The breakpoint counter is cleared by hardware reset.

---

## 10.5 CACHE SUPPORT

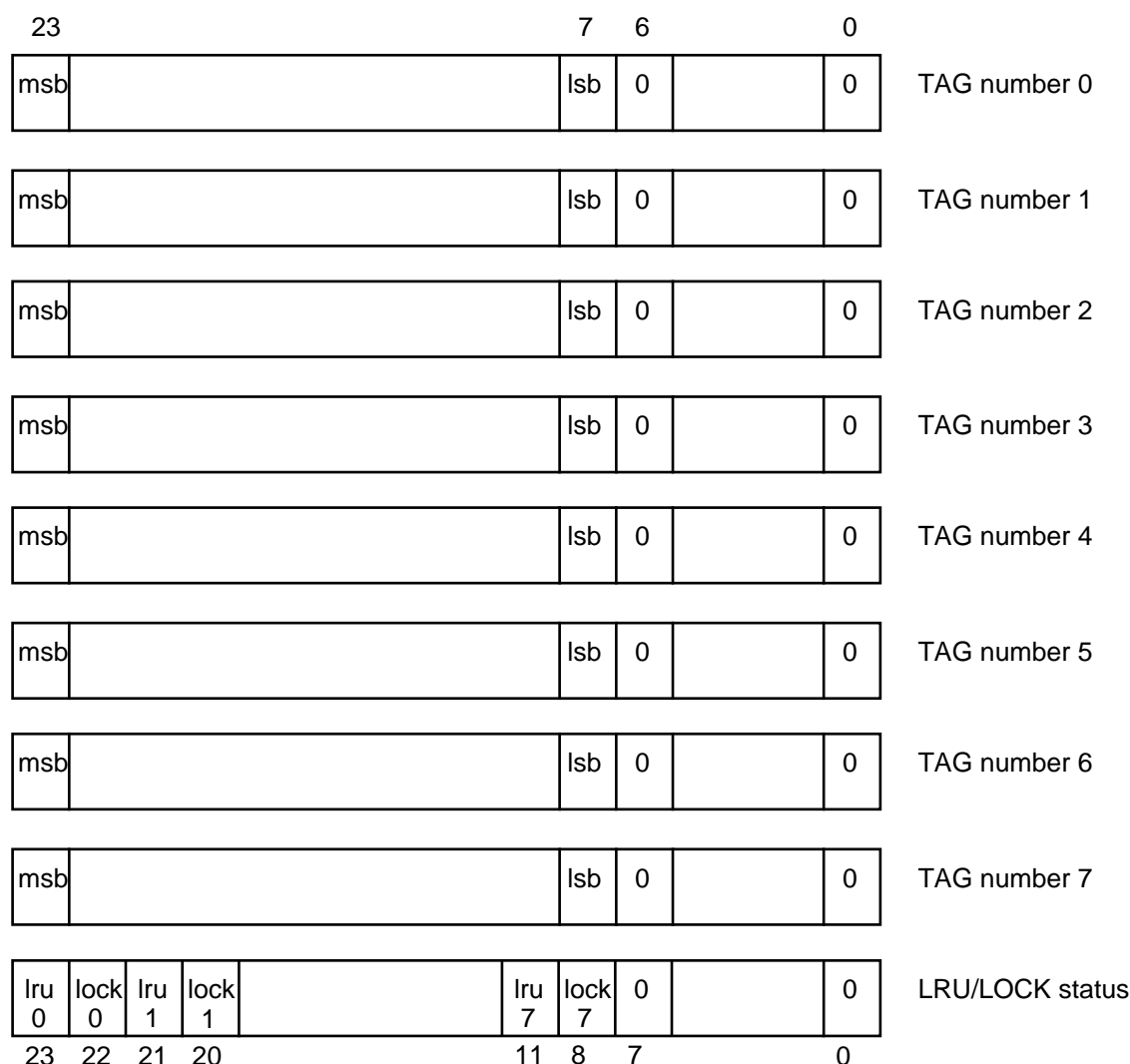
---

To keep track of the cache contents and status, the eight tags values, tags lock/unlock status and LRU status can be read via the OnCE™. Nine 24 bits registers are implemented as a circular buffer with a 4-bit counter. All these registers have the same address but any access to the tags buffer will cause the counter to increment thus pointing to the next register in the circular buffer. When exiting the Debug Mode the counter is cleared and therefore, when entering Debug Mode again, the first read from the tags buffer address will always start from the first register of the nine (tag number 0) and will continue circularly among these nine registers.

