COP8™ MICROCONTROLLER

COP8SAx Designer's Guide

Literature Number 620894-001
January 1997

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
-001	1/97	First Release

PREFACE

This manual is written with the intention to serve as a Designer's Guide for those who are considering to use a microcontroller from National's COP8SAx family of devices.

Chapter 1, MICROCONTROLLER BASICS, reviews microcontroller basics, including definitions, building blocks, operation, applications, and programming. Those who are already familiar with microcontrollers can skip this chapter.

Chapter 2, COP8SAx FAMILY, provides a detailed overview of the COP8SAx family of devices. It describes the features, architecture, instruction set, and electrical characteristics. For more detailed information, refer to the COP8 Feature Family User's Manual.

Chapter 3, DEVELOPMENT TOOLS, describes the range of development tools that are available for developing and testing application software that is run on the COP8SAx microcontroller.

Chapter 4, COP8SAx APPLICATION IDEAS, provides an overview of some design examples using the COP8SAx microcontroller. Within these examples, the users can find application hints that may be useful in implementing a design.

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

CONTENTS

Figures

Tables

Chapter 1

MICROCONTROLLER BASICS

1.1 WHAT IS A MICROCONTROLLER?

Microcontroller is an highly integrated single-chip microcomputer. Some of the key elements of a microcontroller include a CPU to process information, program memory to store instructions, data memory to store information, system timing, and input/output sections to communicate with the outside world.

1.1.1 CPU

Central Processing Unit (CPU) is the heart of a microcontroller where all of the arithmetic and logical operations are performed. This is the calculator part of the microcontroller. The CPU gets program instructions from the program memory.

1.1.2 Program Memory

Program Memory contains a set of CPU instructions organized into a particular sequence to do a particular task. Program Memory is referred to as Read Only Memory (ROM) or OTP/EPROM. OTP or “One-Time Programmable” can be programmed only once and the program is stored permanently, even when the microcontroller power is turned off. Program memory enables the microcontroller to immediately begin running its program as soon as it is turned on.

1.1.3 Data Memory

A form of memory that can be both read and written is required for the program stack, data storage, and program variables. This type of memory is commonly referred to as Random Access Memory (RAM). Each memory location has a unique address which the CPU uses to find the information it needs.

A typical microcontroller contains both ROM and RAM type memory.

1.1.4 Timing

Microcontrollers use a timing signal, called a clock, to provide a timing reference for program execution, and to determine when data should be written to or read from memory. It also provides timing for on-board peripherals.

1.1.5 Inputs/Outputs

Microcontrollers require interface sections to communicate with external circuitry. Input ports allow data and status conditions to be read into the microcontroller while the

output ports allow the microcontroller to affect external logic systems. The interfaces between the microcontroller and the outside world vary with the application, and may include display units, keypads, switches, sensors, relays, motors, and so on.

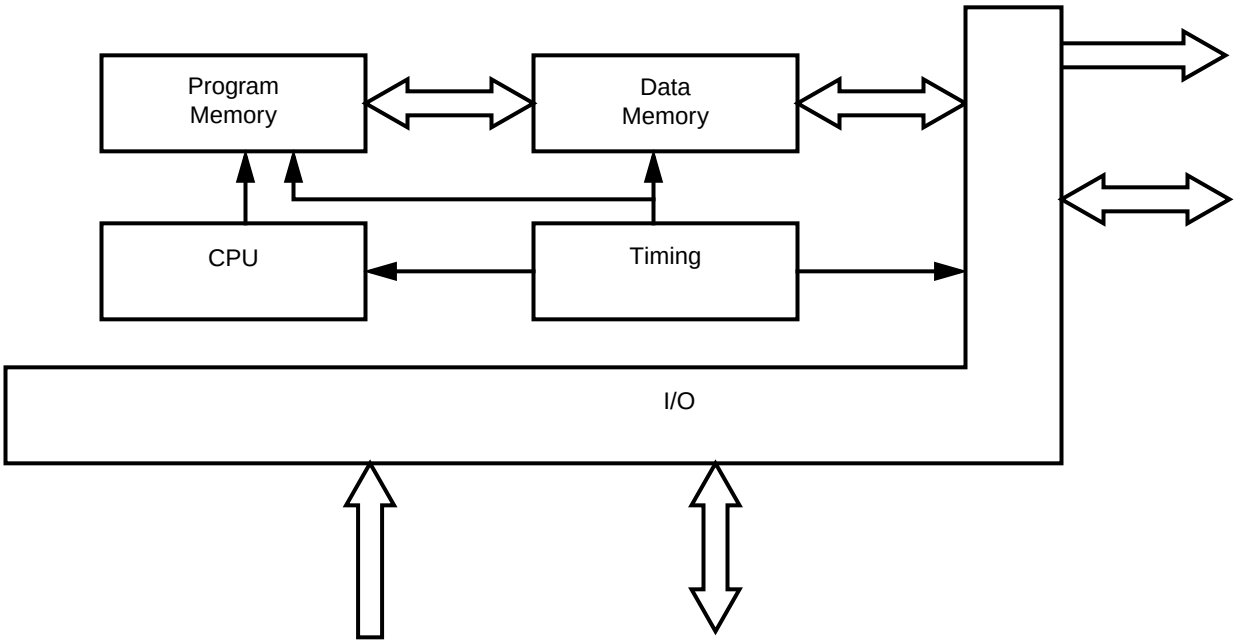


Figure 1-1 Microcontroller General Block Diagram

1.2 WHAT DOES A MICROCONTROLLER REPLACE?

A microcontroller can do the work of many different types of logic circuits. Discrete logic circuits are permanently wired to perform the function they were designed to do. If the design requirements are changed slightly, an entire printed circuit board or many boards may have to be redesigned to accommodate the change. With a microcontroller performing the logic functions, most changes can be made simply by reprogramming the microcontroller. That is, the software (program) is changed rather than the hardware (logic circuits). This makes the microcontroller a very attractive building block in any digital system. With a microcontroller-based design, the designer can simply add a feature set to the product with minimal software/hardware changes.

Microcontrollers can also be used to replace analog circuitry. Special interface circuits can be used to enable a microcontroller to input and output analog signals.

There are also situations where the designer considers using an ASIC (Application-Specific Integrated Circuit) as logic replacement for control applications. Microcontrollers can efficiently be used as single-chip replacements in such applications with significantly lower development cost and fast time to market.

One of the great benefits of the electronic revolution is that it brought intelligence, and with it adaptability, to traditional electro-mechanical devices. By continuously gathering information on the performance of the device, its operating environment, and other factors, microcontroller intelligence can determine a new and proper strategy and then command the surrounding device to react. The most important of these are the new

intelligent applications which are adapted in real time to changing conditions, such as the iron that senses when the cloth beneath it has reached the proper temperature and moisture, or the vacuum cleaner that adjusts its speed and brush height automatically to changes in carpet nap.

1.3 WHAT ARE MICROCONTROLLER APPLICATIONS?

Microcontrollers applications are more or less limited only by the user imagination. So pervasive has been this technological revolution that we barely notice it anymore. Microcontrollers now reside in our televisions, keyboards, modems, printers, wristwatches, telephones, cars, household appliances, and every other part of home and work life. The market for microcontrollers continues to expand rapidly, encompassing a wide range of consumer, industrial, automotive, and telecommunications applications. In fact, a typical home today contains over 35 microcontroller-based products — a figure that some sources estimate could grow to almost 250 by the year 2000.

The emergence of new low cost microcontrollers offers a wealth of benefits for today's consumer applications and represent an entirely new profit source for manufacturers. In the past, the high cost of electronics limited the use of microcontrollers to "high tech" applications such as televideo recorders, stereo systems, and high-end durable goods such as washing machines. Today, the application base has broadened to include systems such as coffee machines, irons, shavers, and cleaners, where the introduction of electronics helps to provide product differentiation and allows the inclusion safety features.

Table 1-1 Microcontroller Features/Applications Matrix (Sheet 1 of 3)

Market Segment		Applications	Applications Features/Functions	Microcontroller Features Required
Consumer	Children Toys and Games	Basketball/Baseball Games Children Electronic Toys Darts Throws Juke Box Pinball Laser Gun	Battery Driven Replacing Discrete with Low Cost Driving Piezo/Speaker/LEDs Directly Very Cost Sensitive	Very Low Price Low Power Consumption Wide Voltage Range High-Current Outputs Small Packages
	Electronic Audio Items	Audio Greeting Cards Electronic Musical Equipment	Battery Driven Tone Generation Low Power	Wide Voltage Range Low Power Consumption Efficient Table Lookup Flexible Timer
	Electronic Appliances and Tools	Small Appliances: Irons Coffee Makers Digital Scales Microwave Ovens Cookers Food Processors Blenders	Low Cost Power Supply Temp Measurement Safety Features Noise Immunity Driving LEDs/Relays/Heating Elements	Brown Out Detection On-Board Comparator High-Current Outputs Watchdog/Software Interrupt Schmitt Trigger Inputs 16-Bit PWM Timer
		Household Appliances: Oven Control Dishwasher Washing Machine/Dryer Vacuum Cleaner Electronic Heater Electronic Home Control (Doorbell, Light Dimmer, Climate) Sewing Machine	Rely on Hard-Wire Relay Circuits, Timers, Counters, Mechanical Sequence Controllers Temp Control Noise Immunity Safety Features Timing Control Main Driven	Brown Out Detection On-Board Comparator On-Board A/D Watchdog/Soft Interrupt Schmitt Trigger Inputs Flexible Timers PWM Outputs High-Current Outputs Safety Features
	Portable, Handheld, Battery Powered	Scales Multimeters (portable) Electronic Key Laptop/Notebook Keyboard Mouse Garage Door Opener TV/Electronic Remote Control Portable or Retail Point-Of-Sale Device Jogging Monitor Smart Cards	Battery Driven Minimal Power Consumption Low Voltage Sensing Measurement Standby Mode Flexible Package Offerings Small Physical Size	Low Voltage Operation Low Power Consumption Wide Voltage Range Power Saving Modes Multi-Input Wakeup On-Board Comparator Small Packages

Table 1-1 Microcontroller Features/Applications Matrix (Sheet 2 of 3)

Market Segment		Applications	Applications Features/Functions	Microcontroller Features Required
Personal Communications		Cordless Phone (base/handset) Phone Dialer Answering Machine Feature Phone PBX Card CB Radios/Digital Tuners Cable Converter	Low Power Timing Serial Interfaces Low Voltage Tone Dialing Battery Saving Functions Small Physical Size	Low Current Drain Low Voltage Operation Standby Mode UART Serial Synchronous Interface 16-Bit Timers Schmitt Trigger Inputs LED Direct Drive Sufficient I/O in Small Packages
Medical	Monitors	Thermometer Pressure Monitors Various Portable Monitors	Battery Driven Sensing/Measurement Data Transmission Low Power Low Voltage	On-Board Comparator (low cost A/D) 16-Bit Timer Low Power Consumption Low Voltage Operation
	Medical Equipment	Bed-side Pump/Timers Ultrasonic Imaging System Analyzers (chemical, data) Electronic Microscopes	Monitoring Data Data Transmission Timing	Serial Interface A/D 16-Bit Timers
Industrial	Motion Control	Motor Control Power Tools	Motor Speed Control Noisy Environment Timing Control	Flexible PWM Timers Schmitt Trigger Inputs High-Current Outputs
	Security/ Monitoring System	Security Systems Burglar Alarms Remote Data Monitoring Systems Emergency Control Systems Security Switches	Data Transmission Monitoring (scan inputs from sensors) Keypad Scan Timing Diagnostic Data Monitoring Drive Alarm Sounders Interface to Phone System Standby Mode	UART Flexible 16-Bit PWM Timers Flexible I/O Single Stop A/D Capability Power Saving Modes (HALT, Multi-Input wakeup) Serial Synchronous Interface
	Misc.	Switch Controls (elevator, traffic, power switches) Sensing Control Systems/Displays Pressure Control (scales) Metering (utility, monetary, industrial) Lawn Sprinkler/Lawn Mowers Taxi Meter Coin Controls Industrial Timers Temperature Meters Gas Pump Gas/Smoke Detectors	Timing/Counting Sensing Measurement	Generic Microcontroller

Table 1-1 Microcontroller Features/Applications Matrix (Sheet 3 of 3)

Market Segment	Applications	Applications Features/Functions	Microcontroller Features Required
Automotive	Radio/Tape Deck Controls Window, Seat, Mirror, and Door Controls Climate Controls Headlight/Antenna Power Steering Anti Theft Slave Controllers	Timing Motion Control Display Control Soft Runaway/Trap Recovery (safety considerations) EMI/Noise Immunity Serial Interfaces Standby Modes Wide Temp Range	Flexible PWM Timers Power Saving Modes Multi-Input Wakeup WATCHDOG Software Trap UART CAN Interface Special Features for Dashboard Control (counters, capture modules, MUL/DIV) Reduced EMI Wide Temp Range

1.4 WHAT IS THE DIFFERENCE BETWEEN A MICROCONTROLLER AND A MICROPROCESSOR?

The broad category of microcomputers is divide into two areas: microcontrollers and microprocessors. This distinction is made because these are really two different types of devices. Microcontrollers generally have a dual-bus architecture rather than the memory mapped von Neumann architecture common in most microprocessors. For control applications, microcontrollers are generally more memory efficient than microprocessors. The microcontroller instruction set is quite different in nature than the microprocessor instruction set. Microcontrollers are invariably single-chip devices and microprocessors are, generally, multi-chip devices. Microcontrollers dominate the microcomputer marketplace in terms of volume. To be sure, the division between microcontroller and microprocessor is sometimes blurred but the distinction is real nonetheless.

1.5 WHAT IS THE ARCHITECTURE OF A MICROCONTROLLER

Microcontrollers have two types of architecture: Von Neumann or Harvard.

1.5.1 Von Neumann Architecture

The Von Neumann architecture was named after John Von Neumann, an early pioneer in the computer field at Princeton. In this architecture, a CPU and memory are interconnected by a common address bus and data bus. Positive aspects of this approach include convenient access to tables stored in program memory (ROM or OTP/EPROM) and a more orthogonal instruction set. The address bus is used to identify which memory location is being accessed, while the data bus is used to transfer information either from the CPU to the selected memory location or vice versa. Von Neumann was the first to realize that his architecture model could have the memory serve as either program memory or data memory. In earlier computers (both electronic and electromechanical), the program store (often a program patchboard) had been completely separate from the data store (often a bank of vacuum tubes or relays).

The single address bus of the Von Neumann architecture is used sequentially to access instructions from program memory and then execute the instructions by retrieving data from and/or storing data to the data memory. This means that an instruction fetch cannot overlap a data access from memory.

The obvious advantage of a Von Neumann architecture is the single address bus and single data bus linking memory with the CPU. A drawback is that code can be inadvertently executed from data memory, opening up the possibility for undesired operation due to corruption of the program counter or other registers.

1.5.2 Harvard Architecture

The Harvard architecture was named after the Harvard Mark 1 and the early electromechanical computers developed at Harvard by Howard Aiken, another computer pioneer. This architecture has separate program memory and data memory with a separate address bus and data bus for each memory. One of the benefits of the Harvard architecture is that the operation of the microcontroller can be controlled more easily in the event of corrupted program counter. A modified (enhanced) Harvard architecture allows accessing data tables from program memory. This is very important with modern day microcomputers, since the program memory is usually ROM or OTP/EPROM) while the data memory is RAM. Consequently, data tables usually need to be in program memory so that they are not lost when the microcontroller is powered down.

The advantage of a modified Harvard architecture is that instruction fetch and memory data transfers can be overlapped with a two stage pipeline, which means the next instruction can be fetched from program memory while the current instruction is being executed using data memory. A drawback is that special instructions are required to access RAM and ROM (or OTP/EPROM) data values, making programming more difficult.

1.6 HOW DOES A MICROCONTROLLER OPERATE?

The CPU can request information from memory (or read an input port) by calling it by its memory address. The address with all its bits is stored in the CPU as binary number in a temporary data latch type memory called a register. The outputs of the register are sent over multiple wires (or single wire) to the microcontroller memory and peripherals. The group of wires (parallel) or the single wire (serial) that carries the address is called the address bus. The word “bus” refers to one or more wires that share a common path to/from multiple places. The address register holds address bits. The number of address bits depends on the microcontroller type.

