Module Introduction

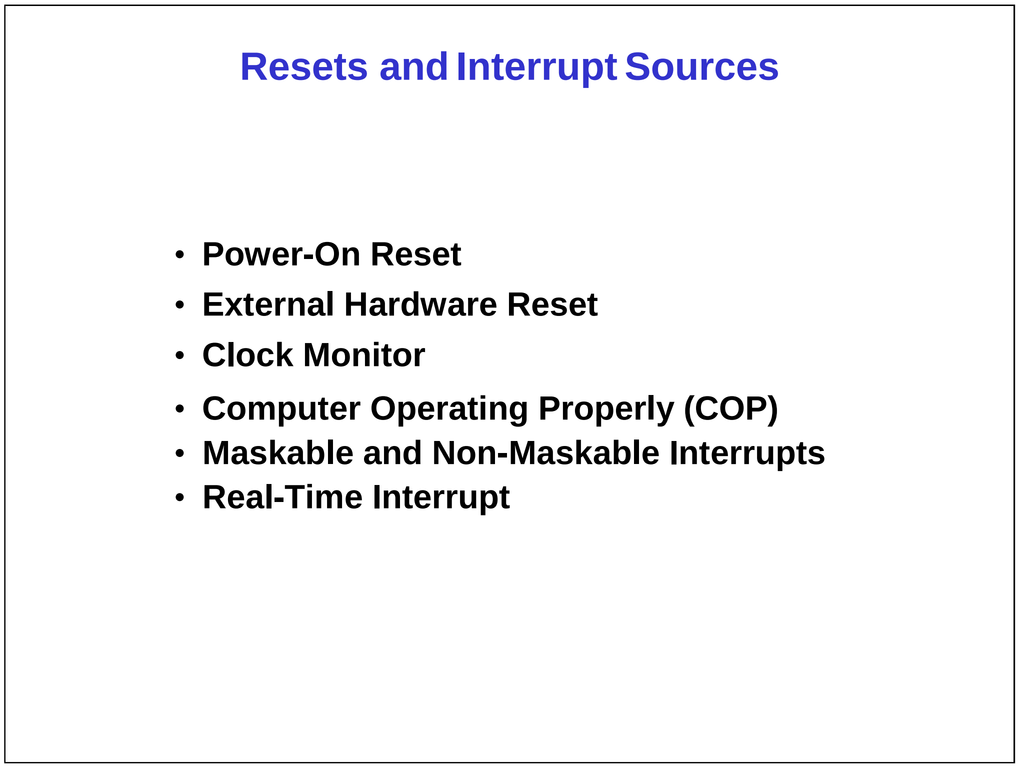
PURPOSE:
The intent of this module is to explain MCU processing of reset and interrupt exception events.

OBJECTIVES:
- Describe the difference between resets and interrupts.
- Identify different sources of resets and interrupts.
- Describe the MCU reset recovery process.
- Identify the steps to configure and service an interrupt event.
- Describe MCU exception processing.

CONTENT:
- 24 pages
- 4 questions

LEARNING TIME:
- 50 minutes

The intent of this module is to explain MCU processing of reset and interrupt exception events. You will learn about the different sources of reset and interrupt events. You will also learn about MCU exception processing. When you have completed this module, you will be able to describe the differences between resets and interrupts, identify different sources of reset and interrupt events, and describe MCU exception processing.



Resets and Interrupt Sources

- **Power-On Reset**
- **External Hardware Reset**
- **Clock Monitor**
- **Computer Operating Properly (COP)**
- **Maskable and Non-Maskable Interrupts**
- **Real-Time Interrupt**

Resets and interrupts are responses to exceptional events during program execution. Six reset and interrupt sources include: Power-On Reset, External Hardware Reset, Clock Monitor, Computer Operating Properly (COP), Maskable and Non-Maskable Interrupts, and Real-Time Interrupts.

Resets can be caused by Power-On condition, assertion of the external reset pin, or by an internal reset signal. Internal reset signals can be generated by the COP module or the Clock Monitor module.

Any of the reset conditions will abort main program execution and cause the Microcontroller Unit (MCU) to vector to one of three reset vectors, depending on the cause of reset. Power-On Reset and External Reset share the same vector, at location $FFFE. Clock Monitor and COP Resets each have a unique vector.

Once the reset is processed, the MCU immediately returns to a known start-up condition. The MCU then starts program execution from a user-defined memory location, which is pointed to by the appropriate vector.

Power-On Reset

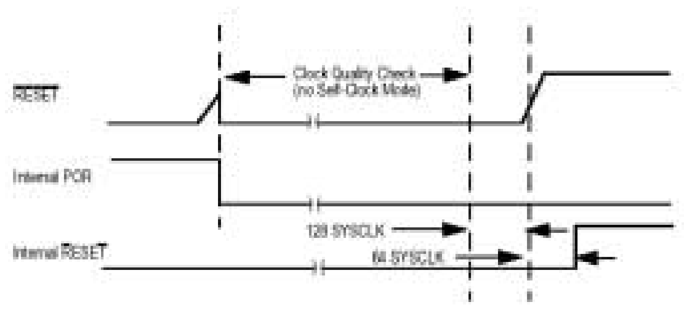
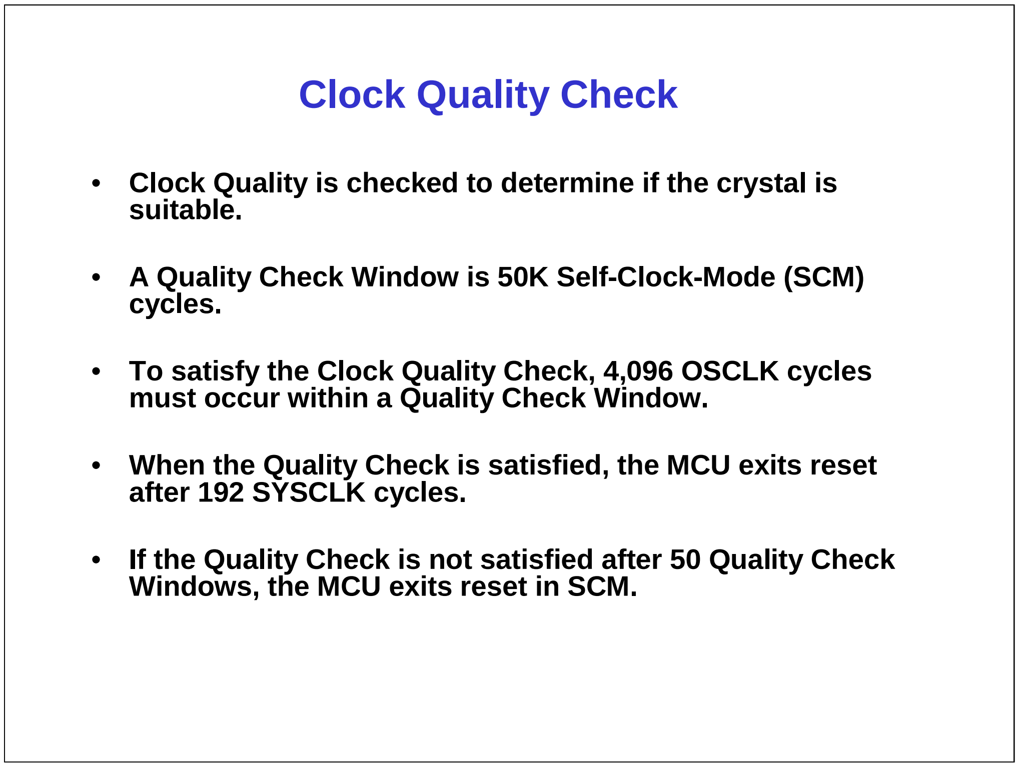


Figure 5-2 $\overline{\text{RESET}}$ pin tied to VDD (by a pull-up resistor)

Let’s take a look at Power-On Reset in more detail. Power-On Reset is initiated by positive transition on VDD. A Clock Quality Check begins after the Power-On-Reset (POR) trigger. The reset pin is tied to the VDD by a pull-up resistor.

In general, subsystems and control bits are initialized to have the least effect on the system. For example, interrupts are masked, ports are read only, timers and serial communication ports are disabled.