The registers mapping in the circular tags buffer is shown in Figure 10-8.

Determining the “next to be replaced sector”: In any time point at least one LRU bit in the “LRU/LOCK status” register will be set. But it is possible for more than one of the LRU bits to be set simultaneously. This is due to the fact that locked sectors could be “least recently used” although they can not be replaced. Therefore the “next to be replaced sector” is the only sector whose LRU bit is set and lock bit cleared. There is one exception to this rule and this is the case where all the 8 sectors are locked and LRU, in which case there is no “next to be replaced sector” since no sector will be replaced until at least one sector is unlocked.

**Figure 10-8. Circular Tags Buffer (TAGB)**



## 10.6 OnCE™ TRACE LOGIC

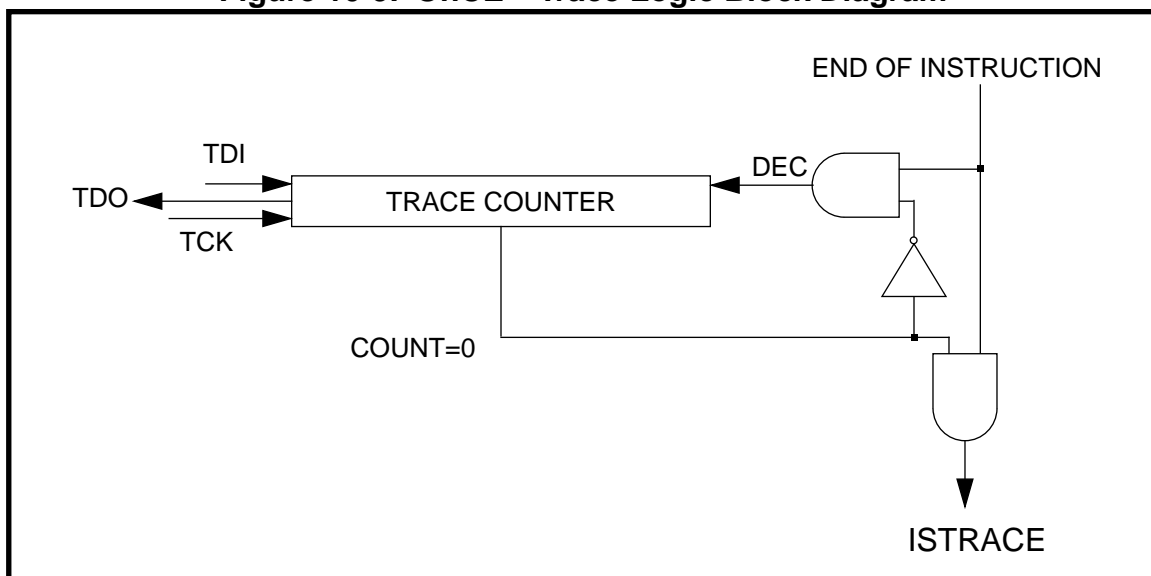
Using the OnCE™ Trace logic, execution of instructions in single or multiple steps is possible. The OnCE™ Trace logic causes the chip to enter the Debug Mode of operation after the execution of one or more instructions and wait for OnCE™ commands from the debug serial port. The OnCE™ Trace logic block diagram is shown in Figure 10-9.

The trace mode has a counter associated with it so that more than one instruction may be executed before returning back to the Debug Mode of operation. The objective of the counter is to allow the user to take multiple instruction steps real-time before entering the Debug Mode. This feature helps the software developer debug sections of code which do not have a normal flow or are getting hung up in infinite loops. The trace counter also enables the user to count the number of instructions executed in a code segment.