Data is sent to the CPU over a data bus. The data bus is different from the address bus in that the CPU uses it to read information from memory or peripherals and to write information to memory or peripherals. Signals on the address bus originate only at the CPU and are sent to the other blocks attached to the bus. Signals on the data bus can either be inputs to the CPU or outputs from the CPU. The information on the data bus is sent or received at the CPU by the data register. In other words, the data bus is bi-directional and the address bus is uni-directional. The width of the address and the data bus may also be different, depending on the microcontroller type and memory size.

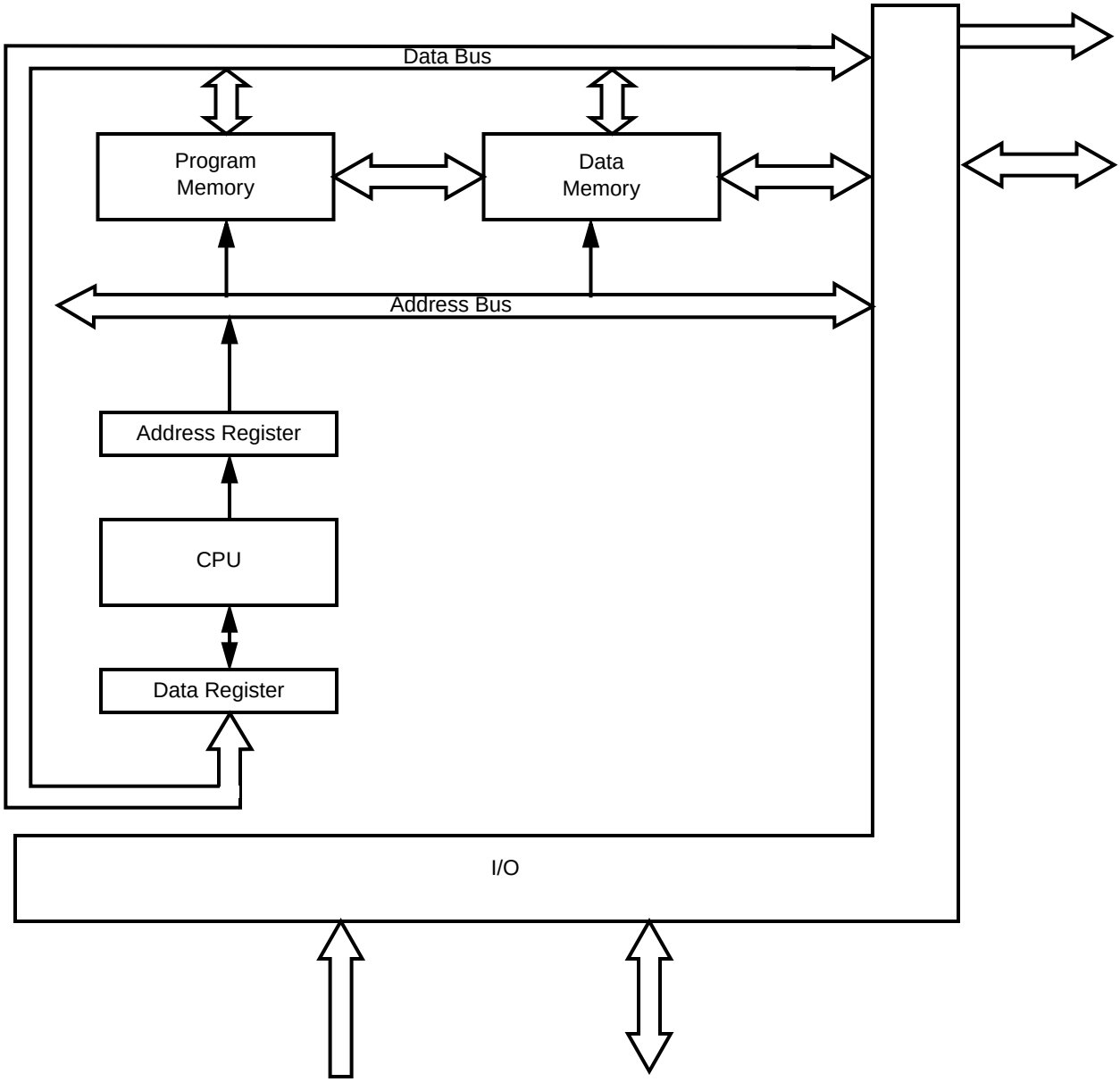


Figure 1-2 Microcontroller Operation

1.7 DESCRIPTION OF MICROCONTROLLER BUILDING BLOCKS

1.7.1 Program Memory

Program memory contains the microcontroller program.

There are several types of program memory—ROM, OTP/EPROM, EEPROM.

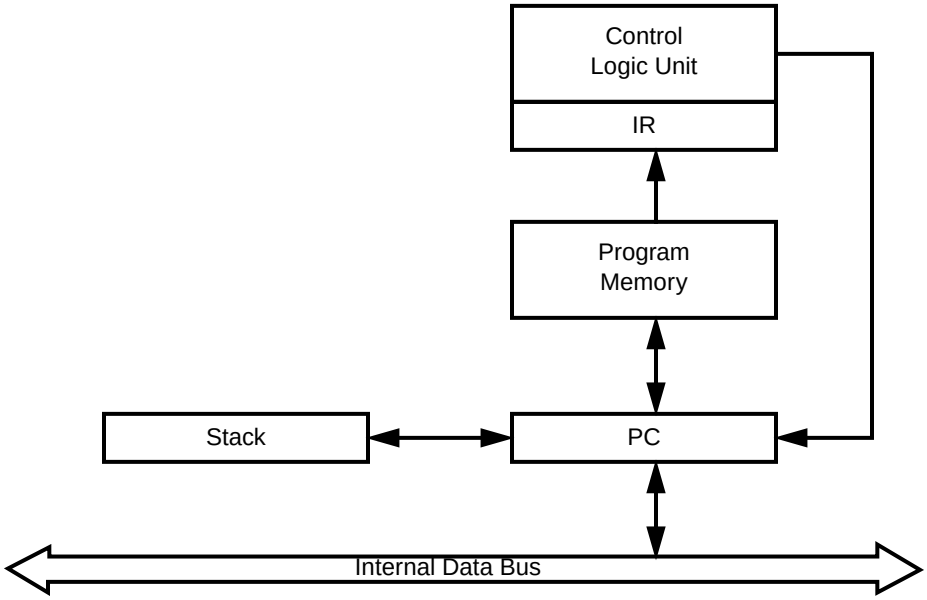


Figure 1-3 Program Memory Section

Structure

The number of bits per address location and number of memory locations varies from one microcontroller type or family to another (8/2/14/16 -bit by 0.5k, 1k, 2k, 4k, etc.).

Program Counter (PC)

The program counter is used by the CPU to address program memory locations that contain the program instructions. Every time an opcode (binary representation of an instruction) is fetched (read) from memory, the program counter in incremented (advanced by one) so that it points to the next byte following the opcode (assuming 8-bit memory organization, an assumption used throughout the rest of this document. Some program instructions require more than one byte. In that case, the program counter supplies the address, the second byte is fetched from memory, and the program counter is incremented. Each time the CPU performs a fetch operation, the program counter is incremented; thus, the program counter always points to the next byte in the program. Therefore, after all bytes required for one complete instruction have been read, the program counter points to the beginning of the next instruction to be executed. The program counter size is base on the size of the program memory.

Instruction Storage

Each byte in program memory contains an instruction like ADD or JMP, represented by a code or opcode (example: 033 = ADD). Instructions are stored in the order to be executed. For example:

```
LD      A,#00
INC     A
JMP     00
```

These instructions (8-bit opcodes) might be stored at memory locations 00, 01, and 02. To execute this code, the program counter (PC) is set to point to location 00 (the PC contains address 00). The opcode stored at location 00 is read into an instruction register in the control block. The control block decodes the opcode and executes the LD instruction. The PC is incremented by one to point to the next instruction (the PC contains 01). Then the fetch, decode, and execute steps are repeated.

1.7.2 Data Memory

Data memory is used to store and retrieve information. Typically, there are two types of data memory that caan be used: RAM and/or EEPROM (electrically erasable memory).

Structure

1. Size

   The data memory size varies from one microcontroller type or family to another (4-bit/8-bit/16-bit by 16, 32, 64, 128 bytes, etc.)

2. Von Neumann Architecture

   With Von Neumann architecture data and program memory share the same memory space. If code is placed in a separate memory space in locations 0000 through 03FF, then data must reside in locations above 03FF.

3. Harvard Architecture

   With Harvard architecture data and program memory have separate memory spaces. Code may be placed in locations 0000 through 03FF and data may reside in locations 000 to 03F.

Data Storage

The data memory can contain intermediate result values from math operations (add/sub), tables, flags, and system stack.

Pointer

A pointer contains an address which specifies a location in memory. A pointer "points" to the location of some data. A data pointer is used to specify the location of a byte/word in

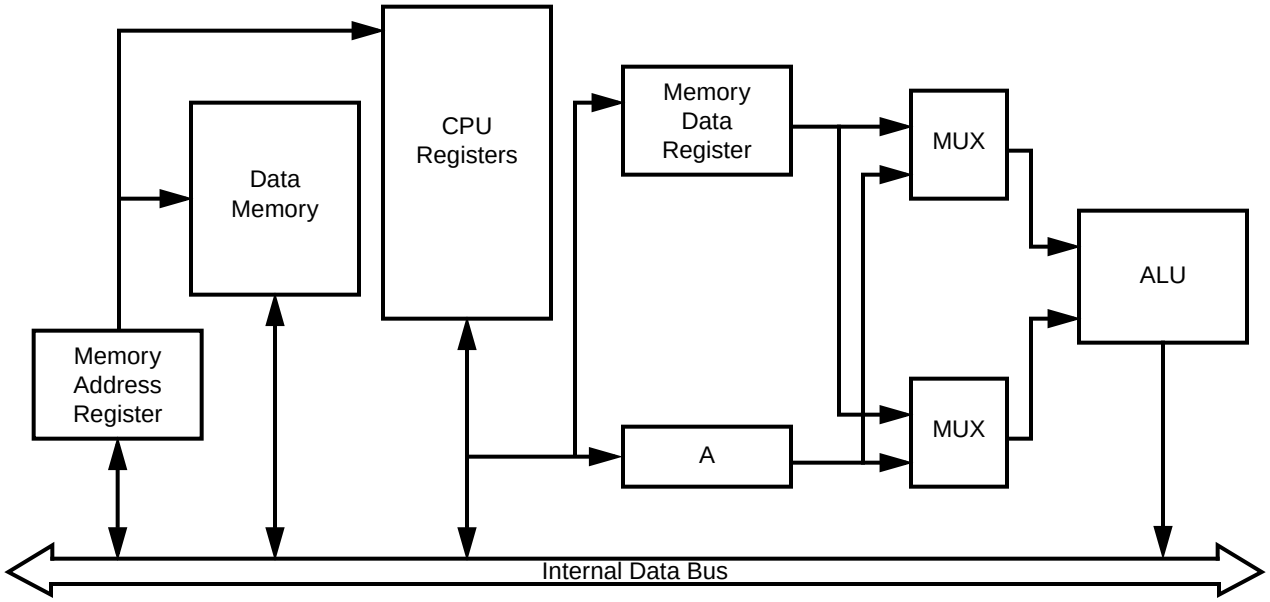


Figure 1-4 Separate Data and Code Space

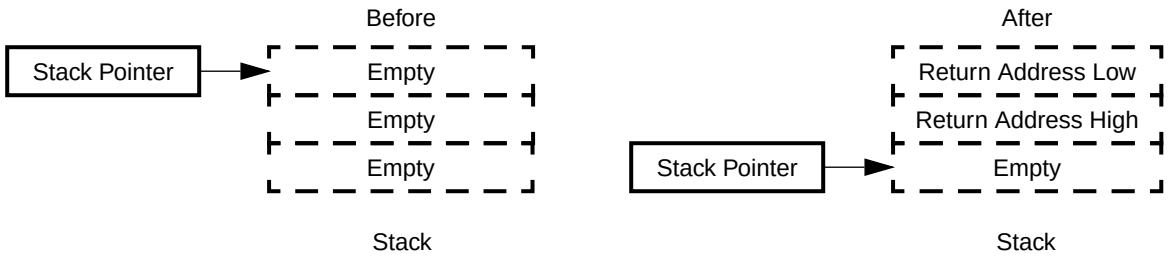
data memory. The data pointer (PTR) is loaded with the address of a byte of data in memory. To access the byte of data, the pointer can be used instead of using the address itself (LD A, [PTR]). This is particularly useful when consecutive locations are accessed. The data pointer can be incremented automatically after each access instead of specifying the address each time.

Stack

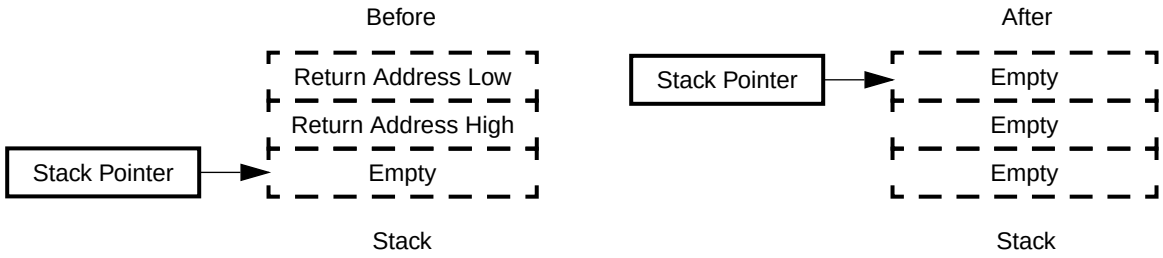
The stack is a section of memory used to store data and address values. Stacks operate in a LIFO (last in first out) manner. A stack pointer is used to address the stack. The stack pointer is a register used to keep the address of the last piece of data put on the stack. Usually the stack pointer is initialized to an address in memory. As data is put on the stack, the stack pointer moves up or down in memory. The microcontroller uses the stack to store return addresses during subroutine calls and interrupts. Some microcontrollers also store status information on the stack before responding to interrupts. The application program can use the stack to store data temporarily, especially data to be passed between subroutines. Not all stacks are user accessible.

Examples:

1. Stack operation as a result of executing Jump-to-Subroutine (JSR), or PUSH instructions.



2. Stack operation as, a result of executing Return-from-Subroutine (RET) or POP instructions.



1.7.3 Microcontroller CPU

The key function of a CPU is to perform instruction fetch/decode/execute.

Fetch — The program counter (PC) addresses a location in program memory containing an instruction. This instruction is latched into a special instruction register (IR). The PC is incremented to point to the next available instruction.

Decode — The instruction in the IR is decoded. The bits of the instruction relate to specific actions. Each bit or group of bits is used to determine the appropriate control signals to activate in order to cause the execution of the instruction.

Execute — The control signals go out to all parts of the microcontroller, causing the desired action to take place.

Arithmetic Logic Unit (ALU)

The arithmetic logic unit is a binary adder. It performs all the arithmetic and logic functions in the microcontroller. The destination for all such operations is the Accumulator (A). The two inputs to an operation are the accumulator and either an immediate data as specified by an instruction or, more commonly, the contents of data memory locations. The one-bit carry register (C register) sometimes is a third input to the ALU.

Status/Control Registers

The status and control registers are special-purpose registers used to store the state of the microcontroller. The control bits are manipulated by the user program to place the microcontroller and its peripherals in particular states. The status bits are read by the application program to inform microcontroller/user of the current state. Examples include:

1. A carry bit, which indicates whether the last operation performed by the ALU generated a carry.
2. Interrupt enable bit, which tells the microcontroller whether an interrupt is enabled.
3. Interrupt pending bit, which tells the program whether a particular interrupt occurred.

4. HALT bit, which tells microcontroller to stop all activities.
5. Timer run, which tells microcontroller to start the timer.