Clock Quality Check

- **Clock Quality is checked to determine if the crystal is suitable.**
- **A Quality Check Window is 50K Self-Clock-Mode (SCM) cycles.**
- **To satisfy the Clock Quality Check, 4,096 OSCLK cycles must occur within a Quality Check Window.**
- **When the Quality Check is satisfied, the MCU exits reset after 192 SYSCLK cycles.**
- **If the Quality Check is not satisfied after 50 Quality Check Windows, the MCU exits reset in SCM.**

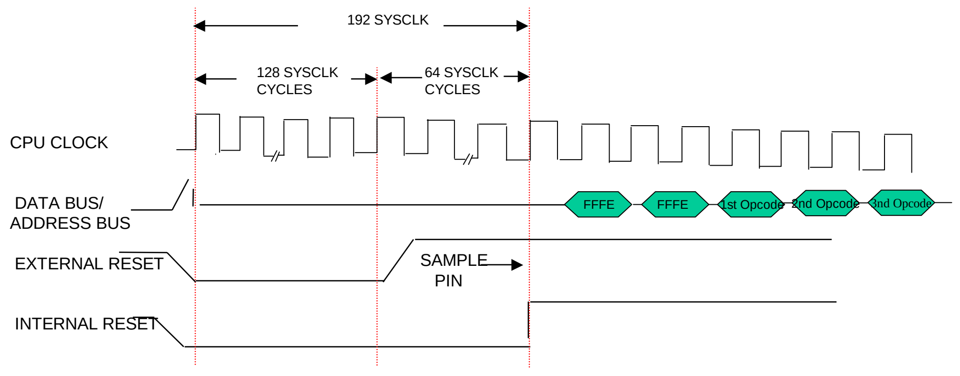
The Clock Quality is checked to determine if the crystal is suitable to run from when POR occurs, when exiting STOP mode, and/or after Clock Monitor failure.

A Quality Check Window is 50K Self-Clock-Mode (SCM) cycles. The SCM cycles is the PLL minimum frequency, from 1 to 5.5 MHz when MCU is coming out of reset. To satisfy the Clock Quality Check, 4,096 OSCLK cycles must occur within a Quality Check Window. The 4,096 counter is reset at the end of each window. This means that all 4,096 OSCLK cycles must occur within the same window.

When the Quality Check is satisfied, the MCU exits reset after 192 SYSCLK cycles. If the Quality Check is not satisfied after 50 Quality Check Windows, the MCU exits reset in SCM.

Providing that the crystal/resonator starts quickly, the MCU will leave reset (or STOP mode) within one or two Quality Check Windows. If the crystal/resonator is faulty or missing, the MCU will exit reset (or STOP mode) after 50 Quality Check Windows. This can be up to 2.5 seconds or 50 x 50K x 1 uS, assuming the SCM PLL frequency is running at minimum frequency of 1Mega Hertz clock.

External & Internal Resets



The address bus is driven with the appropriate vector address upon reset, whereas the data bus is used to read the contents of the vector and places it in the program counter to invoke the reset handler. The MCU reads 3 opcodes from the reset handler before instruction execution is started. Since the HCS12 family has 3 stage instruction queue, the queue has to be filled by executing three read operations".

The HCS12 MCU has other reset sources, one of which is External reset. The external reset generates a system reset when an input signal applied to the reset input pin.

The other system resets are internally generated and refer to as Clock Monitor reset and a COP reset.

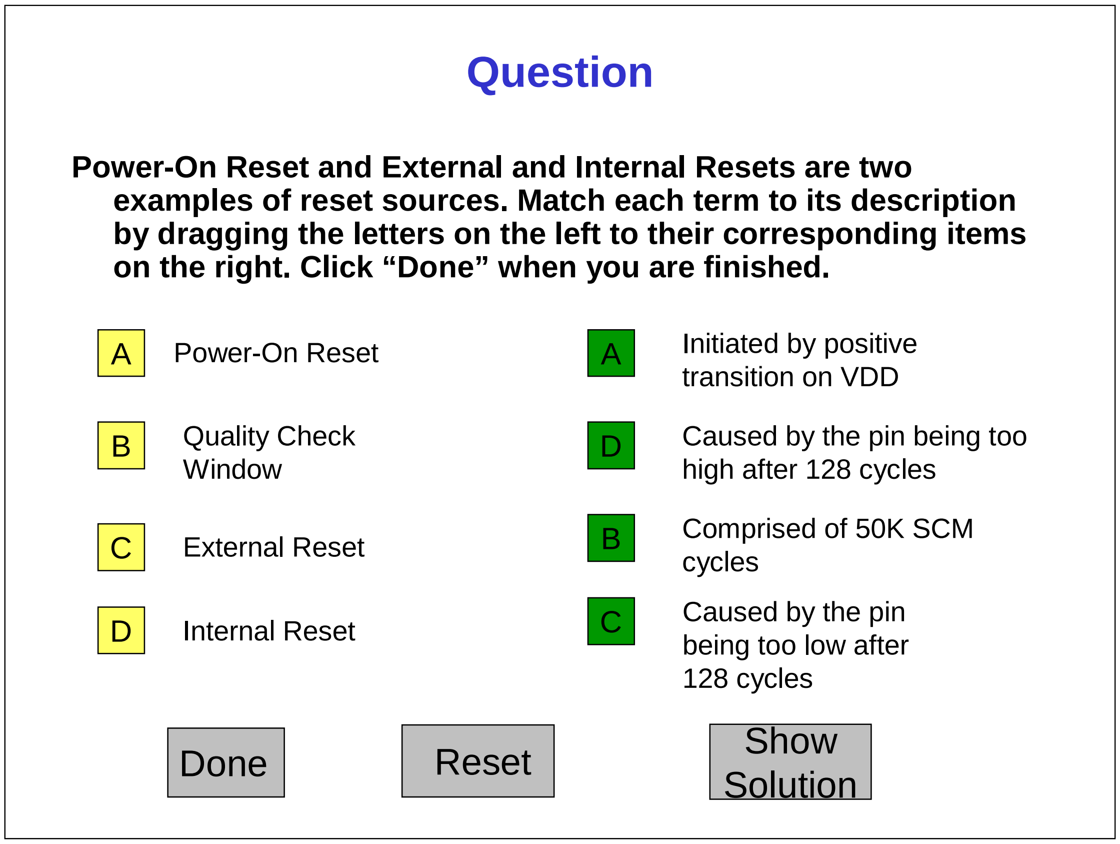
When a system reset is generated, the MCU releases the reset pin after 128 clock cycles and then samples the pin in the next 64 clock. If the pin is still low, it is assumed to be an externally generated reset, otherwise it's assumed to be as one of the internal resets.

Note: When one of the internal resets is generated due to a COP or a Clock Monitor reset, the MCU drives the reset pin as output for exactly 128 clocks.

If the pin is high when checked, it is assumed to be an Internal Reset. The pin will then take the appropriate vector (COP or Clock Monitor).

The reset line must be able to fully charge within 64 OSCLK cycles. This will enable the reset line to properly distinguish between Internal and External Resets and then take the correct vector. The capacitance on the reset line should be very small, less than 100 pF typically.

For External Reset, the reset pin must be asserted for more than two oscillator cycles. The reset pin must also negate before reset service can begin. In this case, there is no delay to stabilize the oscillator.



Question

Power-On Reset and External and Internal Resets are two examples of reset sources. Match each term to its description by dragging the letters on the left to their corresponding items on the right. Click “Done” when you are finished.

A	Power-On Reset	A	Initiated by positive transition on VDD
B	Quality Check Window	D	Caused by the pin being too high after 128 cycles
C	External Reset	B	Comprised of 50K SCM cycles
D	Internal Reset	C	Caused by the pin being too low after 128 cycles

Let’s take a moment to review Power-On Reset, Clock Quality Checks, and External and Internal Resets.

Power-On Reset is initiated by positive transition on VDD. A Quality Check Window is comprised of 50K SCM cycles. To satisfy a Clock Quality Check, 4,096 OSCLK cycles must occur within a Quality Check Window. With External Reset, the reset pin remains low after the MCU releases it and checks it 64 clock cycles later. With Internal Reset, the reset pin is high after it is released and checked.

Clock Monitor

Purpose: To ensure that the system is informed when the main reference frequency is lost due to some unusual condition

Used to: Improve the fault tolerance of the system and initiate an automatic reset or SCM entry if a slow or stopped clock is detected

Based on: An internal resistor-capacitor (RC) time delay

Reset occurs: If the clock frequency falls below a predetermined limit when the clock monitor is enabled

Reset occurs: If the clock frequency of the is less than 500 KHz, the clock monitor should not be used.

The Clock Monitor is another reset source. Its purpose is to ensure that the system is informed when the main reference frequency is lost due to some unusual condition. The Clock Monitor improves the fault tolerance of the system. This reset source also initiates an automatic reset or SCM entry if a slow or stopped clock is detected.