To enable the trace mode of operation 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 the Debug Mode by executing the appropriate command issued by the external command controller.

Upon exiting the Debug Mode, the counter is decremented after each execution of an instruction. Interrupts are serviceable and all instructions executed (including fast interrupt services and the execution of each repeated instruction) will decrement the trace counter. Upon decrementing to zero, the DSP56300 Core will re-enter the Debug Mode, the trace occurrence bit TO in the OSCR will be set, the Core Status bits OS1 and OS0 will be set to 11 and the  $\overline{DE}$  pin will be asserted to indicate that the DSP56300 Core has entered Debug Mode and is requesting service.

**Figure 10-9. OnCE™ Trace Logic Block Diagram**



### 10.6.1 Trace Counter (OTC)

The Trace Counter (OTC) is a 24-bit counter that can be read or written through the JTAG port. If N instructions are to be executed before entering the Debug Mode, the Trace Counter should be loaded with N-1. The Trace Counter is cleared by hardware reset.

## 10.7 METHODS OF ENTERING THE DEBUG MODE

Entering the Debug Mode is acknowledged by the chip by setting the Core Status bits OS1 and OS0 and asserting the  $\overline{DE}$  line. This informs the external command controller that the chip has entered the Debug Mode and is waiting for commands. The DSP56300 Core may disable the OnCE™ circuitry if the ROM Security option is implemented. If the ROM Security is implemented, the OnCE™ will remain inactive until a write operation to the OGDBR register is executed by the DSP56300 Core.

---

### 10.7.1 External Debug Request During $\overline{\text{RESET}}$

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. Note that in this case the chip does not execute any instruction before entering the Debug Mode.

### 10.7.2 External Debug Request During Normal Activity

Holding the  $\overline{\text{DE}}$  line asserted during normal chip activity causes the chip to finish the execution of the current instruction and then enter the Debug Mode. After receiving the acknowledge, the external command controller must negate the  $\overline{\text{DE}}$  line before sending the first command. Note that in this case the chip completes the execution of the current instruction and stops after the newly fetched instruction enters the instruction latch. This process is the same for any newly fetched instruction including instructions fetched by the interrupt processing or instructions that will be aborted by the interrupt processing.

### 10.7.3 Executing the JTAG `DEBUG_REQUEST` Instruction

Executing the JTAG instruction `DEBUG_REQUEST` will assert an internal debug request signal. Consequently, the chip will finish the execution of the current instruction and will stop after the newly fetched instruction enters the instruction latch. After entering the Debug Mode the Core Status bits `OS1` and `OS0` will be set and the  $\overline{\text{DE}}$  line will be asserted thus acknowledging the external command controller that the Debug Mode of operation has been entered.

### 10.7.4 External Debug Request During `STOP`

Executing the JTAG instruction `DEBUG_REQUEST` (or asserting  $\overline{\text{DE}}$ ) while the chip is in the `STOP` state (i. e., has executed a `STOP` instruction) causes the chip to exit the `STOP` state and enter the Debug Mode. After receiving the acknowledge, the external command controller must negate  $\overline{\text{DE}}$  before sending the first command. Note that in this case, the chip completes the execution of the `STOP` instruction and halts after the next instruction enters the instruction latch.

### 10.7.5 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 the Debug Mode. After receiving the acknowledge, the external command controller must negate  $\overline{\text{DE}}$  before sending the first command. Note that in this case, the chip completes the execution of the `WAIT` instruction and halts after the next instruction enters the instruction latch.

### 10.7.6 Software Request During Normal Activity

Upon executing the DSP56300 Core instruction `DEBUG` (or `DEBUGcc` when the specified condition is true), the chip enters the Debug Mode after the instruction following the

---

DEBUG instruction has entered the instruction latch.