Accumulator vs Register

An accumulator-based microcontroller operates in a manner different from a register-based microcontroller. The difference is due to the different ALU architectures. The most common are:

1. Stack based: a no address machine
2. Accumulator based: a one-address machine
3. Register based: a two- or three-address machine

Another difference is due to the difference between the microcontroller architectures in terms of where numbers must be located in order for them to be operated on by the ALU of the microcontrollers.

Example: Adding two numbers

Adding two numbers in an accumulator-based machine requires that one of the numbers be located in the accumulator and the other one in data memory. The result of the addition is placed back in the accumulator.

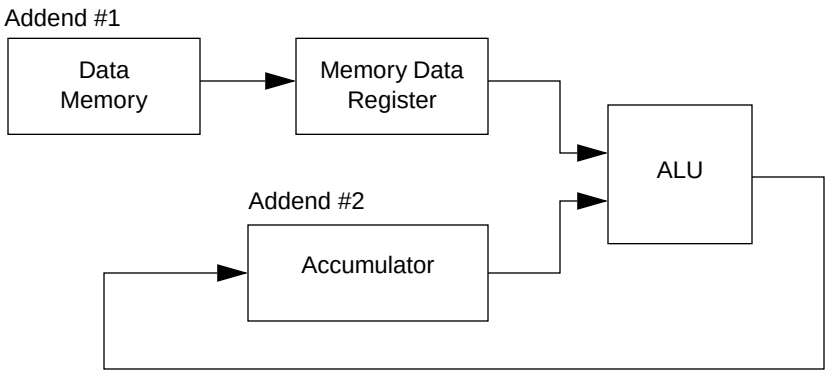


Figure 1-5 Adding Two Numbers Using Accumulator Based Machine

A register-based machine does not require that any of the addends be in the accumulator. Instead, the two numbers may be in registers in memory. The result may be specified to be placed in one of the registers being used in the addition, a third register, or the accumulator.

What's the difference? Is one method better than another?

Accumulator based machines require one of the operands to be located in the Accumulator. Therefore, only one operand location must be specified in instructions.

Register based machines do not require the operands to be in a particular location. Therefore, the location of both operands must be specified in instructions.

The decoding of register-based instructions and the fetching of operands generally requires more time than would be required to execute an accumulator-based instruction.

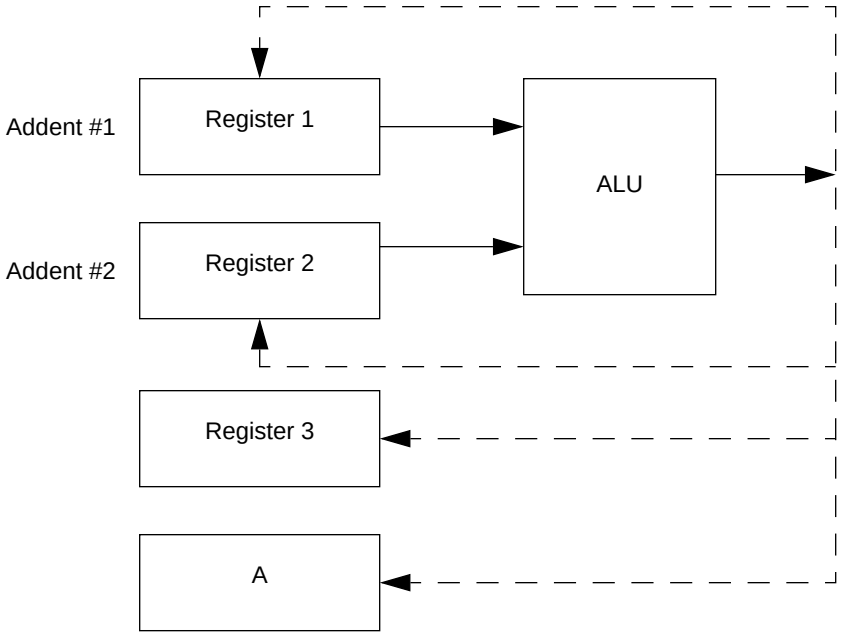


Figure 1-6 Adding Two Numbers Using Register Based Machine

However, the accumulator-based machine requires operands to be moved into the accumulator before instructions are executed. Which takes longer in the end? It depends on how much faster an accumulator-based machine can execute an instruction than a register-based machine.

Trade-off memory efficiency—An accumulator-based machine will generally have shorter opcodes than a register-based machine.

Interrupts

Interrupt is an event which causes the microcontroller to stop executing code in its normal sequence and instead respond to the occurrence of the event. The microcontroller performs the following steps when an interrupt occurs:

1. Stops execution of the "next" instruction.
2. Disables additional interrupts.
3. Saves current status of the microcontroller.
4. Jumps to a special interrupt handling routine.
5. Returns from the special interrupt handling routine.
6. Restores state of the microcontroller
7. Enables interrupts
8. Executes the "next" instruction.

Interrupt Types

External Interrupt	Interrupts which are generated by device/events outside of the microcontroller. An external interrupt can be latched or edge triggered.
Internal Interrupts	Interrupts which are generated by hardware within the microcontroller in response to certain events, such as timer overflow or underflow.
Software Traps	Interrupts caused by executing a particular instruction. Such an instruction may be a special instruction designed to cause a trap (INTR) or may be the result of an error in executing a normal instruction.
Maskable vs. Non-Maskable Interrupts	Non-Maskable interrupts are those interrupts which cannot be disabled by the software and therefore cannot be ignored, such as the Reset and Software Trap interrupts. Maskable interrupts, on the other hand, may be disabled by the software, such as timer overflow/underflow interrupts.. Enabling and disabling is generally accomplished by setting/resetting an enable bit.

1.7.4 Timing

The microcontroller uses the instruction cycle time as an internal timing reference. The instruction cycle time is the amount of time it takes for an instruction to be fetched, decoded, and executed. The instruction cycle time differs for various microcontrollers. It also differs for various instructions. Microcontroller manufacturers specify a minimum instruction cycle time. For example, for the COP8SAC7, the instruction cycle time t_c = 1 µS. This means that the fastest instructions are executed in 1 µS with the microcontroller operating at the maximum frequency. Slower instructions usually take some multiple of the instruction cycle clock.

The instruction cycle time is usually a division of the input clock frequency to the microcontroller. For example, divide by 10 is one possible factor. This means a 10 MHz input clock must be provided to generate a 1 µS instruction cycle time (1 MHz instruction cycle clock). The minimum instruction cycle time may be several instruction cycle clocks or only one instruction cycle clock, depending on the microcontroller. Limiting factors on instruction cycle time are:

1. Memory access time (memory speed)
2. Number of bytes per instruction
3. Width of data bus
4. Level of decoding required of instructions
5. Execution time

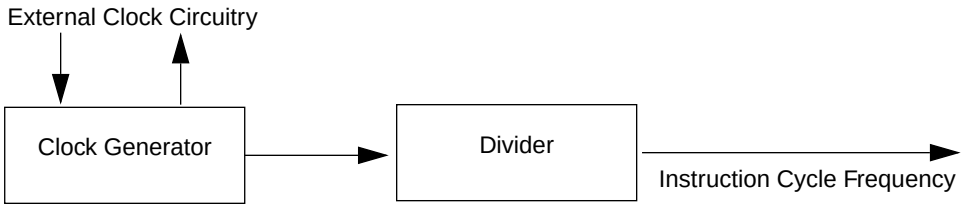


Figure 1-7 Clock Generation

1.7.5 Oscillator Circuits

Typically there are three types of clock oscillator options available: external oscillator, R/C oscillator, or crystal oscillator.

External Oscillator

An external square wave clock source is generated outside and presented to the microcontroller clock input pin. The clock source must meet the specified duty cycle, rise and fall times, and input level.

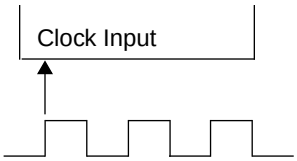


Figure 1-8 External Oscillator

R/C Oscillator

The R/C (resistor/capacitor) oscillator requires the use of external R/C components or R/C integrated on-chip. The oscillator frequency is a function of the resistance and capacitance values.

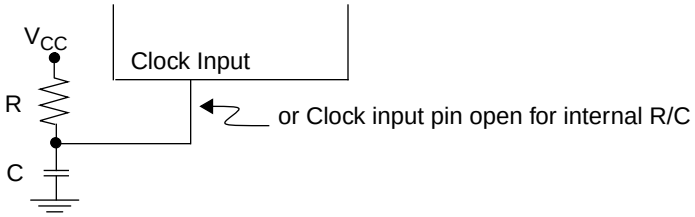


Figure 1-9 R/C Oscillator

Crystal Oscillator

The phase shift oscillator referred to as the Pierce Oscillator has many desirable characteristics. It provides a large output signal and drives the crystal at a low power level. The low power level leads to low power dissipation, especially at higher frequencies.

The circuit has good short-term stability, good waveforms at the crystal, a frequency which is independent of power supply and temperature changes, low cost and usable at any frequency. As compared with other oscillator circuits, this circuit is not disturbed very much by connecting a scope probe at any point in the circuit, because it is a stable circuit and has low impedance. This makes it easier to monitor the circuit without any major disturbance. The Pierce oscillator has one disadvantage. The amplifier used in the circuit must have high gain to compensate for gain losses in the circuitry surrounding the crystal.

Figure 1-10 shows the classic phase shift oscillator found not only on the COP8SAx7 but on most other microcontroller circuits. It is the simplest oscillator in terms of component complexity.

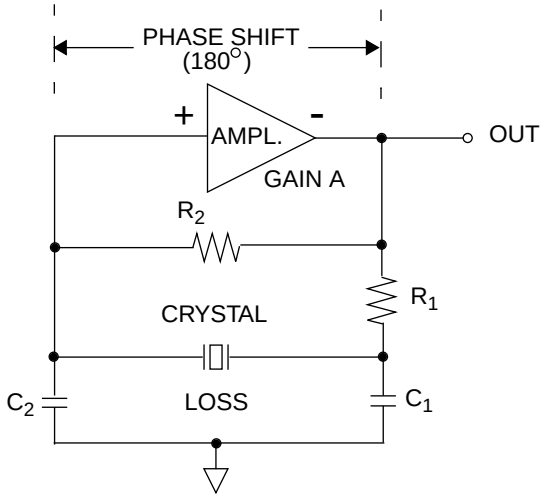


Figure 1-10 Phase Shift Oscillator

The Pierce is a series resonant circuit. For oscillation to occur, two criteria must be met:

1. The loop gain must be greater than one.
2. The phase shift around the loop must be 360°

The amplifier (intergrated on-chip) provides the desired gain and the 180° phase shift. The external C_1 and C_2 capacitors provide the additional 180° phase shift. The R_1 resistor is used at lower frequencies (typically lower than 2.5 MHz) to introduce losses around the loop and prevent the oscillator going into harmonic oscillation. The R_2 resistor is used to bias the amplifier for better and quicker oscillator start-up.

Inputs/Outputs

Inputs and Outputs allow the microcontroller to communicate with external devices.

Inputs — An outside device forces a microcontroller pin high or low. The logic level is read by the microcontroller as a single bit of incoming information.

Outputs	Microcontroller forces one of its pins high or low. The output voltage on the pin corresponds to a single bit of information.
Latches	Often used to store outgoing/incoming bits.
Ports	A port is a group of pins used for sending or receiving information. A port may have all outputs, all inputs, or a combination input/output pins. Some ports are bidirectional. Port pins may be used together for parallel communication or individually as control signals or serial communication.
Memory Mapped I/O or Ports	Each port (group of pins) is assigned an address as if it were a register (byte of data) in memory. Writing to an address assigned to a port causes the pins associated with the addressed port to be forced high or low according to the value written. If ports are not memory mapped, special I/O instructions have to be used to access them.
User-Configurable I/O	User-configurable ports or I/O may be set by the user to be inputs or outputs (some ports are fixed in hardware to be inputs or outputs). A configuration or control register is generally associated with a user-configurable port. Setting or resetting the bits in this special register configures the associated port pins the be inputs or outputs. Some user-configurable I/O ports only allow the user to select pins of a particular port to be all inputs or all outputs. Other devices allow the user to specify the configuration of each individual pin of a port. Some pins/ports have user selectable special functions.
Dedicated I/O	Dedicated I/O pins can only be used for one particular function. For example, the RESET pin can rarely be used for anything other than resetting the device. Other such pins might be the clock pin.

1.7.6 Instruction Set

Each microcontroller has a set of instructions. The user can organize particular instructions in a logical order to create a program. The microcontroller follows this program to perform a given task. The contents of a microcontroller instruction set vary with various microcontrollers.

Opcodes — Numeric codes in the instructions that represent the actual operation to be performed by the CPU are called operation codes (or opcodes for short). An opcode is actually a group of bits which tell the microcontroller to perform a particular operation (a code word for a given operation). For example; 64 might mean clear the accumulator.

Mnemonics — Names assigned to particular operations are called mnemonics. Each mnemonic is associated with a particular opcode. A user may specify a mnemonic such as ADD rather than the opcode 84. The mnemonic is later translated into opcodes which the microcontroller can decode. Using mnemonics makes writing code easier. For example, the mnemonics LD, ADD, JP, CLRA represent the operation load, add, jump, and clear accumulator, respectively.

Categories of Instructions

1. Arithmetic/logic/shift (ADD/AND/RRC)
2. Transfer of control (JP, JMP)
3. Memory reference (LD)
4. Bit manipulation (SBIT)
5. Stack control (PUSH)
6. Conditional/Test (IFEQ)

Opcode fields/Multiple Byte Opcodes‘

An opcode field is a group of bits within an opcode used to represent one specific part of an instruction. Typically the bits in an opcode are broken into groups or fields. Most instructions usually consist of at least two fields: opcode field and address field.

Addressing Modes

Implied — The location/value of the operand(s) is specified implicitly in the definition of the instruction.

Example: CLR A, SC

Immediate — Data to be used as an operand is specified directly in the instruction.

Example: LD A, #023 or AND A, #0FF

Memory Direct	The memory address of the operand(s) is specified in the instruction. Example: LD A, 00
Register Indirect	A register which contains the address of the operand in memory is specified in the instruction. Example: LD A, [B]
Post-Increment or Post-Decrement	Similar to register indirect, a register containing the address of the operand in memory is specified in the instruction. However, the contents of the register (the memory address) is automatically incremented or decremented after the operand is fetched from the specified location. Example: LD A, [B+] or LD A, [B-]
Indexed Addressing	Similar to register indirect except an index number is also specified in the instruction. This index number/displacement is added to the contents of a base register to form the actual address of the operand. The base register may be either implied in the instruction or given as an additional argument. This addressing mode is used to access entries in a table. The base address represents the starting location of the table in memory. The displacement represents the distance between the start of the table and the desired element in the table. Example: LD A, 12[B] or ADD A, 14[R0]

1.7.7 Programming

Routines/Subroutines

A routine is a segment of code which performs a specific function. A subroutine is a segment of code which performs a specific part of function. A subroutine is usually called by several different routines or called several times from a single routine.

Stack Manipulation

On some microcontrollers, the stack is assigned a particular segment of memory and the microcontroller initializes the stack pointer to a specific address upon reset. This is referred to as stack initializing by hardware. There are also cases where the stack pointer is initialized the user program (software) at the beginning of the program to point to a segment of available memory. The stack may be addressed by instructions such as PUSH and POP. The PUSH instruction places a piece of data on the stack and increments or

decrements the stack pointer. The POP instruction does the opposite; it removes a piece of data from the stack and decrements or increments the stack pointer.

Interrupt Routines

There are two methods to get the interrupt routines:

1. Without a Vector Table

   Polling—All interrupts cause an immediate jump to a single location in memory. The user program must poll all interrupt pending bits to determine the cause of the interrupt. Once the cause is determined, the user program may jump to an appropriate interrupt handling routine.

   Preset Location—Each interrupt causes the microcontroller to jump to a different place in memory. The location is specified by the microcontroller designer. The user program must store the interrupt handling routine at the specified location.

2. With a Vector Table

   A vector table is a list of start addresses for each of the program's interrupt routines. This table is used by the microcontroller to determine where to jump when a particular interrupt occurs.

   Direct Jump—The microcontroller may automatically read the vector table when an interrupt occurs and jump directly to the specified address.

   VIS Instruction—The microcontroller may jump to a single location in memory for all interrupts. The user program may call a special instruction (VIS) to cause the microcontroller to jump to the specified address.