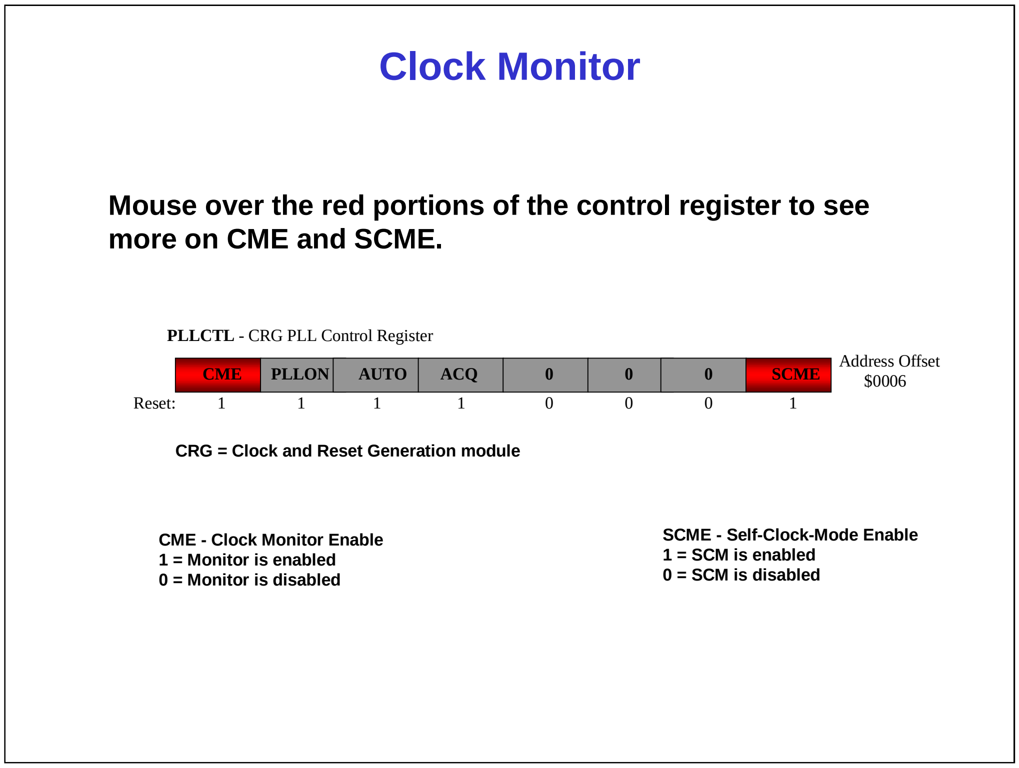
The Clock Monitor circuit is based on an internal Resistor-Capacitor (RC) time delay. If no external clock edges are detected within this RC time delay, the clock Clock Monitor circuit generates a system reset. The clock monitor function is enabled by the Clock Monitor Enable (CME) control bit in the PLL Control (PLLCTL) register. The time-out is based on an RC delay, so that the clock monitor can operate without any MCU clocks. The input to the clock monitor is the Reference Clock (REFCLK).

If the clock frequency falls below a predetermined limit when the clock monitor is enabled, a reset occurs unless the Self Clock Mode is enabled. For example, if a clock drops below a frequency of 500 KHz and the Clock Monitor function has been enabled, then a system reset is asserted on the external reset pin for 64 E- Clock cycles.

If the clock frequency is less than 500 KHz , then a clock monitor reset may occur. However, this reset is not guaranteed.

To ensure proper operation, the Clock Monitor function should not be used if the target system frequency is less than 500 KHz.

It is important to note that the Clock Monitor is automatically disabled in full STOP mode. You should also note that the time-out range for the Clock Monitor is between 5 usec and 20 usec.



Clock Monitor

Mouse over the red portions of the control register to see more on CME and SCME.

PLLCTL - CRG PLL Control Register

	CME	**PLLON**	**AUTO**	**ACQ**	**0**	**0**	**0**	**SCME**
Reset:	1	1	1	1	0	0	0	1

Address Offset
$0006

CRG = Clock and Reset Generation module

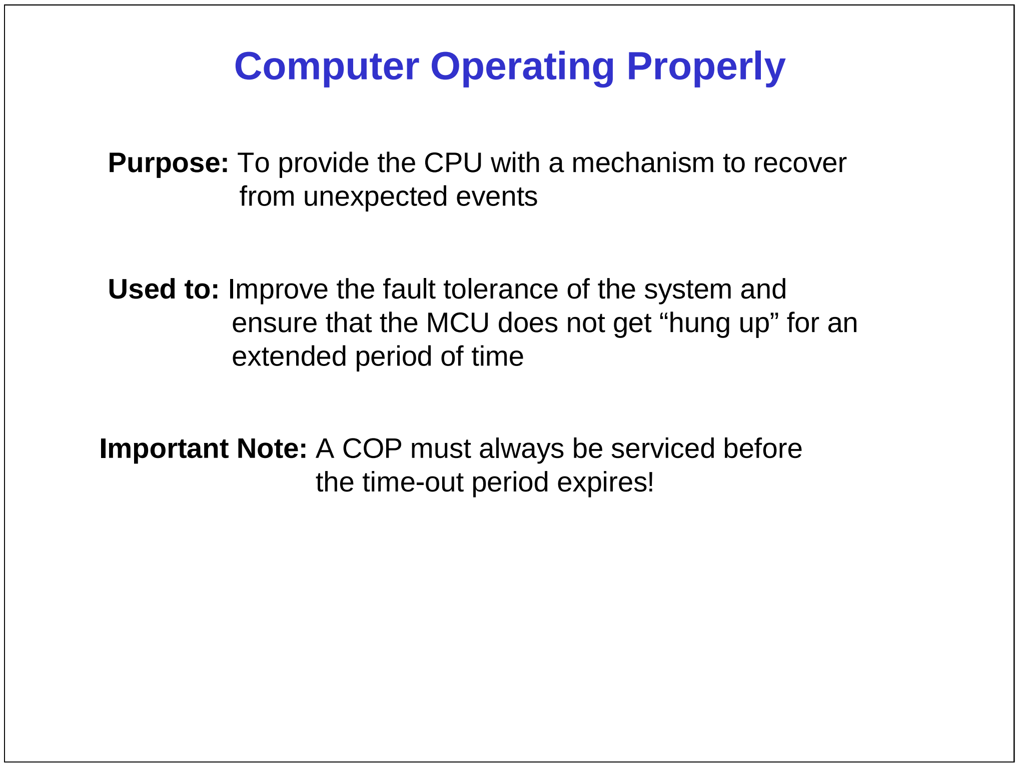
CME - Clock Monitor Enable
1 = Monitor is enabled
0 = Monitor is disabled

SCME - Self-Clock-Mode Enable
1 = SCM is enabled
0 = SCM is disabled

The Clock Monitor and the SCM are enabled out of reset. This allows the MCU to run from the internal RC oscillator if the crystal clock is not present. The internal RC clock ranges from one to 5.5 MHz.

The Clock Monitor function can be enabled or disabled at any time. When the Clock Monitor is enabled, a slow or stopped clock causes a crystal failure to reset the MCU (if the SCME equals zero) and obtain a Clock Monitor vector from $FFFC. A slow or stopped clock can also cause a crystal failure to enter SCM if the SCME bit is set to one.

The reset pin will be driven as output for 128 clock cycles when the Clock Monitor fails. Roll your mouse pointer over the red portions of the control register to see more on CME and SCME.



Computer Operating Properly

Purpose: To provide the CPU with a mechanism to recover from unexpected events

Used to: Improve the fault tolerance of the system and ensure that the MCU does not get “hung up” for an extended period of time

Important Note: A COP must always be serviced before the time-out period expires!

The final reset source is Computer Operating Properly (COP). The purpose of COP is to provide the Central Processing Unit (CPU) with a mechanism to recover from unexpected events, such as runaway software and software processing errors. For example, if the MCU is lost because of a severe noise hit due to a harsh and noisy environment, the COP circuit is able to detect this condition. The COP circuit then causes the system to reset. Resetting the MCU enables the system to recover from the runaway software condition.

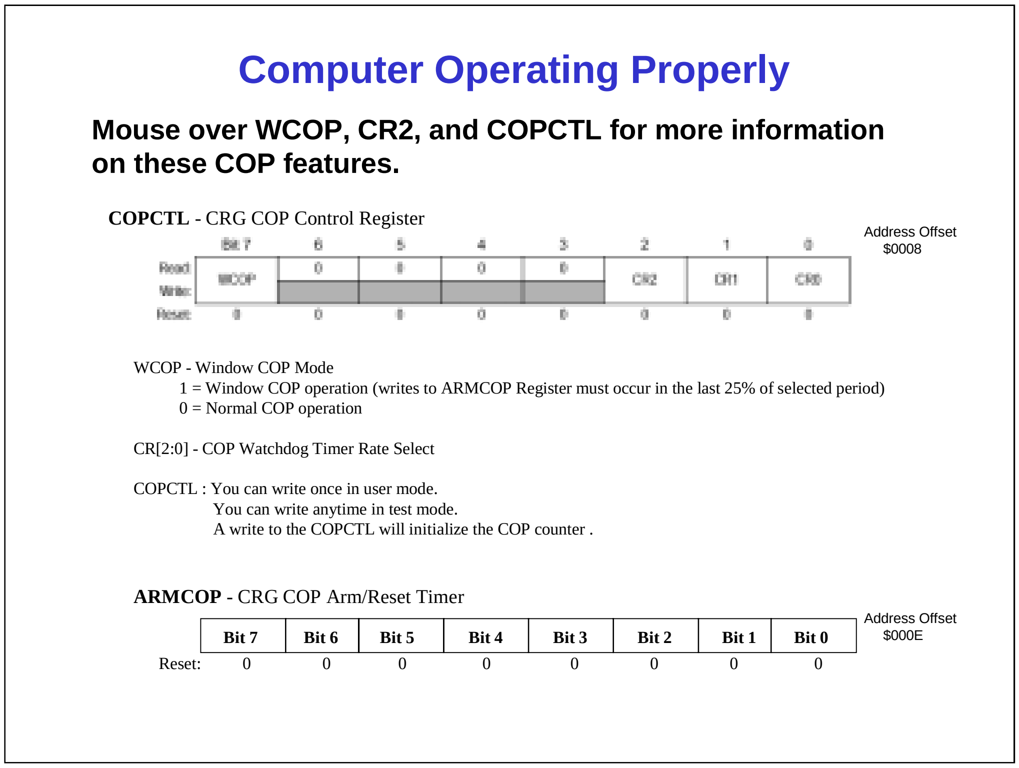
COP improves the fault tolerance of the system and insures that the MCU does not get “hung up” for an extended period of time.

