COP8™ Microcontroller

COP8 Flash Family User's Manual

Literature Number 620xxx-001
April 2000

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
-001	06/00	First Release COP8 Flash Family User's Manual

PREFACE

The COP8™ Flash Family of 8-bit microcontrollers is ideally suited to embedded controller applications such as keyboard interfaces, electronic telephones, home appliances, and ABS systems. The design of this family takes advantage of CMOS technology, providing a useful combination of high performance, low power consumption, and reasonable cost. The rich instruction set and flexible addressing modes of the COP8 controllers contribute to their high performance and code efficiency.

This manual describes the features, architecture, instruction set, and usage of the COP8 microcontrollers. The beginning chapters describe general features shared by all family members. The remaining chapters describe individual family members and their device-specific features.

Chapter 1, OVERVIEW, provides an overview of the COP8 family and compares the features of different family members.

Chapter 2, ARCHITECTURE, describes the overall architecture of the COP8 microcontroller, including the CPU core, registers, memory organization, reset operation, and clock options.

Chapter 3, INTERRUPTS, describes the device interrupts and how they are used. The types of interrupts vary from one family member to another.

Chapter 4, TIMERS, describes the on-chip timers and their operating modes. The number and types of timers vary from one family member to another.

Chapter 5, MICROWIRE/PLUS, describes the microcontroller's MICROWIRE/PLUS serial interface and its operating modes.

Chapter 6, POWER SAVE MODES, describes the special operating modes in which the microcontroller is shut down, reducing power consumption to a very low value while maintaining the processor status and all register contents. All family members have a HALT mode, and some also have an IDLE mode that maintains real time while the processor is shut down.

Chapter 7, INPUT/OUTPUT, describes the input/output ports of the microcontroller and how they are used. The number and types of ports vary from one family member to another.

Chapter 8, WATCHDOG AND CLOCK MONITOR, describes an internal circuit available in some COP8 devices that monitors the operation of the microcontroller and reports an abnormal condition by issuing a signal on an output pin.

Chapter 9, 10-BIT SUCCESSIVE APPROXIMATION A/D CONVERTER, describes a multi-channel Analog-to-Digital Converter unit, which converts an analog signal into a digital value.

Chapter 10, USART, describes a Universal Synchronous/Asynchronous Receiver/ Transmitter unit, which can be used for serial data communications.

Chapter 11, IN SYSTEM PROGRAMMING AND VIRTUAL E2, describes the use of the Flash memory and the built-in capabilities for In System Programming. This chapter also describes the use of Flash memory to provide Virtual EEPROM.

The remaining chapters describe the specific features of different COP8 Flash Family members. The chapters are in alphabetical order by COP8 device name: COP8CBR and COP8SBR. Only device-specific information is provided in these chapters. Features common to several or all COP8 devices are described in the earlier chapters.

Appendix A, INSTRUCTION SET, describes the instruction set of the COP8 Flash Family microcontrollers, including detailed descriptions of each instruction.

Appendix B, APPLICATION HINTS, provides additional information that may be useful in implementing a design.

Appendix C, ELECTRICAL CHARACTERIZATION DATA, shows the general electrical characteristics of COP8 devices such as power consumption, source current, and sink current. For additional information, consult the device-specific data sheets.

The information contained in this manual is for reference only and is subject to change without notice.

1. COP8 is a trademark of National Semiconductor Corporation.

CONTENTS v

LIST OF FIGURES

Figures

Tables

Chapter 1

OVERVIEW

1.1 INTRODUCTION

The COP8 Flash Family 8-bit microcontrollers provide high-performance, low-cost solutions for embedded control applications. COP8 Flash Family devices are fabricated with CMOS technology for low current drain and a wide operating voltage range. Most instructions are single-byte and have an execution time of one instruction cycle, allowing high throughput. Multiple addressing modes and a rich instruction set further enhance throughput efficiency and reduce program size. Other COP8 features such as reconfigurable inputs and outputs, multi-mode general-purpose timers, and the MICROWIRE/PLUS™ serial interface provide the flexibility needed to construct single-chip solutions for a wide variety of applications.

All COP8 Flash Family members share the set of features listed in Section 1.2. Many individual family members also contain other features such as additional timers, an A-to-D converter, or a USART. The device-specific features are described in Section 1.2.2.

1.2 FEATURES

1.2.1 Basic Features

Each member of the COP8 family of microcontrollers offers the following features:

- Flash memory.
- In-System Programming and Virtual EEPROM for firmware update and non-volatile data flexibility.
- Clock Doubler for high speed operation from lower frequency oscillator
- Dual clock for reduced power dissipation when high speed processing is not needed
- 8-bit core processor.
- CMOS technology for low power, fully static operation.
- HALT mode for very low standby power.
- Memory mapped architecture. All RAM, I/O ports, and registers (except A and PC) are mapped into the data memory address space.
- On-chip data memory and program memory.
- Flexible, reconfigurable I/O.

- MICROWIRE/PLUS serial interface, a 3-wire serial data communication system that allows the microcontroller to be programmed for either master or slave mode operation.
- Extremely versatile 16-bit timer with two associated autoload/capture registers, which can operate in any of three modes: PWM (Pulse Width Modulation), external event counter, or input capture register.
- Non-maskable Software Trap interrupt.
- Maskable interrupts (number and type depending on family member).
- Two 8-bit “Register Indirect” data memory pointers.
- 8-bit Stack Pointer (SP) for stack in data memory RAM.
- Choice of clock types: crystal oscillator, or R/C oscillator (not available on all devices).
- IDLE mode for very low standby power while maintaining real time with associated IDLE Timer.
- Reduced electromagnetic radiated emissions using internal power supply filters, a low-current crystal oscillator, and gradual turn-on of output drivers.

1.2.2 Device-Specific Features

In addition to the core features, non-core features are provided by specific COP8 devices. These features are:

- 8-bit Data Segment Address Register (S Register), used for addressing data memory beyond the first 128 bytes of RAM
- One or more additional 16-bit general-purpose timers with high speed capability.
- Multi-Input Wakeup/Interrupt feature, which provides additional inputs for interrupts or to exit the HALT or IDLE mode
- Brownout
- Watchdog and clock monitor
- NMI non-maskable interrupt
- Full-duplex, double-buffered USART (Universal Synchronous/Asynchronous Receiver/Transmitter) for serial communication
- Analog-to-Digital (A/D) Converter with eight single-ended or four differential-pair channels, using successive approximation

1.3 DEVICE NAMING CONVENTIONS

COP8 Flash Family devices described in this manual follow a naming convention.

An example of a device named according to the naming conventions is the COP8CBR9. The letters “COP8” mean for “8-bit Control-Oriented Processor”; this is the same for all family members. The first two letters “CB” indicate the feature set for the device. The letter “R” indicates the ROM size (A=1k, B=2k, C=4k, E=8k, F=12k, G=16k, H=20k, K=24k, L=28k, and R=32k). The “9” indicates the program memory type (5 = masked ROM, 7 = OTP/EPROM and 9 = Flash). Additional characters are appended to “COP8CBR9” to specify the number of pins, package type, temperature range, and customer ROM code.

The general term “COP8” refers to all devices in the COP8 Flash Family of microcontrollers. In some cases, a lower-case “x” refers to a general class of devices. For example, the term “COP8SAx” represents the COP8SAA7, COP8SAB7, and COP8SAC7. These three devices are all the same except for the OTP ROM and RAM sizes.

Chapter 2

ARCHITECTURE

2.1 INTRODUCTION

Each microcontroller in the COP8 Flash Family contains all program memory and data memory internally. In addition, it contains on-chip configurable I/O ports, an on-chip timer, and a built-in MICROWIRE/PLUS interface. The presence of on-chip memory and peripherals allows the COP8 microcontroller to provide a single-chip solution for many applications.

The COP8 memory organization is based on the “Harvard” architecture, in which the program memory is distinct from the data memory. Each of these two types of memory has its own physical memory space, and uses its own internal address bus. The advantage of this type of organization is that accesses to program memory and data memory can take place concurrently, reducing overall execution time. By contrast, in the “Von Neumann” architecture, program memory and data memory share the same address bus, and concurrent accesses cannot occur.

Except for the Accumulator (A) and Program Counter (PC), all registers, I/O ports, and RAM are memory mapped in the data memory address space. Among these registers are the B Register, X Register, Stack Pointer (SP), and I/O port registers. All such registers can be accessed by reading or writing their memory addresses.

The COP8 architecture provides one enhancement to the Harvard architecture: an instruction called Load Accumulator Indirect (LAID), which allows access to data tables stored in program memory. A conventional Harvard architecture does not allow this.

The COP8 device communicates with other devices through several configurable I/O ports or through the MICROWIRE/PLUS serial I/O interface. The I/O ports are designated by letter names, such as Port C, Port D, Port G, Port I, and Port L.

A 16-bit general-purpose timer is provided in all COP8 microcontrollers, together with two associated 16-bit autoload/capture registers. The timer can be configured to operate in any of three modes: Pulse Width Modulation (PWM), external event counter, or input capture mode.

A maximum of fifteen different interrupts are available in the COP8: two non-maskable interrupts and 13 maskable interrupts. All interrupts cause a branch to the same location in program memory. A special instruction (VIS) may be placed at this location to force an automatic branch to the highest priority-interrupt service routine.

2.2 BLOCK DIAGRAM

A block diagram of the COP8 Flash Family architecture is shown in Figure 2-1. All COP8 Family devices contain the elements pictured in the block diagram. These elements include: the Arithmetic Logic Unit (ALU), Data Memory, Program Memory, Timer 1, MICROWIRE/PLUS, Port I/O, and Interrupt Logic. Functional blocks not common to all COP8 Flash Family members are not shown in Figure 2-1. Block diagrams of individual devices are shown in the device-specific chapters of this manual.

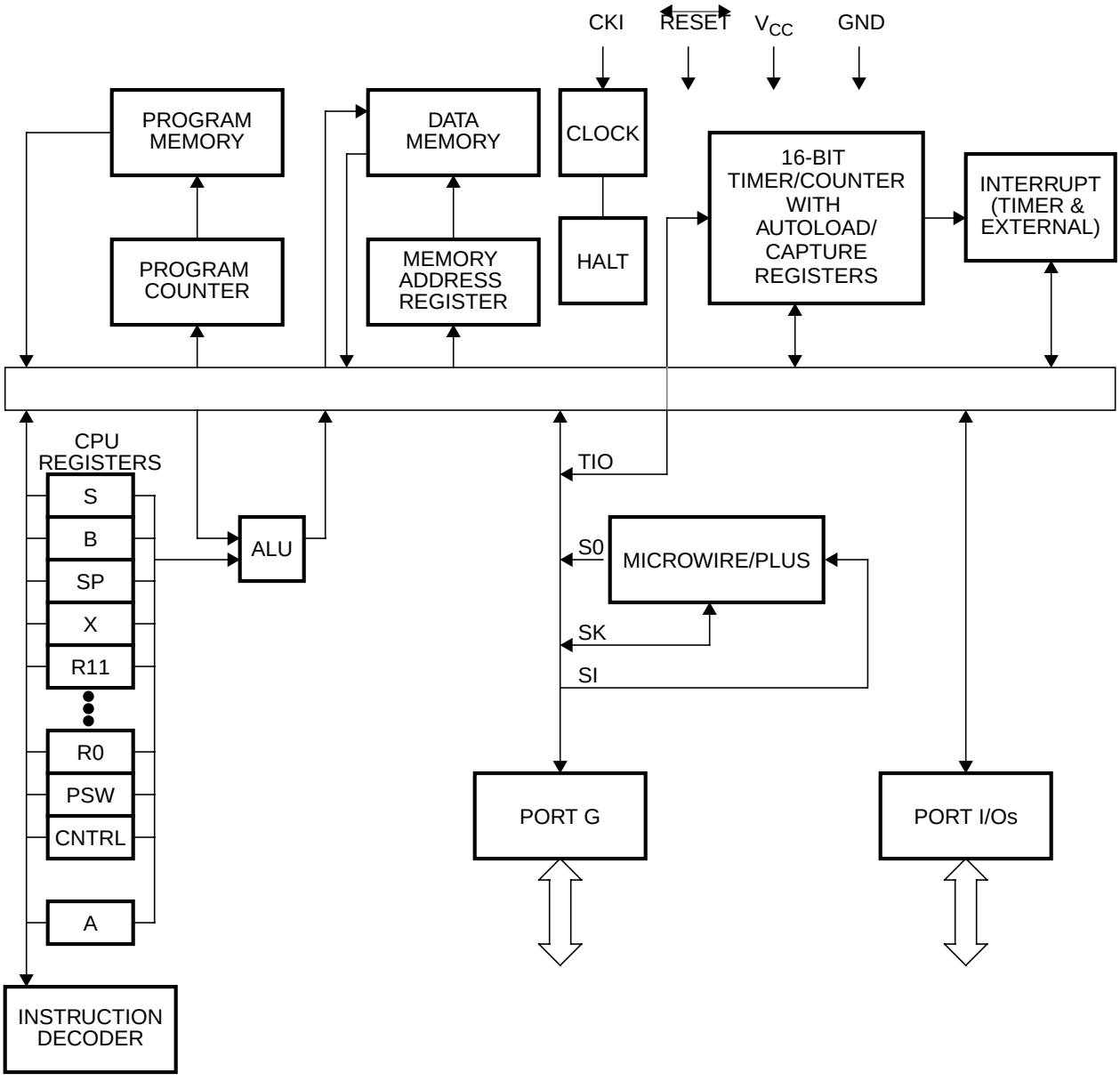


Figure 2-1 COP8 Block Diagram

2.3 MEMORY ORGANIZATION

The COP8 microcontrollers are based on a modified Harvard-style architecture. This type of architecture separates the control program memory from the data memory. Each memory type has its own address space, address bus and data bus. The following sections describe the memory structure.

2.3.1 Program Memory

The COP8 program memory is a block of byte-wide non-volatile ROM or EPROM memory. The program memory addressing range is 32 Kbytes. A 15-bit Program Counter (PC) is used to address the program memory, which is subdivided into 4-Kbyte segments with respect to certain instructions. The program memory may hold program instructions or constant data.

The 4-Kbyte segment divisions within the program memory affect the economical 2-byte Jump Absolute (JMP) and Jump Subroutine (JSR) instructions. These instructions cause the lower 12-bits of the PC to be replaced by the value specified in the instruction while the upper three bits remain unchanged. Thus, these instructions branch only within the currently addressed 4-Kbyte program memory segment.

The indirect instructions, Jump Indirect (JID) and Load Accumulator Indirect (LAID), operate only within a program memory block of 256 bytes. This restriction exists because only the lower eight bits of the PC (PCL) are replaced during program memory table lookups. The upper seven bits of the PC (PCU) remain unchanged. Replacing only the PCL minimizes the execution time of this instruction. Programmers must ensure that LAID/JID instructions and their associated tables do not cross the 256-byte program memory boundaries.

The very economical Jump Relative Short (JP) instruction is completely independent of all program memory block and memory segment boundaries. This single-byte JP instruction allows a branch forward of up to 32 locations or backwards of up to 31 locations relative to the current contents of the program counter. A branch forward of 1 is not allowed, since this may be implemented with a NOP.

2.3.2 Data Memory

All COP8 family members have read/write data memory. Some sections of the data memory space are reserved for the CPU registers, I/O registers, and control registers, all of which are memory mapped. Other sections of the data memory contain RAM and/or EEPROM, which can be used by the application program. The amount of available RAM or EEPROM memory varies from one family member to another. For information on the quantity and type of data memory, see the device-specific chapters later in this manual.

Data memory can be accessed either directly by an address specified in the instruction, or indirectly using the X, SP, or B pointer registers. In all cases, the data memory location is specified as a single byte. Thus, one of 256 memory locations is specified for a data memory access. In the most basic COP8 devices, only the first 256 bytes of data memory are used. In other COP8 devices, a data segment extension register (S register) extends the data memory address range to 32 Kbytes. Data memory extension using the S register is explained in the next section of this manual, Data Segment Extension.

In basic devices that do not use data segment extension, there is a single 256-byte segment of data memory, divided into smaller segments as shown in Figure 2-2. There are 128 bytes of RAM, occupying two non-contiguous address spaces: 112 bytes of lower memory, from 00 to 6F Hex, and 16 bytes at the top of the memory, from F0 to FF Hex. Most of the remaining upper part of this memory, from B0 through EF Hex, is used for the I/O and control registers required by the timers, ports, MICROWIRE interface, CPU core, and optional COP8 peripherals (USART, comparator, etc.). The remaining parts of the 256-byte memory segment (70-7F Hex) are not used.

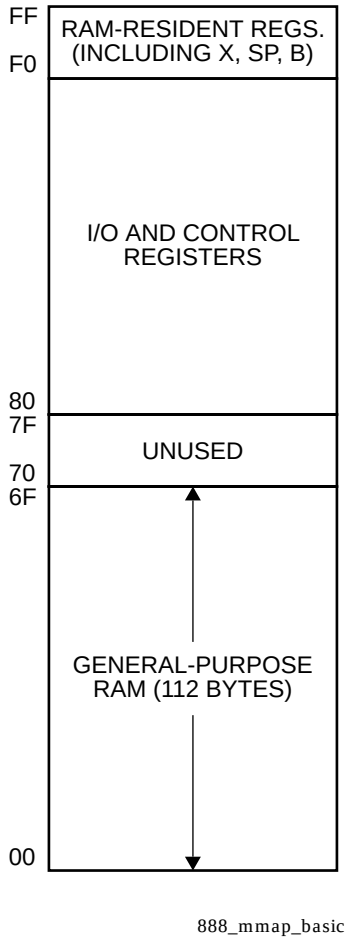


Figure 2-2 Basic Memory Map

The lower, 112-byte segment of RAM is general-purpose read/write memory that is available to the application program. Upon reset, the stack pointer is initialized to the top of this segment (6F Hex), and the stack grows downward from that address as items are pushed onto the stack. Memory from 00 Hex up to the stack can be used for any purpose by the application program. The first 16 addresses (00-0F Hex) have special significance when used with certain instructions (such as LD B,#), because then the instructions are single byte and take only one instruction cycle to execute, rather than two bytes and two instruction cycles.

The upper, 16-byte segment of RAM at the top of memory is used for the RAM-resident registers. The X, SP, and B pointer registers are mapped into memory locations FC, FD, and FE Hex, respectively. The S register, if used, is mapped into address FF Hex. The remaining 12 register locations are available to the application program for any purpose.

Certain COP8 instructions (such as DRSZ) work only with this 16-byte segment of memory. Certain other instructions (such as LD MD,#) are more efficient when used with this 16-byte segment than with other RAM memory locations.

There is at least one segment of unused data: 16 bytes from 70 to 7F Hex. Reading from this unused segment returns FF Hex. Reading from other unused segments returns unknown data.

All RAM, I/O ports, and registers (except A and PC) are mapped into the data memory address space. Table 2-1 shows a basic Register Memory Map for all COP8 devices. Refer to the device-specific chapters for complete memory maps of individual devices.

Table 2-1 Data Memory Map

Address	Contents
00–6F	On-chip RAM Address Space
70–7F	On-chip Data Memory Address Space (Return all '1's)
80–CF	I/O and Register Address Space
D0	Port L Data Register
D1	Port L Configuration Register
D2	Port L Input Pins (read only)
D3	Reserved for Port L
D4	Port G Data Register
D5	Port G Configuration Register
D6	Port G Input Pins (read only)
D7	Reserved for Port I Input Pins (read only)
D8	Port C Data Register
D9	Port C Configuration Register
DA	Port C Input Pins (read only)
DB	Reserved for Port C
DC	Port D Data Register
DD-DF	Reserved for Port D
E0–E5	Reserved
E6	Timer T1 Autoload Register T1RB Lower Byte
E7	Timer T1 Autoload Register T1RB Upper Byte
E8	1CNTRL Register
E9	MICROWIRE/PLUS Shift Register
EA	Timer T1 Lower Byte
EB	Timer T1 Upper Byte
EC	Timer T1 Autoload Register T1RA Lower Byte
ED	Timer T1 Autoload Register T1RA Upper Byte
EE	CNTRL Control Register
EF	PSW Register

Table 2-1 Data Memory Map

Address	Contents
F0–FB	On-chip RAM mapped as Registers
FC	X Register
FD	SP Register
FE	B Register
FF	S Register

Data Segment Extension

In the most basic COP8 devices, there are 128 bytes of RAM residing in two segments within the 256-byte data memory, as described in the previous section. In COP8 devices having more than 128 bytes of RAM, some of the RAM occupies memory above the first 256 addresses (above address FF Hex).

To allow the program to access the RAM residing above this address, the data segment extension register, or “S register,” is used. A 16-bit address is made by combining the 8-bit S register with the normal 8-bit memory address specified by the program instruction. The S register is the high-order byte, and the normal 8-bit memory address is the low-order byte. The S register can be accessed via address FF Hex, and can be read or written like any other register.

The data segment extension feature only works for the lower 128 bytes of each 256-byte memory segment. Therefore, only the lower 128 bytes are used for general-purpose RAM in each 256-byte memory page. The upper 128 bytes are occupied by the same set of registers in all 256-byte memory pages. This concept is more easily understood by looking at the memory map for devices with data segment extension, shown in Figure 2-3.

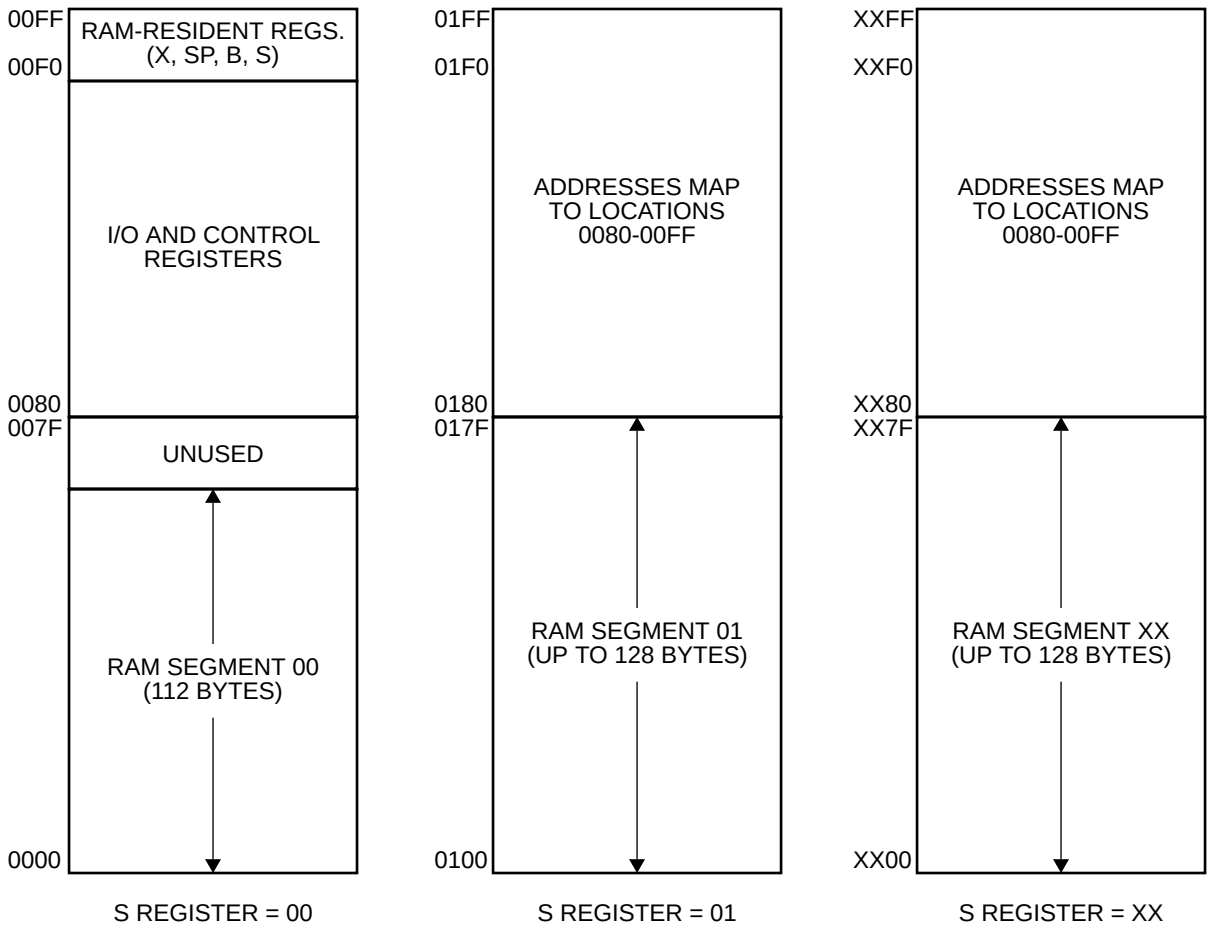
Upon reset, the S register is cleared to zero, and the memory map is the same as for a device without the data segment extension feature. However, if the value 01 is written to the S register, a data memory access instruction specifying an address between 00 and 7F Hex will access the RAM residing at 0100-017F Hex, also called RAM Segment 01. Similarly, if the value 02 is written to the S register, a data memory access between 00 and 7F Hex will access RAM at 0200-027F Hex (RAM Segment 02), and so on. An access to the upper half of any 256-byte memory segment (from 80 to FF Hex), regardless of the S register contents, will always access the same set of registers as a device that does not use data segment extension.

Additional memory beyond the minimum of 128 bytes, if available, resides in the additional RAM segments, starting with Segment 01. For example, the COP888CG has 192 bytes of RAM, or 50% more than the minimum of 128 bytes. The additional 64 bytes of RAM reside in the address range of 0100-013F Hex, or the bottom half of RAM Segment 01. The COP888EG, which has 256 bytes of RAM, has an additional 128 bytes residing in the address range of 0100-017F Hex, thus filling all of RAM Segment 01. A COP8 device having more than 256 bytes of RAM begins to fill Segment 02, an so on, up to the theoretical limit of 32K bytes in 256 segments. Note that each additional 128 bytes of memory fill a contiguous 128-byte segment, unlike the first 128 bytes of RAM in Segment 00.

The software can write to the S register (address xxFF Hex) at any time to change from one memory address segment to another. All addressing modes are available, no matter which segment is chosen. Note that the S register does not need to be changed in order to access registers residing between 80 and FF Hex, because the S register is ignored in that range. Also, note that the Stack Pointer (SP) always points to memory in Segment 00 (starting with address 6F upon reset), regardless of what is contained in the S register. Therefore, the stack must always be stored in Segment 00.

2.3.3 Virtual EEPROM

The Flash memory and the User ISP functions (see Chapter 11, IN SYSTEM PROGRAMMING AND VIRTUAL E2), provide the user with the capability to use the flash program memory to back up user defined sections of RAM. This effectively provides the user with the same nonvolatile data storage as EEPROM. Management, and even the amount of memory used, are the responsibility of the user, however the flash memory read and write functions have been provided in the boot ROM.



888_mmap_data_seg

Figure 2-3 Memory Map with Data Segment Extension

2.3.4 Memory-Mapped I/O Registers

The COP8 devices have three different types of ports: reconfigurable input/output, dedicated output, and dedicated input. Every I/O port has specific memory mapped I/O registers and/or addresses associated with it, depending on the port type. The following sections describe the I/O port register structure for each port type.

NOTE: All port registers and pins are memory mapped in the data memory address space. Therefore, instructions which operate on data memory also operate on port registers and pins. This includes instructions used to set, reset and test individual bits. The I/O register addresses for specific ports are listed in the memory map shown in Table 2-1.

Reconfigurable Input/Outputs

Reconfigurable input/output ports have two associated port registers: a port configuration register and a port data register. These two memory mapped registers allow the port pins to be individually configured as either inputs or outputs, and to be individually changed back and forth in software.

The configuration register is used to configure the pins as inputs or outputs. A pin may be configured as an input by writing a 0 or as an output by writing a 1 to its associated configuration register bit. If a pin is configured as an output, the associated data register bit represents the state of the pin (1 = logic high, 0 = logic low). If the pin is configured as an input, the associated data register bit selects whether the pin is a weak pull-up or Hi-Z input. Table 2-2 shows the port configuration options. The port configuration and data registers are all read/write registers.

A third data memory address is assigned to each I/O port. Reading this memory address returns the value of the port pins regardless of how the pins are configured.

Table 2-2 I/O Port Configuration

Configuration Bit	Data Bit	Port Pin Setup
0	0	Hi-Z input (TRI-STATE output)
0	1	Input with pull-up (weak one output)
1	0	Push-pull zero output
1	1	Push-pull one output

Dedicated Outputs

Dedicated output ports have one associated port register. This memory mapped output data register is used to set the port pins to a logic high or low. A port pin may be individually configured logic high or low by writing a one or zero, respectively, to its associated data register bit. Port data registers may be read or written.

Dedicated Inputs

Dedicated input ports have no associated port registers. However, a data memory address is assigned to the port pins for reading of the port input. Port pin addresses are read-only memory locations.

2.4 CORE REGISTERS

All COP8 microcontrollers share a common block of logic referred to as the COP8 Core. This core includes the COP8 Central Processing Unit (CPU), the Timer 1 Block, the MICROWIRE/PLUS block, and the Interrupt Block. The registers contained within these blocks are the core registers. The registers include: a 15-bit program counter (PC), an 8-bit accumulator (A), a processor status word (PSW), two core control registers (CNTRL, ICNTRL), sixteen 8-bit data memory registers, one 16-bit timer, two 16-bit autoload capture registers, and one 8-bit shift register. All core registers are memory mapped into the data memory address space except for the program counter (PC) and accumulator. The following sections describe in detail the COP8 core registers.

2.4.1 Accumulator

All COP8 family parts have a single 8-bit accumulator. The accumulator is used in all arithmetic and logical operations, such as ADD and XOR. In addition, it is used with the exchange, JID and LAID instructions. The arithmetic and logical instructions use the accumulator as both an operand and result register. A second operand register, if required, is either the instruction register (IR), which contains immediate data or a register in data memory.

2.4.2 Program Counter

The CPU contains a 15-bit program counter used in addressing the byte-wide program memory. The PC is initialized to zero at reset and is incremented once for each byte of an instruction opcode. Jumps, jump subroutines, interrupts, and the JID instruction cause some or all of the PC bits to be replaced. Transfer-of-control instructions that replace only some of the PC bits have a limited jumping range.

2.4.3 Control Registers

The COP8 core contains three 8-bit control registers (PSW, CNTRL and ICNTRL). The following paragraphs and tables show the bits contained in each register. The functions of these bits are described in later chapters.

PSW Register (Address xxEF Hex)

Table 2-3 PSW Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

The Processor Status Word (PSW) register contains eight different flag bits. The PSW register bits are assigned as follows:

HC	Half-Carry Flag
C	Carry Flag
T1PNDA	Timer T1A Interrupt Pending
T1ENA	Timer T1A Interrupt Enable
EXPND	External Interrupt Pending
BUSY	MICROWIRE busy shifting flag
EXEN	External Interrupt Enable
GIE	Global Interrupt Enable

CNTRL Register (Address xxEE Hex)

Table 2-4 CNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

The control register (CNTRL) contains the MICROWIRE/PLUS, External interrupt, and Timer 1 control flags. The CNTRL register bits are assigned as follows:

T1C3	Timer T1 Mode Control Bit
T1C2	Timer T1 Mode Control Bit
T1C1	Timer T1 Mode Control Bit
T1C0	Timer T1 Start/Stop Control in Timer Modes 1 and 2
MSEL	Selects G5 and G4 as MICROWIRE signals SK and SO, respectively
IEDG	External interrupt edge polarity (0 = rising edge, 1 = falling edge)
SL1 & SL0	Select the MICROWIRE clock divide-by (00=2,01=4,1x=8)

ICNTRL Register (Address xxE8 Hex) e

Table 2-5 ICNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FLAG1	LPEN	T0PND	T0EN	μWPND	μWEN	T1PNDB	T1ENB

The interrupt control register (ICNTRL) contains the MICROWIRE/PLUS Interrupt enable and pending flags, the Timer 1 Source B Interrupt enable and pending flags, the general purpose Peripheral Interrupt enable and pending flags, and two spare bits. In most COP8 family members, the Peripheral Interrupt enable and pending flags are used

for the Idle Timer interrupt enable and pending flags, T0EN and T0PND, respectively. If a family member does not have an Idle Timer, these flags are used for other purposes. In COP8 devices that support the Multi-Input Wakeup feature on Port L, one of the spare flags (FLAG2) is used for the Port L Interrupt Enable flag. Many COP8 devices do not use the second spare flag (FLAG1). In these devices, FLAG1 is reserved for future use and should not be set by the user program. Refer to the device specific chapters for information on the assignment of the ICNTRL bits for individual COP8 devices. The ICNTRL register bits are assigned as follows:

FLAG1	General Purpose Flag (reserved)
LPEN	Enable Port L interrupts
T0PND	Idle Timer interrupt pending
T0EN	Idle Timer interrupt enable
µWPND	MICROWIRE/PLUS Interrupt Pending
µWEN	MICROWIRE/PLUS Interrupt Enable
T1PNDB	Timer T1B Interrupt Pending
T1ENB	Timer T1B Interrupt Enable

Data Registers

The COP8 contains sixteen 8-bit data registers, located in data memory from address 00F0 to 00FF Hex. Four of these registers, 00FC through 00FF Hex, have special functions. Locations 00FC and 00FE Hex contain the 8-bit data memory pointers X and B respectively. Location 00FD contains the 8-bit stack pointer (SP) for data memory. Location 00FF is reserved for the data segment extension register. This register is used in some COP8 devices to extend data memory beyond 128 bytes. In devices which contain 128 or fewer bytes of data memory, this register is reserved. The remaining twelve registers, 00F0 through 00FB, are always available for general-purpose use.

Certain COP8 instructions differentiate data registers from other data memory locations, such as the DRSZ (decrement register skip if zero) instruction, DRSZ subtracts one from a specified data register and skips the following instruction if the result of the decrement is zero. This instruction is extremely useful in constructing code loops, and makes the data registers ideal choices for loop counters. Other instructions like the “load memory with immediate data” are more efficient when used with the register memory than when used with the general data memory.

Stack Pointer

The stack pointer (SP) is memory mapped at data memory location 00FD Hex. The stack pointer is automatically initialized during Reset to point to location 06F Hex.

Pushing addresses onto the stack causes the stack to grow downward in data memory toward address zero. Popping addresses off the stack causes the stack to shrink upward. If the stack pointer is initialized to the top of the base segment of memory (06F Hex), over-popping the stack causes a Software Trap error interrupt. The lower limit of the stack is address 0000 Hex. Over-pushing the stack causes the stack to wrap around to addresses 00FF and 00FE Hex (subroutine calls and interrupts cause a double-byte push). This should be avoided because it interferes with the B pointer, which is memory mapped at location 00FE Hex.

The user program may initialize the stack pointer anywhere in the base segment of memory. The stack still grows down toward address zero, but the stack no longer has the Software Trap interrupt over-pop protection. Initializing the stack pointer to one of the upper base segment data register addresses (00F0 to 00FB Hex) is potentially very hazardous. The available stack memory is severely limited, and if the stack pushes downward beyond address location 00F0, interference occurs with the PSW and CNTRL control registers, which are memory mapped at address locations 00EF and 00EE Hex, respectively.

Data Memory Pointers (Index Registers)

The COP8 contains two special-purpose registers, X and B, which may be used as pointers. These registers allow indirect addressing of all locations mapped in the data memory address space. In addition, these registers may be automatically incremented or decremented by certain instructions that use register indirect addressing. The auto-incrementing and auto-decrementing features allow the user program to easily step through data memory locations (i.e., tables).

Data Segment Extension Register

In COP8 family members that have more than 128 bytes of RAM, the data memory register located at FF Hex is used as the Data Segment Extension Register (S). Refer to Section 2.3.2 for more information on this register.

2.4.4 MICROWIRE/PLUS Register

The MICROWIRE/PLUS three-wire serial communication system contains an 8-bit memory-mapped serial shift register (SIOR). The serial data input and output signals to the SIOR register are supplied by SI and SO, respectively. The shift register is clocked by the signal SK. Data is shifted through the SIOR from the low-order end to the high-order end on the falling edge of the SK clock signal.

2.4.5 Timer Registers

The COP8 core contains one timer block. The timer block consists of a 16-bit timer/ counter with two associated 16-bit autoload/capture registers. The 16-bit registers and timer are each organized as two 8-bit memory mapped registers. The upper and lower byte addresses for the memory mapped timer and autoload/capture registers are shown in the data memory address map (Table 2-1).

2.5 CPU OPERATION

This section describes the operation of the Central Processing Unit (CPU). A brief description of the control logic and Arithmetic Logic Unit is given at the beginning of this section. The remainder of this section describes how the microcontroller performs memory fetches, executes instructions, and handles interrupt/error conditions. A block diagram of the main elements that interface with the control logic and ALU is shown in Figure 2-4.

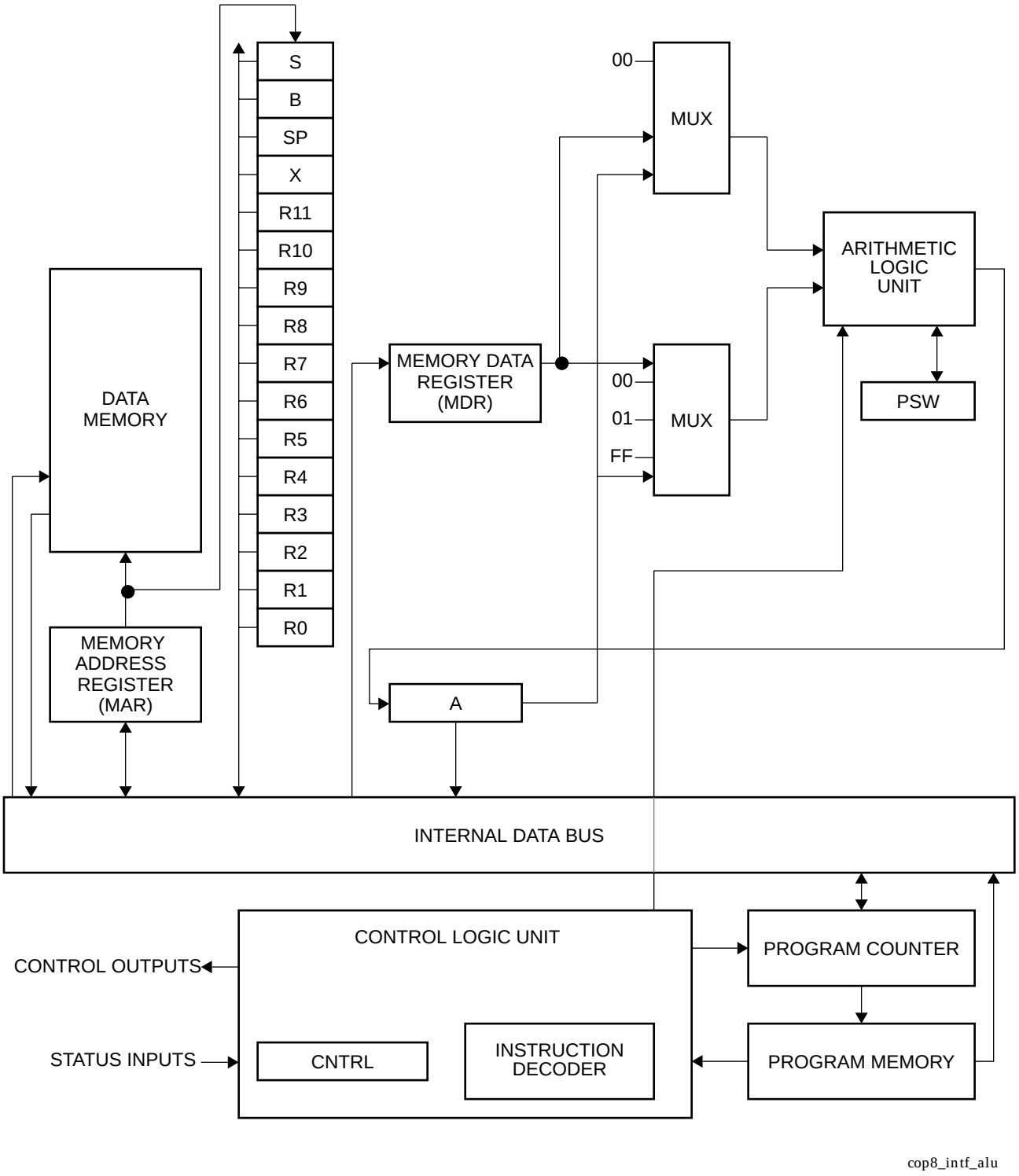


Figure 2-4 Control Logic and ALU Interface

Control Logic

The control logic handles virtually all operations within the device. It includes the program counter, the memory address register, the processor status word register, and the instruction register for storing information. It also includes logic for directing

memory fetches, instruction decoding and execution, and interrupt/error handling. It receives inputs from the ALU and on-chip peripherals including the timer(s) and the MICROWIRE/PLUS interface, and generates control signals for these and other parts of the device.

Arithmetic Logic Unit (ALU)

The ALU performs all logical and arithmetic operations. Inputs to the ALU are provided by the accumulator, several hard-wired data constants, the carry/half-carry bits and the memory data register (MDR). The ALU inputs for a given instruction are specified in the instruction opcode. The accumulator functions as both a source and destination for the ALU, and is used in **all** logical and arithmetic instructions. It always contains the result of the last executed logical, arithmetic, or load/exchange accumulator instruction. The hard-wired data constants, which include 0000, 0001, and 00FF Hex are used in instructions like CLR A, INC A, and DEC A. These instructions have an implicit addressing mode. The carry (C) and half-carry (HC) bits are used in instructions like ADC and SUBC. All arithmetic and logical instructions with two operands use the MDR as one input to the ALU. The MDR may be loaded with operands from data memory or the instruction register (immediate data specified in an instruction opcode). Since only one MDR exists, arithmetic and logical instructions can not be performed directly on two operands from data and/or program memory. Such operations require one operand from memory to be loaded into the accumulator prior to execution.

2.5.1 Memory Fetches

The following two sections describe the manner in which the COP8 accesses data and program memory. Memory access time greatly affects total instruction execution time, and is therefore an important element in understanding the COP8 microcontroller timing.

Data Memory Fetches

All data memory accesses are performed using the internal memory address register (MAR). The contents of MAR selects the location within the data memory address space to be read/written by the current instruction. It should be noted that Memory Direct to Memory Direct data transfers and operations are not supported by the instruction set.

The MAR is loaded with the contents of the B pointer during the last instruction cycle of all instructions. Therefore, instructions which use the Register B Indirect mode of addressing are extremely efficient. This is because the address of the memory location to be accessed during an instruction is already present in the MAR at the start of the instruction. Instructions which use Memory Direct addressing or Register X Indirect addressing to access data memory require an extra one or two instruction cycles to fetch and load the desired memory address into the MAR before the actual instruction is executed.

Some instructions which use Memory Direct addressing are more efficient when addressing the data registers located between 00F0 and 00FF Hex. This is because the complete memory address of the register is contained in the first byte of the instruction opcode. This allows the MAR to be loaded with the new address in the first instruction

cycle of the instruction. Instructions which do not access data memory do not affect the MAR. During the execution of instructions which use the ALU and an operand from data memory, the contents of the memory location addressed by the MAR is loaded into the memory data register (MDR) before being fed into the ALU.

Program Memory Fetches

All program memory accesses are performed using the 15-bit program counter (PC). This includes accesses to program memory for table lookups. At any given time, the PC addresses one byte within program memory. This byte is loaded into the instruction register for decoding, or used as immediate or memory address data. All data/opcode fetches cause the PC to be incremented automatically, so that the PC typically points to one program memory location ahead of the current instruction byte being executed. This allows pre-fetching of opcodes. This is also the reason why table lookup instructions (LAID, JID) located at the last byte within a 256-byte program memory page cause fetches from program memory locations in the following 256-byte page. (The JID and LAID instructions replace the lower eight bits of the PC, and rely on the current upper seven bits of the PC to form the complete address for table lookups. However, the upper seven bits of the PC change when the PC is automatically incremented over a page boundary.)

2.5.2 Instruction Decoding and Execution

All instruction decoding is performed by the CPU control logic. Single-byte opcodes require a single memory fetch. Therefore, many single-byte opcodes are single-cycle. Multiple-byte opcodes require more than one program memory fetch; the first byte is decoded to determine the number of program fetches needed to complete the instruction. Only one program memory fetch can be performed during a single instruction cycle. Therefore, an instruction always requires at least as many instruction cycles to execute as number of opcode bytes.

NOTE: Data and program memory fetches may be performed in the same instruction cycle due to the Harvard-style architecture of the COP8 Family.

The instruction cycle clock (t_C) always equals one-tenth the frequency of the clock signal at the CKI pin. All instructions are executed in multiples of the instruction cycle clock period.

During the last cycle of an instruction, the next instruction's first byte is always fetched from program memory. In addition, the PC is always incremented. This means that at the start of the first cycle of an instruction, the opcode for that instruction is already in the IR and the PC is pointing to the next instruction byte. In order to generate skips (non-execution of an instruction), the microcontroller Skip Logic is activated. This prevents an instruction (already located in the IR) from being executed by the microcontroller. Skipped instructions require X number of cycles to be skipped, where X equals the number of bytes in the skipped instruction's opcode.

The exact number of instruction cycles required for an instruction to execute can be found in Section A.7. As noted previously, memory fetches (and therefore addressing modes) greatly influence instruction execution time. In order to optimize instruction execution time, the programmer should pay special attention to these items when developing code.

The following sections explain the steps performed by the control logic when executing different instructions.

One-Cycle Instructions

During the single cycle of one of these instructions, the following steps are performed:

1. The instruction is decoded and executed. (The instruction opcode is already in the IR at the start of the instruction cycle due to pre-fetching).
2. The next instruction is fetched from program memory.
3. The PC is incremented.

Two-Cycle Instructions

A two-cycle instruction has either a one-byte or two-byte opcode. These instructions each fall into one of five instruction categories: logical, arithmetic, conditional, exchange or load. The operations performed during two-cycle instructions are given below.

The logical, arithmetic and conditional instructions that use the Immediate addressing mode each have the following steps:

Cycle 1: Decode the opcode for the instruction. Fetch the immediate data from program memory. Execute the instruction. Activate Skip Logic if necessary. Increment the PC. (The logical/ arithmetic or conditional instruction is complete at the end of this instruction cycle.)

Cycle 2: Fetch the first byte of the next instruction. Increment the PC.

Two-cycle load and exchange accumulator instructions, and load memory indirect using the B pointer have these steps:

Cycle 1: Decode the opcode for the instruction. If necessary, fetch the immediate data from program memory and increment the PC. Execute the instruction. (The load or exchange is complete at the end of this instruction cycle.)

Cycle 2: If necessary, increment or decrement the B pointer. Load the contents of the B pointer into MAR. Fetch the first byte of the next instruction. Increment the PC.

Three-Cycle Instructions

The device has nine three-cycle load and exchange instructions. A generic overview of the sequence of steps performed by the COP8 in executing these instructions is given below.

Cycle 1: Decode the opcode for the instruction. If necessary, fetch the memory direct address from program memory and increment the PC. Load the MAR with the address of the data memory location to be accessed (either the address fetched from program memory or the contents of the X pointer depending on the instruction). If necessary, increment or decrement the X pointer.

Cycle 2: If necessary, fetch the immediate data from program memory and increment the PC. Execute the instruction. (The load or exchange is complete at the end of this instruction cycle.)

Cycle 3: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

The remaining three-cycle instructions are all unique. Therefore, the sequence of events is given separately for each.

JP Instruction

Cycle 1: At the beginning of this instruction cycle, the PC is one count ahead of the address of the JP instruction. Decode the instruction opcode. Add the lower six bits of the contents of the IR (the JP opcode) to the lower byte of the PC.

Cycle 2: If the offset contained in the JP opcode was positive and the add performed in Cycle 1 had a carry out (overflow), increment the upper byte of the PC. If the offset was negative and no carry out was produced by the add in Cycle 1(underflow), decrement the upper byte of the PC.

Cycle 3: Fetch the first byte of the next instruction (instruction located at the branch address). Increment the PC.

JMP Instruction

Cycle 1: Decode the instruction opcode. Fetch the lower byte of the branch address from program memory. Load the lower byte of the PC with the fetched address.

Cycle 2: Load the four least significant bits of the JMP opcode stored in the IR into the four least significant bits of the upper byte of the PC.

Cycle 3: Fetch the first byte of the next instruction (instruction located at the branch address). Increment the PC.

LAID Instruction

Cycle 1: Decode the instruction opcode. Exchange the lower byte of the PC with the contents of the accumulator.

Cycle 2: Fetch the byte from program memory addressed by the PC. Transfer the contents of the accumulator back to the lower-byte of the PC. Store the fetched byte in the accumulator.

Cycle 3: Fetch the first byte of the next instruction. Increment the PC.

JID Instruction

Cycle 1: Decode the instruction opcode. Exchange the lower byte of the PC with the contents of the accumulator.

Cycle 2: Fetch the byte from program memory addressed by the PC. Transfer the contents of the PC back to the accumulator (restore the contents of the ACC). Store the fetched byte in the lower-byte of the PC.

Cycle 3: Fetch the first byte of the next instruction. Increment the PC.

DRSZ Instruction

Cycle 1: Decode the opcode of the instruction. Load the MAR with the address of the register being decremented.

Cycle 2: Decrement the contents of the register addressed by the MAR. If the result is zero, activate the Skip Logic.

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

PUSH Instruction

Cycle 1: Decode the opcode of the instruction. Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 2: Load the memory location addressed by MAR (the first available stack location) with the contents of the accumulator.

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

POP Instruction

Cycle 1: Decode the opcode of the instruction. Increment the Stack Pointer to point to the last entry in stack. Load the MAR with the address contained in the Stack Pointer.

Cycle 2: Load the accumulator with the contents of the memory location addressed by the MAR (last stack entry).

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

Four-Cycle Instructions

All four-cycle instructions except JMPL use the Memory Direct addressing mode. The following steps outline the general sequence of events performed during the execution of these memory direct instructions.

Cycle 1: Decode the Memory Direct mode opcode prefix. Fetch the memory direct address from program memory and store it in the MAR. Increment the PC.

Cycle 2: Fetch the actual opcode from program memory and store it in the IR.

Cycle 3: Execute the instruction. (The bit manipulation, conditional test, or logical/arithmetic operation is complete at the end of this instruction cycle.)

Cycle 4: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

A JMPL has the following steps:

Cycle 1: Decode the JMPL opcode. Fetch the second byte of the instruction (the high-order byte of the branch address) and store it in IR. Increment the PC.

Cycle 2: Fetch the third byte of the instruction (the low-order byte of the branch address) and load it into the lower byte of the PC.

Cycle 3: Load the high-order byte of the branch address from the IR into the upper byte of the PC.

Cycle 4: Fetch the next instruction (located at the branch address). Increment the PC.

Five-Cycle Instructions

The COP8 has six five-cycle instructions; JSR, JSRB, JSRL, VIS, RET, RETF, RETI and RETSK. All of these instructions force program branches.

The COP8 performs the following steps during the JSR, JSRB and JSRL instructions:

Cycle 1: Decode the opcode for the instruction. Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack locatioN. If JSRL, fetch the next byte of the instruction and increment the PC.

Cycle 2: Increment the PC. Push the low-order byte of the return address onto the stack (store at the location addressed by MAR). Fetch the next byte of the instruction. Load the low-order byte of the subroutine address (addressed by the MAR) into the PC.

Cycle 3: Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 4: Push the high-order byte of the return address onto the stack (store at the location addressed by MAR). If JSR, load the four bits of the high-order byte of the subroutine address stored in the IR into the PC. If JSRB, load zero into the high order byte of the PC and enable the boot ROM. If JSRL, load the seven bits of the high-order byte of the subroutine address stored in the IR into the PC.

Cycle 5: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

The COP8 performs the following steps during the VIS instruction:

Cycle 1: Decode the opcode for the instruction. Load the low-order byte of the PC with the low-order byte of the address of the location of the vector which corresponds the highest priority pending flag. The high-order byte of the PC already contains the high-order byte of the address of the location of the vector since the vector table must reside within the same 256-byte block as the VIS instruction. (The VIS instruction may reside in the last location of the 256-byte block located above the vector table. In this case, the incrementing of the PC at the end of the previous instruction increments the high-order byte of the PC so that it is pointing to the correct block at the time the VIS instruction is actually executed.)

Cycle 2: (At the start of this cycle, the PC is pointing to the high-order byte of the interrupt vector.) Fetch the high-order byte of the interrupt vector from program memory and load it into

the instruction register (IR). Increment the PC to point to the low-order byte of the interrupt vector.

Cycle 3: Fetch the low-order byte of the interrupt vector from program memory and load it into the low-order byte of the PC.

Cycle 4: Transfer the contents of the IR to the high-order byte of the PC.

Cycle 5: Fetch the first byte of the next instruction. Increment the PC.

The COP8 performs the following steps during the RET, RETF, RETSK, and RETI instructions:

Cycle 1: Decode the opcode for the instruction. Increment the stack pointer to point to the last entry on the stack. Load the MAR with the address of the last entry in the stack (the address in the updated SP).

Cycle 2: Pop the high byte of the return address off the stack (the contents of the memory location addressed to by the MAR). Load the upper byte of the PC with the high byte of the return address.

Cycle 3: Increment the stack pointer to point to the next byte of data on the stack. Load the MAR with the address of the last entry in the stack (the address in the updated SP).

Cycle 4: Pop the low-byte of the return address off the stack (the contents of the memory location addressed to by the MAR). Load the lower byte of the PC with the low byte of the return address. If RETF, disable the boot ROM.

Cycle 5: Load the contents of the B pointer into the MAR. If RETI, set the GIE bit. If RETSK, activate skip logic to skip the instruction at the return address. Fetch the first byte of the instruction at the return address. Increment the PC.

Seven-Cycle Instructions

The Software Trap is the only instruction which requires seven cycles to execute. This instruction is performed when a 00 opcode (INTR) is loaded into the Instruction Register. The execution of this instruction when an interrupt is not pending is considered an error. Refer to Section 2.5.3 for information on the execution of this instruction.

2.5.3 Interrupt and Error Handling

The COP8 microcontrollers have a maximum of 16 interrupt sources. All interrupts force a jump to location 00FF Hex in program memory. Therefore, all interrupt and error handling routines/branches should be located at 00FF Hex.

The CPU forces a jump to 00FF Hex by jamming the INTR opcode (00) into the IR upon detecting an interrupt. The detection of an error (Software Trap) is the result of the INTR opcode being loaded into the IR as a part of the normal program sequence. An interrupt that occurs while an instruction is being executed is not acknowledged until the end of the current instruction. If the instruction following the current instruction is to be skipped, the next instruction is skipped before the pending interrupt is acknowledged and the INTR opcode is jammed into the IR. The COP8 requires seven cycles to execute the INTR instruction.

Cycle 1: Decode 00 opcode. If not a Software Trap, reset the GIE bit. Decrement the lower byte of the PC. (Note: The address of the instruction that was ready to be executed is the return address, this needs to be saved on the stack. However, the PC is one count ahead of the current instruction, and therefore must be decremented before being saved on the stack.)

Cycle 2: If the decrementing of the lower byte of the PC caused a borrow, decrement the upper byte of the PC.

Cycle 3: Load the MAR with the address of the first available stack location (the address currently in SP). Increment the stack pointer to point to the next byte of data on the stack.

Cycle 4: Push the low-order byte of the return address onto the stack (store at the location addressed by MAR). Load the low-order byte of the PC with 0FF Hex.

Cycle 5: Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 6: Push the high-order byte of the return address onto the stack (store at the location addressed by MAR). Load the upper byte of the PC with 00 Hex.

Cycle 7: Load the contents of the B pointer into the MAR. Fetch the first byte of the instruction located at 00FF Hex. Increment the PC.

Once a branch to location 00FF Hex occurs, the programmer can use the VIS instruction or poll the available pending flags to determine the source of the interrupt. Refer to Chapter 3 for more information on interrupts and the Software Trap.

2.6 RESET

The COP8 enters a reset state immediately upon detecting a logic low on the $\overline{\text{RESET}}$ pin. The $\overline{\text{RESET}}$ pin must be held low for a minimum of one instruction cycle to guarantee a valid reset. When the $\overline{\text{RESET}}$ pin is pulled to a logic high, the device begins code execution within two instruction cycles.

2.6.1 Reset Initialization

All COP8 microcontrollers contain logic to initialize their internal circuitry during the reset state. The following initializations are performed at reset:

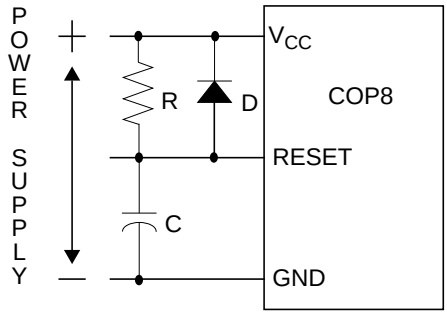
- The Program Counter is loaded with 0000 Hex.
- All bits of the PSW, CNTRL and ICNTRL registers are reset. This disables all interrupts, stops Timer 1, and disables MICROWIRE/PLUS.
- The SP is initialized to 6F Hex.

The accumulator, all data memory, and registers (including the B and X pointers) are not initialized during Reset. Refer to the device-specific chapters for details on the reset initialization of registers not found in the COP8 microcontroller core.

2.6.2 Reset Requirements Upon Power-Up

During power-up initialization, the external circuitry must ensure that the $\overline{\text{RESET}}$ pin is held low until the COP8 device is within the specified V_{CC} voltage range. In addition, if a crystal oscillator or resonator is used, the design must ensure that the oscillator has enough time to stabilize before the $\overline{\text{RESET}}$ pin is pulled high.

A Resistor-Capacitor (RC) network such as the one shown in Figure 2-5 can be used to ensure that the $\overline{\text{RESET}}$ pin is held low long enough while the power supply is being turned on. The RC time constant should be at least five times the expected power supply rise time, and at least 15 microseconds in any case.



RC > 5 X Power Supply Rise Time or
15 microseconds (whatever is greater)

Figure 2-5 External Reset RC Network

2.6.3 On-Chip Brownout Reset

When enabled, the device generates an internal reset as V_{CC} rises. While V_{CC} is less than the specified brownout voltage (V_{bor}), the device is held in the reset condition and the Idle Timer is preset with 00Fx (240 - 256 t_c). When V_{CC} reaches a value greater than V_{bor}, the Idle Timer starts counting down. Upon underflow of the Idle Timer, the internal reset is released and the device will start executing instructions. This internal reset will perform the same functions as external reset. Once V_{CC} is above the V_{bor} and this initial Idle Timer time-out takes place, instruction execution begins and the Idle Timer can be used normally. If, however, V_{CC} drops below the selected V_{bor}, an internal reset is generated, and the Idle Timer is preset with 00Fx. The device now waits until V_{CC} is greater than V_{bor} and the countdown starts over. When enabled, the functional operation of the device is guaranteed down to the V_{bor} level.

One exception to the above is that the brownout circuit will insert a delay of approximately 3mS on power up or any time the V_{CC} drops below a voltage of about 1.8V. The device will be held in Reset for the duration of this delay before the Idle Timer starts counting the 240 to 256 t_C. This delay starts as soon as the V_{CC} rises above the trigger voltage (approximately 1.8V). This behavior is shown in Figure 2-6.

In Case 1, V_{CC} rises from 0 volts and the on-chip RESET is undefined until the supply is greater than approximately 1.0V. At this time the brownout circuit becomes active and holds the device in RESET. As the supply passes a level of about 1.8V, a delay of about 3mS (t_d) is started and the Idle Timer is preset to a value between 00F0 and 00FF (hex).

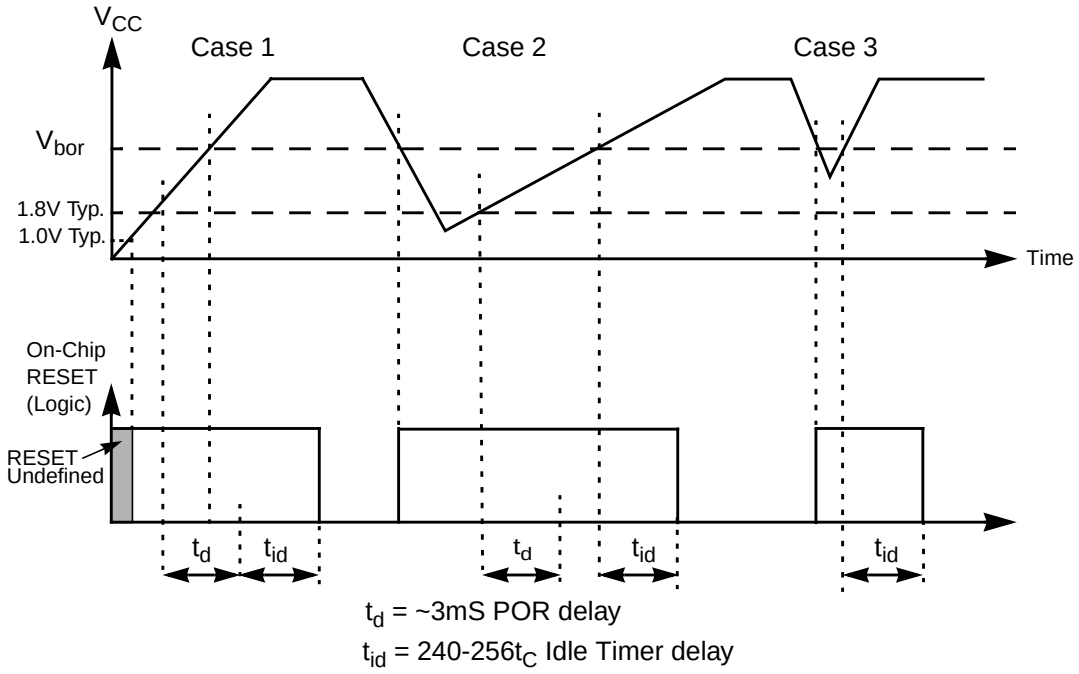


Figure 2-6 Brownout Reset Operation

Once V_{CC} is greater than V_{bor} and t_d has expired, the Idle Timer is allowed to count down (t_{id}).

Case 2 shows a subsequent dip in the supply voltage which goes below the approximate 1.8V level. As V_{CC} drops below V_{bor}, the internal RESET signal is asserted. When V_{CC} rises back above the 1.8V level, t_d is started. Since the power supply rise time is longer for this case, t_d has expired before V_{CC} rises above V_{bor} and t_{id} starts immediately when V_{CC} is greater than V_{bor}.

Case 3 shows a dip in the supply where V_{CC} drops below V_{bor}, but not below 1.8V. On-chip RESET is asserted when V_{CC} goes below V_{bor} and t_{id} starts as soon as the supply goes back above V_{bor}.

If the Brownout Reset feature is enabled, the internal reset will not be turned off until the Idle timer underflows. The internal reset will perform the same functions as external reset. The device is guaranteed to operate at the specified frequency down to the specified brownout voltage. After the underflow, the logic is designed such that no additional internal resets occur as long as V_{CC} remains above the brownout voltage.

The device is relatively immune to short duration negative-going V_{CC} transients (glitches). It is essential that good filtering of V_{CC} be done to ensure that the brownout feature works correctly.

Refer to the device specifications for the actual V_{bor} voltages.

Under no circumstances should the $\overline{\text{RESET}}$ pin be allowed to float. If the on-chip Brownout Reset feature is being used, the $\overline{\text{RESET}}$ pin should be connected directly to V_{CC}. The $\overline{\text{RESET}}$ input may also be connected to an external pull-up resistor or to other external circuitry. The output of the brownout reset detector will always preset the Idle

timer to a value between 00F0 and 00FF(240 to 256t_c). At this time, the internal reset will be generated.

If the BOR feature is disabled, then no internal resets are generated and the Idle Timer will power-up with an unknown value. In this case, the external RESET must be used. When BOR is disabled, this on-chip circuitry is disabled and draws no DC current.

The contents of data registers and RAM are unknown following the on-chip reset.

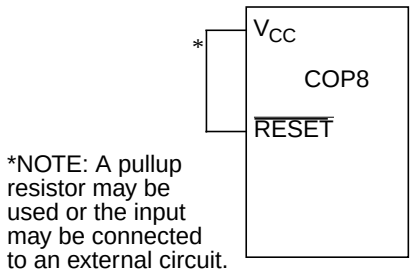


Figure 2-7 Reset Circuit using Power-On Reset

2.7 CLOCK OPTIONS

The COP8 device is driven by a clock signal with a fixed frequency. The instruction clock operates at one-tenth the rate of the internal device clock. For example, if the internal device clock frequency is 10 MHz, the instruction clock frequency is 1 MHz. On devices which include an on-chip clock doubler, the internal device clock frequency will be twice that of the oscillator frequency.

Either one or two device pins are used for device clocking, depending on the device and its configuration. The CKI (Clock Input) pin is a dedicated clock pin in all types of devices. The CKO (Clock Output) pin is used for clocking in some configurations. If it is not used for that purpose, it can be used as the G7 (Port G, bit 7) input pin or HALT/Restart input pin.

Some COP8 devices have a built-in, on-chip resistor-capacitor (RC) network for implementing the clock oscillator, while others do not. Devices with the built-in RC network let you use choose between the internal RC oscillator, an external crystal oscillator, or an externally generated clock signal to operate the device. In devices without the built-in RC network, you can build the oscillator using either an external RC network or an external crystal oscillator. The device-specific chapters and the device data sheets provide information on the availability of the on-chip RC network.

You generally must pre-select the desired type of clock oscillator by selecting a ROM mask option. Selection of a specific clock option affects the operating frequency, clocking accuracy, and power consumption of a particular device. Refer to the device-specific data sheets to obtain detailed information on frequency ranges, power consumption, and component values for the different oscillator circuits.

2.7.1 Devices Without an On-Chip RC Network

In devices without an on-chip RC network, you can choose to implement the clock oscillator either as an external crystal network as shown in Figure 2-8, or an external RC network as shown in Figure 2-9. The crystal oscillator uses both clock pins (CKI and CKO), whereas the RC oscillator uses only the CKI pin, allowing the G7/CKO pin to be used for other purposes. See the device data sheet for the information on the component values to use in the external network.

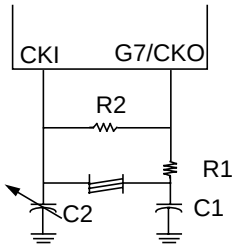


Figure 2-8 Crystal Oscillator Circuit

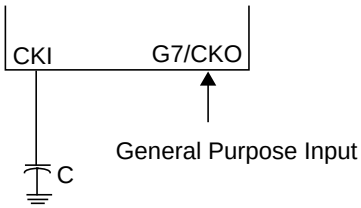


Figure 2-9 RC Oscillator Circuit

There are some exceptions to the general option of using an external crystal or RC network. For example, the COP8CBR/CCR/CDR9 only allow a crystal oscillator to be used. For specific information, see the “Mask Options” section in the applicable device-specific chapter.

2.7.2 Devices With an On-Chip RC Network

In devices with an on-chip RC network, you can choose to implement the clock oscillator in one of the following ways:

- crystal oscillator with on-chip bias resistor (see Figure 2-10)
- crystal oscillator with external bias resistor (see Figure 2-11)
- external clock signal (see Figure 2-12)
- internal RC oscillator (see Figure 2-13)

When you specify the ROM mask or program the OTP ROM with the application program, you specify one of these four clock oscillator options. The choice depends on cost considerations and the oscillator precision requirements. A crystal oscillator offers the

most precise and stable clock, whereas the built-in RC oscillator is the most economical implementation.