Context Switching

Saving and restoring of the microcontroller state during interrupts or when switching between different tasks.

Example of an interrupt routine:

```
INTERRUPT;
PUSH        A              ; Save accumulator
PUSH        CNTRL          ; Save control register
PUSH        PSW            ; Save process status word
IFBIT       0, IPND        ; If timer interrupt pending
JP          TIMERINT       ; then jump to timer interrupt
IFBIT       1, IPND        ; If external interrupt pending
JP          EXTERNALINT    ; then jump to external interrupt
IFBIT       2, IPND        ; If serial interrupt pending
JP          SERIALINT      ; then jump to serial interrupt
EXIT                       ; Exit from interrupt
LD          IPND, #00      ; Clear pending bits
POP         PSW            ; Restore process status word
POP         CNTRL          ; restore control register
POP         A              ; Restore accumulator
RETI                       ; Return from interrupt
```

Assembler

1. Assembler

   The assembler is a software program that converts a source program into an object file. In other words, it converts ASCII representation of instructions to binary representation.

2. Assembler Inputs/Outputs

   Source File - ASCII file containing a software program written in instruction mnemonics (symbolic language).

   Object File - "Executable" file containing instruction opcodes (machine language).

   Listing File - An ASCII file which lists each location in memory and the instruction opcode associated with it. It usually contains assembly error messages. This listing is useful for finding errors, determining size of code, finding location of segments of code, and checking for proper assembling of instructions.

   Symbol Table - Table of constants and symbols with their associated values. It is useful for determining the locations of subroutines in the code.

3. Assembler Directives

   An assembler directive is an instruction for the assembler in the source file. Different assemblers have different sets of directives. Examples:

   `.CHIP` Specifies the particular device for which the code is written

   `.END` Specifies the end of the code

   `.=NUM` Sets the location counter to NUM. The location counter is used by the assembler to store the address of the memory location where the current instruction is being stored. This directive allows the user to specify a particular location for a given segment of code.

   `.BYTE` Tells the assembler to store the following 8-bit value in program memory. Tells the assembler not to interpret it as an instruction. This is useful for storing a table of data.

Linker

1. Linker

   The linker is a software program that combines separate object files produced by the assembler into a single object file. The linker allows the user to create separate code modules for different sections of code. It also allows the user to create a library of functions which may be used in a number of different programs.

2. Linker Inputs/Outputs

   Object File - Same as above but each object file contains only a section/module of the entire program.

   Load Map - Specifies how memory is allocated. A typical load map might contain the following sections:

   Range Definitions: Shows memory ranges with assigned names

   Memory Order Map: Shows starting and ending ranges of each contiguous block of memory

   Memory Type Map: Shows how address space is allocated to different types of memory

   Total Memory Map: Shows allocation of all RAM and ROM/EPROM combined

   Section Table: Shows starting and ending address of each linked module

   Symbol Table: Same as section table

   Cross-Reference Table: Similar to a symbol table, it lists all symbols and tables with their associated values. In addition, it lists all the locations where the symbol and tables are being used.

Compiler

1. Compiler

   Translates programs written in high-level languages such as C into equivalent assembly language programs. The high-level language is usually a modified high-level language which allows a combination of the standard high-level language instructions and assembly language instructions. A compiler increases ease of programming but decreases code efficiency.

2. Compiler Inputs/Outputs

   High-level language Source File - A file containing microcontroller program written in a modified high-level language.

   Assembly Out File - A file containing assembly language source code produced from the high-level language source file.

Simulator

A simulator is software program/model which acts like a hardware device. Code written for the device is executed in the software model exactly as it would be executed in the device. Key features of a simulator include:

1. Can execute object code
2. Can perform Single Step/Breakpoint/Go operations
3. Can display simulated internal device registers and allow them to be modified
4. Can display program and data memory contents and allow them to be modified
5. Requires no hardware

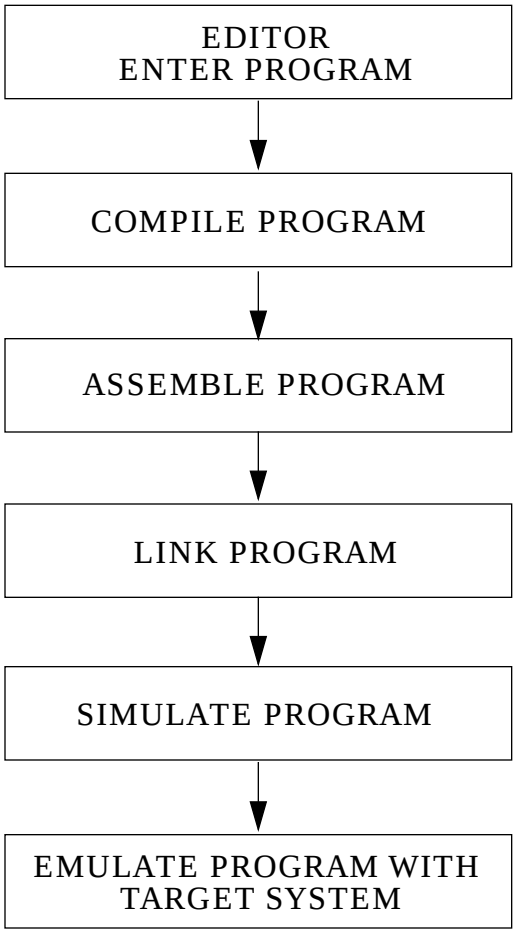
Hardware - Emulators

An emulator is a hardware model of a device with additional circuitry to allow the user to perform special functions such as starting and stopping code execution. An emulator can be used for examining code execution and code debugging. The hardware emulator is plugged into the actual target system where the microcontroller device would normally be plugged in. Thus, the emulator, tests both the internal operation of the microcontroller and its I/O functions.

1. Can execute object code
2. Can display data and program memory contents and allow them to be modified
3. Can display internal registers and allow them to be modified
4. Can perform Single Step/Breakpoint/Go operations
5. Interfaces directly to external hardware
6. Can run code in real-time
7. Can trace code execution
8. Emulates AC and DC electrical characteristics

Code Development Path

Following is the typical path for developing microcontroller programs:



Chapter 2

COP8SAx7 MICROCONTROLLER

2.1 INTRODUCTION

The COPSAx7 OTP microcontrollers are members of the COP8™ feature family using an 8-bit single chip core architecture. These devices are fabricated in National Semiconductor's high-density EPROM process, and offered in a variety of packages, temperature ranges, and voltage ranges to satisfy a wide variety of applications.

Key features include an 8-bit memory-mapped architecture, a 16-bit timer/counter with two associated 16-bit registers supporting three modes (Processor Independent PWM generation, External Event counter, and Input Capture capabilities), two power saving HALT/IDLE modes with a multi-sourced wakeup/interrupt capability, on-chip R/C oscillator high-current outputs, user selectable options such as WATCHDOG™, Oscillator configuration, and power-on reset.

2.2 KEY FEATURES

- Low cost 8-bit OTP (one-time programmable)microcontroller
- OTP program space with read/write protection
- Quiet design (low radiated emissions)
- Multi-Input wakeup pins with optional interrupts (4 to 8 pins)
- 8 bytes of user storage space in EPROM
- User selectable clock options
  - — Crystal/resonator oscillator
  - — Crystal/resonator oscillator with on-chip bias resistor
  - — External oscillator
  - — Internal R/C oscillator
- Internal power-on reset - user selectable
- WATCHDOG and clock monitor logic - user selectable
- Up to 12 high current outputs

Device	EPROM	RAM	Package and I/O	
			Package Types	Number of I/O
COP8SAC7	4k	128	20 DIP/SO 28 DIP/SO 40 DIP 44 PLCC/PQFP	16 24 36 40
COP8SAB7	2k	128	20 DIP/SO 28 DIP/SO	16 24
COP8SAA7	1k	64	16 DIP/SO 20 DIP/SO 28 DIP/SO	12 16 24

2.2.1 CPU Features

- Versatile, easy-to-use instruction set
- 1 µs instruction cycle time
- Eight multi-source vectored interrupts
  - — External Interrupt
  - — Idle Timer T0
  - — One Timer (with 2 Interrupts)
  - — MICROWIRE/PLUS Serial Interface
  - — Multi-Input Wake Up
  - — Software Trap
  - — Default VIS (default interrupt)
- 8-bit stack pointer SP (stack in RAM)
- Two 8-bit register indirect data memory pointers
- True bit manipulation
- Memory-mapped I/O
- BCD arithmetic instructions

2.2.2 Peripheral Features

- Multi-Input Wakeup Logic
- One 16-bit timer with two 16-bit registers supporting:
  - — Processor Independent PWM mode
  - — External Event counter mode
  - — Input Capture mode
- Idle Timer
- MICROWIRE/PLUS™ Serial Interface (SPI Compatible)

2.2.3 I/O Features

- Software selectable I/O options
  - — TRI-STATE® output
  - — Push-pull output
  - — Weak pull up input
  - — High impedance input
- Schmitt trigger inputs on ports G and L
- Up to 12 high-current outputs
- Pin efficient (i.e. 40 pins in 44-pin package are devoted to useful I/O)

2.2.4 Fully Static CMOS Design

- Low current drain (typically < 4 µA)
- Single-supply operation: 2.7V to 5.5V
- Two power saving modes: HALT and IDLE

2.2.5 Temperature Ranges

0˚C to +70˚C, –40˚C to +85˚C, and –40˚C to +125˚C

2.2.6 Development Support

- Windowed packages for DIP and PLCC
- Real-time emulation and full program debug offered by MetaLink Development System

2.3 BLOCK DIAGRAM

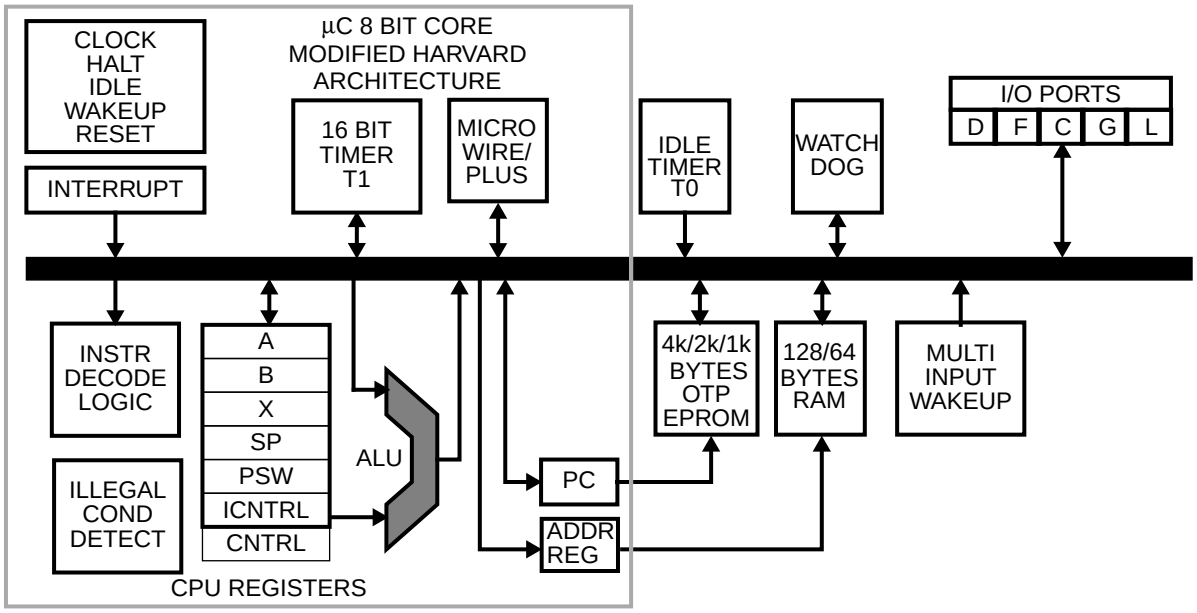


Figure 2-1 COP8SAx7 Block Diagram

2.4 ARCHITECTURE

The COPSAx7 family is based on a modified Harvard architecture, which allows data tables to be accessed directly from program memory. This is very important with modern microcontroller-based applications, since program memory is usually ROM or EPROM, while data memory is usually RAM. Consequently data tables usually need to be contained in ROM or EPROM, so they are not lost when the microcontroller is powered down. In a modified Harvard architecture, instruction fetch and memory data transfers can be overlapped with a two stage pipeline, which allows the next instruction to be fetched from program memory while the current instruction is being executed using data memory. This is not possible with a Von Neumann single-address bus architecture.

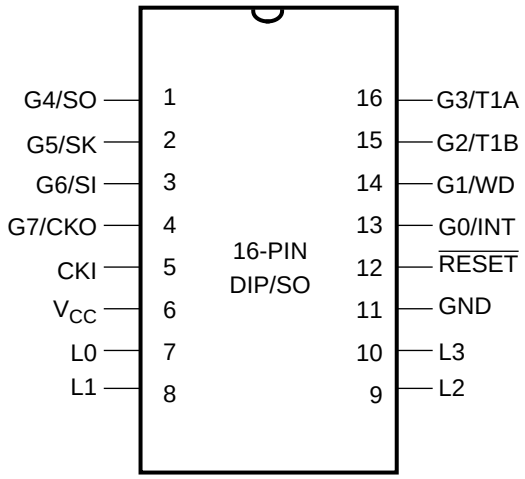
The COPSAx7 family supports a software stack scheme that allows the user to incorporate many subroutine calls. This capability is important when using High Level Languages. With a hardware stack, the user is limited to a small fixed number of stack levels.

2.5 PACKAGING/PIN EFFICIENCY

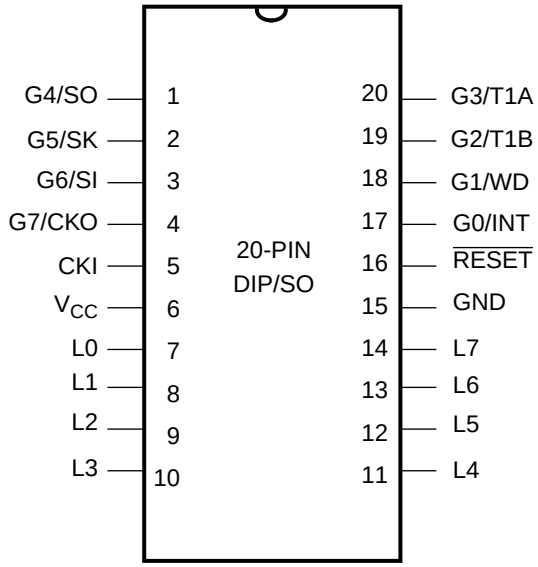
Real estate and board configuration considerations demand maximum space and pin efficiency, particularly given today's high integration and small product form factors. Microcontroller users try to avoid using large packages to get the I/O needed. Large packages take valuable board space and increases device cost, two trade-offs that microcontroller designs can ill afford.

The COP8 family offers a wide range of packages and do not waste pins: up to 90.9% (or 40 pins in the 44 -pin package) are devoted to useful I/O.