### 10.7.7 Enabling Trace Mode

When the Trace Mode mechanism is enabled and the Trace Counter is greater than zero, the Trace Counter is decremented after each instruction execution. Execution of an instruction when the Trace Counter is zero will cause the chip to enter the Debug Mode after completing the execution of the instruction. Only instructions actually executed cause the Trace Counter to decrement, i.e. an aborted instruction will not decrement the Trace Counter and will not cause the chip to enter the Debug Mode.

### 10.7.8 Enabling Memory Breakpoints

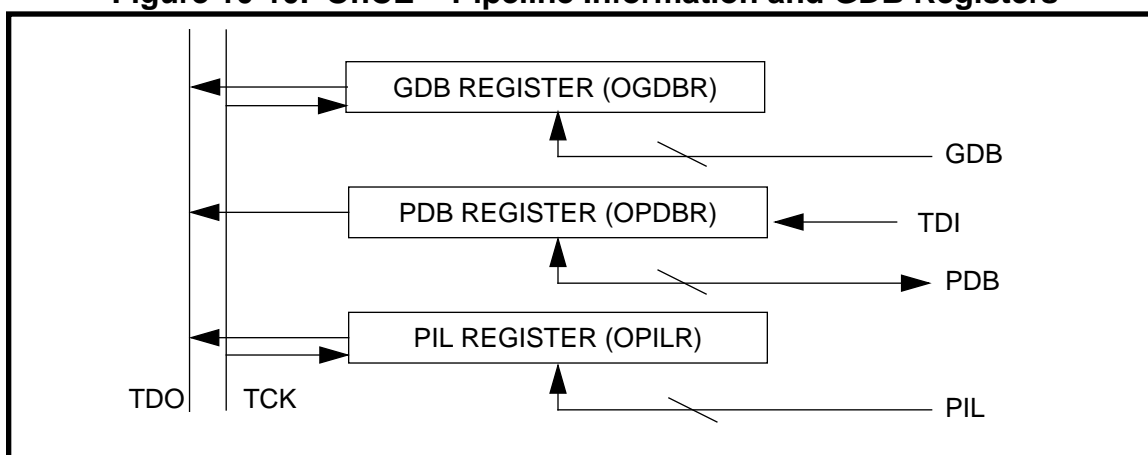
When the memory breakpoint mechanism is enabled with a Breakpoint Counter value of zero, the chip enters the Debug Mode after completing the execution of the instruction that caused the memory breakpoint to occur. In case of breakpoints on executed Program memory fetches, the breakpoint will be acknowledged immediately after the execution of the fetched instruction. In case of breakpoints on accesses to X, Y or P memory spaces by MOVE instructions, the breakpoint will be acknowledged after the completion of the instruction following the instruction that accessed the specified address.

## 10.8 PIPELINE INFORMATION AND GDB REGISTER

---

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 10-10 shows the block diagram of the Pipeline Information Registers with the exception of the PAB registers which are shown in Figure 10-11.

**Figure 10-10. OnCE™ Pipeline Information and GDB Registers**



### 10.8.1 PDB Register (OPDBR)

The PDB Register is a 24-bit latch that stores the value of the Program Data Bus

---



---

generated by the last program memory access of the core before the Debug Mode is entered. OPDBR can be 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.

### **10.8.2 PIL Register (OPILR)**

The PIL Register is a 24-bit latch that stores the value of the Instruction Latch before the Debug Mode is entered. OPILR can only be read through the JTAG port.

Note: Since the Instruction Latch is affected by the operations performed during the 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, the task of restoring is accomplished by writing to OPDBR with no-GO and no-EX. In this case the data written on PDB is transferred into the Instruction Latch

### **10.8.3 GDB Register (OGDBR)**

The GDB Register is a 24-bit latch that can only be read through the JTAG port. OGDBR is not actually required from a pipeline status restore point of view but is required as a means of passing information between the chip and the external command controller. OGDBR is mapped on the X internal I/O space at address \$FFFFFFC. Whenever the external command controller needs the contents of a register or memory location, it will force the chip to execute an instruction that brings that information to OGDBR. Then, the contents of OGDBR will be delivered serially to the external command controller by the command "READ GDB REGISTER".