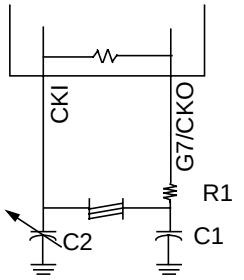


Figure 2-10 Crystal Oscillator with On-Chip Bias Resistor

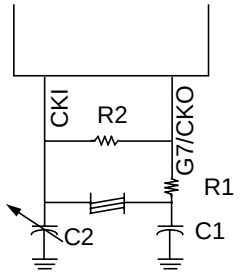


Figure 2-11 Crystal Oscillator with External Bias Resistor

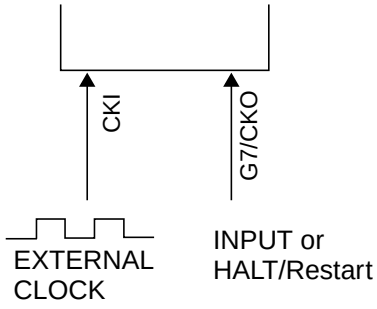


Figure 2-12 External Clock Signal

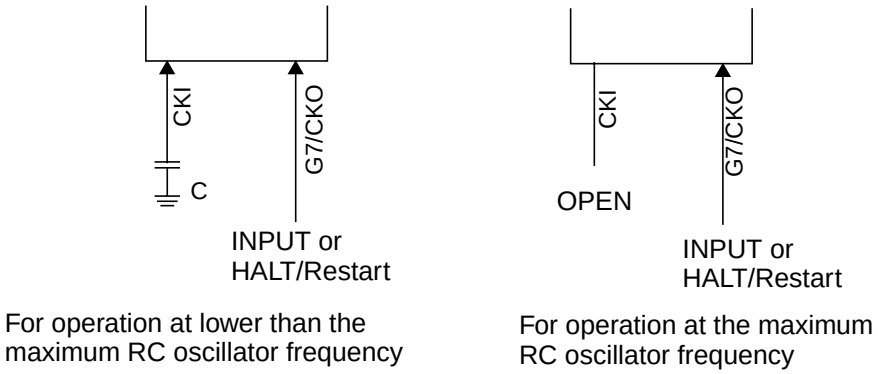


Figure 2-13 Internal RC Oscillator

If you build the oscillator using an external crystal, both the CKI and CKO pins are used to implement the clock. You can either use an on-chip bias resistor as shown in Figure 2-10 or an external one as shown in Figure 2-11. Using the on-chip resistor is more economical, but does not let you specify the resistor value. See the device data sheet for information on component values.

If you use an external clock signal to drive the device as shown in Figure 2-12, only the CKI pin is used, which makes the G7/CKO pin available for other purposes. The external clock signal must meet the AC and DC specifications provided in the device data sheet, such as voltage levels, duty cycle, and rise/fall times.

The most economical implementation uses the internal RC oscillator, as shown in Figure 2-13. Only the CKI pin is used, which makes the G7/CKO pin available for other purposes. If you leave the CKI pin unconnected and floating, the RC oscillator operates at its maximum allowed frequency, which is typically 5 MHz. If you want the device to operate at a slower clock rate, you connect a capacitor between the CKI input and ground. The larger the capacitor, the slower the clock. See the device data sheet for detailed information on operating frequencies and capacitor values.

Chapter 3

INTERRUPTS

3.1 INTRODUCTION

An interrupt is an event that temporarily stops the normal flow of program execution and causes a separate interrupt service routine to be executed. After the interrupt has been serviced, execution continues with the next instruction in the program that would normally have been executed following the point of interruption.

The architecture of the COP8 Flash Family supports up to 15 different interrupts. The actual number of interrupts in a device depends on the individual device type. For example, devices that have a USART also have interrupts associated with the USART, while other devices lack that type of interrupt. The interrupts are vectored, meaning that a set of vectors (memory location pointers) stored in program memory directs the processor to one of several interrupt service routines. A special instruction called VIS (Vector Interrupt Select) directs the processor to the vector location based on the cause of the interrupt.

The various types of interrupts can be divided into two categories, maskable and non-maskable. A maskable interrupt can be enabled or disabled by the software, allowing the software to determine whether or not an occurrence of the event will trigger an interrupt. A non-maskable interrupt cannot be masked, and will always trigger an interrupt, except in certain cases when a non-maskable interrupt is already being serviced.

Figure 3-1 is a block diagram that illustrates the internal interrupt logic of a hypothetical COP8 device. The interrupt logic of actual COP8 devices is very similar to what is shown in the figure, except that the source of interrupts differ from one device type to another. The names along the left side of the diagram represent the events that can cause an interrupt. Some of these events originate from on-chip functions, while others come from off-chip devices connected to the COP8.

An interrupt is serviced when the interrupt signal is activated (the output of the OR gate on the right-hand side of the illustration). The occurrence of any one event can trigger an interrupt. The event can occur at any time: before, during, or after an instruction cycle. An occurrence immediately latches a flag bit called the pending flag. The chip hardware checks the interrupt logic at the start of each instruction, and a pending interrupt is acknowledged and handled at that time if the interrupt is enabled.

The first two interrupts, Software Trap and NMI, are both non-maskable and are always enabled. (All COP8 devices have a Software Trap, but not all have the NMI interrupt.) The remaining interrupts are all maskable. In order for a maskable interrupt event to trigger an interrupt, its own individual interrupt enable bit must be set, and the GIE (Global Interrupt Enable) bit must also be set in the PSW register. All of the maskable interrupts can be disabled together by clearing the GIE bit.

The different types of interrupts are organized by rank. If two or more interrupts are detected at the same time, the interrupt arbitration logic of the device determines which

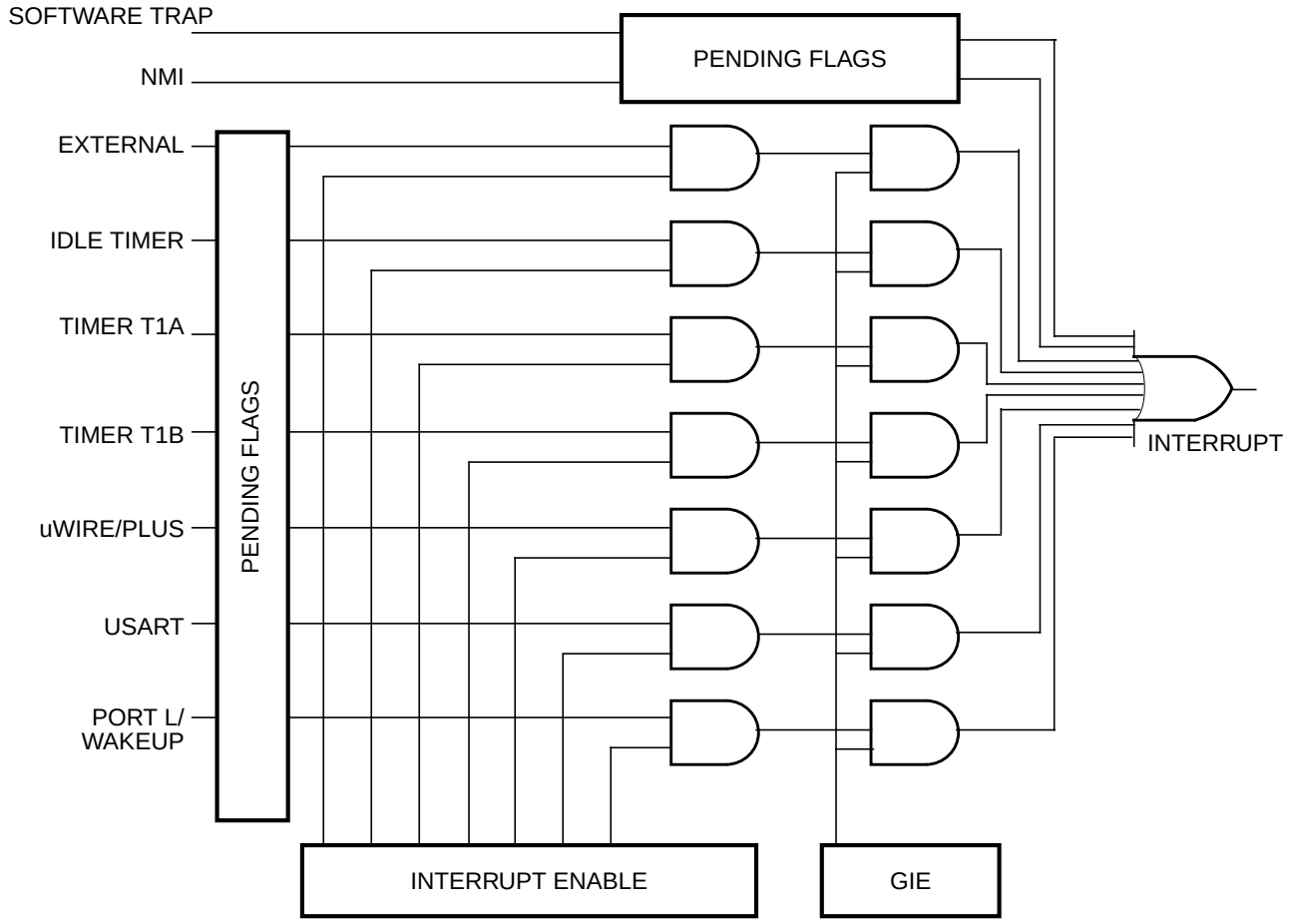


Figure 3-1 COP8 Interrupt Block Diagram

interrupt has the highest priority, and that interrupt is serviced first. The interrupts in Figure 3-1 are shown in order of rank, starting from the highest-priority interrupt, Software Trap.

3.2 VIS INSTRUCTION AND VECTOR TABLE

The general interrupt service routine, which starts at address 00FF Hex, must be capable of handling all types of interrupts. The VIS instruction, together with an interrupted vector table, directs the microcontroller to the specific interrupt handling routine based on the cause of the interrupt.

VIS is a single-byte instruction, typically used at the very beginning of the general interrupt service routine at address 00FF Hex, or shortly after that point, just after the code used for context switching (as described in Section 3.3). The VIS instruction determines which enabled and pending interrupt has the highest priority, and causes an indirect jump to the address corresponding to that interrupt source. The jump addresses (“vectors”) for all possible interrupt sources are stored in a vector table.

The vector table is 32 bytes long and resides at the top of the 256-byte block containing the VIS instruction. However, if the VIS instruction is at the very top of a 256-byte block (such as at 00FF Hex), the vector table resides at the top of the next 256-byte block. Thus, if the VIS instruction is located somewhere between 00FF and 01DF Hex (the usual case), the vector table is located between addresses 01E0 and 01FF hex. If the VIS instruction is located between 01FF and 02DF Hex, then the vector table is located between addresses 02E0 and 02FF Hex, and so on.

Each vector is 15 bits long and points to the beginning of a specific interrupt service routine somewhere in the 32-Kbyte memory space. Each vector occupies two bytes of the vector table, with the higher-order byte at the lower address. The vectors are arranged in order of interrupt priority. The vector of the maskable interrupt with the lowest rank is located at 0yE0 (higher-order byte) and 0yE1 (lower-order byte). The next priority interrupt is located at 0yE2 and 0yE3, and so forth in increasing rank. The NMI has the second highest rank and its vector is always located at 0yFC and 0yFD. The Software Trap has the highest rank and its vector is always located at 0yFE and 0yFF.

Table 3-1 shows the source of interrupts in a hypothetical COP8 device, the interrupt arbitration ranking, and the locations of the corresponding vectors in the vector table. A similar table can be found in the device-specific chapters for each of the COP8 devices. Those tables are similar to this one, except for slight differences in the list of interrupt types.

The vector table should be filled by the user with the memory locations of the specific interrupt service routines. For example, if the Software Trap routine is located at 0310 Hex, then the vector location 01FE–01FF should contain the data 03 and 10 Hex; the high address byte is stored at the lower address. When a Software Trap interrupt occurs and the VIS instruction is executed, the program jumps to the address specified in the vector table, 0310 Hex.

The interrupt sources shown in Table 3-1 are listed in order of rank, from highest to lowest priority. If two or more enabled and pending interrupts are detected at the same time, the one with the highest priority is serviced first. Upon return from the interrupt service routine, the next highest-level pending interrupt is serviced. Interrupts labelled “Reserved” in the table are not used in the device, but are reserved for future use should another feature be added that can generate an interrupt.

If the VIS instruction is executed, but no interrupts are enabled and pending, the lowest-priority interrupt vector is used, and a jump is made to the corresponding address in the vector table. This is an unusual occurrence and may be the result of an error. It can legitimately result from a change in the enable bits or pending flags prior to the execution of the VIS instruction, such as executing a single cycle instruction which clears an enable flag at the same time that the pending flag is set. It can also result, however, from inadvertent execution of the VIS command outside of the context of an interrupt.

The default VIS interrupt vector can be useful for applications in which time critical interrupts can occur during the servicing of another interrupt. Rather than restoring the program context (A, B, X, etc.) and executing the RETI instruction, an interrupt service routine can be terminated by returning to the VIS instruction. In this case, interrupts will be serviced in turn until no further interrupts are pending and the default VIS routine is started. After testing the GIE bit to ensure that execution is not erroneous, the

Table 3-1 Interrupt Vector Table

Arbitration Rank	Interrupt Description	Vector Address[a]
1	Software Trap	01FE–01FF
2	NMI	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	(Reserved)	01EE–01EF
10	(Reserved)	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	(Reserved)	01E6–01E7
14	(Reserved)	01E4–01E5
15	Port L/Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1

a. The location of the vector table depends on the location of the VIS instruction. The vector addresses shown in the table assume a VIS instruction between 00FF and 01DF Hex.

routine should restore the program context and execute the RETI to return to the interrupted program.

This technique can save up to fifty instruction cycles (t_c), or more, (50 µs at 10 MHz cpu clock) of latency for pending interrupts with a penalty of fewer than ten instruction cycles if no further interrupts are pending.

3.3 CONTEXT SWITCHING

In some applications, the first action that should be carried out in the interrupt service routine is to save the contents of the registers so that the “context” of the microcontroller can be restored just before returning from the interrupt. The register values can be saved on the stack. When interrupt servicing is finished, the register values can be popped from the stack and restored to the registers, allowing program execution to proceed with the original register values.

For example, if registers A, X, and B are used within the interrupt service routines, the following assembly code can be used to save the contents of those registers.

```
          .=00FF          ;Start at interrupt address
SAVE:     PUSH      A     ;Push Accumulator contents onto stack
          LD        A,B
          PUSH      A     ;Push B pointer onto stack
          LD        A,X
          PUSH      A     ;Push X pointer onto stack
          VIS
```

Upon completion of interrupt servicing, the following routine can be used to restore the context to the registers and return from the interrupt.

```
RESTOR:   POP       A     ;Pop X pointer from stack
          X         A,X   ;Restore X pointer
          POP       A     :Pop B pointer from stack
          X         A,B   ;Restore B pointer
          POP       A     ;Restore Accumulator contents
          RETI            ;Return from interrupt
```

In the example above, the SAVE routine resides at 00FF Hex, the interrupt address, so it is executed at the beginning for any type of interrupt. The VIS instruction causes a jump to a specific interrupt service routine, depending on the interrupt source. Each specific interrupt service routine (except the Software Trap) should end with a jump to the RESTOR routine.

3.4 MASKABLE INTERRUPTS

All interrupts other than the Software Trap and NMI are maskable. Each maskable interrupt has an associated enable bit and pending flag bit. The pending bit is latched to 1 when the interrupt condition occurs. The state of the interrupt's enable bit, combined with the GIE bit, determines whether an active pending flag actually triggers an interrupt. All of the maskable interrupt pending and enable bits belong to memory-mapped control registers, and thus can be controlled by the software.

A maskable interrupt condition triggers an interrupt under the following conditions:

1. The enable bit associated with that interrupt is set.
2. The GIE bit is set.
3. The microcontroller is not processing a non-makable interrupt. (If a non-maskable interrupt is being serviced, a maskable interrupt must wait until that service routine is completed.)

An interrupt is triggered only when all of these conditions are met at the beginning of an instruction. If different maskable interrupts meet these conditions simultaneously, the highest-priority interrupt will be serviced first, and the other pending interrupts must wait.

Upon Reset, all pending bits, individual enable bits, and the GIE bit are reset to zero. Thus, a maskable interrupt condition cannot trigger an interrupt until the program enables it by setting both the GIE bit and the individual enable bit. When enabling an interrupt, the user should consider whether or not a previously activated (set) pending bit should be acknowledged. If, at the time an interrupt is enabled, any previous occurrences of the interrupt should be ignored, the associated pending bit must be reset to zero prior to enabling the interrupt. Otherwise, the interrupt may be simply enabled; if the pending bit is already set, it will immediately trigger an interrupt. A maskable interrupt is active if its associated enable and pending bits are set.

At the start of interrupt acknowledgment, the control logic halts normal program execution by jamming an INTR (00) opcode into the instruction register. As a result, the following actions occur:

1. The GIE bit is automatically reset to zero, preventing any subsequent maskable interrupt from interrupting the current service routine. This feature prevents one maskable interrupt from interrupting another one being serviced.

2. The address of the instruction about to be executed is pushed onto the stack.

3. The program counter is loaded with 00FF Hex, causing a jump to that program memory location. A general interrupt service routine must start at that address.

The COP8 requires seven instruction cycles to perform the actions listed above.

The interrupt service routine stored at location 00FF Hex should use the VIS instruction (described in Section 3.2) to determine the cause of the interrupt, and jump to the interrupt handling routine corresponding to the highest-priority enabled and active interrupt. Alternately, you may choose to poll all interrupt pending and enable bits to determine the source(s) of the interrupt. If more than one interrupt is active, the program must decide which interrupt to service.

It is possible for one interrupt source to trigger an interrupt, and a different interrupt source to be serviced. For example, a low-priority maskable interrupt occurs, setting its pending flag and triggering an interrupt. While the interrupt is being acknowledged (jump to 00FF Hex) or context saving is being performed by the general interrupt service routine, another (higher-priority) maskable event occurs, setting its own pending flag. When the service routine reaches the VIS instruction, the higher-priority event is serviced first, and the lower-priority event that triggered the interrupt must wait. Assuming that no more interrupt events occur, the lower-priority event will be serviced upon return from the interrupt routine of the higher-priority event.

If you wish to allow nested interrupts, the interrupt service routine may set the GIE bit to 1 by writing to the PSW register, and thus allow other maskable interrupts to interrupt the current service routine. If nested interrupts are allowed, caution must be exercised. You must write the program in such a way as to prevent stack overflow, loss of saved context information, and other unwanted conditions.

Within a specific interrupt service routine, the associated pending bit should be cleared. This is typically done as early as possible in the service routine in order to avoid missing the next occurrence of the same type of interrupt event. Thus, if the same event occurs a

second time, even while the first occurrence is still being serviced, the second occurrence will be serviced immediately upon return from the current interrupt routine.

An interrupt service routine typically ends with an RETI instruction. This instruction sets the GIE bit back to 1, pops the address stored on the stack, and restores that address to the program counter. Program execution then proceeds with the next instruction that would have been executed had there been no interrupt. If there are any valid interrupts pending, the highest-priority interrupt is serviced immediately upon return from the previous interrupt.

3.5 NON-MASKABLE INTERRUPTS

The architecture of the COP8 family supports two non-maskable interrupts, known as the Software Trap (INTR instruction) and NMI (Non-Maskable Interrupt pin). All COP8 devices have the Software Trap interrupt, but some devices do not have an NMI pin, and therefore do not allow the use of the NMI interrupt. The device-specific chapters later in this manual indicate whether the NMI interrupt is available.

The non-maskable interrupts do not have individual enable bits, and they are not affected by (and do not affect) the GIE bit. Thus, they cannot be masked out by software. However, they each have a pending flag. While a non-maskable interrupt is being serviced (when its pending flag is set), maskable interrupts are prevented from being acknowledged, but their pending bits can still be set. Pending maskable interrupts are acknowledged upon return from the non-maskable interrupt.

A non-maskable interrupt is acknowledged in essentially the same manner as a maskable interrupt. If the NMI pending flag is found to be set at the beginning of an instruction, the address the of the next instruction that would normally be executed is saved on the stack. If the Software Trap pending flag is found to be set at the beginning of an instruction, the address of the INTR instruction which triggered the Software Trap is saved on the stack rather than the next normally executed instruction. In either case, the program then jumps to address 00FF Hex. The VIS instruction directs the microcontroller to the appropriate interrupt service routine. Note that with non-maskable interrupts, the GIE bit neither has an effect nor is affected by the interrupt; the GIE bit is used only with maskable interrupts.

To return from a non-maskable interrupt routine, the RET (Return from Subroutine) instruction or RETSK (Return from Subroutine and Skip) rather than the RETI (Return from Interrupt) instruction should be used. Doing so preserves the status of the GIE bit. The RET instruction pops the address stored on the stack and places it in the program counter, and execution of the program continues from that point. The RETI instruction does the same thing, but it also sets the GIE bit to re-enable maskable interrupts; it should be used only to return from a maskable interrupt routine.

3.5.1 Non-Maskable Interrupt Pending Flags

There is a pending flag bit associated with each non-maskable interrupt, called STPND for the Software Trap and NMIPND for the NMI interrupt. These pending flags are not memory-mapped and cannot be accessed directly by the software.

The pending flags are reset to zero when a chip Reset occurs. When a non-maskable interrupt event occurs, the associated pending bit is latched to one. When either flag is set, maskable interrupts are inhibited. The interrupt service routine should contain an RPND instruction to reset the pending flag to zero.

The RPND instruction always resets the STPND flag, but resets the NMIPND flag only if the NMI has been acknowledged and the STPND flag is already reset to zero. The reason for this behavior is because of the interaction between the Software Trap and NMI interrupts. A Software Trap can interrupt an NMI routine, but a NMI cannot interrupt a Software Trap routine. Thus, if both the STPND and NMIPND flags are set, it must be because an NMI routine was in progress and was interrupted by a Software Trap. Under these conditions, an RPND instruction in the Software Trap routine resets the STPND flag, thus ending the Software Trap pending condition, but not the NMIPND flag. Upon return from interrupt (if so programmed), a second RPND in the NMI routine will reset the NMIPND flag.

3.5.2 Software Trap

The Software Trap is a special kind of non-maskable interrupt which occurs when the INTR instruction (used to acknowledge interrupts) is fetched from program memory and placed in the instruction register. This can happen in a variety of ways, usually because of an error condition. Some examples of causes are listed below.

- If the program counter incorrectly points to a memory location beyond the available program memory space, the non-existent memory locations are interpreted as the INTR instruction.
- If the INTR instruction is deliberately placed in unused sections of the program memory, and an unexpected condition occurs which causes the code in these sections to be executed, a Software Trap will occur.
- If the stack is popped beyond the allowed limit (address 006F Hex), a Software Trap is triggered.
- A Software Trap can be triggered by a temporary hardware condition such as a brownout or power supply glitch.

The Software Trap has the highest priority of all interrupts. When a Software Trap occurs, the STPND bit is set, which inhibits all other interrupts. Nothing can interrupt a Software Trap service routine except for another Software Trap. The STPND can be reset only by the RPND instruction or a chip Reset.

The Software Trap indicates an unusual or unknown error condition. Generally, returning to normal execution at the point where the Software Trap occurred cannot be done reliably. Therefore, the Software Trap service routine should re-initialize the stack pointer and perform a recovery procedure that re-starts the software at some known point, similar to a chip Reset, but not necessarily performing all the same functions as a chip Reset. The routine must also execute the RPND instruction to reset the STPND flag. Otherwise, all other interrupts will be locked out. To the extent possible, the Software Trap routine should record or indicate the context of the microcontroller so that the cause of the Software Trap can be determined.

If you wish to return to normal execution from the point at which the Software Trap was triggered, the program must first execute RPND, followed by RETSK (Return from Subroutine and Skip) rather than RETI (Return from Interrupt) or RET (Return from Subroutine). This is because the return address stored on the stack is the address of the INTR instruction that triggered the interrupt. You need to skip that instruction in order to proceed with the next one. Otherwise, an infinite loop of Software Traps and returns will occur.

Programming a return to normal execution requires careful consideration. If the Software Trap routine is interrupted by another Software Trap, the second RPND will reset the STPND flag; upon return to the first Software Trap routine, the STPND flag will have the wrong state. To avoid problems such as this, program the Software Trap routine to perform a recovery procedure rather than a return to normal execution.

Under normal conditions, the STPND flag is reset by a RPND instruction in the Software Trap service routine. If a programming error or hardware condition (brownout, power supply glitch, etc.) sets the STPND flag without providing a way for it to be cleared, all other interrupts will be locked out. To alleviate this condition, the software should contain extra RPND instructions in the main program and in the Watchdog service routine (if present). There is no harm in executing extra RPND instructions in these parts of the program.

3.5.3 NMI

The NMI interrupt is a non-maskable interrupt triggered by a positive-going edge on the NMI input pin. (Some COP8 devices have do not have an NMI pin, and therefore do not allow the NMI interrupt to be used.) A positive-going edge latches the NMIPND flag to a 1, and causes subsequent edges on the NMI pin to be ignored. At the beginning of the next instruction, the NMI request is acknowledged unless a Software Trap interrupt is being serviced.

The NMI interrupt has the second-highest priority among all interrupts. An NMI service routine can be interrupted only by a Software Trap and nothing else. If a second NMI request is received on the NMI pin during an NMI routine with the NMIPND flag still set, that second request is ignored and lost.

The NMI service routine should end with an RPND instruction to reset the NMIPND flag, immediately followed by a RET (Return from Subroutine) instruction to return from the interrupt. (RET should be used instead of RETI to preserve the status of the GIE bit.) This two-instruction sequence (RPND-RET) allows any enabled and pending maskable interrupts to be serviced after execution of the RET instruction. The actual clearing of the NMIPND bit is delayed one instruction cycle from the execution of the RPND bit (the RPND is executed during the first instruction cycle of the RET instruction). This allows the RET to be executed first, before allowing any pending interrupt to be acknowledged.

3.5.4 Software Trap and NMI Interaction

In systems where both the Software Trap and NMI can be used, it is important to understand what happens when both types of interrupts occur very close to each other in time, and to understand the behavior of the RPND instruction. RPND always clears the

STPND flag, but clears the NMIPND flag only if the NMI has been acknowledged and the STPND flag is already cleared. The flag clearing performed by the RPND instruction is delayed one instruction cycle from the execution of the instruction.

First consider the case where an NMI service routine is in progress and a Software Trap occurs. The Software Trap is acknowledged, the program jumps to 00FF hex, and the VIS instruction causes a jump to the Software Trap routine, which has the highest priority. At the end of this routine, an RPND instruction clears the STPND flag, but not the NMIPND flag because the user might want to return to the NMI service routine in order to complete it. If the Software Trap routine re-starts the software at a known point (ignore the return address stored on the stack), a second RPND instruction should be used in the Software Trap service routine to clear the NMIPND flag. (Two RPND instructions in a row will first clear the STPND flag and then the NMIPND flag.) The software routine does not know (and cannot find out) whether the NMIPND flag is set, but it is not a problem to have an extra RPND instruction in the routine.

Now consider the case where a Software Trap routine is in progress and an NMI interrupt request is received. The NMI is not acknowledged and must wait until the STPND flag has been cleared. If the Software Trap service routine ends with a RPND followed by a RETSK, the pending NMI will interrupt the CPU right after the RETSK. If the Software Trap routine ends with two RPND instructions as explained in the previous paragraph, the NMI will be acknowledged after the second RPND instruction. The second RPND instruction does not clear the NMIPND flag because at that point, the NMI has not yet been acknowledged.

In summary, the RPND instruction always resets the STPND bit and resets the NMI pending bit only if the following two conditions are satisfied:

1. The ST pending bit is not set.
2. The NMI has already interrupted the CPU (actually caused an interrupt).

3.6 INTERRUPTS AND VIRTUAL E^2 INTERACTION

While executing from Boot ROM (i.e. during user ISP/Virtual E^2 command processing) all maskable interrupts are inhibited regardless of the state of the GIE bit. Interrupts which occur during this time will remain pending and the processor will respond to them immediately on return to flash execution.

3.7 INTERRUPT SUMMARY

The COP8 uses the following types of interrupts, listed below in order of priority:

1. The Software Trap non-maskable interrupt, triggered by the INTR (00 op-code) instruction. The Software Trap is acknowledged immediately. This interrupt service routine can be interrupted only by another Software Trap. The Software Trap should end with two RPND instructions followed by a re-start procedure.

2. NMI non-maskable interrupt, triggered by a positive-edge transition on the NMI input pin (not available on some COP8 devices). An NMI request will interrupt any lower-level interrupt (i.e. any maskable interrupt) in progress. An NMI service routine can be interrupted only by a Software Trap. If another NMI request is received during an NMI service routine, the second NMI request is ignored and lost. The NMI routine should end with the instructions RPND and RET or RPND followed by a re-start procedure.

3. Maskable interrupts, triggered by an on-chip peripheral block or an external device connected to the COP8. The list of maskable interrupts and their priority ranking vary from one device type to another. Under ordinary conditions, a maskable interrupt will not interrupt any other interrupt routine in progress. A maskable interrupt routine in progress can be interrupted by any non-maskable interrupt request (NMI or Software Trap). A maskable interrupt routine should end with an RETI instruction.

Chapter 4

TIMERS

4.1 INTRODUCTION

Each COP8 Flash Family device contains a versatile 16-bit timer/counter that can satisfy a wide range of application requirements. The timer can be configured to operate in any of three modes:

- Processor-independent Pulse Width Modulation (PWM) mode, which generates pulses of a specified width and duty cycle
- External event counter mode, which counts occurrences of an external event
- Input capture mode, which measures the elapsed time between occurrences of an external event

The 16-bit timer/counter is designated Timer T1. Many COP8 devices have more than one of this type of timer, each timer having its own set of registers and I/O pins. Additional timers of this type are designated T2, T3, and so on. See the device-specific chapters for information on the number of timers contained in any particular device.

Many devices also have an IDLE Timer, designated T0, which is used for different purposes: for timing the duration of the IDLE mode, for timing the Watchdog service window, and for timing the start-up delay when exiting the HALT mode. The structure of the IDLE Timer is described in this chapter. Usage of the IDLE timer in different contexts is covered in other chapters of this manual.

4.2 TIMER/COUNTER BLOCK

The section of the device containing the timer circuitry is called the timer/counter block. This block contains a 16-bit counter/timer register, designated T1, and two associated 16-bit autoload/capture registers, designated R1A and R1B. Each 16-bit register is organized as a pair of 8-bit memory-mapped register bytes. The register bytes reside at the following data memory addresses:

Register	Address
T1:	00EA-00EB
R1A:	00EC-00ED
R1B:	00E6-00E7

In each case, the lower byte resides at the lower memory address.

The timer/counter block uses two I/O pins, designated T1A and T1B, which are alternate functions of G3 and G2 (Port G, bits 3 and 2), respectively.

The timer can be started or stopped under program control. When running, the timer counts down (decrements). Depending on the operating mode, the timer counts either instruction clock cycles or transitions on the T1A pin. Occurrences of timer underflows (transitions from 0000 to FFFF) can either generate an interrupt and/or toggle the T1A pin, also depending on the operating mode.

There are two interrupts associated with the timer, designated the T1A interrupt and the T1B interrupt. When timer interrupts are enabled, the source of the interrupt depends on the timer operating mode: either a timer underflow, a transfer of data to or from the R1A or R1B register, or an input signal received on the T1B pin.

4.3 TIMER CONTROL BITS

Timer T1 is controlled by reading and writing eight bits contained within three registers of the CPU core: the PSW (Processor Status Word), CNTRL (Control), and ICNTRL (Interrupt Control) registers. By programming these control bits, the user can enable or disable the timer interrupts, set the operating mode, and start or stop the timer. The control bits operate as described in Tables 4-1 and 4-2.

Table 4-1 Timer Control Bits

Register/Bit	Name	Function
CNTRL/Bit 7	T1C3	Timer T1 control bit 3 (see Table 4-2)
CNTRL/Bit 6	T1C2	Timer T1 control bit 2 (see Table 4-2)
CNTRL/Bit 5	T1C1	Timer T1 control bit 1 (see Table 4-2)
CNTRL/Bit 4	T1C0	Timer T1 run: 1 = Start timer, 0 = Stop timer; or Timer T1 underflow interrupt pending flag in input capture mode
PSW/Bit 5	T1PNDA	T1A interrupt pending flag: 1 = T1A interrupt pending, 0 = T1A interrupt not pending
PSW/Bit 4	T1ENA	T1A interrupt enable bit: 1 = T1A interrupt enabled, 0 = T1A interrupt disabled
ICNTRL/Bit 1	T1PNDB	T1B interrupt pending flag: 1 = T1B interrupt pending, 0 = T1B interrupt not pending
ICNTRL/Bit 0	T1ENB	T1B interrupt enable bit: 1 = T1B interrupt enabled, 0 = T1B interrupt disabled

Table 4-2 Timer Mode Control Bits

Mode	TxC3	TxC2	TxC1	Descritpion	Interrupt A Source	Interrupt B Source	Timer Counts On
1	1	0	1	PWM: TxA Toggle	Autoreload RA	Autoreload RB	t_C
	1	0	0	PWM: No TxA Toggle	Autoreload RA	Autoreload RB	t_C
2	0	0	0	External Event Counter	Timer Underflow	Pos. TxB Edge	TxA Pos. Edge
	0	0	1	External Event Counter	Timer Underflow	Pos. TxB Edge	TxA Neg. Edge
3	0	1	0	Captures: TxA Pos. Edge TxB Pos. Edge	Pos. TxA Edge or Timer Underflow	Pos. TxB Edge	t_C
	1	1	0	Captures: TxA Pos. Edge TxB Neg. Edge	Pos. TxA Edge or Timer Underflow	Neg. TxB Edge	t_C
	0	1	1	Captures: TxA Neg. Edge TxB Pos. Edge	Neg. TxA Edge or Timer Under-flow	Pos. TxB Edge	t_C
	1	1	1	Captures: TxA Neg. Edge TxB Neg. Edge	Neg. TxA Edge or Timer Under-flow	Neg. TxB Edge	t_C

4.4 ADDITIONAL GENERAL-PURPOSE TIMERS

Some COP8 devices have additional timers that operate in exactly the same manner as the T1 timer. Additional timers of this type are designated T2, T3, and so on. The device-specific chapters indicate the number of timers available in each device.

Like the T1 timer, each additional timer has its own set of three 16- bit memory-mapped registers and two I/O pins that are alternate functions of port pins. The eight control bits for each additional timer are grouped together into a single memory-mapped control byte, rather than being distributed between the PSW, CNTRL, and ICNTRL registers as in the case of Timer T1.

The register addresses and other information for the basic timer T1 and additional timers T2 and T3 are listed below.

Timer T1

T1:	Address 00EA-00EB
R1A:	Address 00EC-00ED
R1B:	Address 00E6-00E7
Control bits in CNTRL, PSW, ICNTRL:	Addresses 00EE, 00EF, 00E8
T1C3-T1C2-T1C1-T1C0:	CNTRL bits 7-6-5-4
T1PNDA-T1ENA:	PSW bits 5-4

T1PNDB-T1ENB:	ICNTRL bits 1-0
T1A pin:	Alternate function of G3
T1B pin:	Alternate function of G2

Timer T2 (If present)

T2:	Address 00C0-00C1
R2A:	Address 00C2-00C3
R2B:	Address 00C4-00C5
Control bits in T2CNTRL:	Address 00C6
T2C3-T2C2-T2C1-T2C0:	T2CNTRL bits 7-6-5-4
T2PNDA-T2ENA:	T2CNTRL bits 3-2
T2PNDB-T2ENB:	T2CNTRL bits 1-0
T2A pin:	Alternate function of L4
T2B pin:	Alternate function of L5

Timer T3 (If present)

T3:	Address 00B0-00B1
R3A:	Address 00B2-00B3
R3B:	Address 00B4-00B5
Control bits in T3CNTRL:	Address 00B6
T3C3-T3C2-T3C1-T3C0:	T3CNTRL bits 7-6-5-4
T3PNDA-T3ENA:	T3CNTRL bits 3-2
T3PNDB-T3ENB:	T3CNTRL bits 1-0
T3A pin:	Alternate function of L6
T3B pin:	Alternate function of L7

4.5 TIMER OPERATING SPEEDS

Each of the Tx timers, except for T1, have the ability to operate at either the instruction cycle frequency (low speed) or the internal clock frequency (MCLK). For 10Mhz CKI, the instruction cycle frequency is 2Mhz and the internal clock frequency is 20Mhz. This feature is controlled by the High Speed Timer Control Register, HSTCR. Its format is shown in Table 4-3. To place a timer, Tx, in high speed mode, set the appropriate TxHS bit to 1. For low speed operation, clear the appropriate TxHS bit to 0. This register is cleared to 00 on Reset.

Table 4-3 High Speed Timer Control Register

HSTCR							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit2**	**Bit 1**	**Bit 0**
T9HS	T8HS	T7HS	T6HS	T5HS	T4HS	T3HS	T2HS

4.6 TIMER OPERATING MODES

The timer can be configured to operate in any one of three modes. Within each mode, there are options related to the use of the TxA and TxB I/O pins. (Refer to the device pinout in the Port Descriptions Chapter for the pin assignments of the TxA and TxB alternate functions.)

The Pulse Width Modulation (PWM) mode can be used to generate precise pulses of known width and duty cycle on the TxA pin (configured as an output). The timer is clocked by the instruction clock or the internal clock (MCLK). An underflow causes the timer register to be reloaded alternately from the TxRA and TxRB registers, and optionally causes the TxA output to toggle. Thus, the values stored in the TxRA and TxRB registers control the width and duty cycle of the signal produced on TxA.

The external event counter mode can be used to count occurrences of an external event. The timer is clocked by the signal appearing on the TxA pin (configured as an input). An underflow causes the timer register to be reloaded alternately from the TxRA and TxRB registers.

The input capture mode can be used to precisely measure the frequency of an external clock that is slower than the selected timer clock, or to measure the elapsed time between external events. The timer is clocked by the instruction clock or the internal clock (MCLK). A transition received on the TxA or TxB pin (with the pins configured as inputs) causes a transfer of the timer contents to the TxRA or TxRB register, respectively.

4.6.1 MODE 1. Processor Independent PWM Mode

In the Pulse Width Modulation (PWM) mode, the timer counts down at the instruction clock rate or the internal clock (MCLK). When an underflow occurs, the timer register is reloaded alternately from the TxRA and TxRB registers, and counting proceeds downward from the loaded value. At the first underflow, the timer is loaded from TxRA, the second time from TxRB, the third time from TxRA, and so on.

The timer can be configured to toggle the TxA output bit upon underflow. This results in the generation of a clock signal on TxA with the width and duty cycle controlled by the values stored in the TxRA and TxRB registers. If the same values are stored in TxRA and TxRB, the resulting signal will have a 50% duty cycle; different values will produce a clock signal with other percentage duty cycles. This is a “processor-independent” PWM clock because once the timer is set up, no more action is required from the CPU. In cases where a clock with a duty cycle other than 50% is desired, no interrupt processing is necessary to change the reload values. By contrast, a microcontroller that has only a single reload register requires an interrupt routine to update the reload value (on-time, off-time) on each underflow.

A block diagram of the timer operating in the PWM mode is shown in Figure 4-1.

There are two interrupts associated with the timer, designated TxA and TxB. The two interrupts are individually maskable by the enable bits TxENA and TxENB. Thus, the user can generate an interrupt on the rising edge, on the falling edge, or on both edges of the PWM output (or not at all).

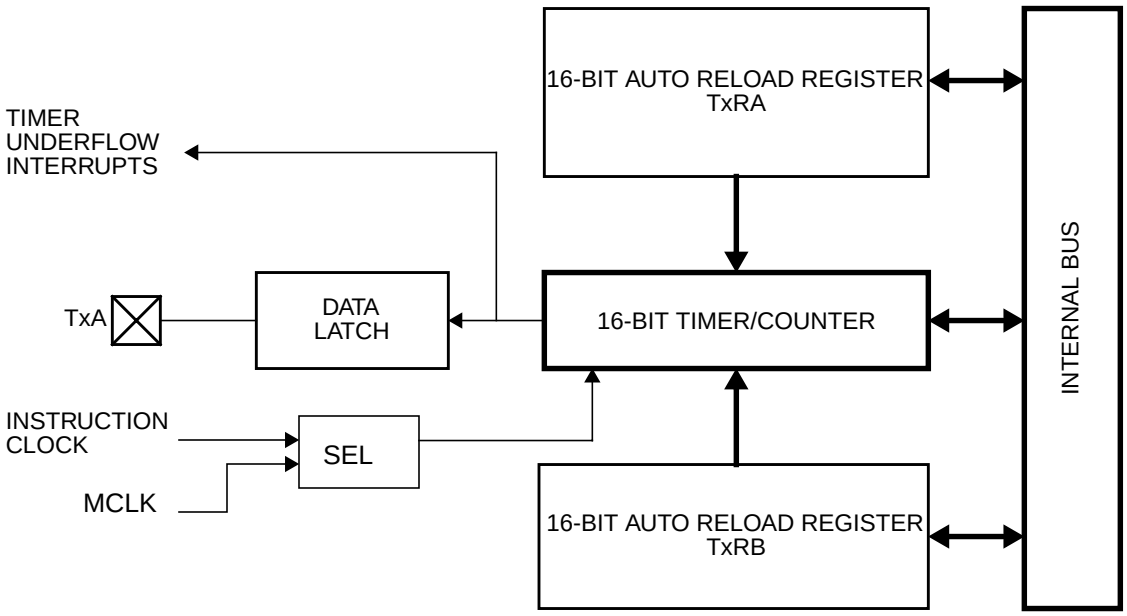


Figure 4-1 Timer in PWM Mode

When an underflow occurs that causes a timer reload from TxRA, the interrupt pending flag bit TxPNDA is set. Similarly, when an underflow occurs that causes a reload from TxRB, the interrupt pending flag bit TxPNDB is set. A CPU interrupt occurs if the corresponding enable bit is set and the GIE (Global Interrupt Enable) bit is also set. The interrupt service routine must reset the pending bit and perform whatever processing is necessary at the interrupt point.

Note: If the timer is running in high speed mode (counting internal MCLK) it is capable of generating interrupts very fast. Once both pending bits are set, any later interrupts received, before the respective pending bit is reset by an interrupt service routine, will be lost. It is up to the user to write software such that these interrupts can be processed without loss.

The following steps can be used to operate the timer in the PWM mode. In this example, the TxA output pin is toggled with every timer underflow, and the “high” and “low” times for the TxA output are set to different values. The TxA output can start out either high or low; the instructions below are for starting with the TxA output high. (Follow the instructions in parentheses to start it low.)

1. Configure the TxA pin as an output by setting the appropriate bit in the configuration register.
2. Configure the timer as either high speed or low speed.
3. Initialize the TxA pin value to 1 (or 0) by setting (or clearing) the appropriate bit in the data register.
4. Load the PWM “high” (or “low”) time into the both the timer register and the TxRB register.

5. Load the PWM “low” (or “high”) time into the TxRA register.

6. Write the appropriate value to the timer control bits TxC3-TxC2- TxC1 to select the PWM mode, and to toggle the TxA output with every timer underflow (see Table 4-2).

7. Set the TxC0 bit to start the timer.

If the user wishes to generate an interrupt on timer output transitions, reset the pending flags and then enable the desired type(s) of interrupts using TxENA and/or TxENB. The GIE bit must also be set. The interrupt service routine must reset the pending flag and perform whatever processing is desired.

For slow speed timing mode, the selectable range for the PWM “high” and “low” times is 1 to 65,536 clock cycles. For a 20 MHz internal clock, this corresponds to a time range of 0.5 microsecond to 32.8 milliseconds. Thus, the pulse period (“high” plus “low” times) can range from 1 microsecond to 65.5 milliseconds.

For high speed timing mode, the selectable range for the PWM “high” and “low” times is 1 to 65,536 clock cycles. For a 20 MHz internal clock, this corresponds to a time range of 50 nanoseconds to 3.277 milliseconds. Thus, the pulse period (“high” plus “low” times) can range from 0.1 microseconds to 6.55 milliseconds.

4.6.2 MODE 2. External Event Counter Mode

The external event counter mode is similar to the PWM mode, except that instead of counting timer clock pulses, the timer counts transitions received on the TxA pin (configured as an input). The TxA pin should be connected to an external device that generates a pulse for each event to be counted. The input signal on TxA must have a pulse width equal to or greater than the selected clock cycle. Refer to the AC Electrical Specs for this device.

The timer can be configured to sense either positive-going or negative-going transitions on the TxA pin. The maximum frequency at which transitions can be sensed is one-half the frequency of the selected clock.

As with the PWM mode, when an underflow occurs, the timer register is reloaded alternately from the TxRA and TxRB registers, and counting proceeds downward from the loaded value.

Interrupt generation in the external event counter mode is different from the PWM mode. If the TxA interrupt is enabled by TxENA, an interrupt is generated with every underflow regardless of whether the timer register is reloaded from TxRA or TxRB. If the TxB interrupt is enabled with TxENB, an interrupt is generated when a positive-going transition is detected on the TxB pin (configured as an input).

A block diagram of the timer operating in the external event counter mode is shown in Figure 4-2.

The following steps can be used to operate the timer in the external event counter mode.

1. Configure the TxA and TxB pins as an inputs by clearing bits in the appropriate configuration register.

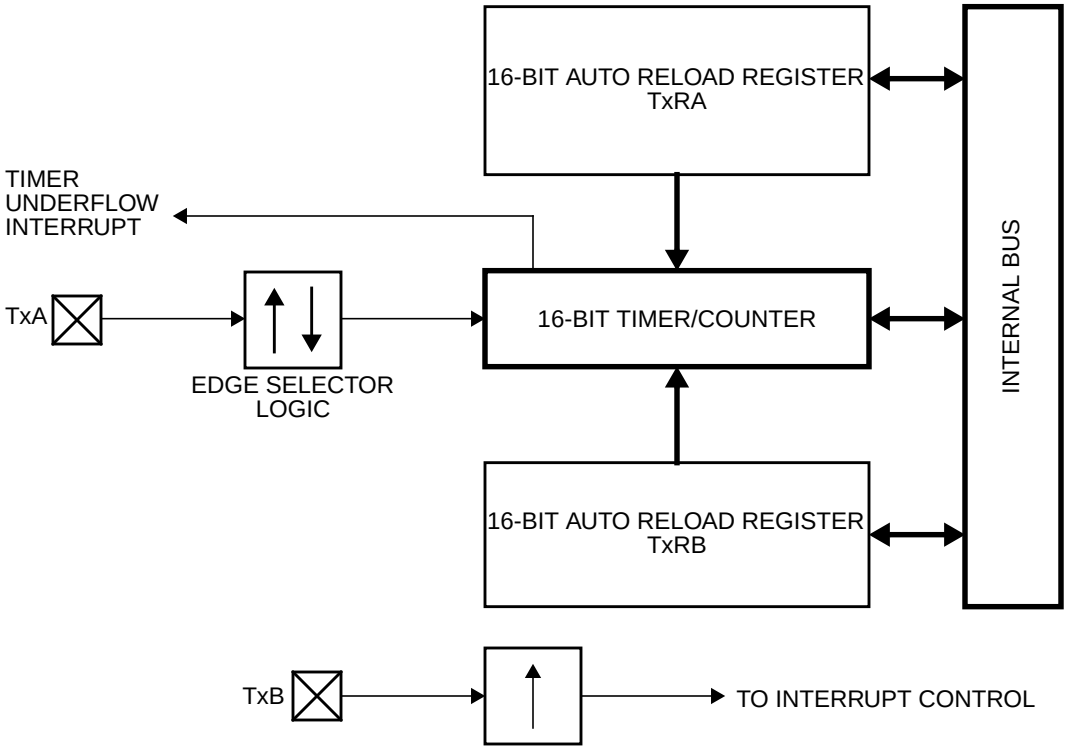


Figure 4-2 Timer in External Event Counter Mode

2. Set the minimum input pulse width by configuring the timer as either high speed (MCLK cycle is the minimum width) or low speed (instruction cycle is the minimum width).

3. Load the initial count into the timer register, the TxRA register, and the TxRB register. When this number of external events is detected, the counter will reach zero, however, it will not underflow until the next event is detected. To count N pulses, load the value N-1 into the registers. If it is only necessary to count the number of occurrences and no action needs to be taken at a particular count, load the value FFFF into the registers.

4. In order to generate an interrupt each time the timer underflows, clear the TxPNDA pending flag and then enable the interrupt by setting the TxENA bit. In order to enable interrupts on the TxB pin, clear the TxPNDB pending flag and then set the TxENB bit. The GIE bit must also be set.

5. Write the appropriate value to the timer control bits TxC3-TxC2- TxC1 to select the external event counter mode, and to select the type of transition to be sensed on the TxA pin (positive-going or negative-going; see Table 4-2).

6. Set the TxC0 bit register to start the timer.

If interrupts are being used, the TxA interrupt service routine must clear the TxPNDA flag and take whatever action is required when the timer underflows. If the user wishes to merely count the number of occurrences of an event, and anticipates that the number

of events may exceed 65,536, the interrupt service routine should record the number of underflows by incrementing a counter in memory. On each underflow, the counter/timer register is reloaded with the value from the TxRA or TxRB register.

The TxB pin and the corresponding interrupt can be used for any purpose. For example, an external device can request the value of the current count by generating a positive transition on the TxB pin. In that case, the TxB interrupt service routine should reset the TxPNDB pending bit, read the timer/counter register, and complement the value to convert a down-count from FFFF Hex into a positive count value.

4.6.3 MODE 3. Input Capture Mode

In the input capture mode, the TxA and TxB pins are configured as inputs. The timer counts down at the instruction clock rate or the internal clock (MCLK). A transition received on the TxA pin causes a transfer of the timer contents to the TxRA register. Similarly, a transition received on the TxB pin causes a transfer of the timer contents to the TxRB register. The input signal on TxA and TxB must have a pulse width equal to or greater than 1 of the selected clock cycle. Refer to the AC Electrical Specs for this device. The values captured in the TxRA register at different times reflect the elapsed time between transitions on the TxA pin. The same is true for the TxRB register and TxRB pin. Each input pin can be configured to sense either positive-going or negative-going transitions.

A block diagram of the timer operating in the input capture mode is shown in Figure 4-3.

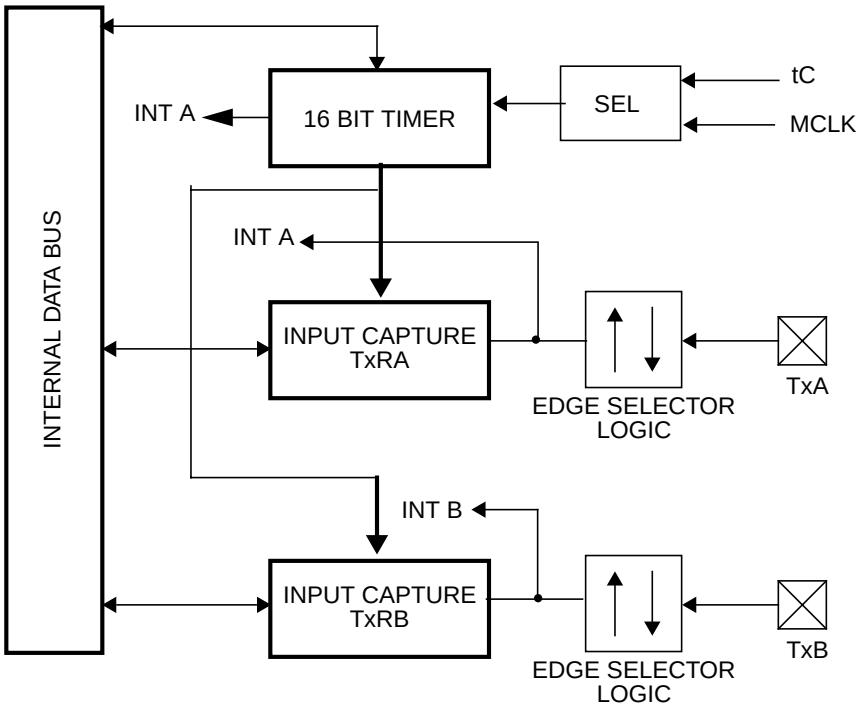


Figure 4-3 Timer in Input Capture Mode

There are three interrupt events associated with the input capture mode: input capture in TxRA, input capture in TxRB, and timer underflow. If interrupts are enabled, a TxA

interrupt is triggered by either an input capture in TxRA or a timer underflow. A TxB interrupt is triggered by an input capture in TxRB.

In this operating mode, the TxC0 control bit serves as the timer underflow interrupt pending flag. The TxA interrupt service routine can look at this flag and the TxPNDA flag to determine what caused the interrupt. A set TxC0 flag means that a timer underflow occurred, whereas a set TxPNDA flag means that an input capture occurred in TxRA. It is possible that both flags will be found set, meaning that both events occurred at the same time. The interrupt routine should take this possibility into consideration.

Note: If the timer is running in high speed mode (using internal MCLK) it is capable of generating interrupts very fast. Once both pending bits are set, any later interrupts received, before the respective pending bit is reset by an interrupt service routine, will be lost. It is up to the user to write software such that these interrupts can be processed without loss.

Because the TxC0 bit is used as the underflow interrupt pending flag, it is not available for use as a start/stop bit as in the other modes. The timer/counter register counts down continuously at the instruction clock rate, starting from the time that the input capture mode is selected with bits TxC3-TxC2-TxC1. To stop the timer from running, you must change from the input capture mode to the PWM or external event counter mode and reset the TxC0 bit.

Each of the two input pins can be independently configured to sense positive-going or negative-going transitions, resulting in four possible input capture mode configurations. The edge sensitivity of pin TxA is controlled by bit TxC1, and the edge sensitivity of pin TxB is controlled by bit TxC3, as indicated in Table 4-2.

The edge sensitivity of a pin can be changed without leaving the input capture mode by setting or clearing the appropriate control bit (TxC1 or TxC3), even while the timer is running. This feature allows you to measure the width of a pulse received on an input pin. For example, the TxB pin can be programmed to be sensitive to a positive-going edge. When the positive edge is sensed, the timer contents are transferred to the TxRB register, and a TxB interrupt is generated. The TxB interrupt service routine records the contents of the TxRB register and also reprograms the input capture mode, changing the TxB pin from positive to negative edge sensitivity. When the negative-going edge appears on the TxB pin, another TxB interrupt is generated. The interrupt service routine reads the TxRB register again. The difference between the previous reading and the current reading reflects the elapsed time between the positive edge and negative edge on the TxB input pin, i.e., the width of the positive pulse.

The TxA pin and TxRA register can be used in a similar manner, perhaps to measure the interval between some different events. Remember that the TxA interrupt service routine must test the TxC0 and TxPNDA flags to determine what caused the interrupt.

The software that measures elapsed time must take into account the possibility that an underflow occurred between the first and second readings. This can be managed by using the interrupt triggered by each underflow. The TxA interrupt service routine, after determining that an underflow caused the interrupt, should record the occurrence of an underflow by incrementing a counter in memory, or by some other means. The software that calculates the elapsed time should check the status of the underflow counter and take it into account in making the calculation.

The following steps can be used to operate the timer in the external event counter mode.

1. Configure the TxA and TxB pins as inputs by clearing bits in the appropriate configuration register.
2. With the timer/counter configured to operate in the PWM or external event counter mode (TxC2 equal to 0), reset the TxC0 bit. This stops the timer register from counting.
3. Configure the timer as either high speed (counting internal MCLK cycles) or low speed (counting instruction cycles).
4. Load the initial count into the timer register, typically the value FFFF to allow the maximum possible number of counts before underflow.
5. Clear the TxPNDA and TxPNDB interrupt pending flags, then set the TxENA and TxENB interrupt enable bits. The GIE enable bit should also be set. The interrupts are now enabled.
6. Write the appropriate value to the timer control bits TxC3-TxC2- TxC1 to select the input capture mode, and to select the types of transitions to be sensed on the TxA and TxB pins (positive-going or negative-going; see Table 4-2). As soon as the input capture mode is enabled, the timer starts counting.

When the programmed type of edge is sensed on the TxB pin, the TxRB register is loaded and a TxB interrupt is triggered. The interrupt service routine resets the TxPNDB pending bit and performs the required task such as recording the TxRB register contents.

When the programmed type of edge is sensed on the TxA pin, the TxRA register is loaded and a TxA interrupt is triggered. A TxA interrupt is also triggered when an underflow occurs in the timer register. The interrupt service routine tests both the TxPNDA and TxC0 flags to determine the cause of the interrupt, resets the pending bit, and performs the required task, such as recording the TxRA register contents or incrementing an underflow counter.

Chapter 5

MICROWIRE/PLUS

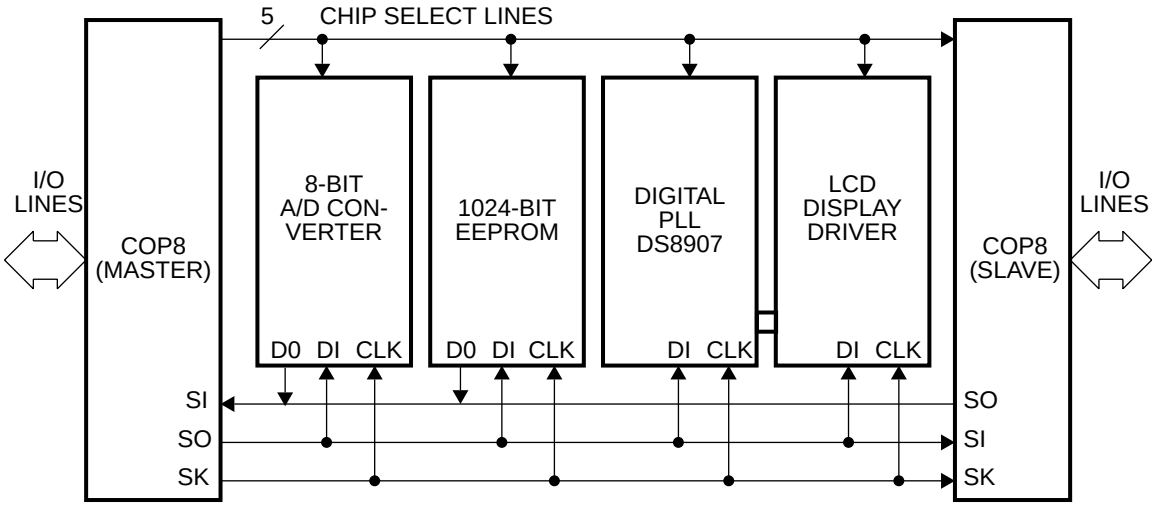
5.1 INTRODUCTION

MICROWIRE/PLUS is a synchronous serial communication system that allows the COP8 Flash Family microcontroller to communicate with any other device that also supports the MICROWIRE/PLUS system. Examples of such devices include A/D converters, comparators, EEPROMs, display drivers, telecommunications devices, and other processors (e.g., HPC and COP400 processors). The MICROWIRE/PLUS serial interface uses a simple and economical 3-wire connection between devices.

Several MICROWIRE/PLUS devices can be connected to the same 3-wire system. One of these devices, operating in what is called the master mode, supplies the synchronous clock for the serial interface and initiates data transfers. Another device, operating in what is called the slave mode, responds by sending (or receiving) the requested data. The slave device uses the master's clock for serially shifting data out (or in), while the master device shifts the data in (or out).

On the COP8 device, the three interface signals are called SI (Serial Input), SO (Serial Output), and SK (Serial Clock). To the master, SO and SK are outputs (connected to slave inputs), and SI is an input (connected to slave outputs).

The COP8 can operate either as a master or a slave, depending on how it is configured by the software. Figure 5-1 shows an example of how several devices can be connected together using the MICROWIRE/PLUS system, with the COP8 on the left operating as the master, and other devices operating as slaves. The protocol for selecting and enabling slave devices is determined by the system designer.



DD11208-23
cop8_uwirep

Figure 5-1 MICROWIRE/PLUS Example

5.2 THEORY OF OPERATION

Figure 5-2 is a block diagram illustrating the internal operation of the MICROWIRE/PLUS circuit of the COP8.

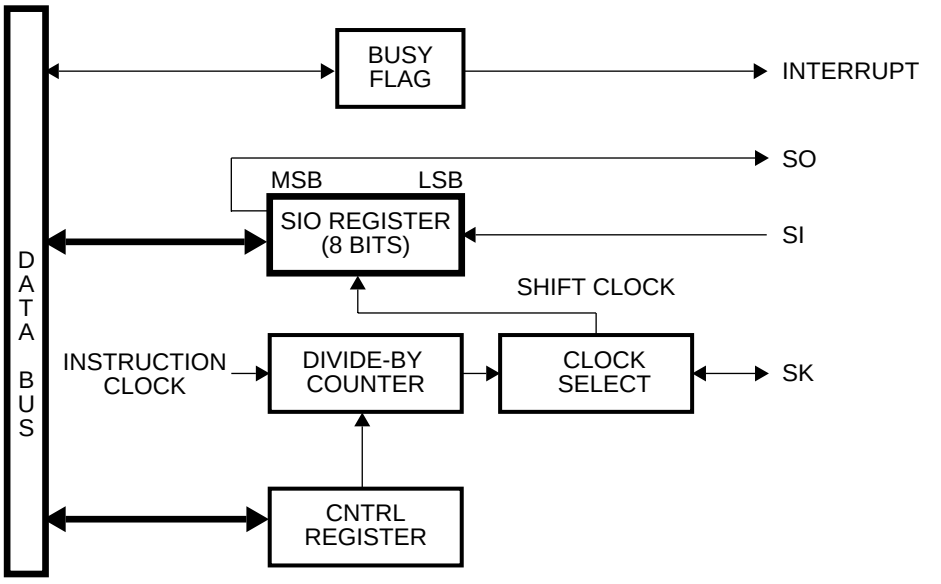


Figure 5-2 MICROWIRE/PLUS Circuit Block Diagram

An 8-bit shift register, called the SIO (Serial Input/Output) register, is used for both sending and receiving data. In either type of data transfer, bits are shifted left through the register. When a data byte is being sent, bits are shifted out through the SO output, most significant bit first. When a data byte is being received, bits are shifted in through the SI input, most significant bit first also.

The SIO register is memory-mapped in the microcontroller's data memory space, allowing the software to write a data byte to be sent, or to read a full data byte that has been received. The Busy flag in the PSW register indicates whether the SIO register is ready to be read or written. Interrupts or polling can be used to synchronize the reading or writing of the SIO register to completion of each 8-bit shift operation, or a carefully timed software loop can be programmed for this purpose.

The new MICROWIRE/PLUS circuit also supports inverted Serial clock polarity in both master and slave modes. This feature allows MICROWIRE/PLUS to be compatible with SPI serial interface. The clock polarity is selected by programming G6/J0 Configuration register bit. Accordingly, SK clock inactive state will be either low or high.

The software should write the SIO register only when the SK clock is low. A data byte is generally written at the end of an 8-bit shifting cycle, when the SK clock is low anyway. If the software inadvertently writes to the register when SK is high, unknown data may be placed in the register.

5.2.1 Timing

The SK clock signal is generated by the master device. The timing of the MICROWIRE/PLUS interface is synchronized to this signal.

There are four operating modes for the interface, relating to the external shift clock (SK) polarity and to the sample/shift phase. Table 5-2 summarizes the different options available on this part:.

Table 5-1 MICROWIRE/PLUS Shift Clock Polarity and Sample/Shift Phase

Port G		SO Clocked out on:	SI Sampled on:	SK Idle Phase
G6 (SKSEL) Config. Bit	G5 (SKPOL)* Data Bit			
0	0	SK Falling edge	SK Rising edge	Low
1	0	SK Rising edge	SK Falling edge	Low
0	1	SK Rising edge	SK Falling edge	High
1	1	SK Falling edge	SK Rising edge	High

* Only if MSEL= 1

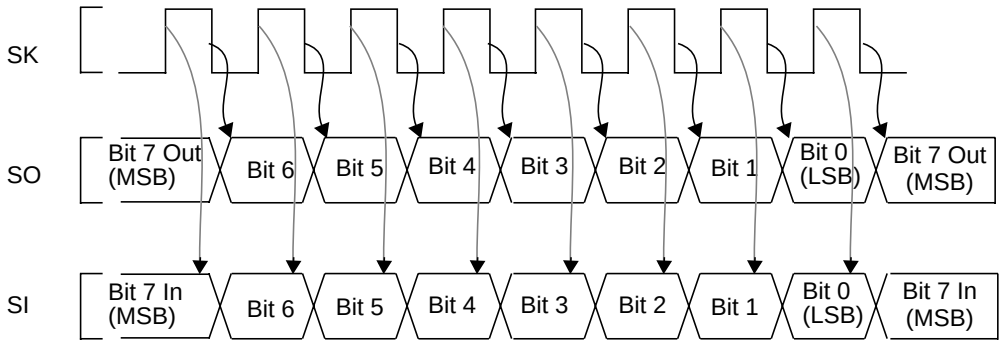
The four modes differ in the edge in which the data is driven out on SO and sampled on SI and in the Idle state of the SK clock output in between transfers.

The first two modes operate the same as the previous parts MICROWIRE/PLUS module, and are selected when G5 data bit (SKPOL) is cleared; When G6 configuration bit (SKSEL) is cleared, output data on SO is clocked out on the falling edge of the SK clock, and input data on SI is sampled on the rising edge of the SK clock. This is the standard SK mode. When G6 configuration bit (SKSEL) is set, the SK clock edge functions are reversed: output data on SO is clocked out on the rising edge of the SK clock, and the input data on SI is sampled on the falling edge of the SK clock. These 2 modes are known there as “standard SK” and “alternate SK” modes.

The other two modes compliment the MICROWIRE/PLUS operation to that of an SPI module, making it fully SPI compatible, as far as clocking is concerned. These are selected when G5 data bit (SKPOL) is set. When G6 configuration bit (SKSEL) is set, output data on SO is clocked out on the falling edge of the SK clock, and input data on SI is sampled on the rising edge of the SK clock. When G6 configuration bit (SKSEL) is cleared, the SK clock edge functions are reversed: output data on SO is clocked out on the rising edge of the SK clock, and the input data on SI is sampled on the falling edge of the SK clock.