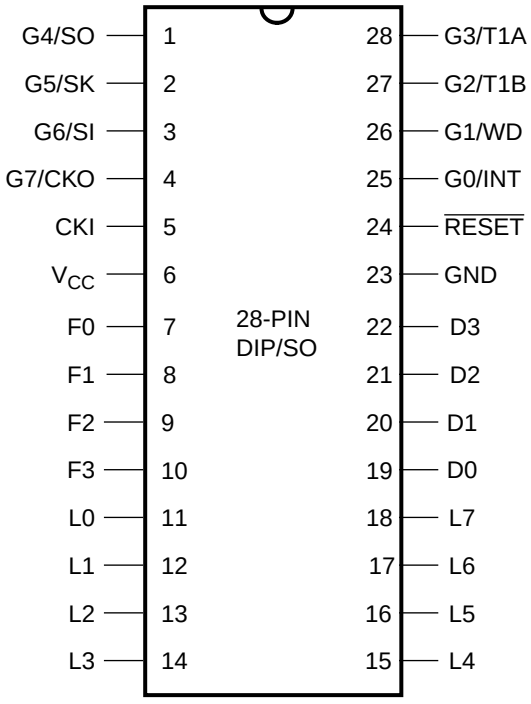
2.6 CONNECTION DIAGRAMS



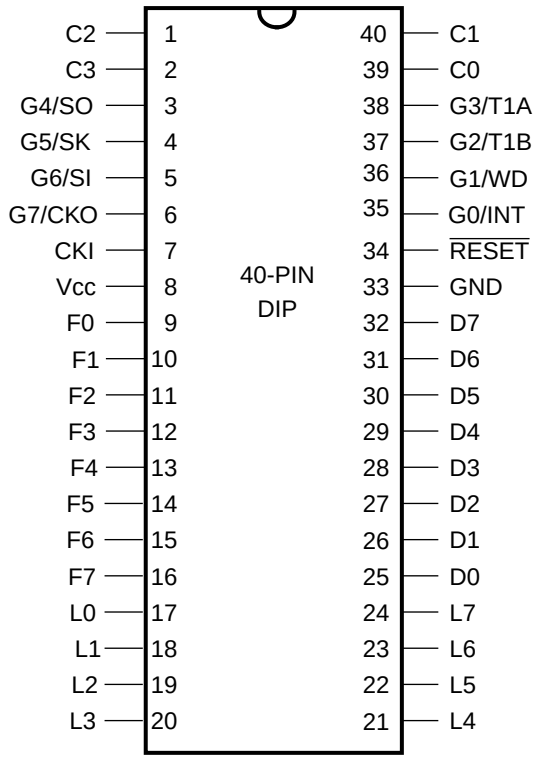
Top View



Top View



Top View



Top View

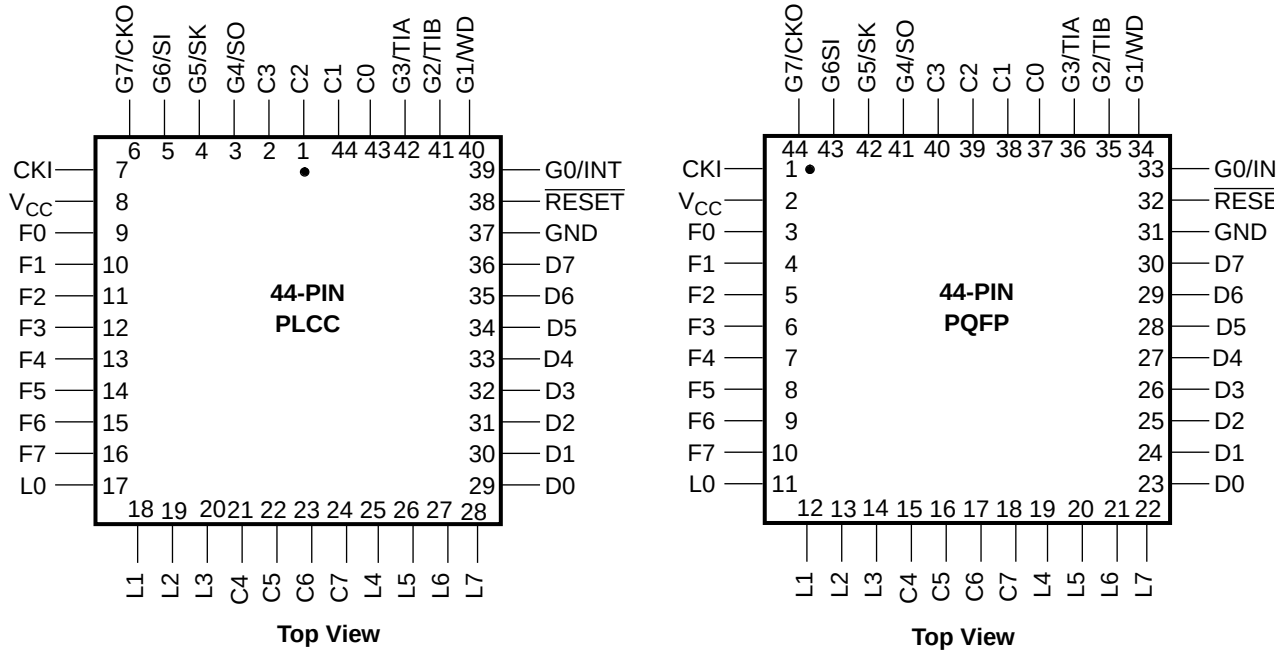


Figure 2-2 Connection Diagrams

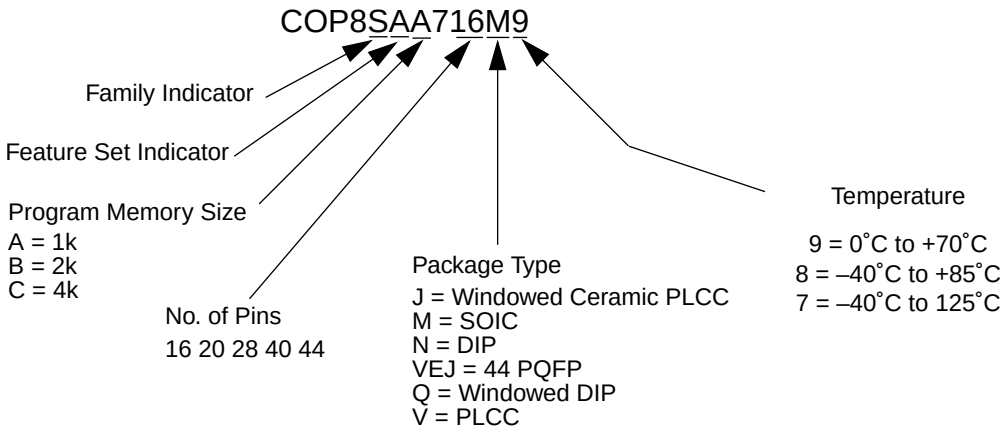


Figure 2-3 Part Numbering Scheme

2.6.1 ORDERING INFORMATION

	1k EPROM		2k EPROM		4k EPROM		4k EPROM	
Temperature	Order Number	Package	Order Number	Package	Order Number	Package	Windowed Device Order Number	Package
0˚C to +70˚C	COP8SAA716M9	16M						
	COP8SAA720M9	20M	COP8SAB720M9	20M	COP8SAC720M9	20M		
	COP8SAA728M9	28M	COP8SAB728M9	28M	COP8SAC728M9	28M		
	COP8SAA716N9	16N						
	COP8SAA720N9	20N	COP8SAB720N9	20N	COP8SAC720N9	20N	COP8SAC720Q9	20Q
	COP8SAA728N9	28N	COP8SAB728N9	28N	COP8SAC728N9	28N	COP8SAC728Q9	28Q
					COP8SAC740N9	40N	COP8SAC740Q9	40Q
					COP8SAC744V9	44V	COP8SAC744J9	44J
					COP8SAC7VEJ9	44PQFP		
–40˚C to +85˚C	COP8SAA716M8	16M						
	COP8SAA720M8	20M	COP8SAB720M8	20M	COP8SAC720M8	20M		
	COP8SAA728M8	28M	COP8SAB728M8	28M	COP8SAC728M8	28M		
	COP8SAA716N8	16N						
	COP8SAA720N8	20N	COP8SAB720N8	20N	COP8SAC720N8	20N		
	COP8SAA728N8	28N	COP8SAB728N8	28N	COP8SAC728N8	28N		
					COP8SAC740N8	40N		
					COP8SAC744V8	44V		
					COP8SAC7VEJ8	44PQFP		
–40˚C to +125˚C					COP8SAC720M7	20M		
					COP8SAC728M7	28M		
					COP8SAC720N7	20N		
					COP8SAC728N7	28N		
					COP8SAC740N7	40N		
					COP8SAC744V7	44V		
					COP8SAC7VEJ7	44PQFP		

2.7 PIN DESCRIPTIONS

COPSAx7 I/O structure minimizes external component requirements. Software-switchable I/O enables designers to reconfigure the microcontroller's I/O functions with a single instruction. Each individual I/O pin can be independently configured as an output pin low, an output high, an input with high impedance or an input with a weak pull-up device. A typical example is the use of I/O pins as the keyboard matrix input lines. The input lines can be programmed with internal weak pull-ups so that the input lines read logic high when the keys are all up. With a key closure, the corresponding input line will read a logic zero since the weak pull-up can easily be overdriven. When the key is released, the internal weak pull-up will pull the input line back to logic high. This flexibility eliminates the need for external pull-up resistors. The High current options are available for driving LEDs, motors and speakers. This flexibility helps to ensure a cleaner design, with less external components and lower costs. Below is the general description of all available pins.

V_{CC} and GND are the power supply pins. All V_{CC} and GND pins must be connected.

CKI is the clock input. This can come from the Internal R/C oscillator, external, or a crystal oscillator (in conjunction with CKO). See Oscillator Description section.

$\overline{\text{RESET}}$ is the master reset input. See Reset description section.

The device contains four bidirectional 8-bit I/O ports (C, G, L and F), where each individual bit may be independently configured as an input (Schmitt trigger inputs on ports L and G), output or TRI-STATE under program control. Three data memory address locations are allocated for each of these I/O ports. Each I/O port has two associated 8-bit memory mapped registers, the CONFIGURATION register and the output DATA register. A memory mapped address is also reserved for the input pins of each I/O port. (See the memory map for the various addresses associated with the I/O ports.) Figure 2-4 shows the I/O port configurations. The DATA and CONFIGURATION registers allow for each port bit to be individually configured under software control as shown below:

CONFIGURATION Register	DATA Register	Port Set-Up
0	0	Hi-Z Input (TRI-STATE Output)
0	1	Input with Weak Pull-Up
1	0	Push-Pull Zero Output
1	1	Push-Pull One Output

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. The 16-pin device does not have a full complement of Port L pins. The unavailable pins are not terminated. A read operation these unterminated pins will return unpredictable values. To minimize current drain, the unavailable pins must be programmed as outputs.

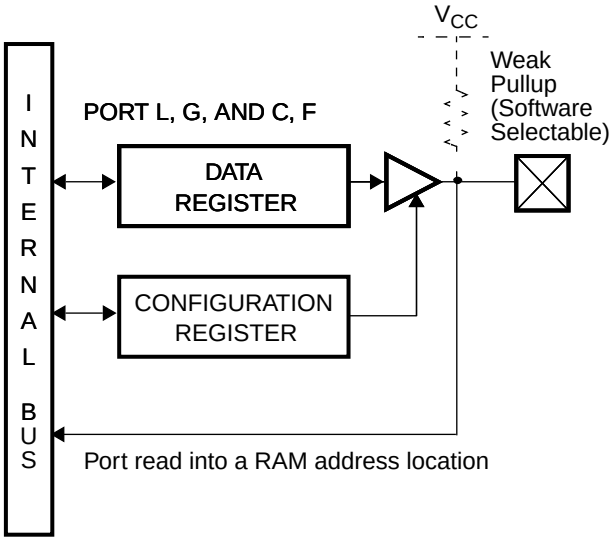


Figure 2-4 I/O Port Configurations

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 WDOUT WATCHDOG output with weak pullup if WATCHDOG feature is selected by the ECON register. The pin is a general purpose I/O if WATCHDOG feature is not selected.** If WATCHDOG feature is selected, bit 1 of the Port G configuration and data register does not have any effect on Pin G1 setup. Pin G7 is either input or output depending on the oscillator option selected. With the crystal oscillator option selected, G7 serves as the dedicated output pin for the CKO clock output. With the internal R/C or the external oscillator option selected, G7 serves as a general purpose Hi-Z input pin and is also used to bring the device out of HALT mode with a low to high transition on G7. There are two registers associated with Port G, a data register and a configuration register. Using these registers, each of the 5 I/O pins (G0, G2-G5) can be individually configured under software control.

Since G6 is an input only pin and G7 is the dedicated CKO clock output pin (crystal clock option) or general purpose input (R/C or external clock option), 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:

- G0 INTR (External Interrupt Input)
- G2 T1B (Timer T1 Capture Input)
- G3 T1A (Timer T1 I/O)
- G4 SO (MICROWIRE Serial Data Output)
- G5 SK (MICROWIRE Serial Clock)
- G6 SI (MICROWIRE Serial Data Input)

Port G has the following dedicated functions:

- G1 WDOUT WATCHDOG and/or Clock Monitor if WATCHDOG enabled, otherwise it is a general purpose I/O
- G7 CKO Oscillator dedicated output or general purpose input

Port C is an 8-bit I/O port. The 40-pin device does not have a full complement of Port C pins. The unavailable pins are not terminated. A read operation on these unterminated pins will return unpredictable values. Only the COP8SAC7 device contains Port C. The 20/28 pin devices do not offer Port C. On these devices, the associated Port C Data and Configuration registers should not be used.

Port F is an 8-bit I/O port. The 28--pin device does not have a full complement of Port F pins. The unavailable pins are not terminated. A read operation on these unterminated pins will return unpredictable values.

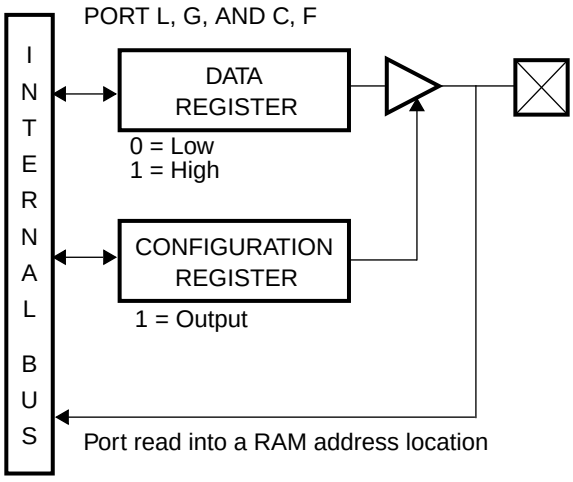


Figure 2-5 I/O Port Configurations–Output Mode

Port D is an 8-bit output port that is preset high when $\overline{\text{RESET}}$ goes low. The user can tie two or more D port outputs (except D2) together in order to get a higher drive.

NOTE: Care must be exercised with the D2 pin operation. At RESET, the external loads on this pin must ensure that the output voltages stay above 0.7 V_{CC} to prevent the chip from entering special modes. Also keep the external loading on D2 to less than 1000 pF.

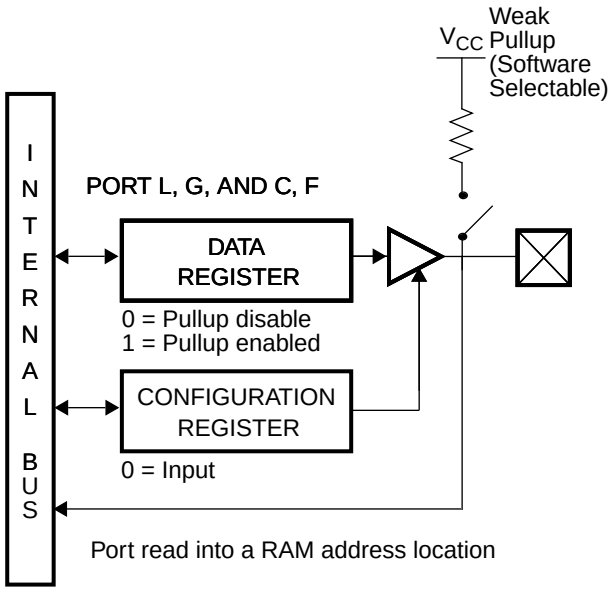


Figure 2-6 I/O Port Configurations–Input Mode

2.8 FUNCTIONAL DESCRIPTION

The architecture of the device is a modified Harvard architecture. With the Harvard architecture, the program memory EPROM is separated from the data store memory (RAM). Both EPROM and RAM have their own separate addressing space with separate address buses. The architecture, though based on the Harvard architecture, permits transfer of data from EPROM to RAM.

2.8.1 CPU Registers

The CPU can do an 8-bit addition, subtraction, logical or shift operation in one instruction (tc) cycle time.

There are six CPU registers:

A is the 8-bit Accumulator Register

PC is the 15-bit Program Counter Register

PU is the upper 7 bits of the program counter (PC)

PL is the lower 8 bits of the program counter (PC)

B is an 8-bit RAM address pointer, which can be optionally post auto incremented or decremented.