## **10.9 TRACE BUFFER**

---

To ease debugging activity and keep track of program flow the DSP56300 Core provides a number of on-chip dedicated resources. There are three read-only PAB registers which give pipeline information when the Debug Mode is entered and a Trace Buffer which stores the address of the last instruction that was executed as well as the addresses of the last 12 change of flow instructions.

### **10.9.1 PAB Register for Fetch (OPABFR)**

The PAB Register for Fetch is a 24-bit register that stores the address of the last instruction whose fetch was started before the Debug Mode was entered. OPABFR can only be read through the JTAG port. This register is not affected by the operations performed during the Debug Mode.

### **10.9.2 PAB Register for Decode (OPABDR)**

The PAB Register for Decode is a 24-bit register that stores the address of the instruction

---

---

currently on the PDB. This is the instruction whose fetch was completed before the chip has entered the Debug Mode. OPABDR can only be read through the JTAG port. This register is not affected by the operations performed during the Debug Mode.

### **10.9.3 PAB Register for Execute (OPABEX)**

The PAB Register for Execute is a 24-bit register that stores the address of the instruction currently in the Instruction Latch. This is the instruction that would have been decoded and executed if the chip would not have entered the Debug Mode. OPABEX can only be read through the JTAG port. This register is not affected by the operations performed during the Debug Mode.