The following timing diagrams show the timings described above. The solid lines show the clock edges which cause the output data to be clocked out on the SO pin, and the dotted arrows indicate the clock edges that cause the input data on the SI pin to be sampled, for all of the above cases:

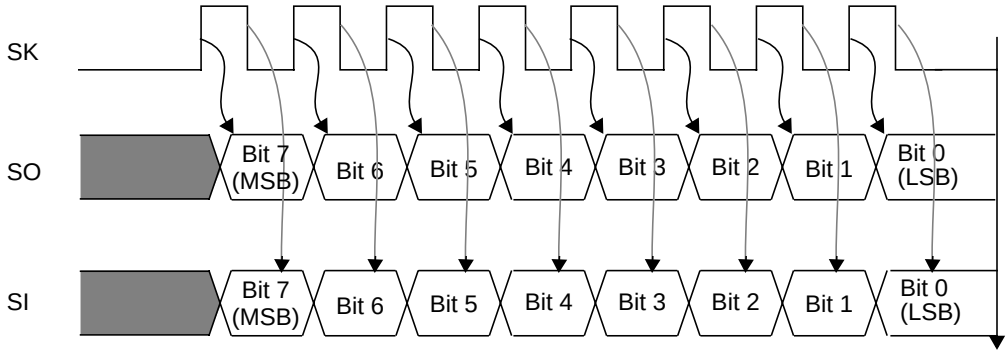
SKSEL = 0, SKPOL = 0:



COP888-02

Figure 5-3 MICROWIRE/PLUS Interface Timing, Standard SK Mode

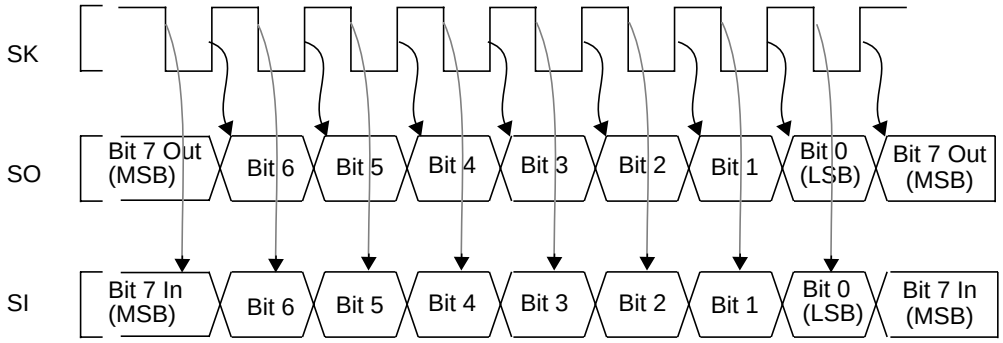
SKSEL = 1, SKPOL = 0:



COP888-08

Figure 5-4 MICROWIRE/PLUS Interface Timing, Alternate SK Mode

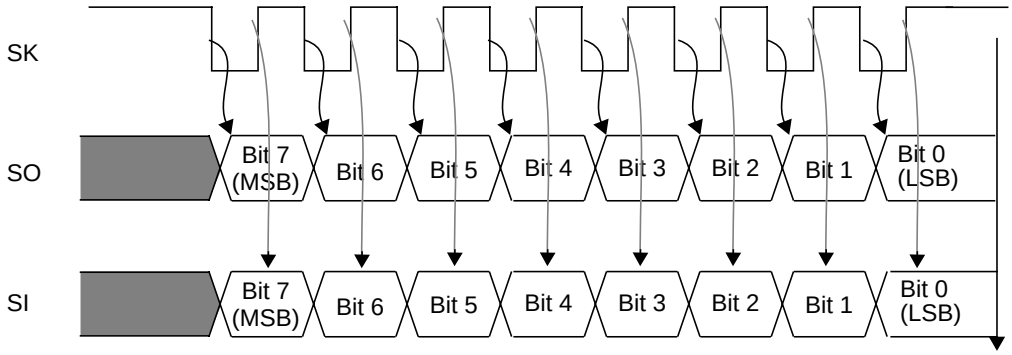
SKSEL = 0, SKPOL = 1:



COP888-02

Figure 5-5 MICROWIRE/PLUS Interface Timing, Inverted SK, Leading-Edge SO

SKSEL = 1, SKPOL = 1:



COP888-08

Figure 5-6 MICROWIRE/PLUS Interface Timing, Inverted SK, Trailing-Edge SO

5.2.2 Port G Configuration

The three MICROWIRE/PLUS signals SO, SK, and SI are alternate functions of Port G pins G4, G5, and G6, respectively. To enable the use of these pins for the MICROWIRE/PLUS interface, the MSEL (MICROWIRE Select) bit of the CNTRL register must be set to 1. (The SL1 and SL0 bits, also in the CNTRL register, are used to control the SK clock speed in master mode, as described below.)

Port G must be properly configured for operation of the interface. This is accomplished by writing certain bit values to the Port G configuration register. Pin G4 (SO) should be configured as an output for sending data. It can be placed in the TRI-STATE (high-

impedance) mode for receiving data, thereby preventing unknown data from being placed on the SO output pin. Pin G5 (SK) should be configured as an output in master mode, or as an input in slave mode. G6 (SI) serves only as an input, so it need not be specifically configured as such. The Port G configuration register programming options are summarized in Table 5-2.

Table 5-2 Port G Configuration Register Bits

Port G Config. Reg. Bits G5-G4	MICROWIRE Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
0-0	Slave, data in	TRI-STATE (unused)	SK Input	SI Input
0-1	Slave, data out and data in	SO Output	SK Input	SI Input
1-0	Master, data in	TRI-STATE (unused)	SK Output	SI Input
1-1	Master, data out and data in	SO Output	SK Output	SI Input

5.2.3 SK Clock Frequency

When the COP8 operates in master mode, it generates the SK clock signal. A divide-by counter lowers the frequency of the instruction clock, producing an SK clock period that is 2, 4, or 8 times the period of the instruction clock. The divide-by factor is programmed by writing two bits to the CNTRL register, designated SL1 and SL0 (Select 1 and Select 0 bits), as indicated in Table 5-3.

Table 5-3 Master Mode Clock Select Bits

SL1 (CNTRL Bit 1)	SL0 (CNTRL Bit 0)	SK Clock Period
0	0	2 times instruction clock period
0	1	4 times instruction clock period
1	X	8 times instruction clock period

The internal divide-by counter is reset when the MICROWIRE Busy flag (described below) goes to 1. Because of this, the divide-by counter always starts from 0 at the beginning of an 8-bit shift cycle, ensuring uniform SK clock pulses.

When the COP8 microcontroller operates in slave mode, the SK clock is generated by the external master device. In this case, SK is an input, and the SK clock-generating circuit of the COP8 is inactive.

5.2.4 Busy Flag and Interrupt

A flag bit in the PSW (Processor Status Word) register indicates the status of the SIO shift register. To initiate an 8-bit shifting operation, the software sets this bit to 1. Shifting then starts and continues automatically at the SK clock rate. With each shift, the high-order bit of the register is shifted out on SO (if enabled), and simultaneously, the low-order bit of the register is shifted in from SI.

When the 8-bit shifting operation is finished, the Busy flag is automatically reset to 0 by the hardware, and a MICROWIRE/PLUS interrupt is generated (if enabled). When this occurs, the software should write the next byte to be sent (and/or read the full byte just received) and then set the Busy flag to initiate transfer of the next byte. Either polling or interrupts can be used to determine when this needs to be done.

If polling is used, the CPU runs in a continuous loop in which the status of the Busy flag is read. When the flag is found to be 0, the software reads or writes the SIO register, sets the Busy flag for the next transfer, and then returns to the polling loop.

If interrupts are used, the CPU can perform other tasks while the data byte is being shifted in or out of the SIO register. The MICROWIRE interrupt service routine resets the MICROWIRE pending flag (μWPND), reads or writes the SIO register, sets the Busy flag to one for the next transfer, and then returns from the interrupt. To enable the MICROWIRE interrupt, set the μWEN (MICROWIRE interrupt enable) bit. The GIE (Global Interrupt Enable) bit must also be set. The μWPND flag and μWEN enable bit are bits 3 and 2 of the ICNTRL register, respectively.

The software can control the timing of the transfer by setting and clearing the Busy flag directly. If the Busy bit is cleared directly by the software, shifting stops immediately. In this case an interrupt is not generated.

The handshaking protocol between the master and slave should ensure that the slave device is given enough time to respond after being enabled by the master. An example of a MICROWIRE/PLUS master/slave protocol is provided in the applications chapter.

It is possible to eliminate the need for polling or interrupts, thereby speeding up the transfer. This is accomplished by writing a software loop that executes in the exact amount of time necessary to allow an 8-bit shift operation. At the end of the loop, the software initiates the next 8-bit transfer immediately, without checking the Busy bit. This is called the MICROWIRE “fast burst” mode. An example of this type of program is presented in the applications chapter.

Some external devices may require a continuous bit stream, without any pauses between bytes. This mode, called the MICROWIRE “continuous” mode, is also accomplished by writing a software loop that executes in a specific number of cycles. The clock divide-by factor must be 8. An example of this type of program is presented in the applications chapter.

When the COP8 operates in slave mode, the Busy flag should be set only when the SK clock signal (an input) is low. This is because the Busy bit is ANDed internally with the SK signal to produce the clock-shifting signal. If the Busy flag is set while SK is already high, the current SK pulse is gated-in immediately, resulting in a clock pulse with an unknown width (perhaps very narrow), causing unreliable shifting.

5.3 MASTER MODE OPERATION EXAMPLE

When the COP8 operates in master mode, it generates the SK clock and initiates the transfer. The application software can perform a data transfer using the numbered steps shown below.

1. Write the proper value to the CNTRL register. To enable use of the Port G pins, set the MSEL bit. To set the divide-by factor for the SK clock, write the desired 2-bit value to the SL1 and SL0 bits (Table 5-3).
2. Write the proper value to the Port G configuration register, bits G5 and G4, to make the G5 (SK) pin an output and the G4 (SO) pin either TRI-STATE or an output, depending on whether the COP8 is transmitting (Table 5-2). Write the proper value to the Port G configuration register bit 6 to select the Standard or Alternate SK Clocking mode.
3. If necessary, enable the desired slave device.
4. If sending data, write the data byte to the SIO register.
5. Set the Busy flag in the PSW register to initiate the transfer. Shifting proceeds automatically at the SK clock rate. The Busy flag is automatically reset upon completion of the 8-bit transfer.
6. Run in a loop and test the Busy flag for completion of the 8-bit transfer.
7. If receiving data, read the data byte in the SIO register.
8. Repeat steps 4 through 7 until all data bytes are transferred.

Polling is used in the example above. If interrupts are to be used instead, the software should reset the µWPND pending flag and set the µWEN enable bit at the beginning. The GIE bit must also be set. The MICROWIRE interrupt service routine should be programmed to reset the pending flag, read or write the SIO register, and set the Busy flag to initiate the next transfer, thus replacing steps 4 through 8 above.

5.4 SLAVE MODE OPERATION EXAMPLE

When the COP8 operates in slave mode, the external master device generates the SK clock and initiates the transfer; SK is an input to the COP8. The application software can set up the COP8 device to allow a data transfer using the numbered steps shown below.

1. To enable use of the Port G pins, set the MSEL bit of the CNTRL register.
2. Write the proper value to the Port G configuration register to make the G5 (SK) pin an input and the G4 (SO) pin either TRI-STATE or an output, depending on whether the COP8 is transmitting (Table 5-2). Write the proper value to the Port G configuration register bit 6 to select the Standard or Alternate SK Clocking mode.
3. If sending data, write the data byte to the SIO register.

4. Set the Busy flag in the PSW register to allow the transfer. This should be done only when the SK signal is low. The handshaking protocol between the master and slave should ensure that the COP8 is given enough time to set the Busy flag before the data transfer starts. Once started, shifting proceeds at the SK clock rate. The Busy flag is automatically reset upon completion of the 8-bit transfer.

5. Run in a loop and test the Busy flag for completion of the 8-bit transfer.

6. If receiving data, read the data byte in the SIO register.

7. Repeat steps 3 through 6 until all data bytes are transferred.

Again, polling is used in the example above. Interrupts can be used instead as described under the master mode example.

Chapter 6

POWER SAVE MODES

6.1 INTRODUCTION

This device supports three operating modes, each of which have two power save modes of operation. The three operating modes are: High Speed, Dual Clock, and Low Speed. Within each operating mode, the two power save modes are: HALT and IDLE. In the HALT mode of operation, all microcontroller activities are stopped and power consumption is reduced to a very low level. In this device, the HALT mode is enabled and disabled by a bit in the Option register. The IDLE mode is similar to the HALT mode, except that certain sections of the device continue to operate, such as: the on-board oscillator, the IDLE Timer (Timer T0), and the Clock Monitor. This allows real time to be maintained. During power save modes of operation, all on board RAM, registers, I/O states and timers (with the exception of T0) are unaltered.

Two oscillators are used to support the three different operating modes. The high speed oscillator refers to the oscillator connected to CKI and the low speed oscillator refers to the 32Khz oscillator connected to pins L0 & L1. When using L0 and L1 for the low speed oscillator, the user must ensure that the L0 and L1 pins are configured for hi-Z input, L1 is not using CKX on the USART, and Multi-Input-Wakeup for these pins is disabled.

A diagram of the three modes is shown in Figure 6-1.

NOTE: This description is written for all three high speed oscillator types: crystal. RC, and external, even though they may not all be implemented on a specific device. The power save logic supports all three oscillator types.

6.2 POWER SAVE MODE CONTROL REGISTER

The ITMR control register allows for navigation between the three different modes of operation. It is also used for the Idle Timer. The register bit assignments are shown below. This register is cleared to 40 by Reset as shown below.

ITMR							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LSON	HSON	DCEN	CCKSEL	ITSTMD	ITSEL2	ITSEL1	ITSEL0
0	1	0	0	0	0	0	0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

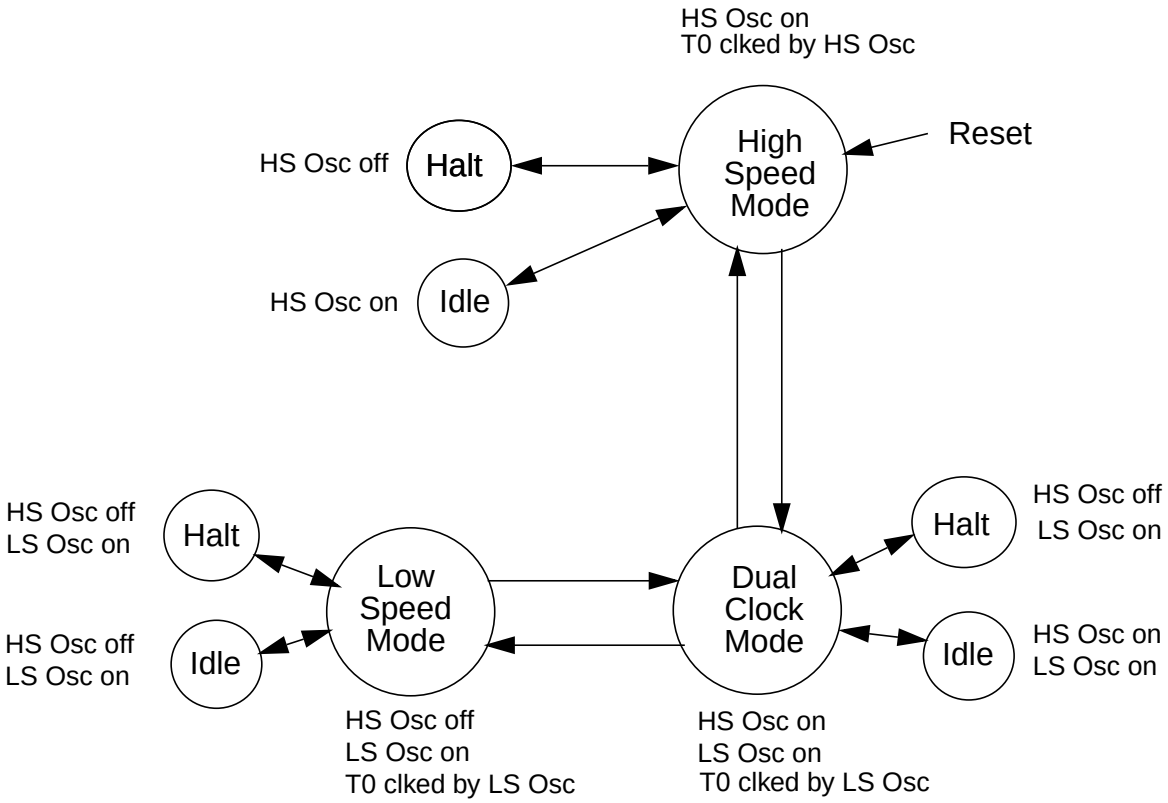


Figure 6-1 Diagram of Power Save Modes

LSON: This bit is used to turn-on the low-speed oscillator. When LSON = 0, the low speed oscillator is off. When LSON = 1, the low speed oscillator is on. There is a startup time associated with this oscillator. See the Oscillator Circuits chapter.

HSON: This bit is used to turn-on the high speed oscillator. When HSON = 0, the high speed oscillator is off. When HSON = 1, the high speed oscillator is on. There is a startup time associated with this oscillator. See the startup time table in the Oscillator Circuits chapter.

DCEN: This bit selects the clock source for the Idle Timer. If this bit = 0, then the high speed clock is the clock source for the Idle Timer. If this bit = 1, then the low speed clock is the clock source for the Idle Timer. The low speed oscillator must be started and stabilized before setting this bit to a 1.

CCKSEL: This bit selects whether the high speed clock or low speed clock is gated to the microcontroller core. When this bit = 0, the Core clock will be the high speed clock. When this bit = 1, then the Core clock will be the low speed clock. Before switching this bit to either state, the appropriate clock should be turned on and stabilized.

DCEN	**CCKSEL**	
0	0	High Speed Mode. Core and Idle Timer clock = high speed
1	0	Dual Clock Mode. Core clock = high speed; Idle Timer = low speed
1	1	Low Speed Mode. Core and Idle Timer clock = low speed

0 1 Invalid. If this is detected, the Low Speed Mode will be forced.

Bits 3-0: These are bits used to control the Idle Timer. See the Timers Chapter for the description of these bits.

Table 6-1 lists the valid contents for the four most significant bits of the ITMR Register. Any other value is illegal and will result in an unrecoverable loss of a clock to the CPU core. To prevent this condition, the device will automatically reset if any illegal value is detected.

Table 6-1 Valid contents of Dual Clock Control Bits.

LSON	HSON	DCEN	CCKSEL	Mode
0	1	0	0	High Speed
1	1	0	0	High Speed/Dual Clock Transition
1	1	1	0	Dual Clock
1	1	1	1	Dual Clock/Low Speed Transition
1	0	1	1	Low Speed

6.3 OSCILLATOR STABILIZATION

Both the high speed oscillator and low speed oscillator have a startup delay associated with them. When switching between the modes, the software must ensure that the appropriate oscillator is started up and stabilized before switching to the new mode. See the Oscillator Circuits chapter for startup times for both oscillators.

6.4 HIGH SPEED MODE OPERATION

This mode of operation allows high speed operation for both the main Core clock and also for the Idle Timer. This is the default mode of the device and will always be entered upon any of the Reset conditions described in the Reset Chapter. It can also be entered from Dual Clock mode. It cannot be directly entered from the Low Speed mode without passing through the Dual Clock mode first.

To enter from the Dual Clock mode, the following sequence must be followed:

1. Software clears DCEN to 0.
2. Software clears LSON to 0.

These should be done in the above sequence.

6.4.1 High Speed HALT Mode

The fully static architecture of this device allows the state of the microcontroller to be frozen. This is accomplished by stopping the internal clock of the device during the HALT mode. If an R/C or crystal type clock is used, the controller also stops the CKI pin from

oscillating during the HALT mode. The processor can be forced to exit the HALT mode and resume normal operation at any time.

During normal operation, typical power consumption is in the range of TBD to TBD milliamperes. The actual power consumption depends heavily on the clock speed and operating voltage used in an application. In the HALT mode, the device only draws a small leakage current, plus any current necessary for driving the outputs. Typically, power consumption is reduced to less than TBD microampere. Since total power consumption is affected by the amount of current required to drive the outputs, all I/Os should be configured to draw minimal current prior to entering the HALT mode, if possible. In order to reduce power consumption even further, the power supply (V_{CC}) can be reduced to a very low level during the HALT mode, just high enough to guarantee retention of data stored in RAM. The allowed lower voltage level (Vr) is specified in the Electrical Specs Chapter.

Entering The High Speed Halt Mode:

There are two ways to enter the high speed HALT mode. One method is to simply stop the high speed processor clock (if the hardware implementation allows it). The other method is to set bit 7 of the Port G data register, if the Option register bit allows it.

Clock-Stopping Method

This method is not recommended for devices with eprom and flash program memory. The clock-stopping method of entering the HALT mode can be used only if the hardware implementation of the processor clock allows it (i.e. external clock). The clock signal at CKI can be stopped in either state. When the clock stops, the device stops running but maintains all register and RAM contents. Power consumption is reduced to a very low level (not exactly same current level as the HALT mode if EPROM or Flash is active). When the clock starts running again, the processor begins running again from the point at which it was stopped. Please note that this method may cause a clock monitor error.

Port G Method

Using the Port G method, the device enters the HALT mode under software control when the Port G data register bit 7 is set to 1. All processor action stops in the middle of the next instruction cycle, and power consumption is reduced to a very low level.

Exiting The High Speed Halt Mode

If the HALT mode was entered by externally stopping the high speed processor clock, it is exited by re-starting the clock. The processor begins running again from the point at which it was stopped.

If the HALT mode was entered by setting bit 7 of the Port G data register, there is a choice of methods for exiting the HALT mode: a chip Reset using the $\overline{\text{RESET}}$ pin, a Multi-Input Wakeup or a low-to-high transition on the G7 pin of Port G. The Reset method and Multi-Input Wakeup method can be used with any clock option, but the availability of the G7 input is dependent on the clock option.

HALT Exit Using Reset

A device Reset, which is invoked by a low-level signal on the $\overline{\text{RESET}}$ input pin, takes the device out of the HALT mode and starts execution from address 0000H. The initialization software should determine what special action is needed, if any, upon start-up of the device from HALT. The initialization of all registers following a $\overline{\text{RESET}}$ exit from HALT is described in the Reset Chapter of this manual.

HALT Exit Using Multi-Input Wakeup

The device can be brought out of the HALT mode by a transition received on one of the available Wakeup pins. The pins used and the types of transitions sensed on the Multi-input pins are software programmable. For information on programming and using the Multi-Input Wakeup feature, refer to the Multi-Input Wakeup Chapter.

A start-up delay may be required between the device wakeup and the execution of program instructions, depending on the type of chip clock. If a single-pin clock is used (R-C) or external clock, the start-up delay is optional. It can be invoked under software control by setting the CLKDLY bit, which is bit 7 of the Port G configuration register. (Upon Reset, this bit is cleared, resulting in no start-up delay.)

If a two-pin, closed-loop crystal oscillator is used, the start-up delay is mandatory, and is implemented whether or not the CLKDLY bit is set. This is because all crystal oscillators and resonators require some time to reach a stable frequency and full operating amplitude.

If the start-up delay is used, the IDLE Timer (Timer T0) provides a fixed delay from the time the clock is enabled to the time the program execution begins. Upon exit from the HALT mode, the IDLE Timer is enabled with a starting value of 256 and is decremented with each instruction cycle. (The instruction clock runs at one-fifth the frequency of the high speed oscillator.) An internal Schmitt trigger connected to the on-chip CKI inverter ensures that the IDLE Timer is clocked only when the oscillator has a large enough amplitude. (The Schmitt trigger is not part of the oscillator closed loop.) When the IDLE Timer underflows, the clock signals are enabled on the chip, allowing program execution to proceed. Thus, the delay is equal to 256 instruction cycles.

Note: To ensure accurate operation upon start-up of the device using Multi-input Wakeup, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

HALT Exit Using G7 Pin

Using the G7 input pin is possible only if the R/C oscillator or external clock is being used. If a crystal is being used, the G7/CKO pin is used as CKO, and is therefore unavailable for use as the HALT/Restart pin.

If the G7 pin is available, a low-to-high transition on the pin takes the processor out of the HALT mode, and program execution resumes from the point at which it stopped.

Note: To ensure accurate operation upon start-up of the device using the G7 pin, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

Options:

This device has two options associated with the HALT mode. The first option enables the HALT mode feature, while the second option disables HALT mode operation. Selecting the disable HALT mode option will cause the microcontroller to ignore any attempts to HALT the device under software control. Note that this device can still be placed in the HALT mode by stopping the clock input to the microcontroller, if the program memory is masked ROM. See the Option chapter for more details on this option bit.

6.4.2 High Speed IDLE Mode

In the IDLE mode, program execution stops and power consumption is reduced to a very low level as with the HALT mode. However, the high speed oscillator, IDLE Timer (Timer T0), and Clock Monitor continue to operate, allowing real time to be maintained. The device remains idle for a selected amount of time up to 65,536 instruction cycles, or 32.768 milliseconds with a 2 MHz instruction clock frequency, and then automatically exits the IDLE mode and returns to normal program execution.

The device is placed in the IDLE mode under software control by setting the IDLE bit (bit 6 of the Port G data register).

The IDLE timer window is selectable from one of five values, 4k, 8k, 16k, 32k or 64k instruction cycles. Selection of this value is made through the ITMR register.

The IDLE mode uses the on-chip IDLE Timer (Timer T0) to keep track of elapsed time in the IDLE state. The IDLE Timer runs continuously at the instruction clock rate, whether or not the device is in the IDLE mode. Each time the bit of the timer associated with the selected window toggles, the T0PND bit is set, an interrupt is generated (if enabled), and the device exits the IDLE mode if in that mode. If the IDLE timer interrupt is enabled, the interrupt is serviced before execution of the main program resumes. (However, the instruction which was started as the part entered the IDLE mode is completed before the interrupt is serviced. This instruction should be a NOP which should follow the enter IDLE instruction.) The user must reset the IDLE timer pending flag (T0PND) before entering the IDLE MODE

As with the HALT mode, this device can also be returned to normal operation with a reset, or with a Multi-Input Wakeup input. Upon reset the ITMR register is cleared and the ITMR register selects the 4,096 instruction cycle tap of the Idle Timer.

The IDLE timer cannot be started or stopped under software control, and it is not memory mapped, so it cannot be read or written by the software. Its state upon Reset is

unknown. Therefore, if the device is put into the IDLE mode at an arbitrary time, it will stay in the IDLE mode for somewhere between 1 and the selected number of instruction cycles.

In order to precisely time the duration of the IDLE state, entry into the IDLE mode must be synchronized to the state of the IDLE Timer. The best way to do this is to use the IDLE Timer interrupt, which occurs on every underflow of the bit of the IDLE Timer which is associated with the selected window. Another method is to poll the state of the IDLE Timer pending bit T0PND, which is set on the same occurrence. The Idle Timer interrupt is enabled by setting bit T0EN in the ICNTRL register.

Any time the IDLE Timer window length is changed there is the possibility of generating a spurious IDLE Timer interrupt by setting the T0PND bit. The user is advised to disable IDLE Timer interrupts prior to changing the value of the ITSEL bits of the ITMR Register and then clear the TOPND bit before attempting to synchronize operation to the IDLE Timer.

NOTE: As with the HALT mode, it is necessary to program two NOP's to allow clock resynchronization upon return from the IDLE mode. The NOP's are placed either at the beginning of the IDLE timer interrupt routine or immediately following the “enter IDLE mode” instruction.

For more information on the IDLE Timer and its associated interrupt, see the description in the Timers Chapter.

6.5 DUAL CLOCK MODE OPERATION

This mode of operation allows for high speed operation of the Core clock and low speed operation of the Idle Timer. This mode can be entered from either the High Speed mode or the Low Speed mode.

To enter from the High Speed mode, the following sequence must be followed:

1. Software sets the LSON bit to 1.
2. Software waits until the low speed oscillator has stabilized. See Table 6-2.
3. Software sets the DCEN bit to 1.

To enter from the Low Speed mode, the following sequence must be followed:

1. Software sets the HSON bit to 1.
2. Software waits until the high speed oscillator has stabilized. See Table 6-2.
3. Software clears the CCKSEL bit to 0.

.

Table 6-2 Startup Times

CKI Frequency	Startup Time
10 MHz	1 - 10 msec
3.33 MHz	3 - 10 msec
1 MHz	3 - 20 msec
455 kHz	10 - 30 msec
32 kHz (low speed oscillator)	2 - 5 sec

6.5.1 Dual Clock HALT Mode

The fully static architecture of this device allows the state of the microcontroller to be frozen. This is accomplished by stopping the high speed clock of the device during the HALT mode. If an R/C or crystal type clock is used, the controller also stops the CKI pin from oscillating during the HALT mode. The processor can be forced to exit the HALT mode and resume normal operation at any time. The low speed clock remains on during HALT in the Dual Clock mode.

During normal operation, typical power consumption is in the range of TBD to TBD milliamperes. The actual power consumption depends heavily on the clock speed and operating voltage used in an application. In the HALT mode, the device only draws a small leakage current, plus any current necessary for driving the outputs. Typically, power consumption is reduced to less than TBD microampere. Since total power consumption is affected by the amount of current required to drive the outputs, all I/Os should be configured to draw minimal current prior to entering the HALT mode, if possible.

Entering The Dual Clock Halt Mode:

There is only one way to enter the Dual Clock HALT mode.The device enters the HALT mode under software control when the Port G data register bit 7 is set to 1. All processor action stops in the middle of the next instruction cycle, and power consumption is reduced to a very low level. In order to expedite exit from HALT, the low speed oscillator is left running when the device is Halted in the Dual Clock mode. However, the Idle Timer will not be clocked.

Exiting The Dual Clock Halt Mode

When the HALT mode is entered by setting bit 7 of the Port G data register, there is a choice of methods for exiting the HALT mode: a chip Reset using the $\overline{\text{RESET}}$ pin, a Multi-Input Wakeup or a low-to-high transition on the G7 pin of Port G. The Reset method and Multi-Input Wakeup method can be used with any clock option, but the availability of the G7 input is dependent on the clock option.

HALT Exit Using Reset

A device Reset, which is invoked by a low-level signal on the $\overline{\text{RESET}}$ input pin, takes the device out of the Dual Clock mode and puts it into the High Speed mode.

HALT Exit Using Multi-Input Wakeup

The device can be brought out of the HALT mode by a transition received on one of the available Wakeup pins. The pins used and the types of transitions sensed on the Multi-input pins are software programmable. For information on programming and using the Multi-Input Wakeup feature, refer to the Multi-Input Wakeup Chapter.

A start-up delay may be required between the device wakeup and the execution of program instructions, depending on the type of chip clock. If a single-pin high speed clock is used (R-C) or external clock, the start-up delay is optional. It can be invoked under software control by setting the CLKDLY bit, which is bit 7 of the Port G configuration register. (Upon Reset, this bit is cleared, resulting in no start-up delay.)

If a two-pin, closed-loop crystal oscillator is used for the high speed clock, the start-up delay is mandatory, and is implemented whether or not the CLKDLY bit is set. This is because all crystal oscillators and resonators require some time to reach a stable frequency and full operating amplitude.

If the start-up delay is used, the IDLE Timer (Timer T0) provides a fixed delay from the time the clock is enabled to the time the program execution begins. Upon exit from the HALT mode, the IDLE Timer is enabled with a starting value of 256 and is decremented with each instruction cycle using the high speed clock. (The instruction clock runs at one-fifth the frequency of the high speed oscillatory.) An internal Schmitt trigger connected to the on-chip CKI inverter ensures that the IDLE Timer is clocked only when the high speed oscillator has a large enough amplitude. (The Schmitt trigger is not part of the oscillator closed loop.) When the IDLE Timer underflows, the clock signals are enabled on the chip, allowing program execution to proceed. Thus, the delay is equal to 256 instruction cycles. After exiting HALT, the Idle Timer will return to being clocked by the low speed clock.

Note: To ensure accurate operation upon start-up of the device using Multi-input Wakeup, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

HALT Exit Using G7 Pin

Using the G7 input pin is possible only if the R/C oscillator or external clock is being used for the high speed clock. If a crystal is being used, the G7/CKO pin is used as CKO, and is therefore unavailable for use as the HALT/Restart pin.

If the G7 pin is available, a low-to-high transition on the pin takes the processor out of the HALT mode, and program execution resumes from the point at which it stopped.

Note: To ensure accurate operation upon start-up of the device using the G7 pin, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

Options:

This device has two options associated with the HALT mode. The first option enables the HALT mode feature, while the second option disables HALT mode operation. Selecting the disable HALT mode option will cause the microcontroller to ignore any attempts to HALT the device under software control. See the device specific chapter for more details on this option bit.

6.5.2 Dual Clock IDLE Mode

In the IDLE mode, program execution stops and power consumption is reduced to a very low level as with the HALT mode. However, both oscillators, IDLE Timer (Timer T0), and Clock Monitor continue to operate, allowing real time to be maintained. The Idle Timer is clocked by the low speed clock. The device remains idle for a selected amount of time up to 1 second, and then automatically exits the IDLE mode and returns to normal program execution using the high speed clock.

The device is placed in the IDLE mode under software control by setting the IDLE bit (bit 6 of the Port G data register).

The IDLE timer window is selectable from one of five values, 0.125 seconds, 0.25 seconds, 0.5 seconds, 1 second and 2 seconds. Selection of this value is made through the ITMR register.

The IDLE mode uses the on-chip IDLE Timer (Timer T0) to keep track of elapsed time in the IDLE state. The IDLE Timer runs continuously at the low speed clock rate, whether or not the device is in the IDLE mode. Each time the bit of the timer associated with the selected window toggles, the T0PND bit is set, an interrupt is generated (if enabled), and the device exits the IDLE mode if in that mode. If the IDLE timer interrupt is enabled, the interrupt is serviced before execution of the main program resumes. (However, the instruction which was started as the part entered the IDLE mode is completed before the interrupt is serviced. This instruction should be a NOP which should follow the enter IDLE instruction.) The user must reset the IDLE timer pending flag (T0PND) before entering the IDLE MODE

As with the HALT mode, this device can also be returned to normal operation with a Multi-Input Wakeup input.

The IDLE timer cannot be started or stopped under software control, and it is not memory mapped, so it cannot be read or written by the software. Its state upon Reset is unknown. Therefore, if the device is put into the IDLE mode at an arbitrary time, it will stay in the IDLE mode for somewhere between 30 us and the selected time period.

In order to precisely time the duration of the IDLE state, entry into the IDLE mode must be synchronized to the state of the IDLE Timer. The best way to do this is to use the IDLE Timer interrupt, which occurs on every underflow of the bit of the IDLE Timer which is

associated with the selected window. Another method is to poll the state of the IDLE Timer pending bit T0PND, which is set on the same occurrence. The Idle Timer interrupt is enabled by setting bit T0EN in the ICNTRL register.

Any time the IDLE Timer window length is changed there is the possibility of generating a spurious IDLE Timer interrupt by setting the T0PND bit. The user is advised to disable IDLE Timer interrupts prior to changing the value of the ITSEL bits of the ITMR Register and then clear the TOPND bit before attempting to synchronize operation to the IDLE Timer.

NOTE: As with the HALT mode, it is necessary to program two NOP's to allow clock resynchronization upon return from the IDLE mode. The NOP's are placed either at the beginning of the IDLE timer interrupt routine or immediately following the “enter IDLE mode” instruction.

For more information on the IDLE Timer and its associated interrupt, see the description in the Timers Chapter.

6.5.3 Low Speed Mode Operation

This mode of operation allows for low speed operation of the core clock and low speed operation of the Idle Timer. Because the low speed oscillator draws very little operating current, and also to expedite restarting from HALT mode, the low speed oscillator is left on at all times in this mode, including HALT mode. This is the lowest power mode of operation on the device. This mode can only be entered from the Dual Clock mode.

To enter the Low Speed mode, the following sequence must be followed:

1. Software sets the CCKSEL bit to 1.
2. Software clears the HSON bit to 0.

These steps should be done in the above sequence. Since the low speed oscillator is already running, there is no clock startup delay.

6.5.4 Low Speed HALT Mode

The fully static architecture of this device allows the state of the microcontroller to be frozen. Because the low speed oscillator draws very minimal operating current, it will be left running in the low speed halt mode. However, the Idle timer will not be running. This also allows for a faster exit from HALT. The processor can be forced to exit the HALT mode and resume normal operation at any time.

During normal operation, typical power consumption is in the range of TBD to TBD milliamperes. The actual power consumption depends heavily on the clock speed and operating voltage used in an application. In the HALT mode, the device only draws a small leakage current, plus any current necessary for driving the outputs. Typically, power consumption is reduced to less than TBD microampere. Since total power consumption is affected by the amount of current required to drive the outputs, all I/Os should be configured to draw minimal current prior to entering the HALT mode, if possible.

Entering The Low Speed Halt Mode:

There is only one way to enter the Low Speed HALT mode.The device enters the HALT mode under software control when the Port G data register bit 7 is set to 1. All processor action stops in the middle of the next instruction cycle, and power consumption is reduced to a very low level. In order to expedite exit from HALT, the low speed oscillator is left running when the device is Halted in the Low Speed mode. However, the Idle Timer will not be clocked.

Exiting The Low Speed Halt Mode

When the HALT mode is entered by setting bit 7 of the Port G data register, there is a choice of methods for exiting the HALT mode: a chip Reset using the $\overline{\text{RESET}}$ pin, a Multi-Input Wakeup or a low-to-high transition on the G7 pin of Port G. The Reset method and Multi-Input Wakeup method can be used with any clock option, but the availability of the G7 input is dependent on the clock option.

HALT Exit Using Reset

A device Reset, which is invoked by a low-level signal on the $\overline{\text{RESET}}$ input pin, takes the device out of the Low Speed mode and puts it into the High Speed mode.

HALT Exit Using Multi-Input Wakeup

The device can be brought out of the HALT mode by a transition received on one of the available Wakeup pins. The pins used and the types of transitions sensed on the Multi-input pins are software programmable. For information on programming and using the Multi-Input Wakeup feature, refer to the Multi-Input Wakeup Chapter.

As the low speed oscillator is left running, there is no start up delay when exiting the low speed halt mode, regardless of the state of the CLKDLY bit.

Note: To ensure accurate operation upon start-up of the device using Multi-input Wakeup, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

HALT Exit Using G7 Pin

Using the G7 input pin is possible only if the R/C oscillator or external clock is being used for the high speed clock. If a crystal is being used, the G7/CKO pin is used as CKO, and is therefore unavailable for use as the HALT/Re-start pin.

If the G7 pin is available, a low-to-high transition on the pin takes the processor out of the HALT mode, and program execution resumes from the point at which it stopped.

Note: To ensure accurate operation upon start-up of the device using the G7 pin, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

Options:

This device has two options associated with the HALT mode. The first option enables the HALT mode feature, while the second option disables HALT mode operation. Selecting the disable HALT mode option will cause the microcontroller to ignore any attempts to HALT the device under software control. See the Option chapter for more details on this option bit.

6.5.5 Low Speed IDLE Mode

In the IDLE mode, program execution stops and power consumption is reduced to a very low level as with the HALT mode. However, the low speed oscillator, IDLE Timer (Timer T0), and Clock Monitor continue to operate, allowing real time to be maintained. The device remains idle for a selected amount of time up to 2 seconds, and then automatically exits the IDLE mode and returns to normal program execution using the low speed clock.

The device is placed in the IDLE mode under software control by setting the IDLE bit (bit 6 of the Port G data register).

The IDLE timer window is selectable from one of five values, 0.125 seconds, 0.25 seconds, 0.5 seconds, 1 second, and 2 seconds. Selection of this value is made through the ITMR register.

The IDLE mode uses the on-chip IDLE Timer (Timer T0) to keep track of elapsed time in the IDLE state. The IDLE Timer runs continuously at the low speed clock rate, whether or not the device is in the IDLE mode. Each time the bit of the timer associated with the selected window toggles, the T0PND bit is set, an interrupt is generated (if enabled), and the device exits the IDLE mode if in that mode. If the IDLE timer interrupt is enabled, the interrupt is serviced before execution of the main program resumes. (However, the instruction which was started as the part entered the IDLE mode is completed before the interrupt is serviced. This instruction should be a NOP which should follow the enter IDLE instruction.) The user must reset the IDLE timer pending flag (T0PND) before entering the IDLE MODE

As with the HALT mode, this device can also be returned to normal operation with a Multi-Input Wakeup input.

The IDLE timer cannot be started or stopped under software control, and it is not memory mapped, so it cannot be read or written by the software. Its state upon Reset is unknown. Therefore, if the device is put into the IDLE mode at an arbitrary time, it will stay in the IDLE mode for somewhere between 30 us and the selected time period.

In order to precisely time the duration of the IDLE state, entry into the IDLE mode must be synchronized to the state of the IDLE Timer. The best way to do this is to use the IDLE Timer interrupt, which occurs on every underflow of the bit of the IDLE Timer which is associated with the selected window. Another method is to poll the state of the IDLE Timer pending bit T0PND, which is set on the same occurrence. The Idle Timer interrupt is enabled by setting bit T0EN in the ICNTRL register.

Any time the IDLE Timer window length is changed there is the possibility of generating a spurious IDLE Timer interrupt by setting the T0PND bit. The user is advised to disable IDLE Timer interrupts prior to changing the value of the ITSEL bits of the ITMR

Register and then clear the TOPND bit before attempting to synchronize operation to the IDLE Timer.

NOTE: As with the HALT mode, it is necessary to program two NOP's to allow clock resynchronization upon return from the IDLE mode. The NOP's are placed either at the beginning of the IDLE timer interrupt routine or immediately following the “enter IDLE mode” instruction.

For more information on the IDLE Timer and its associated interrupt, see the description in the Timers Chapter.

Chapter 7

INPUT/OUTPUT

7.1 INTRODUCTION

All COP8 family devices have at least four dedicated input pins ($\overline{\text{RESET}}$, V_{CC}, GND, CKI) and at least one 8-bit I/O port, designated Port G. Additional bidirectional I/O ports, dedicated input ports, and/or dedicated output ports are available on all devices. Refer to the device-specific chapters for information on available ports, packages, and pinouts.

Figure 7-1 is a block diagram illustrating the general structure of bidirectional, dedicated input, and dedicated output pins. A bidirectional I/O pin can be configured by the software to operate as a high-impedance input, an input with a weak pull-up, or as a push-pull output. The operating state is determined by the contents of the corresponding bits in the data register and configuration register. Each bidirectional I/O pin can be used for general-purpose I/O, or in some cases, for a specific alternate function determined by the on-chip hardware. Similarly, a dedicated input pin or dedicated output pin can be used for general-purpose input or output, or for a specific alternate function. (The internal interfaces to alternate functions are not shown in the Figure 7-1.)

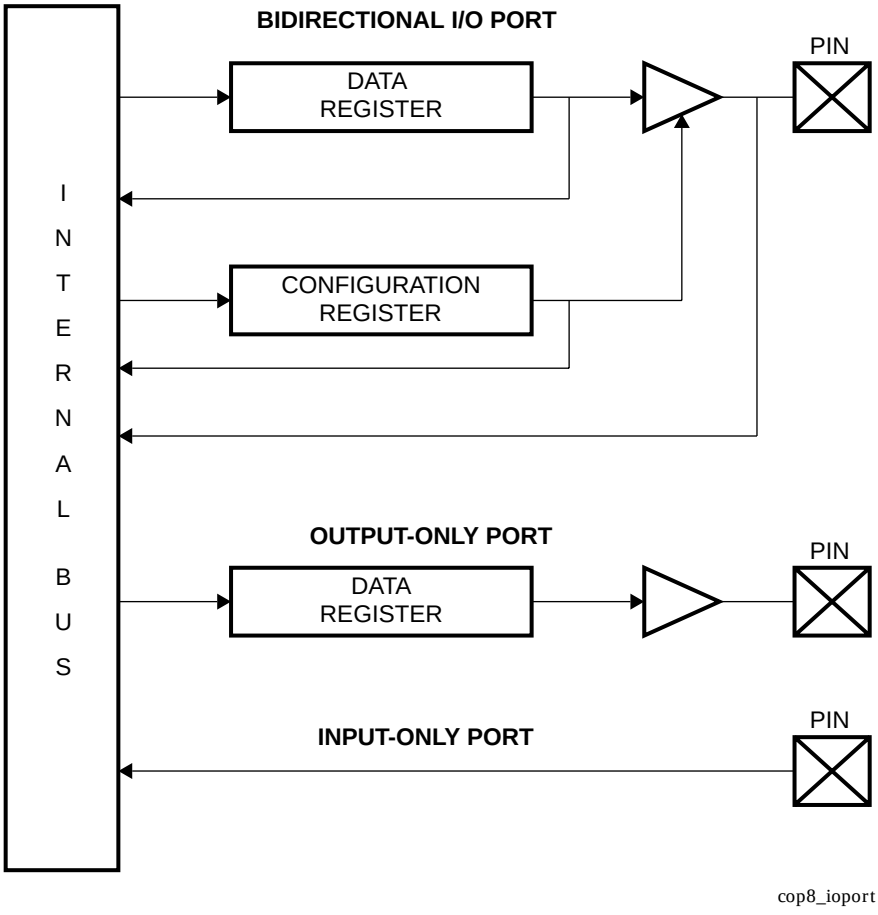


Figure 7-1 COP8 Port Structure

The following sections of this chapter describe each of the individual ports used in COP8 devices. In general, each port contains eight bits. There is usually a memory-mapped data register associated with each port that holds the bit values for the port pins. For bidirectional pins, there is also a configuration register, that specifies whether an I/O pin operates as an input or as an output. Use of the data and configuration registers is described in Section 2.3.4. In some cases, a third memory location is available for reading the state of the port pins directly.

Many pins can be used either for general-purpose I/O or for a specific alternate function. Alternate functions vary from one COP8 device to another, so these functions are described in the device-specific chapters.

Note that not all ports are available in all devices.

7.2 PORT A

Port A is a general-purpose, bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

Any device package that has a Port A, but has fewer than eight Port A pins, contains unbonded, floating pads internally on the chip. The binary value read from these bits is undetermined. The application software should mask out these unknown bits when reading the Port A register, or use only bit-access program instructions when reading Port A. The unconnected bits draw power only when they are addressed (i.e., in brief spikes). All Port A pins have Schmitt triggers on their inputs.

7.3 PORT B

Port B is a general-purpose, bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

Any device package that has a Port B, but has fewer than eight Port B pins, contains unbonded, floating pads internally on the chip. The binary value read from these bits is undetermined. The application software should mask out these unknown bits when reading the Port B register, or use only bit-access program instructions when reading Port B. The unconnected bits draw power only when they are addressed (i.e., in brief spikes). All Port B pins have Schmitt triggers on their inputs.

7.4 PORT C

Port C is a general-purpose, bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

Any device package that has a Port C, but has fewer than eight Port C pins, contains unbonded, floating pads internally on the chip. The binary value read from these bits is

undetermined. The application software should mask out these unknown bits when reading the Port C register, or use only bit-access program instructions when reading Port C. The unconnected bits draw power only when they are addressed (i.e., in brief spikes). All Port G pins have Schmitt triggers on their inputs.

7.5 PORT D

Port D is a general-purpose, dedicated output port. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits.

Port D output pins have high-sink drive capability. Refer to the COP8 data sheets for information on the electrical specifications.

Port D is preset high when the RESET signal goes active (low). If the D2 pin is held low by the external circuit during the Reset state, the microcontroller enters a special mode of operation upon exiting the Reset state. This special mode is used for testing purposes. To avoid entering this mode, the hardware design should ensure that D2 is not pulled low during a Reset operation.

7.6 PORT E

Port E is a general-purpose, bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

Any device package that has a Port E, but has fewer than eight Port E pins, contains unbonded, floating pads internally on the chip. The binary value read from these bits is undetermined. The application software should mask out these unknown bits when reading the Port E register, or use only bit-access program instructions when reading Port E. The unconnected bits draw power only when they are addressed (i.e., in brief spikes). All Port E pins have Schmitt triggers on their inputs.

7.7 PORT F

Port F is a general purpose, bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

Any device package that has a Port F, but has fewer than eight Port F pins, contains unbonded, floating pads internally on the chip. The binary value read from these bits is undetermined. The application software should mask out these unknown bits when reading the Port F register, or use only bit-access program instructions when reading Port F. The unconnected bits draw power only when they are addressed (i.e., in brief spikes).

7.8 PORT G

Port G is a bidirectional I/O port that is present in all COP8 devices. The number of pins available in this port depends on the device type and package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs.

Pins G0, G2, G3, G4, and G5 are general-purpose I/O pins, that support alternate functions such as the timer interface control, external interrupt, and MICROWIRE/PLUS interface. The alternate pin functions are listed in Section 7.10, and described in detail in the chapters devoted to specific COP8 features. For general purposes, these pins are programmed as described in Section 2.3.4.

Pins G0 and G1 have an internal weak pull-up which is enabled during Reset. Pin G1 serves as WDOUT, if the Watchdog and Clock Monitor. Pin G6, if present, is a dedicated input pin. Pin G7 is either a dedicated input or dedicated output, depending on the oscillator mask option selected. With the R-C oscillator or external clock mask option, G7 can be used as a general-purpose input pin, or as the HALT/Restart input pin as described in Chapter 6. With the crystal oscillator mask option, G7 is the clock output pin, CKO.

7.9 PORT L

Port L is a bidirectional I/O port that can be used for general-purpose I/O or for alternate functions such as Multi-Input Wakeup/Interrupt and/or timer I/O. The number of pins available in this port depends on the device type and package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

The Port L inputs have Schmitt triggers. Refer to the COP8 data sheets for information on the electrical specifications.

7.10 ALTERNATE FUNCTIONS

Many general-purpose I/O pins have alternate functions. The software can program each I/O pin to be used either for general-purpose or for a specific alternate function. The chip hardware determines which of the I/O pins have alternate functions. Alternate functions vary from one COP8 device type to another, and from one package type to another.

Port G is present in all COP8 devices. The alternate functions of this port are described below. The Multi-Input Wakeup/Interrupt feature, an alternate function of some ports, is also described below. This feature can be used for waking up from (exiting) the HALT or IDLE mode, and also for providing an additional eight maskable interrupts. For information on the alternate functions of other ports, refer to the device-specific chapters and the chapters describing individual COP8 features.

7.10.1 Port G Alternate Functions

The pins of Port G have the alternate functions listed below.

G0 INTR (External Interrupt Input) (Weak pull-up enabled during Reset)

G1 Watchdog Output (on some COP8 devices) (If enabled in the Option Register)

G2 T1B (Timer T1B input) (Weak pull-up enabled during Reset)

G3 T1A (Timer T1A input/output)

G4 SO (MICROWIRE Serial Output)

G5 SK (MICROWIRE Serial Clock)

G6 SI (MICROWIRE Serial Input)

G7 If crystal oscillator mask option: Dedicated CKO (Clock Output)

If R-C/ external oscillator mask option: HALT/Restart input

For more information on interrupts, timers, MICROWIRE/PLUS, and HALT mode, see the separate chapters covering these subjects.

7.10.2 Multi-Input Wakeup/Interrupt

The Multi-Input Wakeup/Interrupt feature is an alternate function of some COP8 ports. This feature can be used for either of two purposes: to provide separate inputs for waking up (exiting) from the HALT or IDLE mode, or to provide separate edge-triggered maskable interrupts. Figure 7-2 is a block diagram showing the internal logic of the Multi-Input Wakeup/Interrupt circuit.

There are three memory-mapped registers associated with this circuit: WKEDG (Wakeup Edge), WKEN (Wakeup Enable), and WKPND (Wakeup Pending). Each register has eight bits, with each bit corresponding to one of the eight input pins shown in Figure 7-2. All three registers are initialized to zero with a Reset.

The WKEDG register establishes the edge sensitivity for each of the port input pins: either positive-going edges (0) or negative-going edges (1).

The WKEN register enables (1) or disables (0) each of the port pins for the Wakeup/ Interrupt function. Any pin that is to used for the Wakeup/Interrupt function must also be configured as an input pin in its associated configuration register.

The WKPND register contains the pending flags corresponding to each of the port pins (1 for wakeup/interrupt pending, 0 for wakeup/interrupt not pending).

To use the Multi-Input Wakeup/Interrupt circuit, perform the steps listed below. Performing the steps in the order shown will prevent false triggering of a Wakeup/ Interrupt condition. This same procedure should be used following a Reset because the Wakeup inputs are left floating as a result of a Reset, resulting in unknown data on the Port L inputs.

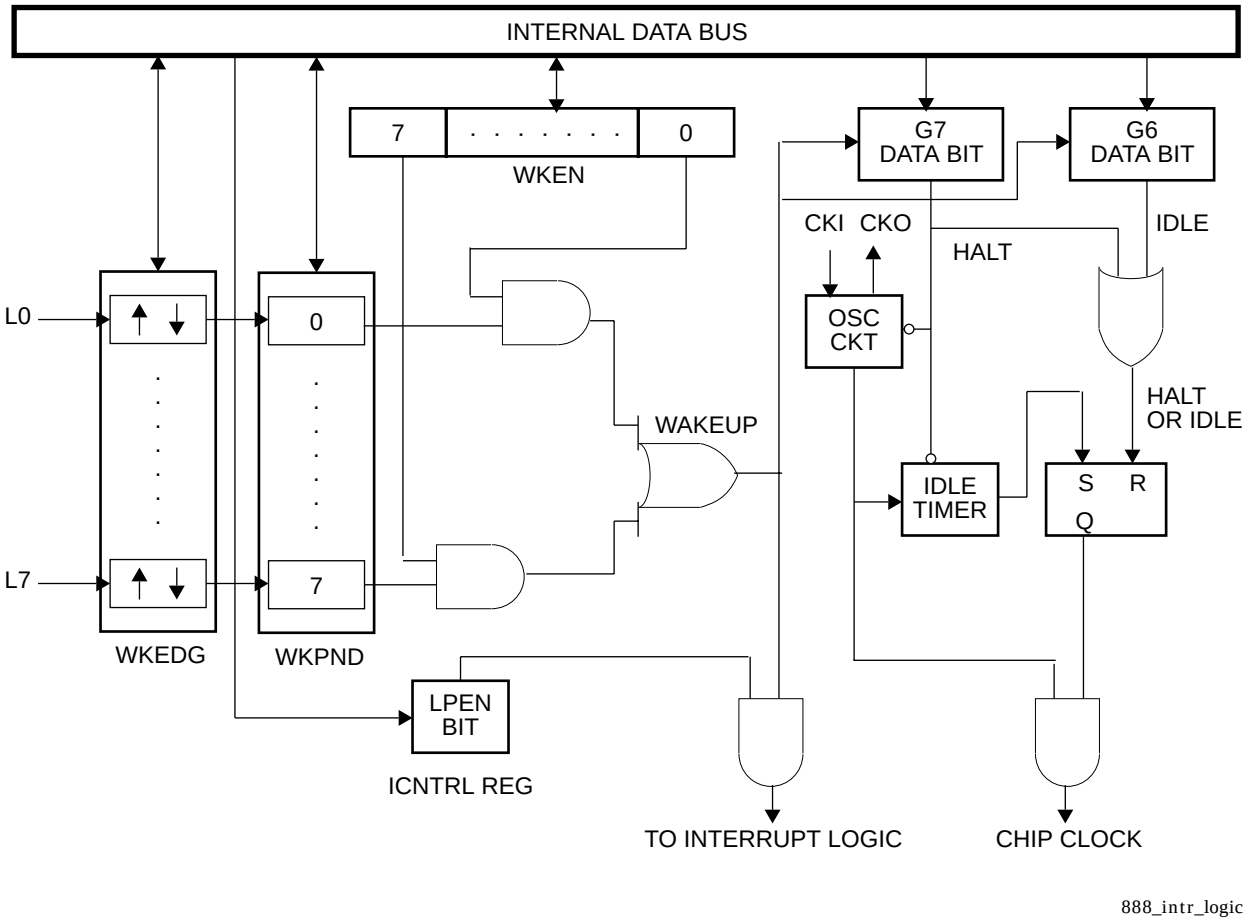


Figure 7-2 Multi-Input Wakeup/Interrupt Logic

1. If necessary, write to the port configuration register to change the desired port pins from outputs to inputs.
2. Write the WKEDG register to select the desired type of edge sensitivity for each of the pins used.
3. Clear the WKPND register to cancel any pending bits.
4. Set the WKEN bits associated with the pins to be used, thus enabling those pins for the Wakeup/Interrupt function.

Once the Multi-Input Wakeup/Interrupt function has been set up, a transition sensed on any of the enabled pins will set the corresponding bit in the WKPND register. This brings the device out of the HALT or IDLE mode (if in that mode), and also triggers a port maskable interrupt if that interrupt is enabled. To enable the port interrupt, set the enable bit in the appropriate control register (refer to the register bit map sections of the device-specific chapters) and the GIE bit in the PSW register. If the port interrupt is enabled, the microcontroller is vectored to the same port interrupt service routine, regardless of which port pin sensed the interrupt condition. The interrupt service routine can read the WKPND register to determine which pin sensed the interrupt.

The interrupt service routine or other software should clear the pending bit. The COP8 will not enter the HALT or IDLE mode as long as any WKPND pending bit is pending and enabled.

Upon Reset, the WKEDG register is configured to select positive-going edge sensitivity for all Wakeup inputs. If the user wishes to change the edge sensitivity of a port pin, use the following procedure to avoid false triggering of a Wakeup/Interrupt condition.

1. Clear the WKEN bit associated with the pin to disable that pin.
2. Write the WKEDG register to select the new type of edge sensitivity for the pin.
3. Clear the WKPND bit associated with the pin.
4. Set the WKEN bit associated with the pin to re-enable it.

The following example illustrates the program steps for changing the edge sensitivity of port bit 5 from positive to negative edges.

```
RBIT    5,WKEN    ;Disable Port bit 5 for Wakeup/Interrupt
SBIT    5,WKEDG   ;Select negative-going edge sensitivity
RBIT    5,WKPND   ;Clear the pending bit
SBIT    5,WKEN    ;Re-enable the bit for Wakeup/Interrupt
```

For more information on interrupts or the HALT/IDLE power-save modes, see the respective chapters on those subjects.

Chapter 8

WATCHDOG AND CLOCK MONITOR

8.1 INTRODUCTION

Some COP8 devices have a circuit called the Watchdog and Clock Monitor. This circuit monitors the operation of the device and reports an abnormal condition by issuing a signal on the G1 output pin, WDOUT. The device-specific chapters indicate whether the Watchdog and Clock Monitor is present in each COP8 family member.

The Watchdog is a circuit that monitors the number of instruction cycles being executed and detects certain types of program execution errors. The application program must periodically write a specific value to the Watchdog Service Register, within specific time intervals. The Watchdog reports any failure of the software to perform this function. Thus, a runaway program or infinite program loop will result in a Watchdog error. The output signal resulting from an error can be used to reset the chip or to perform any other function.

The Clock Monitor is a circuit that detects the absence of a clock signal, or a clock that is running too slowly. A clock error is reported on the same output pin as a Watchdog error.

Using the Watchdog and Clock Monitor functions is optional. The Clock Monitor function can be disabled by the software following a Reset. The Watchdog function always operates, but the user application can simply ignore the output signal generated on the WDOUT pin.

8.2 WATCHDOG OPERATION

The Watchdog circuit looks at the contents of the IDLE Timer (Timer T0) to keep track of the number of instruction cycles being executed. The IDLE Timer is a 16-bit register that counts down continuously at the instruction clock rate or the 32KHz clock as selected in the ITMR Register. (The instruction clock runs at one-tenth the frequency of MCLK.) The IDLE Timer is not memory-mapped, and cannot be read or written by the software.

The application software must write a specific value at periodic intervals into the Watchdog Service Register (WDSVR). This must happen within a “time window”: no more often than once every 2,048 instruction cycles, but at least as often as once every 65,536 cycles initially. The lower limit is fixed at 2,048 instruction cycles instruction cycles, but the upper limit can be programmed to be either 8,192, 16,384, 32,768 or 65,536 instruction cycles. Programming of the upper limit can be performed only once following a chip Reset.

Figure 8-1 is a block diagram showing the logic of the Watchdog circuit.

The Watchdog Service Register, WDSVR, is an 8-bit memory-mapped register. The software “services” this register at periodic intervals by writing a specific value to it. If

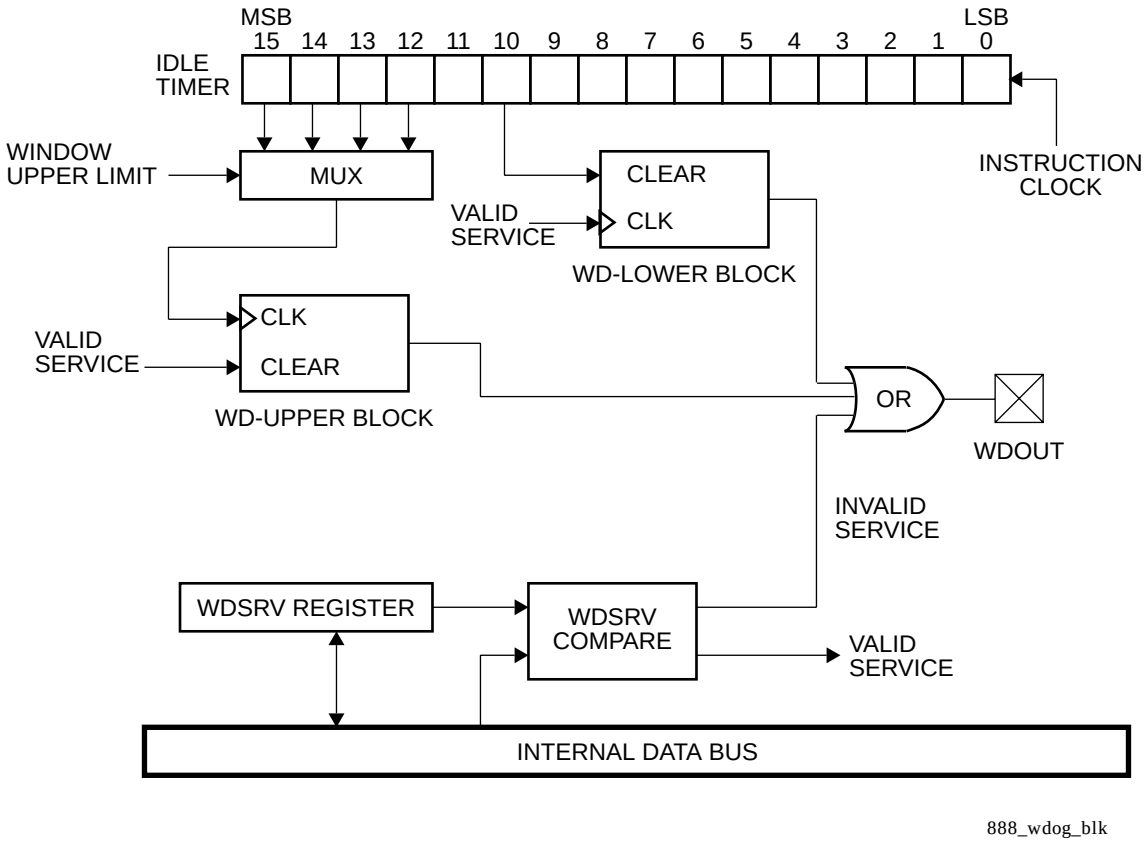


Figure 8-1 Watchdog Logic Block Diagram

the written value is correct, the “Valid Service” signal is asserted; if the value is not correct, it is considered an “Invalid Service,” which triggers a Watchdog error.

The Watchdog logic contains two internal logic blocks called WD-Upper and WD-Lower. The WD-Upper logic block establishes the upper limit of the time window, and the WD-Lower logic block establishes the lower limit. The WD-Upper block can be programmed to use any of the four most significant bits of the IDLE Timer register to clock its counter, thus establishing the upper limit at 8,192, 16,384, 32,768, or 65,536 instruction cycles. The WD-Lower logic block uses bit 10 of the IDLE Timer register to clear its counter, thus establishing a fixed lower limit of 2,048 instruction cycles.

Each of the two logic blocks contains a 1-bit counter. If either counter overflows (is clocked twice without being cleared), a Watchdog error is reported on WDOUT. In the WD-Upper block, the counter is clocked by an IDLE Timer bit at intervals of 8,192, 16,384, 32,768, or 65,536 instruction cycles. The counter is cleared by a valid servicing of WDSVR. Thus, the Watchdog Service Register must be serviced at least as often as the programmed upper limit. In the WD-Lower block, the counter is clocked by a valid servicing of WDSVR, and cleared by bit 10 of the IDLE Timer register ever 2,048 instruction cycles. Thus, the Watchdog Service Register must be serviced no more than once every 2,048 instruction cycles instruction cycles. If WDSVR is serviced not often enough for the WD-Upper block or too often for the WD-Lower block, an overflow occurs and an error is reported.

The specific value that must be written periodically to the WDSVR register is explained in Section 8.4.

Operation of the Watchdog circuit is inhibited during the HALT and IDLE modes. The contents of the IDLE timer are affected by the HALT mode under certain conditions (i.e., when a start-up delay is used) and by the IDLE mode. Upon exit from the HALT or IDLE mode in these cases, the on-chip hardware services the Watchdog circuit automatically. Therefore, upon exit from the HALT or IDLE mode, the application software should service the Watchdog Service Register within the established upper limit of 8,192, 16,384, 32,768, or 65,536 instruction cycles as usual, but not before 2,048 instruction cycles instruction cycles. Otherwise, a Watchdog error may be triggered.

For the HALT mode, the re-starting of the Watchdog count only occurs if a start-up delay is used. If a start-up delay is not used, the Watchdog service window resumes counting normally upon exit from the HALT mode, as if the HALT had never occurred. For more information on the IDLE mode, HALT mode, and HALT mode start-up delay, see Chapter 6. For more information on the IDLE Timer, see Chapter 4.

8.3 CLOCK MONITOR OPERATION

The Clock Monitor constantly checks the chip clock and issues an error signal on the WDOUT pin if the clock is not running, or is running very slowly. Upon Reset, the microcontroller starts out with the Clock Monitor enabled. If the clock is not operating properly at that time, an error signal is issued on the WDOUT pin. If the clock fails to operate correctly at a later time, an error signal is issued when the error is detected. The Clock Monitor works even in the HALT and IDLE modes. Therefore, any entry into the HALT mode is reported as a clock error.

The Clock Monitor circuit monitors the instruction clock, which runs at one-tenth the frequency of the chip clock. An instruction clock running at 10 KHz or higher is interpreted as normal operation. An instruction clock running at 10 Hz or slower is considered too slow, and is reported as an error. The pass/fail threshold frequency can vary within the range of 10 Hz and 10 KHz, so an instruction clock running in that range may or may not trigger an error.

Following a Reset, the Clock Monitor function can be disabled by the software. Configuration of the Watchdog and Clock Monitor functions can be performed only once following a Reset, as explained below.

8.4 CONFIGURATION

The Watchdog and Clock Monitor circuits are inhibited during a Reset (with the $\overline{\text{RESET}}$ signal active), and begin operating as soon as the microcontroller comes out of a Reset. Initially, the Watchdog operates with the service window set to the maximum upper limit of 65,536 instruction cycles. The Clock Monitor will detect a non-functioning clock immediately following the Reset, so the active $\overline{\text{RESET}}$ signal should be made long enough to allow the clock circuit to reach its full amplitude and a stable operating frequency.

The Watchdog and Clock Monitor functions can be configured only once following a chip Reset. Once configured, they cannot be configured again unless there is another Reset. They are configured by writing a data byte to the Watchdog Service Register, WDSVR, a read/write register residing at address C7 Hex. The format of the data byte is shown in Figure 8-2.

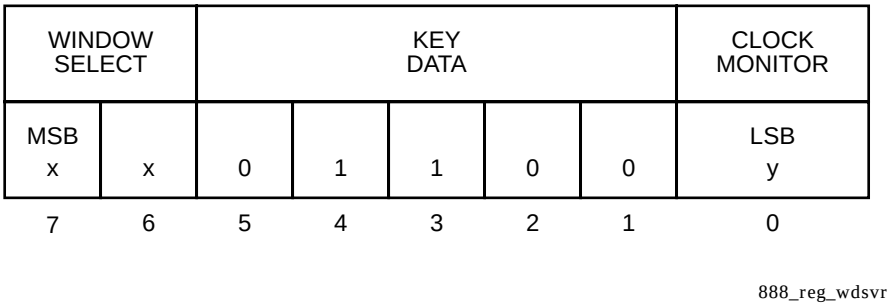


Figure 8-2 Watchdog Service Register (WDSVR) Format

The data byte written to WDSVR for the first time following a Reset configures the Watchdog circuit and the Clock Monitor. To avoid a Watchdog error, this must be done no later than 65,536 instruction cycles after the $\overline{\text{RESET}}$ signal goes inactive. However, it may be done within the first 2,048 instruction cycles without triggering a Watchdog error. Immediately following Reset is the recommended time.

The first time WDSVR is written, the two most significant bits are used to select the upper limit of the Watchdog service window, as indicated in the table below.