X is an 8-bit alternate RAM address pointer, which can be optionally post auto incremented or decremented.

SP is the 8-bit stack pointer, which points to the subroutine/interrupt stack (in RAM). With reset the SP is initialized to RAM address 02F Hex (devices with 64 bytes of RAM), or initialized to RAM address 06F Hex (devices with 128 bytes of RAM).

All the CPU registers are memory mapped with the exception of the Accumulator (A) and the Program Counter (PC).

2.8.2 Program Memory

The program memory consists of 1024, 2048, or 4096 bytes of EPROM. Table 2-1 shows the program memory sizes for the different devices. These bytes may hold program instructions or constant data (data tables for the LAID instruction, jump vectors for the JID instruction, and interrupt vectors for the VIS instruction). The program memory is addressed by the 15-bit program counter (PC). All interrupts in the device vector to program memory location 0FF Hex. The contents of the program memory read 00 Hex in the erased state.

Table 2-1 Program/Data Memory Sizes

Device	Program Memory (Bytes)	Data Memory (Bytes)	User Storage (Bytes)
COP8SAA7	1024	64	8
COP8SAB7	2048	128	8
COP8SAC7	4096	128	8

2.8.3 Data Memory

The data memory address space includes the on-chip RAM and data registers, the I/O registers (Configuration, Data and Pin), the control registers, the MICROWIRE/PLUS SIO shift register, and the various registers, and counters associated with the timers (with the exception of the IDLE timer). Data memory is addressed directly by the instruction or indirectly by the B, X and SP pointers.

The data memory consists of 64 or 128 bytes of RAM. Table 2-1 shows the data memory sizes for the different devices. Fifteen bytes of RAM are mapped as “registers” at addresses 0F0 to 0FE Hex. These registers can be loaded immediately, and also decremented and tested with the DRSZ (decrement register and skip if zero) instruction. The memory pointer registers X, SP and B are memory mapped into this space at address locations 0FC to 0FE Hex respectively, with the other registers (except 0FF) being available for general usage. Address location 0FF is reserved for future RAM expansion. If compatibility with future devices (with more RAM) is not desired, this location can be used as a general purpose RAM location.

The instruction set permits any bit in memory to be set, reset or tested. All I/O and registers (except A and PC) are memory mapped; therefore, I/O bits and register bits can be directly and individually set, reset and tested. The accumulator (A) bits can also be directly and individually tested.

RAM contents are undefined upon power-up.

2.8.4 ECON (EPROM Configuration) Register

The ECON register is used to configure the user selectable clock, security, RAM size, power-on reset, WATCHDOG, and HALT options. The register can be programmed and read only in EPROM programming mode. Therefore, the register should be programmed at the same time as the program memory. The contents of the ECON register shipped from the factory read 00 Hex (windowed device) or 80 Hex (OTP device).

The format of the ECON register is as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	POR	SECURITY	CKI 2	CKI 1	WATCDOG	Reserved	HALT

Bit 7 = x This is for factory test. The polarity is "Don't Care."

Bit 6 = 1 Power-on reset enabled.

= 0 Power-on reset disabled.

Bit 5 = 1 Security enabled. EPROM read and write are not allowed.

= 0 Security disabled. EPROM read and write are allowed.

Bits 4,3

= 0, 0 External CKI option selected.
G7 is available as a HALT restart and/or general purpose input. CKI is clock input.

= 0,1 R/C oscillator option selected.
G7 is available as a HALT restart and/or general purpose input. CKI clock input. Internal R/C components are supplied for maximum R/C frequency.

= 1,0 Crystal oscillator with on-chip crystal bias resister disabled. G7 (CKO) is the clock generator output to crystal/resonator.

= 1, 1 Crystal oscillator with on-chip crystal bias resister enabled. G7 (CKO) is the clock generator output to crystal/resonator.

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 = Reserved

Bit 0 = 1 HALT mode disabled.

= 0 HALT mode enabled.

2.8.5 User Storage Space In EPROM

There are 8 bytes of user storage space in the EPROM that are not read protected by the security bit in the ECON register. When the security bit in the ECON register is set, data in User Storage Space is write-enabled, and read-enabled. This allows the user to read and write this information while still protecting the main EPROM from tampering.

The 8 bytes of user storage space are outside the normal address range of the device, and cannot be accessed by software. This allows for the storage of non-secure information. Typical uses of this are serial numbers, date codes, copyright informations, software version, or lot numbers.

To place information into this area, the user can place the following in the assembly file:

1. Place data in the 8-bytes of user storage space

```
Data is 1 2 3 4 5 6 7 8
.USER=0x01 0x02 0x03 m0x04 0x05 0x06 0x07 0x08
```

2. Place assembly date in 8-bytes of user storage space

```
Date is in format DD MM YY 00 HH MM SS (all information is
in Hex)
.USER=ASS_DATE
```

3. Place programming date in 8-bytes of user storage space

```
Date is in format DD MM YY 00 HH MM SS (all information is
in Hex)
.USER=PRG_DATE
```

4. To place data in both memory space (for example, 4F9-4FF) to be read out using the LAID instruction, and the user storage space

```
Data is 1 2 3 4 5 6 7 8
.=0x4F9
.BYTE 0x01 0x02 0x03 m0x04 0x05 0x06 0x07 0x08
.USER=0x01 0x02 0x03 m0x04 0x05 0x06 0x07 0x08
```

NOTE: Not all programmers support the PRG_Date option. Other serialization schemes are currently being worked on with device programming manufactures. Place contact your device programmer supplier or National for more information.

2.8.6 OTP Security

The device has a security feature, when enabled, that prevents external reading of the OTP program memory. The security bit in the ECON register determines, whether security is enabled or disabled. If the security feature is disabled, the contents of the internal EPROM may be read.

If the security feature is enabled, then any attempt to externally read the contents of the EPROM will result in the value FF Hex being read from all program locations. In addition, with the security feature enabled, the write operation to the EPROM program memory and ECON register is inhibited. The ECON register is readable regardless of the state of the security bit.

If security is being used, it is recommended that all other bits in the ECON register be programmed first. Then the security bit can be programmed.

2.8.7 Reset

The device is initialized when the $\overline{\text{RESET}}$ pin is pulled low or the On-chip Power-On Reset is enabled.

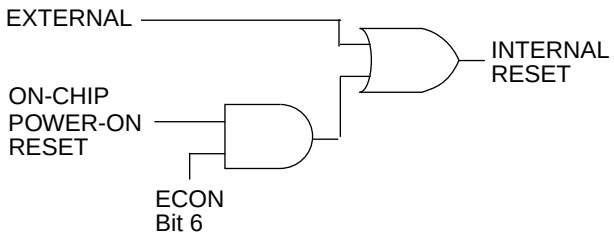


Figure 2-7 Reset Logic

The following occurs upon initialization:

Port L: TRISTATE

Port C: TRISTATE

Port G: TRISTATE

Port F: TRISTATE

Port D: HIGH

PC: CLEARED to 0000

PSW, CNTRL and ICNTRL registers: CLEARED

SIOR:

UNAFFECTED after RESET with power already applied

RANDOM after RESET at power-on

T1CNTRL: CLEARED

Accumulator, Timer 1:

RANDOM after RESET with crystal clock option (power already applied)

UNAFFECTED after RESET with R/C clock option (power already applied)

RANDOM after RESET at power-on

WKEN, WKEDG: CLEARED

WKPND: RANDOM

SP (Stack Pointer):

Initialized to RAM address 02F Hex (devices with 64 bytes of RAM), or initialized to RAM address 06F Hex (devices with 128 bytes of RAM).

B and X Pointers:

UNAFFECTED after RESET with power already applied

RANDOM after RESET at power-on

RAM:

UNAFFECTED after RESET with power already applied

RANDOM after RESET at power-on

WATCHDOG (if enabled):

The device comes out of reset with both the WATCHDOG logic and the Clock Monitor detector armed, with the WATCHDOG service window bits set and the Clock Monitor bit set. The WATCHDOG and Clock Monitor circuits are inhibited during reset. The WATCHDOG service window bits being initialized high default to the maximum WATCHDOG service window of 64k t_C clock cycles. The Clock Monitor bit being initialized high will cause a Clock Monitor error following reset if the clock has not reached the minimum specified frequency at the termination of reset. A Clock Monitor error will cause an active low error output on pin G1. This error output will continue until 16 t_C–32 t_C clock cycles following the clock frequency reaching the minimum specified value, at which time the G1 output will go high.

External Reset

The $\overline{\text{RESET}}$ input when pulled low initializes the device. The $\overline{\text{RESET}}$ pin must be held low for a minimum of one instruction cycle to guarantee a valid reset. During Power-Up initialization, the user must ensure that the $\overline{\text{RESET}}$ pin is held low until the device is within the specified V_{CC} voltage. An R/C circuit on the $\overline{\text{RESET}}$ pin with a delay 5 times (5x) greater than the power supply rise time or 15 µsec whichever is greater, is recommended. Reset should also be wide enough to ensure crystal start-up upon Power-Up.

$\overline{\text{RESET}}$ may also be used to cause an exit from the HALT mode.

With a slowly rising power supply, the device may start running before VCC is within the guaranteed range. In this case, the user must provide an external RC network and a diode shown in Figure 2-8.

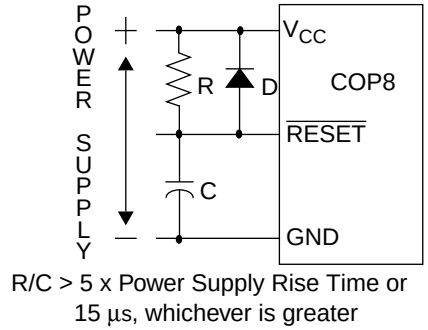


Figure 2-8 Reset Circuit Using External Reset

The external RC network is there to hold the RESET pin below VIL until VCC reaches at least VCC (min). The desired response is shown in Figure 2-9.

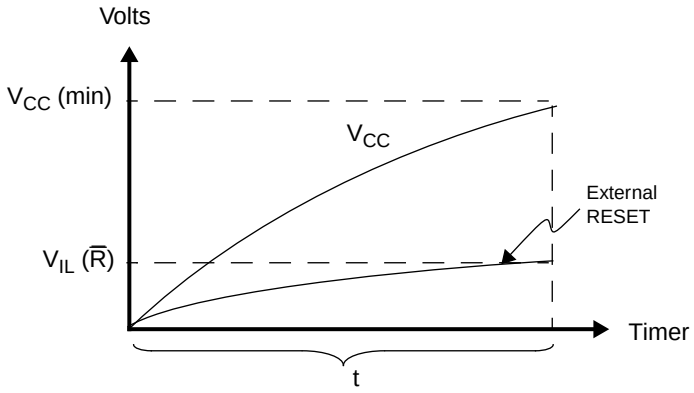


Figure 2-9 Dddesired Reset Response Time

On-chip Power-On Reset

The on-chip reset circuit is selected by a bit in the ECON register. When enabled, the device generates an internal reset as V_{CC} rises to a voltage level above 2.0V. The on-chip reset circuitry is able to detect both fast and slow rise times on V_{CC} (V_{CC} rise time between 10 ns and 50 ms).

Under no circumstances should the $\overline{\text{RESET}}$ pin be allowed to float. If the on-chip Power-On Reset feature is being used, $\overline{\text{RESET}}$ pin should be connected directly to V_{CC}. The output of the power-on reset detector will always preset the Idle timer to 0FFF(4096 tc). At this time, the internal reset will be generated.

If the Power-On 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 user is responsible for ensuring that V_{CC} is at the minimum level for the operating frequency within the 4096 tc. After the underflow, the logic is designed such that no additional internal resets occur as long as V_{CC} remains above 2.0V.

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

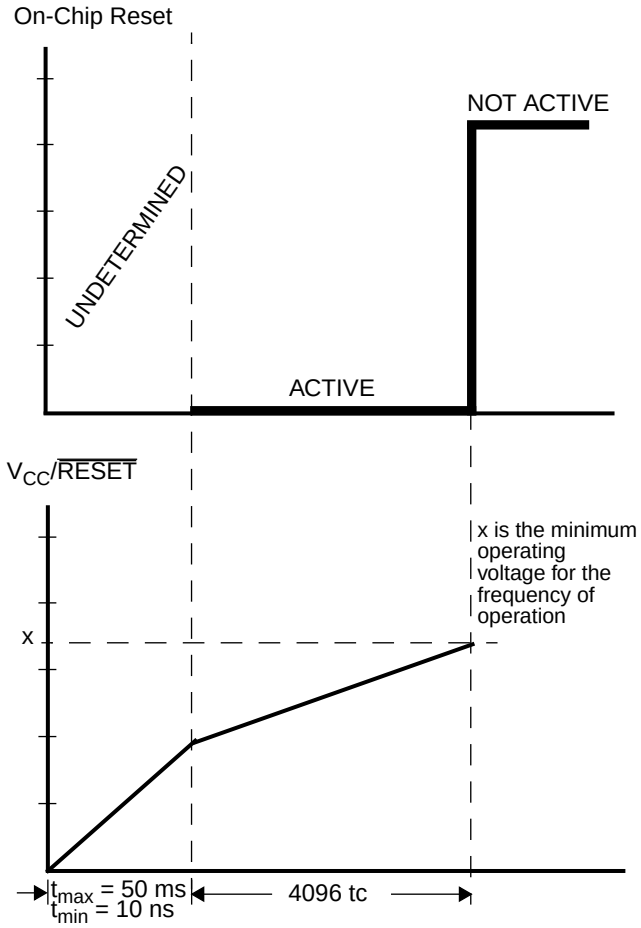


Figure 2-10 Reset Timing (Power-On Reset enabled) With V_{CC} Tied to $\overline{RESET}$

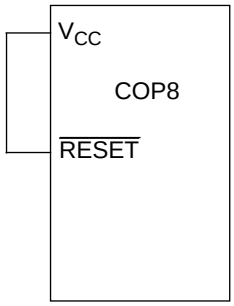


Figure 2-11 Reset Circuit Using Power-On Reset

2.8.8 Oscillator Circuits

There are four clock oscillator options available: Crystal Oscillator with or without on chip bias resistor, R/C Oscillator with on-chip resistor and capacitor, and External Oscillator. The oscillator feature is selected by programming the ECON register, which is summarized in Table 2-2.

Table 2-2 Oscillator Option

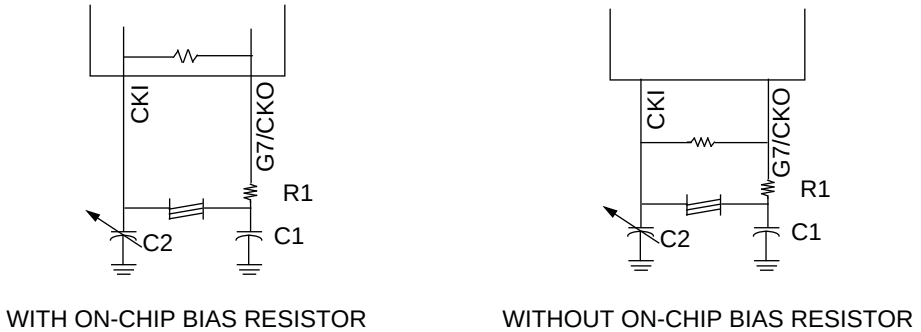
ECON4	ECON3	Oscillator Option
0	0	External Oscillator
1	0	Crystal Oscillator without bias resistor
0	1	R/C Oscillator
1	1	Crystal Oscillator with bias resistor

Crystal Oscillator

The Crystal Oscillator mode can be selected by programming ECON Bit 4 to 1. CKI is the clock input while G7/CKO is the clock generator output to the crystal. An on-chip bias resistor connected between CKI and CKO can be enabled by programming ECON Bit 3 to 1 with the crystal oscillator option selection. The value of the resistor is in the range of 0.5 M to 2M (typically 1.0 M). Table 2-3 shows the component values required for various standard crystal values. Resistor R2 is only used when the on-chip bias resistor is disabled. Figure 2-12 shows the crystal oscillator connection diagram.