If the COP is enabled and is not serviced within the specified time period, then a system reset is asserted on the external reset pin. A COP vector is also obtained from address ($FFFA).

A COP reset causes the CPU to vector to location $FFFA in the vector table. The vector location contents are automatically loaded into the Program Counter (PC) in order to point to the COP reset service routine.

To use COP, you simply enable it and select the desired time-out period. Most importantly, you must remember to always service it before the time-out period expires. If the COP timer is enabled, and it is not serviced within the time-out period, the COP will reset the CPU.

The reset pin for the COP is asserted for 128 SYSCLK cycles.



Computer Operating Properly

Mouse over WCOP, CR2, and COPCTL for more information on these COP features.

COPCTL - CRG COP Control Register

Address Offset
$0008

	Bit 7	6	5	4	3	2	1	0
Read:	WCOP	0	0	0	0	CR2	CR1	CR0
Write:		▨	▨	▨	▨			
Reset:	0	0	0	0	0	0	0	0

WCOP - Window COP Mode
1 = Window COP operation (writes to ARMCOP Register must occur in the last 25% of selected period)
0 = Normal COP operation

CR[2:0] - COP Watchdog Timer Rate Select

COPCTL : You can write once in user mode.
You can write anytime in test mode.
A write to the COPCTL will initialize the COP counter .

ARMCOP - CRG COP Arm/Reset Timer

Address Offset
$000E

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset:	0	0	0	0	0	0	0	0

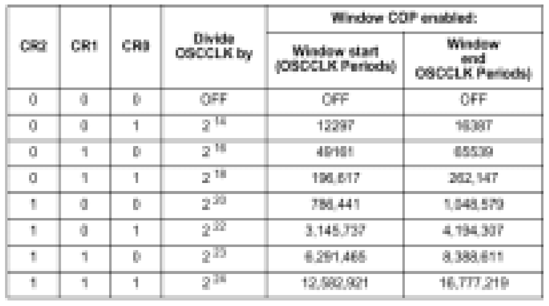
The COP is disabled out of reset because the COP Rate Select CR[2:0] bit field is cleared. To enable the COP, software must write the CR[2:0] with a value in order to select the required timeout. The COPCTL register is a write-once register that is used to protect against runaway code, which may inadvertently disable the COP. Once the control register is written with the COP time-out value in the CR[2:0] bit field, is enabled and cannot be disabled until the device is reset. Roll your mouse pointer over WCOP, CR2, and COPCTL for more information on these COP features.

The COP can be enabled to run normally or in windowed operation. In the windowed operation mode, software must service the COP in the last 25% of the selected time-out period. If the software attempts to write the ARMCOP register outside this window, the COP will generate a reset condition.

Software writes $55 followed by $AA to the ARMCOP register in order to reset the internal COP counter. The COP service sequence must be initiated before the time-out period expires by writing the ARMCOP register with the value $55 followed by $AA. Any attempt to write out of sequence or any other value to the ARMCOP register will cause an immediate reset.

Computer Operating Properly

COP Rate Selection Bit Definition



CR2	CR1	CR0	Divide OSCCLK by	Window COP enabled:	
				Window start (OSCCLK Periods)	Window end OSCCLK Periods)
0	0	0	OFF	OFF	OFF
0	0	1	2^{14}	12297	16387
0	1	0	2^{16}	49161	65539
0	1	1	2^{18}	196,617	262,147
1	0	0	2^{20}	786,441	1,048,579
1	0	1	2^{22}	3,145,737	4,194,307
1	1	0	2^{23}	6,291,465	8,388,611
1	1	1	2^{24}	12,582,921	16,777,219

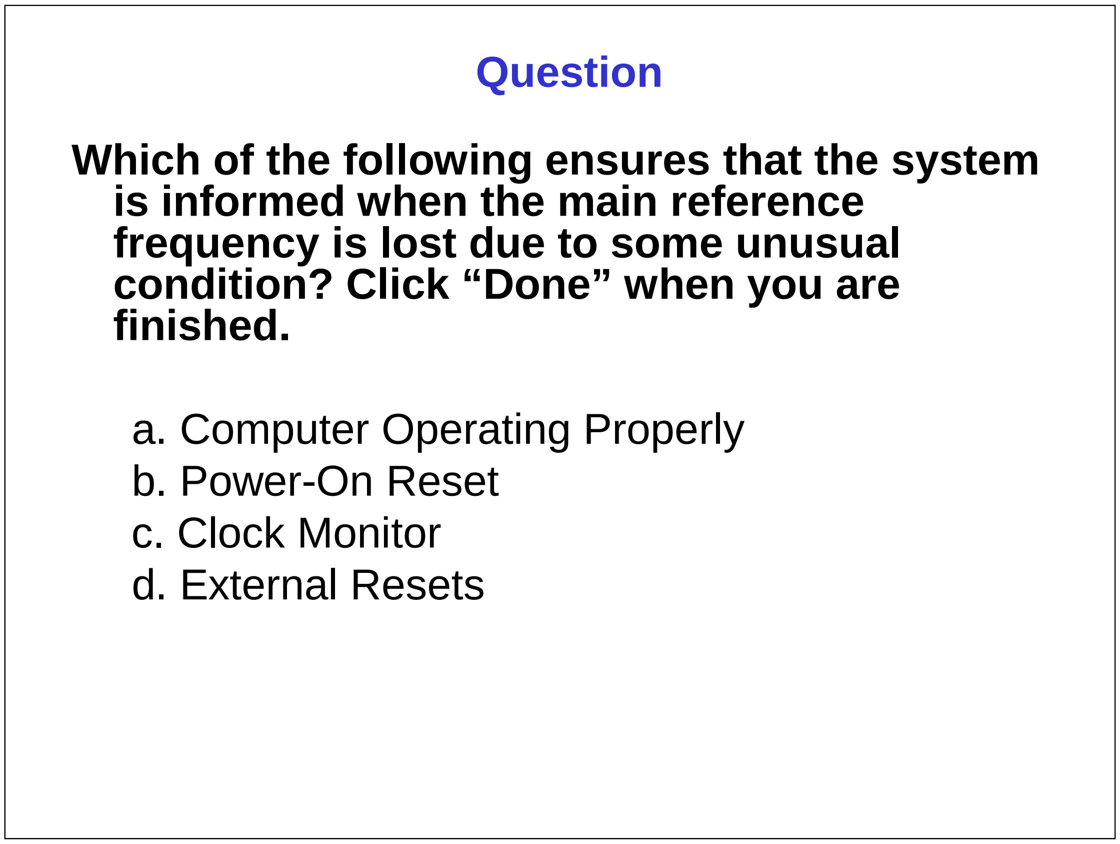
Equation 1: Timeout = WindowEnd = OSCCLK Period * (OSCCLK Divider + 3)

Equation 2: WindowStart = OSCCLK Period * (0.75* OSCCLK Divider) + 9)

You can use the following equation to select one of the possible COP timeouts.

In normal COP operation, the user may use the first equation to calculate the timeout period.

In Windowed COP, the user may select the second equation to calculate the window timeout period.