WDSVR Bit 7	WDSVR Bit 6	Clock Monitor Bit 0	Service Window for High Speed Mode (Lower-Upper Limits)	Service Window for Dual Clock & Low Speed Modes (Lower-Upper Limits)
0	0	X	2048–8k t_c Cycles	2048–8k Cycles of LS Clk
0	1	X	2048–16k t_c Cycles	2048–16k Cycles of LS Clk
1	0	X	2048–32k t_c Cycles	2048–32k Cycles of LS Clk
1	1	X	2048–64k t_c Cycles	2048–64k Cycles of LS Clk
X	X	0	Clock Monitor Disabled	Clock Monitor Disabled
X	X	1	Clock Monitor Enabled	Clock Monitor Enabled

Bits 5-4-3-2-1 of WDSVR must always be written with the binary value 01100. This is a fixed code that cannot be changed. If any other value is written at any time, a Watchdog error is triggered.

The first time WDSVR is written, the least significant bit controls the Clock Monitor feature. If a 1 is written, the Clock Monitor remains enabled. If a 0 is written, the Clock Monitor is disabled.

The Watchdog and Clock Monitor can be programmed only once following a Reset. The exact same 8-bit value used for programming must be written to the WDSVR register at each Watchdog service. Any attempt to write a different value to WDSVR after the first time will trigger a Watchdog error, and will not affect the Watchdog or Clock Monitor configuration.

The WDSVR register can be read as well as written. The values read back from bits 7, 6, and 0 reflect the values that were written to that register at the time of programming (the first time following Reset). However, bits 5 through 1 always read back as all zeros. The code bits written to the register (01100) cannot be read back.

8.5 ERROR REPORT ON WDOUT

An abnormal condition detected by the Watchdog or Clock Monitor is reported by issuing a signal on the G1 output pin, WDOUT. In devices that use the Watchdog and Clock Monitor, the G1 pin operates independently from the Port G configuration and data registers, allowing bit 1 of those registers to be used as independent software flags. Enabling the WDOUT function may be a programmable option; see the device specific chapter for details.

Under normal conditions, the WDOUT pin remains in the high-impedance state. When a Watchdog error is detected, the pin is pulled low for a period ranging from 16 to 32 instruction cycles, after which it returns to the high-impedance state. During this time that WDOUT is low, any Watchdog service (writing to WDSVR) is ignored. After a Watchdog error, the Watchdog service window restarts when the WDOUT pin returns to the high-impedance state.

In the case of a Clock Monitor error, the WDOUT pin remains in the low state as long as the clock is not operating properly. When the clock returns to normal operation, the WDOUT pin returns to the high-impedance state after a delay of 16 to 32 instruction cycles.

The WDOUT pin can be used in a variety of ways. It can be simply ignored, in which case there is no need to service the Watchdog Service Register. If WDOUT is used, a pullup resistor is needed to pull the pin high during normal operation. (In some newer COP8 devices, this pullup resistor is provided internally; see the next section of this chapter for details.) When a Watchdog or Clock Monitor error is detected, the pin goes low. This signal can be used to reset the microcontroller or the whole system. In that case, the circuit hardware should ensure that the $\overline{\text{RESET}}$ signal generated from WDOUT going low meets the system requirements. Note that the WDOUT signal itself will go low for only 16 to 32 instruction cycles for a Watchdog error.

The state of the WDOUT pin is unknown immediately following a Reset. If it starts out active-low, it will return to the high-impedance state within 16 to 32 instruction cycles if there is no Watchdog or Clock Monitor error.

8.6 G1/WDOUT PIN OPTION

Some newer COP8 devices, such as the COP8 Flash Family have an improved design with respect to the G1/WDOUT pin:

- A pullup resistor is provided internally, making it unnecessary to use an external pullup resistor for WDOUT operation.

- If the Watchdog feature is not needed, you can disable it by programming a bit in the Configuration register at the same time that you program the application software. In that case, G1 becomes available to use as a general-purpose I/O pin.
- This resistance should be considered if you are converting an application from an older device without the pullup resistor to a newer device with the pullup resistor.

The WDOUT signal can be used to reset the device. In older devices, some external circuitry is required to generate a signal from WDOUT which can be applied to the $\overline{\text{RESET}}$ pin, due to Reset timing requirements upon power-up. However, in newer devices with the built-in WDOUT pullup resistor and with the on-chip power-on reset or Brownout Reset option enabled, you can make a direct wire connection from the WDOUT pin to the $\overline{\text{RESET}}$ pin, without using any external components at all. In this implementation, the only conditions that reset the device are power-up, Brownout and Watchdog errors.

8.7 WATCHDOG OPERATION DURING USER ISP/VIRTUAL E^2 OPERATION

The User ISP/ Virtual e commands service the WATCHDOG. The WATCHDOG lower window limit is suppressed during Boot ROM execution and all commands service the WATCHDOG immediately prior to returning to Flash execution. The user must ensure that the watchdog is not serviced within the lower window after returning from Boot ROM.

Chapter 9

10-BIT SUCCESSIVE APPROXIMATION A/D CONVERTER

9.1 INTRODUCTION

Some COP8 Feature Family devices have a 10-bit multi-channel, multiplexed-input Analog-to-Digital (A/D) Converter. The A/D Converter receives an analog voltage signal on an input pin (or a pair of input pins), and converts the analog signal into a 10-bit digital value using successive approximation. The digital value can then be read by the software from a pair of memory-mapped registers. The input voltage range is defined by the AV_{CC} and AGND voltages.

The A/D Converter is useful in applications where the software needs to read a value (for example, temperature or speed) from an analog sensor. Up to sixteen different single-ended or eight different differential inputs can be handled by a single COP8 device through the multiplexed input channels.

In both Single Ended and Differential modes, there is the capability to connect the analog multiplexor output and A/D converter input to external pins. This provides the ability to externally connect a common filter/signal conditioning circuit for the A/D Converter.

9.2 A/D OPERATION

Figure 9-1 is a block diagram showing the structure of the COP8 successive-approximation A/D Converter. There are three memory-mapped registers in the A/D Converter circuit: the A/D Converter control register (ENAD), used for configuring and enabling the A/D Converter, and the A/D Result registers (ADRSTH and ADRSTL), a pair of read-only registers containing the result of the conversion.

The input voltage range is determined by two fixed reference voltages supplied to the device through pins AGND (analog ground) and AV_{CC} (analog power supply voltage). AGND is the lower limit and AV_{CC} is the upper limit of the input voltage range. AV_{CC} and AGND must be connected to V_{CC} and GND respectively.

Analog voltages are received on pins ACH0 through ACH15 (Analog Channels 0 through 15), which are alternate functions of the Port A pins and the Port B pins. A multiplexer selects one channel for a single-ended conversion, or a pair of channels for differential-pair conversion, based on the value programmed into the ENAD register. The input voltage (or for a differential pair, the difference between the two input voltages) is sent to a comparator. The comparator compares the input voltage with a voltage generated by an internal digital-to-analog converter.

The A/D Converter uses successive approximation to determine the analog input voltage. The internal digital-to-analog converter generates a test voltage, which is initially set to the middle of the possible voltage range, halfway between the voltages on AGND and

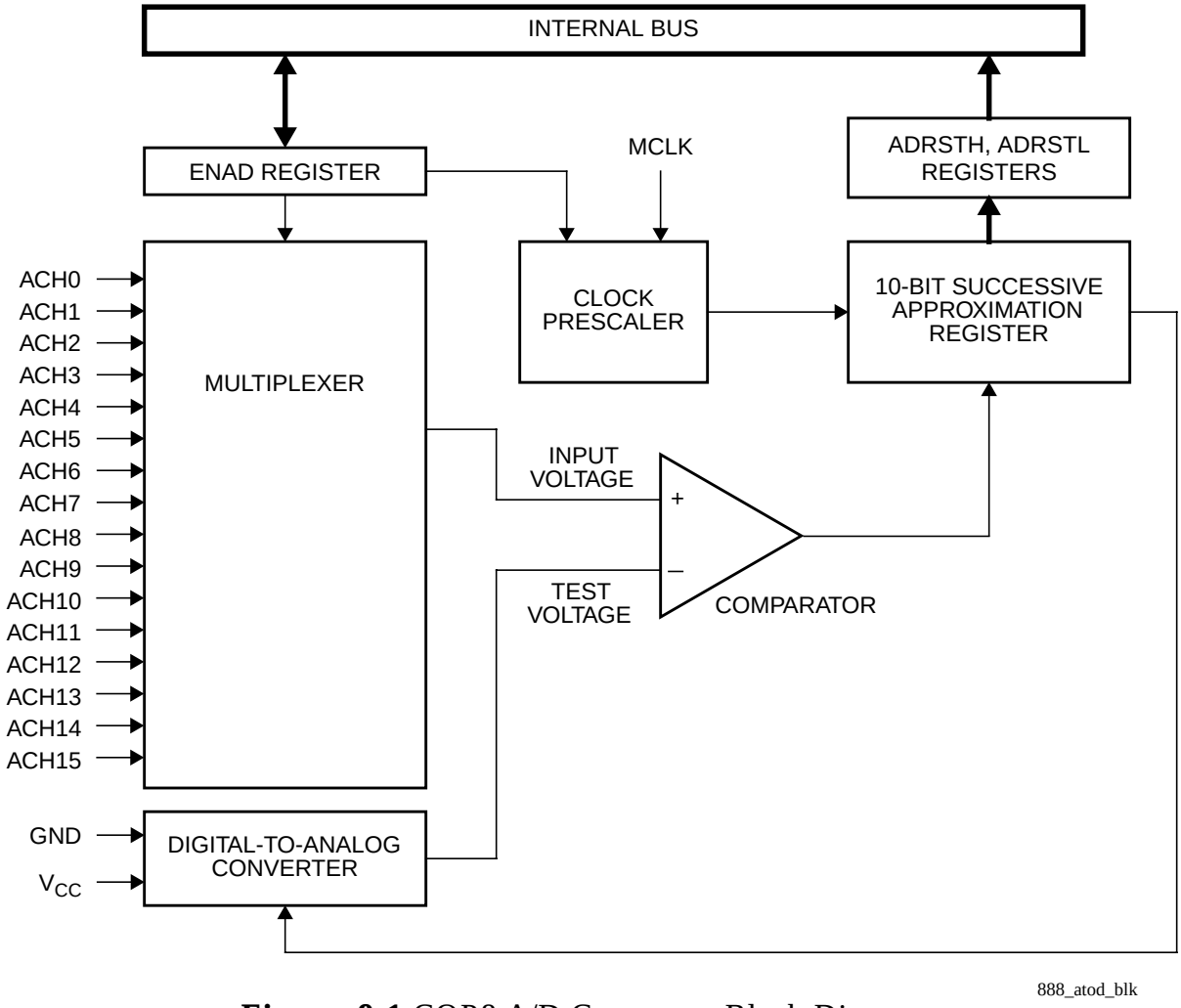


Figure 9-1 COP8 A/D Converter Block Diagram

AV_{CC}. After the comparison is made, another comparison is made with the test voltage set to the middle of the new possible range (either 1/4 or 3/4 between AGND and AV_{CC}). This is repeated until the input voltage is determined to a precision of 1/1024 of the full input voltage range (ten iterations).

Each time a comparison is made, a bit is set or cleared in the Successive Approximation Register, depending on whether the input voltage is higher or lower than the test voltage. The bit values in the Successive Approximation Register are determined from most to least significant bit, generating a 10-bit digital value that is proportional to the analog input voltage. The value 000 Hex represents the lowest voltage (AGND), and a value of 3FF Hex represents the highest voltage (AV_{CC}). At the end of the conversion, the 10-bit value is transferred to the ADRSTH and ADRSTL registers, allowing the final result to be read by the software. The Successive Approximation Register cannot be accessed directly by the software.

The A/D Converter uses its own clock, which is generated by scaling down the Main CPU Clock (MCLK) to a lower frequency and must be within a specified range of frequencies. The clock prescaler circuit allows a choice of two different divide-by factors for generating the A/D clock. The choice is made by programming a bit in the ENAD register. This feature allows a certain amount of control over the trade-off between speed and accuracy

of the A/D Converter without changing the chip clock speed, as explained later in this chapter.

A single conversion takes 15 A/D clock cycles: 3 cycles for sampling, 1 cycle for auto-zeroing the comparator, 10 cycles for converting, 1 cycle for loading the result into the result registers, for stopping and for re-initializing. The total conversion time depends on the chip clock speed and the divide-by factor used for generating the A/D clock. If analog voltages on different channels are to be monitored simultaneously, they must be time-multiplexed by the application software.

The A/D Converter circuit is powered down when the device enters the HALT or IDLE mode. If the A/D Converter is running when this happens, the conversion is canceled, and the conversion is restarted, without resampling, upon exit from the HALT or IDLE mode. The application program should always terminate and restart any pending conversions due to possible corruption of the sampled voltage upon exiting the HALT or IDLE mode.

9.3 A/D CONVERTER REGISTERS

Three memory-mapped registers are used with the A/D Converter: the ENAD register, the ADRSTH register and the ADRSTL register. The ENAD register is a read-write memory location used for controlling the operation of the A/D Converter. The ADRSTH and ADRSTL registers are read-only registers that contain the most recent A/D conversion result.

The A/D Converter is controlled by writing a byte to the ENAD register. The data byte written to this register enables (or disables) the A/D Converter clock, sets the divide-by factor for generating the clock, selects the operating mode (single-ended or differential inputs), and selects the channel or channel pair that is to receive the analog input. The register bit map for the ENAD register is shown in Table 9-1.

Table 9-1 ENAD, A/D Converter Control Register

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
ADCH3	ADCH2	ADCH1	ADCH0	ADMOD	MUX	PSC	ADBSY

ADCH3–ADCH0:	Channel Selection bits
ADMOD:	Mode Selection (0 = single-ended, 1 = differential)
MUXOUT:	Analog Mux Output and Sample and Hold Input Enable
PSC:	A/D clock prescaler selection bit
ADBSY:	A/D enable and Busy bit

The ENAD register is cleared upon reset. This inhibits operation of the A/D clock and disables the A/D converter. To begin using the A/D converter, it is only necessary to write the proper bit values to the ENAD register, as described in detail below.

The ADRSTH and ADRSTL register contents are unknown following a reset.

9.3.1 Channel and Mode Selection

Analog voltages are received on pins ACH0 through ACH15, which are alternate functions of Port A, pins A0 through A7, and Port B, pins B0 through B7. These pins should be configured as inputs.

The A/D Converter uses one channel for a single-ended conversion, or a pair of channels for differential-pair conversion. The single-ended or differential mode is selected with bit 3 of the ENAD register: 0 for single-ended, or 1 for differential. The channel number (or set of two channel numbers) is selected with bits 7, 6, 5 and 4 of the same register, as indicated in Table 9-2. This 4-bit field selects one of the sixteen channels to be the V_{IN+}. The mode selection and the mux output determine the V_{IN-} input. When MUX = 0, all sixteen channels are available, as shown in Table 9-2. When MUX = 1, only fourteen

Table 9-2 A/D Converter Channel Selection when the Multiplexor Output is Disabled

Select Bits				Mode Select ADMOD = 0 Single Ended Mode	Mode Select ADMOD = 1 Differential mode	Mux Output Disabled
ADCH3	ADCH2	ADCH1	ADCH0	Channel No.	Channel Pairs (+. -)	MUX
0	0	0	0	0	0, 1	0
0	0	0	1	1	1, 0	0
0	0	1	0	2	2, 3	0
0	0	1	1	3	3, 2	0
0	1	0	0	4	4, 5	0
0	1	0	1	5	5, 4	0
0	1	1	0	6	6, 7	0
0	1	1	1	7	7, 6	0
1	0	0	0	8	8, 9	0
1	0	0	1	9	9, 8	0
1	0	1	0	10	10, 11	0
1	0	1	1	11	11, 10	0
1	1	0	0	12	12, 13	0
1	1	0	1	13	13, 12	0
1	1	1	0	14	14, 15	0
1	1	1	1	15	15, 14	0

channels are available as shown in Table 9-3.

When the A/D Converter is used in differential mode, the differential voltage is converted to a single ended voltage in the sample and hold circuit. This single-ended voltage is used in the conversion.

9.3.2 Multiplexor Output Select

This 1-bit field allows the output of the A/D multiplexor and the input to the A/D to be connected directly to external pins. This allows for an external, common filter/signal conditioning circuit to be applied to all channels. The output of the external conditioning circuit can then be connected directly to the input of the Sample and Hold input on the A/D Converter. See Figure 9-2 for the single ended mode diagram, where the multiplexor

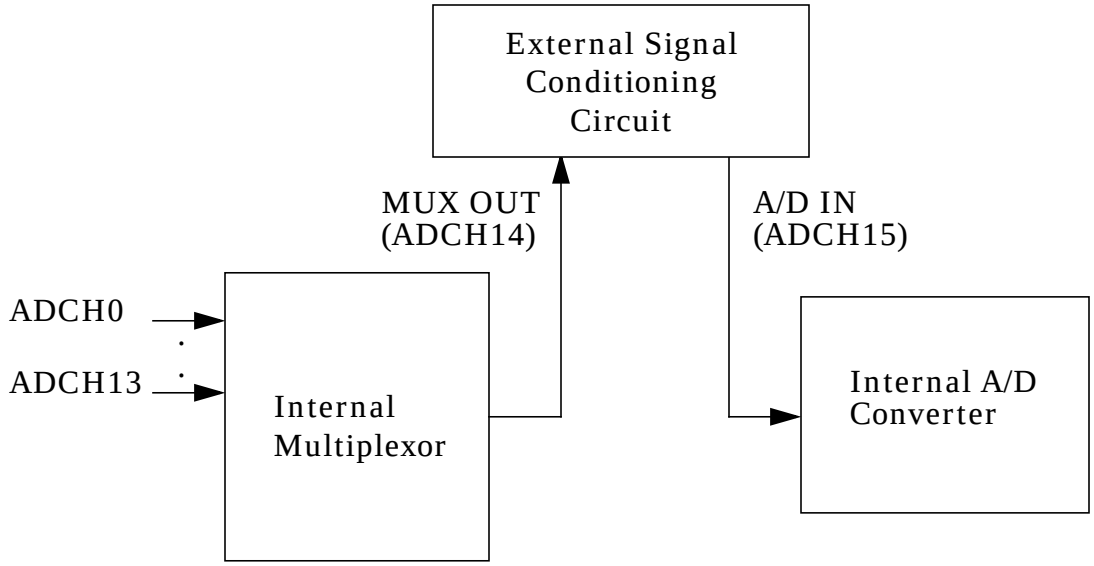


Figure 9-2 A/D with Single Ended Mux Output Feature Enabled

output is connected to ADCH14 and the A/D input is connected to ADCH15. For Differential mode, the differential multiplexor outputs are available and should be

converted to a single ended voltage for connection to the A/D Converter Input. See Figure 9-3.

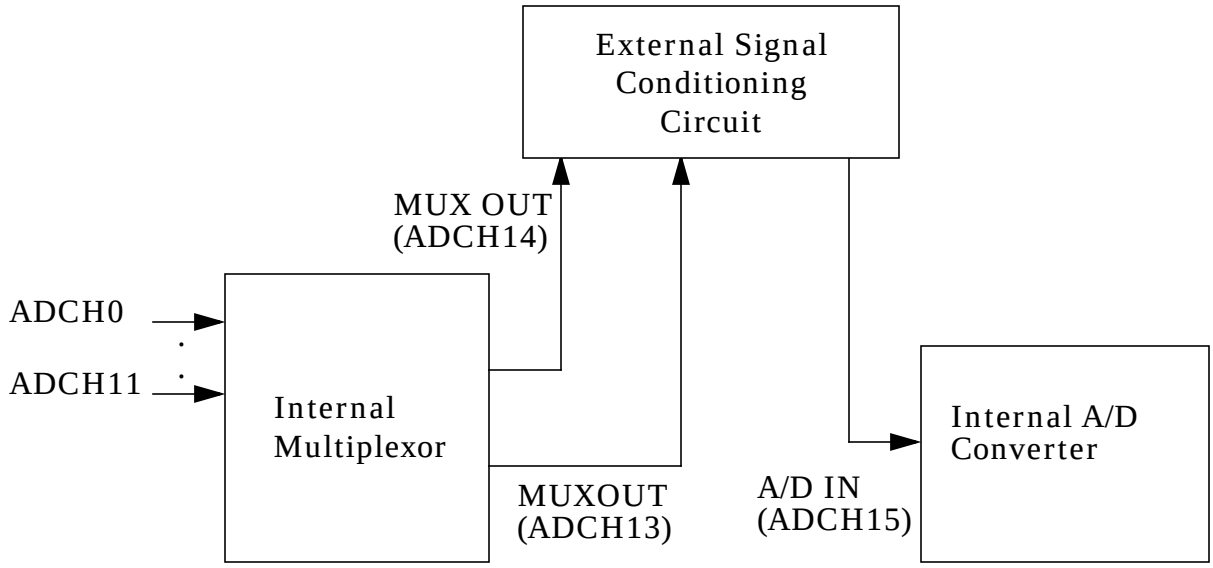


Figure 9-3 A/D with Differential Mux Output Feature Enabled

The channel assignments for this mode are shown in Table 9-3.

Table 9-3 A/D Converter Channel Selection when the Multiplexor Output is Enabled

Select Bits				Mode Select ADMOD = 0 Single Ended Mode	Mode Select ADMOD = 1 Differential mode	Mux Output Enabled
ADCH3	**ADCH2**	**ADCH1**	**ADCH0**	**Channel No.**	**Channel Pairs (+. -)**	**MUX**
0	0	0	0	0	0, 1	1
0	0	0	1	1	1, 0	1
0	0	1	0	2	2, 3	1
0	0	1	1	3	3, 2	1
0	1	0	0	4	4, 5	1
0	1	0	1	5	5, 4	1
0	1	1	0	6	6, 7	1
0	1	1	1	7	7, 6	1
1	0	0	0	8	8, 9	1
1	0	0	1	9	9, 8	1

Table 9-3 A/D Converter Channel Selection when the Multiplexor Output is Enabled

Select Bits				Mode Select ADMOD = 0 Single Ended Mode	Mode Select ADMOD = 1 Differential mode	Mux Output Enabled
ADCH3	**ADCH2**	**ADCH1**	**ADCH0**	**Channel No.**	**Channel Pairs (+. -)**	**MUX**
1	0	1	0	10	10, 11	1
1	0	1	1	11	11, 10	1
1	1	0	0	12	Not used (Note 2)	1
1	1	0	1	13	ADCH13 is Mux Output - (Note 2)	1
1	1	1	0	ADCH14 is Mux Output (Note 1)	ADCH14 is Mux Output + (Note 1)	1
1	1	1	1	ADCH15 is A/D input (Note 1)	ADCH15 is A/D input (Note 1)	1

NOTES: 1. This Input Channel Selection should not be used when the Multiplexor Output is enabled

2. This Input Channel Selection should not be used in Differential Mode when the Multiplexor Output is enabled

When using the Mux Output feature, the delay though the internal multiplexor to the pin, plus the delay of the external filter circuit, plus the internal delay to the Sample and Hold may exceed the three clock cycles that's allowed in the conversion. For this reason, whenever the MUX bit = 1, the channel selected by ADCH3:0 bits is constantly enabled and gated to the mux output pin. The input path to the A/D converter is also enabled. This allows the input channel to be selected and settled before starting a conversion. The sequence to perform conversions using the Mux Out feature is a multistep process and is listed below.

1. Select the desired channel and operating mode and load them into ENAD without setting ADBSY.
2. Wait the appropriate time until the analog input has settled. This will depend on the application and the response of the external circuit.
3. Select the same desired channel and operating modes used in step 1 and load them into ENAD and also set ADBSY. This will start the conversion. This can be done by using the instruction "SBIT ADBUSY,ENAD"
4. Retreive the results from the result registers.

9.3.3 Prescaler Selection

The A/D clock is generated by dividing MCLK clock. The resulting clock period is an integer multiple of MCLK. The divide-by factor can be set to 1 or 12. However, the resulting A/D clock frequency must be between 33 KHz and 1.67 MHz for proper operation of the A/D circuit.

With a CKI clock running at 10 MHz (20 MHZ MCLK and 2 MHz instruction cycle), a divide-by factor of 12 results in an A/D clock frequency of 1.67 MHz, the maximum allowed A/D clock speed. Therefore, with the CKI clock running at 10 MHz, the selected divide-by factor must be 12. A lower divide-by factor can be used only if the CKI clock is running slower than 0.833 MHz (1.67 MHz MCLK). In any case, the resulting A/D clock frequency must be between 65.536 KHz and 1.67 MHz.

The prescaler divide-by factor is programmed by writing bit 1 in the ENAD register. Table 9-4 shows the binary values used for programming this part of the register, and the resulting A/D clock for each binary value.

Table 9-4 A/D Converter Clock Prescale

PSC	Clock Select
0	MCLK Divide by 1
1	MCLK Divide by 12

9.3.4 Busy Bit

The ADBSY bit of the ENAD register is used to start and stop the A/D conversion. When ADBSY is cleared, the prescale logic is disabled and the A/D clock is turned off. Setting the ADBSY bit starts the A/D clock and initiates a conversion based on the mode select value currently in the ENAD register. Normal completion of an A/D conversion clears the ADBSY bit and turns off the A/D converter.

To stop and restart a conversion already in progress, first write a zero to the ADBSY bit to stop the current conversion and then write a one to ADBSY to start a new conversion. This can be done in two consecutive instructions.

9.4 MULTI-CHANNEL CONVERSION

The A/D Converter allows the use of up to sixteen single-ended or eight differential-pair inputs. However, there is only one A/D converter, with only one pair of result registers. If multiple channels are to be monitored simultaneously, they must be time-multiplexed by the application software.

Figure 9-4 shows an example of an assembly language program that performs this function. The program makes a request for an analog-to-digital conversion from each of

sixteen single-ended channels in sequence, reads the thirty-two result bytes, and writes the results into a block of data memory. Operation of the program is explained below.

```
                        ;A/D, convert 16 channels    Bytes/Cycles
ADC8:   RC              ;Reset carry bit                        1/1
        LD   ENAD,#3    ;Select ACH0 conversion and PSC=12      3/3
                        ;Start the conversion
        LD   B,#ENAD    ;B pointer to ENAD register             2/2
        LD   X,#DEST    ;X pointer to result destination        2/3
        LD   X,#DEST    ;Needed for timing on first conversion2/3
        LD   X,#DEST    ;Needed for timing on first conversion2/3
        LD   X,#DEST    ;Needed for timing on first conversion2/3
LOOP:   LD   A,[B]      ;Get previous ENAD contents             1/1
        ADC  A,#011     ;Change to next input channel           2/2
        X    A,[B+]     ;Request next conversion                1/2
        LD   A,[B+]     ;Get previous result from ADRSTH        1/2
        X    A,[X+]     ;Store result and incr. pointer         1/3
        LD   A,[B-]     ;Get previous result from ADRSTL        1/2
        X    A,[X+]     ;Store result and incr. pointer         1/3
        DRSZ B          ;Decrement B to point back to ENAD      1/3
                        ; (Never goes to zero, never skips)
        IFNC            ;Test ADC instr. for overflow           1/1
        JP   LOOP       ;Loop back for next channel             1/3
```

Figure 9-4 A/D Conversion Routine

The program was written with execution speed as the primary consideration. The comment text in the program listing shows the purpose of each instruction, the number of bytes of program memory used by the instruction, and the number of instruction clock cycles required to execute the instruction.

At the start of the program, the first conversion is initiated, specifying channel ACH0 with a clock divide-by factor of 12. ENAD, ADRSTH and ADRSTL will be accessed with the B pointer, so the B register is initialized to ENAD. The results will be stored in a block of data memory occupying 32 bytes starting at DEST. The data memory will be accessed with the X pointer, so the X register is initialized to DEST.

The program loop first reads the ENAD register, changes the control byte to specify the next single-ended channel (ACH1 the first time through the loop), and uses the X (exchange) instruction to request the next conversion. The X instruction also increments the B register, setting up the next instruction to access the ADRSLT register. (The ADRSLT register immediately follows the ENAD register in data memory.) The most significant result byte from the previous conversion is loaded into the Accumulator from ADRSTH, and at the same time, the B pointer is incremented. The result byte is written into data memory using the X pointer. The least significant result byte from the previous conversion is loaded into the Accumulator from ADRSTL, and at the same time, the B pointer is decremented in preparation for the next pass through the loop. The result byte is written into data memory using the X pointer. The B pointer must be decremented one more time to reset it to point to ENAD for the next conversion. The IFNC instruction tests for overflow from the ADC instruction, which will happen after all 32 result bytes have been written (sixteen passes through the loop).

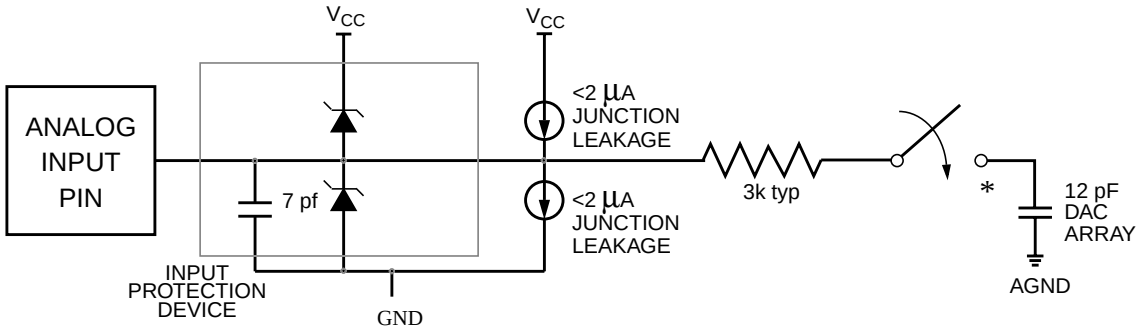
In each iteration of the loop, the program requests a new conversion and quickly reads the results of the previous conversion. The old result remains in the ADRSTx registers until overwritten with the new result, which takes 15 A/D clock cycles, or 180 MCLK clock cycles. The loop section of the program takes 22 instruction clock cycles to execute, or 220 MCLK clock cycles. Thus, the new result becomes available before it is needed in the next iteration of the loop.

The program code uses 25 bytes of program memory. For execution, the whole routine takes 368 instructions cycles: 18 cycles for initial setup, 330 cycles for the first fifteen loops, and 20 cycles for the last iteration of the loop (1 cycle to skip over the JP instruction). With a 10 MHz CKI clock, the routine takes 184 microseconds.

9.5 SPEED, ACCURACY, AND HARDWARE CONSIDERATIONS

The maximum allowed A/D clock speed of 1.67 MHz results in a clock period of 600 ns. The time required for one A/D conversion is 15 A/D clock cycles, or 9.0 microseconds. This is the shortest possible conversion time. In some cases, a longer conversion time may be necessary or desirable in order to ensure accurate conversions. This is because it takes some time for the analog input signal to charge the capacitive load on the A/D input.

Figure 9-5 shows the internal operation of an analog input pin in the single-ended mode, together with the capacitive loads and input protection components on the pin. The analog switch is kept open most of the time. For an A/D conversion, the switch is closed for the duration of three A/D clock cycles, and the voltage is sampled during this period.



*The analog switch is closed only during the sample time.

Figure 9-5 Analog Input Pin Internal Operation

The A/D clock period should be made long enough to allow the capacitor to charge up to the full voltage (three A/D clock periods) supplied to the analog input pin. Whether the minimum A/D clock period of 600 ns is sufficient depends on the output impedance of the analog source and the accuracy requirements of the application. In general, the benefits and costs of using a slower A/D clock should be considered when the analog source impedance exceeds 3 KΩ. As a rule of thumb, you should increase the A/D clock period in proportion to the source impedance. For example, for a source impedance of 6 KΩ, use twice the minimum clock period of 600 ns, or 1200 ns (833 KHz). The A/D clock can be slowed down either by increasing the programmed divide-by factor or by reducing the

CKI clock frequency, or a combination of both. The A/D clock frequency should not be reduced below 65.536 KHz.

Both electrical noise and digital clock coupling to the analog inputs can cause inaccurate conversions. For the most accurate possible results, design the circuit board to minimize noise on these inputs. Keep the leads to the analog input pins as short as possible, and as far away as possible from clock leads.

Chapter 10

USART

10.1 USART

Some COP8 devices have an on-chip Universal Synchronous/Asynchronous Receiver/ Transmitter (USART), which can be used for transmitting and receiving data serially, either synchronously or asynchronously. This full-duplex, double-buffered USART is fully programmable, and can be configured into a wide variety of operating modes. It offers the following major features:

- Full-duplex operation
- Fully programmable serial interface options, including baud rate, start bit, data length (7, 8, or 9 bits), parity bit (even/odd, mark, space, or none), and stop bits (1 or 2)
- Accurate baud rate generation using the chip clock (no additional crystal necessary)
- Complete status reporting
- Separate interrupt vectors for Receive Buffer Full and Transmit Buffer Empty
- Independent clock inputs for the transmit and receive sections (either internal or external)
- Asynchronous or synchronous operation
- Receiver Attention mode for better networking capability
- Error detection circuitry and diagnostic self-test capability

10.1.1 USART Operation Overview

Figure 10-1 is a block diagram of the COP8 USART circuit. There are three major sections to the circuit: the receiver, transmitter, and control sections.

The receiver section receives serial data on the RDX pin. Serial data bits are shifted into the Receive Shift register, low-order bit first. When a full byte is received, it is transferred to the receive buffer (RBUF), a memory-mapped register, and the receive buffer full flag (RBFL) is set. If the USART receiver interrupt is enabled, an interrupt is also generated. The software routine reads and processes the byte in RBUF.

The transmitter section sends out serial data on the TDX pin. When the transmit buffer register (TBUF) is empty, the transmit buffer empty flag (TBMT) is set. If the USART transmitter interrupt is enabled, an interrupt is also generated on the buffer empty condition. The software then writes the next byte to be transmitted into the transmit buffer register (TBUF), a memory-mapped register. At the appropriate time, the data

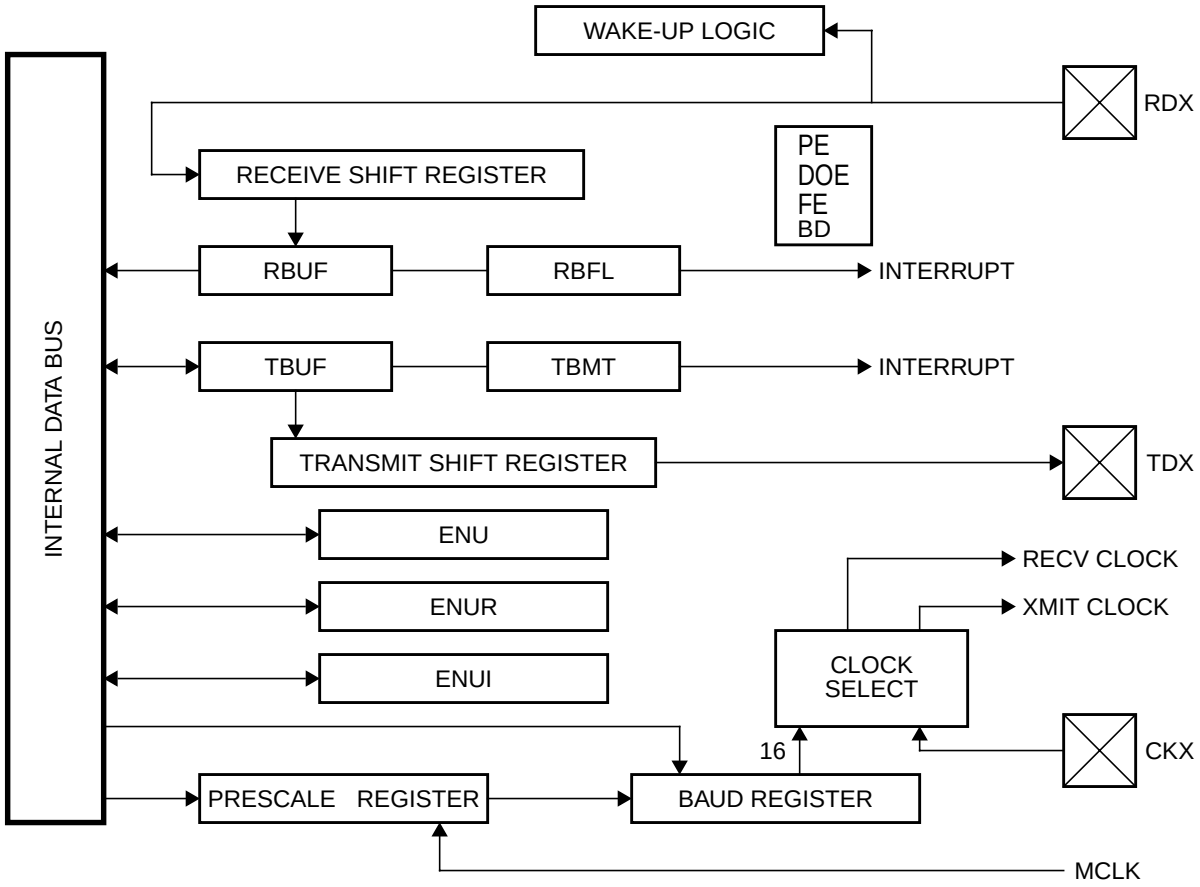


Figure 10-1 USART Block Diagram

byte is transferred from TBUF into the Transmit Shift register. Serial data bits are shifted out of the Transmit Shift register on the TDX pin, low-order bit first.

The software controls the USART using three memory-mapped registers: the Control and Status Register (ENU), the Receive Control and Status Register (ENUR), and the Interrupt and Clock Source Register (ENUI). Some bits in these registers are control bits, while others are status bits. Two more memory-mapped registers are used to select the basic USART baud clock rate: the Prescaler (PSR) and Baud (BAUD) registers. The clock source for generating the baud clock can be either the chip clock (CKI) or an external clock signal received on the CKX pin.

10.1.2 USART Registers

The USART contains seven memory-mapped registers, located in data memory at addresses xxB8 through xxBE Hex. Three registers (ENU, ENUR, and ENUI) are used for configuring the USART and two registers (PSR and BAUD) are used for setting the baud rate. The USART transmitter section contains two linked registers, the Transmit Buffer (TBUF) and Transmit Shift registers. The USART receiver section also contains two linked registers, the Receive Shift and Receive Buffer (RBUF) registers. The two shift registers are not memory mapped, and cannot be accessed directly by the software.

The bit maps for the control and status registers ENU, ENUR, and ENUI, and the baud selection registers PSR and BAUD, are shown in Tables 10-1 to 10-5.

Table 10-1 ENU, USART Control and Status Register (Address xxBA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEN	PSEL1	XBIT9/PSEL0	CHL1	CHL0	ERR	RBFL	TBMT

PEN:	Parity enable bit (0 = Parity disabled, 1 = Parity Enabled
PSEL1:	Parity selection bit (with PSEL0), as indicated below:
	PSEL1-PSEL0 = 00 Odd parity, if parity is enabled
	PSEL1-PSEL0 = 01 Even parity, if parity is enabled
	PSEL1-PSEL0 = 10 Mark (1), if parity is enabled
	PSEL1-PSEL0 = 11 Space (0), if parity is enabled
XBIT9/PSEL0:	For nine data bits per frame, this bit is XBIT9, or Transmit Bit 9, the ninth data bit transmitted. For seven or eight data bits per frame, this bit serves as PSEL0 (see PSEL1 above).
CHL1-CHL0:	Frame format selection bits (number of data bits):
	CHL1-CHL0 = 00 Eight data bits per frame.
	CHL1-CHL0 = 01 Seven data bits per frame.
	CHL1-CHL0 = 10 Nine data bits per frame (using XBIT9, RBIT9).
	CHL1-CHL0 = 11 Diagnostic Loopback Mode. Transmitter output is internally looped back to receiver input. Nine-bit framing is used.
ERR:	Error flag. Set upon occurrence of any DOE, FE, or PE or BD error.
RBFL:	Receive buffer full flag. This bit is set when the USART has received a complete byte and has copied it into the RBUF register. The bit is reset automatically when the software reads from the RBUF register.
TBMT:	Transmit buffer empty flag. This bit is set when the USART transfers a byte of data from the TBUF register into the Transmit Shift register for transmission. The bit is reset automatically when the software writes into the TBUF register.

Table 10-2 ENUR, USART Receive Control and Status Register (Address xxBB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOE	FE	PE	BD	RBIT9	ATTN	XMTG	RCVG

DOE:	Data Overrun Error flag; set by a Data Overrun error, cleared by reading the ENUR register
FE:	Framing Error flag; set by a Framing error, cleared by reading the ENUR register
PE:	Parity Error flag; set by a Parity error, cleared by reading the ENUR register
BD:	Set by detection of a line break condition, cleared by reading the ENUR register.
RBIT9:	Receive bit 9; ninth data bit received when USART operates with nine bits per frame
ATTN:	Attention mode enable bit, cleared automatically upon receiving a character with data bit nine set
XMTG:	USART transmitting; set to indicate the transmitter is transmitting, reset at end of last frame
RCVG:	USART receiving error; set to indicate a framing error, reset when RDX goes high

Data Registers

The transmit section contains a pair of linked registers TBUF (Transmit Buffer) and TSFT (Transmit Shift). The TBUF and TSFT registers double-buffer data for transmission, with data shifting out low-order bit first from TSFT to the TDX output. The next byte for transmission is loaded into TBUF by the software while the previous byte is shifting out of TSFT. The next byte from TBUF is automatically loaded into TSFT once the previous byte has shifted out. TSFT can not be read or written by the software.

The receive section contains a pair of linked registers RSFT (Receive Shift) and RBUF (Receive Buffer). The RSFT and RBUF registers double-buffer data being received, with data shifting in low-order bit first from RDX to the RSFT input. While the next byte is shifting into RSFT, the previous byte received is read from RBUF by the software. The next byte from RSFT is automatically loaded into RBUF once the byte has finished shifting into RSFT. Note that RBUF is a read only register. RSFT can not be read or written by the software.

Prescaler and Baud Select Registers

The USART baud clock selection is programmed through the two registers PSR (Prescaler Select) and BAUD (Baud Select). The Prescaler factor is selected by the upper five bits of the PSR register, while the baud rate divisor is determined by the lower three bits of the PSR register in conjunction with the eight bits of the BAUD register. This

Table 10-3 ENUI, USART Interrupt and Clock Source Register (Address xxBC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STP2	BRK	ETDX	SSEL	XRCLK	XTCLK	ERI	ETI

STP2:	Number of Stop bits (0 = 1 Stop bit, 1 = 2 Stop bits)
BRK:	Holds TDX low to generate a line break. Timing of the line break is under software control.
ETDX:	Enable TDX transmit pin, an alternate function of Port L pin L2 (0 = Port L pin, 1 = TDX pin).
SSEL:	Synchronous select bit (0 = asynchronous, 1 = synchronous)
XRCLK:	External Receive Clock; selects clock source for receiver section (0 = internal baud rate generator clock, 1 = external CKX clock)
XTCLK:	External Transmit Clock; selects clock source for transmitter section (0 = internal baud rate generator clock, 1 = external CKX clock)
ERI:	Enable Receive Interrupt (0 = disable, 1 = enable interrupts from the receiver section)
ETI:	Enable Transmit Interrupt (0 = disable, 1 = enable interrupts from the transmitter section)

Table 10-4 BAUD, USART Baud Register (Address xxBD)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BRD7	BRD6	BRD5	BRD4	BRD3	BRD2	BRD1	BRD0

BRD7–BRD0: Baud rate divisor (with PSR register)

Table 10-5 PSR, USART Prescaler Select Register (Address xxBE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSR4	PSR3	PSR2	PSR1	PSR0	BRD10	BRD9	BRD8

PSR4-PSR0: Baud rate prescaler

BRDA-BRD8: Baud rate divisor (with BAUD register)

allows an 11-bit baud rate divisor, ranging from 1 to 2048. Note that the programmed value is equal to the baud rate divisor minus one, which ranges from 0 to 2047. An all zero prescaler value is reserved for selecting NO CLOCK.

Control and Status Registers

The operation of the USART is programmed through the three registers ENU, ENUR and ENUI. All bits in these registers are cleared as a result of a reset with the exception of the TBMT (Transmit Buffer Empty) bit in the ENU register which is initialized high.

Several of the bits in the ENU and ENUR registers are read-only and cannot be written by software. These read-only bits include ERR, RBFL and TBMT in the ENU register and RBIT9, XMTG, RCVG, DOE, FE, PE and BD in the ENUR register. The DOE, FE, PE and BD bits of the ENUR register are cleared automatically when the ENUR register is read. Note that bit manipulation instructions (test bit, set bit, reset bit) or comparison instructions (if equal, if not equal, if greater than) with the ENUR register do not clear the error flags. Moreover, a reset bit instruction on the error flag itself does not clear the error flag, since these error flags are read-only bits.

10.1.3 USART Interface

Port L pins L1, L2, and L3 are used for the USART interface. These three pins are used for CKX (clock), TDX (transmit), and RDX (receive), respectively. RDX is an inherent function associated with the L3 pin and requires no setup following reset, except for ensuring that the associated L3 data register and configuration register bits both remain reset in order to select L3 as an input. The TDX function is assigned to the L2 pin by setting the ETDX bit in the ENUI control register and setting the associated L2 configuration bit in order to configure L2 as an output pin. Once the ETDX bit is set, the associated L2 data register bit may be used as a software flag, since the TDX output from the Transmit Shift register is connected to the L2 pin directly.

The baud rate clock for the USART can be generated on-chip, or can be selected from an external source. The L1 pin is used as the external clock I/O pin (CKX) if either the XTCLK or XRCLK bits in the ENUI control register are set. If neither of these control bits is set, then L1 serves as a normal I/O pin. The CKX pin can be used as either an input or output, as determined by the associated L1 configuration register bit. As an input, CKX represents an external clock input, which may be selected to drive the USART transmitter and/or receiver. As an output, CKX represents the internal baud rate generator clock output.

10.1.4 Asynchronous Mode

The asynchronous mode is selected by resetting the SSEL bit to zero in the ENUI register. The input frequency to the USART is 16 times the baud rate. The Transmit Shift and TBUF registers double-buffer data for transmission. While the Transmit Shift register is shifting out the current character on the TDX pin, the TBUF register may be loaded by software with the next byte to be transmitted. When the Transmit Shift register finishes transmitting the current character, the contents of TBUF are transferred to the Transmit Shift register and the Transmit Buffer Empty flag (TBMT in the ENU register) is set. The TBMT flag is automatically reset by the USART when software loads a new character into the TBUF register. There is also the XMTG bit, which is set to indicate that the USART is transmitting. This bit is reset at the end of the last frame (end of last Stop bit). Figure 10-2 shows the transmitter timing diagram.

The Receive Shift and RBUF registers double-buffer data being received. The USART receiver continually monitors the signal on the RDX pin for a low level to detect the beginning of a Start bit. Upon sensing this low level, it waits for half a bit time and samples again. If the RDX pin is still low, the receiver considers this to be a valid Start bit, and the remaining bits in the character frame are each sampled 3 times around the center of the bit time. Serial data received on the RDX pin is shifted into the Receive Shift

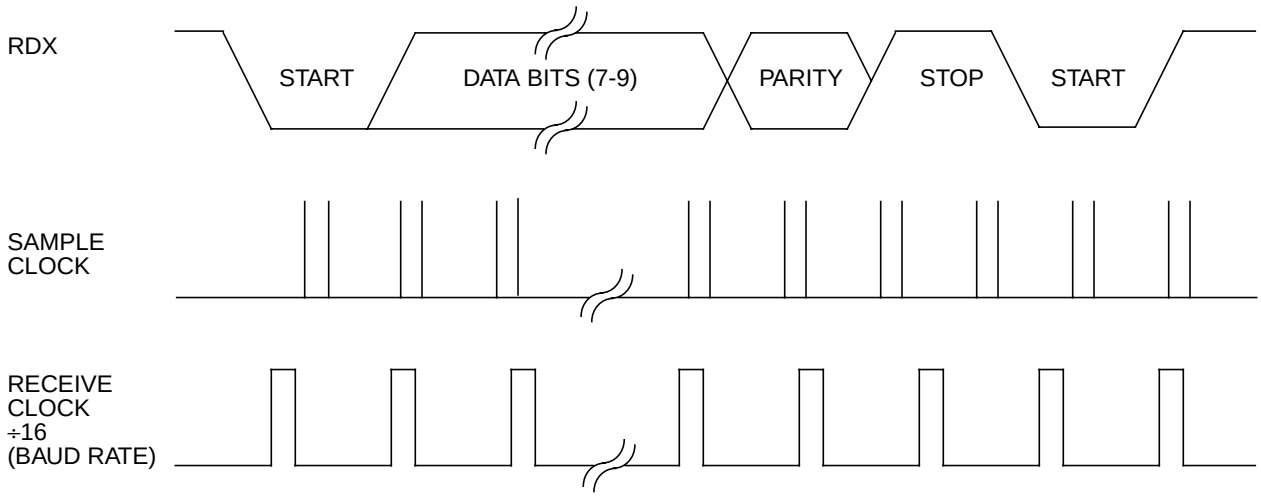


Figure 10-2 USART Receiver Timing, Asynchronous Mode

register. Upon receiving the complete character, the contents of the Receive Shift register are copied into the RBUF register and the Receive Buffer Full flag (RBFL) is set. RBFL is automatically reset when software reads the character from the RBUF register. RBUF is a read-only register. There is also the RCVG bit, which is set high whenever a framing error occurs. The RCVG bit is reset whenever RDX goes high. TBMT, XMTG, RBFL and RCVG are read-only bits. Figures 10-3 and 10-4 show the timing diagrams for the receiver.

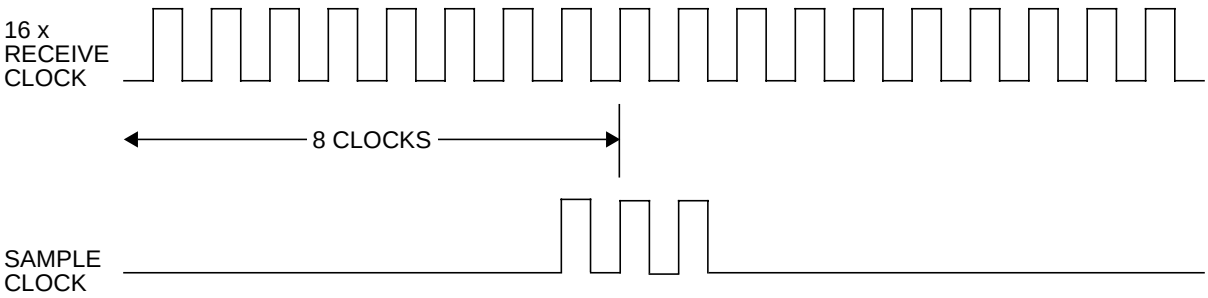


Figure 10-3 USART Receiver Bit Sampling, Asynchronous Mode

The clock source for the transmitter and/or receiver can be selected to come from an external source (at the CKX pin) or from the internal baud rate generator. If the internal baud rate generator is used, the internal clock can be output to the CKX pin.

10.1.5 Synchronous Mode

The synchronous mode is selected by setting the SSEL, XTCLK, and XRCLK bits to all ones in the ENUI register. In this mode, data is transmitted on the rising edge and received on the falling edge of the synchronous clock on the CKX pin.

The input frequency to the USART is the same as the baud rate on the CKX pin. If data transmit and receive are selected with the CKX pin as the clock output, the microcontroller generates the synchronous clock output at the CKX pin. The internal

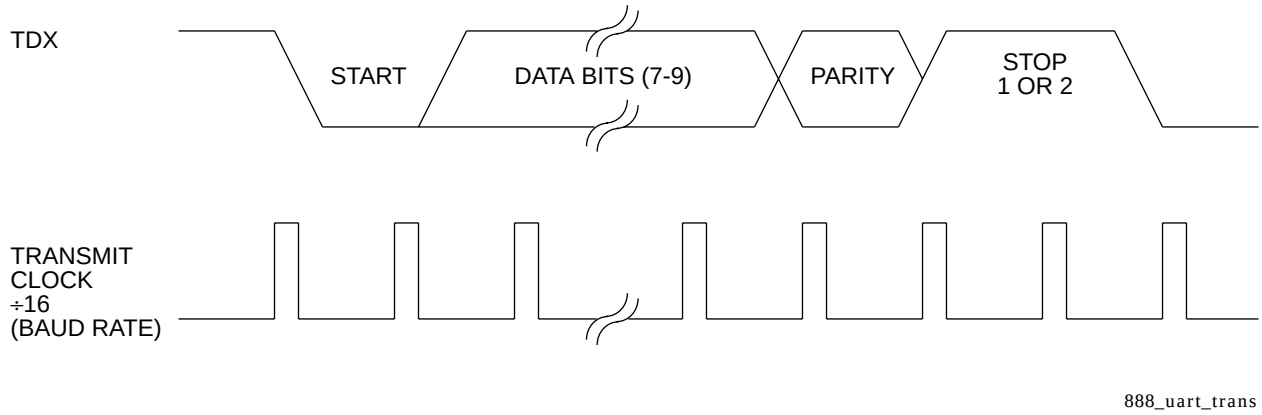


Figure 10-4 USART Transmitter Timing, Asynchronous Mode

baud rate generator is used to produce the synchronous clock. Data transmit and receive are performed synchronously with this clock through the TDX and RDX pins. Note that if CKX is selected as an output pin, then the selected clock from the internal Baud Rate Generator is divided by 2 before being output to the CKX pin. This results in a 50 percent duty cycle on the CKX clock. Figure 10-5 shows the timing diagram for the synchronous mode.

When an external clock input is selected at the CKX pin, data transmit and receive are performed synchronously with this clock through the TDX and RDX pins.

10.1.6 Framing Formats

The USART supports several framing formats as shown in Figure 10-6. The format is selected by writing certain control bits in the ENU and ENUI registers. A framing format consists of a Start bit followed by seven, eight, or nine data bits (excluding parity), followed by an optional parity bit, followed by 1 or 2 Stop bits. In applications using parity, the parity bit is generated and verified by hardware. The optional parity bit may be selected as either odd or even parity, a mark, or a space. No parity is possible with the nine-bit format.

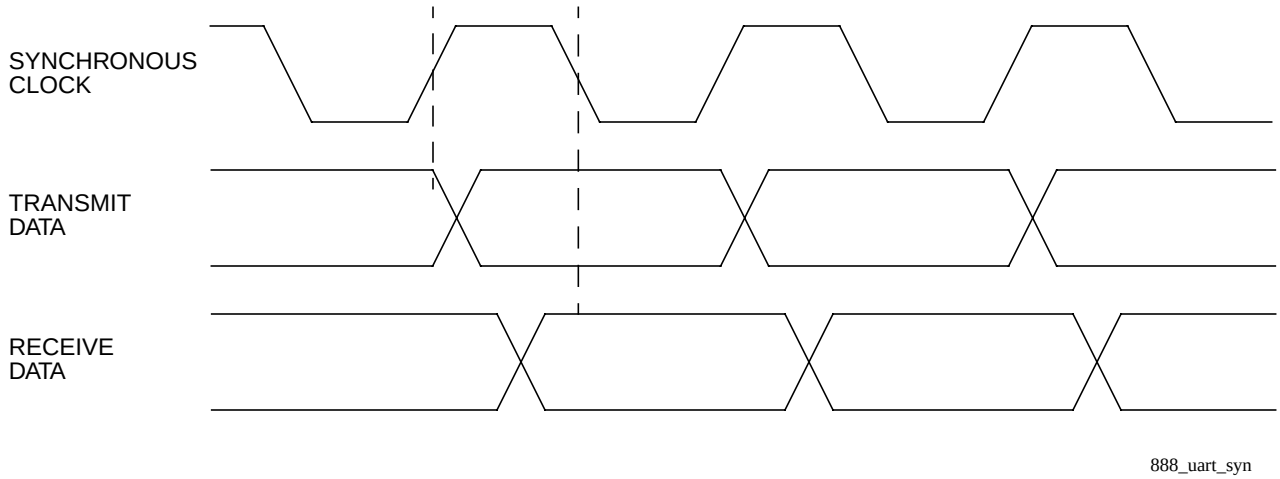
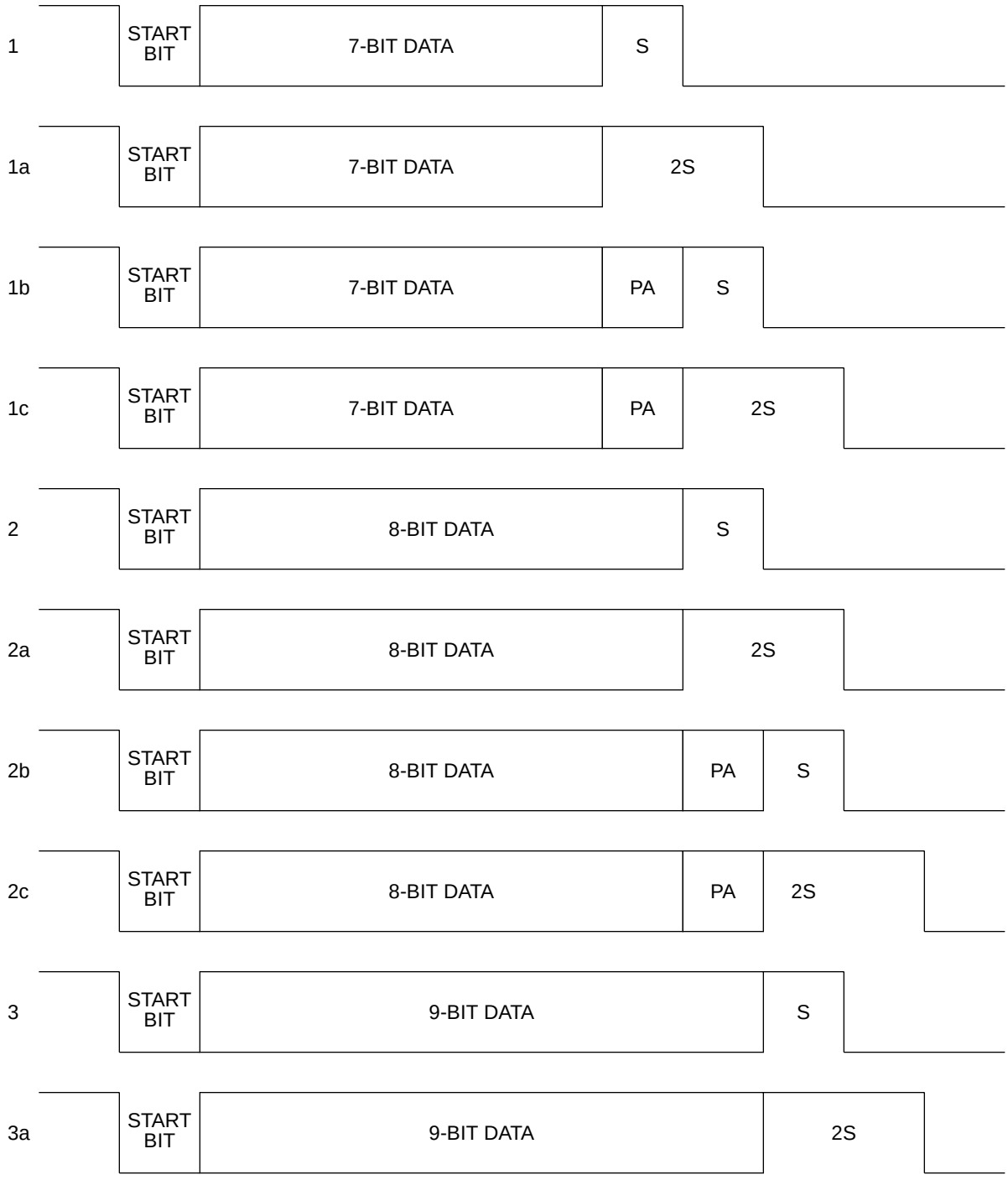


Figure 10-5 USART Synchronous Mode Timing



888_uart_frame

Figure 10-6 USART Framing Formats

The number of data bits (7, 8, or 9) is selected with the CHL0 and CHL1 bits in the ENU register. The first data transmission format shown in Figure 10-6 (1, 1a, 1b, 1c) consists of a Start bit, seven data bits, an optional parity bit, and 1 or 2 Stop bits. This format is selected with CHL1 = 0, CHL0 = 1.

The second data transmission format (2, 2a, 2b, 2c) consists of a Start bit, eight data bits, an optional parity bit, and 1 or 2 Stop bits. This format is selected with CHL1 = 0, CHL0 = 0. The optional parity bit with these first two formats is generated and verified by hardware. The parity bit is enabled or disabled by the PEN (Parity Enable) bit in the ENU register. If parity is enabled (PEN = 1), the parity selection is made with the PSEL0 and PSEL1 bits of the ENU register. The parity selections include odd parity (PSEL1 = 0, PSEL0 = 0), even parity (PSEL1 = 0, PSEL0 = 1), mark (PSEL1 = 1, PSEL0 = 0), and space (PSEL1 = 1, PSEL0 = 1). The mark consists of a 1 bit, while the space consists of a 0 bit.

The third data transmission format (3, 3a) consists of a Start bit, nine data bits, and 1 or 2 Stop bits. This format is selected with CHL1 = 1, CHL0 = 0. Parity is not generated or verified with this format. The USART Attention mode feature is supported by this 9-data-bit format. When operating in this format, all eight bits of TBUF and RBUF are used for data. The ninth data bit is transmitted and received using the two bits XBIT9 and RBIT9, located in the ENU and ENUR registers, respectively. Note that RBIT9 is a read-only bit.

Note that the XBIT9/PSEL0 bit located in the ENU register serves two mutually exclusive functions. This bit programs the ninth bit for transmission when the USART is operating with the nine-data-bit framing format selected. There is no parity selection with this framing format. For the other framing formats, XBIT9 is not needed, so this same bit is defined instead as the PSEL0 bit, which is used in conjunction with the PSEL1 bit to select the type of parity (odd, even, mark, space) if parity generation is enabled.

The framing formats for the receiver only differ from those for the transmitter in the number of Stop bits required. The receiver requires only one Stop bit in a frame, regardless of the setting of the Stop-bit selection bits in the ENUI control register. Note that an implicit assumption is made for full-duplex USART operation that the framing formats are the same for the transmitter and receiver.

10.1.7 Reset Initialization

All bits in the USART Control and Status registers ENU, ENUR, and ENUI are cleared as a result of RESET, with the exception of the TBMT (Transmit Buffer Empty) bit in the ENU register, which is initialized high.

The PSR register is cleared as a result of RESET, while the Transmit Shift register is initialized high to all ones. The reason for initializing the Transmit Shift register high is to provide a high-to-low transition on the TDX output to signify a start bit once the USART has started. Note that the Transmit Shift and Receive Shift registers are non-addressable registers, while the RBUF register is a read-only register.

The TBUF, RBUF, Receive Shift, and BAUD registers are not initialized with RESET.

10.1.8 HALT/IDLE Mode Reinitialization

The USART is initialized as a result of reset and is initialized again whenever the microcontroller enters the HALT or IDLE mode. Note that a HALT or IDLE mode reinitialization is a subset of the RESET initialization, in that much of the USART control selection (baud rate, framing format, etc.) is left unchanged following HALT or IDLE.

The HALT/IDLE mode reinitialization is the same as the reset initialization described above, with the exception of the three USART control and status registers ENU, ENUR, and ENUI. All of the read/write bits in these three registers are left unchanged as a result of HALT/IDLE mode reinitialization. The read-only bits ERR, RBFL, DOE, FE, PE, BD, RBIT9, XMTG, and RCVG are cleared as a result of the HALT/IDLE mode reinitialization, while the TBMT (Transmit Buffer Empty) read-only bit is initialized high.

In summary, the HALT/IDLE mode reinitialization is identical to the reset initialization, with the exception of the read/write bits in the ENU, ENUR, and ENUI registers. These read/write bits are cleared with reset initialization, but are left unchanged with HALT/ IDLE mode reinitialization.

The microcontroller may exit from the HALT/IDLE mode when the Start bit of a character is detected at the RDX input (pin L3 of Port L). This feature is activated by using the Multi-Input Wakeup selected for pin L3. Note that the COP8 microcontroller also exits the IDLE mode whenever the thirteenth bit of the IDLE counter (T0) toggles.

The application program must set up the RDX Start bit wakeup on pin L3 before entering the HALT/IDLE mode. This section of the program consists of the following steps:

```
RBIT 3,WKEN
SBIT 3,WKEDG
RBIT 3,WKPND
SBIT 3,WKEN
```

The Wakeup trigger condition for an RDX start bit is programmed as a high-to-low transition by setting bit 3 of the WKEDG register. Bit 3 of the WKPND register is cleared to eliminate any previous RDX transition still pending, followed by setting bit 3 of the WKEN register to enable the Wakeup.

If a crystal or resonator closed loop oscillator is used with the microcontroller, the IDLE timer (T0) is used to generate a fixed delay following the HALT mode. Crystals and resonators require a certain amount of startup time to reach full amplitude and frequency stability. The fixed delay from the IDLE timer following HALT is necessary to ensure that the oscillator has indeed stabilized before the microcontroller is allowed to execute program code. The program must consider this fixed delay when a USART data transfer is expected immediately following the HALT mode.

10.1.9 Baud Clock Generation

The clock inputs to the transmitter and receiver sections of the USART can be individually selected to come from either an external source on the CKX pin (pin L1 of the Port L) or a source selected by dividing the CPU clock (MCLK) by the Baud Rate

Generator. The Baud Rate Generator consists of the PSR (prescaler) and BAUD registers along with the associated selection circuitry. The clock input selection for the USART Transmitter and Receiver sections is shown in Figure 10-7. The selection of CKX versus BRG (Baud Rate Generator) is programmed using the XTCLK and XRCLK bits in the ENUI control register. Tables 10-6 and 10-7 show the USART clock source selection relative to the XTCLK and XRCLK bits for the Asynchronous and Synchronous modes, respectively. The PSR and BAUD registers are memory mapped at data memory address locations xxBE and xxBD Hex, respectively.