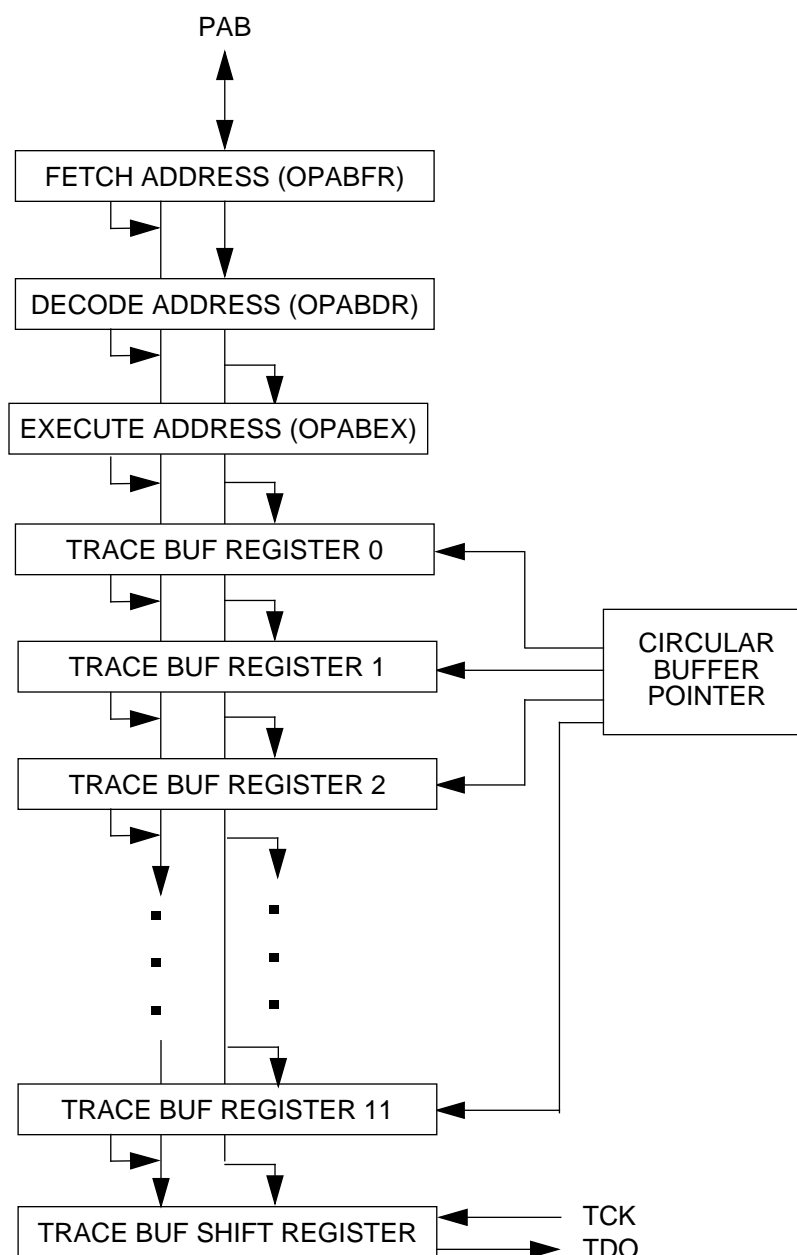
### **10.9.4 Trace Buffer**

The Trace Buffer stores the addresses of the last 12 change of flow instructions that were executed as well as the address of the last executed instruction. The Trace Buffer is implemented as a circular buffer containing 12 25-bit registers and one 4-bit counter. All the registers have the same address but any read access to the Trace Buffer address will cause 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 10-11 shows the block diagram of the Trace Buffer. The Trace Buffer is not affected by the operations performed during the Debug Mode except for the Trace Buffer pointer increment when reading the Trace Buffer. When entering the Debug Mode, the Trace Buffer counter will be pointing to the Trace Buffer register containing the address of the last executed instructions. The first Trace Buffer read will obtain the oldest address and the following Trace Buffer reads will get the other addresses from the oldest to the newest (the order of execution).

Note 1: To ensure Trace Buffer coherence, a complete set of 12 reads of the Trace Buffer must be performed. This is necessary due to the fact that each read increments the Trace Buffer pointer thus pointing to the next location. After 12 reads the pointer will point to the same location as before starting the read procedure.

Note 2: 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 (i.e. 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” will be set thus marking this instruction as “not taken”. Therefore it is imperative to read 25 bits of data when reading the 12 Trace Buffer registers. Since data is read lsb first, the “invalid bit” is the first bit to be read.

**Figure 10-11. OnCE™ Trace Buffer**



## 10.10 SERIAL PROTOCOL DESCRIPTION

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 has to wait for an acknowledge on the

---

$\overline{DE}$  line indicating that the chip has entered the Debug Mode (optionally the external command controller can poll the OS1, OS0 bits in JTAG instruction shift register). The external command controller communicates with the chip by sending 8-bit commands that may 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 may send a new command only after the chip has acknowledged execution of the previous command.

### **10.10.1 OnCE™ Commands**

The OnCE™ commands may be classified as follows:

- read commands (when the chip will deliver the required data).
- write commands (when the chip will receive data and write 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 10-4.

## **10.11 TARGET SITE DEBUG SYSTEM REQUIREMENTS**

---

A typical debug environment consists of a target system where the DSP56300 Core based device resides in the user defined hardware. The JTAG port interfaces to the external command controller over a 8-wire link consisting of the five JTAG wires, one OnCE™ wires, a ground and a reset wire. The reset wire is optional and is only used to reset the DSP56300 Core based device and its associated circuitry.

The external command controller acts as the medium between the DSP56300 Core target system and a host computer. The external command controller circuit acts as a JTAG port driver and host computer command interpreter. The controller issues commands based on the host computer inputs from a user interface program which communicates with the user.

### **10.12 EXAMPLES OF USING THE OnCE™**

---

Following are some examples of debugging procedures. Note that all the examples assume that the DSP is the only device in the JTAG chain. If there are more than one device in the chain (other DSP's or even other devices), the other devices can be forced to execute the JTAG BYPASS instruction such as their effect in the serial stream will be 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. Please refer to Chapter 11 for a detailed description of the JTAG protocol.

### **10.12.1 Checking whether the chip has entered the Debug Mode**

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

1. Every time the chip enters the Debug Mode, a pulse is generated on the  $\overline{DE}$  line. A pulse will also be generated every time the chip acknowledges the execution of an instruction while in Debug Mode. An external command controller can connect the  $\overline{DE}$  line to an interrupt pin in order to sense the acknowledge.
2. 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.

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

### **10.12.2 Polling the JTAG instruction shift register**

In order to poll the core status bits in the JTAG instruction shift 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 the Debug Mode.