Table 2-3 Crystal Oscillator Configuration, T_A = 25˚C, V_{CC} = 5V

R1 (k Ω)	R2 (M Ω)	C1 (pF)	C2 (pF)	CKI Freq (MHz)
0	1	30	30	15
0	1	32	32	10
0	1	45	30–36	4
5.6	1	100	100-156	0.455



WITH ON-CHIP BIAS RESISTOR WITHOUT ON-CHIP BIAS RESISTOR

Figure 2-12 Crystal Oscillator

External Oscillator

The External Oscillator mode can be selected by programming ECON Bit 3 to 0 and ECON Bit 4 to 0. CKI can be driven by an external clock signal provided it meets the specified duty cycle, rise and fall times, and input levels. G7/CKO is available as a general purpose input G7 and/or Halt control. Figure 2-13 shows the external oscillator connection diagram.

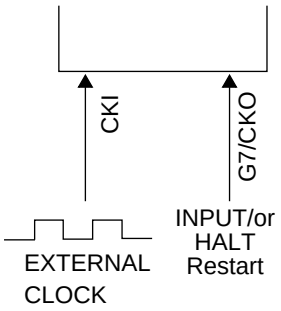


Figure 2-13 External Oscillator

R/C Oscillator

The R/C Oscillator mode can be selected by programming ECON Bit 3 to 1 and ECON Bit 4 to 0. In R/C oscillation mode, CKI is left floating, while G7/CKO is available as a general purpose input G7 and/or HALT control. The R/C controlled oscillator has on-chip resistor and capacitor for maximum R/C oscillator frequency operation. The maximum frequency is 5 MHz +/- 35% for V_{CC} between 4.5V to 5.5V and temperature range of -40˚C to +85˚C. For max frequency operation, the CKI pin should be left floating. For lower frequencies, an external capacitor should be connected between CKI and GND. Table 2-4 shows the oscillator frequency as a function of external capacitance on the CKI pin. Figure 2-14 shows the R/C oscillator configuration.

Table 2-4 R/C Oscillator Configuration, -40˚C to +85˚C, V_{CC} = 4.5V to 5.5V, OSC Freq Variation of ± 35%

External Capacitor (pF)	R/C OSC Freq (MHz)	Instr. Cycle (µSec)
0	5	2.0
9	4	2.5
52	2	5.0
150	1	10
TBD	32 KHz	312.5

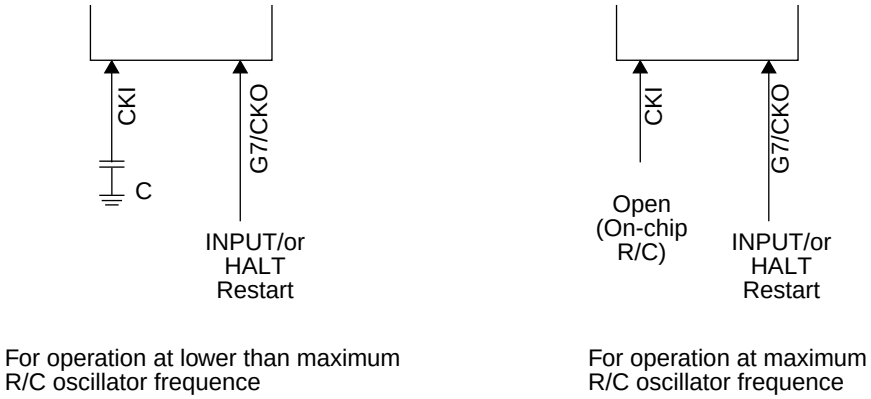


Figure 2-14 R/C Oscillator

2.8.9 Control Registers

CNTRL Register (Address X'00EE)

The Timer1 (T1) and MICROWIRE/PLUS control register contains the following bits:

SL1 & SL0 Select the MICROWIRE/PLUS clock divide by (00 = 2, 01 = 4, 1x = 8)

IEDG External interrupt edge polarity select
(0 = Rising edge, 1 = Falling edge)

MSEL Selects G5 and G4 as MICROWIRE/PLUS signals SK and SO respectively

T1C0 Timer T1 Start/Stop control in timer modes 1 and 2.

T1 Underflow Interrupt Pending Flag in timer mode 3

T1C1 Timer T1 mode control bit

T1C2 Timer T1 mode control bit

T1C3 Timer T1 mode control bit

T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

Bit 7 Bit 0

PSW Register (Address X'00EF)

The PSW register contains the following select bits:

GIE Global interrupt enable (enables interrupts)

EXEN Enable external interrupt

BUSY MICROWIRE/PLUS busy shifting flag

EXPND External interrupt pending

T1ENA Timer T1 Interrupt Enable for Timer Underflow or T1A Input capture edge

T1PNDA Timer T1 Interrupt Pending Flag (Autoreload RA in mode 1, T1 Underflow in Mode 2, T1A capture edge in mode 3)

C Carry Flag

HC Half Carry Flag

HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE
Bit 7							Bit 0

The Half-Carry flag is also affected by all the instructions that affect the Carry flag. The SC (Set Carry) and R/C (Reset Carry) instructions will respectively set or clear both the carry flags. In addition to the SC and R/C instructions, ADC, SUBC, RRC and RLC instructions affect the Carry and Half Carry flags.

ICNTRL Register (Address X'00E8)

The ICNTRL register contains the following bits:

T1ENB Timer T1 Interrupt Enable for T1B Input capture edge

T1PNDB Timer T1 Interrupt Pending Flag for T1B capture edge

µWEN Enable MICROWIRE/PLUS interrupt

µWPND MICROWIRE/PLUS interrupt pending

T0EN Timer T0 Interrupt Enable (Bit 12 toggle)

T0PND Timer T0 Interrupt pending

LPEN L Port Interrupt Enable (Multi-Input Wakeup/Interrupt)

Bit 7 could be used as a general purpose status flag

Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB
Bit 7							Bit 0

2.9 TIMERS

The device contains a very versatile set of timers (T0, T1). Timer T1and associated autoreload/capture registers power up containing random data.

2.9.1 Timer T0 (IDLE Timer)

The device supports applications that require maintaining real time and low power with the IDLE mode. This IDLE mode support is furnished by the IDLE timer T0. The Timer T0 runs continuously at the fixed rate of the instruction cycle clock, t_C. The user cannot read or write to the IDLE Timer T0, which is a count down timer.

The Timer T0 supports the following functions:

- Exit out of the Idle Mode (See Idle Mode description)
- WATCHDOG logic (See WATCHDOG description)
- Start up delay out of the HALT mode
- Timing the width of the internal power-on-reset

The IDLE Timer T0 can generate an interrupt when the twelfth bit toggles. This toggle is latched into the T0PND pending flag, and will occur every 4 .096 ms at the maximum clock frequency (t_C = 1 µs). A control flag T0EN allows the interrupt from the twelfth bit of Timer T0 to be enabled or disabled. Setting T0EN will enable the interrupt, while resetting it will disable the interrupt.

2.9.2 Timer T1

One of the main functions of a microcontroller is to provide timing and counting capability for real-time control tasks. The COP888 family offers a very versatile 16-bit timer/counter structure, and two supporting 16-bit autoreload/capture registers (R1A and R1B), optimized to reduce software burdens in real-time control applications. The timer block has two pins associated with it, T1A and T1B. Pin T1A supports I/O required by the timer block, while pin T1B is an input to the timer block.

The timer block has three operating modes: Processor Independent PWM mode, External Event Counter mode, and Input Capture mode.

The control bits T1C3, T1C2, and T1C1 allow selection of the different modes of operation.

Mode 1. Processor Independent PWM Mode

One of the timer's operating modes is the Processor Independent PWM mode. In this mode, the timer generates a "Processor Independent" PWM signal because once the timer is setup, no more action is required from the CPU which translates to less software overhead and greater throughput. The user software services the timer block only when the PWM parameters require updating. This capability is provided by the fact that the timer has two separate 16 -bit reload registers. One of the reload registers contains the "ON" timer while the other holds the "OFF" time. By contrast, a microcontroller that has

only a single reload register requires an additional software to update the reload value (alternate between the on-time/off-time).

The timer can generate the PWM output with the width and duty cycle controlled by the values stored in the reload registers. The reload registers control the countdown values and the reload values are automatically written into the timer when it counts down through 0, generating interrupt on each reload. Under software control and with minimal overhead, the PMW outputs are useful in controlling motors, triacs, the intensity of displays, and in providing inputs for data acquisition and sine wave generators.

In this mode, the timer T1 counts down at a fixed rate of tc. Upon every underflow the timer is alternately reloaded with the contents of supporting registers, R1A and R1B. The very first underflow of the timer causes the timer to reload from the register R1A. Subsequent underflows cause the timer to be reloaded from the registers alternately beginning with the register R1B.

The T1 Timer control bits, T1C3, T1C2 and T1C1 set up the timer for PWM mode operation.

Figure 2-15 shows a block diagram of the timer in PWM mode.

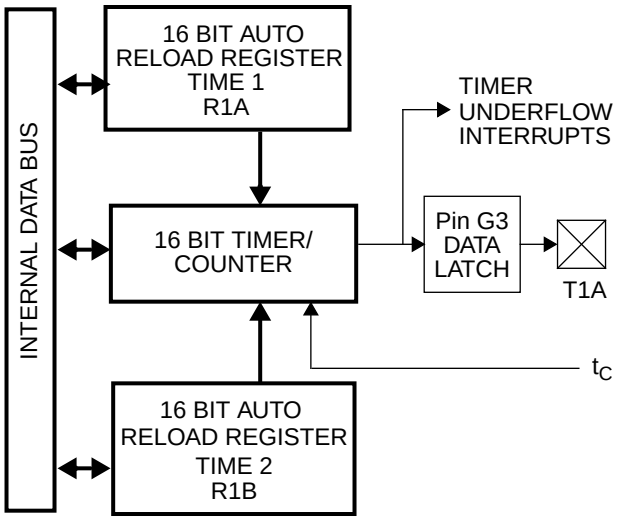


Figure 2-15 Timer in PWM Mode

The underflows can be programmed to toggle the T1A output pin. The underflows can also be programmed to generate interrupts.

Underflows from the timer are alternately latched into two pending flags, T1PNDA and T1PNDB. The user must reset these pending flags under software control. Two control enable flags, T1ENA and T1ENB, allow the interrupts from the timer underflow to be enabled or disabled. Setting the timer enable flag T1ENA will cause an interrupt when a timer underflow causes the R1A register to be reloaded into the timer. Setting the timer enable flag T1ENB will cause an interrupt when a timer underflow causes the R1B register to be reloaded into the timer. Resetting the timer enable flags will disable the associated interrupts.

Either or both of the timer underflow interrupts may be enabled. This gives the user the flexibility of interrupting once per PWM period on either the rising or falling edge of the PWM output. Alternatively, the user may choose to interrupt on both edges of the PWM output.

Mode 2. External Event Counter Mode

This mode is quite similar to the processor independent PWM mode described above. The main difference is that the timer, T1, is clocked by the input signal from the T1A pin. The T1 timer control bits, T1C3, T1C2 and T1C1 allow the timer to be clocked either on a positive or negative edge from the T1A pin. Underflows from the timer are latched into the T1PNDA pending flag. Setting the T1ENA control flag will cause an interrupt when the timer underflows.

In this mode the input pin T1B can be used as an independent positive edge sensitive interrupt input if the T1ENB control flag is set. The occurrence of a positive edge on the T1B input pin is latched into the T1PNDB flag.

Figure 2-16 shows a block diagram of the timer in External Event Counter mode.

NOTE: The PWM output is not available in this mode since the T1A pin is being used as the counter input clock.

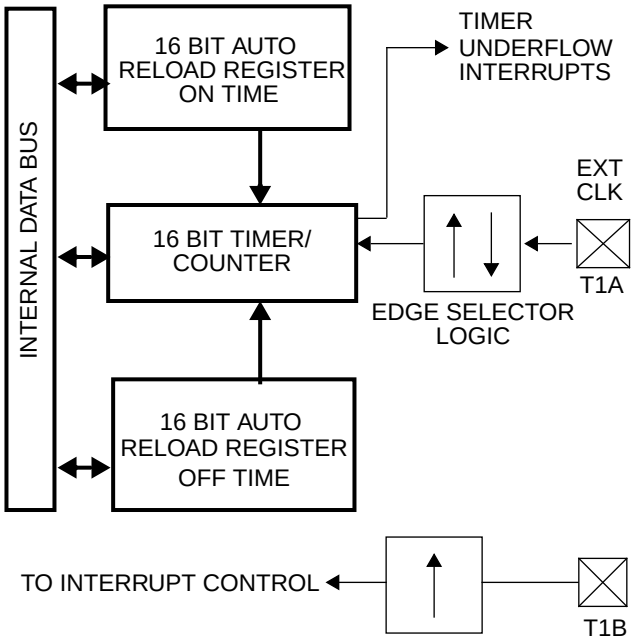


Figure 2-16 Timer in External Event Counter Mode

Mode 3. Input Capture Mode

The device can precisely measure external frequencies or time external events by placing the timer block, T1, in the input capture mode.In this mode, the reload registers serve as independent capture registers, capturing the contents of the timer when an external event occurs (transition on the timer input pin). The capture registers can be read while maintaining count, a feature that lets the user measure elapsed time and time between events. By saving the timer value when the external event occurs, the time of the

external event is recorded. Most microcontrollers have a latency time because they cannot determine the timer value when the external event occurs. The capture register eliminates the latency time, thereby allowing the applications program to retrieve the timer value stored in the capture register.

In this mode, the timer T1 is constantly running at the fixed tc rate. The two registers, R1A and R1B, act as capture registers. Each register acts in conjunction with a pin. The register R1A acts in conjunction with the T1A pin and the register R1B acts in conjunction with the T1B pin.

The timer value gets copied over into the register when a trigger event occurs on its corresponding pin. Control bits, T1C3, T1C2 and T1C1, allow the trigger events to be specified either as a positive or a negative edge. The trigger condition for each input pin can be specified independently.

The trigger conditions can also be programmed to generate interrupts. The occurrence of the specified trigger condition on the T1A and T1B pins will be respectively latched into the pending flags, T1PNDA and T1PNDB. The control flag T1ENA allows the interrupt on T1A to be either enabled or disabled. Setting the T1ENA flag enables interrupts to be generated when the selected trigger condition occurs on the T1A pin. Similarly, the flag T1ENB controls the interrupts from the T1B pin.

Underflows from the timer can also be programmed to generate interrupts. Underflows are latched into the timer T1C0 pending flag (the T1C0 control bit serves as the timer underflow interrupt pending flag in the Input Capture mode). Consequently, the T1C0 control bit should be reset when entering the Input Capture mode. The timer underflow interrupt is enabled with the T1ENA control flag. When a T1A interrupt occurs in the Input Capture mode, the user must check both the T1PNDA and T1C0 pending flags in order to determine whether a T1A input capture or a timer underflow (or both) caused the interrupt.

Figure 2-17 shows a block diagram of the timer in Input Capture mode.

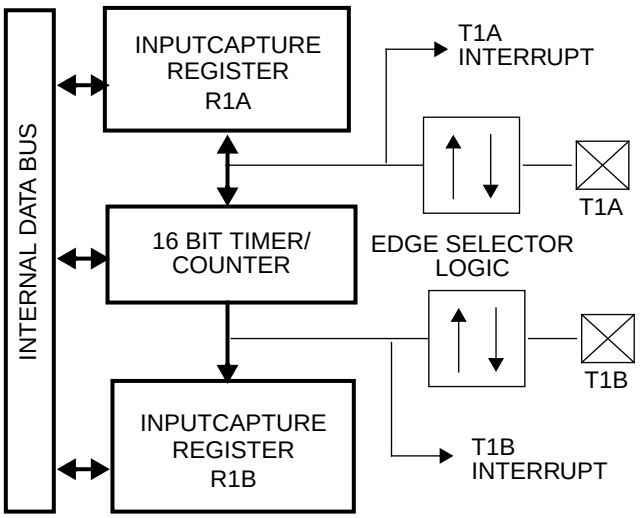


Figure 2-17 Timer in Input Capture Mode

2.10 TIMER CONTROL FLAGS

The control bits and their functions are summarized below.