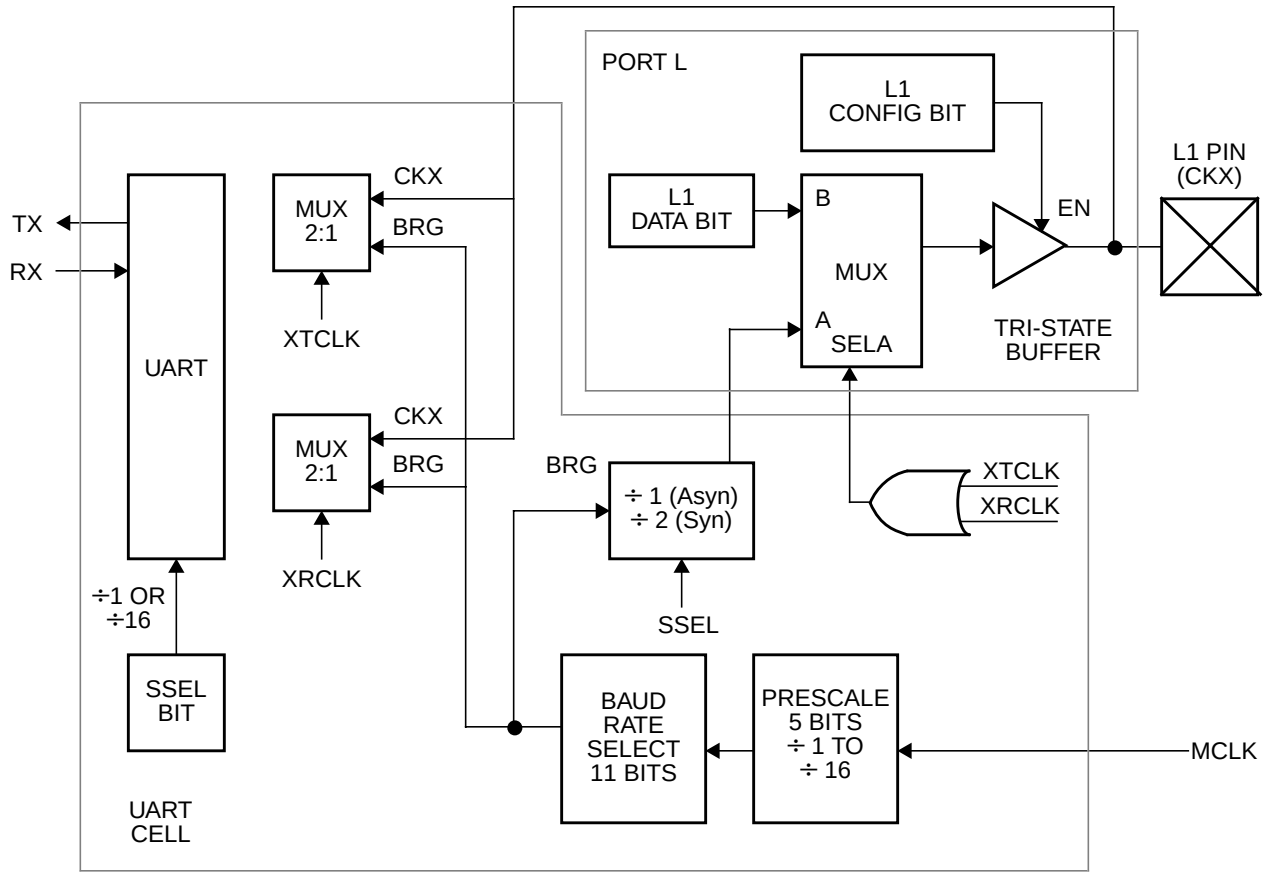


Figure 10-7 USART Baud Clock Generation Block Diagram

Table 10-6 USART Clock Sources (Asynchronous Mode)

XTCLK	XRCLK	TRANSMIT CLOCK	RECEIVE CLOCK	L1 PIN
0	0	BRG	BRG	Normal I/O
0	1	BRG	CKX	CKX I/O
1	0	CKX	BRG	CKX I/O
1	1	CKX	CKX	CKX I/O

Table 10-7 USART Clock Sources (Synchronous Mode)

XTCLK	XRCLK	TRANSMIT CLOCK	RECEIVE CLOCK	L1 PIN
1	1	CKX	CKX	CKX I/O

CKX I/O in Tables 10-6 and 10-7 means that the L1 pin can be configured as either an input or an output. With L1 configured as an input, an external clock can be used for the transmitter and/or receiver. With L1 configured as an output, the BRG clock is routed to the L1 pin, from where the BRG clock can be selected as the CKX clock for the transmitter and/or receiver. This is the data path used in the Synchronous mode to route the BRG clock (divided by 2) to the USART.

The selected internal Baud Rate Generator clock (BRG) can be output to the CKX pin in both the Asynchronous and Synchronous modes. In the Synchronous mode, the selected internal BRG clock is divided by two before it is output to the CKX pin. The division by two is necessary to obtain an output CKX clock with a 50 percent duty cycle. Note also that integer prescaler values (no fractional values such as 2.5) are necessary in the Synchronous mode to produce a 50 percent duty cycle output CKX clock. The CKX pin can also be selected as an external clock input for both the Synchronous and Asynchronous modes. In the Synchronous mode, both the XTCLK and XRCLK clock select bits should be set high to a “1” as shown in Table 10-7.

As an example of USART clock selection, consider the Asynchronous mode with XTCLK = 0 and XRCLK = 1 as selected from Table 10-6. The transmitter is clocked from the internal Baud Rate Generator (BRG), while the receiver is clocked from the CKX pin, which may be sourced from either an external clock input or from the BRG. Pin L1 is dedicated for the CKX clock whenever the CKX clock is selected. The pin L1 configuration bit determines whether a selected CKX clock is an input from an external source or an output from the BRG. This configuration bit is set to select the CKX pin L1 as an output and is reset to select CKX as an external input. Note that both the transmitter and receiver are clocked from BRG when the CKX pin is selected as an output.

In the Asynchronous mode, the USART divides the clock by 16, regardless of whether the clock is selected from BRG or CKX. In the Synchronous mode, the USART divides the CKX clock by 1. Note, however, that if the CKX clock is selected as an output clock in the Synchronous mode, then the CKX clock will result from the selected BRG clock divided by 2 as shown in Figure 10-7. Consequently, a selected BRG clock in the Synchronous mode is divided by 2 before it reaches the USART by way of CKX.

Internally, the basic baud clock is created from the MCLK CPU clock frequency through a two-section divider chain. This divider chain consists of a five-bit prescaler ranging from 1 to 16 in increments of 0.5, coupled with an 11-bit binary counter. The division factors are specified through two registers, PSR (Prescaler Select) register and BAUD (Baud Select) register, as shown in Figure 10-8. Note that the 11-bit Baud Rate Divisor spills over from the BAUD register into the PSR Prescaler Select register.

The Prescaler factor is selected by the upper five bits of PSR, while the baud rate divisor is determined by the lower three bits of PSR in conjunction with the eight bits of BAUD.

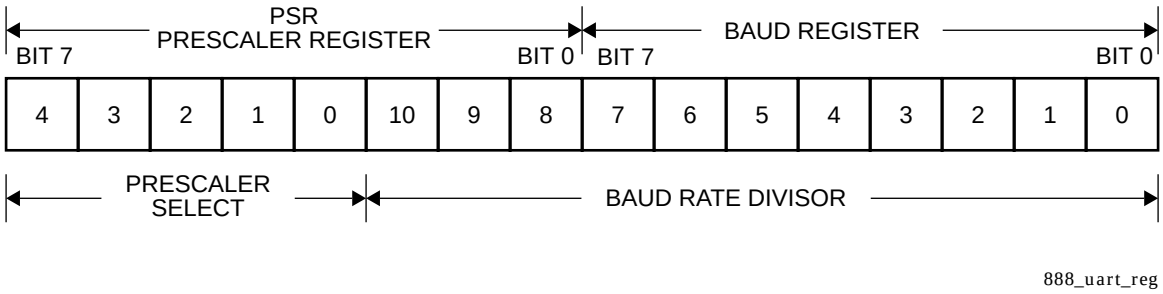


Figure 10-8 USART Baud Clock Divisor Registers

This allows an eleven-bit baud rate divisor, ranging from 1 to 2048. Note that the programmed value is equal to the baud rate divisor minus one, which ranges from 0 to 2047. An all-zero prescaler value is reserved for selecting NO CLOCK. The NO CLOCK condition is the USART Power Down mode where the USART clock is turned off to save power. The application program must also turn off the USART clock when selecting a different baud rate.

Note that many features of the USART are disabled when the NO CLOCK option is selected. Consequently, even when an external CKX clock is being used for both the USART transmitter and receiver, the NO CLOCK option should not be used when any of the parity, Synchronous mode, 7 data bit, or Loopback Diagnostic mode options are selected.

The correspondence between the 5-bit Prescaler Select and the Prescaler factors is shown in Table 10-8. There are many different ways to calculate the two division factors (Prescaler and Baud Rate Divisor). One effective method is to achieve a 1.8432-MHz frequency coming out of the Prescaler. One possible method of producing this 1.8432-MHz frequency is to select a 9.216 MHz CKI oscillator clock rate (18.432 MHz MCLK) coupled with a prescaler selection of a divide-by-ten factor. The 1.8432 MHz prescaler output is then used to drive the software programmable baud rate counter to produce an X16 clock for the following baud rates: 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200, and 38400. These baud rates, together with their associated Baud Rate Division factors, are shown in Table 10-9. Note that Table 10-9 actually contains the baud rate divisor minus one (N-1) associated with each baud rate, where N is the baud rate divisor. The value (N-1) from Table 10-9 is the value to be programmed in the 11-bit baud rate counter (lower 3 bits of PSR and 8 bits of BAUD). The X16 clock is then divided by 16 in the USART (Asynchronous mode only) to provide the clocks for the serial shift registers of the USART transmitter and receiver.

Table 10-8 USART Prescaler Factors (Sheet 1 of 2)

Prescaler Select	Prescaler Factor
00000	NO CLOCK
00001	1
00010	1.5
00011	2

Table 10-8 USART Prescaler Factors (Sheet 2 of 2)

Prescaler Select	Prescaler Factor
00100	2.5
00101	3
00110	3.5
00111	4
01000	4.5
01001	5
01010	5.5
01011	6
01100	6.5
01101	7
01110	7.5
01111	8
10000	8.5
10001	9
10010	9.5
10011	10
10100	10.5
10101	11
10110	11.5
10111	12
11000	12.5
11001	13
11010	13.5
11011	14
11100	14.5
11101	15
11110	15.5
11111	16

Table 10-9 USART Baud Rate Divisors, 1.8432 MHz Prescaler Output

Baud Rate	Baud Rate Divisor Minus One (N-1)
110 (110.03)	1046
134.5 (134.58)	855
150	767
300	383
600	191
1200	95
1800	63
2400	47
3600	31
4800	23
7200	15
9600	11
19200	5
38400	2

The entries in Table 10-9 assume a prescaler output of 1.8432 MHz.

Example 1:

This example further clarifies the usage of Tables 10-8 and 10-9 in selecting a desired baud rate. Consider a CKI clock frequency of 4.608 MHz (MCLK 0f 9.216 MHz) with the USART Asynchronous mode selected, and suppose a baud rate of 19200 is desired. If a prescaler output frequency of 1.8432 MHz is selected in order to use Table 10-9, then 9.216 divided by 1.8432 yields a prescaler factor of 5 which is an available entry in Table 10-8. For a baud rate of 19200, the corresponding entry in Table 10-9 is 5. This means a value of 5 should be programmed as the value in the 11-bit counter (lower 3 bits of PSR and 8 bits of BAUD) to select a baud rate divisor of 6. Consequently, N = 6 where N is the baud rate divisor corresponding to 19200 baud, and N-1 = 5 is the value from Table 10-9 to be programmed in the 11-bit counter. A value of 01001 would be programmed as the prescaler value in the upper 5 bits of PSR to match the prescaler factor of 5 from Table 10-8. The baud rate of 19200 can be calculated as the prescaler output frequency (1.8432 MHz) divided by 16 times the baud rate divisor of 6. The divide by 16 results from the USART input frequency being 16 times the baud rate in the Asynchronous mode. Note that in the Synchronous mode, this division factor of 16 is replaced by a division factor of 2.

The actual baud rate (BR) may be calculated as follows:

$$BR = CKIFx2/(16 \times P \times N)$$

where CKIF is the CKI input oscillator frequency, P is the prescaler division factor, and N is the baud rate divisor. This calculation is for Asynchronous mode USART operation, as indicated by the division factor of 16. Using Example 1 produces a calculation as follows:

$$BR = CKIFx2/(16 \times P \times N)$$

$$= (4.608 \times 10^6)x2/(16 \times 5 \times 6)$$

$$= 19200$$

Note that in the Synchronous mode this division factor of 16 is replaced by a division factor of 2. Consequently, baud rate may be calculated for the Synchronous mode as follows:

$$BR = CKIFx2/CKXF$$

$$= CKIFx2/(2 \times P \times N) = CKIF/(P \times N)$$

where CKXF is the CKX output frequency. Note that the division factor of 2 results from the BRG frequency being divided by 2 to produce a 50 percent duty cycle for the output CKX frequency.

Example 2:

As a further example, consider a CKI crystal frequency of 5 MHz with the USART in Asynchronous mode, where the desired baud rate is 9600. Using the asynchronous BR equation yields:

$$P \times N = CKIFx2/(16 \times BR)$$

$$= (5 \times 10^6)x2/(16 \times 9600)$$

$$= 65.104$$

The value 65.104 is now divided by each prescaler factor in Table 10-8 to obtain a value closest to an integer. A prescaler factor of 13 yields a result very close to an integer as follows:

$$N = 65.104/P$$

$$= 65.104/13$$

$$= 5.008$$

Consequently, N = 5 approximately, and a value of N-1 = 4 should be programmed in the 11-bit counter for the baud rate divisor value.

The calculated values of P and N are now used to calculate the actual baud rate that is produced:

$$BR = CKIFx2/(16 \times P \times N)$$

$= (5 \times 10^6)/(16 \times 13 \times 5)$

$= 9615.385$

The percentage error of the Baud Rate produced is:

% ERROR = (9615.385 - 9600)/9600

= 0.0016

= 0.16%

10.1.10 USART Interrupts

The USART is capable of generating interrupts as result of Receive Buffer Full and Transmit Buffer Empty conditions. Both interrupts have individual interrupt vectors. Two bytes of program memory space are reserved for each interrupt vector. The two vectors are located at addresses xxEE to xxF1 Hex in the program memory space, with the high-order address byte depending on the location of the VIS instruction. The interrupts can be individually enabled or disabled using the Enable Transmit Interrupt (ETI) and Enable Receive Interrupt (ERI) bits in the ENUI register.

The interrupt from the Transmitter is set pending, and remains pending, as long as both the TBMT and ETI bits are set. To remove this interrupt, software must either clear the ETI bit or write to the TBUF register (thus clearing the TBMT bit).

The interrupt from the receiver is set pending, and remains pending, as long as both the RBFL and ERI bits are set. To remove this interrupt, software must either clear the ERI bit or read from the RBUF register (thus clearing the RBFL bit).

10.1.11 USART Error Flags

Error conditions detected by the USART receiver are brought to the software's attention by setting three error flags in the ENUR register. The flag bit FE (Framing Error) is set when the receiver fails to see a valid Stop bit at the end of the frame. The flag bit DOE (Data Overrun Error) is set if a new character is received in RBUF while it is still full (before the software has read the previous character from RBUF). The selected parity is verified by hardware, and the flag bit PE (Parity Error) is set if a parity error is detected. If a line break condition is detected, the BD (Break Detect) bit is set. A global flag ERR (in the ENU register) is set if any of the four error conditions (FE, DOE, PE or BD) occur.

If another character comes into the Receive Buffer (RBUF) before the software has read the previous character from RBUF, any errors associated with the new character will augment any previous errors already present. The receipt of a new character in RBUF can cause any of the four error bits to go from a 0 to a 1, but not from a 1 to a 0. In other words, the receipt of a new character can set new errors but cannot reset previous errors.

The five error flags (ERR, FE, DOE, BD and PE) are all read-only bits and cannot be written by the software. These five error bits are cleared with RESET. They are also cleared as a result of the HALT/IDLE mode. The four error flags FE, DOE, PE and BD are reset whenever they are read by software. Consequently, there are two considerations in accessing the ENUR register where the FE, DOE, PE and BD flags are located:

1. When reading the ENUR register, always check for receiver errors in the copy of the ENUR register read into the accumulator (or save the copy for error checking later).

2. Never perform a read type operation (other than LD or X) involving the accumulator on the ENUR register (examples: ADD, ADC, SUBC, AND, OR, XOR) since this clears the error flags without saving them for later test. Bit manipulation instructions (test bit, set bit, reset bit) or comparison instructions (including those on the accumulator) with the ENUR register do not clear the error flags. Moreover, a reset bit instruction on the error flag itself does not clear the error flag, since these error flags are read-only bits.

10.1.12 Diagnostic Testing

A loopback feature is provided for diagnostic testing of the USART. The CHL0 and CHL1 bits in the ENU register are used to select the loopback mode. When these bits are both set to one, the following internal connections are made:

1. The receiver input pin RDX is internally connected to the transmitter output pin TDX.

2. The output of the Transmitter Shift register is “looped back” into the Receiver Shift register input.

In this diagnostic mode, data that is transmitted is immediately received. This diagnostic feature allows the microcontroller to verify the transmit and receive data paths of the USART.

Note that the framing format for the diagnostic mode is the 9-bit format, consisting of one Start bit, nine data bits, and 7/8, 1, 1-7/8 or 2 Stop bits. Parity is not generated or verified in the Diagnostic mode.

A block diagram illustrating the Diagnostic mode “loopback” connection is shown in Figure 10-9.

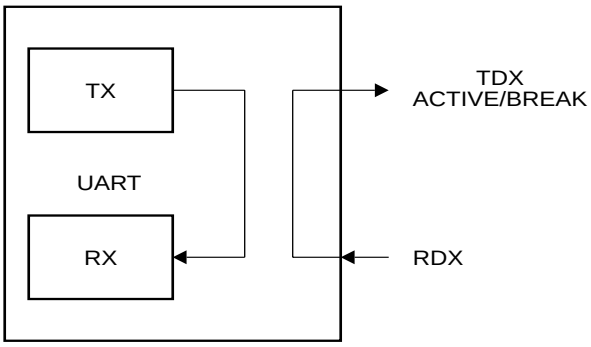


Figure 10-9 USART Diagnostic Mode Loopback Connection

10.1.13 Attention Mode

The USART receiver section supports an alternate mode of operation, called the Attention mode. This mode of operation is selected by the ATTN bit in the ENUR register.

The framing format for transmission in the Attention mode is the 9-bit format, consisting of one Start bit, nine data bits, and 1 or 2 Stop bits. Parity is not generated or verified in the Attention mode.

The Attention mode of operation is intended for use in networking the microcontroller with other processors. Typically, in such environments, the messages consist of device addresses and data, with the device address indicating which of several destinations should receive the data. The device addresses are flagged in the messages by having the ninth bit of the data field set to a one. If the ninth bit of the data field is a zero, then the data field contains a data byte.

While in Attention mode, the USART monitors the communication flow but ignores all characters until an address character (a character whose ninth bit is set to 1) is received. Upon receiving an address character, the USART signals that the character is ready by setting the RBFL flag. This interrupts the microcontroller if the USART receiver interrupt is enabled. The ATTN bit is cleared automatically at this point, resulting in data characters as well as address characters being received. The software examines the contents of the RBUF register and responds by deciding either to accept the subsequent data stream (by leaving the ATTN bit reset) or to wait until the next address character is received (by setting the ATTN bit again).

The operation of the USART transmitter is not affected by selection of the Attention mode. The value of the ninth data bit to be transmitted is programmed by setting the XBIT9 bit appropriately. The value of the ninth data bit received is obtained by reading the RBIT9 bit. This ninth data bit in the Attention mode is used to distinguish between address and data characters.

Since RBIT9 is located in the ENUR register where the three error flags (FE, DOE, PE) reside, care should be taken not to reset the error flags while reading RBIT9. To avoid clearing the error flags, use a test bit instruction to read RBIT9.

10.1.14 Break Generation and Detection

Two methods may be used to emulate a line Break. Reset the ETDX bit in the ENUI register to change the TDX pin into a general-purpose Port L pin, and output a logic 0 on the same pin using the Port L data and configuration registers, bit 2, or set the BRK bit in the ENUI Register.

If a framing error occurs and the contents of the RSFT is 000, the Break Detect flag is set in the ENUR register, but the FE bit is not set. Otherwise, the USART will behave exactly as for any other framing error. If a break occurs in the middle of an otherwise valid frame (i.e., a frame containing ones), both FE and BD will be set to one. If a break occurs while the line is marking time, BD will be set to one, but FE will not.

Chapter 11

IN SYSTEM PROGRAMMING AND VIRTUAL E^2

11.1 OVERVIEW

The COP8 Flash family of products have been designed with the capability to update the program memory without the need to remove the device from the application. This has been done, in part, through the incorporation of a 1Kbyte Boot ROM which resides in parallel with the user program memory.

Available techniques for updating the program memory include MICROWIRE/PLUS communication through a program contained in the Boot ROM and user custom communication, which utilizes subroutines in the Boot ROM that perform the actual Flash memory access.

Subroutines in the Boot ROM also allow the user to utilize otherwise unused portions of the Flash memory as non-volatile backup of program variables. This provides for a custom sized EEPROM capability without the need for the actual existence of EEPROM in the device.

11.2 FLASH MEMORY DURABILITY CONSIDERATIONS

The endurance of the Flash Memory (number of possible Erase/Write cycles) is a function of the erase time and the lowest temperature at which the erasure occurs. If the device is to be used at low temperature, additional erase operations can be used to extend the erase time. The user can determine how many times to erase a page based on what endurance is desired for the application (e.g. four page erase cycles, each time a page erase is done, may be required to achieve the typical 100k Erase/Write cycles in an application which may be operating down to 0°C). Also, the customer can verify that the entire page is erased, with software, and request additional erase operations if desired.

Table 11-1 Typical Flash Memory Endurance

Low end of operating temp range					
Erase Time	**-40°C**	**-20°C**	**0°C**	**25°C**	**>25°C**
1ms	60K	60k	60k	100k	100k
2ms	60k	60k	60k	100k	100k
3ms	60K	60k	60k	100k	100k
4ms	60K	60k	100k	100k	100k
5ms	70K	70K	100k	100k	100k
6ms	80K	80K	100k	100k	100k
7ms	90K	90K	100k	100k	100k
8ms	100k	100k	100k	100k	100k

11.3 HARDWARE

11.3.1 Block Diagram

Figure 11-1 shows a block diagram of the hardware used for the In-System Programming (ISP).

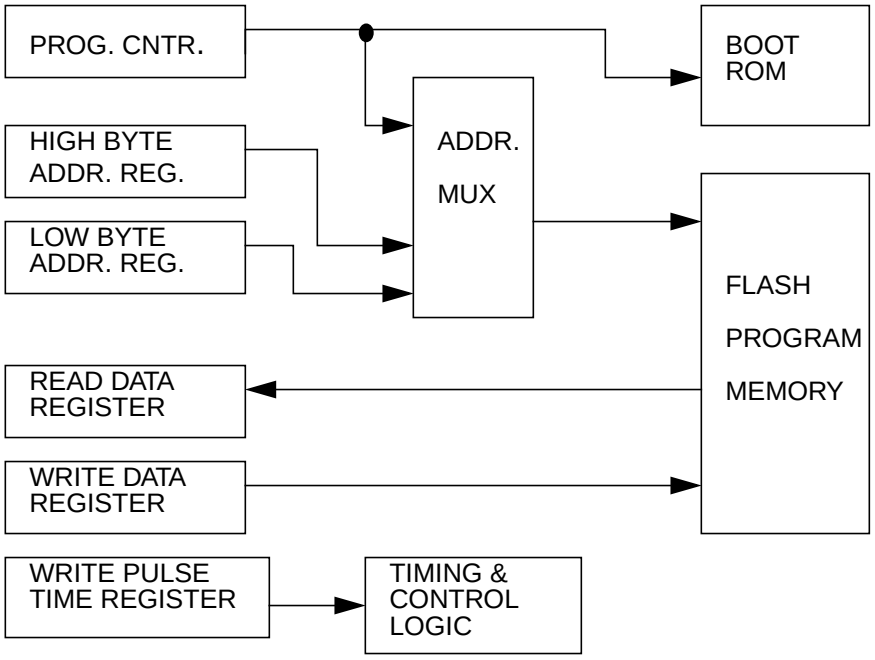


Figure 11-1 Block Diagram of ISP

11.3.2 Register Set

There are six registers required to support ISP: Address Register Hi byte (ISPADHI), Address Register Low byte (ISPADLO), Read Data Register (ISPRD), Write Data Register (ISPWR), Write Timing Register (PGMTIM), and the Control Register (ISPCNTRL). The ISPCNTRL Register is not available to the user.

ISP Address Registers

The address registers (ISPADHI & ISPADLO) are used to specify the address of the byte of data being written or read. For page erase operations, the address of the beginning of the page should be loaded. For mass erase operations, 0000 must be placed into the address registers. When reading the Option register, FFFF (hex) should be placed into the address registers. Registers ISPADHI and ISPADLO are cleared to 00 on Reset. These registers can be loaded from either flash program memory or Boot ROM and must be maintained for the entire duration of the operation.

NOTE: The actual memory address of the Option Register is 7FFF(hex), however the MICROWIRE/PLUS ISP routines require the address FFFF(hex) to be used to read the Option Register when the Flash Memory is secured.

Table 11-2 High Byte of ISP Address

ISPADHI							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Addr15	Addr14	Addr13	Addr12	Addr11	Addr10	Addr9	Addr8

Table 11-3 Low Byte of ISP Address

ISPADLO							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Addr7	Addr6	Addr5	Addr4	Addr3	Addr2	Addr1	Addr0

ISP Read Data Register

The Read Data Register (ISPRD) contains the value read back from a read operation. This register can be accessed from either flash program memory or Boot ROM. This register is undefined on Reset.

Table 11-4 ISP Read Data Register

ISPRD							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

ISP Write Data Register

The Write Data Register (ISPWR) contains the data to be written into the specified address. This register is undetermined on Reset. This register can be accessed from either flash program memory or Boot ROM. The Write Data register must be maintained for the entire duration of the operation.

Table 11-5 ISP Write Data Register

ISPWR							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

ISP Write Timing Register

The Write Timing Register (PGMTIM) is used to control the width of the timing pulses for write and erase operations. The value to be written into this register is dependent on the frequency of CKI and is shown in Table 11-6. This register must be written before any write or erase operation can take place. It only needs to be loaded once, for each value of CKI frequency. This register can be loaded from either flash program memory or Boot ROM and must be maintained for the entire duration of the operation. The MICROWIRE/PLUS ISP routine that is resident in the boot ROM requires that this Register be defined prior to any access to the Flash memory. Refer to Section 11.4, for more information on available ISP commands. On Reset, the PGMTIM register is loaded with the value that corresponds to 10Mhz frequency for CKI.

References are made through this chapter, e.g. Table 11-22, to a delay time which is called PGMTIM. This time is in the range of 30 - 40 µsec and is calculated in the following manner.

$Time_{PGMTIM} = [(PGMTIM(5:0)+1)*0.5/F_{CKI}]*10^{PGMTIM(6)}$

where:

$Time_{PGMTIM}$ = the actual PGMTIM delay

PGMTIM(5:0) = the value in the PGMTIM register bits 5 through 0

F_{CKI} = = the current oscillator frequency

PGMTIM(6) = the value in bit 6 of the PGMTIM register.

Table 11-6 PGMTIM Register Format

PGMTIM								
Register Bit								CKI Frequency Range
7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	25Khz - 33.3Khz
0	0	0	0	0	0	1	0	37.5Khz - 50Khz
0	0	0	0	0	0	1	1	50Khz - 66.67Khz
0	0	0	0	0	1	0	0	62.5Khz - 83.3Khz
0	0	0	0	0	1	0	1	75Khz - 100Khz
0	0	0	0	0	1	1	1	100Khz - 133Khz
0	0	0	0	1	0	0	0	112.5Khz - 150Khz
0	0	0	0	1	0	1	1	150Khz - 200Khz
0	0	0	0	1	1	1	1	200Khz - 266.67Khz
0	0	0	1	0	0	0	1	225Khz - 300Khz
0	0	0	1	0	1	1	1	300Khz - 400Khz

Table 11-6 PGMTIM Register Format

PGMTIM								
Register Bit								CKI Frequency Range
0	0	0	1	1	1	0	1	375Khz - 500Khz
0	0	1	0	0	1	1	1	500Khz - 666.67Khz
0	0	1	0	1	1	1	1	600Khz - 800Khz
0	0	1	1	1	1	1	1	800Khz - 1.067Mhz
0	1	0	0	0	1	1	1	1Mhz - 1.33Mhz
0	1	0	0	1	0	0	0	1.125Mhz - 1.5Mhz
0	1	0	0	1	0	1	1	1.5Mhz - 2Mhz
0	1	0	0	1	1	1	1	2Mhz - 2.67Mhz
0	1	0	1	0	1	0	0	2.625Mhz - 3.5Mhz
0	1	0	1	1	0	1	1	3.5Mhz - 4.67Mhz
0	1	1	0	0	0	1	1	4.5Mhz - 6Mhz
0	1	1	0	1	1	1	1	6Mhz - 8Mhz
0	1	1	1	1	0	1	1	7.5Mhz - 10Mhz
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Key Register

The Key Register is used in conjunction with the JSRB instruction to facilitate the communication between the user program in Flash Memory and the User ISP/Virtual E^2 subroutines in the Boot ROM. The user is required to write a six bit key, as shown in Table 11-7, to the Key Register before executing the JSRB instruction. If the key matches a Key bit will be set and will remain set for eight instruction cycles, and will then reset. If the key bit is not set when the JSRB instruction is executed, the instruction will jump to the equivalent address within the first 256 bytes of flash program memory.

Table 11-7 KEY Register Write Format

KEY when Writing							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
1	0	0	1	1	0	X	X

Bits 7 - 2: Key value that must be written to set the KEY bit.

Bits 1 - 0: Don't care.

11.4 MICROWIRE/PLUS ISP

When the Flash is empty (the Flex bit of the Option Register is zero) and the device is reset, program execution commences from the Boot ROM. Code resident in this Boot ROM allows the user to program the device in the application system through the use of the MICROWIRE/PLUS interface.

National Semiconductor provides a program, which is available from our web site at www.national.com/cop8, that is capable of programming a device from the parallel port of a PC. The software accepts manually input commands and is capable of downloading standard Intel HEX Format files.

Users who wish to write their own MICROWIRE/PLUS ISP host software should refer to the COP8 FLASH ISP User Manual, available from the same web site. This document includes details of command format and delays necessary between command bytes.

11.4.1 Commands

The MICROWIRE/PLUS ISP supports the following features and commands:

- Write a value to the ISP Write Timing Register. NOTE: This must be the first command after entering MICROWIRE/PLUS ISP mode.
- Erase the entire flash program memory (mass erase).
- Erase a page at a specified address.
- Read Option register.
- Read a byte from a specified address.
- Write a byte to a specified address.
- Read multiple bytes starting at a specified address.
- Write multiple bytes starting at a specified address.
- Exit ISP and return execution to flash program memory.

PGMTIM_SET - Sets the flash write timing register to match that of the CKI frequency.

Description: Figure 11-2 shows the format of the PGMTIM_SET command. The PGMTIM_SET command will transfer the next byte sent into the flash programming time register. No acknowledgment will be sent. The symbol t_1 denotes the time delay between the command byte and the setting of the PGMTIM register. This command is always available. This command must be used before any “writes” can occur (i.e., page erase, mass erase, write byte or block write). See Table 11-21 for the value(s) of t_1

and t_2. Table 11-8 shows valid values for the PGMTIM register. This command is security independent.

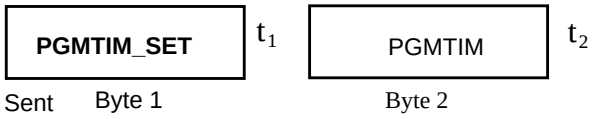


Figure 11-2 The Set PGMTIM Command

PAGE_ERASE - Erase a page of flash memory.

Description: Figure 11-3 shows the format of the PAGE_ERASE command. The PAGE_ERASE command will erase a 128 byte page (depends on the array size, 64 byte for devices containing 1k and 4k) from the flash memory. The next two bytes after the PAGE_ERASE byte refer to the high and low bytes of the beginning address of the target flash page. A WAIT/READY technique is used to delay the host when the controller is executing and writing to the flash memory. For a full description of the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY (**section 11.4.5 on page 15**)**. The symbol t_1, t_2 denotes the time delay between the command byte, the delay required after loading the high address byte, and the delay after loading of the low address byte. The symbol t_3 denotes the time delay after loading the ADDRESS_LO value. The PAGE_ERASE command is **NOT** always available (i.e., it is security dependent). If security is set, then the command will be aborted and no acknowledgment will be sent back. See section 10.4 for details on the number of endurance cycles and the number of page erase commands that should be issued prior to writing data into the erased page. See Table 11-12 for the value(s) of t_1, t_2, and t_3.

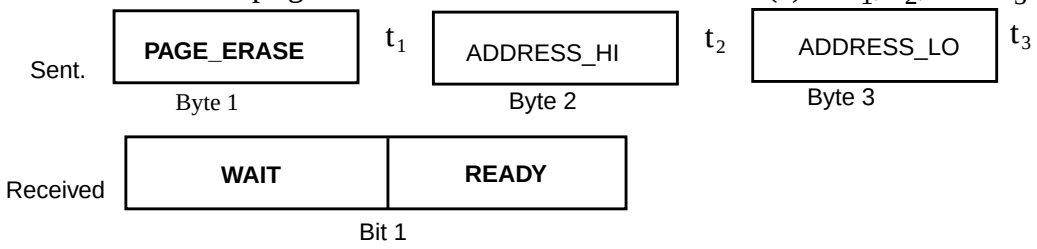


Figure 11-3 The PAGE ERASE command.

MASS_ERASE - Erase the entire flash memory array.

Description: Figure 11-4 shows the format of the MASS_ERASE command. The MASS_ERASE command will erase the entire flash memory, including the Option Register. The next byte after the MASS_ERASE command refers to the confirmation key used to double check that a mass erase request was actually sent. The confirmation key must equal 0x55 in order for the MASS_ERASE command to continue. The symbol t_1 denotes the time delay between the command byte and the transmission of the CONFIRM_KEY. The symbol t_2 denotes the time delay after the

CONFIRM_KEY has been checked. A WAIT/READY technique is used to delay the host when the controller is executing and writing to the flash memory. For a full description regarding the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY (**section 11.4.5 on page 15). The MASS_ERASE command is always available. It is security independent. See Table 11-12 for the value(s) of t_1, and t_2.

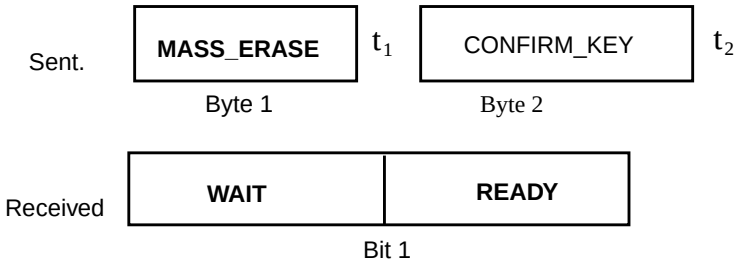


Figure 11-4 The MASS_ERASE command.

READ_BYTE - Read a byte from the flash memory array.

Description: Figure 11-5 shows the format of the READ_BYTE command. The READ_BYTE command will read a byte from the flash memory. The next two bytes after the READ_BYTE refer to the address of the target flash location. The symbol t_1, t_2 denotes the time delay between the command byte, the delay after loading of the high address byte. Data is sent back after t_3 delay(s) has elapsed. If security is set, the user is only allowed to read location 0xFFFF (Option Register). In other words, if security is set and ADDRESS_HI and ADDRESS_LO=0xFFFF then the firmware will allow that operation, otherwise it will return a 0xFF in the DATA_RTN byte. See Table 11-12 for the value(s) of t_1, t_2, and t_3.

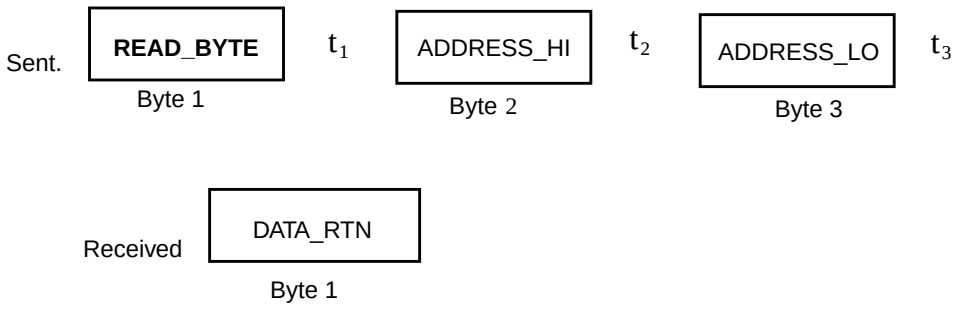


Figure 11-5 The READ_BYTE Command

WRITE_BYTE - Write a byte to the flash memory array.

Description: Figure 11-6 shows the format of the WRITE_BYTE routine. The WRITE_BYTE command will write a byte to the flash memory. The next two bytes after the WRITE_BYTE byte refer to the high and low byte address of the target flash location. The next byte (DATA_REC) after the

ADDRESS_LO byte will contain the value that will be stored into the flash location. The symbols t_1, t_2 denote the time delay between the command byte and the delay after loading of the high address byte. The symbol t_3 denotes the time delay after loading the ADDRESS_LO value. Data is saved into the flash location after a t_4 delay. A WAIT/READY signal is used to delay the host. For full description regarding the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY** (section 11.4.5 on page 15). This command, WRITE_BYTE, is **NOT** always available (i.e. it is security dependent.) If security is set, then the command will be aborted and no acknowledgment will be sent back. See Table 11-12 for the value(s) of t_1, t_2, t_3, and t_4.

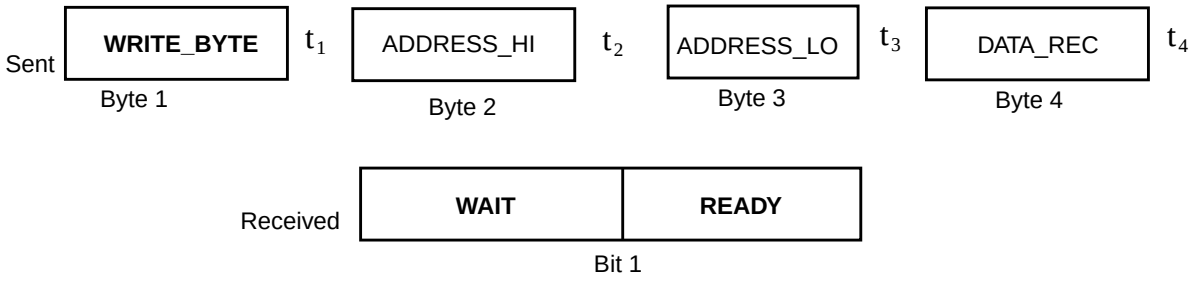


Figure 11-6 The WRITE_BYTE Command

BLOCK WRITE - Write a block of data to the flash memory array.

Description: Figure 11-7 is a symbolic representation of the BLOCK_WRITE routine. Data is written in sequential order. This routine is intended to write bytes of data which will reside in a page of flash memory. The next two bytes after the BLOCK_WRITE byte refer to the beginning high and low byte address of the target flash location. The next byte after the ADDRESS_LO byte refers to the BYTECOUNTLO variable. The BYTECOUNTLO variable is used by the microcontroller to transfer N bytes (i.e, N=BYTECOUNTLO). The maximum number of bytes that can be written is 16. If the number of bytes exceeds 16, it may not be guaranteed that all of the bytes were written. The data cannot cross page boundaries. Data must be placed with-in the same 1/2 page segment, 64 bytes for 32k devices and 32 bytes for 1k and 4k devices. This is due to the multi-byte write limitation. If N=0 then the firmware will abort. The symbols t_1 and t_2 denote the time delay between the command byte and the delay after loading of the high address byte. The symbol t_3 denotes the time delay after loading the ADDRESS_LO value. The symbol t_4 denotes the necessary time delay after loading the BYTECOUNTLO variable. Data arrives at t_5 cycles after the ADDRESS_LO value is loaded (i.e. DATA1 - DATA2 has the same delay as DATA2 - DATA3). After the last byte (DATA_N) is received, a WAIT/ READY signal will be sent to delay the host. For full description regarding the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY** (section 11.4.5 on page 15)**.** The command (BLOCK_WRITE) is **NOT** always

available (i.e. it is security dependent). If security is set, then the command will be aborted after the last data (DATA_N) is received and no acknowledgment will be sent back. See Table 11-12 for the value(s) of t_1, t_2, t_3, t_4, t_5, and t_6.

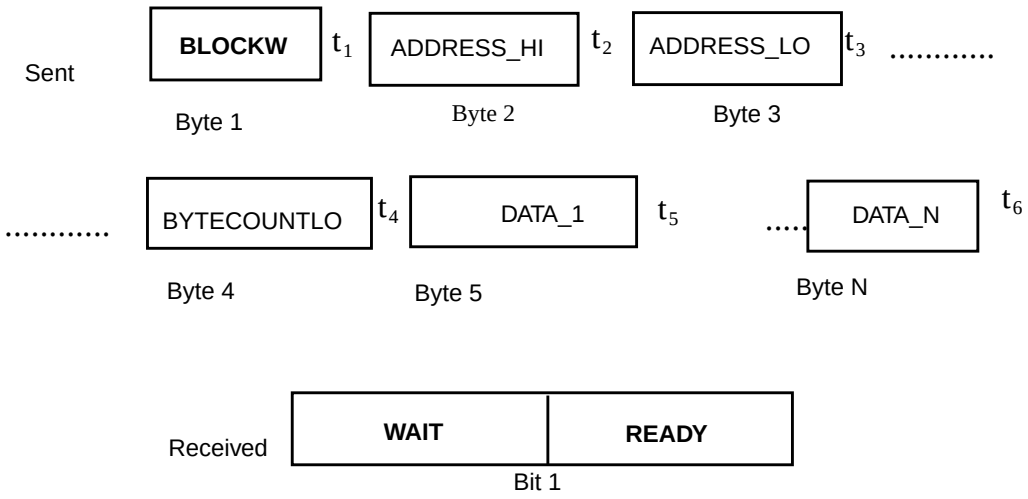


Figure 11-7 The Block Write Routine.

BLOCK_READ - Read a block from the flash memory array.

Description: Figure 11-8 shows the format of the BLOCK_READ command. The BLOCK_READ command will read multiple bytes from the flash memory. The next two bytes after the BLOCK_READ byte refer to the beginning high and low byte address of the target flash location. The next two bytes after the ADDRESS_LO byte refer to the upper and lower byte of BYTECOUNT. The BYTECOUNT variable is used by the microcontroller to return N bytes (i.e, N=BYTECOUNT). The maximum value of N is 32k. If N=0 then the firmware will abort. The symbols t_1, t_2, t_3 denotes the time delay between the command byte, the delay in loading of the ADDRESS_HI, and the delay after loading the ADDRESS_LO. The symbol t_4 denotes the required time delay between loading BYTECOUNTHI and BYTECOUNTLO. Subsequent data are sent to the host at t_5 cycles after BYTECOUNTLO (i.e. DATA1 - DATA2 has the same delay as DATA2 - DATA3). This command is capable of sending up to 32kB of flash memory through the MICROWIRE/PLUS. This command is always available however, if security is set, the user is only allowed to read 0xFFFF (Option Register). In other words, if at anytime ADDRESS_HI and ADDRESS_LO=0xFFFF, the firmware will allow that operation. If at any time ADDRESSHI and ADDRESS_LO does not equal 0xFFFF

and security is set, then the firmware will return 0xFF. This routine will acknowledge by returning data to the host.

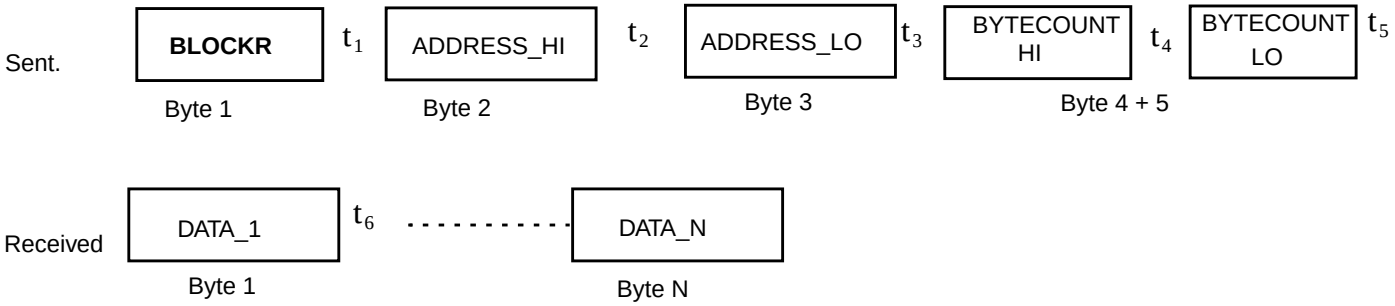


Figure 11-8 The Block Read Command.

EXIT - Reset the microcontroller.

Description: Figure 11-9 shows the format of the EXIT command. The EXIT command will reset the microcontroller. There is no additional information required after the EXIT byte is received. No acknowledgment will be sent back regarding the operation. This command is always available. It is security independent.

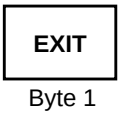


Figure 11-9 The EXIT Command.

Summary

The following table summarizes the MICROWIRE/PLUS ISP commands and provides information on required parameters and return values.

Table 11-8 MICROWIRE/PLUS ISP Commands

Command	Function	Command Value (Hex)	Parameters	Return Data
PGMTIM_SET	Write Pulse Timing Register	0x3B	Value	N/A
PAGE_ERASE	Page Erase	0xB3	Starting Address of Page	N/A
MASS_ERASE	Mass Erase	0xBF	Confirmation Code	N/A (The entire Flash Memory will be erased)

Table 11-8 MICROWIRE/PLUS ISP Commands

Command	Function	Command Value (Hex)	Parameters	Return Data
READ_BYTE	Read Byte	0x1D	Address High, Address Low	Data Byte if Security not set. 0xFF if Security set Option Register if address = 0xFFFF, regardless of Security
BLOCKR	Block Read	0xA3	Address High, Address Low, Byte Count (n) High, Byte Count (n) Low $0 \le n \le 32767$	n Data Bytes if Security not set. n Bytes of 0xFF if Security set
WRITE_BYTE	Write Byte	0x71	Address High, Address Low, Data Byte	N/A
BLOCKW	Block Write	0x8F	Address High, Address Low, Byte Count ($0 \le n \le 16$), n Data Bytes	N/A
EXIT	EXIT	0xD3	N/A	N/A (Device will Reset)
INVALID	N/A		Any other invalid command will be ignored	N/A

11.4.2 Firmware - MICROWIRE/PLUS Initialization

The MICROWIRE/PLUS support block will initialize the internal communication block with the following parameters: CTRL.MSEL=1, PORTGD.SO=1, PORTGD.SK=1, PORTGC.SI=1, and PORTGC.SK=0. Refer to Section 5.2 for more information on MICROWIRE/PLUS operation. Figure 11-10 shows waveforms of the MICROWIRE/PLUS operation.

.

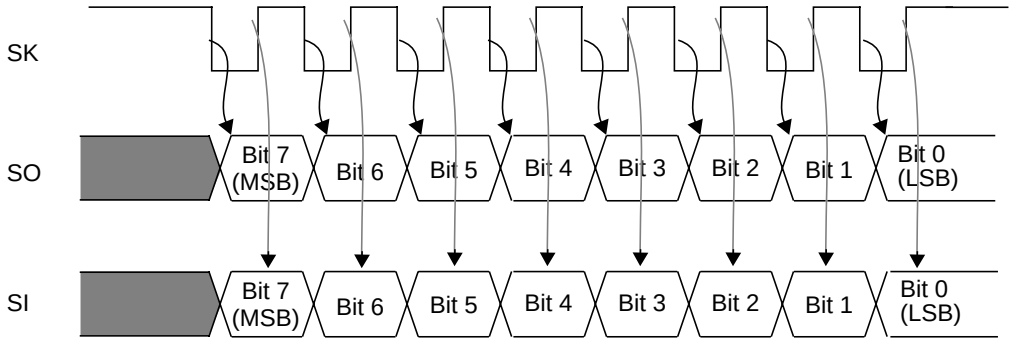


Figure 11-10 MICROWIRE/PLUS Interface Timing, Normal SK Mode, SK Idle Phase being High

Table 11-9 Initialization of the MICROWIRE/PLUS by the firmware.

Port G Config. Reg. Bits G5-G4	MICROWIRE/ PLUS Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
0-1	Slave, data out and data in	SO Output	SK Input	SI Input

Table 11-10 MICROWIRE/PLUS mode selected by the firmware.

Port G		SO Clocked out on:	SI Sampled on:	SK Idle Phase
G6 (SKSEL) Config. Bit	G5 Data Bit			
1	1	SK Falling edge	SK Rising edge	High

11.4.3 The MICROWIRE/PLUS Packet Composition

A typical MICROWIRE/PLUS packet is composed of a three byte frame (although this varies with the chosen command). Figure 11-11 is a symbolic representation of the ISP-MICROWIRE/PLUS packet. A trigger byte is a value which will cause an ISP (In System Programming) command to be executed (e.g. erase, read or write a byte of flash). The COMMAND Byte holds this trigger byte value. Refer to Table 11-8 for valid MICROWIRE/PLUS commands and their trigger byte values. Bytes ADDRESS_HI and ADDRESS_LO refer to the high and low byte address of the flash memory that is to be operated upon. The symbol t_{delay} represents the delay that is required when sending the

command, ADDRESS_HI and ADDRESS_LO bytes. Table 11-8 shows the valid commands and their corresponding byte value.

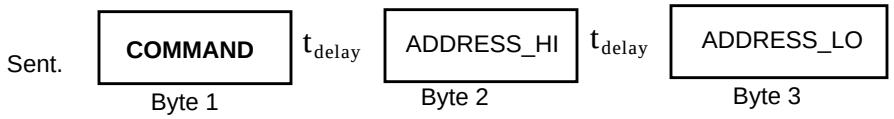


Figure 11-11 ISP Command Frame

11.4.4 Required Delays In Cascading MICROWIRE/PLUS Command Frames

A certain amount of delay must be observed when sending multiple command frames in a data stream. The symbol $t_{cascade\text{-}delay}$ represents the delay that is required when sending several commands in a data stream. The host must wait $t_{cascade\text{-}delay}$ cycles before sending the next command frame to the COP8 Flash Family device. Figure 11-12

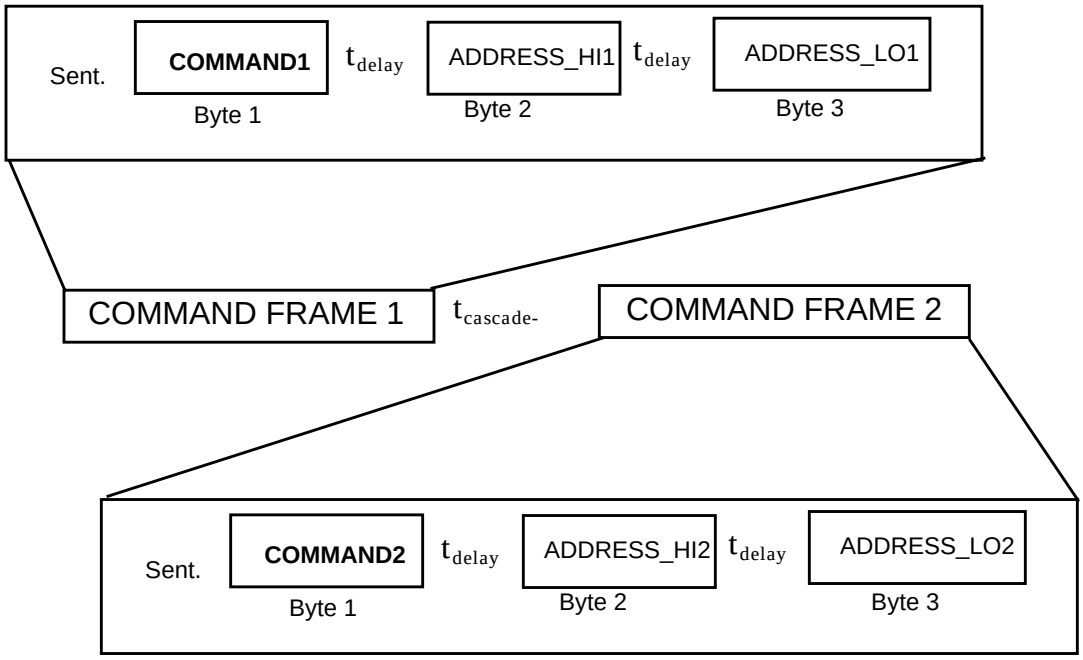


Figure 11-12 Cascade Delay Requirement

shows the delay relationship. Refer to Table 11-11 for the values of $t_{cascade\text{-}dela}$. Refer to Table 11-12 for the values of t_{delay}. The symbol $t_1 ... t_N$ denotes individual delay requirements (which varies among different commands).

Table 11-11 Required time delays (in instruction cycles) for cascading command frames **after** an initial command was executed.

COMMAND	$t_{CASCADE-DELAY}$
READ_BYTE	6
WRITE_BYTE	6
BLOCKR	13
BLOCKW	6
PGERASE	6
MASS_ERASE	6
EXIT	N/A
PGMTIM_SET	6

Table 11-12 Required time delays (in instruction cycles).

COMMAND	t_1	t_2	t_3	t_4	t_5	t_6	t_N
READ_BYTE	35	100	100	N/A	N/A	N/A	N/A
WRITE_BYTE	35	100	20	10	N/A	N/A	N/A
BLOCKR	35	100	100	100	140	140	140
BLOCKW	35	100	100	100	100	100	52
PGERASE	35	100	100	N/A	N/A	N/A	N/A
MASS_ERASE	35	100	N/A	N/A	N/A	N/A	N/A
EXIT	N/A	N/A	N/A	N/A	N/A	N/A	N/A
PGMTIM_SET	35	35	N/A	N/A	N/A	N/A	N/A

11.4.5 Variable Host Delay

A special type of communication has been implemented in the device firmware in order to allow the microcontroller enough time to complete a write or erase operation. This type of communication was developed since the microcontroller may be used in situations where the clock is extremely slow and writes to the flash memory will take a large amount of time. This implementation relieves the user of having to manually change the write delays in their host software. Figure 11-13 shows how the VARIABLE HOST DELAY configuration is implemented on a byte write. Figure 11-14 shows how the VARIABLE HOST DELAY configuration is implemented on a block write. Figure 11-15 shows how the VARIABLE HOST DELAY configuration is implemented on a page erase.

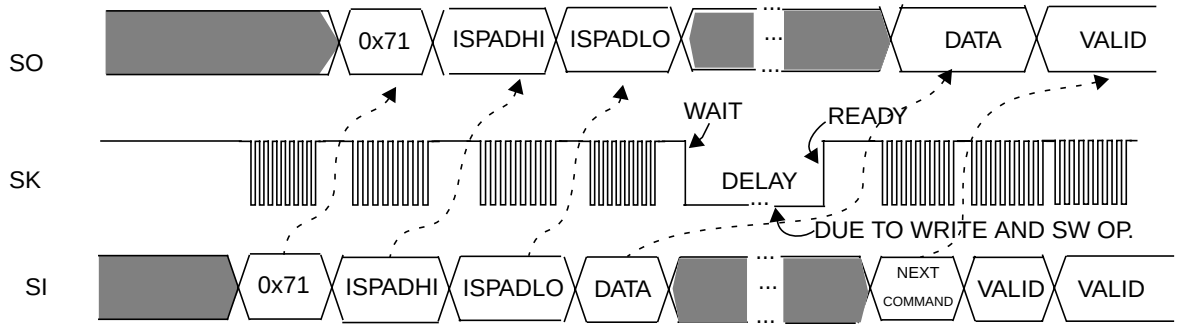


Figure 11-13 Byte Write Waveform (Relative Bytes Are Shown)

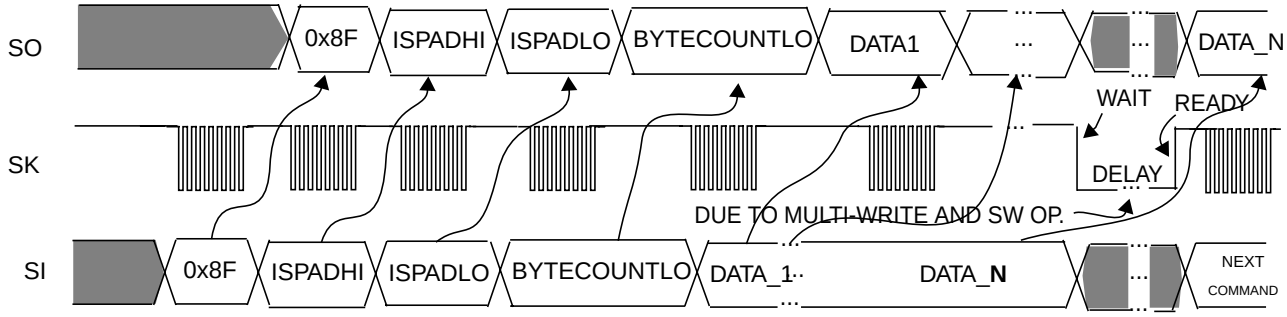


Figure 11-14 Block Write Waveform (Relative Bytes Are Shown)

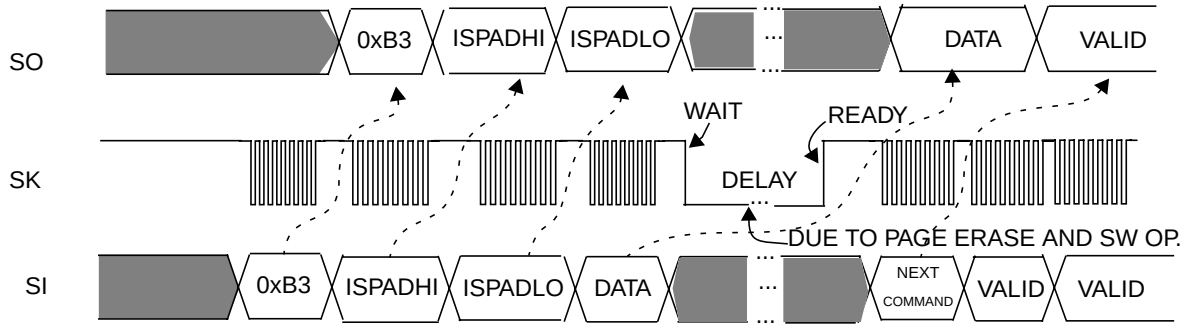


Figure 11-15 Page Erase Waveform (Relative Bytes Are Shown)

Figure 11-16 shows how the VARIABLE HOST DELAY configuration is implemented on a mass erase. Since the SK (Serial CLOCK) is normally high, the microcontroller brings SK low to indicate to the host that a WAIT condition (i.e. the SK pin is low) exists. The host then goes into a loop until the WAIT condition changes to a READY condition (i.e.,

the SK pin is high again). The controller then returns to command decode and waits for the next command.

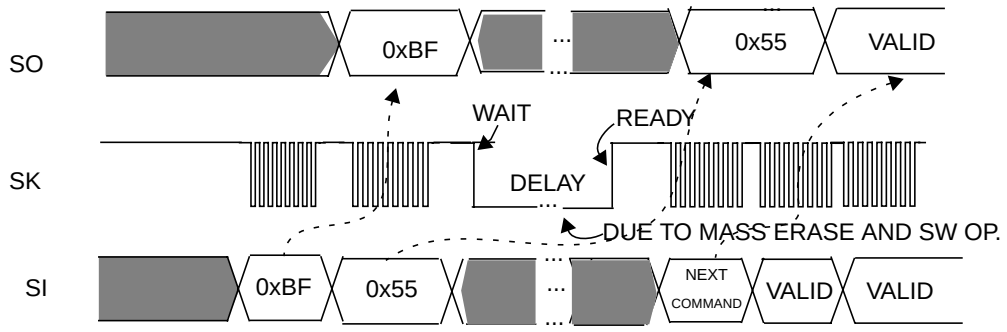


Figure 11-16 Mass Erase Waveform (Relative Bytes Are Shown)

11.4.6 MICROWIRE/PLUS - Boot ROM Startup Behavior

Upon startup, the ISP boot ROM will detect if the G6 pin is high. This is used to detect if a high voltage condition on the G6 pin is present (i.e., a forced boot ROM re-entry due to code lockup, for additional information refer to section 11.5.3 on page 24. By using this technique the boot ROM avoids any bit that may be inadvertently entered on the SI pin. If the G6 pin is not high at startup, the ISP boot ROM will try to detect if a valid command is received on a transmission. If a valid command is received, the boot ROM firmware will check to see if the **SECURITY** bit is set. Table 11-8 shows the valid MICROWIRE/PLUS commands. If security is set, the boot ROM will disable all ISP functions except the reading of the **OPTION** register at 0xFFFF, the execution of a mass erase on the flash memory and the setting of the PGMTIM Register. Read attempts of flash memory, other than location 0xFFFF, Option Register, while security is set, will result with a 0xFF sent back through the MICROWIRE/PLUS. See Figure 11-17 for the ISP - MICROWIRE/PLUS Control flow. In general, the boot ROM firmware will decode the command, check security, execute the command (if security is off) and MICROWIRE/PLUS Main Support Block (e.g., triggering the PSW.BUSY bit in order to send the data back to the host.)

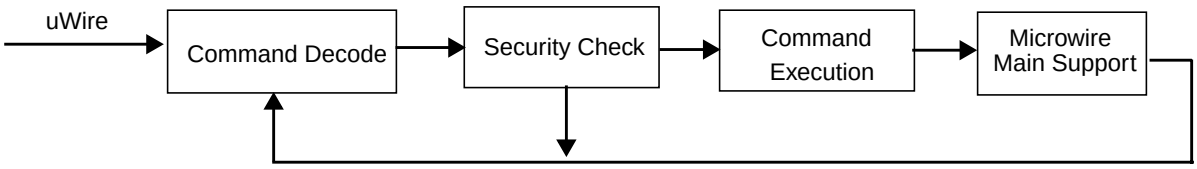


Figure 11-17 The ISP - MICROWIRE Con-

11.5 USER ISP

Some users may wish to define and provide their own firmware update procedure and provide this capability within the application software. These devices provide for this

capability with subroutines which are resident in the Boot ROM which perform the function of copying information between RAM and Flash.

These subroutines allow the user to provide the means of communication and the memory management for firmware updates while using the built-in functions for the actual read and write of the Flash. This allows the user to utilize any available peripheral for host communications, e.g. USART, parallel port, etc., and to define the protocol.

The user is also required to enforce the security required by the application. Whereas the MICROWIRE/PLUS ISP and parallel programming are available to anyone, User ISP is only available to the programmer of a specific application. The user may thus secure applicaton code from external access while still providing a means of partial or complete firmware update.

11.5.1 Flash/Boot ROM Communication

When using ISP, at some point, it will be necessary to maneuver between the flash program memory and the Boot ROM, even when using customized ISP routines. This is because it's not possible to execute from the flash program memory while it's being programmed.

A new instruction is available to perform the link between the User's code in Flash and the program in Boot ROM: Jump to Subroutine in Boot ROM(JSRB). The JSRB instruction is used to jump from flash memory to Boot ROM. See Appendix A, INSTRUCTION SET for specific details on the operation of these instructions.

The JSRB instruction must be used in conjunction with the Key register. This is to prevent jumping to the Boot ROM in the event of run-away software. For the JSRB instruction to actually jump to the Boot ROM, the Key bit must be set. This is done by writing the value shown in Table 11-7 to the Key register. The Key is a 6 bit key and if the key matches, the KEY bit will be set for 8 instruction cycles. The JSRB instruction must be executed while the KEY bit is set. If the KEY does not match, then the KEY bit will not be set and the JSRB will jump to the specified location in the flash memory. In emulation mode, if a breakpoint is encountered while the KEY is set, the counter that counts the instruction cycles will be frozen until the breakpoint condition is cleared. The Key register is a memory mapped register. Its format when writing is shown in Table 11-7. Its format when reading is shown in Table 11-8. In normal operation, it is not necessary to test the KEY bit before using the JSRB instruction. The additional instructions required to test the key may cause the key to time-out before the JSRB can be executed.

Restrictions On Software When Calling ISP Routine In Boot ROM

1. The hardware will disable interrupts from occurring. The hardware will leave the GIE bit in its current state, and if set, the hardware interrupts will occur when execution is returned to Flash Memory. Subsequent interrupts, during ISP operation, from the same interrupt source will be lost.

2. The security feature in the MICROWIRE/PLUS ISP is guaranteed by software and not hardware. When executing the MICROWIRE/PLUS ISP rou-

tine, the security bit is checked prior to performing all instructions. Only the mass erase command, write PGMTIM register, and reading the Option register is permitted within the MICROWIRE/PLUS ISP routine. When the user is performing his own ISP, all commands are permitted. The entry points from the user's ISP code do not check for security. It is the burden of the user to guarantee his own security. See the Security bit description in device specfic chapter for more details on security.

3. When using any of the ISP functions in Boot ROM, the ISP routines will service the WATCHDOG within the selected upper window. Upon return to flash memory, the WATCHDOG is serviced, the lower window is enabled, and the user can service the WATCHDOG anytime following exit from Boot ROM, but must service it within the selected upper window to avoid a WATCHDOG error.

11.5.2 Commands

The following commands will support transferring blocks of data from RAM to flash program memory, and vice-versa. The user is expected to enforce application and program security in this case.

- Erase the entire flash program memory (mass erase). NOTE: Execution of this command will force the device into the MICROWIRE/PLUS ISP mode.
- Erase a page of flash memory at a specified address.
- Read a byte from a specified address.
- Write a byte to a specified address.
- Copy a block of data from RAM into flash program memory.
- Copy a block of data from program flash memory to RAM.

cpgerase - User Entry Point: Erase a page of flash memory

This routine requires that ISPADHI and ISPADLO are loaded before the jump. A KEY is a number which must be loaded into the KEY Register (location 0xE2) before issuing a JSRB instruction. Table 11-7 shows the format of the KEY number. Loading the KEY, and a "JSRB cpgerase" are all that is needed to complete the call to the routine. No acknowledgement will be returned regarding the operation. For details regarding the registers ISPADHI and ISPADLO please refer to Table 11-21. See Section 11.2 for details on the number of endurance cycles and the number of page erase commands that should be issued prior to writing data into the erased page. Since this is a user command, this routine will work regardless of security (security independent). Table 11-13 shows the resources needed to run the routine.

Table 11-13 Resource utilization for the command: cpgerase (Page Erase)

INPUT DATA	REGISTERS USED	WD SERVICED	JSRB/KEY REQUIRED	RETURNED DATA/ LOCATION	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
ISPADHI ISPADLO	A and B	YES	YES	NONE	120 + 100*PGMTIM	4

cmserase - User Entry Point: Mass erase the flash memory

This routine requires the Accumulator A to contain 0x55 prior to the jump. The value 0x55 is used to verify that a mass erase was requested. “LD A,#055”, loading the KEY and a “JSRB cmserase” are all that is needed to complete the function. No acknowledgement will be returned regarding the operation. Since this is a user command, this routine will work regardless of security (security independent). After a mass erase is executed the user will be brought back (after 112 instruction cycles) to the beginning of the boot ROM. Control and execution will be returned to the MICROWIRE/ PLUS ISP handling routine. Table 11-14 shows the resources needed to run the routine.

Table 11-14 Resource utilization for the command: cmserase (Mass Erase)

INPUT DATA	REGISTERS USED	WD SERVICED	JSRB/KEY REQUIRED	RETURNED DATA/ LOCATION	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
CONFIRMATION CODE “0X55” IS LOADED INTO ACCUMULATOR (A)	A and B	YES	YES	NONE	120 + 300*PGMTIM	6

creadbf - User Entry Point: Read a byte of flash memory

This routine requires that ISPADHI and ISPADLO are loaded before the jump. Loading the KEY, and a “JSRB creadbf” are all that is needed to complete the call to the routine. Data will be returned in the ISPRD Register. No acknowledgement will be returned regarding the operation. For details regarding the register ISPADHI, ISPADLO, and ISPRD please refer to Table 11-21. Since this is a user command, this routine will work regardless of security (security independent). Table 11-15 shows the resources needed to run the routine.