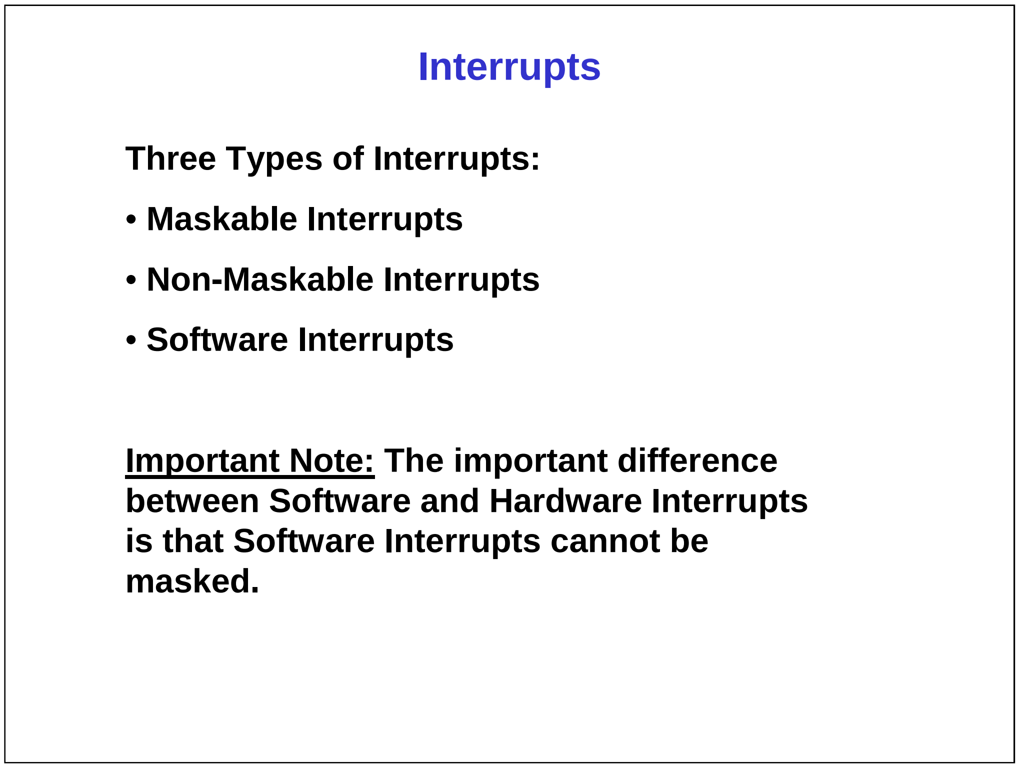
Question

Which of the following ensures that the system is informed when the main reference frequency is lost due to some unusual condition? Click “Done” when you are finished.

a. Computer Operating Properly
b. Power-On Reset
c. Clock Monitor
d. External Resets

Let’s take a moment to review Clock Monitor and Computer Operating Properly (COP). Complete this question to test your knowledge.

The purpose of the Clock Monitor is to ensure that the system is informed when the main reference frequency is lost due to some unusual condition. The purpose of COP is to provide the CPU with a mechanism to recover from runaway events.



Interrupts

Three Types of Interrupts:

- **Maskable Interrupts**
- **Non-Maskable Interrupts**
- **Software Interrupts**

Important Note: The important difference between Software and Hardware Interrupts is that Software Interrupts cannot be masked.

Now that we have discussed different types of reset sources, let's move on to interrupts and their various sources. There are three types of interrupts: Maskable, Non-Maskable, and Software Interrupts. When an interrupt event is detected, the MCU temporarily suspends the current program execution, which allows the MCU to service the interrupt request.

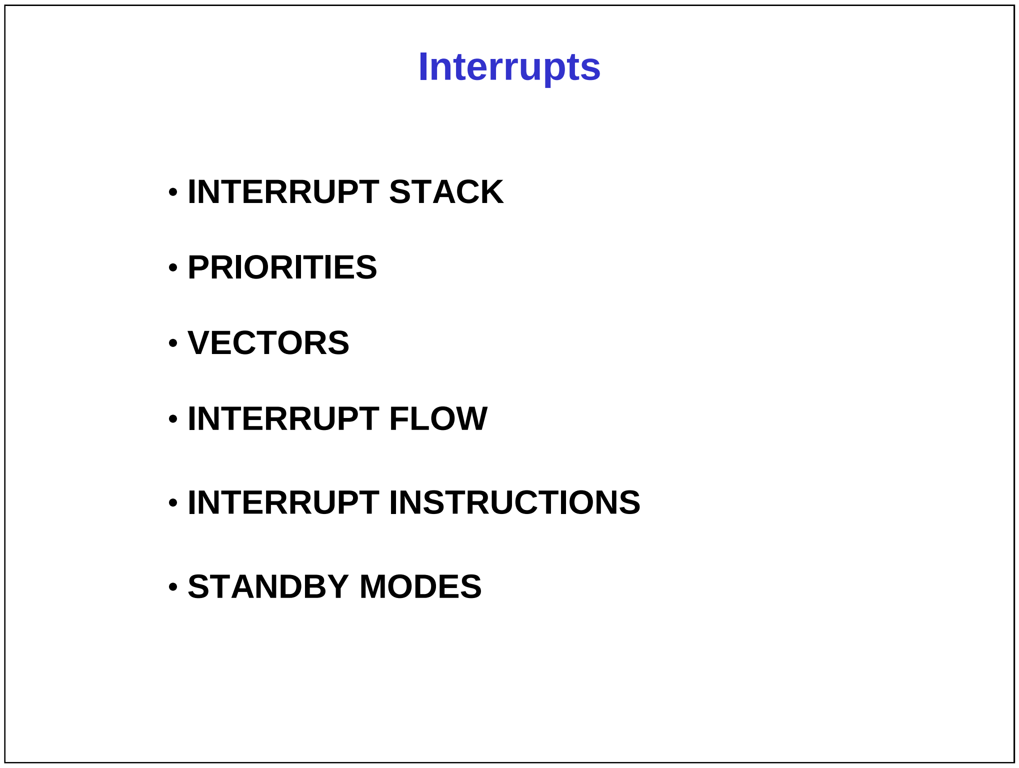
An interrupt is similar to a reset in that it causes the MCU to obtain a new address for the program counter and sets the interrupt mask bit, or I-bit. Unlike a reset, interrupts suspend normal program execution only temporarily, so the processor can service the interrupt. After the interrupt is serviced, the processor is returned to where it left off when executing normal program code.

A Maskable Interrupt occurs when the external IRQ pin is asserted as a result of one of the internal peripherals requesting service from the CPU.

A Non-Maskable Interrupt is generated when the external XIRQ pin is asserted by some Input/Output (I/O) device.

Out of reset, both Maskable and Non-Maskable Interrupts are disabled. This allows the MCU to configure the application to a known state prior to recognizing any of the interrupt requests. Maskable Interrupts are masked by the I-bit in the Condition Code Register (CCR), and Non-Maskable Interrupts are masked by the X-bit.

A Software Interrupt occurs as a result of the Software Interrupt (SWI) instruction. A Software Interrupt is always executed as part of the instruction flow. The important difference between Software and Hardware Interrupts is that Software Interrupts cannot be masked. This means that the value of the I-bit has no effect on Software Interrupts. Otherwise, a Software Interrupt is processed the same way as a Hardware Interrupt.



Interrupts

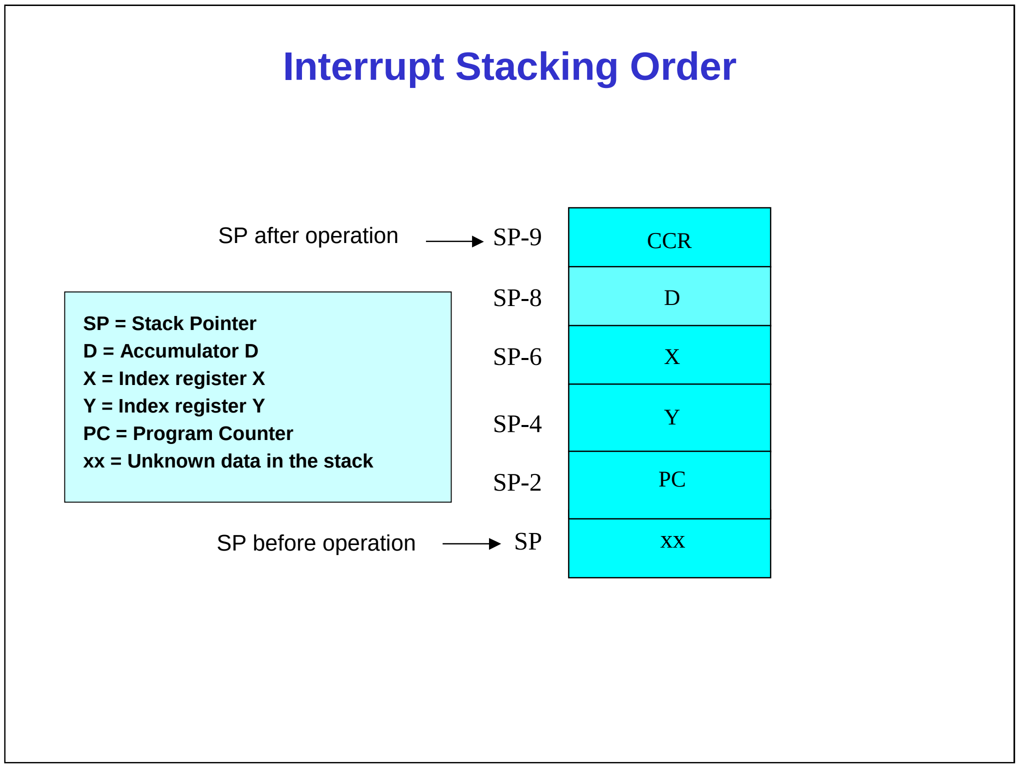
- **INTERRUPT STACK**
- **PRIORITIES**
- **VECTORS**
- **INTERRUPT FLOW**
- **INTERRUPT INSTRUCTIONS**
- **STANDBY MODES**

Sometimes, an interrupt may be referred to as an exception event. Let’s take a moment to explain what is meant by an exception event.

