7 PROCESSING STATES

The DSP56300 Core is always in one of five processing states: normal, exception, reset, wait, or stop. These states are described in the following paragraphs.

7.1 NORMAL PROCESSING STATE

The normal processing state is associated with instruction execution. Instructions are executed using a seven-stage pipeline, which is described in the following paragraphs.

7.1.1 Instruction Pipeline

DSP56300 Core instruction execution is performed using a seven-stage pipeline, allowing most instructions to execute at a rate of one instruction every clock cycle. However, certain instructions require additional time to execute: all of the double-word instructions, instructions using an addressing mode that requires more than one cycle for the address calculation, and instructions causing a change of flow.

Instruction pipelining allows overlapping of instruction execution so that a pipeline stage of a given instruction occurs concurrently with other pipeline stages of other instructions. Only one word is fetched per cycle, so that in the case of double-word instructions, the second word of an instruction will be fetched before the next instruction is fetched. Table 7-1 describes the DSP56300 Core pipeline. The pipeline consists of seven stages: fetch 1, fetch 2, decode, address generation 1, address generation 2, execute1 and execute 2. n1 and n2 refer to first and second instructions respectively. The third instruction (n3), which contains an instruction extension word (n3e) takes two clock cycles to execute. The extension word will be either an absolute address or immediate data. Although it takes seven clock cycles for the pipeline to fill and the first instruction to execute, further instructions usually completes on each clock cycle.

Each instruction requires a minimum of seven clock cycles to be fetched, decoded, and executed. This means that there is a delay of seven clock cycles on power-up to fill the pipeline. A new instruction may begin immediately following the previous instruction. Two-word instructions require a minimum of eight clock cycles to execute (seven cycles for the first instruction word to move through the pipe and execute and one more cycle for the second word to execute). For a complete description of the execution timing of the various instructions, addressing modes etc., see Appendix B - INSTRUCTION EXECUTION TIMING.

Table 7-1. Instruction Pipeline

Operation	Instruction Cycle										
	1	2	3	4	5	6	7	8	9	10	11
PreFetch 1	n1	n2	n3	n3e	n4	n5	n6	n7	n8	n9	n10
PreFetch 2		n1	n2	n3	n3e	n4	n5	n6	n7	n8	n9
Decode			n1	n2	n3	n3e	n4	n5	n6	n7	n8
Address Gen 1				n1	n2	n3	n3e	n4	n5	n6	n7
Address Gen 2					n1	n2	n3	n3e	n4	n5	n6
Execute 1						n1	n2	n3	n3e	n4	n5
Execute 2							n1	n2	n3	n3e	n4

7.2 EXCEPTION PROCESSING STATE (INTERRUPT PROCESSING)

The exception processing state is associated with interrupts that can be generated by conditions inside the DSP or from external sources. There are many sources for interrupts to the DSP56300 Core; some of these sources can generate more than one interrupt. An interrupt vector scheme with 128 vectors of predefined priorities is used to provide fast interrupt service. The following list outlines how interrupts are processed by the DSP56300 Core:

1. A hardware interrupt is synchronized with the DSP56300 Core clock, and the interrupt pending flag for that particular hardware interrupt is set. An interrupt source can have only one interrupt pending at any given time.
2. All pending interrupts (external and internal) are arbitrated to select which interrupt will be processed. The arbiter automatically ignores any interrupts with an Interrupt Priority Level (IPL) lower than the interrupt mask level in the SR and selects the remaining interrupt with the highest IPL.
3. The interrupt controller then freezes the program counter (PC) and fetches two instructions at the two interrupt vector addresses associated with the selected interrupt.
4. The interrupt controller inserts the two instructions into the instruction stream and releases the PC, which is used for the next instruction fetch. The next interrupt arbitration then begins.

If neither of the two instructions is a Jump To Subroutine (JSR) instruction (e.g. a JSCLR), the state of the machine is not saved on the stack, and a fast interrupt is executed. A long interrupt is executed if one of the interrupt instructions fetched is a JSR instruction. The PC is immediately released, the SR and the PC are saved in the stack, and the jump instruction controls where the next instruction is fetched from.

Note: All the various types of Jump To Subroutine instructions may be used as the “JSR” needed to make the interrupt long, e.g. JScc, BSSET etc.