Table 11-15 Resource utilization for the command: creadbf (Read a byte of flash memory)

INPUT DATA	REGISTERS USED	WD SER-VICED	JSRB/KEY REQUIRED	RETURNED DATA/ LOCATION	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
ISPADHI ISPADLO	A and B	YES	YES	DATA /ISPRD REGISTER	100	4

cblockr - User Entry Point: Read a block of flash memory

The cblockr routine will read multiple bytes from the flash memory. ISPADHI and ISPADLO are assumed to be loaded before the jump. Register BYTECOUNTLO is also

assumed to be loaded. The X pointer contains the address where the data will be placed. The BYTECOUNTLO register is used by the microcontroller to send back N bytes (i.e., N=BYTECOUNTLO). If N=0 then the firmware will abort. Data is saved into the RAM address pointed to by the X pointer. It is up to the user to setup the segment register. This routine is capable of reading up to 255 bytes of flash memory (limited due to the mem available) to RAM. This routine is limited to reading blocks of 128 bytes due to the RAM segmentation. If an attempt to read greater than 128 bytes is issued, the firmware will begin to write to RAM locations beginning at 0x80 (possibly corrupting the I/O and CONTROL REGISTERS) and above. After the X pointer and the BYTECOUNTLO is set, the KEY must be loaded, and a "JSRB cblockr" must be issued. No acknowledgement will be returned regarding the operation. For details regarding the register ISPADHI, ISPADLO, and BYTECOUNTLO please refer to Table 11-21. Since this is a user command, this routine will work regardless of security (security independent). Table 11-16 shows the resources needed to run the routine.

Table 11-16 Resource utilization for the command: cblockr (Block read of the flash memory)

INPUT DATA	REGISTERS USED	WD SERVICED	JSRB/KEY REQUIRED	RETURNED DATA/LOCATION	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
BYTCOUNTLO ISPADHI ISPADLO	A, B and X	YES	YES	DATA /RAM[X]	140/BYTE	6

cblockw - User Entry Point: Write to a block flash memory

ISPADHI and ISPADLO must be set by the user prior to the jump into the command. The BYTECOUNTLO variable is used by the microcontroller to transfer N bytes (i.e, N=BYTECOUNTLO). This variable also must be set prior to the jump into the command. The maximum number of bytes that can be written is 16. This number is determined by Flash timing requirements and the minimum allowed clock speed. If the number of bytes exceed 16, then the user may not be guaranteed that all of the bytes were written. The data cannot cross 1/2 page boundaries (i.e. all data must be within the 64 bytes segment for 32k devices and within 32 bytes for 4k, and 1k devices). If N=0 then the firmware will abort. Data is read from the RAM address pointed to by the X pointer. It is up to the user to setup the segment register. Data transfers will take place from wherever the RAM locations are specified by the segment register. However, if the X pointer exceeds the top of the segment, the firmware will begin to transfer from 0x80 (I/O and CONTROL REGISTERS) and above. After the X pointer and the BYTECOUNTLO is set, the KEY must be loaded, and a "JSRB cblockw" must be issued. For details regarding the register ISPADHI, ISPADLO, and BYTECOUNTLO please refer to Table 11-21. Since this is a user command, this routine will work regardless of security (security independent). Table 11-17 shows the resources needed to run the routine.

Table 11-17 Resource utilization for the command: cblockw (Write to a block of flash memory)

INPUT DATA	REGISTERS USED	WD SERVICED	JSRB/KEY REQUIRED	RETURN DATA	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
BYTECOUNTLO DATA IS ASSUMED TO BE IN THE ***RAM[X]*** LOCATION(S)	A, B and X	YES	YES	NONE	100+3.5*PGMTIM/BYTE + 68/BYTE	6

cwritebf - User Entry Point: Write a byte to the flash memory.

This routine requires that ISPADHI, ISPADLO, and ISPWR be loaded prior to the jump. Loading the KEY and a "JSRB cwritebf" are all that is needed to complete this call. No acknowledgement will be returned regarding the operation. For details regarding the register ISPADHI, ISPADLO, and ISPRD please refer to Table 11-21. Since this is a user command, this routine will work regardless of security (security independent). Table 11-18 shows the resources needed to run the routine.

Table 11-18 Resource utilization for the command: cwritebf (Write a byte to the flash).

INPUT DATA	REGISTERS USED	WD SERVICED	JSRB/KEY REQUIRED	RETURNED DATA/ LOCATION	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
ISPWR CONTAINS THE DATA	A and B	YES	YES	NONE	168 + 3.5 * PGMTIM	4

Exit - Reset the Microcontroller

This routine will cause the microcontroller to reset itself. Loading the KEY, and a "JSRB Exit" are the only actions needed to complete the call. No additional parameters need to be passed. Since this is a user command, this routine will work regardless of security (security independent). Table 11-19 shows the resources needed to run the routine.

Table 11-19 Resource utilization for the command: exit (reset the microcontroller)

INPUT DATA	REGISTERS USED	JSRB/KEY REQUIRED	RETURN DATA	INTERRUPT LOCK OUT CYCLES	STACK USAGE (IN BYTES)
N/A	A	YES	NONE	100	2

Summary

The following table summarizes the User ISP commands, required parameters and return data, if applicable. The command entry point is used as an argument to the JSRB instruction. Table 11-21 lists the Ram locations and Peripheral Registers used for User ISP and their expected contents. Please refer to the COP8 FLASH ISP User Manual for additional information and programming examples on the use of User ISP.

Table 11-20 User ISP/Virtual E^2 Entry Points

Command/ Label	Function	Command Entry Point	Parameters	Return Data
cpgerase	Page Erase	0x17	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address.	N/A (A page of memory beginning at ISPADHI, ISPADLO will be erased)
cmserase	Mass Erase	0x1A	Accumulator A contains the confirmation key 0x55.	N/A (The entire Flash Memory will be erased)
creadbf	Read Byte	0x11	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address.	Data Byte in Register ISPRD.
cblockr	Block Read	0x26	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. X pointer contains the beginning RAM address where the result(s) will be returned. Register BYTECOUNTLO contains the number of n bytes to read ($0 \le n \le 255$). It is up to the user to setup the segment register.	n Data Bytes, Data will be returned beginning at a location pointed to by the RAM address in X.
cwritebf	Write Byte	0x14	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. Register ISPWR contains the Data Byte to be written.	N/A
cblockw	Block Write	0x23	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. Register BYTECOUNTLO contains the number of n bytes to write ($0 \le n \le 16$) X pointer contains the beginning RAM address of the data to be written. It is up to the user to setup the segment register.	N/A
exit	EXIT	0x62	N/A	N/A (Device will Reset)

Table 11-21 Register and Bit Name Definitions

Register Name	Purpose	RAM LOCATION
ISPADHI	High byte of Flash Memory Address	0xA9
ISPADLO	Low byte of Flash Memory Address	0xA8
ISPWR	The user must store the byte to be written into this register before jumping into the write byte routine	0xAB
ISPRD	Data will be returned to this register after the read byte routine execution	0xAA
ISPKEY	the ISPKEY Register is required to validate the JSRB instruction and must be loaded within 6 instruction cycles before the JSRB	0xE2
BYTECOUNTLO	Holds the count of the number of bytes to be read or written in block operations	0xF1
PGMTIM	Write Timing Register. This register must be loaded, by the user, with the proper value before execution of any USER ISP Write or Erase operation. Refer to Table 11-6 for the correct value.	0xE1
Confirmation Code	The user must place this code in the accumulator before execution of a Flash Memory Mass Erase command	A
KEY	Must be transferred to the ISPKEY register before execution of a JSRB instruction	0x98

11.5.3 Forced Execution From Boot ROM

When the user is developing a customized ISP routine, code lockups due to software errors may be encountered. There is a hardware method to get out of these lockups and force execution from the Boot ROM MICROWIRE/PLUS routine, so that the customer can erase the Flash Memory code and start over. The method to force this condition is to drive the G6 pin to high voltage ($2 \times V_{cc}$) and activate Reset. The high voltage condition on G6 must be held for at least 3 instruction cycles longer than Reset is active. This special condition will start execution from location 0000 in the Boot ROM where the user can input the appropriate commands, using MICROWIRE/PLUS, to erase the flash program memory and reprogram it.

While the G6 pin is at high voltage, the Load Clock will be output onto G5, which will look like an SK clock to the MICROWIRE/PLUS routine executing in slave mode. However, when G6 is at high voltage, the G6 input will also look like a logic 1. The MICROWIRE/PLUS routine in Boot ROM monitors the G6 input, waits for it to go low, debounces it, and then enables the ISP routine. CAUTION: The Load clock on G5 could be in conflict with the user's external SK. It is up to the user to resolve this conflict, as

this condition is considered a minor issue that's only encountered during software development. **The user should also be cautious of the high voltage applied to the G6 pin. This high voltage could damage other circuitry connected to the G6 pin (e.g. the parallel port of a PC).** The user may wish to disconnect other circuitry while G6 is connected to the high voltage.

11.5.4 Interrupt Lock Out Time

Interrupts are inhibited during execution from Boot ROM. Table 11-22 shows the amount of time that the user is **LOCKED OUT** of interrupt service. The servicing of interrupts will be resumed once the ISP boot ROM returns the user to the flash. Any interrupt(s) that are pending during user ISP will be serviced after execution has returned to the flash area. The user should take into account the amount of time the application is locked out of interrupts. Some of the **LOCK OUT** times are dependent upon the PGMTIM. PGMTIM is a value entered into the PGMTIM register (refer to Section 11.3.2 regarding PGMTIM). The user code **MUST** set the PGMTIM register before any write or erase routines occur (e.g., a LD PGMTIM,#06F is needed to specify a CKI frequency of 6 Mhz).

Table 11-22 REQUIRED INTERRUPT LOCKOUT TIME (IN INSTRUCTION CYCLES)

FLASH ROUTINES (USER ACCESSABLE)	MINIMUM INTERRUPT LATENCY TIME (IN INSTRUCTION CYCLES)
cpgerase	$120 + 100 \times PGMTIM$
cmserase	$120 + 300 \times PGMTIM$
creadbf	100
cblockr	$140/(BYTE)$
cblockw	$100 + 3.5 \times PGMTIM/(BYTE) + 68/(BYTE)$
cwritebf	$168 + 3.5 \times PGMTIM$
exit	N/A

11.5.5 WATCHDOG Services

The Watchdog register will be serviced periodically in order to ensure that a watchdog event does not occur. All routines in the ISP boot ROM incorporate automatic watchdog services. Periodically, the boot ROM firmware will service the watchdog if the routine takes a large number of cycles. The boot ROM firmware will service the watchdog below the 8k upper window requirement. The lower Watchdog service window is inhibited during execution from Boot ROM. The subroutines in Boot ROM will service the Watchdog immediately prior to returning execution to the Flash program. The user must not service the Watchdog within the lower service window after returning from the User ISP subroutines.

11.6 VIRTUAL E^2

Since the requirements of Virtual E^2 are identical to some of the requirements of User ISP, i.e. the need to copy blocks of information between RAM and the Flash memory, the Virtual E^2 uses the same commands as the User ISP. Obviously the Mass Erase command will not be useful in Virtual E^2 applications.

Chapter 12

COP8SBR/SCR/SDR

12.1 INTRODUCTION

The COP8SBR/SCR/SDR is a member of the COP8 Flash Family 8-bit microcontrollers. Like all members of this family, it provides high-performance, economical solutions for embedded control applications.

The types of features available in the COP8 Feature Family are listed below, together with the quantity or availability of each feature in the COP8SBR/SCR/SDR.

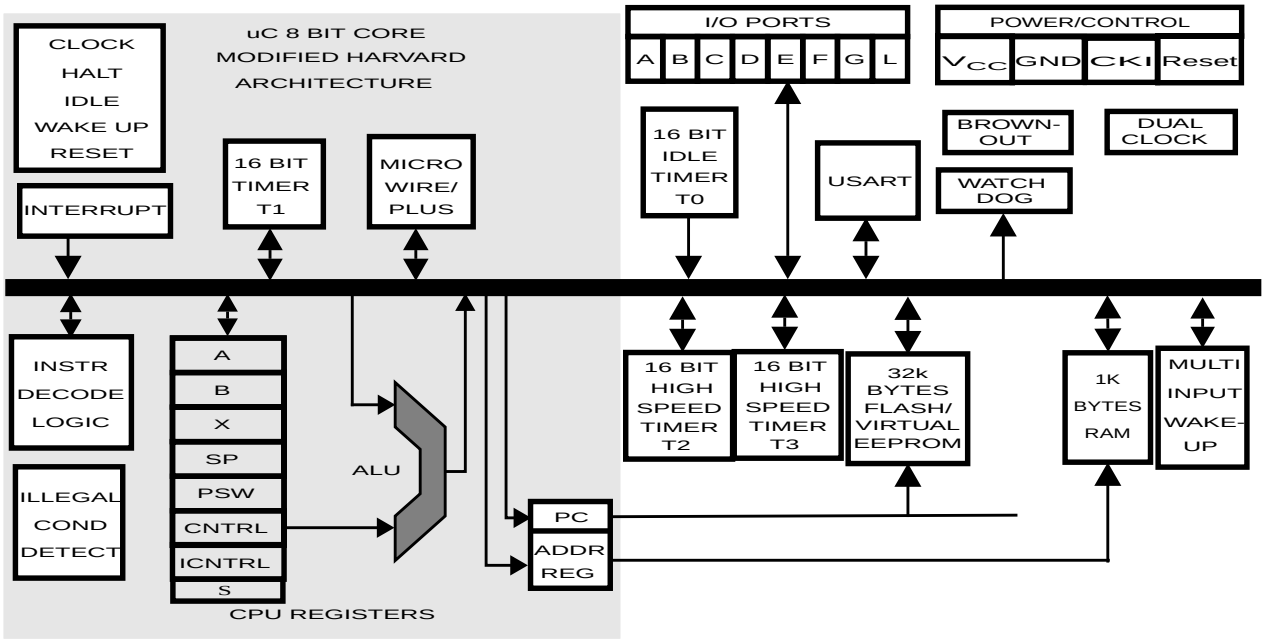
- Program Memory: Flash, 32K bytes
- Data Memory: Static RAM, 1K bytes
- S-Register Data Memory Extension: Yes
- Clock Doubler: Yes
- Dual Clock: Yes
- 16-Bit Programmable Timers: Timers T1, T2, and T3 (T2 and T3 capable of High Speed Operation)
- IDLE Mode and Timer: Yes, with programmable IDLE interrupt interval
- Multi-Input Wakeup/Interrupt: Yes
- Watchdog and Clock Monitor: Yes (can be disabled in the Option Register)
- Additional feature: Brown-out Reset (COP8SBR and COP8SCR Only)

Refer to the COP8SBR/SCR/SDR data sheet for more specific information on different voltage and temperature ranges for operation.

This chapter describes the device-specific features of the COP8SBR/SCR/SDR. Information that applies to all COP8 Flash Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

12.2 BLOCK DIAGRAM

The following figure is a block diagram showing the basic functional blocks of the COP8SBR/SCR/SDR. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP8 device communicate with the core through an internal bus.



12.3 DEVICE PINOUTS/PACKAGES

The COP8SBR/SCR/SDR is available in 68 and 44-pin PLCC packages. Figure 12-1 shows the COP8SBR/SCR/SDR device package pinout.

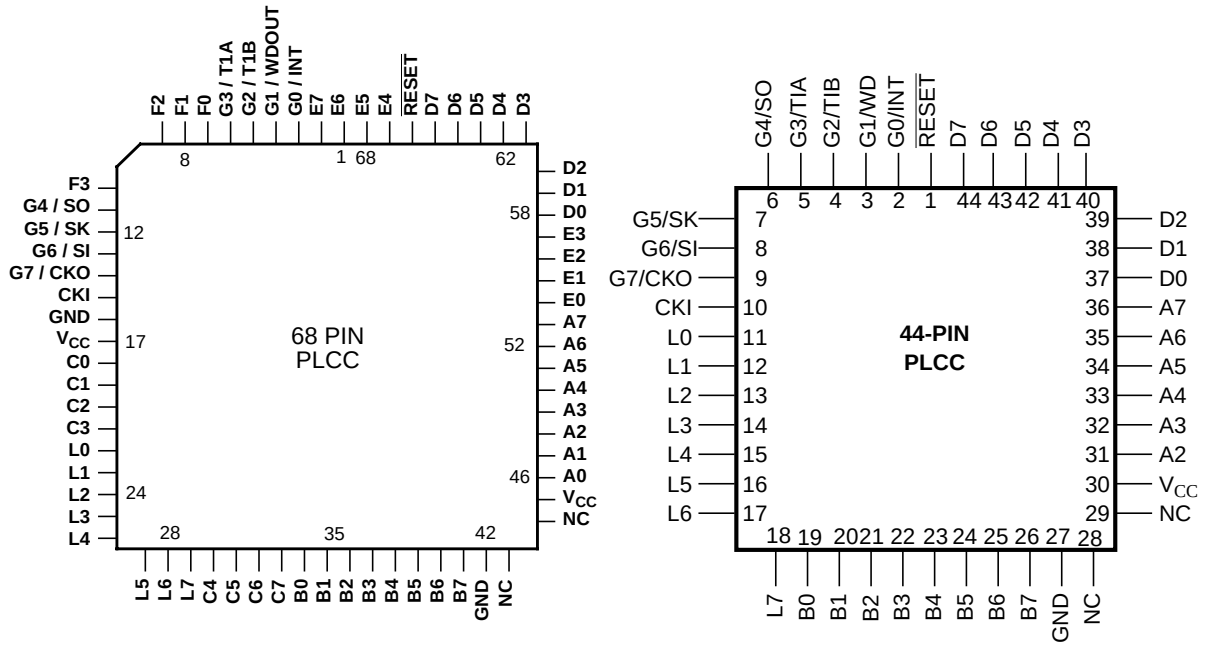


Figure 12-1 Device Package Pinout

Refer to the COP8SBR/SCR/SDR data sheet for more detailed package information.

12.4 PIN DESCRIPTIONS

The COP8SBR/SCR/SDR has four dedicated function pins: V_{CC}, GND, CKI and $\overline{\text{RESET}}$. V_{CC} and GND function as the power supply pins as well as provide the lower and upper reference voltages that define the input range for the A/D Converter. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 12-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package type.

Table 12-1 COP8SBR/SCR/SDR Pinouts

Port	Type	Alt. Fun	In System Emulation Mode	44 Pin PLCC	68 Pin PLCC
L0	I/O	MIWU or Low Speed Osc In		11	22
L1	I/O	MIWU or CKX or Low Speed Osc. Out		12	23
L2	I/O	MIWU or TDX		13	24
L3	I/O	MIWU or RTX		14	25
L4	I/O	MIWU or T2A		15	26
L5	I/O	MIWU or T2B		16	27
L6	I/O	MIWU or T3A		17	28
L7	I/O	MIWU or T3B		18	29
G0	I/O	INT	INPUT	2	3
G1	I/O	WDOUT[a]	POUT	3	4
G2	I/O	T1B	OUTPUT	4	5
G3	I/O	T1A	CLOCK	5	6
G4	I/O	SO		6	11
G5	I/O	SK		7	12
G6	I	SI		8	13
G7	I	CKO		9	14
D0	O			37	58
D1	O			38	59
D2	O			39	60
D3	O			40	61
D4	O			41	62
D5	O			42	63
D6	O			43	64
D7	O			44	65
E0	I/O				54
E1	I/O				55
E2	I/O				56
E3	I/O				57

Table 12-1 COP8SBR/SCR/SDR Pinouts

Port	Type	Alt. Fun	In System Emulation Mode	44 Pin PLCC	68 Pin PLCC
E4	I/O				67
E5	I/O				68
E6	I/O				1
E7	I/O				2
C0	I/O				18
C1	I/O				19
C2	I/O				20
C3	I/O				21
C4	I/O				30
C5	I/O				31
C6	I/O				32
C7	I/O				33
A0	I/O				46
A1	I/O				47
A2	I/O			31	48
A3	I/O			32	49
A4	I/O			33	50
A5	I/O			34	51
A6	I/O			35	52
A7	I/O			36	53
B0	I/O			19	34
B1	I/O			20	35
B2	I/O			21	36
B3	I/O			22	37
B4	I/O			23	38
B5	I/O			24	39
B6	I/O			25	40
B7	I/O			26	41
F0	I/O				7
F1	I/O				8
F2	I/O				9
F3	I/O				10
VCC			VCC	30	17,45
GND			GND	27	16,42
CKI	I			10	15
RESET	I		Reset	1	66

a. G1 operation as WDOUT is controlled by Option Register bit 2.

12.5 INPUT/OUTPUT PORTS

The general I/O port functions are described in Chapter 7. The COP8SBR/SCR/SDR device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate function of the port pin.

Port A is an 8-bit I/O port. All A pins have Schmitt triggers on the inputs. The 44-pin package does not have a full 8-bit port and contains some unbonded, floating pads internally on the chip. The binary value read from these bits is undetermined. The application software should mask out these unknown bits when reading the Port A register, or use only bit-access program instructions when accessing Port A. These unconnected bits draw power only when they are addressed (i.e., in brief spikes).

Port B is an 8-bit I/O port. All B pins have Schmitt triggers on the inputs.

Port C is an 8-bit I/O port. The 44 pin device does not offer Port C. The unavailable pins are not terminated. A read operation on these unterminated pins will return unpredictable values. On this device, the associated Port C Data and Configuration registers should not be used. All C pins have Schmitt triggers on the inputs. Port C draws no power when unbonded.

Port E is an 8-bit I/O Port. The 44 pin device does not offer Port E. The unavailable pins are not terminated. A read operation on these unterminated pins will return unpredictable values. On this device, the associated Port E Data and Configuration registers should not be used. All E pins have Schmitt triggers on the inputs. Port E draws no power when unbonded.

Port F is an 4-bit I/O Port. All F pins have Schmitt triggers on the inputs.

The 68 pin package has fewer than eight Port F pins, and contains unbonded, floating pads internally on the chip. The binary values read from these bits are undetermined. The application software should mask out these unknown bits when reading the Port F register, or use only bit-access program instructions when accessing Port F. The unconnected bits draw power only when they are addressed (i.e., in brief spikes).

Port G is an 8-bit port. Pin G0, G2-G5 are bi-directional I/O ports. Pin G6 is always a general purpose Hi-Z input. All pins have Schmitt Triggers on their inputs. **Pin G1 serves as the dedicated WATCHDOG output with weak pullup if the WATCHDOG feature is selected by the Option register. The pin is a general purpose I/O if the WATCHDOG feature is not selected.** If the WATCHDOG feature is selected, bit 1 of the Port G configuration and data register does not have any effect on Pin G1 setup. G7 serves as the dedicated output pin for the CKO clock output.

Since G6 is an input only pin and G7 is the dedicated CKO clock output pin, the associated bits in the data and configuration registers for G6 and G7 are used for special purpose functions as outlined below. Reading the G6 and G7 data bits will return zeros.

The device will be placed in the HALT mode by writing a “1” to bit 7 of the Port G Data Register. Similarly the device will be placed in the IDLE mode by writing a “1” to bit 6 of the Port G Data Register.

Writing a “1” to bit 6 of the Port G Configuration Register enables the MICROWIRE/PLUS to operate with the alternate phase of the SK clock. The G7 configuration bit, if set high, enables the clock start up delay after HALT when the R/C clock configuration is used.

	Config Reg.	Data Reg.
G7	CLKDLY	HALT
G6	Alternate SK	IDLE

Port G has the following alternate features:

G7 CKO Oscillator dedicated output

G6 SI (MICROWIRE/PLUS Serial Data Input)

G5 SK (MICROWIRE/PLUS Serial Clock)

G4 SO (MICROWIRE/PLUS Serial Data Output)

G3 T1A (Timer T1 I/O)

G2 T1B (Timer T1 Capture Input)

G1 WDOUT WATCHDOG and/or Clock Monitor if WATCHDOG enabled, otherwise it is a general purpose I/O

G0 INTR (External Interrupt Input)

G0 through G3 are also used for In-System Emulation.

Port L is an 8-bit I/O port. All L-pins have Schmitt triggers on the inputs.

Port L supports the Multi-Input Wake Up feature on all eight pins.Port L has the following alternate pin functions:

L7 Multi-input Wakeup or T3B (Timer T3B Input)

L6 Multi-input Wakeup or T3A (Timer T3A Input)

L5 Multi-input Wakeup or T2B (Timer T2B Input)

L4 Multi-input Wakeup or T2A (Timer T2A Input)

L3 Multi-input Wakeup and/or RDX (USART Receive)

L2 Multi-input Wakeup or TDX (USART Transmit)

L1 Multi-input Wakeup and/or CKX (USART Clock) (Low Speed Oscillator Output)

L0 Multi-input Wakeup (Low Speed Oscillator Input)

12.6 PROGRAM MEMORY

The COP8SBR/SCR/SDR contains 32K bytes of Flash memory, used for storing application programs and fixed program data. The Flash occupies the program memory address space from 0000 to 07FFF hex. It is addressed by the 15-bit Program Counter (PC).

12.7 DATA MEMORY

The COP8SBR/SCR/SDR contains 1K bytes of static RAM memory, used for temporary data storage. The first 128 bytes of RAM occupy two address ranges in the lowest 256-byte data memory space: 112 bytes from 00–6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers. The remaining RAM, which is used for general-purpose storage, occupies the bottom of the next seven 256-byte pages of data memory space, starting at 0100 Hex and ending at 077F Hex.

12.8 REGISTER BIT MAPS

The COP8SBR/SCR/SDR has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 12-4 through 12-9 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4), Watchdog and Clock Monitor (Chapter 8), Multi-Input Wakeup/ Interrupt and Timer T0 (Chapter 6).

.

Table 12-2 HSTCR Register (Address X'00AF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESERVED						T3HS	T2HS

Reserved	These bits are reserved and must be zero.
T3HS	Places Timer T3 in High Speed Mode.
T2HS	Places Timer T2 in High Speed Mode.

Table 12-3 T3CNTRL, Timer T3 Control Register (Address xxB6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T3C3	T3C2	T3C1	T3C0	T3PNDA	T3ENA	T3PNDB	T3ENB

T3C3-T3C2-T3C1-T3C0: Timer T3 control bits

T3PNDA: Timer T3 interrupt A pending flag

T3ENA: Timer T3 interrupt A enable bit

T3PNDB: Timer T3 interrupt B pending flag

T3ENB: Timer T3 interrupt B enable bit

Table 12-4 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB

T2C3-T2C2-T2C1-T2C0: Timer T2 control bits

T2PNDA: Timer T2 interrupt A pending flag

T2ENA: Timer T2 interrupt A enable bit

T2PNDB: Timer T2 interrupt B pending flag

T2ENB: Timer T2 interrupt B enable bit

Table 12-5 WDSVR, Watchdog Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN

WS1–WS0: Watchdog Window Select bits

KEY4–KEY0: Watchdog Key Data (01100)

CMEN: Clock Monitor Enable bit

Table 12-6 ITMR, IDLE Timer Control Register (Address xxCF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LSON	HSON	DCEN	CCKSEL	RESERVED	ITSEL2	ITSEL1	ITSEL0

LSON	Turns the low speed oscillator on or off.
HSON	Turns the high speed oscillator on or off.
DCEN	Selects the high speed oscillator or the low speed oscillator as the Idle Timer Clock.
CCKSEL	Selects the high speed oscillator or the low speed oscillator as the primary CPU clock.
RESERVED:	This bit is Reserved and should be zero.
ITSEL2–ITSEL0:	IDLE Timer Interrupt Period Select Bits: 000 = 4,096 Idle Timer Clocks 001 = 8,192 Idle Timer Clocks 010 = 16,384 Idle Timer Clocks 011 = 32,768 Idle Timer Clocks 100 = 65,536 Idle Timer Clocks 101 = Reserved - Undefined 110 = Reserved - Undefined 111 = Reserved - Undefined

Table 12-7 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	uWPND	uWEN	T1PNDB	T1ENB

LPEN:	Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)
T0PND:	Timer T0 (IDLE Timer) interrupt pending flag
T0EN:	Timer T0 (IDLE Timer) interrupt enable bit
uWPND:	MICROWIRE interrupt pending flag
uWEN:	MICROWIRE interrupt enable bit
T1PNDB:	Timer T1 interrupt B pending flag
T1ENB:	Timer T1 interrupt B enable bit

Table 12-8 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0
T1C3-T1C2-T1C1-T1C0: Timer T1 control bits							
MSEL: MICROWIRE Select bit							
IEDG: External Interrupt Edge selection bit							
SL1–SL0: MICROWIRE clock divide-by selection bits							

Table 12-9 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE
HC: Half-Carry bit							
C: Carry bit							
T1PNDA: Timer T1 interrupt A pending flag							
T1ENA: Timer T1 interrupt A enable bit							
EXPND: External interrupt pending flag							
BUSY: MICROWIRE Busy flag							
EXEN: External interrupt enable bit							
GIE: Global Interrupt Enable							

12.9 MEMORY MAP

Table 12-10 is a memory map showing the organization of the data memory and the specific memory addresses of the COP8SBR/SCR/SDR registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 12-10 COP8SBR/SCR/SDR Data Memory Map

Address S/ADD REG	Contents
0000 to 006F	On-Chip RAM bytes (112 bytes)
0070 to 007F	Unused RAM Address Space (Reads As All Ones)
xx80 to xx90	Unused RAM Address Space (Reads Undefined Data)
xx90	Port E Data Register
xx91	Port E Configuration Register
xx92	Port E Input pins (Read Only)
xx93	Reserved for Port E
xx94	Port F Data Register
xx95	Port F configuration Register
xx96	Port F Input Pins (read only)
xx97	Reserved for Port F (Reads Undefined Data)
xx98 to xx9F	Reserved (Reads Undefined Data)
xxA0	Port A Data Register
xxA1	Port A Configuration Register
xxA2	Port A Input pins (Read Only)
xxA3	Reserved for Port A (Reads Undefined Data)
xxA4	Port B Data Register
xxA5	Port B Configuration Register
xxA6	Port B Input pins (Read Only)
xxA7	Reserved for Port B (Reads Undefined Data)
xxA8	ISP Address Register Low Byte (ISPADLO)
xxA9	ISP Address Register High Byte (ISPADHI)
xxAA	ISP Read Data Register (ISPRD)
xxAB	ISP Write Data Register (ISPWR)
xxAC to xxAF	Reserved (Reads Undefined Data)
xxB0	Timer T3 Lower Byte
XXB1	Timer T3 Upper Byte
xxB2	Timer T3 Autoload Register T3RA Lower Byte
xxB3	Timer T3 Autoload Register T3RA Upper Byte
xxB4	Timer T3 Autoload Register T3RB Lower Byte
xxB5	Timer T3 Autoload Register T3RB Upper Byte
xxB6	Timer T3 Control Register
xxB7	Reserved (Reads Undefined Data)
xxB8	USART Transmit Buffer (TBUF)

Table 12-10 COP8SBR/SCR/SDR Data Memory Map (Continued)

Address S/ADD REG	Contents
xxB9	USART Receive Buffer (RBUF)
xxBA	USART Control and Status Register (ENU)
xxBB	USART Receive Control and Status Register (ENUR)
xxBC	USART Interrupt and Clock Source Register (ENUI)
xxBD	USART Baud Register (BAUD)
xxBE	USART Prescale Select Register (PSR)
xxBF	Reserved for USART (Reads Undefined Data)
xxC0	Timer T2 Lower Byte
xxC1	Timer T2 Upper Byte
xxC2	Timer T2 Autoload Register T2RA Lower Byte
xxC3	Timer T2 Autoload Register T2RA Upper Byte
xxC4	Timer T2 Autoload Register T2RB Lower Byte
xxC5	Timer T2 Autoload Register T2RB Upper Byte
xxC6	Timer T2 Control Register
xxC7	WATCHDOG Service Register (Reg:WDSVR)
xxC8	MIWU Edge Select Register (Reg:WKEDG)
xxC9	MIWU Enable Register (Reg:WKEN)
xxCA	MIWU Pending Register (Reg:WKPND)
xxCB to xxCE	Reserved (Reads Undefined Data)
xxCF	Idle Timer Control Register (ITMR)
xxD0	Port L Data Register
xxD1	Port L Configuration Register
xxD2	Port L Input Pins (Read Only)
xxD3	Reserved for Port L
xxD4	Port G Data Register
xxD5	Port G Configuration Register
xxD6	Port G Input Pins (Read Only)
xxD7	Reserved (Reads Undefined Data)
xxD8	Port C Data Register
xxD9	Port C Configuration Register
xxDA	Port C Input Pins (Read Only)
xxDB	Reserved for Port C (Reads Undefined Data)
xxDC	Port D
xxDD to DF	Reserved for Port D (Reads Undefined Data)

Table 12-10 COP8SBR/SCR/SDR Data Memory Map (Continued)

Address S/ADD REG	Contents
xxE0	Reserved (Reads Undefined Data)
xxE1	Flash Memory Write Timing Register (PGMTIM)
xxE2	ISP Key Register (ISPKEY)
xxE3 to xxE5	Reserved (Reads Undefined Data)
xxE6	Timer T1 Autoload Register T1RB Lower Byte
xxE7	Timer T1 Autoload Register T1RB Upper Byte
xxE8	ICNTRL Register
xxE9	MICROWIRE/PLUS Shift Register
xxEA	Timer T1 Lower Byte
xxEB	Timer T1 Upper Byte
xxEC	Timer T1 Autoload Register T1RA Lower Byte
xxED	Timer T1 Autoload Register T1RA Upper Byte
xxEE	CNTRL Control Register
xxEF	PSW Register
xxF0 to FB	On-Chip RAM Mapped as Registers
xxFC	X Register
xxFD	SP Register
xxFE	B Register
xxFF	S Register
0100–017F	On-Chip 128 RAM Bytes
0200–027F	On-Chip 128 RAM Bytes
0300–037F	On-Chip 128 RAM Bytes
0400–047F	On-Chip 128 RAM Bytes
0500–057F	On-Chip 128 RAM Bytes
0600–067F	On-Chip 128 RAM Bytes
0700–077F	On-Chip 128 RAM Bytes
Reading memory locations 0070H–007FH (Segment 0) will return all ones. Reading unused memory locations 0080H–0093H (Segment 0) will return undefined data. Reading memory locations from other Segments (i.e., Segment 8, Segment 9, ... etc.) will return undefined data.	

12.10 RESET

Upon reset of the COP8SBR/SCR/SDR, the ports and registers are initialized as follows:

Port A data register, PORTAD: 00

Port A configuration register, PORTAC: 00

Port B data register, PORTBD: 00

Port B configuration register, PORTBC: 00

Port C data register, PORTCD:	00
Port C configuration register, PORTCC:	00
Port D data register, PORTD:	FF
Port E data register, PORTED:	00
Port E configuration register, PORTEC:	00
Port F data register, PORTFD:	00
Port F configuration register, PORTFC:	00
Port G data register, PORTGD:	00 (If the Watchdog is enabled in the Option Register, the G1 pin will be enabled as the Watchdog output with a weak pullup.)
Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timers T2 and T3 control registers, T2CNTRL and T3CNTRL:	00
Timers T1, T2 and T3 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
High Speed Timer Control Register, HSTCR	00
Accumulator, A and Timers T1, T2 and T3:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
S RAM Segment Register	00

Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
Watchdog service register, WDSVR:	D9
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
IDLE Timer and Dual Clock Control Register, ITMR	40

12.11 INTERRUPTS

Table 12-11 shows the types of interrupts in the COP8SBR/SCR/SDR, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP8 interrupts, see Chapter 3.

Table 12-11 COP8SBR/SCR/SDR Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	USART (Receive)	01EE–01EF
10	USART (Transmit)	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	Timer T3A/Underflow	01E6–01E7
14	Timer T3B	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1
* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.		

12.12 OPTION REGISTER

The Option register, located at address 0x7FFF in the Flash Program Memory, is used to configure the user selectable security, WATCHDOG, and HALT options. The register can be programmed only in external Flash Memory programming or ISP Programming modes. Therefore, the register must be programmed at the same time as the program memory. The contents of the Option register shipped from the factory read 00 Hex.

The format of the Option register is as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved		SECURITY	Reserved		WATCHDOG	HALT	FLEX

Bits 7,6		These bits are reserved and must be 0
Bit 5	= 1	Security enabled. Flash Memory read and write are not allowed. Mass Erase is Allowed
	= 0	Security disabled. Flash Memory read and write are allowed.
Bits 4,3		These bits are reserved and must be 0
Bit 2	= 1	WATCHDOG feature disabled. G1 is a general purpose I/O.
	= 0	WATCHDOG feature enabled. G1 pin is WATCHDOG output with weak pullup.
Bit 1	= 1	HALT mode disabled.
	= 0	HALT mode enabled.
Bit 0	= 1	Execution following RESET will be from Flash Memory
	= 0	Flash Memory is erased. Execution following RESET will be from Boot ROM with the MICROWIRE/PLUS™ ISP routines.

The user must ensure that the FLEX bit will be set when the device is programmed.

Appendix A

INSTRUCTION SET

A.1 INTRODUCTION

This chapter defines the instruction set. It provides detailed information about every instruction including addressing modes, opcodes, and functions. In addition, it covers instruction decoding, execution, and timing for all instructions.

A.2 INSTRUCTION FEATURES

The strength of the instruction set is based on the following features:

- Majority of single-byte opcode instructions to minimize program size.
- One instruction cycle for the majority of single-byte instructions to minimize program execution time.
- Many single-byte, multiple function instructions such as DRSZ.
- Three memory mapped pointers; two for register indirect addressing, and one for the software stack.
- Sixteen memory mapped registers which allow an optimized implementation of certain instructions.
- Ability to set, reset and test any individual bit in data memory address space, including the memory mapped I/O ports and registers.
- Register-Indirect LOAD and EXCHANGE instructions with optional automatic post-incrementing or decrementing of the register pointer. This allows for greater efficiency (both in cycle time and program code) in loading, walking across and processing fields in data memory.
- Unique instructions to optimize program size and throughput efficiency. Some of these instructions are: DRSZ, IFBNE, DCOR, RETSK, VIS and RRC.

A.3 ADDRESSING MODES

The COP800 instruction set offers a variety of methods for specifying memory addresses. Each method is called an “addressing mode.” These modes are classified into two categories: “operand” addressing modes and “transfer-of-control” addressing modes. Operand addressing modes are the various methods of specifying an address for accessing (reading or writing) data. Transfer-of-control addressing modes are used in conjunction with “Jump” instructions to control the execution sequence of the software program.

A.3.1 Operand Addressing Modes

The operand of an instruction specifies what memory location is to be affected by that instruction. Several different operand addressing modes are available, allowing memory locations to be specified in a variety of ways. An instruction can specify an address directly by supplying the specific address, or indirectly by specifying a register pointer. The contents of the register (or in some cases, two registers) point to the desired memory location. In the “immediate” mode, the data byte to be used is contained in the instruction itself.

Each addressing mode has its own advantages and disadvantages with respect to flexibility, execution speed, and program compactness. Not all modes are available with all instructions. The Load (LD) instruction offers the largest number of addressing modes.

The available addressing modes are:

- Direct
- Register B or X Indirect
- Register B or X Indirect with Post-Incrementing/Decrementing
- Immediate
- Immediate Short
- Indirect from Program Memory

The addressing modes are described below. Each description includes an example of an assembly language instruction using the described addressing mode.

Direct. The memory address is specified directly as a byte in the instruction. In assembly language, the direct address is written as a numerical value (or a label that has been defined elsewhere in the program as a numerical value).

Example: Load Accumulator Memory Direct

LD A,05

Reg/Data Memory	Contents Before	Contents After
Accumulator	XX Hex	A6 Hex
Memory Location 0005 Hex	A6 Hex	A6 Hex

Register B or X Indirect. The memory address is specified by the contents of the B Register or X register (pointer register). In assembly language, the notation [B] or [X] specifies which register serves as the pointer.

Example: Exchange Memory with Accumulator, B Indirect

X A,[B]

Reg/Data Memory	Contents Before	Contents After
Accumulator	01 Hex	87 Hex
Memory Location 0005 Hex	87 Hex	01 Hex
B Pointer	05 Hex	05 Hex

Register B or X Indirect with Post-Incrementing/Decrementing. The relevant memory address is specified by the contents of the B Register or X register (pointer register). The pointer register is automatically incremented or decremented after execution, allowing easy manipulation of memory blocks with software loops. In assembly language, the notation [B+], [B-], [X+], or [X-] specifies which register serves as the pointer, and whether the pointer is to be incremented or decremented.

Example: Exchange Memory with Accumulator, B Indirect with Post-Increment

X A,[B+]

Reg/Data Memory	Contents Before	Contents After
Accumulator	03 Hex	62 Hex
Memory Location 0005 Hex	62 Hex	03 Hex
B Pointer	05 Hex	06 Hex

Immediate. The data for the operation follows the instruction opcode in program memory. In assembly language, the number sign character (#) indicates an immediate operand.

Example: Load Accumulator Immediate

LD A,#05

Reg/Data Memory	Contents Before	Contents After
Accumulator	XX Hex	05 Hex

Immediate Short. This is a special case of an immediate instruction. In the “Load B immediate” instruction, the 4-bit immediate value in the instruction is loaded into the lower nibble of the B register. The upper nibble of the B register is reset to 0000 binary.

Example: Load B Register Immediate Short

LD B,#7

Reg/Data Memory	Contents Before	Contents After
B Pointer	12 Hex	07 Hex

Indirect from Program Memory. This is a special case of an indirect instruction that allows access to data tables stored in Program Memory. In the "Load Accumulator Indirect" (LAID) instruction, the upper and lower bytes of the Program Counter (PCU and PCL) are used temporarily as a pointer to Program Memory. For purposes of accessing Program Memory, the contents of the Accumulator and PCL are exchanged. The data pointed to by the Program Counter is loaded into the Accumulator, and simultaneously, the original contents of PCL are restored so that the program can resume normal execution.

Example: Load Accumulator Indirect

LAID

Reg/Data Memory	Contents Before	Contents After
PCU	04 Hex	04 Hex
PCL	35 Hex	36 Hex
Accumulator	1F Hex	25 Hex
Memory Location 041F Hex	25 Hex	25 Hex

A.3.2 Transfer-of-Control Addressing Modes

Program instructions are usually executed in sequential order. However, "Jump" instructions can be used to change the normal execution sequence. Several transfer-of-control addressing modes are available to specify jump addresses.

A change in program flow requires a non-incremental change in the Program Counter contents. The Program Counter consists of two bytes, designated the upper byte (PCU) and lower byte (PCL). The most significant bit of PCU is not used, leaving 15 bits to address the program memory.

Different addressing modes are used to specify the new address for the Program Counter. The choice of addressing mode depends primarily on the distance of the jump. Farther jumps sometimes require more instruction bytes in order to completely specify the new Program Counter contents.

The available transfer-of-control addressing modes are:

- Jump Relative
- Jump Absolute

- Jump Absolute Long
- Jump Indirect

The transfer-of-control addressing modes are described below. Each description includes an example of a "Jump" instruction using a particular addressing mode, and the effect on the Program Counter bytes of executing that instruction.

Jump Relative. In this 1-byte instruction, six bits of the instruction opcode specify the distance of the jump from the current program memory location. The distance of the jump can range from –31 to +32. A JP+1 instruction is not allowed. The programmer should use a NOP instead.

Example: Jump Relative

JP 0A

Reg	Contents Before	Contents After
PCU	02 Hex	02 Hex
PCL	05 Hex	0F Hex

Jump Absolute. In this 2-byte instruction, 12 bits of the instruction opcode specify the new contents of the Program Counter. The upper three bits of the Program Counter remain unchanged, restricting the new Program Counter address to the same 4-Kbyte address space as the current instruction. (This restriction is relevant only in devices using more than one 4-Kbyte program memory space.)

Example Jump Absolute

JMP 0125

Reg	Contents Before	Contents After
PCU	0C Hex	01 Hex
PCL	77 Hex	25 Hex

Jump Absolute Long. In this 3-byte instruction, 15 bits of the instruction opcode specify the new contents of the Program Counter.

Example: Jump Absolute Long

JMP 03625

Reg/ Memory	Contents Before	Contents After
PCU	42 Hex	36 Hex
PCL	36 Hex	25 Hex

Jump Indirect. In this 1-byte instruction, the lower byte of the jump address is obtained from a table stored in program memory, with the Accumulator serving as the low order byte of a pointer into program memory. For purposes of accessing program memory, the contents of the Accumulator are written to PCL (temporarily). The data pointed to by the Program Counter (PCH/PCL) is loaded into PCL, while PCH remains unchanged.

Example: Jump Indirect

JID

Reg/Memory	Contents Before	Contents After
PCU	01 Hex	01 Hex
PCL	C4 Hex	32 Hex
Accumulator	26 Hex	26 Hex
Memory Location 0126 Hex	32 Hex	32 Hex

NOTE: The VIS instruction is a special case of the Indirect Transfer of Control addressing mode, where the double-byte vector associated with the interrupt is transferred from adjacent addresses in program memory into the Program Counter in order to jump to the associated interrupt service routine.

A.4 INSTRUCTION TYPES

The instruction set contains a fairly wide variety of instructions. The available instructions are listed below, organized into related groups.

Some instructions test a condition and skip the next instruction if the condition is not true. Skipped instructions are executed as no-operation (NOP) instructions.

Arithmetic Instructions

The arithmetic instructions perform binary arithmetic such as addition and subtraction, with or without the Carry bit.

Add (ADD)

Add with Carry (ADC)

Subtract (SUB)

Subtract with Carry (SUBC)

Increment (INC)

Decrement (DEC)

Decimal Correct (DCOR)

Clear Accumulator (CLR)

Set Carry (SC)

Reset Carry (RC)

Transfer-of Control Instructions

The transfer-of-control instructions change the usual sequential program flow by altering the contents of the Program Counter. The Jump to Subroutine instructions save the Program Counter contents on the stack before jumping; the Return instructions pop the top of the stack back into the Program Counter.

Breakpoint (BRK)

Jump Relative (JP)

Jump Absolute (JMP)

Jump Absolute Long (JMPL)

Jump Indirect (JID)

Jump to Subroutine (JSR)

Jump to Subroutine Long (JSRL)

Jump to Boot ROM Subroutine (JSRB)

Return from Subroutine (RET)

Return from Subroutine and Skip (RETSK)

Return from Interrupt (RETI)

Return from Subroutine to Flash (RETF)

Software Trap Interrupt (INTR)

Vector Interrupt Select (VIS)

Load and Exchange Instructions

The load and exchange instructions write byte values in registers or memory. The addressing mode determines the source of the data.

Load (LD)

Load Accumulator Indirect (LAID)

Exchange (X)

Logical Instructions

The logical instructions perform the basic logical operations AND, OR, and XOR (Exclusive OR). Other logical operations can be performed by combining these basic operations. For example, complementing is accomplished by exclusive-ORing the Accumulator with FF Hex.

Logical AND (AND)

Logical OR (OR)

Exclusive OR (XOR)

Accumulator Bit Manipulation Instructions

The Accumulator bit manipulation instructions allow the user to shift the Accumulator bits and to swap its two nibbles.

Rotate Right Through Carry (RRC)

Rotate Left Through Carry (RLC)

Swap Nibbles of Accumulator (SWAP)

Stack Control Instructions

Push Data onto Stack (PUSH)

Pop Data off of Stack (POP)

Memory Bit Manipulation Instructions

The memory bit manipulation instructions allow the user to set and reset individual bits in memory.

Set Bit (SBIT)

Reset Bit (RBIT)

Reset Pending Bit (RPND)

Conditional Instructions

The conditional instructions test a condition. If the condition is true, the next instruction is executed in the normal manner; if the condition is false, the next instruction is skipped.

If Equal (IFEQ)

If Not Equal (IFNE)

If Greater Than (IFGT)

If Carry (IFC)

If Not Carry (IFNC)

If Bit (IFBIT)

If B Pointer Not Equal (IFBNE)

And Skip if Zero (ANDSZ)

Decrement Register and Skip if Zero (DRSZ)

No-Operation Instruction

The no-operation instruction does nothing, except to occupy space in the program memory and time in execution.

No-Operation (NOP)

A.5 DETAILED FUNCTIONAL DESCRIPTIONS OF INSTRUCTIONS

The instruction set contains 58 different instructions. Most of the arithmetic, comparison, and data transfer (load, exchange) instructions operate with three different addressing modes (register indirect with **B** pointer, memory direct, and immediate). These various addressing modes increase the instruction total to 87. The detailed instruction descriptions contain the following:

- Opcode mnemonic
- Instruction syntax with operand field descriptor
- Full instruction description
- Register level instruction description
- Number of instruction cycles (each cycle equal to one microsecond at full clock speed)
- Number of bytes (1, 2, or 3) in instruction
- Hexadecimal code for the instruction bytes

The following abbreviations represent the nomenclature used in the detailed instruction description and the COP800 cross-assembler:

A	Accumulator.
B	B Pointer, located in RAM register memory location 00FE.
[B]	Contents of RAM data memory location indicated by B pointer.
[B+]	Same as [B], except that B pointer is post-incremented.
[B-]	Same as [B], except that B pointer is post-decremented.
C	Carry flag, located in bit 6 of the PSW register at memory location 00EF Hex.

HC — Half Carry flag, located in bit 7 of the PSW register at memory location 00EF Hex.

MA — 8-bit memory address for RAM data store memory.

MD — Memory Direct, which may be represented by an implicit label (B, X, SP), a defined label (TEMP, COUNTER, etc.), or a direct memory address (12, 0EF, 027, etc., where a leading 0 indicates hexadecimal).

PC — Program Counter (15 bits, with a program memory addressing range of 32768).

PCU — Program Counter Upper, which contains the upper 7 bits of PC.

PCL — Program Counter Lower, which contains the lower 8 bits of PC.

PSW — Processor Status Word Register, found at memory location 00EF.

REG — Selected Register (1 of 16) from the RAM data store memory at addresses 00F0-00FF.

REG# — # of the memory register to be used (# = 0-F hexadecimal).

— # symbol is used to indicate an immediate value, with a leading zero (0) indicating hexadecimal.

EXAMPLES:

#045 = immediate value of hexadecimal 45

#45 = immediate value of decimal 45

may also be used to indicate bit position, where # = 0-7

EXAMPLE:

RBIT #, [B]

SP — Stack Pointer, located in RAM register memory location 00FD.

X — X pointer, located in RAM register memory location 00FC.

[X] — Contents of RAM data memory location indicated by the X pointer.

[X+] — Same as [X], except that the X pointer is post-incremented.

[X-] — Same as [X], except that the X pointer is post-decremented.

A.5.1 ADC— Add with Carry

Syntax: a)ADC A,[B]

b)ADC A,#

c)ADC A,MD

Description: The contents of

a) the data memory location referenced by the **B** pointer

b) the immediate value found in the second byte of the instruction

c) the data memory location referenced by the second byte of the instruction

are added to the contents of the accumulator, and the result is simultaneously incremented if the Carry flag is found previously set. The result is placed back in the accumulator, and the Carry flag is either set or reset, depending on the presence or absence of a carry from the result. Similarly, the Half Carry flag is either set or reset, depending on the presence or absence of a carry from the low-order nibble.

Operation: A <- A + VALUE + C

C <- CARRY; HC <- HALF CARRY

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
ADC A,[B]	Register Indirect (B Pointer)	1	1	80
ADC A,#	Immediate	2	2	90/Imm #
ADC A,MD	Memory Direct	4	3	BD/MA/80

A.5.2 ADD — Add

Syntax: a)ADD A,[B]

b)ADD A,MD

c)ADD A,#

Description: The contents of the data memory location referenced by

a) the **B** pointer

b) the address in the second byte of the instruction

c) the immediate value found in the second byte of the instruction

are added to the contents of the accumulator, and the result is placed back in the accumulator. The Carry and Half Carry flags are not changed.

Operation: A <- A + VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
ADD A,[B]	Register Indirect (B Pointer)	1	1	84
ADD A,MD	Memory Direct	4	3	BD/MA/84
ADD A,#	Immediate	2	2	94/Imm.#

A.5.3 AND — And

Syntax: a)AND A,[B]

b)AND A,#

c)AND A,MD

Description: An AND operation is performed on corresponding bits of the accumulator and

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **AND** VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
AND A,[B]	Register Indirect (B Pointer)	1	1	85
AND A,#	Immediate	2	2	95/Imm.#
AND A,MD	Memory Direct	4	3	BD/MA/85

A.5.4 ANDSZ — And, Skip if Zero

Syntax: ANDSZ A,#

Description: An AND operation is performed on corresponding bits of the accumulator and the immediate value found in the second byte of the instruction. If the result is zero, the next instruction is skipped. The accumulator remains unchanged. This instruction may be used in testing for the presence of any selected bits in the accumulator. The mask in the second byte is used to select which bits are tested.

Operation: IF (A AND #) = 0, THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
ANDSZ A,#	Immediate	2	2	60/Imm.#

A.5.5 BRK — Software Breakpoint

Syntax: BRK

Description: The BRK instruction causes execution to begin at the address 0004 hex in the Boot ROM. The switch to Boot ROM is only successful if the COP8 is executing in ICE emulation mode. The instruction pushes the return address onto the software stack in data memory and then jumps to the address 0004 in Boot ROM. The breakpoint enable signal is also set internally. If the COP8 is not in emulation mode, the instruction jumps to the address 0004 in program memory, and the internal breakpoint signal is not set.

The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, PCL is loaded with 04 hex, and PCU is loaded with 0. The program then jumps to the program memory location accessed by PC in the Boot ROM, if the COP8 is in emulation mode, or in program memory if it is not.

Operation: [SP] <- PCL, Set BPI

[SP - 1] <- PCU

[SP - 2]: SET UP FOR NEXT STACK REFERENCE

PC14-8 <- 00

PC7-0 <- 04)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
BRK	Implicit	5	1	62

A.5.6 CLR — Clear Accumulator

Syntax: CLR A

Description: The accumulator is cleared to all zeros.

Operation: A <- 0

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
CLR A	Implicit	1	1	64

A.5.7 DCOR — Decimal Correct

Syntax: DCOR A

Description: This instruction when used following an ADC (add with carry) or SUBC (subtract with carry) instruction will decimal correct the result from the binary addition or subtraction. Note that the ADC instruction must be preceded with an ADD A, #066 (add hexadecimal 66) instruction for the decimal addition correction. This instruction assumes that the two operands are in BCD (Binary Coded Decimal) format and produces the result in the same BCD format. The Carry and Half Carry flags remain unchanged.

Operation: A (BCD FORMAT) <- A (BINARY FORMAT)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DCOR A	Implicit	1	1	66

A.5.8 DEC — Decrement Accumulator

Syntax: DECA

Description: This instruction decrements the contents of the accumulator and places the result back in the accumulator. The Carry and Half Carry flags remain unchanged.

Operation: A <- A - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DEC A	Implicit	1	1	8B

A.5.9 DRSZ REG# — Decrement Register and Skip if Result is Zero

Syntax: DRSZ REG#

Description: This instruction decrements the contents of the selected memory register (selected by #, where # = 0 to F) and places the result back in the same register. If the result is zero, the next instruction is skipped. This instruction is useful where it is desired to repeat an instruction sequence a given number of times. The desired number of times that the instruction sequence is to be executed is placed in a register, and a DRSZ instruction with that register is coded at the end of the sequence followed by a JP (Jump Relative) instruction that branches back to the start of the instruction sequence. The JP branch-back instruction is executed each time around the instruction sequence loop until the register count is decremented down to zero, at which time the JP instruction is skipped as the program branches (skips) out of the loop.

Operation: REG <- REG - 1

IF (REG - 1) = 0,

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DRSZ REG#	Register Direct (Implicit)	3	1	C (REG#)

A.5.10 IFBIT — Test Bit

Syntax: a)IFBIT #,[B]

b)IFBIT #,MD

c)IFBIT #,A

Description: The selected bit (# = 0 to 7, with 7 being high-order) from

a) the memory location reference by the **B** pointer is tested.

b) the memory location referenced by the address in the second byte of the instruction is tested.

c) the accumulator is tested.

If the selected bit is high (=1), then the next instruction is executed. Otherwise, the next instruction is skipped.

Operation: IF BIT (#) SELECTED

IS EQUAL TO 0,

THEN SKIP NEXT INSTRUCTION

Instruction	Address Mode	Instruction Cycle	Bytes	Hex Op Code
IFBIT #,[B]	Register Indirect (B Pointer)	1	1	7#
IFBIT #,MD	Memory Direct	4	3	BD/MA/7#
IFBIT #,A	Immediate	2	2	60/2#

NOTE: The IFBIT #,A is a special subset of the more generalized ANDSZ instruction and shares the same opcode of 60. This instruction disassembles into the ANDSZ instruction.

IFBIT 0,A equivalent to ANDSZ A,#1

IFBIT 1,A equivalent to ANDSZ A,#2

IFBIT 2,A equivalent to ANDSZ A,#4

IFBIT 3,A equivalent to ANDSZ A,#8

IFBIT 4,A equivalent to ANDSZ A,#16

IFBIT 5,A equivalent to ANDSZ A,#32

IFBIT 6,A equivalent to ANDSZ A,#64

IFBIT 7,A equivalent to ANDSZ A,#128

A.5.11 IFBNE # — If B Pointer Not Equal

Syntax: IFBNE #

Description: If the low-order nibble of the **B** pointer is not equal to # (where # = 0 to F), then the next instruction is executed. Otherwise, the next instruction is skipped. This instruction is useful where the **B** pointer is walked across a data field as part of a closed loop instruction sequence. The IFBNE instruction is coded at the end of the sequence followed by a JP (Jump Relative) instruction that branches back to the start of the instruction sequence. The # coded with the IFBNE represents the next address beyond the data field. The **B** pointer instruction with post-increment or decrement of the pointer may be used in walking across the data field in either direction. The instruction sequence branches back and repeats until the low-order nibble of the **B** pointer is found equal to the # (representing the next address beyond the data field), at which time the JP instruction is skipped as the program branches (skips) out of the loop.

Operation: IF **B** POINTER LOW-ORDER NIBBLE EQUALS #,

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFBNE #	Implicit	1	1	4#

A.5.12 IFC — Test if Carry

Syntax: IFC

Description: The next Instruction is executed if the Carry flag is found set. Otherwise, the next instruction is skipped. The Carry flag is left unchanged.

Operation: IF NO CARRY (C = 0),

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFC	Implicit	1	1	88

A.5.13 IFEQ — Test if Equal

Syntax: a)IFEQ A,[B]

b)IFEQ A,#

c)IFEQ A,MD

d)IFEQ MD,#

a) The contents of the data memory location referenced by the **B** pointer are compared for equality with the contents of the accumulator.

b) The immediate value found in the second byte of the instruction is compared for equality with the contents of the accumulator.

c) The contents of the data memory location referenced by the address in the second byte of the instruction are compared for equality with the contents of the accumulator.

d) The contents of the memory location referenced by the address in the second byte of the instruction are compared for equality with the immediate value found in the third byte of the instruction.

A successful equality comparison results in the execution of the next instruction. Otherwise, the next instruction is skipped.

Operation: IF CONTENTS OF SPECIFIED LOCATION ≠ VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFEQ A,[B]	Register Indirect (B Pointer)	1	1	82
IFEQ A,#	Immediate	2	2	92/Imm.#
IFEQ A,MD	Memory Direct	4	3	BD/MA/82
IFEQ MD,#	Memory Direct, Immediate	3	3	A9/MA/Imm.#

A.5.14 IFGT — Test if Greater Than

Syntax: a)IFGT A,[B]

b)IFGT A,#

c)IFGT A,MD

Description: The contents of the accumulator are tested for being greater than

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

A successful greater than test results in the execution of the next instruction. Otherwise, the next instruction is skipped.

Operation: IF A ≤ VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFGT A,[B]	Register Indirect (B Pointer)	1	1	83
IFGT A,#	Immediate	2	2	93/Imm.#
IFGT A,MD	Memory Direct	4	3	BD/MA/83

A.5.15 IFNC — Test if No Carry

Syntax: IFNC

Description: The next instruction is executed if the Carry flag is found reset. Otherwise, the next instruction is skipped. The Carry flag is left unchanged.

Operation: IF CARRY (C=1),

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFNC	Implicit	1	1	89

A.5.16 IFNE — Test If Not Equal

Syntax: a)IFNE A,[B]

b)IFNE A,#

c)IFNE A,MD

Description:

a) The contents of the data memory location referenced by the **B** pointer are compared for inequality with the contents of the accumulator.

b) The immediate value found in the second byte of the instruction is compared for inequality with the contents of the accumulator.

c) The contents of the data memory location referenced by the address in the second bye of the instruction are compared for inequality with the contents of the accumulator.

A successful inequality comparison results in the execution of the next instruction; otherwise, the next instruction is skipped.

Operation: IF A = VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFNE A,[B]	Register Indirect (B Pointer)	1	1	B9
IFNE A,#	Immediate	2	2	99/Imm.#
IFNE A,MD	Memory Direct	4	3	BD/MA/B9

A.5.17 INC — Increment Accumulator

Syntax: INC A

Description: This instruction increments the contents of the accumulator and places the result back in the accumulator. The Carry and Half Carry flags remain unchanged.

Operation: A <- A + 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
INC A	Implicit	1	1	8A

A.5.18 INTR — Interrupt (Software Trap)

Syntax: INTR

Description: This zero opcode software trap instruction first stores its return address in the data memory software stack and then branches to program memory location 00FF. This memory location is the common switching point for all COP800 interrupts, both hardware and software. The program starting at memory location 00FF sorts out the priority of the various interrupts and then vectors to the correct interrupt service routine.

In order to save the return address, the contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Up-

per 7 bits of PC) being transferred to the new data memory location referenced by SP. Then SP is again decremented to set up the software stack for the next interrupt or subroutine. The return address has now been saved on the software stack in data memory RAM.

The INTR instruction is not meant to be programmed explicitly, but rather to be automatically invoked when certain error conditions occur. The reading of undefined (non-existent) ROM program memory produces all zeros, which in turn invokes the INTR instruction. A similar software trap can be set up if the subroutine Stack Pointer (SP) is initialized to the data memory location at the top of user RAM space. Then if the software stack is ever overpopped (more subroutine or interrupt returns than calls), all ones will be returned from the undefined (non-existent) RAM. This will cause the program to return to the program address FFFF Hex, which in turn will read all zeros and once again invoke the software trap INTR instruction.

Operation: [SP] <- PCL
[SP - 1] <- PCU
[SP - 2] : SET UP FOR NEXT STACK REFERENCE
PC <- 0FF

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
INTR	Implicit	7	1	00

A.5.19 JID — Jump Indirect

Syntax: JID

Description: The JID instruction uses the contents of the accumulator to point to an indirect vector table of program addresses. The contents of the accumulator are transferred to PCL (Lower 8 bits of PC), after which the data accessed from the program memory location addressed by PC is transferred to PCL. The program then jumps to the program memory location accessed by PC. It should be observed that PCU (Upper 7 bits of PC) is never changed during the JID instruction, so that the Jump Indirect must jump to a location in the current program memory page of 256 addresses. However, if the JID instruction is located at the last address of the page, the PC counter will have already incremented over the page boundary, and both accesses to ROM program memory (vector table and the new instruction) will be fetched from the next page of 256 bytes.

Operation: PCL <- A

PCL <- ROM (PCU,A)JMP — Jump Absolute

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JID	Indirect	3	1	A5

Syntax: JMP ADDR

Description: This instruction jumps to the programmed memory address. The value found in the lower nibble (4 bits) of the first byte of the instruction is transferred to the lower nibble of PCU (Upper 7 bits of PC), and then the value found in the second byte of the instruction is transferred to PCL (Lower 8 bits of PC). The program then jumps to the program memory location accessed by PC.

It should be noted that the upper 3 bits of PC (12-14) are not changed, so the Jump Absolute instruction must jump to an address located in the current 4-Kbyte program memory segment. However, if a JMP instruction is programmed in the last address of the memory segment, the PC counter will have already incremented over the memory segment boundary; therefore, the jump is to a memory location in the following 4-Kbyte memory segment.

Operation: PC11-8 <- HIADDR (LOW NIBBLE OF FIRST BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JMP ADDR	Absolute	3	2	2HIADDR/LOADDR

A.5.20 JMPL — Jump Absolute Long

Syntax: JMPL ADDR

Description: The JMPL instruction allows branching to anywhere in the 32-Kbyte program memory space. The values found in the second and third bytes of the instruction are transferred to PCU (Upper 7 bits of PC) and PCL (Lower 8 bits of PC) respectively. The program then jumps to the program memory location accessed by PC.

Operation: PC14-8 <- HIADDR (SECOND BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (THIRD BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JMPL ADDR	Absolute	4	3	AC/HIADDR/LOADDR

A.5.21 JP — Jump Relative

Syntax: JP DISP

Description: The relative displacement value found in the instruction opcode (all 8 bits) is added to the Program Counter (PC). The normal PC incrementation is also performed. The displacement value allows a branch back from 0 to 31 places (with the 0 representing an infinite closed loop branch to itself) and a branch forward from 2 to 32 places. A branch forward of 1 is not allowed, since this zero opcode conflicts with the INTR software trap instruction.

Operation: PC <- PC + DISP + 1 (DISP 0)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JP DISP	Relative	3	1	0, 1, E, F + DISP #

A.5.22 JSR — Jump Subroutine

Syntax: JSR ADDR

Description: This instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address. The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address has now been saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the value found in the lower nibble (4 bits of the first byte of the instruction) is transferred to the lower nibble of PCU, and the value found in the second byte of the instruction is transferred to PCL. The

program then jumps to the memory location accessed by PC. It should be noted that the upper 3 bits of PC (12-14) are not changed, so the subroutine must be located in the current 4-Kbyte program memory segment. If a JSR instruction is programmed in the last address of the memory segment, however, the PC counter will have already incremented over the memory segment boundary; therefore, the subroutine must be located in the next memory segment.

Operation: [SP] <- PCL
[SP - 1] <- PCU
[SP - 2]: SET UP FOR NEXT STACK REFERENCE
PC11-8 <- HIADDR (LOW NIBBLE OF FIRST BYTE OF INSTRUCTION)
PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSR ADDR	Absolute	5	2	3HIADDR/LOADDR

A.5.23 JSRB — Jump Subroutine in Boot ROM

Syntax: JSRB ADDR

Description: The JSRB instruction causes execution to begin at the address specified within the first 256 bytes of the Boot ROM. The switch to Boot ROM is only successful if the JSRB instruction was immediately preceded by writing a valid key to the ISP KEY register. The instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address in Boot ROM. If the key has not been written, or if the key was invalid, the instruction jumps to the same address in program memory.

The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the values found in the second byte of the instruction is transferred to PCL. PCU is loaded with 0. The program then jumps to the program memory location accessed by PC in the Boot ROM, if the key write was successful, or in program memory if it was not.

Operation: [SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: SET UP FOR NEXT STACK REFERENCE

PC14-8 <- 00

PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSRB ADDR	Absolute	5	2	61/LOADDR

A.5.24 JSRL — Jump Subroutine Long

Syntax: JSRL ADDR

Description: The JSRL instruction allows the subroutine to be located anywhere in the 32-Kbyte program memory space. The instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address.

The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the values found in the second and third bytes of the instruction are transferred to PCU and PCL respectively. The program then jumps to the program memory location accessed by PC.

Operation: [SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: SET UP FOR NEXT STACK REFERENCE

PC14-8 <- HIADDR (SECOND BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (THIRD BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSRL ADDR	Absolute	5	3	AD/HIADDR/LOADDR

A.5.25 LAID — Load Accumulator Indirect

Address Mode: INDIRECT

Description: The LAID instruction uses the contents of the accumulator to point to a fixed data table stored in ROM program memory. The data table usually represents a translation matrix, such as the input from a keyboard or the output to a display.

The contents of the accumulator are exchanged with the contents of PCL (Lower 8 bits of PC). The data accessed from the program memory location addressed by PC is then transferred to the accumulator. Simultaneously, the original contents of PCL are transferred back to PCL from the accumulator. It should be observed that PCU (Upper 7 bits of PC) is not changed during the LAID instruction, so that the load accumulator indirect along with the associated fixed data table must both be located in the current memory page of 256 bytes. However, if the LAID instruction is located at the last address of the page, the PC counter will have already incremented over the page boundary resulting in the operand being fetched from the next page. Consequently, in this instance, the fixed data table must reside in the next page of 256 bytes in the program memory.

