

))

CHAPTER 8

M68HC12 INTERRUPTS

))q

OBJECTIVES

This chapter covers the interrupt system of the M68HC12. You will learn about the vectors and the hardware prioritization that can be modified dynamically by a program. Nonmaskable interrupts are covered. Examples are given for interrupt service routines used in dedicated applications and when operating in debugging environments such as the HC12A4EVB and M68EVB912B32 evaluation boards. The discussion of the interrupts of the parallel I/O system started in Chapter 7 is continued here.

- 8.1 Introduction
- 8.2 M68HC12 and M68HC11 Interrupt Comparison
- 8.3 The Interrupt Process
 - The Interrupt Enable
 - The Interrupt Disable
 - The Interrupt Request
 - The Interrupt Sequence
 - The Interrupt Return
- 8.4 Interrupt Vectors
 - M68HC12 System Vectors
 - Initializing the Interrupt Vectors
 - D-Bug12 Monitor Interrupt Vector Jump Table
- 8.5 Interrupt Priorities
- 8.6 Nonmaskable Interrupts
 - &&&&T**
 - Clock Monitor Failure
 - Computer Operating Properly - COP
 - Unimplemented Instruction Opcode Trap
 - Software Interrupt - SWI
 - Nonmaskable Interrupt Request**&&&&Q**
- 8.7 External Interrupt Sources
 - &&&Q**
 - Key Wakeups
 - Interrupt Enable and Flags Registers
 - Port J Rising and Falling Edge Selection

- Port J Pull-up/Pull-down Selection
- Key Wakeup Initialization
- 8.8 Interrupt Flags
 - Multiple Key Wakeup Interrupts
 - Resetting Interrupt Flags
- 8.9 Internal Interrupt Sources
- 8.10 Advanced Interrupts
 - Shared $\&\&\&Q$ and Parallel I/O Interrupt Vector
 - Polling for Multiple External Devices
 - Selecting Edge or Level Triggering
 - What to do While Waiting for an Interrupt
- 8.11 The Interrupt Service Routine
 - Interrupt Service Routine Hints
 - M68HC12 Dedicated Application System ISR Examples
 - D-Bug12 Monitor ISR Examples
- 8.12 Conclusion and Chapter Summary Points

The M68HC12 has a variety of interrupting sources within the microcontroller itself and these systems will be covered in following chapters. Each interrupt source has its own vector with a limited, programmable hardware prioritization. To activate and use the interrupt system, the programmer must be sure to do the following:

- ! Initialize the stack pointer before unmasking interrupts.
- ! Initialize the user vector jump table if using a development environment such as the EVB evaluation board.
- ! Initialize the interrupt vector location(s) if not working in a development environment.
- ! Reset any interrupt-causing bits in the I/O devices to be used.
- ! Enable the I/O device's interrupt capability by setting the appropriate bit in a control register.
- ! Unmask global interrupts by clearing the I-bit in the condition code register.

The interrupt service routine is entered after the processor finishes the current instruction, pushes the registers and program counter onto the stack, and masks further interrupts. During the interrupt service routine, the programmer must do the following:

- ! Use global data for inter-process communications.
- ! Keep the interrupt service routine short.
- ! Reset any flags that caused the interrupts.
- ! Use the RTI instruction to return to the interrupted program.