In digital signal processing, one of the main uses of interrupts is to transfer data between DSP memory or registers and a peripheral device. When such an interrupt occurs, a limited context switch with minimum overhead is often desirable. This limited context switch is accomplished by a fast interrupt. The long interrupt is used when a more complex task must be accomplished to service the interrupt.

7.2.1 Interrupt Sources

Exceptions may originate from any of the 128 vector addresses listed in Table 7-2. Exceptions may originate from one of two groups: core and peripherals. Table 7-2 lists only the core-originating sources. The peripheral-originating sources are described in the specific chip configuration specification document. The corresponding interrupt starting address for each interrupt source is shown. These addresses are located in the 256 locations of program memory pointed to by the VBA (Vector Base Address) register in the program control unit. When an interrupt is serviced, the instruction at the interrupt starting address is fetched first. Because the program flow is directed to a different starting address for each interrupt, the interrupt structure of the DSP56300 Core is said to be vectored. A vectored interrupt structure has low overhead execution. If it is known a-priori that certain interrupts will not be used, those interrupt vector locations can be used for program or data storage.

Table 7-2. Interrupt Sources

Interrupt Starting Address	IPL	Interrupt Source
VBA:$00	-	Reserved
VBA:$02	3	Stack Error
VBA:$04	3	Illegal Instruction
VBA:$06	3	Debug Request Interrupt
VBA:$08	3	Trap
VBA:$0A	3	Non-Maskable Interrupt ($\overline{\text{NMI}}$)
VBA:$0C	3	Reserved for Future Level-3 Interrupt Source
VBA:$0E	3	Reserved for Future Level-3 Interrupt Source
VBA:$10	0 - 2	$\overline{\text{IRQA}}$
VBA:$12	0 - 2	$\overline{\text{IRQB}}$
VBA:$14	0 - 2	$\overline{\text{IRQC}}$
VBA:$16	0 - 2	$\overline{\text{IRQD}}$
VBA:$18	0 - 2	DMA Channel 0
VBA:$1A	0 - 2	DMA Channel 1
VBA:$1C	0 - 2	DMA Channel 2
VBA:$1E	0 - 2	DMA Channel 3
VBA:$20	0 - 2	DMA Channel 4
VBA:$22	0 - 2	DMA Channel 5
VBA:$24	0 - 2	Peripheral interrupt request 1
VBA:$26	0 - 2	Peripheral interrupt request 2
:	:	
VBA:$FE	0 - 2	Peripheral interrupt request 110

The 128 interrupts are prioritized into four levels. Level 3, the highest priority level, is not maskable. Levels 0 – 2 are maskable. The interrupts within each level are prioritized according to a predefined priority.

7.2.1.1 Hardware Interrupt Source

There are two types of hardware interrupts to the DSP56300 Core: internal and external. The internal interrupts include the on-chip sources, namely: Stack Error, Illegal Instruction, Debug Request, Trap, the DMAs and the peripherals. Each internal interrupt source is serviced if it is not masked. When serviced, the interrupt request is cleared. Each maskable internal hardware source has independent enable control.

The external hardware interrupts include $\overline{\text{NMI}}$, $\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, $\overline{\text{IRQC}}$ and $\overline{\text{IRQD}}$. The $\overline{\text{NMI}}$ interrupt is an edge triggered non-maskable interrupt that can be used for software development, watch-dog, power fail detect etc. The $\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, $\overline{\text{IRQC}}$ and $\overline{\text{IRQD}}$ interrupts can be programmed to be level sensitive or edge triggered. Since the level-sensitive interrupts will not be cleared automatically when they are serviced, they must be cleared by other means to prevent multiple interrupts, usually by an external hardware that will detect the acknowledge of the core to the interrupt request. The edge-sensitive interrupts are latched as pending on the high-to-low transition of the interrupt input and are automatically cleared when the interrupt is serviced. $\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, $\overline{\text{IRQC}}$ and $\overline{\text{IRQD}}$ can be programmed to one of three priority levels: 0, 1, or 2, all of which are maskable. Additionally, these interrupts have independent enable control.