An exception causes the MCU to deviate from normal processing. Normal processing is typically when the MCU is executing the main application program. When an interrupt or exception event occurs, the MCU temporarily suspends main program execution to handle and service the exception.

Once the interrupt or exception event is serviced, the MCU returns to the main program that was momentarily suspended and no state information is lost.

The HCS12 MCU has a number of interrupt sources. Each internal I/O peripheral device can generate one or more interrupt requests. All internally generated interrupts, as well as XIRQ interrupts, are level sensitive. The IRQ pin is software programmable to be either level sensitive or edge sensitive.



Earlier, we discussed Maskable, Non-Maskable, and Software Interrupts. Now, we are going to revisit these three interrupts and learn how they relate to interrupt stacking order.

A Maskable Interrupt is generated by internal or external hardware conditions. This type of interrupt can also be initiated by an external pin or an internal I/O peripheral module. When a Maskable Interrupt occurs, the program context is stacked, the I-bit in the CCR is set, and the interrupt vector is fetched. Maskable Interrupts can be masked. This means that Maskable Interrupts can be recognized only when the I-bit is cleared.

A Non-Maskable Interrupt is always recognized, assuming that the X-bit in the CCR is cleared. Although the X-bit will mask XIRQ interrupts when set, typically, the X-bit is cleared at system initialization time. Once the X-bit is cleared, it can be set again until another reset occurs. This means that XIRQ interrupt requests become non-maskable immediately after clearing the X-bit in the CCR.

A Software Interrupt occurs as a result of the SWI instruction and is always executed as part of the instruction flow. Remember that Software Interrupts cannot be masked, which means that the value of the I-bit has no effect on software interrupts.

When any of the interrupts are recognized, the MCU saves all of its registers onto the stack to remember the exact program state that was interrupted. The MCU hardware automatically sets the appropriate mask bit in the CCR to prevent further interrupts from being recognized during the execution of the interrupt service routine.

When the HCS12 acknowledges an interrupt, it will stack it's registers and then determines which vector to fetch from the vector table. Stack operation is typically performed in 5-BUS cycles, even if the SP is misaligned.

Non-Maskable Exception Priority

Non-Maskable exception priorities:

1. **$\overline{\text{RESET}}$**
2. **Clock Monitor**
3. **COP**
4. **TRAP**
5. **$\overline{\text{XIRQ}}$**
6. **SWI**

It is important to note that there are over 50 interrupt sources and each reset or interrupt source has a separate vector.

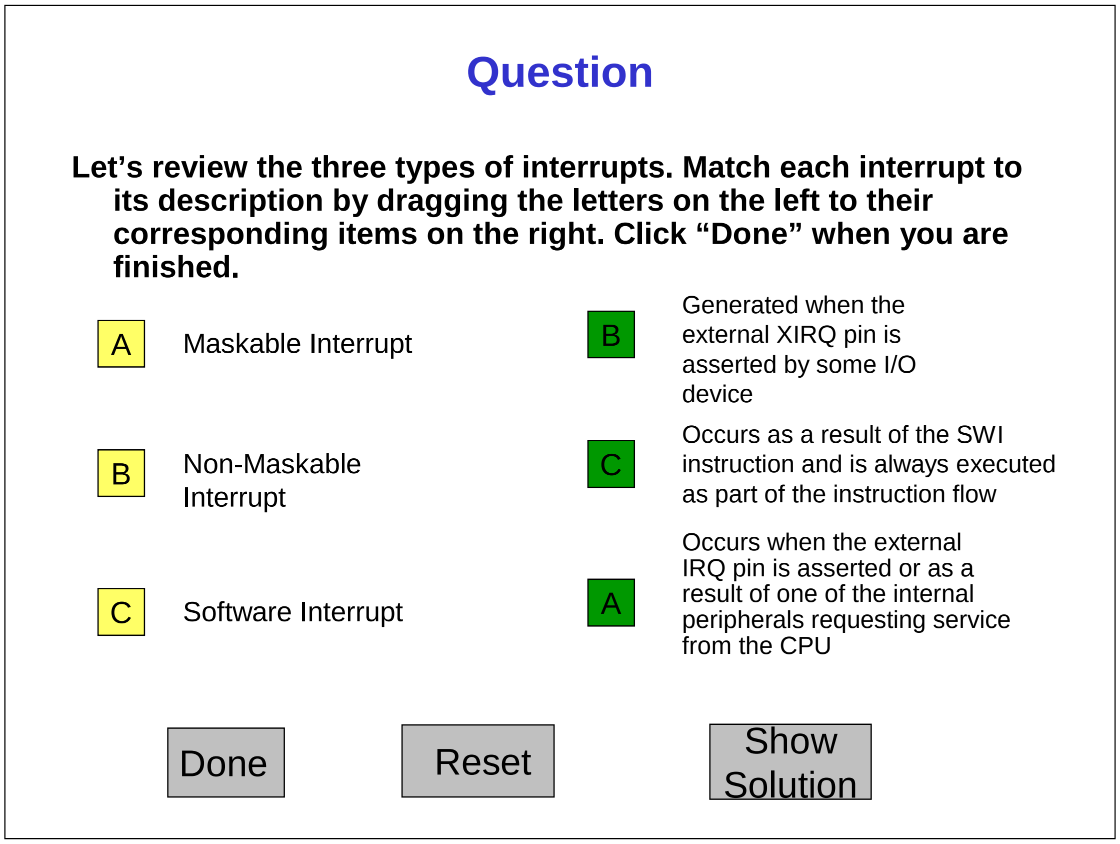
There are six Non-Maskable Interrupts available on the HCS12 MCU. These interrupts have fixed priorities.

Earlier, we discussed Power-On Resets and External Hardware Resets. These two reset sources have the highest priority. We also discussed Clock Monitor reset and COP reset. These two reset sources have the next highest priority. It is important to note here that both Clock Monitor and COP resets can generate an external reset.

The next highest priority is a Trap exception. A Trap exception occurs when the MCU attempts to execute an illegal instruction. This condition may occur due to a software runaway condition. When the MCU detects this condition, it will trap and execute a recovery routine written by the system designer to correct this condition.

Earlier, we spent some time discussing the XIRQ Non-Maskable Interrupt request pin. The one thing to add at this point is that this interrupt is usually related to some external I/O device requiring service from the CPU. Once enabled, the XIRQ cannot be masked. Usually this interrupt request should be used for catastrophic condition indication to the MCU system.

The last Non-Maskable exception priority is Software Interrupt (SWI) exception. This exception is initiated by the user’s program executing SWI instruction. This instruction may be used in multi-tasking systems to perform system calls to the operating system. This instruction may also be used for development and insertion of breakpoints during debugging.



Question

Let’s review the three types of interrupts. Match each interrupt to its description by dragging the letters on the left to their corresponding items on the right. Click “Done” when you are finished.

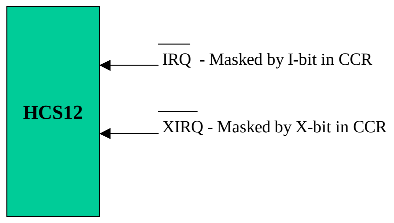
A	Maskable Interrupt	B	Generated when the external XIRQ pin is asserted by some I/O device
B	Non-Maskable Interrupt	C	Occurs as a result of the SWI instruction and is always executed as part of the instruction flow
C	Software Interrupt	A	Occurs when the external IRQ pin is asserted or as a result of one of the internal peripherals requesting service from the CPU

Done Reset Show Solution

Let’s take a moment to review the three types of interrupts.

A Maskable Interrupt occurs when the external IRQ pin is asserted or as a result of one of the internal peripherals requesting service from the CPU. A Non-Maskable Interrupt is generated when the external XIRQ pin is asserted by some external I/O device. A Software Interrupt occurs as a result of the SWI instruction and is always executed as part of the instruction flow. The important difference between Software and Hardware Interrupts is that Software Interrupts cannot be masked.