### **10.12.3 Saving Pipeline Information**

The debugging activity is accomplished by means of DSP56300 Core instructions supplied from the external command controller. Therefore the current state of the DSP56300 Core pipeline must be saved prior to starting the debug activity and of course the state must be restored prior to returning to the Normal Mode of operation. Following is the description of the saving procedure (assume that ENABLE\_ONCE has been executed and Debug Mode has been entered and verified as described in 10.12.1):

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.

Note that there is no need to verify acknowledge between steps 1 and 2 as well as 3 and 4 because completion is guaranteed by design.

#### **10.12.4 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. Following is the description of the read Trace Buffer procedure (assume that all actions described in 10.12.3 have been 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).

Note that the user must read the entire FIFO since each read will increment the FIFO pointer thus pointing to the next FIFO location. At the end of this procedure the FIFO pointer will point back to the beginning of the FIFO.

The information that has been read by the external command controller contains now 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 as well as the addresses of the last 12 instructions that have been executed and are change of flow. 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.

#### **10.12.5 Displaying a specified register**

The DSP56300 must be in Debug Mode and all actions described in 10.12.3 have been executed. 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 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 will select OGDBR as the data register for read).
  5. Select shift-DR. Shift out the OGDBR contents. Pass through update-DR. Wait for next command.

#### 10.12.6 Displaying X memory area starting at address \$xxxxxx

The DSP56300 must be in Debug Mode and all actions described in 10.12.3 have been executed. Since R0 will be used as pointer for the memory, R0 will be first saved. 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 will select 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 will select 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 will execute again 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.

#### **10.12.7 Returning from Debug Mode to Normal Mode to current program**

This is the case in which the user has finished examining the current state of the machine, changed some of the registers and wishes to return and continue execution of its program from the point where it stopped. Therefore the user has to restore the pipeline of the machine and enable normal instruction execution. The sequence of actions is:

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 bit 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 bit 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.

#### **10.12.8 Returning from Debug Mode to Normal Mode to a new program**

This is the case in which the user has finished examining the current state of the machine, changed some of the registers and wishes to start the execution of a new program (the GOTO command). Therefore the user has to force a "change of flow" to the starting address of the new program (\$xxxxxx). The sequence of actions is:

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 bit "\$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 bit 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.

Note that if the entering of the Debug Mode happened during a DO LOOP, REP instruction or other special cases like interrupt processing, STOP, WAIT, conditional branching etc. it is mandatory that the user will first reset the DSP56300 and only afterwards proceed with the execution of the new program.



---

## 10.13 EXAMPLES OF JTAG-OnCE INTERACTION

---

This paragraph exemplifies the details of the JTAG-OnCE™ interaction by describing the TMS sequencing required in order to achieve the communication depicted in the Paragraph 10.12.

The external command controller can force the DSP56300 into Debug Mode by executing the JTAG instruction `DEBUG_REQUEST`. In order to check that the DSP56300 has entered the Debug Mode the external command controller must poll the status by reading the `OS1,OS0` bits in the JTAG instruction shift register. The TMS sequencing is depicted in Table 10-9.

The sequencing of enabling the OnCE™ is described in Table 10-10.

After executing the JTAG instructions `DEBUG_REQUEST` and `ENABLE_ONCE` and after the core status was polled to verify that the chip is in Debug Mode, the pipeline saving procedure must take place. The TMS sequencing for this procedure is depicted in Table 10-9.

**Table 10-9. 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
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 com- mand controller to poll the status
.....				
n	0	Run-Test/Idle	Idle	

In “step n” the external command controller verifies that the OS1-OS0 have the value 11 indicating that the chip has entered the Debug Mode. If the chip has not yet entered the Debug Mode the external command controller will go to “step b”, “step c” etc. until the Debug Mode is acknowledged.

---

**Table 10-10. 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
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
.....				
1	0	Run-Test/Idle	Idle	

**Table 10-11. TMS Sequencing for reading pipeline registers**

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	
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	

step	TMS	JTAG	OnCE™	Note
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.