When the $\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, $\overline{\text{IRQC}}$ and $\overline{\text{IRQD}}$ interrupts are disabled in the interrupt priority register, the pending request will be ignored, regardless of whether the interrupt input was defined as level sensitive or edge sensitive. Additionally, if the interrupt is defined as edge sensitive, its edge-detection latch will remain in the reset state as long as the interrupt is disabled; if the interrupt is defined as level sensitive, its edge-detection latch will remain in the reset state. If the level-sensitive interrupt is disabled while the interrupt is pending, the pending interrupt will be cancelled. However, if the interrupt has been fetched, it normally will not be cancelled.

Note: On all external, level-sensitive interrupt sources, the interrupt should be serviced (i.e. clear the source for the interrupt) by the instruction at Vector location, if it is a fast interrupt, or by a long interrupt.

7.2.1.2 Software Interrupt Source

There are two software interrupt sources — Illegal Instruction Interrupt (III) and TRAP.

The III is a nonmaskable interrupt (IPL 3), which is serviced immediately following the execution of the illegal instruction or the attempted execution of an illegal instruction (any undefined operation code).

TRAP is a nonmaskable interrupt (IPL 3), which is serviced immediately following the TRAP or TRAPcc (condition true) instruction execution.

7.2.2 Interrupt Priority Structure

Four levels of interrupt priority are provided. IPLs numbered 0, 1, and 2 are maskable (level 0 is the lowest level). Level 3 (highest level) is nonmaskable. The IPL 3 interrupts are: Hardware Reset, III, Stack Error and TRAP. The interrupt mask bits (I1, I0) in the SR reflect the current processor priority level and indicate the IPL needed for an interrupt source to interrupt the processor (see Table 7-3). Interrupts are inhibited for all priority levels less than the current processor priority level. However, level 3 interrupts are not maskable and therefore can always interrupt the processor.

Table 7-3. Status Register Interrupt Mask Bits

I1	I0	Exceptions Permitted	Exceptions Masked
0	0	IPL 0, 1, 2, 3	None
0	1	IPL 1, 2, 3	IPL 0
1	0	IPL 2, 3	IPL 0, 1
1	1	IPL 3	IPL 0, 1, 2

7.2.2.1 Interrupt Priority Levels

There are two Interrupt priority registers in the DSP56300 Core: IPRC that is dedicated for DSP56300 Core interrupt sources and IPRP that is dedicated for the specific chip peripherals interrupt sources. These control registers are mapped on the internal X I/O memory space.

The IPL for each interrupting source is software programmable. Each on-chip or external peripheral device can be programmed to one of the three maskable priority levels (IPL 0, 1, or 2). IPLs are set by writing to the interrupt priority registers shown in Figure 7-1 and Figure 7-2. These two read/write registers specify the IPL for each of the interrupting devices. In addition, IPRC register specifies the trigger mode of each external interrupt source and is used to enable or disable the individual external interrupts. These registers are cleared on hardware RESET or by the RESET instruction. Table 7-4 defines the IPL bits. Table 7-5 defines the external interrupt trigger mode bits.

Figure 7-1. Interrupt Priority Register C (IPRC)

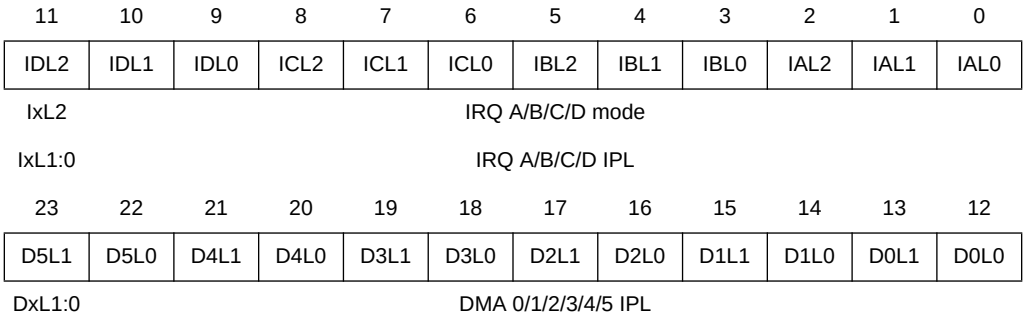

Figure 7-2. Interrupt Priority Register P (IPRP)