Interrupt Sensitivity


HCS12
$\overline{IRQ}$ - Masked by I-bit in CCR
$\overline{XIRQ}$ - Masked by X-bit in CCR

INTCR - Interrupt Control Register

	Bit 7	6	5	4	3	2	1	Bit 0
Write once	IRQE	IRQEN	0	0	0	0	0	0
Reset:	0	1	0	0	0	0	0	0

Address Offset
$001E

IRQE – Interrupt request Edge Select
0 =Level sensitive
1 = Edge sensitive

IRQEN – Interrupt Request Enable
0 = Interrupt request pin is disconnected from interrupt logic.
1 = Interrupt request pin is enabled

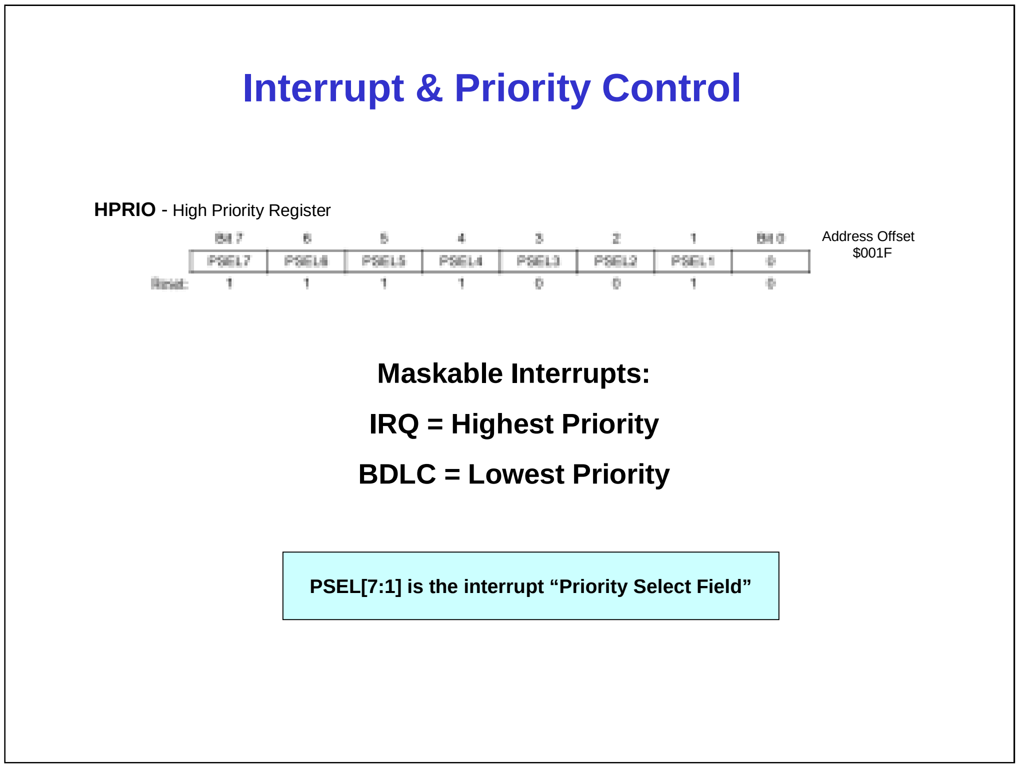
The HCS12 has two external pins called IRQ and XIRQ. The IRQ pin generates Maskable Interrupts, while the XIRQ pin generates Non-Maskable Interrupts. Here, you can see that the IRQ pin is masked by the I-bit in the CCR, and the XIRQ interrupt is masked by the X-bit in the CCR. When the XIRQ interrupt is enabled, it cannot be disabled and becomes a non-maskable interrupt source.

The XIRQ pin assertion is always detected by the MCU as a level-sensitive signal, whereas the IRQ pin assertion is programmable to be either level- or edge-sensitive. The Interrupt Control Register (INTCR) provides the user with the flexibility to program the IRQ pin as either level- or edge-sensitive. Furthermore, this register may be used to enable the IRQ pin or disconnect it from the system interrupt logic.

With the Interrupt Select Edge Sensitive (IRQE), “1” means that the IRQ pin is configured for a negative edge. A “0” means that the IRQ pin is configured to be level sensitive.

With the External IRQ Enable (IRQEN), “1” means that the IRQ pin is connected to interrupt logic. A “0” means that the IRQ pin is disconnected from interrupt logic.

The XIRQ and IRQ pins have internal pull-ups and are enabled out of reset. A pull-up can be turned off by clearing Pull-up enable (PUPEE) bit in the Pull-up Control Register (PUCR).



Interrupt & Priority Control

HPRIO - High Priority Register

	Bit 7	6	5	4	3	2	1	Bit 0	Address Offset $001F
	PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0	
Reset:	1	1	1	1	0	0	1	0	

Maskable Interrupts:

IRQ = Highest Priority

BDLC = Lowest Priority

PSEL[7:1] is the interrupt "Priority Select Field"

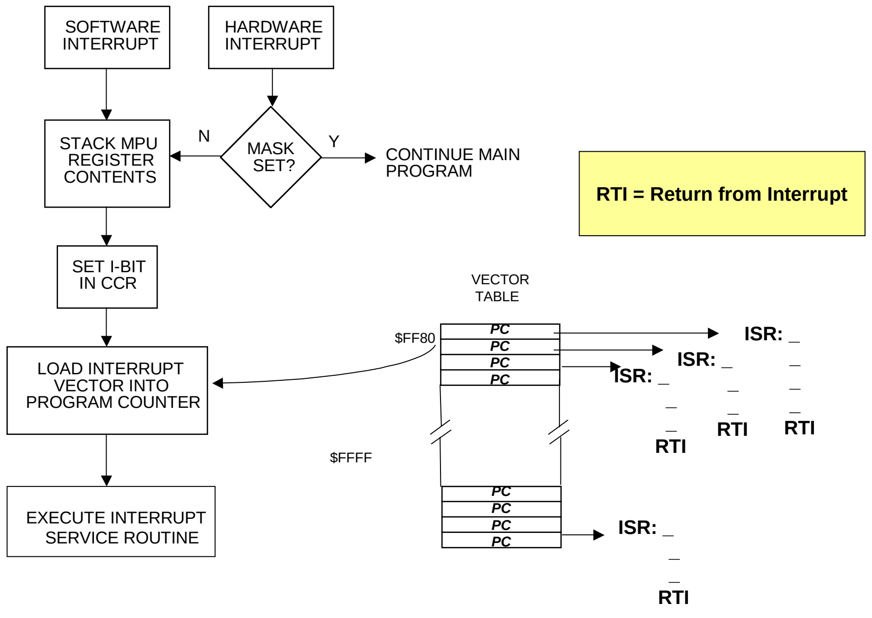
A maskable interrupt source can be elevated to the highest priority by writing to the High Priority (HPRIO) register, bits 7 to 1. Interrupt priority can only be changed when I-bit is set to a '1' in the Condition Code Register, CCR.

To promote an interrupt, the user writes the least significant byte of the associated interrupt vector address to the HPRIO register. If an unimplemented vector address or a non I-masked vector address (with a value higher than $F2) is written, then $FFF2 vector will be the default or highest priority interrupt.

There are many Maskable Interrupt requests found in the HCS12 system. With Maskable Interrupts, the IRQ has the highest priority, and the Byte Data Link Controller (BDLC) has the lowest priority.

It is sometimes desirable to elevate the priority of one Maskable Interrupt over another. This is accomplished by programming the HPRIO register. For example, to elevate a Real-Time Interrupt (RTI) to the highest level, the software writes the least significant byte of the RTI vector number into the HPRIO register. This will cause the priority order to have RTI as the highest Maskable Interrupt and IRQ as the second highest and so on.

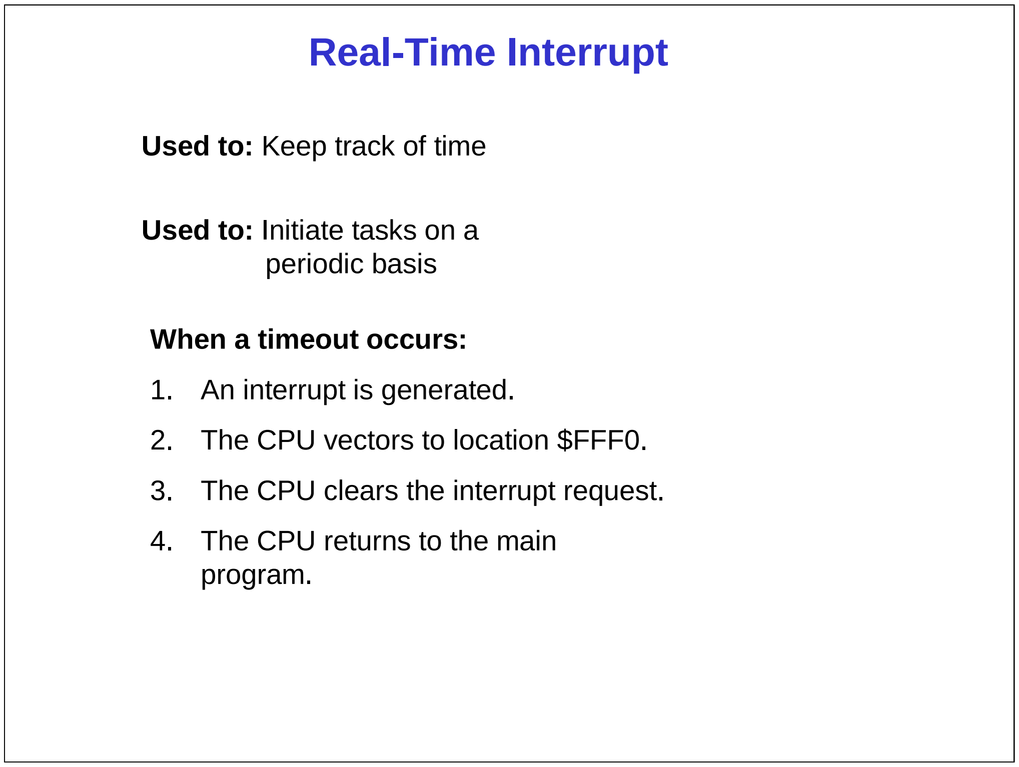
Interrupt Flow


SOFTWARE INTERRUPT
HARDWARE INTERRUPT
STACK MPU REGISTER CONTENTS
N
MASK SET?
Y
CONTINUE MAIN PROGRAM
RTI = Return from Interrupt
SET I-BIT IN CCR
VECTOR TABLE
$FF80
PC
ISR: _
RTI
LOAD INTERRUPT VECTOR INTO PROGRAM COUNTER
$FFFF
EXECUTE INTERRUPT SERVICE ROUTINE