Operation: A <- ROM (PCU, A)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LAID	Indirect	3	1	A4

A.5.26 LD — Load Accumulator

Syntax: a)LD A,[B]

b)LD A,[B+]

c)LD A,[B-]

d)LD A,#

e)LD A,MD

f)LD A,[X]

g)LD A,[X+]

h)LD A,[X-]

Description:

a) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator.

b) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator, and then the **B** pointer is post-incremented.

c) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator, and then the **B** pointer is post-decremented.

d) The immediate value found in the second byte of the instruction is loaded into the accumulator.

e) The contents of the data memory location referenced by the address in the second byte of the instruction are loaded into the accumulator.

f) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator.

g) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator, and then the **X** pointer is post-incremented.

h) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator, and then the **X** pointer is post-decremented.

Operation: A <- VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD A,[B]	Register Indirect (B Pointer)	1	1	AE
LD A,[B+]	Register Indirect With Post-Incrementing B Pointer	2	1	AA
LD A,[B-]	Register Indirect With Post-Decrementing B Pointer	2	1	AB
LD A,#	Immediate	2	2	98/Imm.#
LD A,MD	Memory Direct	3	2	9D/MA
LD A,[X]	Register Indirect (X Pointer)	3	1	BE
LD A,[X+]	Register Indirect With Post-Incrementing X Pointer	3	1	BA
LD A,[X-]	Register Indirect With Post-Decrementing X Pointer	3	1	BB

A.5.27 LD — Load B Pointer

Syntax: a) LD B,# (# < 16)

b) LD B,# (# > 15)

Description:

a) The one's complement of the value found in the lower nibble (4 bits) of the instruction is transferred to the lower-nibble position of the **B** pointer register, with the upper-nibble position being cleared to all zeros.

b) The immediate value found in the second byte of the instruction is transferred to the **B** pointer register.

Operation:

a) B3-B0 <- # and B7-B4 <- 0

b) B <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD B,#	Short Immediate	1	1	5(15-#)
LD B,#	Immediate	2	2	9F/Imm.#

A.5.28 LD — Load Memory

Syntax: a)LD [B],#

b)LD [B+],#

c)LD [B-],#

d)LD MD,#

Description:

a) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer.

b) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer, and then the **B** pointer is post-incremented.

c) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer, and then the **B** pointer is post-decremented.

d) The immediate value found in the third byte of the instruction is loaded into the data memory location referenced by the address in the second byte of the instruction.

Operation: a)[B] <- #

b)[B] <- #; B <- B + 1

c)[B] <- #; B <- B - 1

d)MD <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD [B],#	Register Indirect/Immediate	2	2	9E/Imm.#
LD [B+],#	Register Indirect With Post-Incrementing/Immediate	2	2	9A/Imm.#
LD [B-],#	Register Indirect With Post-Decrementing/Immediate	2	2	9B/Imm.#
LD MD,#	Memory Direct/Immediate	3	3	BC/MA/Imm.#

A.5.29 LD — Load Register

Syntax: LD REG,#

Description: The immediate value found in the second byte of the instruction is loaded into the data memory register referenced by the low-order nibble of the first byte of the instruction.

Operation: REG <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD REG,#	Implicit/Immediate	3	2	D(REG#)/Imm.#

A.5.30 NOP — No Operation

Syntax: NOP

Description: No operation is performed by this instruction, so the net result is a delay of one instruction cycle time.

Operation: NO OPERATION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
NOP	Implicit	1	1	B8

A.5.31 OR — Or

Syntax: a)OR A,[B]

b)OR A,#

c)OR A,MD

Description: An OR operation is performed on corresponding bits of the accumulator with

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **OR** VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
OR A,[B]	Register Indirect (B Pointer)	1	1	87
OR A,#	Immediate	2	2	97/Imm.#
OR A,MD	Memory Direct	4	3	BD/MA/87

A.5.32 POP — Pop Stack

Syntax: POP A

Description: The Stack Pointer (SP) is incremented, and then the contents of the data memory location referenced by the SP are transferred to the accumulator.

Operation: SP <- SP + 1

A <- [SP]

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
POP	Implicit	3	1	8C

A.5.33 PUSH — Push Stack

Syntax: PUSH A

Description: The contents of the accumulator are transferred to the data memory location referenced by the Stack Pointer (SP), and then the SP is decremented

Operation: [SP] <- A

SP <- SP - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
PUSH	Implicit	3	1	67

A.5.34 RBIT — Reset Memory Bit

Syntax: a)RBIT #,[B]

b)RBIT #,MD

Description: The selected bit (# = 0 to 7, with 7 being high-order) of the data memory location referenced by the

a) **B** pointer is reset to 0.

b) address in the second byte of the instruction is reset to 0.

Operation: [Address:#] <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RBIT #,[B]	Register Indirect (B Pointer)	1	1	6(8 + #)
RBIT #,MD	Memory Direct	4	3	BD/MA/6(8+#)

A.5.35 RC — Reset Carry

Syntax: RC

Description: Both the Carry and Half Carry flags are reset to 0.

Operation: C <- 0

HC <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RC	Implicit	1	1	A0

A.5.36 RET — Return from Subroutine

Syntax: RET

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), after which SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program memory location accessed by PC.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RET	Implicit	5	1	8E

A.5.37 RETF — Return from Subroutine to Flash

Syntax: RETF

Description: The RETF instruction should never be called by code in program memory. It exists for use in the Boot ROM only. The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), after which SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program memory location accessed by PC, and execution continues out of program memory. If breakpoint mode is on, RETF clears it. If the RETF instruction is called by code in program memory, it will operate exactly like a RET instruction.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2]: SET UP FOR NEXT STACK REFERENCE. Clear BPI.

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETF	Implicit	5	1	63

A.5.38 RETI — Return from Interrupt

Syntax: RETI

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), and SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program

memory location accessed by PC. The Global Interrupt Enable flag (GIE) is set to 1.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

GIE <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETI	Implicit	5	1	8F

A.5.39 RETSK — Return and Skip

Syntax: RETSK

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), and SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to and then skips the instruction in the program memory location accessed by PC.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETSK	Implicit	5	1	8D

A.5.40 RLC — Rotate Accumulator Left Through Carry

Address Mode: RLC A

Description: The contents of the accumulator and Carry flag are rotated left one bit position, with the Carry flag serving as a ninth bit position linking the ends of the 8-bit accumulator. The previous carry is transferred to the low-order bit position of the accumulator. The high-order accumulator bit (A7) is transferred to the Carry flag. The A3 (high-order bit of the low-order nibble) of the accumulator is transferred into the Half Carry flag (HC) as well as into the A4 bit position.

Operation: C <- A7 <- A6 <- A5 <- A4 <- A3 <- A2 <- A1 <- A0 <- C

HC <- A3

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RLC A	Implicit	1	1	A8

A.5.41 RPND — Reset Pending

Syntax: RPND

Description: The RPND instruction resets the Non-Maskable Interrupt Pending flag (NMIPND) provided that the NMI interrupt has already been acknowledged and the Software Trap Pending flag was not found set. Also, RPND unconditionally resets the Software Trap Pending flag (STPND).

Operation: IF NMI interrupt acknowledged and STPND = 0

THEN NMPND <- 0 and STPND <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RPND	Implicit	1	1	B5

A.5.42 RRC — Rotate Accumulator Right Through Carry

Address Mode: RRC A

Description: The contents of the accumulator and Carry flag are rotated right one bit position, with the Carry flag serving as a ninth bit position linking the ends of the 8-bit accumulator. The previous carry is transferred to the

high-order bit position of the accumulator. The low-order accumulator bit (A0) is transferred to both the Carry flag and the Half Carry flag.

Operation: C -> A7 -> A6 -> A5 -> A4 -> A3 -> A2 -> A1 -> A0 -> C

A0 -> HC

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RRC A	Implicit	1	1	B0

A.5.43 SBIT — Set Memory Bit

Syntax: a)SBIT #,[B]

b)SBIT #,MD

Description: The selected bit (# = 0 to 7, with 7 being high-order) of the data memory location referenced by the

a) **B** pointer is set to 1.

b) address in the second byte of the instruction is set to 1.

Operation: [Address:#] <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SBIT #,[B]	Register Indirect (B Pointer)	1	1	7(8 + #)
SBIT #,MD	Memory Direct	4	3	BD/MA/7(8+#)

A.5.44 SC — Set Carry

Syntax: SC

Description: Both the Carry and Half Carry flags are set to 1.

Operation: C <- 1

HC <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SC	Implicit	1	1	A1

A.5.45 SUBC — Subtract with Carry

Syntax: a)SUBC A,[B]

b)SUBC A,#

c)SUBC A,MD

Description:

a) The contents of the data memory location referenced by the **B** pointer are subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

b) The immediate value found in the second byte of the instruction is subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

c) The contents of the data memory location referenced by the address in the second byte of the instruction are subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

The result is placed back in the accumulator, and the Carry flag is either reset or set, depending on the presence or absence of a borrow from the result. Similarly, the Half Carry flag is either reset or set, depending on the presence or absence of a borrow from the low-order nibble.

This instruction is implemented by adding the one's complement of the subtrahend to the accumulator and then incrementing the result. Consequently, the borrow is the equivalent of the absence of carry and vice versa. Similarly, the half carry is the equivalent of the absence of half borrow and vice versa. A previous borrow (absence of previous carry) will inhibit the incrementation of the result.

Operation: $A \leftarrow A - VALUE - \overline{C}$

C <- ABSENCE OF BYTE BORROW

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SUBC A,[B]	Register Indirect (B Pointer)	1	1	81
SUBC A,#	Immediate	2	2	91/Imm.#
SUBC A,MD	Memory Direct	4	3	BD/MA/81

A.5.46 SWAP — Swap Nibbles of Accumulator

Syntax: SWAP A

Description: The upper and lower nibbles of the accumulator are exchanged.

Operation: A(7-4) <--> A(3 - 0)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SWAP A	Implicit	1	1	65

A.5.47 VIS — Vector Interrupt Select

Syntax: VIS

Description: The purpose of the VIS instruction is to vector to the interrupt service routine for the interrupt with the highest priority and arbitration ranking that is currently enabled and requesting. The VIS instruction expedites this procedure of vectoring to the interrupt service routine.

All interrupts branch to program memory location 00FF Hex once an interrupt is acknowledged. Thus, any desired context switching (such as storing away the contents of the accumulator or B or X pointer) is normally programmed starting at location 00FF Hex, followed by the VIS instruction. The VIS instruction can be programmed at memory location 00FF Hex if no context switching is desired.

The VIS instruction first jumps to a double-byte vector in a 32-byte interrupt vector program memory table that is located at the top of a program memory block from address xyE0 to xyFF Hex. Note that xy is the block number (usually 01) where the VIS instruction is located (each block of program memory contains 256 bytes). This double-byte vector is transferred to PC (high-order byte first), and then the program jumps to the associated interrupt service routine indicated by the vector. These

interrupt service routines can be anywhere in the 32-Kbyte program memory space.

Should the VIS instruction be programmed at the top location of a memory block (such as address 00FF Hex), the associated 32-byte vector table is resident at the top of the next higher block (locations 01E0 to 01FF Hex with the VIS instruction at 00FF Hex).

Operation: PCL <- VA (Interrupt Arbitration Vector generated by hardware)

PCU <- ROM (PCU,VA)

PCL <- ROM (PCU,VA+1)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
VIS	Implicit	5	1	B4

A.5.48 X — Exchange Memory with Accumulator

Syntax: a)X A,[B]

b)X A,[B+]

c)X A,[B-]

d)X A,MD

e)X A,[X]

f)X A,[X+]

g)X A,[X-]

Description:

a) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator.

b) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator, and then the **B** pointer is post-incremented.

c) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator, and then the **B** pointer is post-decremented.

d) The contents of the data memory location referenced by the address in the second byte of the instruction are exchanged with the contents of the accumulator.

e) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator.

f) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator, and then the **X** pointer is post-incremented.

g) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator, and then the **X** pointer is post-decremented.

Operation: a)A <-> [B]

b)A <-> B; B <- B + 1

c)A <-> B; B <- B - 1

d)A <-> MD

e)A <-> X;

f)A <-> X; X <- X + 1

g)A <-> X; X <- X - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
X A,[B]	Register Indirect (B Pointer)	1	1	A6
X A,[B+]	Register Indirect With Post-Incrementing B Pointer	2	1	A2
X A,[B-]	Register Indirect With Post-Decrementing B Pointer	2	1	A3
X A,MD	Memory Direct	3	2	9C/MA
X A,[X]	Register Indirect (X Pointer)	3	1	B6
X A,[X+]	Register Indirect With Post-Incrementing X Pointer	3	1	B2
X A,[X-]	Register Indirect With Post-Decrementing X Pointer	3	1	B3

A.5.49 XOR — Exclusive Or

Syntax: a)XOR A,[B]

b)XOR A,#

c)XOR A,MD

Description: An XOR (Exclusive OR) operation is performed on corresponding bits of the accumulator with

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A XOR VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
XOR A,[B]	Register Indirect (B Pointer)	1	1	86
XOR A,#	Immediate	2	2	96/Imm.#
XOR A,MD	Memory Direct	4	3	BD/MA/86

A.6 Instruction Operations Summary

INSTR		FUNCTION	REGISTER OPERATION
ADD	A, MemI	Add	A <- A + MemI
ADC	A, MemI	Add with carry	A <- A + MemI + C, C <- Carry
SUBC	A, MemI	Subtract with carry	A <- A - MemI + C, C <- Carry
AND	A, MemI	Logical AND	A <- A and MemI
ANDSZ	A, Imm	Logical AND Imm, Skip if Zero	Skip next if (A and Imm) = 0
OR	A, MemI	Logical OR	A <- A or MemI
XOR	A, MemI	Logical Exclusive-OR	A <- A xor MemI
IFEQ	A, MemI	IF equal	Compare A and MemI, Do next if A = MemI
IFEQ	MD, Imm	IF equal	Compare MD and Imm, Do next if MD = Imm
IFNE	A, MemI	IF not equal	Compare A and MemI, Do next if A $\neq$ MemI
IFGT	A, MemI	IF greater than	Compare A and MemI, Do next if A > MemI
IFBNE	#	IF B not equal	Do next if lower 4 bits of B not = Imm
DRSZ	Reg	Decrement Reg, skip if zero	Reg <- Reg - 1, skip if Reg goes to zero
SBIT	#, Mem	Set bit	1 to Mem.bit (bit = 0 to 7 immediate)
RBIT	#, Mem	Reset bit	0 to Mem.bit (bit = 0 to 7 immediate)
IFBIT	#, Mem	If bit	If Mem.bit is true, do next instruction
RPND		Reset Pending Flag	Reset Software Interrupt Pending Flag
X	A, Mem	Exchange A with memory	A <-> Mem
LD	B, Imm	Load B with Immed.	B <- Imm
LD	A, MemI	Load A with memory	A <- MemI
LD	Mem, Imm	Load Direct memory Immed.	Mem <- Imm
LD	Reg, Imm	Load Register memory immed.	Reg <- Imm
X	A, [B$\pm$]	Exchange A with memory [B]	A <-> [B] (B <- B $\pm$ 1)
X	A, [X$\pm$]	Exchange A with memory [X]	A <-> [X] (X <- X $\pm$1)
LD	A, [B$\pm$]	Load A with memory [B]	A <- [B] (B <- B $\pm$1)
LD	A, [X$\pm$]	Load A with memory [X]	A <- [X] (X <- X $\pm$1)
LD	[B$\pm$], Imm	Load memory immediate	[B] <- Imm (B <- B $\pm$1)
CLRA		Clear A	A <- 0
INC	A	Increment A	A <- A + 1
DEC	A	Decrement A	A <- A - 1
LAID		Load A indirect from ROM	A <- ROM(PU, A)
DCOR	A	Decimal correct A	A <- BCD correction (follows ADC, SUBC)
RRC	A	Rotate right through carry	C -> A7 -> .... -> A0 -> C
RLC	A	Rotate left through carry	C <- A7 <- .... <- A0 <- C
SWAP	A	Swap nibbles of A	A7...A4 <-> A3...A0
SC		Set C	C <- 1
RC		Reset C	C <- 0
IFC		If C	If C is true, do next instruction
IFNC		If Not C	If C is not true, do next instruction
POP	A	Pop the Stack into A	SP <- SP + 1, A <- [SP}
PUSH	A	Push A onto the Stack	[SP] <- A, SP <- SP - 1
VIS		Vector to Interrupt Service	PCL <- [VL], PCU <- [VU]

INSTR		FUNCTION	REGISTER OPERATION
BRK		Breakpoint	SP] <- PL, [SP - 1] <- PU, SP - 2, PC <-0004 switch to ROM, set bkp
JMPL	Addr.	Jump absolute long	PC <- ii (ii = 15 bits, 0 to 32K)
JMP	Addr.	Jump absolute	PC11...PC0 <- i (i = 12 bits) PC15...PC12 remain unchanged
JP	Disp.	Jump relative short	PC <- PC + r (r is -31 to +32, not 1)
JSRL	Addr.	Jump subroutine long	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- ii
JSR	Addr.	Jump subroutine	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC11..PC0 <- ii
JSRB	Addr	Jump subroutine Boot ROM	[SP] <- PL, [SP - 1] <- PU, SP - 2, PL<-Addr.,PU<-00, switch to flash
JID		Jump indirect	PL <- ROM(PU, A)
RET		Return from subroutine	SP + 2, PL <- [SP], PU <- [SP - 1]
RETF		Return to flash	SP + 2, PL <- [SP], PU <- [SP - 1], switch to flash, clear bkp
RETSK		Return and skip	SP + 2, PL <- [SP], PU <- [SP - 1], Skip next instr.
RETI		Return from interrupt	SP + 2, PL <- [SP], PU <- [SP - 1], GIE <- 1
INTR		Generate an interrupt	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- 0FF
NOP		No operation	PC <- PC + 1

A.7 Instruction Bytes and Cycles

Table A-1 Instructions Using A and C

INSTR	BYTES/ CYCLES
CLRA	1/1
INCA	1/1
DECA	1/1
LAID	1/3
DCOR	1/1
RRCA	1/1
RLCA	1/1
SWAPA	1/1
SC	1/1
RC	1/1
IFC	1/1
IFNC	1/1
PUSHA	1/3
POPA	1/3
ANDSZ	2/2

Table A-2 Transfer of Control Instructions

INSTR	BYTES/ CYCLES
BRK	1/5
JMPL	3/4
JMP	2/3
JP	1/3
JSRL	3/5
JSR	2/5
JSRB	2/5
JID	1/3
VIS	1/5
RET	1/5
RETF	1/5
RETSK	1/5
RETI	1/5
INTR	1/7
NOP	1/1

Table A-3 Memory Transfer Instructions

INSTR	REGISTER INDIRECT		DIRECT	IMMEDIATE	REGISTER INDIRECT AUTO INCR & DECR	
	[B]	[X]			[B+, B-]	[X+, X-]
X A,[a]	1/1	1/3	2/3		1/2	1/3
LD A,[a]	1/1	1/3	2/3	2/2	1/2	1/3
LD B, Imm				1/1[b]		
LD B, Imm				2/2[c]		
LD Mem, Imm	2/2		3/3		2/2	
LD Reg, Imm			2/3			

a. Memory location addressed by B or X or directly
b. IF B < 16
c. IF B > 15

Table A-4 Arithmetic and Logic Instruction

INSTR	[B]	DIRECT	IMMEDIATE
ADD	1/1	3/4	2/2
ADC	1/1	3/4	2/2
SUBC	1/1	3/4	2/2
AND	1/1	3/4	2/2
OR	1/1	3/4	2/2
XOR	1/1	3/4	2/2
IFEQ	1/1	3/4	2/2
IFNE	1/1	3/4	2/2
IFGT	1/1	3/4	2/2
IFBNE	1/1		
DRSZ		1/3	
SBIT	1/1	3/4	
RBIT	1/1	3/4	
IFBIT	1/1	3/4	

RPND	1/1

Table A-5 Opcode Table

UPPER NIBBLE																
F	**E**	**D**	**C**	**B**	**A**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**LOWER NIBBLE**
JP-15	JP-31	LD 0F0,#i	DRSZ 0F0	RRCA	RC	ADC A,#i	ADC A,[B]	IFBIT 0,[B]	ANDSZ A,#i	LD B,#0F	IFBNE 0	JSR x000-x0FF	JMP x000-x0FF	JP+17	INTR	0
JP-14	JP-30	LD 0F1,#i	DRSZ 0F1	*	SC	SUBC A,#i	SUBC A,[B]	IFBIT 1,[B]	JSRB	LD B,#0E	IFBNE 1	JSR x100-x1FF	JMP x100-x1FF	JP+18	JP + 2	1
JP-13	JP-29	LD 0F2,#i	DRSZ 0F2	X A,[X+]	X A,[B+]	IFEQ A,#i	IFEQ A,[B]	IFBIT 2,[B]	BRK	LD B,#0D	IFBNE 2	JSR x200-x2FF	JMP x200-x2FF	JP+19	jp + 3	2
JP-12	JP-28	LD 0F3,#i	DRSZ 0F3	X A,[X-]	X A,[B-]	IFGT A,#i	IFGT A,[B]	IFBIT 3,[B]	RETF	LD B,#0C	IFBNE 3	JSR x300-x3FF	JMP x300-x3FF	JP+20	JP + 4	3
JP-11	JP-27	LD 0F4,#i	DRSZ 0F4	VIS	LAID	ADD A,#i	ADD A,[B]	IFBIT 4,[B]	CLRA	LD B,#0B	IFBNE 4	JSR x400-x4FF	JMP x400-x4FF	JP+21	JP = 5	4
JP-10	JP-26	LD 0F5,#i	DRSZ 0F5	RPND	JID	AND A,#i	AND A,[B]	IFBIT 5,[B]	SWAPA	LD B,#0A	IFBNE 5	JSR x500-x5FF	JMP x500-x5FF	JP+22	JP + 6	5
JP-9	JP-25	LD 0F6,#i	DRSZ 0F6	X A,[X]	X A,[B]	XOR A,#i	XOR A,[B]	IFBIT 6,[B]	DCORA	LD B,#09	IFBNE 6	JSR x600-x6FF	JMP x600-x6FF	JP+23	JP + 7	6
JP-8	JP-24	LD 0F7,#i	DRSZ 0F7	*	*	OR A,#i	OR A,[B]	IFBIT 7,[B]	PUSHA	LD B,#08	IFBNE 7	JSR x700-x7FF	JMP x700-x7FF	JP+24	JP + 8	7
JP-7	JP-23	LD 0F8,#i	DRSZ 0F8	NOP	RLCA	LD A,#i	IFC	SBIT 0,[B]	RBIT 0,[B]	LD B,#07	IFBNE 8	JSR x800-x8FF	JMP x800-x8FF	JP+25	JP + 9	8
JP-6	JP-22	LD 0F9,#i	DRSZ 0F9	IFNE A,[B]	IFEQ Md,#i	IFNE A,#i	IFNC	SBIT 1,[B]	RBIT 1,[B]	LD B,#06	IFBNE 9	JSR x900-x9FF	JMP x900-x9FF	JP+26	JP + 10	9
JP-5	JP-21	LD 0FA,#i	DRSZ 0FA	LD A,[X+]	LD A,[B+]	LD [B+],#i	INCA	SBIT 2,[B]	RBIT 2,[B]	LD B,#05	IFBNE 0A	JSR xA00-xAFF	JMP xA00-xAFF	JP+27	JP + 11	A
JP-4	JP-20	LD 0FB,#i	DRSZ 0FB	LD A,[X-]	LD A,[B-]	LD [B-],#i	DECA	SBIT 3,[B]	RBIT 3,[B]	LD B,#04	IFBNE 0B	JSR xB00-xBFF	JMP xB00-xBFF	JP+28	JP + 12	B
JP-3	JP-19	LD 0FC,#i	DRSZ 0FC	LD Md,#i	JMPL	X A,Md	POPA	SBIT 4,[B]	RBIT 4,[B]	LD B,#03	IFBNE 0C	JSR xC00-xCFF	JMP xC00-xCFF	JP+29	JP + 13	C
JP-2	JP-18	LD 0FD,#i	DRSZ 0FD	DIR	JSRL	LD A,Md	RETSK	SBIT 5,[B]	RBIT 5,[B]	LD B,#02	IFBNE 0D	JSR xD00-xDFF	JMP xD00-xDFF	JP+30	JP + 14	D
JP-1	JP-17	LD 0FE,#i	DRSZ 0FE	LD A,[X]	LD A,[B]	LD [B],#i	RET	SBIT 6,[B]	RBIT 6,[B]	LD B,#01	IFBNE 0E	JSR xE00-xEFF	JMP xE00-xEFF	JP+31	JP + 15	E
JP-0	JP-16	LD 0FF,#i	DRSZ 0FF	*	*	LD B,#i	RETI	SBIT 7,[B]	RBIT 7,[B]	LD B,#00	IFBNE 0F	JSR xF00-xFFF	JMP xF00-xFFF	JP+32	JP + 16	F

where,

i is the immediate data
Md is a directly addressed memory location
* is an unused opcode

The opcode 60 Hex is also the opcode for IFBIT #i,A

Appendix B

APPLICATION HINTS

B.1 INTRODUCTION

This chapter describes several application examples using the COP8 Feature Family of microcontrollers. Design examples often include block diagrams and/or assembly code. Certain hardware design considerations are also presented.

Topics covered in this chapter include the following:

- MICROWIRE/PLUS implementation examples
- Timer application examples
- Triac control example
- Analog to digital conversion technique
- Battery-powered weight measurement example
- Zero Cross Detection
- Industrial timer example
- Programming examples (clear RAM, binary arithmetic)
- External power wakeup circuit
- Watchdog Reset Circuit
- Input protection on COP8 pins
- Electromagnetic interference (EMI) considerations

B.2 MICROWIRE/PLUS INTERFACE

A large number of off-the-shelf devices are directly compatible with the MICROWIRE/PLUS interface. This allows direct interface of the COP8 Feature Family microcontrollers with a large number of peripheral devices. The following sections provide examples of the MICROWIRE/PLUS interface. These examples include a master/slave mode protocol, code for a continuous mode of operation, code for a fast burst mode of operation, and a COP888CL to an NMC93C06 interface.

B.2.1 MICROWIRE/PLUS Master/Slave Protocol

This section gives a sample MICROWIRE/PLUS master/slave protocol, the slave mode operating procedure for the sample protocol, and a timing illustration of the sample protocol.

Master/Slave Protocol:

1. CS from the master device is connected to G0 of the slave device. An active-low level on the CS line causes the slave to interrupt.
2. From the high-to-low transition on the CS line, there is no data transfer on the MICROWIRE interface until the setup time T has elapsed (see Figure B-1).
3. The master initiates data transfer on the MICROWIRE interface by turning on the SK clock.
4. A series of data transfers takes place between the master and slave devices.
5. The master pulls the CS line high to end the MICROWIRE operation. The slave device returns to normal mode of operation.

Slave Mode Operating Procedure (for the previous protocol):

1. Set the MSEL bit in the CNTRL register to enable MICROWIRE; G0 and G5 are configured as inputs and G4 as an output. Reset bit 6 of the Port G configuration register to select Standard SK Clocking mode.
2. Normal mode of operation until interrupted by CS going low.
3. Set the BUSY flag and load SIOR register with the data to be sent out on S0. (The shift register shifts eight bits of data from S0 at the high-order end of the shift register. Concurrently, eight new bits of data from SI are loaded into the low-order end of the shift register.)
4. Wait for the BUSY flag to be reset. (The BUSY flag automatically resets after 8 bits of data have been shifted.)
5. If data is being read in, the contents of the SIO register are saved.
6. The prearranged set of data transfers are performed.

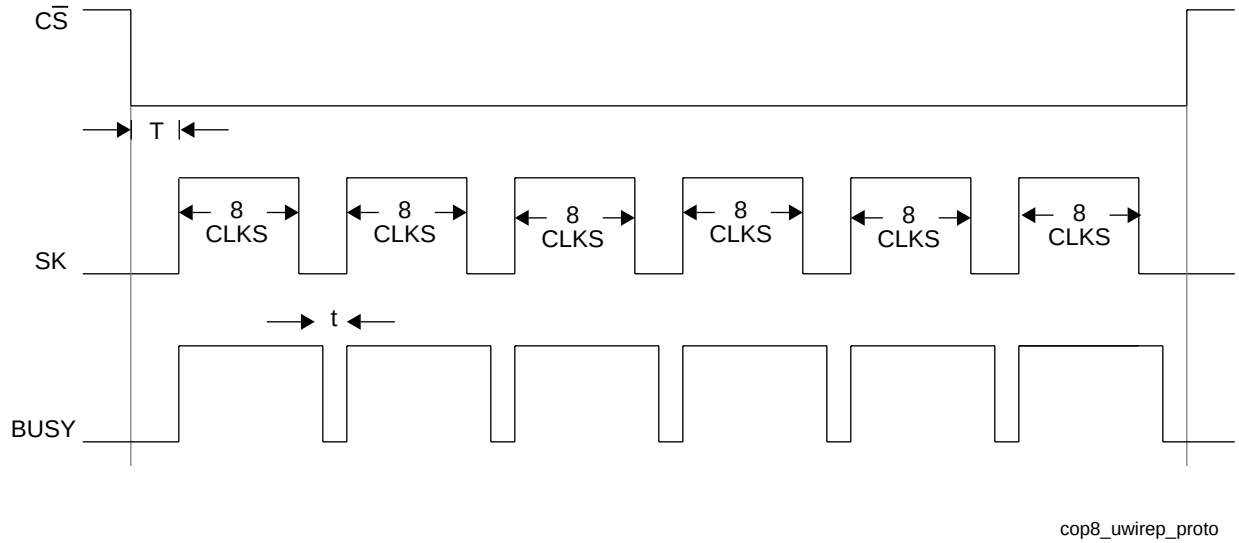


Figure B-1 MICROWIRE/PLUS Sample Protocol Timing

7. Repeat steps 3 through 6. The user must ensure step 3 is performed within *t-time* (refer to Figure B-1) as agreed upon in the protocol.

B.2.2 MICROWIRE/PLUS Continuous Mode

The MICROWIRE/PLUS interface can be used in continuous clock mode with the master mode divide-by-eight clock division factor selected. The maximum data transfer rate for this MICROWIRE/PLUS continuous clock mode is 64 microseconds per byte (equivalent to 125 KHz) for parts operating with a 1 µsec instruction cycle.

The continuous clock mode is achieved by resetting the BUSY bit under program control just before it would automatically be reset with the hardware, and then immediately setting the BUSY bit with the next instruction. The SIO MICROWIRE shift register is then loaded (or read) with the following instruction. This loading of SIO occurs before the SK clock goes high, even though the previous set BUSY bit instruction has started the divide-by-eight (3-stage) counter. The B pointer must be already set up to point at the PSW register where the MICROWIRE BUSY bit is located. This three-instruction sequence is programmed as follows:

Instruction			Bytes/Cycles
RBIT	BUSY,	[B]	1/1
SBIT	BUSY,	[B]	1/1
X	A,	SIOR	2/3

This three-instruction sequence must be embedded in an instruction program loop that is exactly 64 instruction cycles (t_C cycles) in length. This yields a 125-KHz (64 microseconds per byte) data transfer rate at the maximum instruction cycle rate of 1 MHz.

The following program demonstrates the use of the MICROWIRE/PLUS continuous clock mode. The program continually outputs the 256 bytes of the current program memory block on the MICROWIRE S0 output pin (G4). The low-order bit (G0) of Port G is set to cover the transition period of the three-instruction sequence outlined previously, where the SIO register is loaded with a new byte.

```
          .SECT   MW, REG
MWMEM:    .DSB 2
          MWTEMP = MWMEM
          MWCNT  = MWMEM+1

          MARK = 0
          BUSY = 2
          .SECT   CODE, ROM                    CYCLES
MWCONT:   LD      PORTGC, #031
          LD      PORTGD, #0
          LD      CNTRL, #0B
          LD      B, #PSW
          LD      MWCNT, #0
MWLOOP:   LD      A, MWCNT                      3
          INC     A                             1
          X       A, MWCNT                      3
          LAID                                  3
          SBIT    MARK, PORTGD                  4
          RBIT    BUSY, [B]                     1
          SBIT    BUSY, [B]                     1
          X       A, SIOR                       3
          RBIT    MARK, PORTGD                  4
          LD      MWTEMP, #6                    3
```

```
MWLUP:  DRSZ    MWTEMP          3 }
        JP      MWLUP           3 } x6 - 2 = 34
        NOP                               1
        JP      MWLOOP                    3

        TOTAL CYCLES IN MWLOOP =         64
```

B.2.3 MICROWIRE/PLUS Fast Burst Output

The maximum COP8 MICROWIRE/PLUS master mode burst clock rate (using the divide-by-two clock division factor) is 500 KHz. This assumes that the COP8 microcontroller is running at the maximum instruction cycle frequency of 1 MHz. The equivalent time of one extra master mode SK clock cycle is necessary to set up the next byte (and/or read the previous byte) in SIOR when using the burst mode SK frequency. This yields an equivalent minimum data transfer time of 18 microseconds per byte.

The following program demonstrates the use of the MICROWIRE/PLUS burst clock mode at the maximum data transfer rate (with the divide-by-two master mode clock option selected). The X pointer is initialized to the TOP of a RAM table, where SIZE represents the size of the table. This subroutine outputs the contents of the RAM table on the MICROWIRE S0 output pin (G4).

```
        .SECT   MW, REG
MWCNT:  .DSB 1

        BUSY = 2
        .SECT   MWFB, ROM          INSTRUCTION    CYCLE NUMBER
                                   CYCLES          IN Figure B-2
MWBRST: LD      CNTRL, #8
        LD      B, #PSW
        LD      MWCNT, #SIZE
        LD      X, #TOP
MWLOOP: LD      A, [X-]             3             13, 14, 15
        X       A, SIOR             3             16, 17, 18
        SBIT    BUSY, [B]           1                   1
        NOP*                        1                   2
        NOP*                        1                   3
        LAID*                       3              4, 5, 6
        DRSZ    MWCNT               3              7, 8, 9
        JP      MWLOOP              3             10, 11, 12
        RET

        TOTAL CYCLES IN MWLOOP =   18
```

* Time Delay

The MICROWIRE BUSY bit is allowed to reset automatically with the hardware following the eighth SK clock. The transfer of new data into the SIO register and the transfer of the new input data from SIO to A occurs at the end of the second cycle of the three-cycle “exchange A with SIO” instruction. This exchange instruction is immediately followed by the “set BUSY bit” instruction to initiate another MICROWIRE serial byte transfer. The associated timing for this 18-instruction cycle MICROWIRE loop is shown in Figure B-2.

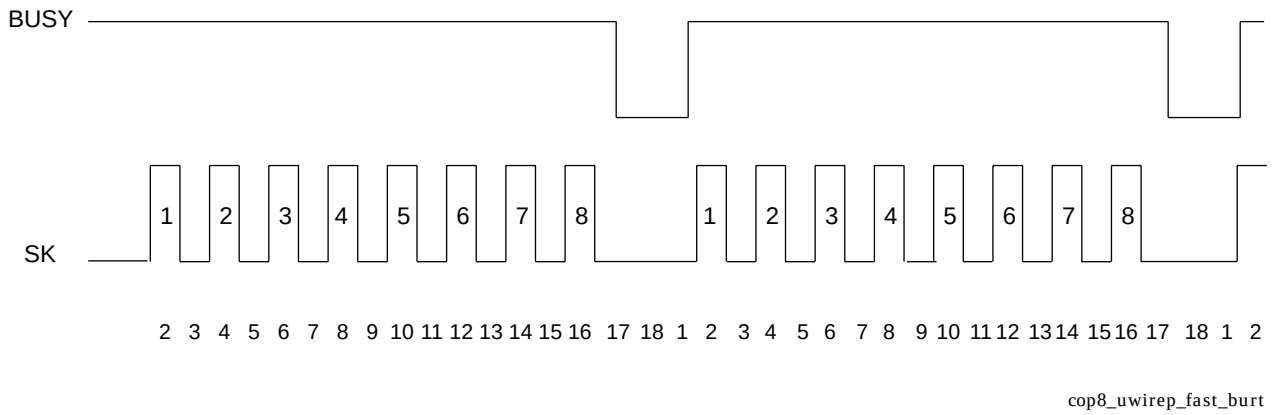


Figure B-2 MICROWIRE/PLUS Fast Burt Timing

B.2.4 NMC93C06-COP888CL Interface

This example shows the COP888CL interface to a NMC93C06, a 256-bit E^2PROM, using the MICROWIRE interface. The pin connection for interfacing a NMC93C06 with the COP888CL microcontroller is shown in Figure B-3. Some notes on the NMC93C06 interface requirements are:

1. The SK clock frequency should be less than 250 KHz. The SK clock should be configured for standard SK mode.
2. The CS low period following an Erase/Write instruction must not exceed 30 ms maximum. It should be set at the typical or minimum specification of 10 ms.

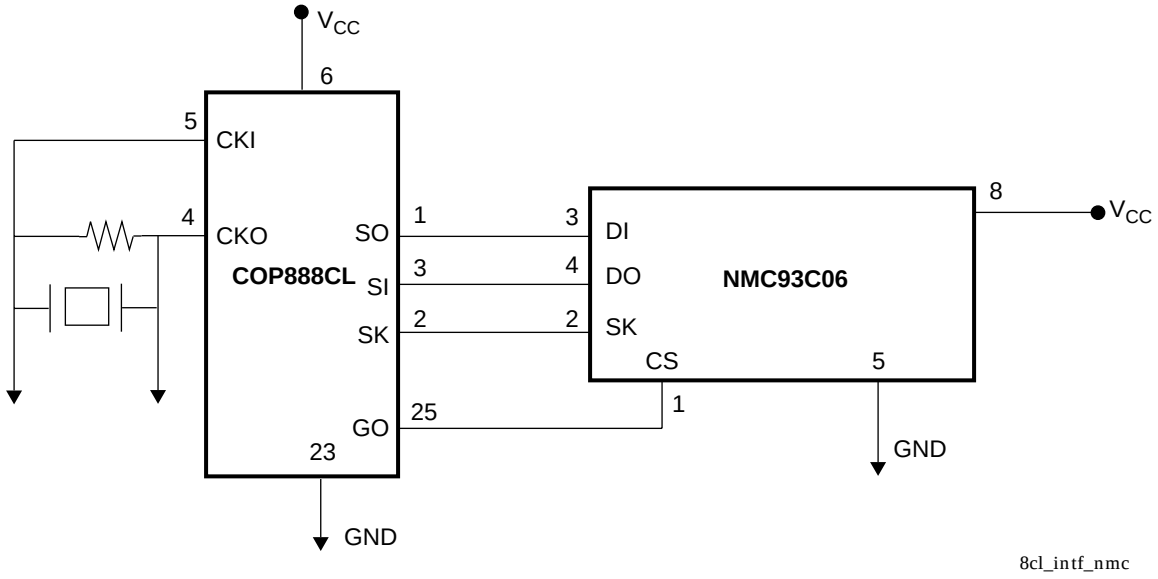


Figure B-3 NMC93C06-COP888CL Interface

3. The start bit on DI must be programmed as a “0” to “1” transition following a CS enable (“0” to “1”) when executing any instruction. One CS enable transition can execute only one instruction.

4. In the read mode, following an instruction and data train, the DI is a “don’t care” while the data is being output for the next 17 bits or clocks. The same is true for other instructions after the instruction and data has been fed in.

5. The data out train starts with a dummy bit 0 and is terminated by chip deselect. Any extra SK cycle after 16 bits is ignored. If CS is held on after all 16 of the data bits have been output, the DO will output the state of DI until another CS low to high transition starts a new instruction cycle.

6. After a read cycle, the CS must be brought low for one SK clock cycle before another instruction cycle starts.

The following table describes the instruction set of the NMC93C06. In the table A3A2A1A0 corresponds to one of the sixteen 16-bit registers.

Commands	Start Bit	Opcode	Address	Comments
READ	1	0000	A3A2A1A0	Read Register 0–15
WRITE	1	1000	A3A2A1A0	Write Register 0–15
ERASE	1	0100	A3A2A1A0	Erase Register 0–15
EWEN	1	1100	0001	Write/Erase Enable
EWDS	1	1100	0010	Write/Erase Disable
WRAL	1	1100	0100	Write All Registers
ERAL	1	1100	0101	Erase All Registers

All commands and data are shifted in/out on the rising edge of the SK clock. All instructions are initiated by a low-to-high transition on CS followed by a low-to-high transition on DI.

A detailed explanation of the NMC93C06 E^2PROM timing, instruction set, and other considerations can be found in the datasheet. A source listing of the software to interface the NMC93C06 with the COP888CL is provided below.

```
;This program provides in the form of subroutines, the ability to erase,
;enable, disable, read and write to the NMC93C06 EEPROM.

.INCLD   COP888.INC
         .SECT    NMC, RAM
NMCMEM:  .DSB 5
         SNDBUF = NMCMEM      ;Contains the command byte to be written NMC93C06
         RDATL  = NMCMEM+1    ;Lower byte of NMC93C06 register data read
         RDATH  = NMCMEM+2    ;Upper byte of NMC93C06 register data read
         WDATL  = NMCMEM+3    ;Lower byte of data to be written to NMC93C06 register
         WDATH  = NMCMEM+4    ;Upper byte of data to be written to NMC93C06 register
ADRESS:  .DSB 1               ;Lower 4-bits of this location contain the address of
                              ;the NMC93C06 register to read/write
FLAGS:   .DSB 1               ;Used for setting up flags
                              ;Flag valueAction
                              ;00Erase, enable, disable, erase all
                              ;01Read contents of NMC93C06 register
```

```
                              ;03Write to NMC93C06 register
                              ;OthersIllegal combination
        .SECT   CNT, REG
DLYH:   .DSB 1                ;Delay counter register
DLYL:   .DSB 1                ;Delay counter register

;The interface between the COP888 and the NMC93C06 (256-bit EEPROM) consists of four
;lines: The G0 (chip select line), G4 (serial out SO), G5 (serial clock SK), and
;G6 (serial in SI).
;
; Initialization
;
        .SECT   NMCODE, ROM
        LD      PORTGC,#031 ;Setup G0, G4, G5 as outputs and
                            ;select standard SK mode
        LD      PORTGD,#00  ;Initialize G data reg to zero
        LD      CNTRL,#08   ;Enable MSEL, select MW rate of 2tc
        LD      B,#PSW
        LD      X,#SIOR


;This routine erases the memory location pointed to by the address contained in the
;location “ADRESS.” The lower nibble of “ADRESS” contains the NMC93C06 register
;address and the upper nibble should be set to zero.
;
ERASE:  LD      A,ADRESS
        OR      A,#0C0
        X       A,SNDBUF
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine enables programming of the NMC93C06. Programming must be preceded
;once by a programming enable (EWEN).
;
EWEN:   LD      SNDBUF,#030
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine disables programming of the NMC93C06
;
EWDS:   LD      SNDBUF,#0
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine erases all registers of the NMC93C06
;
ERAL:   LD      SNDBUF,#020
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine reads the contents of a NMC93C06 register. The NMC93C06 address is
;specified in the lower nibble of location “ADRESS.” The upper nibble should be set
;to zero. The 16-bit contents of the NMC93C06 register are stored in RDATL and RDATH.
;
READ:   LD      A,ADRESS
        OR      A,#080
        X       A,SNDBUF
        LD      FLAGS,#1
        JSR     INIT
        RET
```

```
;This routine writes a 16-bit value stored in WDATL and WDATH to the NMC93C06 register
;whose address is contained in the lower nibble of the location “ADRESS.” The upper
;nibble of address location should be set to zero.
;
WRITE:  LD      A,ADRESS
        OR      A,#040
        X       A,SNDBUF
        LD      FLAGS,#3
        JSR     INIT
        RET

;This routine sends out the start bit and the command byte. It also deciphers the
;contents of the flag location and takes a decision regarding write, read or return
;to the calling routine.
;
INIT:   SBIT    0,PORTGD    ;Set chip select high
        LD      SIOR,#001   ;Load SIOR with start bit
        SBIT    BUSY,[B]    ;Send out the start bit
PUNT1:  IFBIT   BUSY,[B]
        JP      PUNT1
        LD      A,SNDBUF
        X       A,[X]       ;Load SIOR with command byte
        SBIT    BUSY,[B]    ;Send out command byte
PUNT2:  IFBIT   BUSY,[B]
        JP      PUNT2
        IFBIT   0,FLAGS     ;Any further processing?
        JP      NOTDON      ;Yes
        RBIT    0,PORTGD    ;No, reset CS and return
        RET
;
NOTDON: IFBIT   1,FLAGS     ;Read or write?
        JP      WR494       ;Jump to write routine
        LD      SIOR,#000   ;No, read NMC93C06
        SBIT    BUSY,PSW    ;Dummy clock to read zero
        RBIT    BUSY,[B]
        SBIT    BUSY,[B]
PUNT3:  IFBIT   BUSY,[B]
        JP      PUNT3
        X       A,[X]
        SBIT    BUSY,[B]
        X       A,RDATH
PUNT4:  IFBIT   BUSY,[B]
        JP      PUNT4
        LD      A,[X]
        X       A,RDATL
        RBIT    0,PORTGD
        RET
;
WR494:  LD      A,WDATH
        X       A,[X]
        SBIT    BUSY,[B]
PUNT5:  IFBIT   BUSY,[B]
        JP      PUNT5
        LD      A,WDATL
        X       A,[X]
        SBIT    BUSY,[B]
PUNT6:  IFBIT   BUSY,[B]
        JP      PUNT6
        RBIT    0,PORTGD
        JSR     TOUT
        RET
```

```
;Routine to generate delay for write
;
TOUT:   LD      DLYH,#00A
WAIT:   LD      DLYL,#0FF
WAIT1:  DRSZ    DLYL
        JP      WAIT1
        DRSZ    DLYH
        JP      WAIT
        RET
        .END
```

B.3 TIMER APPLICATIONS

This section describes two applications that use the 16-bit on-chip timer: speed measurement with the Input Capture mode, and an external event counter with the External Event Counter mode.

B.3.1 Timer Capture Example

The Timer Input Capture Mode can be used to measure the time between events. The simple block diagram in Figure B-4 shows how the COP8 can be used to measure motor speed based on the time required for one revolution of the wheel. A magnetic sensor is used to produce a pulse for each revolution of the wheel.

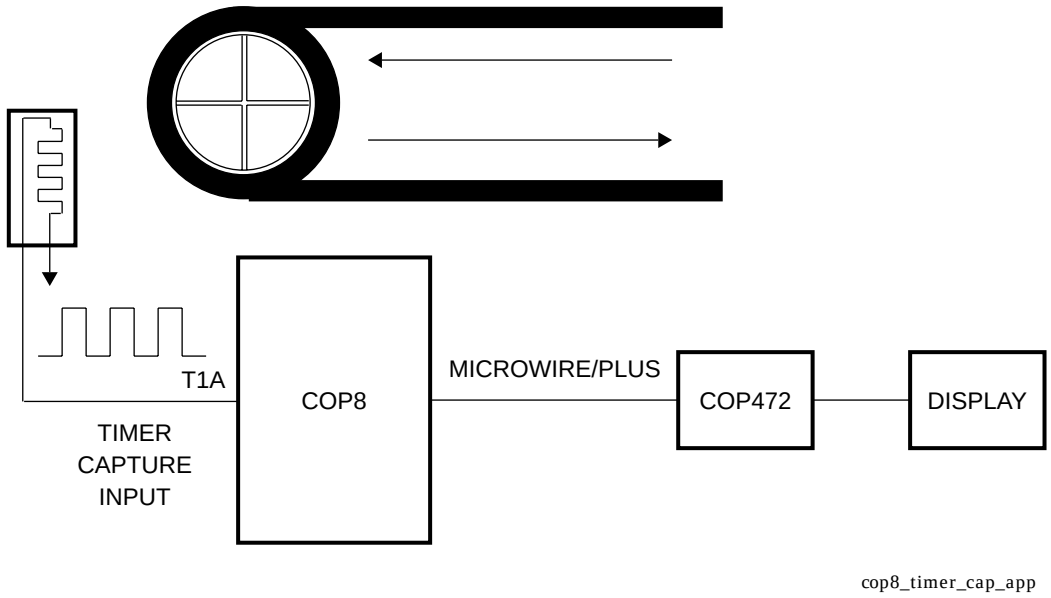


Figure B-4 Timer Capture Application

In the capture mode of operation, the timer counts down at the instruction cycle rate. In this application, the timer T1 is set up to generate an interrupt on a T1A positive edge transition. The timer is initialized to 0FFFF Hex and begins counting down. An edge transition on the T1A input pin of the timer causes the current timer value to be copied into the R1A register. In addition, it sets the timer interrupt pending flag, which causes a program branch to memory location 0FF Hex. The interrupt service routine for the timer is vectored to using the VIS instruction. The interrupt service routine resets the

T1PNDA pending flag. It then reads the contents of R1A and stores it in RAM for later processing. An RETI instruction is used to return to normal program execution and re-enable subsequent interrupts (by setting the GIE bit).

On the next rising edge transition on T1A, the program returns to the interrupt service routine. The value in R1A is read again, and compared with the previously read value. The difference between the two captured values, multiplied by the instruction cycle time, gives the time for one revolution. This is easily converted to a frequency. The frequency may be displayed on an LCD using the COP MICROWIRE/PLUS interface and a COP472-3 LCD Driver.

NOTE: The T1B input may be used simultaneously to measure the time between different events.

An example of the code that can be used for this application is provided below.

```
          .INCLD   COP888XX.INC
          .SECT    TMR, ROM
                                ;TIMER T1 CONFIGURATION
          LD       PORTGC,#00     ;Configure G3/T1A as input
          LD       PORTGD,#08     ;Weak pull-up on G3
          LD       TMR1L0,#0FF    ;Timer lower byte initialization
          LD       TMR1HI,#0FF    ;Timer upper byte initialization
          LD       ICNTRL,#00     ;Disable T1B interrupt
          LD       CNTRL,#0C0     ;Timer capture mode, positive edge on T1A
          LD       PSW,#011       ;Enable T1A and global interrupts
SELF:     JP       SELF           ;Wait for capture

          .SECT    INTERRUPT,ROM, ABS=0FF  ;TIMER INTERRUPT HANDLING ROUTINE
                                ;Set location counter to 00FF Hex
          VIS                   ;Vector to appropriate interrupt routine

                                ;TIMER SERVICE ROUTINE
T1SERV:   IFBIT    T1C0,CNTRL   ;If T1 underflow
          JP       UNDFLW       ;then handle timer underflow
          RBIT     T1PNDA,PSW   ;else Reset T1PNDA pending flag
          .                     ;(Process Timer Capture)
          .                     ;(Process Timer Capture)
          RETI                  ;Return from interrupt
UNDFLW:   RBIT     T1C0,CNTRL   ;Reset timer underflow pending flag
          .                     ;(Process Timer Underflow)
          RETI                  ;Return from interrupt

                                ;ERROR ROUTINE (NO INTERRUPT PENDING)
ERROR:    RETI                  ;Return from interrupt
          .SECT    INTTABLE, ROM, ABS=01E0  ;VECTOR TABLE
                                ;Set location counter to 01E0 Hex
.ADDRW    ERROR                 ;Store Error routine vector address
          ...                   ;{Store vector addresses 01E2 - 01F5 Hex}
                                ;Set location counter to 01F6 Hex
.ADDRW    T1SERV                ;Store T1PNDA routine vector address
          ...                   ;{Store vector addresses 01F8 - 01FF Hex}
.ENDSECT
```

B.3.2 External Event Counter Example

This mode of operation is very similar to the PWM Mode of operation. The only difference is that the timer is clocked from an external source. This mode provides the ability to perform control of a system based on counting a predetermined number of external

events, such as searching for the nth sector on a disk or testing every nth part on an assembly line. The code for this example is provided below.

```
          .INCLD    COP888XX.INC
          .SECT     EEC, ROM
                                    ;TIMER T1 CONFIGURATION
          RBIT      3,PORTGC        ;Configure G3/T1A as Hi-Z input
          RBIT      3,PORTGD        ;
          LD        CNTRL,#00       ;Select timer T1 as external event counter
          LD        ICNTRL,#00      ;Disable T1B interrupt
          LD        TMR1LO,#COUNT0  ;Timer T1 lower byte
          LD        TMR1HI,#COUNT1  ;Timer T1 upper byte
          LD        T1RALO,#Count0  ;Initialize Auto-reload R1A lower byte
          LD        T1RAHI,#Count1  ;Initialize Auto-reload R1A upper byte
          LD        T1RBLO,#Count0  ;Initialize Auto-reload R1B lower byte
          LD        T1RBHI,#Count1  ;Initialize Auto-reload R1B upper byte
          SBIT      T1C0,CNTRL      ;Start timer
          LD        PSW,#011        ;Enable timer T1A and global interrupts
SELF:     JP        SELF            ;Wait for the n-th count

          .SECT     INTERRUPT, ROM, ABS=0FF  ;TIMER INTERRUPT HANDLING ROUTINE
                                    ;Set location counter to 00FF Hex
          VIS                       ;Vector to appropriate interrupt routine

                                    ;TIMER SERVICE ROUTINE
T1SERV:   RBIT      T1PNDA,PSW      ;Reset T1A pending flag
          RBIT      T1C0,PSW        ;Stop timer
          .                         ;{Process timer interrupt}
          .                         ;{Process timer interrupt}
          SBIT      T1C0,PSW        ;Start timer
          RETI                      ;Return from interrupt

                                    ;ERROR ROUTINE (NO INTERRUPT PENDING)
Error:    RETI                      ;Return from interrupt

          .SECT     INTTABLE, ROM, ABS=01E0  ;VECTOR TABLE
                                    ;Set location counter to 01E0 Hex
.ADDRW    ERROR                     ;Store Error routine vector address
          ...                       ;{Store vector addresses 01E2 - 01F5 Hex}
                                    ;Set location counter to 01F6 Hex
.ADDRW    T1SERV                    ;Store T1PNDA routine vector address
          ...                       ;{Store vector addresses 01F8 - 01FF Hex}
.ENDSECT
```

B.4 TRIAC CONTROL

The COP8 Feature Family devices provide the computational ability and speed that is suitable for intelligently managing power control. In order to control a triac on a cyclic basis, an accurate time base must be established. This may be in the form of an AC 60Hz sync pulse generated by a zero voltage detection circuit or a simple real-time clock. The COP8 Feature Family is suited to accommodate either of these time base schemes while accomplishing other tasks.

Zero voltage detection is the most useful scheme in AC power control because it affords a real-time clock base as well as a reference point in the AC waveform. With this information it is possible to minimize RFI by initiating power-on operations near the AC line voltage zero crossing. It is also possible to fire the triac only a portion of the cycle, thus utilizing conduction angle manipulation. This is useful in both motor control and light-intensity control.

COP8 Feature Family software is capable of compensating for noisy or semi-accurate zero voltage detection circuits. This is accomplished by using delays and debounce algorithms in the software. With a given reference point in the AC waveform, it becomes easy to divide the waveform to efficiently allocate processing time.

These techniques are demonstrated in the following code listing. This application example is based on the half cycle approach of AC power for triac light intensity control. The code intensifies and deintensifies a lamp under program control.

This program example is not intended to be a final functional program. It is a general-purpose light intensifying/deintensifying routine which can be modified for a light dimmer application. The delay routines are based on a 10 MHz crystal clock (1 µs instruction cycle). The COP8's 16-bit timer can be used for timing the half cycle of an AC power line, and the timer can be started or stopped under software control. Timer T1 is a read/write memory mapped counter with two associated 16-bit auto-reload registers. In this example, only one reload register is used because the timer is stopped after each timer reload from R1A. Zero crossings of the 60 Hz line are detected and software debounced to initiate each half cycle, so the triac is serviced on every half cycle of the power line. This program divides the half cycle of a 60 Hz AC power line into 16 levels. Intensity is varied by increasing or decreasing the conduction angle by firing the triac at various levels. Each level is a fixed time which is loaded into the timer. Once a true zero cross is detected, the timer starts and the triac is serviced.

A level/sublevel approach is utilized to vary the conduction angle and to provide a prolonged intensifying period. The maximum intensity reached is at the maximum conduction angle (level), and the time required to get to that level is loaded into the timer in increments. Once a level has been specified, the remaining time in the half cycle is then divided into sublevels. The sublevels are increased in steps to the maximum level and the triac is fired 16 times per sublevel, thus creating the intensity time base. For deintensifying, the sublevels are decremented.

```
;This is a general purpose light dimmer program
;it uses a 10 MHz clock (1 us instruction cycle time)

        .INCLD COP888.INC
        .TITLE TIMER, 'TIMER APPLICATION EXAMPLE'

        .SECT   TRIAC, REG      ;INITIALIZATIONS
        TEMP:   .DSB 1          ;Temporary storage location
        LEVEL:  .DSB 1          ;Level storage location
        FIN:    .DSB 1          ;Fire number storage location
        REG1:   .DSB 1          ;Register1 definition
        .SECT   MAIN,ROM
        LD      FIN,#000        ;Set fire number to zero
        LD      LEVEL,#040      ;Set sublevel to 40 Hex
        LD      PORTGC,#000     ;Configure Port G as all inputs
        LD      PORTGD,#004     ;Weak-up on pin G2
        LD      CNTRL,#080      ;Configure Timer T1 in autoreload mode
        LD      PSW,#000        ;Disable all interrupts
        LD      TMR1LO,#07D     ;Initialize T1 and T1RA with 0.5mS delay
        LD      TMR1HI,#000     ;
        LD      T1RALO,#0EB     ;
        LD      T1RAHI,#003     ;

                                ;POWER UP SYNCHRONIZATION OR RESET SYNCH.
BEG:    IFBIT   2,PORTGP        ;If Bit G2 = 1
        JP      HI              ;then re-check bit
        JP      BEG             ;else keep looping to synch up 60Hz
HI:     IFBIT   2,PORTGP        ;If Bit G2 is still 1
```

```
        JP      HI              ;then wait until it is zero
                                ;else test for true zero cross

                                ;TEST FOR TRUE ZERO CROSS (Valid Transition)
                                ;Debounce for zero cross detection
        JSR     DELAY           ;When Bit G2 = 0, perform debounce delay
        IFBIT   2,PORTGP        ;If Bit G2 is high after the delay
        JP      BEG             ;then false alarm, go back to beginning
DOIT:   JMP     INIT            ;else go start program
                                ;Debounce 0 to 1
LO:     IFBIT   2,PORTGP        ;If Bit G2 is high
        JP      D1              ;then go perform debounce delay
        JP      LO              ;else loop back and wait for a 1
D1:     JSR     DELAY           ;Debounce delay (clean transition)
        IFBIT   2,PORTGP        ;If Bit G2 is still 1
        JP      DOIT            ;then go start program
        JP      LO              ;else false alarm, keep debouncing

                                ;MAIN ROUTINE FOR INTENSIFY/DE-INTENSIFY
                                ;A true zero cross has been detected
INIT:   JSR     TIMER           ;Delay for 1ms to get to MAX
        LD      A,FIN           ;Load accumulator with fire number
        IFEQ    A,#015          ;If the fire number equals 15
        JP      THER            ;then finished firing, continue on
BEGG:   INC     A               ;else increment fire number
        X       A,FIN           ;Save new fire number
        JP      FIRE            ;Keep firing
THER:   LD      FIN,#000        ;Reset fire number to zero
        LD      A,LEVEL         ;Load accumulator with sublevel number
        DEC     A               ;Decrement sublevel
        X       A,LEVEL         ;Save new sublevel
        LD      A,LEVEL         ;Load sublevel back into accumulator
        IFEQ    A,#000          ;If sublevel = MAX level
        JP      LP2             ;then go reset level
        JP      LP3             ;else go check level
LP2:    LD      LEVEL, #040     ;Reset level to 40 Hex
        JP      FIRE            ;Go fire (exit)
LP3:    IFBIT   5,LEVEL         ;If current level is greater than 1F Hex
        JSR     ADD             ;then MAX not yet reached, add delay
        JSR     SUB             ;else MAX has been reached, subtract delay
        NOP                     ;No-operation
        NOP                     ;No-operation

                                ;FIRE SUBROUTINE
FIRE:   LD      PORTD,#0FF      ;Set Port D HIGH for 32uSec
        X       A,TEMP          ;Save accumulator in temp location
        CLR     A               ;Clear accumulator
LP6:    INC     A               ;Increment accumulator
        IFEQ    A,#03           ;If accumulator equals three
        JP      LP5             ;then 32uSec done, continue on
        JP      LP6             ;else not done, keep looping
LP5:    CLR     A               ;Clear accumulator
        LD      PORTD,#00       ;Set Port D low
TWO:    X       A,TEMP          ;Restore accumulator
HI1:    IFBIT   2,PORTGP        ;If Bit G2 is high
        JMP     HI              ;then go debounce from High
        JMP     LO              ;else go debounce from Low

                                ;DELAY SUBROUTINE
DELAY:  LD      REG1,#00F       ;Load Reg1 with 0F Hex
LOOP:   DRSZ    REG1            ;Decr Reg1, If Reg1 not equal to 0
        JP      LOOP            ;then keep looping
        RET                     ;else return from delay routine

                                ;DECREMENT THE TIMER BY THE DESIRED DELAY
SUB:    LD      A, T1RALO       ;Load accumulator with value from T1RALO
        SUBC    A,#07D          ;Subtract 7D Hex
        X       A,T1RALO        ;Store result in T1RALO
        LD      A,T1RAHI        ;Load accumulator with value from T1RAHI
```

```
        SUBC    A,#000          ;Subtract zero and borrow (if occurred)
        RC                      ;Reset carry flag
        X       A,T1RAHI        ;Store result in T1RAHI
        RET                     ;Return from subtract routine

                                ;INCREMENT THE TIMER BY THE DESIREd DELAY
ADD:    LD      A, T1RALO       ;Load accumulator with value from T1RALO
        ADC     A,#07D          ;Add 7D Hex
        X       A,T1RALO        ;Store result in T1RALO
        LD      A,T1RAHI        ;Load accumulator with value from T1RAHI
        ADC     A,#000          ;Add zero and carry bit
        RC                      ;Reset carry flag
        X       A,T1RAHI        ;Store result in T1RAHI
        RETSK                   ;Return and skip from add routine

                                ;TIMER Subroutine
TIMER:  SBIT    T1C0,CNTRL      ;Start the timer
LP1:    IFBIT   T1PNDA,PSW      ;If underflow (reload from R1A) occurred
        JP      LP4             ;then go stop the timer
        JP      LP1             ;else keep looping
LP4:    RBIT    T1C0,CNTRL      ;Stop the timer
        RBIT    T1PNDA,PSW      ;Reset the T1 source A pending flag
        RET                     ;Return from timer subroutine
        .END                    ;end of program
```

B.5 ANALOG-TO-DIGITAL CONVERSION USING ON-CHIP COMPARATOR

Some microcontroller applications require a low-cost, but effective way of performing analog-to-digital conversion. A number of techniques for doing this are described in COP NOTE 1: “Analog to Digital Conversion Techniques with COPS Family Microcontrollers” and in Application Note 607: “Pulse Width Modulation A/D Conversion Techniques with COP800 Family Microcontrollers”. These notes may be found in the Embedded Controllers databook. This section explains how the COP8 feature family devices that contain an on-board comparator can be integrated into two of the solutions described in these notes: the single slope A/D conversion technique and the pulse width modulation A/D technique.

Figure B-5 shows the hardware connections for either type of A/D conversion technique. The voltage to be measured, V_{IN}, is connected to the inverting terminal, COMPIN-, of the comparator. The non-inverting terminal, COMPIN+, is connected to an RC network via a current-limiting resistor. For the single slope technique, the comparator output pin, COMPOUT is connected to the Timer T1A input pin. This is not required for the pulse width modulation technique.

The principle of the single slope conversion technique is to measure the time it takes for the RC network to charge up to the voltage level on the inverting terminal, by using Timer T1 in the input capture mode. The cycle count obtained in Timer T1 can be converted into real time if it is scaled by the COP8 clock frequency. If the COP8 is clocked by a crystal, this parameter is known very accurately. Applications connected to the power line using an RC clock can use the line frequency as a reference with which to measure the RC clock. The time measurement is then converted into the voltage, either by direct calculation or by using a suitable approximation.

This very low cost technique is ideally suited to cost-sensitive applications which do not require high accuracy. The pulse width modulation A/D conversion technique will improve the accuracy at the cost of a higher conversion time. Application Note 607 describes this technique in detail.

The accuracy can be improved further by using a low-cost MM74HC4016 to multiplex the analog input voltage with an accurate voltage reference used for calibration. Replacing the resistor in the RC network with a current source will linearize the charging curve, offering better resolution.

The user must ensure that the input voltage supplied to the comparator lies within its input common mode range, which is shown in the characterization curves in the datasheet.

Before the start of conversion, the capacitor must be discharged. The program reconfigures the COMPIN+ pin as an output logic low to perform the discharge. Timer T1 is stopped and configured for input capture mode on a low-to-high transition. The T1A register is cleared and pin T1A set up as a Hi-Z input. The comparator initialization is performed. The conversion begins when timer T1 is started and the COMPIN+ pin is configured back to an input.

The initial value of the comparator is zero. A capture event occurs when the RC voltage rises above the input voltage. If desired, the Timer T1 interrupt can be enabled to produce an interrupt on this capture event. The capture time can then be read and converted into voltage. This measurement technique has a resolution of 8 bits if the value of the timer is scaled to contain 1000 (or more) counts after five RC periods. The accuracy is primarily dependent on the accuracy of the user's estimation of the RC time constant, the offset voltage, and the user's approximation routine.