11	10	9	8	7	6	5	4	3	2	1	0
Per6L1	Per6L0	Per5L1	Per5L0	Per4L1	Per4L0	Per3L1	Per3L0	Per2L1	Per2L0	Per1L1	Per1L0

23	22	21	20	19	18	17	16	15	14	13	12
PerCL1	PerCL0	PerBL1	PerBL0	PerAL1	PerAL0	Per9L1	Per9L0	Per8L1	Per8L0	Per7L1	Per7L0

Table 7-4. Interrupt Priority Level Bits

xxL1	xxL0	Enabled	IPL
0	0	No	—
0	1	Yes	0
1	0	Yes	1
1	1	Yes	2

Table 7-5. External Interrupt Trigger Mode Bits

IxL2	Trigger Mode
0	Level
1	Negative Edge

7.2.2.2 Exception Priorities within an IPL

If more than one exception is pending when an instruction is executed, the interrupt with the highest priority level is serviced first. When multiple interrupt requests having the same IPL are pending, a second fixed-priority structure within that IPL determines which interrupt is serviced. The fixed priority of interrupts within an IPL and the interrupt enable bits for all interrupts are shown in Table 7-6.

7.2.3 Instructions Preceding the Interrupt Instruction Fetch

Every instruction which takes more than one cycle to execute is aborted when it is fetched in the cycle preceding the fetch of the first interrupt instruction word.

Aborted instructions are refetched again when program control returns from the interrupt routine. The PC is adjusted appropriately before the end of the decode cycle of the aborted instruction.

If the first interrupt word fetch occurs in the cycle following the fetch of a one-word-one-

cycle instruction, that instruction will complete normally before the start of the interrupt routine.

During an interrupt instruction fetch, two instruction words are fetched — the first from the interrupt starting address and the second from the interrupt starting address +1 locations.

7.2.4 Interrupt Types

Two types of interrupt routines may be used: fast and long. The fast routine consists of the two automatically inserted interrupt instruction words. These words can contain any unrestricted, single two-word instruction or any two unrestricted one-word instructions. Fast interrupt routines are never interruptible.

CAUTION

Status is not preserved during a fast interrupt routine; therefore, instructions that modify status should not be used at the interrupt starting address and interrupt starting address+1.

If one of the instructions in the fast routine is a JSR, then a long interrupt routine is formed. The following actions occur during execution of the JSR instruction when it occurs in the interrupt starting address or in the interrupt starting address +1:

1. The PC (containing the return address) and the SR are stacked.
2. The loop flag is reset.
3. The scaling mode bits are reset.
4. The sixteen-bit mode bit is reset.
5. The IPL is raised to disallow further interrupts of the same or lower levels (except Illegal Instruction, stack error and TRAP that can always interrupt).

The long interrupt routine should be terminated by an RTI. Long interrupt routines are interruptible by higher priority interrupts.

7.2.5 Interrupt Arbitration

External interrupts are internally synchronized with the processor clock before their interrupt-pending flags are set. Each external interrupt and internal interrupt has its own flag. After each instruction is executed, all interrupts are arbitrated — i.e., all hardware interrupts that have been latched into their respective interrupt-pending flags and all internal interrupts. During arbitration, each interrupt's IPL is compared with the interrupt mask in the SR, and the interrupt is either allowed or disallowed. The remaining interrupts are prioritized according to the priority shown in Table 7-6, and the highest priority interrupt is chosen. The interrupt vector is then calculated so that the program interrupt controller can fetch the first interrupt instruction. The interrupt-pending flag for the chosen interrupt is not cleared until the second interrupt vector of the chosen interrupt is being fetched. A new interrupt from the same source will not be accepted for the next interrupt arbitration until that time.