When the CPU receives an interrupt request, it first completes the current instruction being executed. The CPU also pushes all CPU registers onto the system stack in order to remember the previous state of the machine. The CPU then updates the CCR to mask out further interrupts. The interrupt event causes the MCU to vector to one of the many in the vector table in order to point to the appropriate service routine.

After the MCU services the interrupt, it executes a Return from Interrupt instruction. The Return from Interrupt instruction is the last instruction in the interrupt handler. This restores the machine context by pulling all CPU registers from the stack in order to return to the program that was interrupted.

The flow chart depicted here shows the exact sequence of how the MCU handles interrupt processing.



Real-Time Interrupt

Used to: Keep track of time

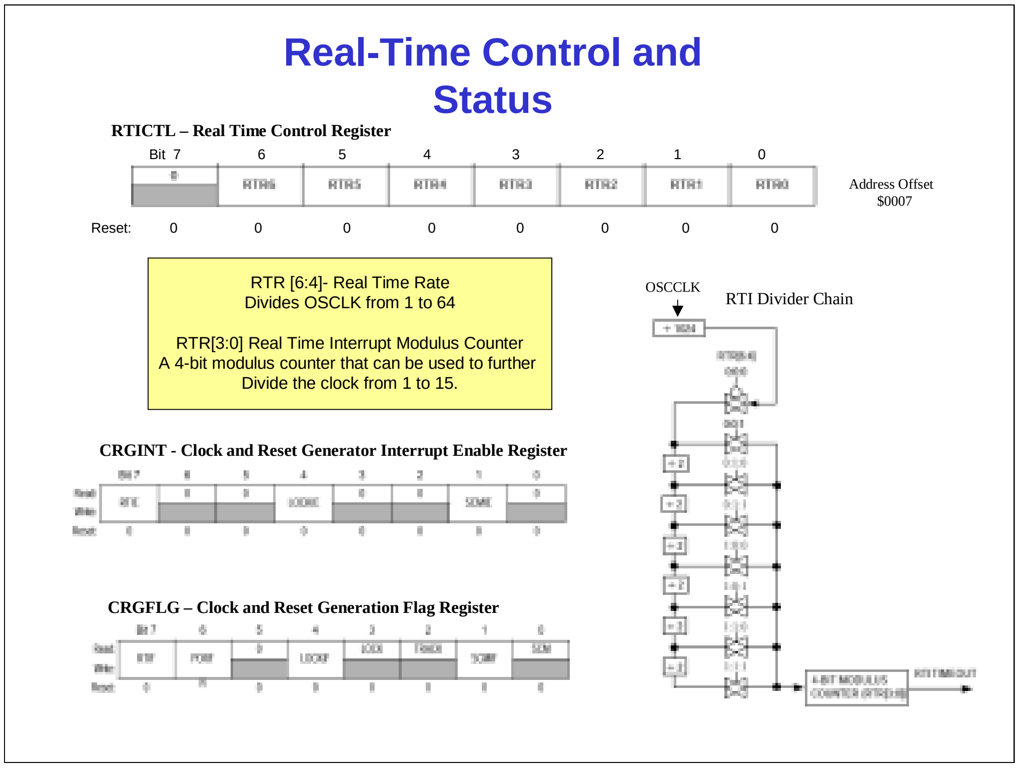
Used to: Initiate tasks on a periodic basis

When a timeout occurs:

1. An interrupt is generated.
2. The CPU vectors to location $FFF0.
3. The CPU clears the interrupt request.
4. The CPU returns to the main program.

Now, let’s take a look at Real-Time Interrupts (RTIs). RTIs are used to keep track of time and to initiate tasks on a periodic basis. When a timeout occurs, an interrupt request to the CPU is generated and the RTI vector is fetched from address ($FFF0).

The HCS12 offers a timer that can generate periodic interrupts for the purpose of scheduling events on a periodic basis. As the RTI timer times out, an interrupt is signaled to the CPU. The CPU then vectors to location $FFF0 in order to load the Program Counter (PC) with the address that points to the interrupt service routine. The CPU handles the interrupt request based on system requirements. It then clears the interrupt request and returns back to the main program.



Real-Time Control and Status

RTICTL – Real Time Control Register

	Bit 7	6	5	4	3	2	1	0	
	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0	Address Offset $0007
Reset:	0	0	0	0	0	0	0	0	

RTR [6:4]- Real Time Rate
Divides OSCLK from 1 to 64

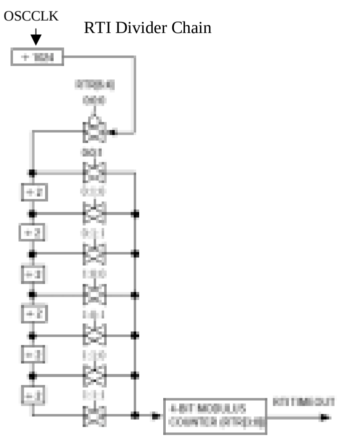
RTR[3:0] Real Time Interrupt Modulus Counter
A 4-bit modulus counter that can be used to further
Divide the clock from 1 to 15.

CRGINT - Clock and Reset Generator Interrupt Enable Register

	Bit 7	6	5	4	3	2	1	0
Read:	RTIE	0	0	LOCKIE	0	0	SCMIE	0
Write:		▨	▨		▨	▨		▨
Reset:	0	0	0	0	0	0	0	0

CRGFLG – Clock and Reset Generation Flag Register

	Bit 7	6	5	4	3	2	1	0
Read:	RTIF	PORF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM
Write:			▨		▨	▨		▨
Reset:	0	[illegible]	0	0	0	0	0	0



A Real-Time Interrupt uses three registers to control its operation. The Real-Time Clock Control (RTICTL) register allows the CPU to choose out of many different timeouts, based on the value written into the register. In order to initialize the internal RTI counter, the CPU must write to the RTICTL register. An RTR[6:4] is a Real-Time Interrupt Prescale Rate Select, and an RTR[3:0] is a Real-Time Interrupt Modulus Counter Select. The maximum time-out can be set by writing the Real Time Rate bit field with hex $7F. This will divide the input clock by 64 and then by 15.

In the Clock and Reset Generation Interrupt Enable (CRGINT) register, the RTI has an interrupt enable, which is the Real-Time Interrupt Enable (RTIE) bit. Setting this bit enables the RTI, Else disables it. In this case, “0” means that the interrupt is disabled, and “1” means that the interrupt is enabled.

In the CRG Flag (CRGFLG) register, the RTI timer has a flag bit called Real-Time Interrupt Flag (RTIF) that will set whenever the timer times out. Whenever this flag is set and its interrupt enable bit is also set, the RTI timer signals an interrupt request to the CPU. The CPU must clear the RTIF by writing it with a logic of “1.”

The RTIF bit is automatically set to “1” at the end of every RTI period. This flag can only be cleared by writing a “1.” Note that a “0” means that a timeout has not yet occurred. A “1” is set when the time-out period is met.

Question

We have learned about various types of interrupts. Which interrupt is used to keep track of time and to initiate tasks on a periodic basis? Click "Done" when you are finished.

a. Maskable Interrupts

b. Real-Time Interrupts

c. Priority Interrupts

d. Power-On Interrupts

Complete this question to test your knowledge of the various types of interrupts.

Real-Time Interrupts are used to keep track of time and to initiate tasks on a periodic basis. When a timeout occurs, an interrupt request to the CPU is generated and the RTI vector is obtained from address ($FFF0).

Module Summary

- **Power-On Resets**
- **External and Internal Resets**
- **Clock Monitor**
- **Computer Operating Properly (COP)**
- **Interrupt Exceptions, Sources, Priorities, and Flow**
- **Maskable and Non-Maskable Interrupts**
- **Real-Time Interrupts**

This module has explained MCU processing of reset and interrupt exception events. This module has also introduced you to different sources of reset and interrupt events, as well as MCU exception processing.