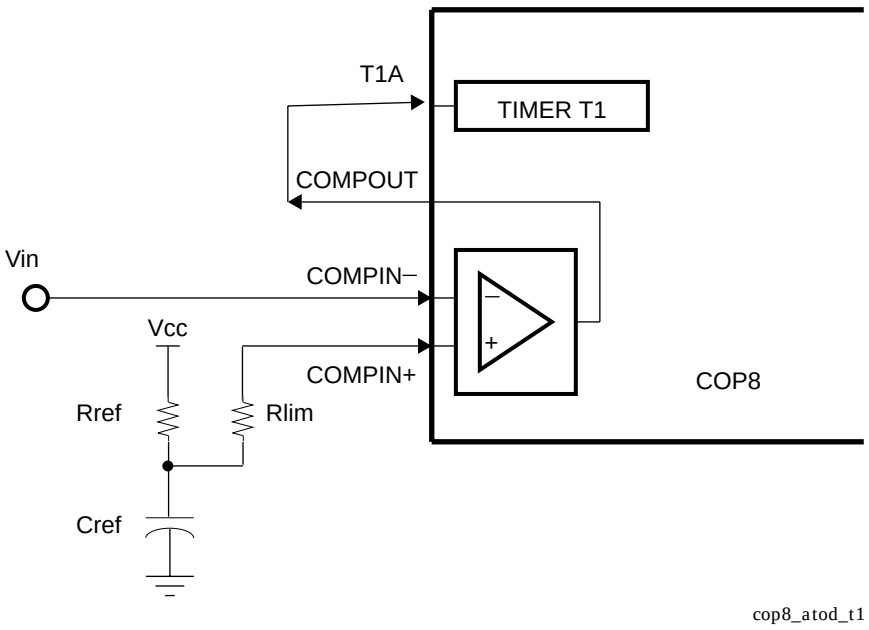


Figure B-5 A/D Conversion Using On-board Comparator and Timer T1

The following code example demonstrates how this technique is implemented in assembly code. In this example, polling the Timer T1A pending flag is used instead of interrupts. The 16-bit timer value is stored in REFHI:REFLO.

```
          ;PORTxC Addr of port config reg assigned comp alt. fun.
          ;PORTxD Addr of port data reg assigned comp alt. fun.
          .INCLD      COP888XX.INC
          .SECT       REGDEC,REG
R0:       .DSB 1
REF:      .DSB 2
          REFLO = REF
          REFHI = REF+1
          .SECT       CONVRT,ROM      ;DISCHARGE THE CAPACITOR
CONVRT:   SBIT        CMPINN,PORTxC   ;Configure COMPIN- as output
          RBIT        CMPINN,PORTxD   ;Set COMPIN- low
DELAY:                                ;Wait for capacitor to discharge
          LD          R0,#020         ;Load register with 20 Hex
DR0:      DRSZ        R0              ;Decrement register, if not equal to zero
          JP          DR0             ;then keep looping, else

                                      ;SET UP THE COMPARATOR AND CONFIGURE T1A
STCMP:    SBIT        CMPOUT,PORTxC   ;Configure COMPOUT pin as an output
          RBIT        CMPINP,PORTxC   ;Configure COMPIN+ pin as an input
          RBIT        CMPINN,PORTxC   ;Configure COMPIN- pin as an input
          LD          PORTxD,#00      ;Configure outputs low and inputs as Hi-Z
          RBIT        T1A,PORTGC      ;Configure T1A pin as an input
          RBIT        T1A,PORTGD      ;Configure T1A pin as Hi-Z

                                      ;PRELOAD TIMER T1
LDTIM:    LD          B,#T1MRLO       ;Load B pointer with T1 address
          LD          [B+],#0FF       ;Load T1 lower byte with FF Hex
          LD          [B],#0FF        ;Load T1 upper byte with FF Hex

                                      ;START TIMER T1 AND CHARGE THE CAPACITOR
STIM:     LD          PSW,#00         ;Disable all interrupts
          LD          CNTRL,#040      ;Configure T1 for input capture mode, positive
                                      T1A

                                      ;WAIT FOR CAPTURE AND SAVE VALUE
WAITC1:   IFBIT       T1PNDA,CNTRL    ;If capture occurred
          JP          STORE1          ;then go store capture value
          JP          WAITC1          ;else keep waiting

STORE1:   RBIT        T1PNDA,CNTRL    ;Reset T1PNDA flag
          LD          A,T1RALO        ;Read T1RA low byte
          X           A,REFLO         ;Save value in REFLO
          LD          A,T1RAHI        ;Read T1RA high byte
          X           A,REFHI         ;Save value in REFHI

                                      ;WAIT FOR 2nd CAPTURE AND COMPUTE ELAPSED TIME
WAITC2:   IFBIT       T1PNDA,CNTRL    ;If capture occurred
          JP          COMP            ;then go compute elapsed time
          JP          WAITC2          ;else keep waiting
COMP:                                 ;Compute elapsed time
.ENDSECT                              ;End of example
```

B.6 BATTERY-POWERED WEIGHT MEASUREMENT

Figure B-6 shows the block diagram of a simple weight scale application. This implementation of weight measurement may be used with any COP888 device that has the Multi-input Wakeup feature. The pressure sensor circuit is based on a buffered Wheatstone bridge arrangement. A current source and a capacitor generate the linear ramp for the A/D conversion. A crystal oscillator is required for an accurate time base. A

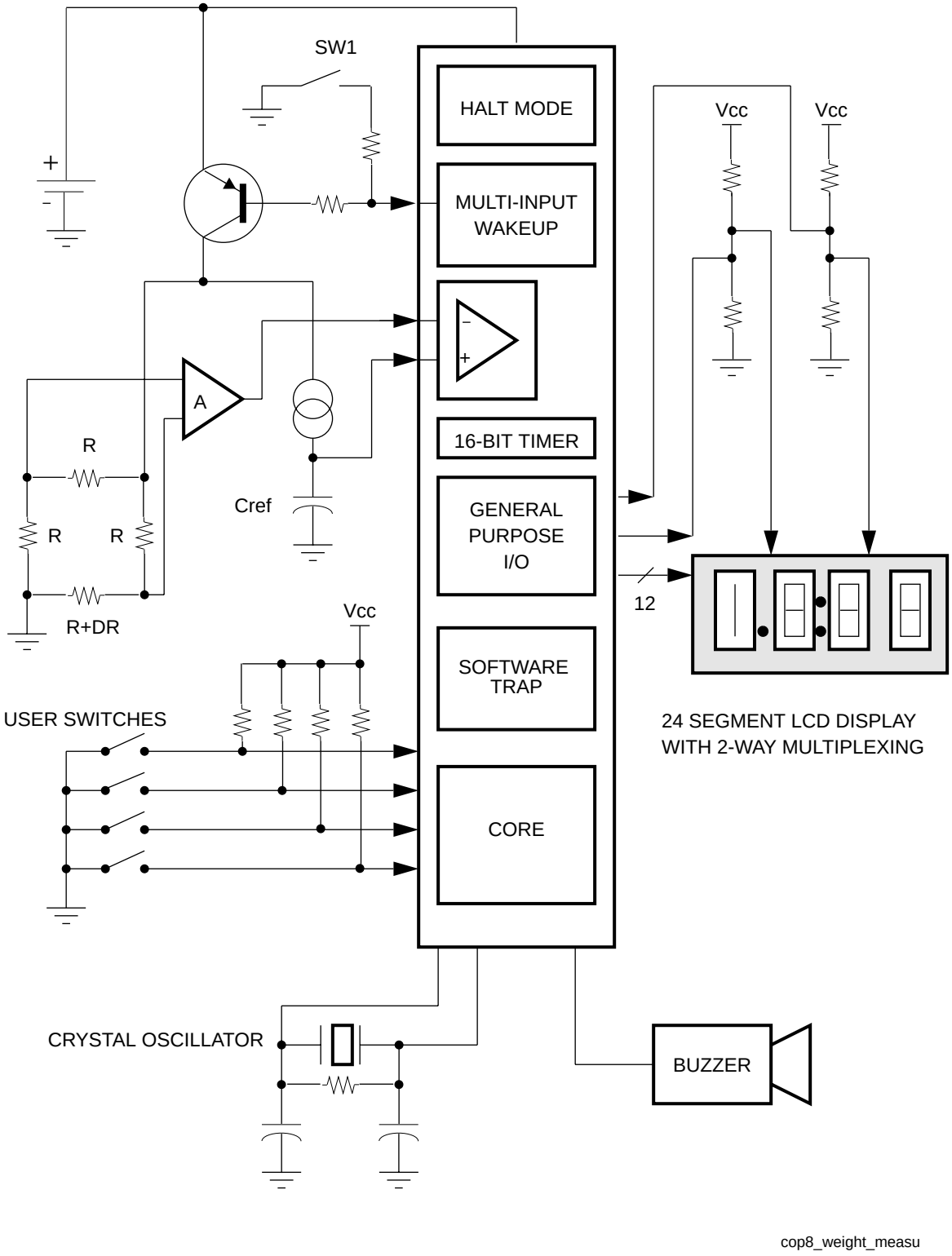


Figure B-6 Battery-powered Weight Measurement

50% duty cycle signal is generated for the audible tone. A 24-segment LCD display indicates the weight to the user. Four inputs are used for configuring the scale.

If the application is not in use, the COP888 is held in HALT mode. As soon as a weight is applied to the system, SW1 closes and the COP exits the HALT mode via a Multi-Input Wakeup pin. The MIWU pin is then reconfigured as an output to power up the sensor circuit, thus power remains even when the switch is open. The measurement and display are then performed. After completing the measurement and display routines, the COP888 reconfigures the sensor power pin as a Wakeup pin, thereby disconnecting power from the sensor circuit. The device then re-enters the HALT mode.

The 16-bit timer can be used to generate the interrupts required to refresh the LCD display. A power-on reset circuit (not shown) is required in this application.

B.7 ZERO CROSS DETECTION

Zero cross detection is often used in appliances connected to the AC power line. The line frequency is a useful time base for applications such as industrial timers or irons which switch off if not used for five minutes. Phase-controlled applications require a consistent timing reference in phase with the line voltage.

The COP888 requires a square wave, magnitude V_{CC}, at the same frequency as the power line voltage, connected to a input port pin for a simple time base. For a phase-control time base, this waveform should preferably be in phase with the line voltage, although control is still possible if there is a predictable, constant phase lag, less than the phase lag introduced by the load. The choice of zero cross detection circuit depends on factors such as cost, the type of power supply used in the appliance, and the expected interference.

The zero cross detection input can either be polled by software or can be connected to the G0 interrupt line. Polling the pin by software is the simplest technique and saves the interrupt for another function, but has the disadvantage that the polling procedure can be interrupted, causing inaccuracies in synchronization. Disabling the interrupt during the polling is not always possible, as the interrupt may be required for the implementation of other features.

Connecting the zero cross detection input to the external interrupt pin guarantees synchronization. It has the additional advantage that a regular interrupt is generated, which could interrupt the processor out of a fault condition. The interrupt routine only needs to test the integrity of the stack to determine whether such a fault has occurred.

The following software example shows how software polling of the zero cross line is implemented.

```
ZCD:
        LD      B,#STATUS       ;Save bytes using the B pointer
        IFBIT   SYNCHRO,[B]     ;If SYNCHRO is 1, wait for a rising edge
        JP      WLOHI           ;otherwise wait for a falling edge.
WHILO:  IFBIT   3,PORTLP        ;Wait for falling edge
        JP      WHILO
        SBIT    SYNCHRO,[B]     ;SYNCHRO = 1, so wait for rising edge
        JP      ENDZCD          ;next time.
WLOHI:  IFBIT   3,PORTLP        ;Wait for a rising edge
        JP      RSYNC
        JP      WLOHI
RSYNC:  RBIT    SYNCHRO,[B]     ;SYNCHRO = 0, so wait for a falling edge
                                ;next time.
ENDZCD:                         ;End of example
```

B.8 INDUSTRIAL TIMER

Figure B-7 shows the block diagram for an industrial timer. The user turns the potentiometer to set the required delay time. When the delay time has elapsed, a load is switched on or off, as selected by the input switches. The time base is derived from the power line using a simple zero cross detection circuit, thereby allowing the use of an inexpensive RC clock instead of a crystal oscillator. There are two indicator diodes and a buzzer.

The A/D conversion routine used by this industrial timer is based on the single slope technique defined in Section B.5, but it has an important difference. Instead of connecting the variable resistor into a voltage divider circuit and measuring the voltage using the single slope technique, the variable resistor forms part of the RC network. The time that the variable RC circuit takes to exceed the fixed reference voltage is directly proportional to the value of the resistor, simplifying the conversion from time into resistance. The circuit as shown can be used to program a time proportional to the angle of the potentiometer setting. The potentiometer can be replaced by a rotary switch connected to a series of resistors, so that each position of the switch generates a different resistance. Here the COP888 can identify the switch positions if the difference in each resistance for each position is greater than the inaccuracy in measuring the absolute resistance.

B.9 PROGRAMMING EXAMPLES

For more detailed and varied programming examples, refer to the Microcontroller Databook or call the Customer Response Center at 1-800-272-9959.

Programming examples can also be obtained from our web site at:
www.national.com/cop8

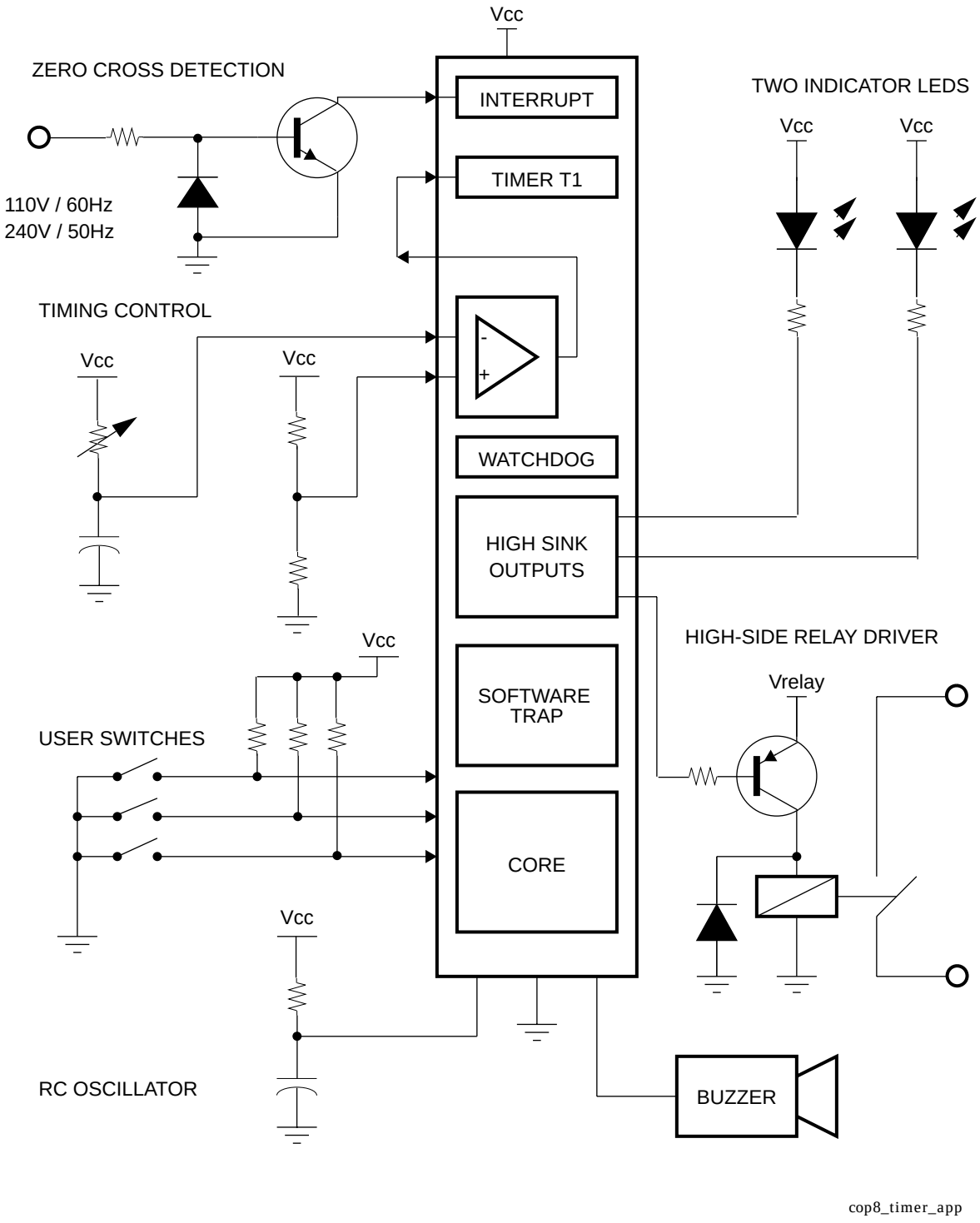


Figure B-7 Industrial Timer Application

B.9.1 Clear RAM

The following program clears all RAM locations in the base segment except for the stack pointer. The value of the argument to IFBNE may need to be adjusted, depending on the size of RAM in specific family members.

```
COP888 PROGRAM TO CLEAR ALL RAM EXCEPT SP


CLRAM:   LD       0FC,#070    ;Define X-pointer as counter
         LD       B,#0        :Initialize B pointer
CLRAM2:  LD       [B+],#0     ;Load mem with 0 and incr B pointer
         DRSZ     0FC         ;Decrement counter
         JP       CLRAM2      ;Skip if lower half RAM is cleared
         LD       B,#0F0      ;Point B to upper half of RAM
CLRAM3:  LD       [B+],#0     ;Load upper RAM half with 0
         IFBNE    #0D         ;until B points to 0FD (=SP)
         JP       CLRAM3      ;Skip if B=0FD
         LD       B,#0        ;Initialize B to 0
```

B.9.2 Binary/BCD Arithmetic Operations

The arithmetic instructions include the Add (ADD), Add with Carry (ADC), Subtract with Carry (SUBC), Increment (INC), Decrement (DEC), Decimal Correct (DCOR), Clear Accumulator (CLR), Set Carry (SC), and Reset Carry (RC). The shift and rotate instructions, which include the Rotate Right through Carry (RRC), the Rotate Left through Carry (RLC), and the Swap accumulator nibbles (SWAP), may also be considered arithmetic instruction variations. The RRC instruction is instrumental in writing a fast multiply routine.

In subtraction, a borrow is represented by the absence of a Carry and vice versa. Consequently, the Carry flag needs to be set (no borrow) before a subtraction, just as the Carry flag is reset (no carry) before an addition. The ADD instruction does not use the Carry flag as an input. It should also be noted that both the Carry and Half Carry flags (Bits 6 and 7, respectively, of the PSW control register) are cleared with RESET and remain unchanged with the ADD, INC, DEC, DCOR, CLR, and SWAP instructions. The DCOR instruction uses both the Carry and Half Carry flags. The SC instruction sets both the Carry and Half Carry flags, while the RC instruction resets both these flags.

The following program examples illustrate additions and subtractions of 4-byte data fields in both binary and BCD (Binary Coded Decimal). The four bytes from data memory locations 24 through 27 are added to or subtracted from the four bytes in data memory locations 16 through 19. The results replace the data in memory locations 24 through 27.

These operations are performed both in binary and BCD. It should be noted that the BCD preconditioning of adding (ADD) the hexadecimal value 66 is necessary only with the BCD addition, not with the BCD subtraction. The binary coded decimal DCOR (Decimal Correct) instruction uses both the Cary and Half Carry flags as inputs but does not change the Carry and Half Carry flags. Also note that the #12 with the IFBNE

instruction represents 28 minus 16, since the IFBNE operand is modulo 16 (remainder when divided by 16).

```
BINARY ADDITION
        LD      X,#16   ;No leading zero indicates decimal
        LD      B,#24
        RC
LOOP:   LD      A,[X+]
        ADC     A,[B]
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFC
        JP      OVFLOW  ;OverFLow if C

BINARY SUBTRACTION
        LD      X,#010  ;Leading zero indicates hex
        LD      B,#018
        SC
LOOP:   LD      A,[X+]
        SUBC    A,[B]
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFNC
        JP      NEGRSLT ;Neg. result if no C (No C = Borrow)

BCD ADDITION
        LD      X,#010  ;Leading zero indicates hex
        LD      B,#018
        RC
LOOP:   LD      A,[X+]
        ADD     A,#066  ;Add hex 66
        ADC     A,[B]
        DCOR    A       ;Decimal correct
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFC
        JP      OVFLOW  ;Overflow if C

BCD SUBTRACTION
        LD      X,#16   ;No leading zero indicates decimal
        LD      B,#24
        SC
LOOP:   LD      A,[X+]
        SUBC    A,[B]
        DCOR    A       ;Decimal correct
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFNC
        JP      NEGRSLT ;Neg. result if no C (No C = Borrow)
```

Note that the previous additions and subtractions are not “adding machine” type arithmetic operations in that the result replaces the second operand rather that the first. The following program examples illustrate “adding machine” type operations where the result replaces the first operand. With subtraction, this entails the result replacing the minuend rather that the subtrahend.

```
BINARY ADDITION
        LD      B,#16
        LD      X,#24
        RC
LOOP:   LD      A,[X+]
        ADC     A,[B]
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFC
        JP      OVFLOW   ;Overflow if C

BINARY SUBTRACTION
        LD      B,#010
        LD      X,#018
        SC
LOOP:   LD      A,[X+]
        X       A,[B]
        SUBC    A,[B]
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFNC
        JP      NEGRSLT  ;Neg. result if no C (No C = Borrow)

BCD ADDITION
        LD      B,#010
        LD      X,#018
        RC
LOOP:   LD      A,[X+]
        ADD     A,#066
        ADC     A,[B]
        DCOR    A
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFC
        JP      OVFLOW   ;Overflow if C

BCD SUBTRACTION
        LD      B,#16
        LD      X,#24
        SC
LOOP:   LD      A,[X+]
        X       A,[B]
        SUBC    A,[B]
        DCOR    A
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFNC
        JP      NEGRSLT  ;Neg. result if no C (No C = Borrow)
```

The following hybrid arithmetic example adds five successive bytes of a data table in program memory to a two-byte SUM, and then subtracts the SUM from a two-byte total TOT. Assume that the table is located starting a program memory address 0401, while SUM and TOT are at RAM data memory locations 1, 0 and 3, 2, respectively. The program is encoded as a subroutine.

```
          .SECT    MATH,RAM
MATHMEM:  .DSB4                ;CONSTANT DECLARATIONS
          SUMLO = MATHMEM      ;Sum lower byte storage location
          SUMHI = MATHMEM+1    ;Sum upper byte storage location
          TOTLO = MATHMEM+2    ;Total lower byte storage location
          TOTHI = MATHMEM+3    ;Total upper byte storage location
          .SECT    CODE, ROM, ABS=0401
                               ;ROM TABLE
          .BYTE    102         ;Store 102
          .BYTE    41          ;Store 41
          .BYTE    31          ;Store 31
          .BYTE    26          ;Store 26
          .BYTE    5           ;Store 5
                               ;PERFORM ADDITION AND SUBTRACTION
ARITH1:   LD       X,#5        ;Set up ROM table pointer
          LD       B,#SUMLO    ;Set up sum pointer
LOOP:     RC                   ;Reset carry flag
          LD       A,X         ;Load ROM pointer into accumulator
          LAID                 ;Read data from ROM
          ADC      A,[B]       ;Add SUMLO to ROM value
          X        A,[B+]      ;Store result in SUMLO, point to SUMHI
          CLR      A           ;Clear accumulator
          ADC      A,[B]       ;Add SUMHI and carry bit to the accumulator
          X        A,[B-]      ;Store result in SUMHI, point to SUMLO
          DRSZ     X           ;Decrement ROM pointer, If not equal to zero
          JP       LOOP        ;then repeat the loop
          SC                   ;else set the carry flag
          LD       B,#2        ;Load B pointer with 2 (point to TOTLO)
LUP:      LD       A,[X+]      ;Load accumulator with subtrahend
          X        A,[B]       ;Reverse operands for subtraction
          SUBC     A,[B]       ;Subtract
          X        A,[B+]      ;Increment minuend pointer
          IFBNE    #4          ;If B pointer not equal to 4
          JP       LUP         ;then repeat the loop
          RET                  ;else return
```

B.9.3 Binary Multiplication

The following program listing shows the code for a 16-by-16-bit binary multiply subroutine. The multiplier starts in the lower 16 bits of the 32-bit result location. As the multiplier is shifted out of the low end of the result location with the RRC instruction, each multiplier bit is tested in the Carry flag. The multiplicand is conditionally added (depending on the multiplier bit) into the high end of the result location, after which the partial product is shifted down one bit position following the multiplier. Note that one additional terminal shift cycle is necessary to align the result.

```
COP888 MULTIPLY (16X16) SUBROUTINE
MULTIPLICAND IN [1,0] MULTIPLIER IN [3,2]
PRODUCT IN [5, 4, 3, 2]

         .SECT   MEMCNT, REG
CNTR:    .DSB 1
         .SECT   CODE, ROM
MULT:    LD      CNTR,#17
         LD      B,#4
         LD      [B+],#0
         LD      [B],#0
         LD      X,#0
         RC
MLOOP:   LD      A,[B]
         RRC     A
         X       A,[B-]
         LD      A,[B]
         RRC     A
         X       A,[B-]
         LD      A,[B]
         RRC     A
         X       A,[B-]
         LD      A,[B]
         RRC     A
         X       A,[B]
         LD      B,#5
         IFNC
         JP      TEST
         RC
         LD      B,#4
         LD      A,[X+]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[X-]
         ADC     A,[B]
         X       A,[B]
TEST:    DRSZ    CNTR
         JP      MLOOP
         RET
```

B.9.4 Binary Division

The following program shows a subroutine for a 16-by-16-bit binary division. A 16-bit quotient is generated along with a 16-bit remainder. The dividend is left shifted up into an initially-cleared 16-bit test window where the divisor is test-subtracted. If the test subtraction generates no high-order borrow, then the real subtraction is performed with the result stored back in the test window. At the same time, a quotient bit (equal to 1) is inserted into the low end of the dividend window to record that a real subtraction has taken place. The entire dividend and test window is then shifted up (left shifted) one bit position with the quotient following the dividend.

Note that the four left shifts (LD, ADC, X) in the LSHFT section of the program are repeated as straight-line code rather than a loop in order to optimize throughput time.

```
COP888 DIVIDE (16X16)
SUBROUTINE
DIVIDEND IN [3,2]
DIVISOR IN [1,0]
QUOTIENT IN [3,2]
REMAINDER IN [5,4]

         .SECT   MEMCNT, REG
CNTR:    .DSB 1
         .SECT   CODE, ROM
DIV:     LD      CNTR,#16
         LD      B,#5
         LD      [B-],#0
         LD      [B],#0
         LD      X,#4
LSHFT:   RC
         LD      B,#2
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
TSUBT:   SC
         LD      B,#0
         LD      A,[X+]
         SUBC    A,[B]
         LD      B,#1
         LD      A,[X-]
         SUBC    A,[B]
         IFNC
         JP      TEST
SUBT:    LD      B,#0
         LD      A,[X]
         SUBC    A,[B]
         X       A,[X+]
         LD      B,#1
         LD      A,[X]
         SUBC    A,[B]
         X       A,[X-]
         LD      B,#2
         SBIT    0,[B]
TEST:    DRSZ    CNTR
         JMP     LSHFT
         RET
```

With a division where the dividend is larger than the divisor (relative to the number of bytes), an additional test step must be added. This test determines whether a high-order carry is generated from the left shift of the dividend through the test window. When this carry occurs, the program branches directly to the SUBT subtract routine. This carry can occur only if the divisor contains a high-order bit. Moreover, the divisor must also be larger than the shifted dividend when the shift has placed a high-order bit in the test window. When this case occurs, the TSUBT test subtract shows the divisor to be larger than the shifted dividend and no real subtraction occurs. Consequently, the high-order bit of the shifted dividend is again left shifted and results in a high-order carry. This test is illustrated in the following program for a 24-by-8-bit binary division.

Note that the four left shifts (LD, ADC, X) in the LSHFT section of the program are repeated with the JP jump to LUP instruction in order to minimize program size.

```
COP888 DIVIDE (24X8) SUBROUTINE
DIVIDEND IN [2,1,0]
DIVISOR IN [4]
QUOTIENT IN [2,1,0]
REMAINDER IN [3]

         .SECT    MEMCNT, REG
CNTR:    .DSB 1
         .SECT    CODE, ROM
DIV:     LD       CNTR,#24
         LD       B,#3
         LD       [B],#0
LSHFT:   RC
         LD       B,#0
LUP:     LD       A,[B]
         ADC      A,[B]
         X        A,[B+]
         IFBNE    #4
         JP       LUP
         IFC
         JP       SUBT
TSUBT:   SC
         LD       B,#3
         LD       A,[B+]
         SUBC     A,[B]
         IFNC
         JP       TEST
SUBT:    LD       A,[B-]
         X        A,[B]
         SUBC     A,[B]
         X        A,[B]
         LD       B,#0
         SBIT     0,[B]
TEST:    DRSZ     CNTR
         JMP      LSHFT
         RET
```

B.10 EXTERNAL POWER WAKEUP CIRCUIT

Power-on wakeup is a technique used in battery powered applications such as electronic keys or digital scales to save battery power. Instead of using the HALT mode when the application is not in use, the COP device is powered off. If there is only one input switch in the application, the implementation is simple. This switch is put in series with the battery, providing power to the circuit when the switch is closed.

If there is more than one switch, power-on wakeup can be achieved by using an NPN transistor and one resistor per switch as shown in Figure B-8. Here, the circuit ground is connected to the battery negative terminal via the NPN transistor. If the base is floating, it will not conduct. If the base is pulled to V_{CC} via a current-limiting resistor, it will conduct, powering up the circuit.

An alternative technique is shown in Figure B-6. Here the positive terminal of the battery is connected to the V_{CC} line via a switch, a diode and two resistors per line. If a switch is pressed, power is applied to the V_{CC} line. The pull-down resistors pull any ports connected to open switches to ground. If the switch is closed, the voltage on the switch will be V_{CC} plus the diode voltage drop. If this potential were directly applied to the Port

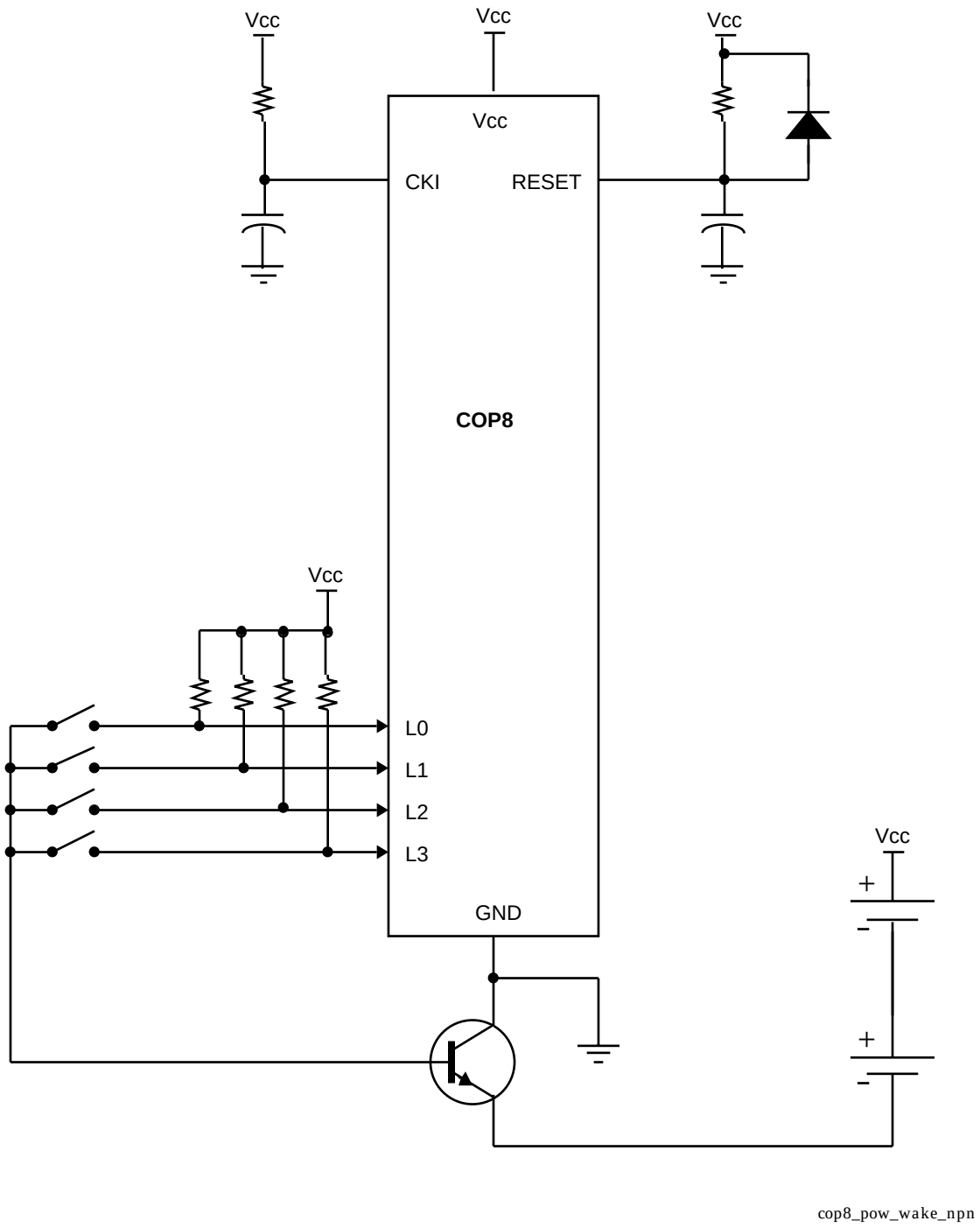


Figure B-8 Power Wakeup Using An NPN Transistor

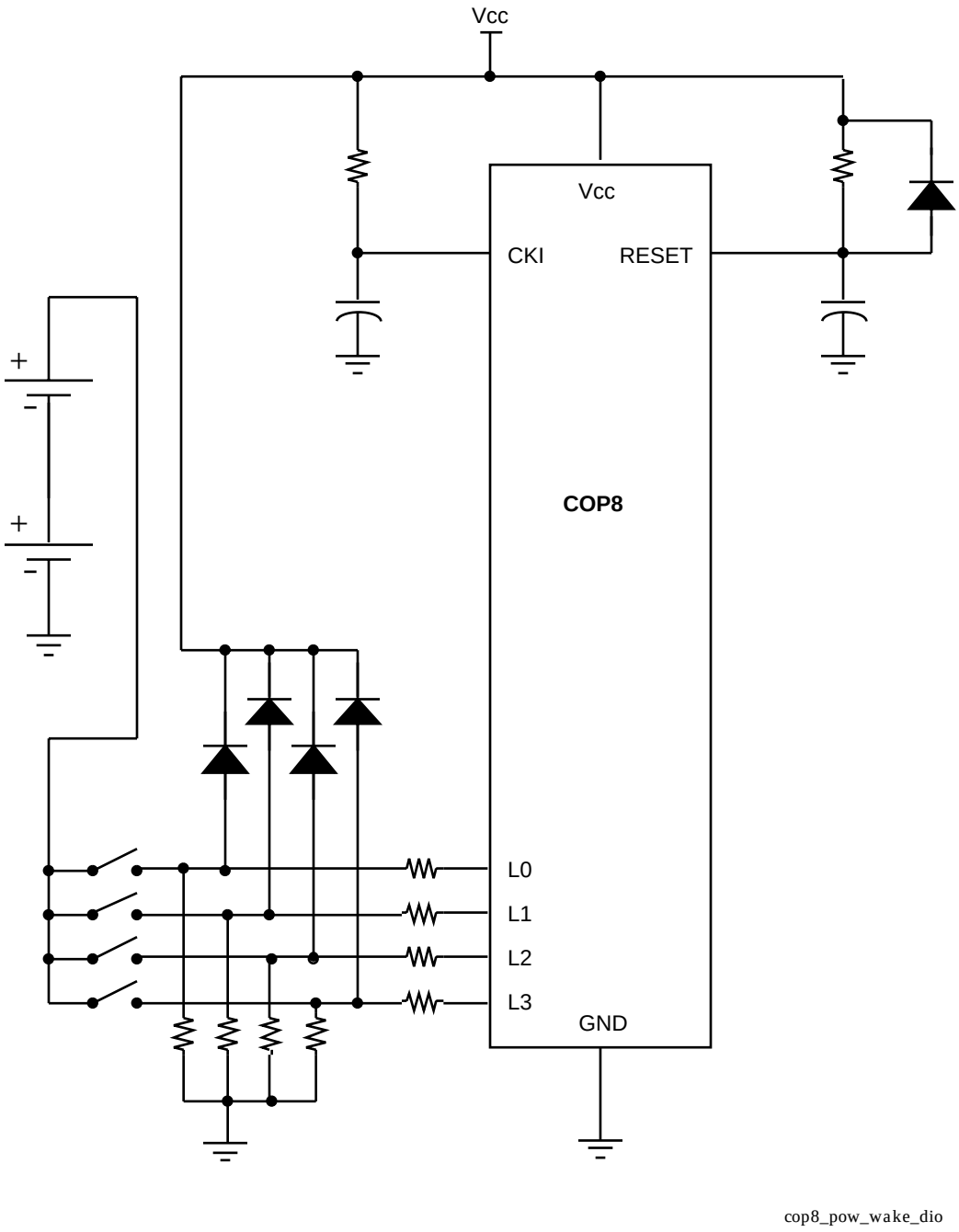


Figure B-9 Power Wakeup Using Diodes And Resistors

L pin, the COP device would be driven outside the operating specification. Therefore, series protection resistors are used on all Port L pins connected to the switches.

B.11 WATCHDOG RESET CIRCUIT

The following circuit is recommended for connecting the $\overline{\text{WDOUT}}$ pin to the $\overline{\text{RESET}}$ pin (Figure B-10). This circuit guarantees that the COP888 receives a valid reset signal as the result of a Watchdog/Clock Monitor error. With this circuit, the $\overline{\text{RESET}}$ is held low a minimum of 16 instruction cycles even if C_1 fails to discharge completely. The 2.5kΩ resistor limits the current into the $\overline{\text{WDOUT}}$ pin. This prevents damage to the $\overline{\text{WDOUT}}$ pin during the discharge of C_1. The components R_1 and C_1 are chosen to ensure a reset risetime of five times the power supply risetime.

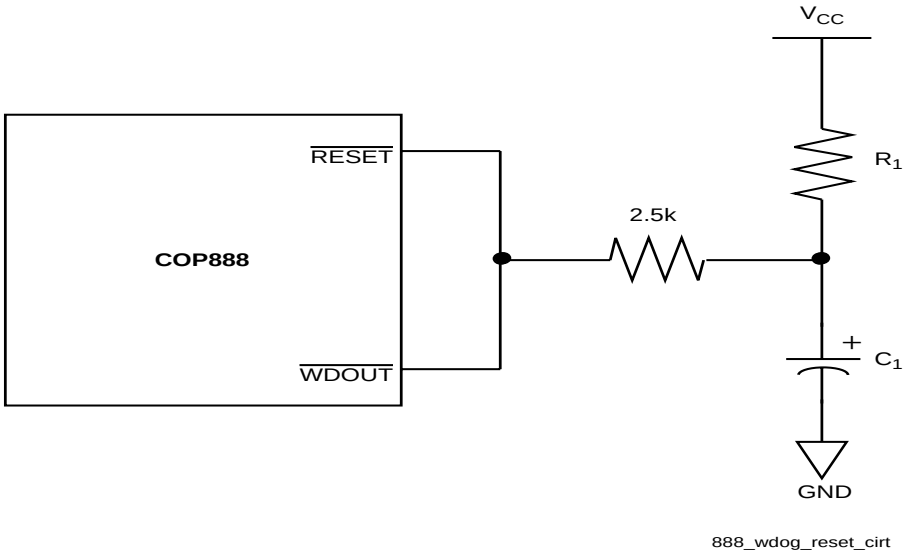


Figure B-10 Watchdog Reset Circuit

B.12 INPUT PROTECTION ON COP888 PINS

The COP888 input pins have internal circuitry for protection from ESD. The internal circuitry is shown in Figures B-11 and B-12.

The input protection circuitry is implemented with the P_channel transistors. The equivalent diode circuit is shown in Figure B-13.

When the inputs are tri-stated and the input voltage on the pin is between GND and V_{CC}, the input protection diodes are off. The only current drawn into or out of the pin is leakage current. If the input is expected to be below GND and/or above V_{CC}, an external series resistor must be used to limit the input current below the maximum allowable current.

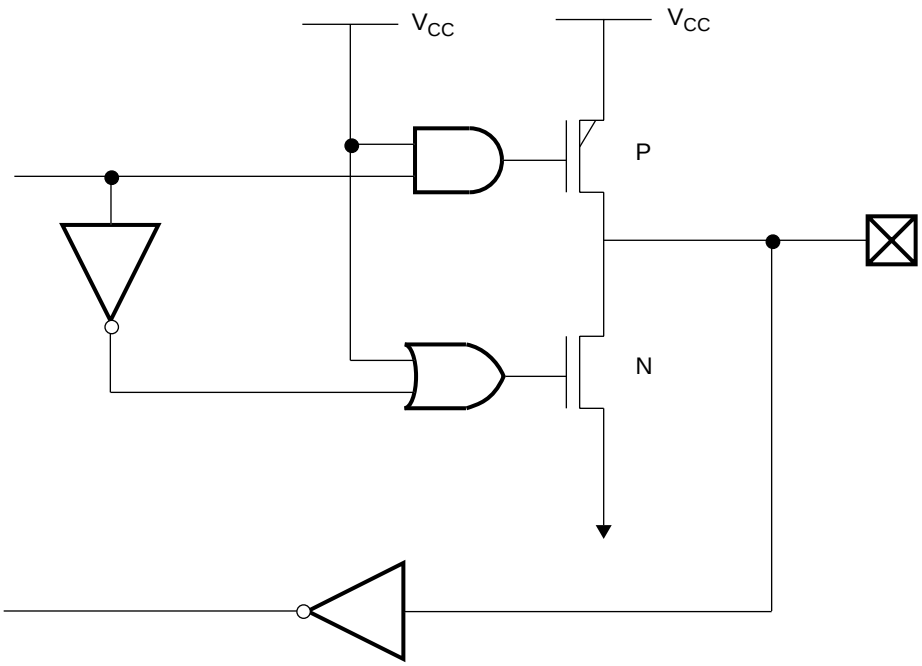


Figure B-11 Ports L/C/G Input Protection (Except G6)

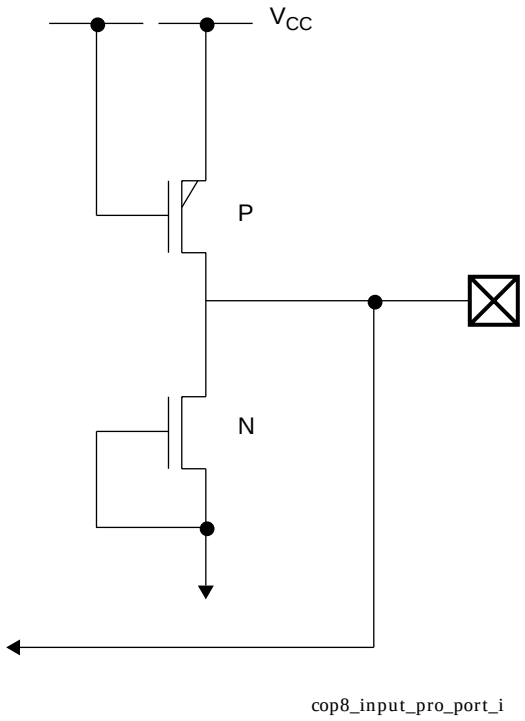


Figure B-12 Port I Input Protection

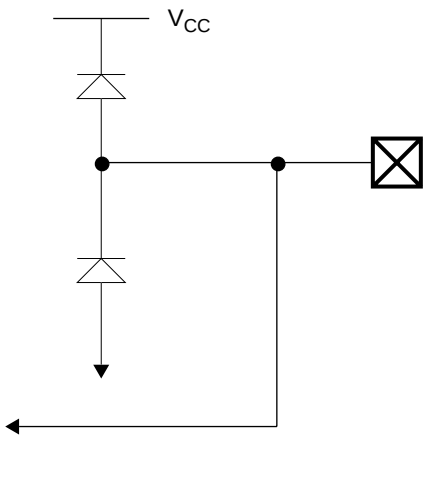


Figure B-13 Diode Equivalent of Input Protection

In addition to limiting the input current to below the maximum latchup spec (specified in the datasheet), the user should also consider the fact that drawing excessive continuous current into the pin, even though below the maximum latchup current, may cause overstress.

A typical example of drawing continuous current is in an automotive application where the ignition signal (battery) is connected to an input pin through a series resistor. Assuming a 100K series resistor with a tolerance of ±10%, the worst case resistor value is 90K. The battery voltage is assumed to be 12V for normal operation and 24V for a "jump start." The high voltage applied to the pin causes the on-chip protection diode to be forward biased, resulting in current into the associated V_{CC} metal trace. Based on a diode threshold voltage of 0.6V, the voltage at the pin will be V_{CC} + 0.6V. Based on a V_{CC} value of 5V, the input current can be calculated as follows:

Normal Operation:

$$\text{Input current} = \frac{[12 - (5 + 0.6)]}{90K} = 71\mu A$$

Jump Start:

$$\text{Input current} = \frac{[24 + (5 + 0.6)]}{90K} = 204\mu A$$

A study of the internal circuitry indicates that the input pin can draw about 200 µA without causing any damage or reliability problem.

Another approach is to use appropriate external circuitry that prevents the input protection diodes from being biased. An example is shown in Figure B-14.

The resistors are required to drop the +12V and the diode prevents the –12V from being applied to the pin.

For V_I = 12V ±5% and V_{CC} = 5V ± 5%, the resistor values are calculated to be:

R_1 = 47K +5%

R_2 = 82K ±5%

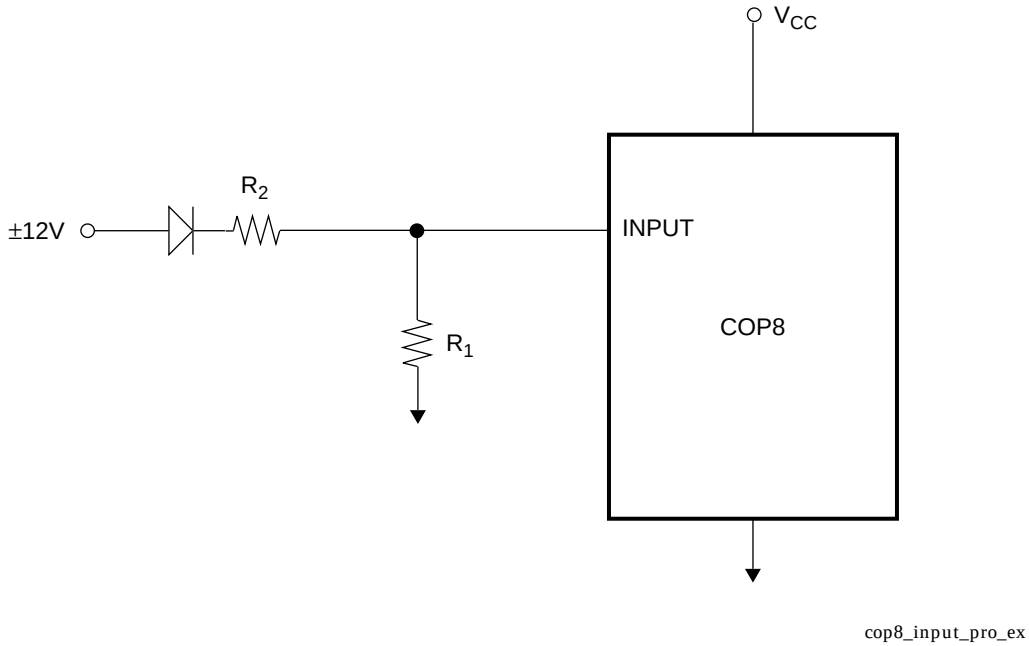


Figure B-14 External Protection of Inputs

This analysis does not apply to G6, $\overline{\text{RESET}}$, and CKI which do not have the protection diodes. Implementation of the above circuit will result in a V_{IH} that is between 0.7 V_{CC} and V_{CC}, and a V_{IL} that is between V_{SS} (0V) and 0.2 V_{CC}.

B.13 ELECTROMAGNETIC INTERFERENCE (EMI) CONSIDERATIONS

B.13.1 Introduction

CMOS has become the technology of choice for the processors used in many embedded systems due to its capability for low standby power consumption. However, CMOS is prone to high current transients on the power supply as the internal logic switches. These transients can easily be the source of high-frequency emissions from the system. The system designer should anticipate and minimize unwanted electromagnetic interference (EMI).

B.13.2 Emission Predictions

"EMI in a typical electronic circuit is generated by a current flowing in a loop configured within the circuit. These paths can be either V_{CC}-to-GND loops or output-to-GND loops. EMI generation is a function of several factors. Transmitted signal frequency, duty cycle, edge rates, and output voltage swings are the major factors of the resultant EMI levels."[1]

1. "FACT™ Advanced CMOS Logic Databook", National Semiconductor, 1993

The formula for predicting the Electric Field emissions from such a loop is as follows:

$$|E|_{MAX} = \frac{1.32 \times 10^{-3} IA(Freq)^2}{D} \times \left(1 + \left(\frac{\lambda}{2\pi D}\right)^2\right)^{1/2}$$

where:

- $|E|_{MAX}$ is the maximum E-field in the plane of the loop in µV/m
- I is the current amplitude in milliamps
- A is the loop area in square cm
- λ is the wavelength at the frequency of interest in meters
- D is the observation distance in meters
- $Freq$ is the frequency in MHz
- and the perimeter of the loop $P << \lambda$.

Applying this equation to a single standard output for a National Semiconductor Microcontroller, and performing a Fourier analysis of the output switching at a frequency of 20 MHz, yields the results shown in Table B-1. These calculations assume a trace length of 5 inches, a board thickness of 0.062 inches and a full ground plane. The load capacitance is 100 pf.

Table B-1 Electric Field Calculation Results

Harmonic (MHz)	Current (mA)	$\|E\|_{Max}$ (µV/M)	$\|E\|_{Max}$ (dBµV/M)
20	37.56	8.3	18.4
40	3.66	0.3	-10.2
60	26.13	44.2	33.0
80	4.44	0.6	-4.4
100	16.82	80.2	38.1
120	4.71	2.0	6.0
140	11.21	104.0	40.4
160	4.86	5.8	15.2
180	7.82	127.4	42.1

Note that the assumption is made that the output is switching at 20 MHz, which is rarely the case for a port output. There is noise, however, on the output at these frequencies due to switching within the device. This is the noise which is coupled to the output through V_{CC} and GND. Another point to keep in mind is that rarely does one single output switch, but usually several at one time, thus adding the effective magnetic fields from all the outputs which are switching.

Accurate analysis requires characterization of the noise present at the output due to V_{CC} or GND noise which is dependent on many factors, including internal peripherals in use, execution code, and address of memory locations in use.

B.13.3 Board Layout

There are two primary techniques for reducing emissions from within the application. This can be done either by reducing the noise or by controlling the antenna. Control of the antenna is accomplished through careful PC board layout.

General

Standard good PC layout practices will go a long way toward reducing emissions. Traces carrying large AC currents (such as signals with fast transition times, that drive large loads) should be kept as short as possible. Traces that are sensitive to noise should be surrounded by ground to the greatest extent possible. Ground and V_{CC} traces should be kept as short and wide as possible to reduce the supply impedance.

Ground Plane

One of the most effective ways to control emissions through board layout is with a ground plane. The use of a plane can help by providing a return path for fast switching signals, thus reducing loop size for both power and signals.

Multilayer Board

The best way to provide a ground plane is through the use of a multilayer printed circuit board. The large area and the proximity of the V_{CC} and GND planes provide additional decoupling for the power, and provide effective return paths for both power and signals.

The problem with the use of a multilayer board, particularly in consumer related industries, is cost. Due to the volumes involved, an addition of several dollars to the cost of an item may be prohibitive.

B.13.4 Decoupling

Control of the emitted noise can be accomplished by several techniques, including decoupling, reduced power supplies, and limitation of signal strength by the addition of series resistance.

It is important to take the time to properly design the decoupling for CMOS processors. Two decoupling techniques can and should be used to minimize both voltage and current switching noise in the system.

Capacitive Decoupling

Capacitive decoupling is commonly used to control voltage noise on the V_{CC} and GND lines of the board, but if the decoupling is properly designed and is kept as close as possible to the power pins of the device, it can also reduce the effective loop area and thus

the antenna efficiency. Capacitive decoupling can prevent high-frequency current transients from being seen by the power supply.

One factor of capacitive decoupling which is often overlooked is the frequency response of the capacitors. Each capacitor, dependent on value, lead length, and dielectric material, possesses a series resonant frequency beyond which the device has inductive characteristics. This inductance inhibits the capacitor from responding quickly to the current needs of the processor and forces the current to use the longer path back to the main power supply.

These inductive characteristics can be countered by the addition of extra capacitors of different values in parallel with the original device. As the value of the capacitor decreases (for capacitors of similar manufacture), the resonant frequency increases.

Placing multiple decoupling capacitors across the power pins of the processor can effectively improve the high frequency performance of the decoupling network. Capacitance values are normally selected which are separated by a decade. However, it is best to check the specifications of the capacitors which are used.

Inductive Decoupling

Another very effective method of decoupling which is rarely used is inductive decoupling. The proper placement of ferrite beads between the decoupling capacitors and the processor can significantly reduce the current noise on the power pins.

The use of inductive decoupling, which will increase the series impedance of the power supply, appears to be contradictory to the effect of capacitive decoupling. However, the purpose of inductive decoupling is to force nodes internal to the processor, which are not switching, into providing the charge for the nodes which are switching.

Ferrite beads are very effective for this type of decoupling due to their lossy nature. Rather than storing the energy and returning it to the circuit later, ferrites will dissipate the energy as a resistor.

One should be aware of potential repercussions from the use of any type of series isolation from the power supply. Due to the reduced V_{CC} which may be present during switching transients, interfacing to other devices in the system may be a problem. Since the V_{CC} should only be reduced for the duration of the switching transient, this should only be a problem if the other devices have especially sensitive and fast-responding inputs.

B.13.5 Output Series Resistance

The addition of resistance in series with outputs can have a significant effect on the emissions caused by the switching of the outputs.

Outputs that drive large capacitive loads can have a lot of current flowing when they switch. While the series resistance may slow the switching speed of the node and thus affect the propagation delay, it can also have a large effect on emissions by reducing the amplitude of the current spike that charges or discharges the load.

B.13.6 Oscillator Control

One very definite source of emissions is the system clock. The oscillator is intended to switch at high speed and therefore will emit some noise. Keeping the circuit loop of the oscillator as small as possible will help considerably.

Ceramic resonators are available with the capacitive load included in a single three terminal package. The use of these devices and placing them right next to the processor can reduce emissions as much as 10 dB.

RC oscillators are particularly troublesome for emissions due to the high transient current when the processor turns on the N-channel device that discharges the capacitor. The transistor is meant to be large and to turn on strongly in order to discharge the capacitor as quickly as possible. This allows simple control over the frequency of oscillation but causes difficulty for the designer of systems for EMI-sensitive applications.

B.13.7 Mechanical Shielding

A last resort for controlling emissions is the addition of mechanical shielding. While shielding can be effective and can be easier from an electrical design standpoint, the implementation and installation of a proper electromagnetic shield can be excessively costly and time consuming.

It is much better to design the system with the control of emissions in mind from the start rather than to apply bandages when it is time to begin production.

B.13.8 Conclusion

While electromagnetic emissions can be a problem for the designer of any electronic system, it is particularly troublesome in the design of high speed CMOS systems. With knowledge of the primary sources of noise, and the ways to combat that noise, it is possible to design and build systems which are electromagnetically quiet.

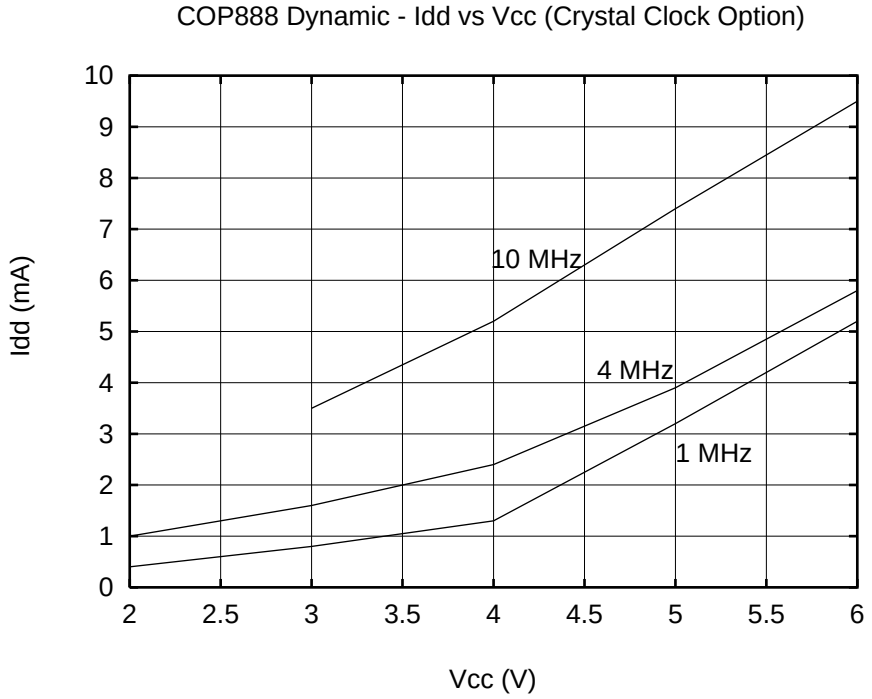
Very few references to specific values of capacitance, resistance, or inductance have been made in this document. The reason for this is that a value which works well in one application may not be effective in another. The best way to determine the values which will work well for a particular application is by experimentation.

Appendix C

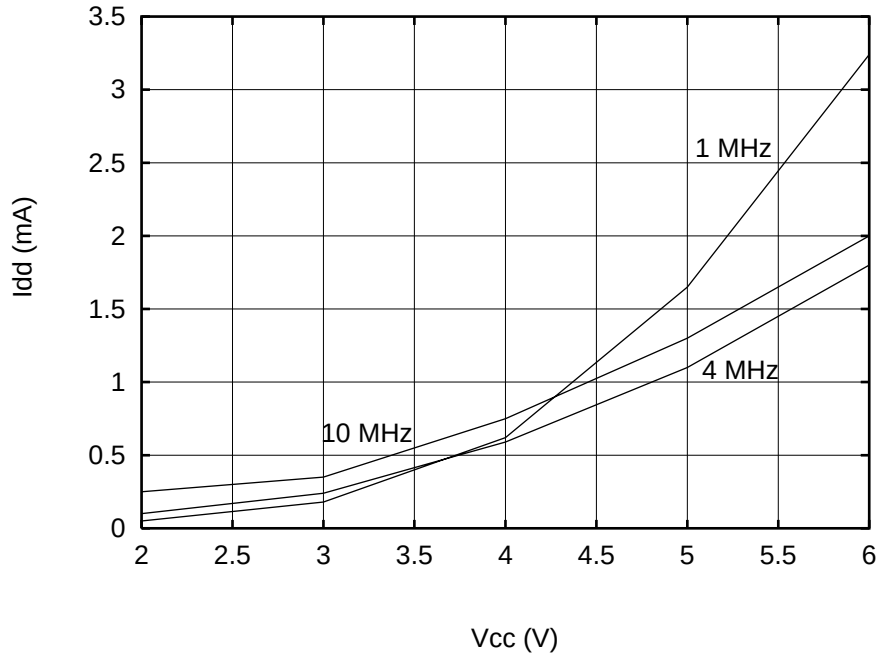
ELECTRICAL CHARACTERIZATION DATA

This appendix presents characterization data for the COP8 Flash Family members.

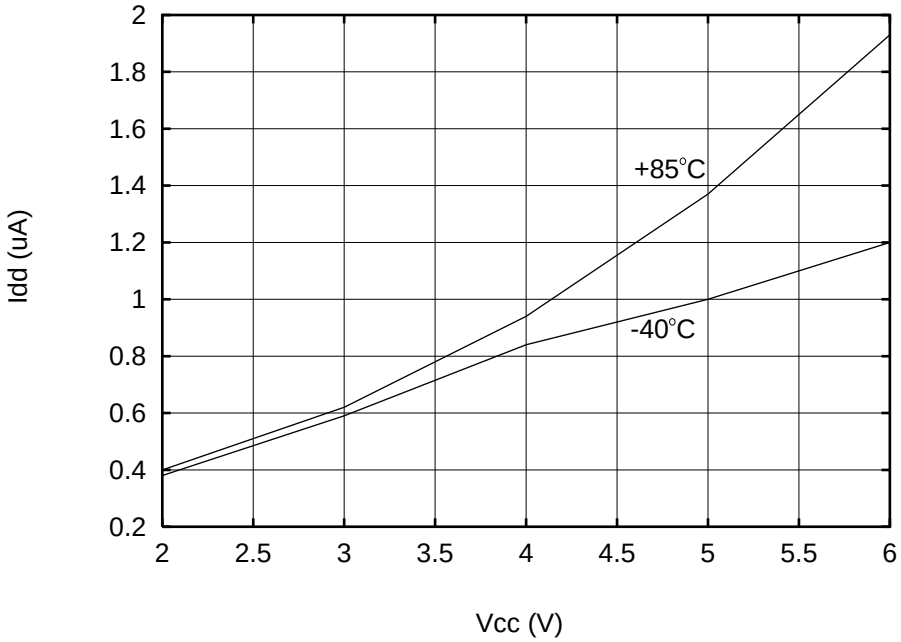
Characterization data is information gained from testing a wide range of sample of parts. Most tests are performed over the full temperature and operating voltage range of the COP8 devices. All information provided in the graphs represents typical values. Most parts will meet these typical values. However, National Semiconductor does not guarantee these values on all parts. Guaranteed numbers are provided in the AC and DC Electrical Characteristics tables found in every datasheet. Guaranteed numbers are tested on every device shipped to our customers.



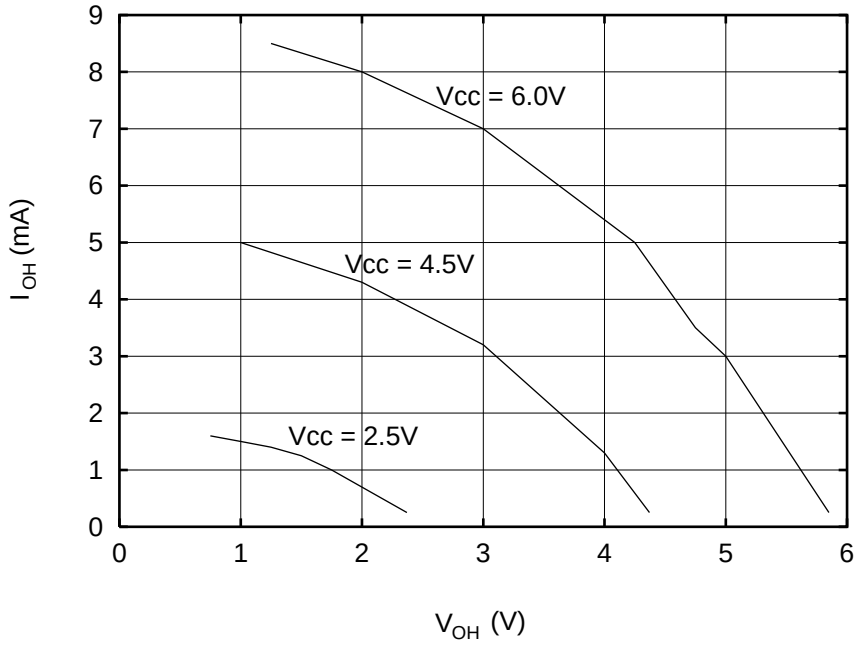
COP888 Idle - Idd vs Vcc (Crystal Clock Option)



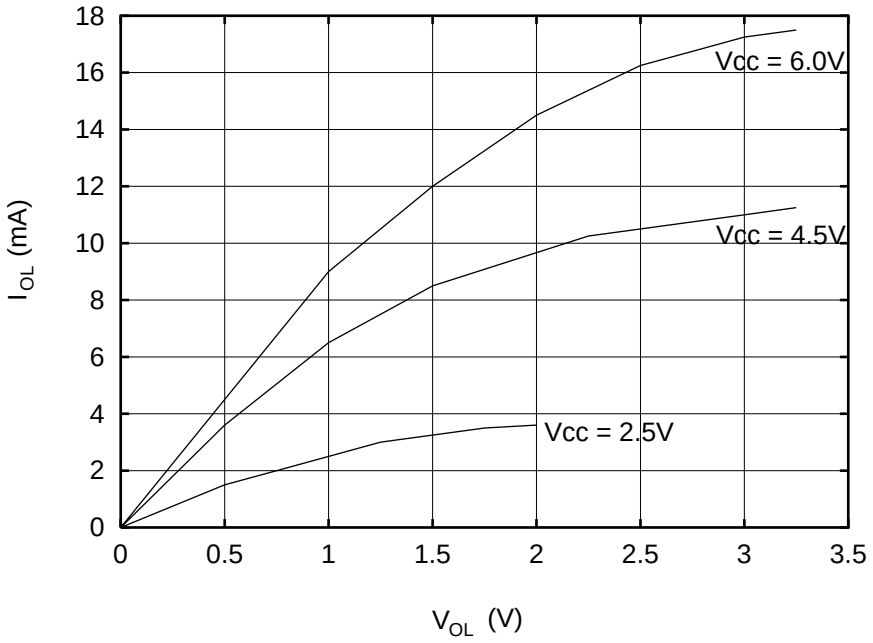
COP888 Halt - Idd vs Vcc



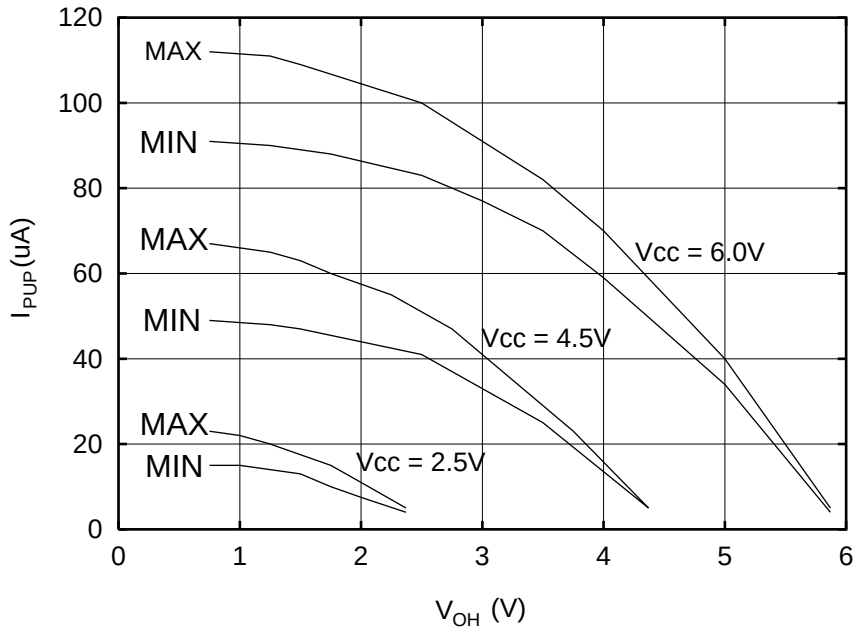
Port L/C/G Push-Pull Source Current



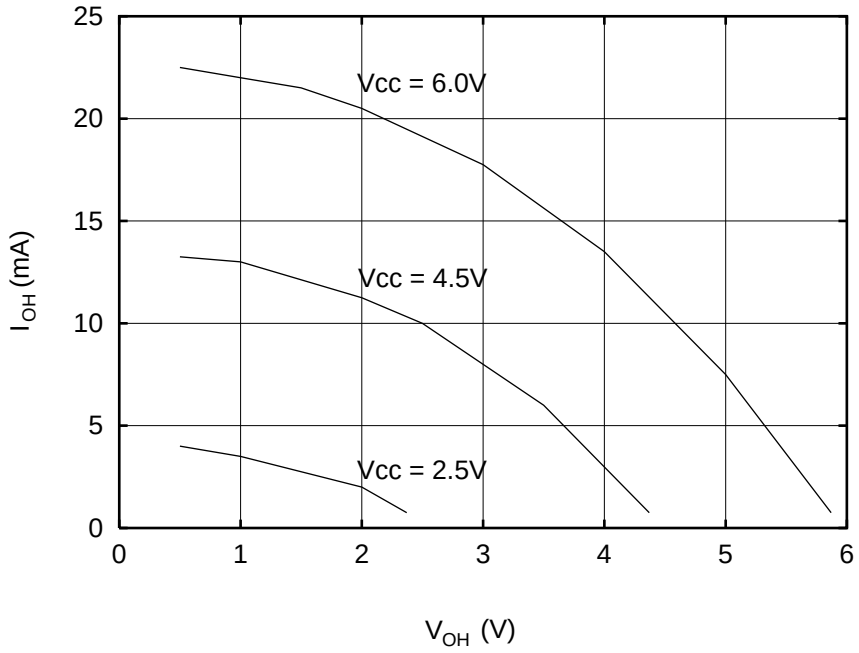
Port L/C/G Push-Pull Sink Current



Port L/C/G Pull-up Source Current



Port D Source Current



Port D Sink Current