Table 7-6. Exception Priorities within an IPL

Priority	Exception
Level 3 (Nonmaskable)	
Highest	Stack Error
	Illegal Instruction
	Debug Request Interrupt
	Trap
	Non-Maskable Interrupt ($\overline{\text{NMI}}$)
Lowest	Non-Maskable Peripheral Interrupt
Levels 0, 1, 2 (Maskable)	
Highest	$\overline{\text{IRQA}}$ (External Interrupt)
	$\overline{\text{IRQB}}$ (External Interrupt)
	$\overline{\text{IRQC}}$ (External Interrupt)
	$\overline{\text{IRQD}}$ (External Interrupt)
	DMA Channel 0 Interrupt
	DMA Channel 1 Interrupt
	DMA Channel 2 Interrupt
	DMA Channel 3 Interrupt
	DMA Channel 4 Interrupt
	DMA Channel 5 Interrupt
Lowest	Peripheral interrupt sources

7.2.6 Interrupt Instruction Fetch

The interrupt controller generates an interrupt instruction fetch address, which points to the first instruction word of a two-word interrupt routine. This address is used for the next instruction fetch, instead of the contents of the PC, and the interrupt instruction fetch address +1 is used for the subsequent instruction fetch. While the interrupt instructions are being fetched, the PC is inhibited from being updated. After the two interrupt words have been fetched, the PC is used for any subsequent instruction fetches.

7.2.7 Interrupt Instruction Execution

Interrupt instruction execution is considered “fast” if neither of the instructions of the interrupt service routine cause a change of flow. A JSR within a fast interrupt routine forms a long interrupt, which is terminated with an RTI instruction to restore the PC and SR from the stack and return to normal program execution. Reset is a special exception, which will normally contain only a JMP instruction at the exception start address. At the programmer’s option, almost any instruction can be used in the fast interrupt routine. A fast interrupt routine may contain either two single-word instructions or one double-word instruction. Table 7-7 shows the effect of a fast interrupt routine on the instruction pipeline. The fast interrupt executes only two instructions (ii1 and ii2) and then automatically resumes execution of the main program. Table 7-8 shows the effect of a long interrupt routine on the instruction pipeline. A short JSR (ii1) is used to call the long interrupt routine which includes the 4 instructions sr1, sr2, sr3 and an rti. Instructions ii2, n3, sr5 and sr6 are not decoded nor executed.

Table 7-7. Fast Interrupt Pipeline

Operation	Instruction Cycle											
	1	2	3	4	5	6	7	8	9	10	11	12
PreFetch 1	n1	n2	ii1	ii2	n3	n4						
PreFetch 2		n1	n2	ii1	ii2	n3	n4					
Decode			n1	n2	ii1	ii2	n3	n4				
Address Gen 1				n1	n2	ii1	ii2	n3	n4			
Address Gen 2					n1	n2	ii1	ii2	n3	n4		
Execute 1						n1	n2	ii1	ii2	n3	n4	
Execute 2							n1	n2	ii1	ii2	n3	n4
n = normal instruction word ii = interrupt instruction word												

Table 7-8. Long Interrupt Pipeline

Operation	Instruction Cycle															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
PreFetch 1	n1	n2	ii1	ii2	n3	sr1	sr2	sr3	rti	sr5	sr6	n3	n4	n5	n6	n7
PreFetch 2		n1	n2	jsr	ii2	n3	sr1	sr2	sr3	rti	sr5	sr6	n3	n4	n5	n6
Decode			n1	n2	jsr	-	-	sr1	sr2	sr3	rti	-	-	n3	n4	n5
Address Gen 1				n1	n2	jsr	-	-	sr1	sr2	sr3	rti	-	-	n3	n4
Address Gen 2					n1	n2	jsr	-	-	sr1	sr2	sr3	rti	-	-	n3
Execute 1						n1	n2	jsr	-	-	sr1	sr2	sr3	rti	-	-
Execute 2							n1	n2	jsr	-	-	sr1	sr2	sr3	rti	-

n = normal instruction word
ii = interrupt instruction word
sr = service routine word

Execution of a fast interrupt routine always conforms to the following rules:

1. The processor status is not saved.
2. The fast interrupt routine may modify the status of the normal instruction stream e.g. use DO instruction, but such instructions should not be used in order to assure proper operation.
3. The PC, which contains the address of the next instruction to be executed in normal processing, remains unchanged during a fast interrupt routine.
4. The fast interrupt returns without an RTI.
5. Normal instruction fetching resumes using the PC following the completion of the fast interrupt routine.
6. A fast interrupt is not interruptible.
7. A JSR instruction within the fast interrupt routine forms a long interrupt routine.

Execution of a long interrupt routine always adheres to the following rules:

1. A JSR to the starting address of the interrupt service routine is located at one of the two interrupt vector addresses.
2. During execution of the JSR instruction, the PC and SR are stacked. The interrupt mask bits of the SR are updated to mask interrupts of the same or lower priority. The loop flag and scaling mode bits are cleared.
3. The interrupt service routine can be interrupted — i.e., nested interrupts are supported.
4. The long interrupt routine, which can be any length, should be terminated by an RTI, which restores the PC and SR from the stack.

Either one of the two instructions of the fast interrupt can be the JSR instruction that forms the long interrupt.

A REP instruction is treated as a single two-word instruction, regardless of how many times it repeats the second instruction of the pair. Instruction fetches are suspended and will be reactivated only after the LC is decremented to one. During the execution of the repeated instruction, no interrupts will be serviced. When LC finally decrements to one, the fetches are reinitiated, and pending interrupts can be serviced.

7.3 RESET PROCESSING STATE

The reset processing state is entered when the external RESET pin is asserted (a hardware reset). Upon entering the reset state:

1. Internal peripheral devices are reset.
2. The modifier registers (M0-M7) are set to $FFFFFF.
3. The interrupt priority registers are cleared.
4. The Bus Control Register (BCR), the Address Attribute Registers (AAR3-AAR0) and the Dram Control Register (DCR) are set to their initial values as described in Chapter 2. The initial value causes a maximum number of wait states to be added to every external memory access.
5. The Stack Pointer (SP) and the Stack Counter (SC) are cleared.
6. The scaling mode, loop flag, sixteen-bit mode, double precision mode and condition code bits of the SR are cleared, and the interrupt mask bits of the SR are set.
7. The Instruction Cache Controller is initialized as described in Chapter 5.
8. The cache-enable (CE) bit in SR and the burst-mode bit in OMR are cleared.
9. The PLL Control register is initialized as described in Chapter 9.
10. The Vector Base Address (VBA) register is cleared.

The DSP56300 Core remains in the reset state until RESET is deasserted. Upon leaving the reset state, the chip operating mode bits of the OMR are loaded from the external mode select pins (MODA, MODB, MODC,MODD), and program execution begins at the program memory address as described in Chapter 12.

7.4 WAIT PROCESSING STATE

The wait processing state is a low power-consumption state entered by execution of the WAIT instruction. In the wait state, the internal clock is disabled from all internal circuitry except the internal peripherals. All internal processing is halted until an unmasked interrupt occurs, the DSP is reset, or $\overline{DE}$ is asserted. If exit from wait state was caused by asserting $\overline{DE}$, the processor will enter the debug mode.

7.5 STOP PROCESSING STATE

The stop processing state is the lowest power consumption mode and is entered by the execution of the STOP instruction. In the stop mode, the clock oscillator activity depends on the PSTP bit in the PLL control register. If this bit is cleared, the clock oscillator is turned off, while if the bit is set, the VCO remains active and the global clock to the entire chip is gated off.

All activity in the processor is halted until one of the following actions occurs:

1. A low level is applied to the $\overline{\text{IRQA}}$ pin ($\overline{\text{IRQA}}$ asserted)
2. A low level is applied to the $\overline{\text{RESET}}$ pin ($\overline{\text{RESET}}$ asserted)
3. A low level is applied to the $\overline{\text{DE}}$ pin.

Either of these actions will gate on the oscillator and, after a clock stabilization delay, clocks to the processor and peripherals will be re-enabled.

When the clocks to the processor and peripherals are re-enabled then the processor will enter the reset processing state if the exit from stop state was caused by a low level on the $\overline{\text{RESET}}$ pin.

If the exit from stop state was caused by a low level on the $\overline{\text{IRQA}}$ pin then the processor will service the highest priority pending interrupt. If no interrupt is pending (i. e. $\overline{\text{IRQA}}$ was negated before interrupts were arbitrated) or if no interrupt is enabled then the processor resumes execution at the instruction following the STOP instruction that caused the entry into the stop state.

If the exit from stop state was caused by a low level on the $\overline{\text{DE}}$ pin, the processor will enter the debug mode.

For minimum power consumption during the STOP state at the cost of longer recovery time, PSTP bit of the PLL Control register should be cleared. To enable rapid recovery when exiting the STOP state, at the cost of higher power consumption, PSTP should be set. PSTP is cleared by hardware reset.