T1C0 Timer Start/Stop control in Modes 1 and 2 (Processor Independent PWM and External Event Counter), where 1 = Start, 0 = Stop
Timer Underflow Interrupt Pending Flag in Mode 3 (Input Capture)

T1PNDA Timer Interrupt Pending Flag

T1PNDB Timer Interrupt Pending Flag

T1ENA Timer Interrupt Enable Flag

T1ENB Timer Interrupt Enable Flag

1 = Timer Interrupt Enabled

0 = Timer Interrupt Disabled

T1C3 Timer mode control

T1C2 Timer mode control

T1C1 Timer mode control

The timer mode control bits (T1C3, T1C2 and T1C1) are detailed below:

T1C3	T1C2	T1C1	Timer Mode	Interrupt A Source	Interrupt B Source	Timer Counts On
0	0	0	MODE 2 (External Event Counter)	Timer Underflow	Pos. T1B Edge	T1A Pos. Edge
0	0	1	MODE 2 (External Event Counter)	Timer Underflow	Pos. T1B Edge	T1A Neg. Edge
1	0	1	MODE 1 (PWM) T1A Toggle	Autoreload RA	Autoreload RB	t_C
1	0	0	MODE 1 (PWM) No T1A Toggle	Autoreload RA	Autoreload RB	t_C
0	1	0	MODE 3 (Capture) Captures: T1A Pos. Edge T1B Pos. Edge	Pos. T1A Edge or Timer Underflow	Pos. T1B Edge	t_C
1	1	0	MODE 3 (Capture) Captures: T1A Pos. Edge T1B Neg. Edge	Pos. T1A Edge or Timer Underflow	Neg. T1B Edge	t_C
0	1	1	MODE 3 (Capture) Captures: T1A Neg. Edge T1B Pos. Edge	Neg. T1A Edge or Timer Underflow	Pos. T1B Edge	t_C
1	1	1	MODE 3 (Capture) Captures: T1A Neg. Edge T1B Neg. Edge	Neg. T1A Edge or Timer Underflow	Neg. T1B Edge	t_C

2.11 POWER SAVING FEATURES

Today, the proliferation of battery-operated based applications has placed new demands on designers to drive power consumption down. Battery- operated systems are not the only type of applications demanding low power. The power budget constraints are also imposed on those consumer/industrial applications where well regulated and expensive power supply costs cannot be tolerated. Such applications rely on low cost and low power supply voltage derived directly from the "mains" by using voltage rectifier and passive components. Low power is demanded even in automotive applications, due to increased vehicle electronics content. This is required to ease the burden from the car battery. Low power 8-bit microcontrollers supply the smarts to control battery-operated, consumer/ industrial, and automotive applications.

The COPSAx7 devices offers system designers a variety of low-power consumption features that enable them to meet the demanding requirements of today's increasing range of low -power applications. These features include low voltage operation, low current drain, and power saving features such as HALT, IDLE, and Multi-Input wakeup (MIWU).

The devices offers the user two power save modes of operation: HALT and IDLE. In the HALT mode, all microcontroller activities are stopped. In the IDLE mode, the on-board oscillator circuitry and timer T0 are active but all other microcontroller activities are stopped. In either mode, all on-board RAM, registers, I/O states, and timers (with the exception of T0) are unaltered.

Clock Monitor if enabled can be active in both modes.

2.11.1 HALT Mode

The device can be placed in the HALT mode by writing a “1” to the HALT flag (G7 data bit). All microcontroller activities, including the clock and timers, are stopped. The WATCHDOG logic on the device is disabled during the HALT mode. However, the clock monitor circuitry, if enabled, remains active and will cause the WATCHDOG output pin (WDOUT) to go low. If the HALT mode is used and the user does not want to activate the WDOUT pin, the Clock Monitor should be disabled after the device comes out of reset (resetting the Clock Monitor control bit with the first write to the WDSVR register). In the HALT mode, the power requirements of the device are minimal and the applied voltage (V_{CC}) may be decreased to V_r (V_r = 2.0V) without altering the state of the machine.

The device supports three different ways of exiting the HALT mode. The first method of exiting the HALT mode is with the Multi-Input Wakeup feature on Port L. The second method is with a low to high transition on the CKO (G7) pin. This method precludes the use of the crystal clock configuration (since CKO becomes a dedicated output), and so may only be used with an R/C or external clock configuration. The third method of exiting the HALT mode is by pulling the $\overline{\text{RESET}}$ pin low.

Since a crystal or ceramic resonator may be selected as the oscillator, the Wakeup signal is not allowed to start the chip running immediately since crystal oscillators and ceramic resonators have a delayed start up time to reach full amplitude and frequency stability. The IDLE timer is used to generate a fixed delay to ensure that the oscillator has indeed

stabilized before allowing instruction execution. In this case, upon detecting a valid Wakeup signal, only the oscillator circuitry is enabled. The IDLE timer is loaded with a value of 256 and is clocked with the tc instruction cycle clock. The tc clock is derived by dividing the oscillator clock down by a factor of 10. The Schmitt trigger following the CKI inverter on the chip ensures that the IDLE timer is clocked only when the oscillator has a sufficiently large amplitude to meet the Schmitt trigger specifications. This Schmitt trigger is not part of the oscillator closed loop. The start-up time-out from the IDLE timer enables the clock signals to be routed to the rest of the chip.

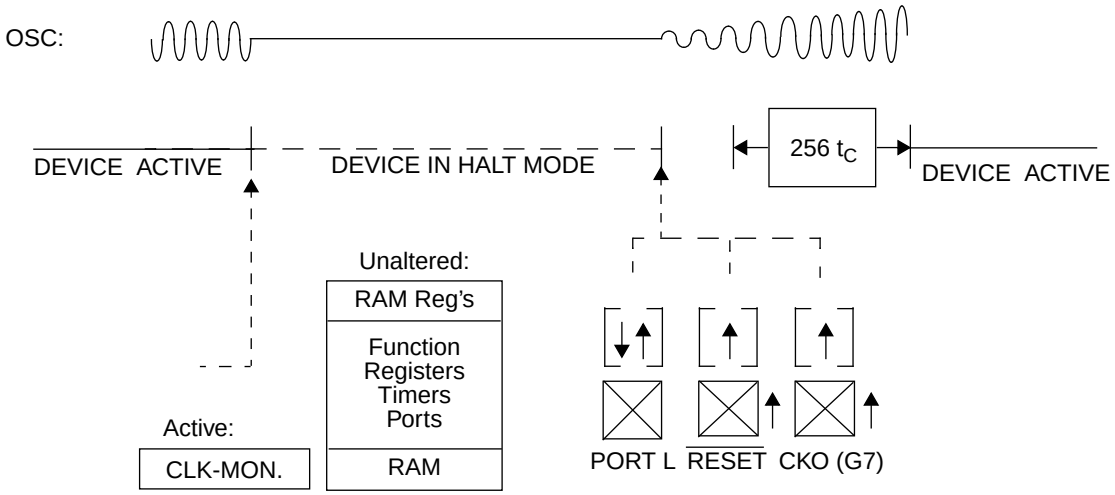


Figure 2-18 Wakeup From HALT

If an R/C clock option is being used, the fixed delay is introduced optionally. A control bit, CLKDLY, mapped as configuration bit G7, controls whether the delay is to be introduced or not. The delay is included if CLKDLY is set, and excluded if CLKDLY is reset. The CLKDLY bit is cleared on reset.

The device has two options associated with the HALT mode. The first option enables the HALT mode feature, while the second option disables the HALT mode selected through bit 0 of the ECON register. With the HALT mode enable option, the device will enter and exit the HALT mode as described above. With the HALT disable option, the device cannot be placed in the HALT mode (writing a "1" to the HALT flag will have no effect, the HALT flag will remain "0").

The WATCHDOG detector circuit is inhibited during the HALT mode. However, the clock monitor circuit if enabled remains active during HALT mode in order to ensure a clock monitor error if the device inadvertently enters the HALT mode as a result of a runaway program or power glitch.

If the device is placed in the HALT mode, with the R/C oscillator selected, the clock input pin (CKI) is forced to a logic high internally. With the crystal or external oscillator the CKI pin is TRI-STATE.

2.11.2 IDLE Mode

The device is placed in the IDLE mode by writing a "1" to the IDLE flag (G6 data bit). In this mode, all activities, except the associated on-board oscillator circuitry and the IDLE Timer T0, are stopped.

As with the HALT mode, the device can be returned to normal operation with a reset, or with a Multi-Input Wakeup from the L Port. Alternately, the microcontroller resumes normal operation from the IDLE mode when the twelfth bit (representing 4.096 ms at internal clock frequency of 10 MHz, t_C= 1 µs) of the IDLE Timer toggles.

This toggle condition of the twelfth bit of the IDLE Timer T0 is latched into the T0PND pending flag.

The user has the option of being interrupted with a transition on the twelfth bit of the IDLE Timer T0. The interrupt can be enabled or disabled via the T0EN control bit. Setting the T0EN flag enables the interrupt and vice versa.

The user can enter the IDLE mode with the Timer T0 interrupt enabled. In this case, when the T0PND bit gets set, the device will first execute the Timer T0 interrupt service routine and then return to the instruction following the “Enter Idle Mode” instruction.

Alternatively, the user can enter the IDLE mode with the IDLE Timer T0 interrupt disabled. In this case, the device will resume normal operation with the instruction immediately following the “Enter IDLE Mode” instruction.

NOTE: It is necessary to program two NOP instructions following both the set HALT mode and set IDLE mode instructions. These NOP instructions are necessary to allow clock resynchronization following the HALT or IDLE modes.

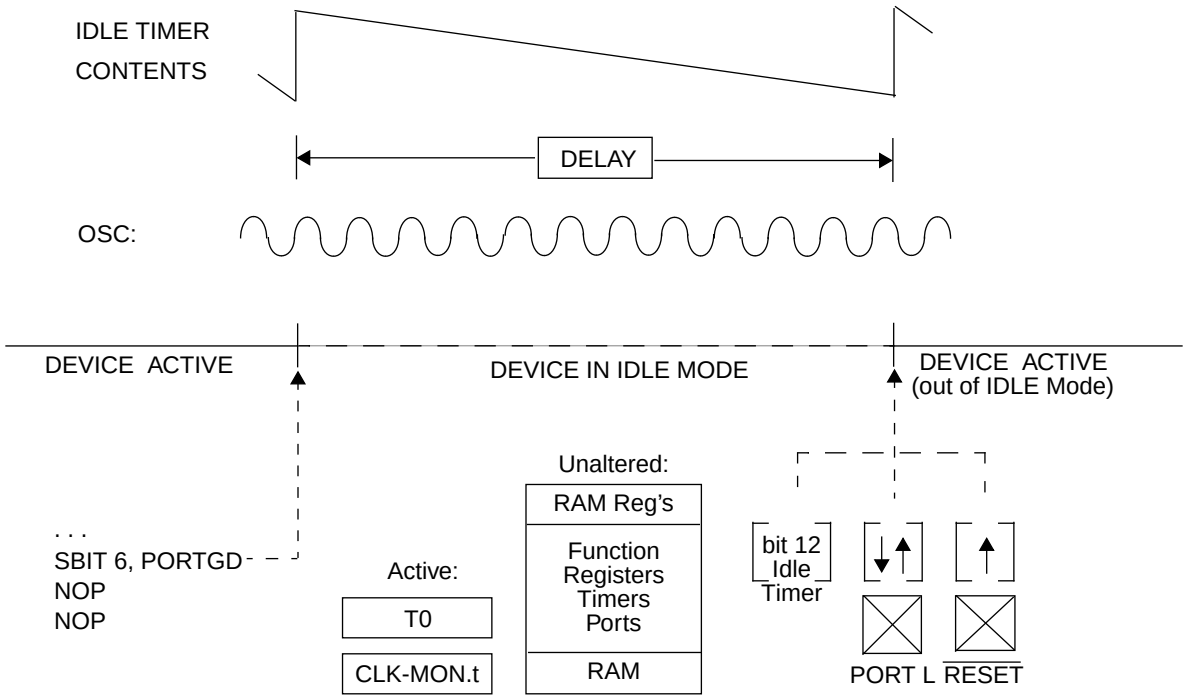


Figure 2-19 Wakeup From IDLE

2.12 MULTI-INPUT WAKEUP

The Multi-Input Wakeup feature is used to return (wakeup) the device from either the HALT or IDLE modes. Alternately Multi-Input Wakeup/Interrupt feature may also be used to generate up to 8 edge selectable external interrupts.

Figure 2-20 shows the Multi-Input Wakeup logic.

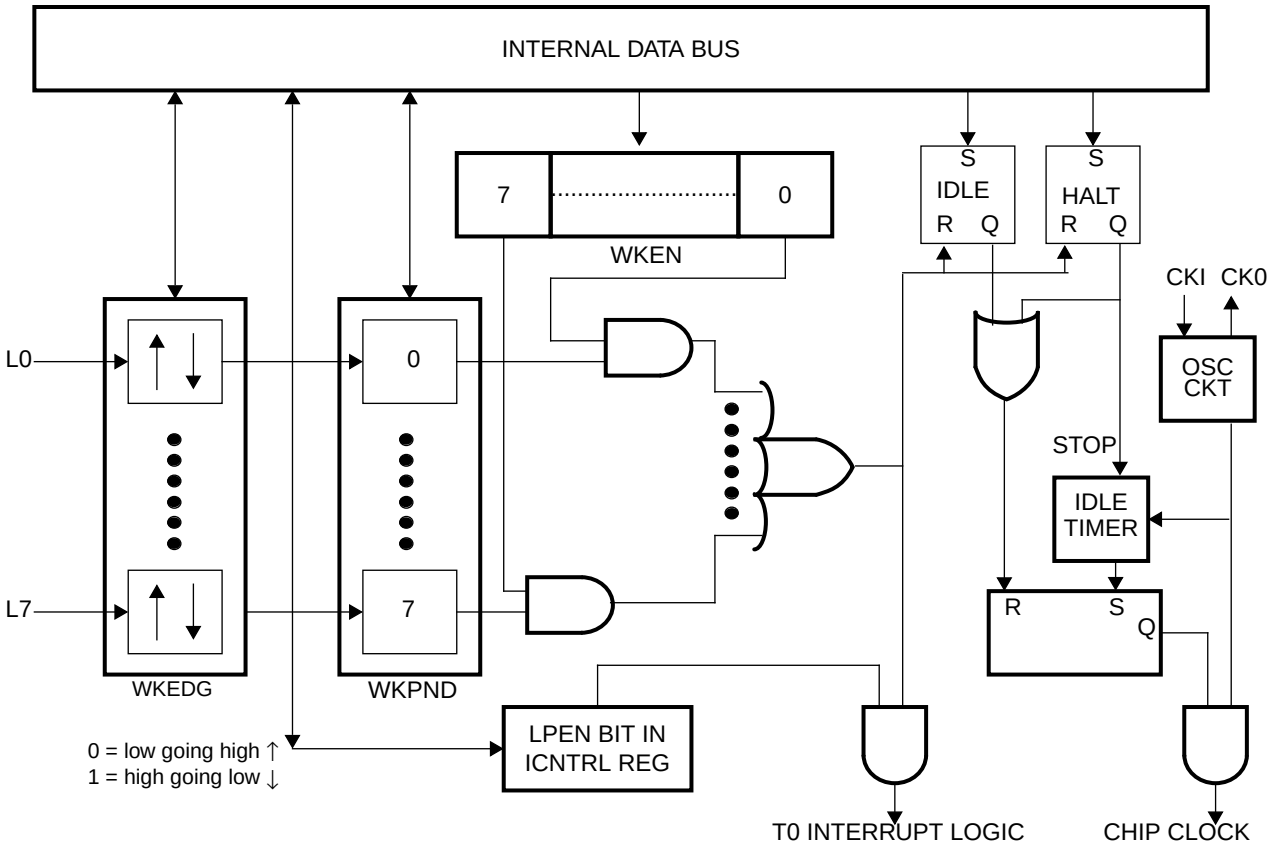


Figure 2-20 Multi-Input Wake Up Logic

Multi-Input Wakeup (MIWU) enables designers to specify whichever of the controller's 8 pins they want to be used to "wake up" the device to process instructions. This capability is activated during a transition on the input pin — from low to high, or high to low — which is recorded internally in the chip's registers. Without multi-input wakeup, the microcontroller would be required to keep its software and code running constantly (Figure 2-21).

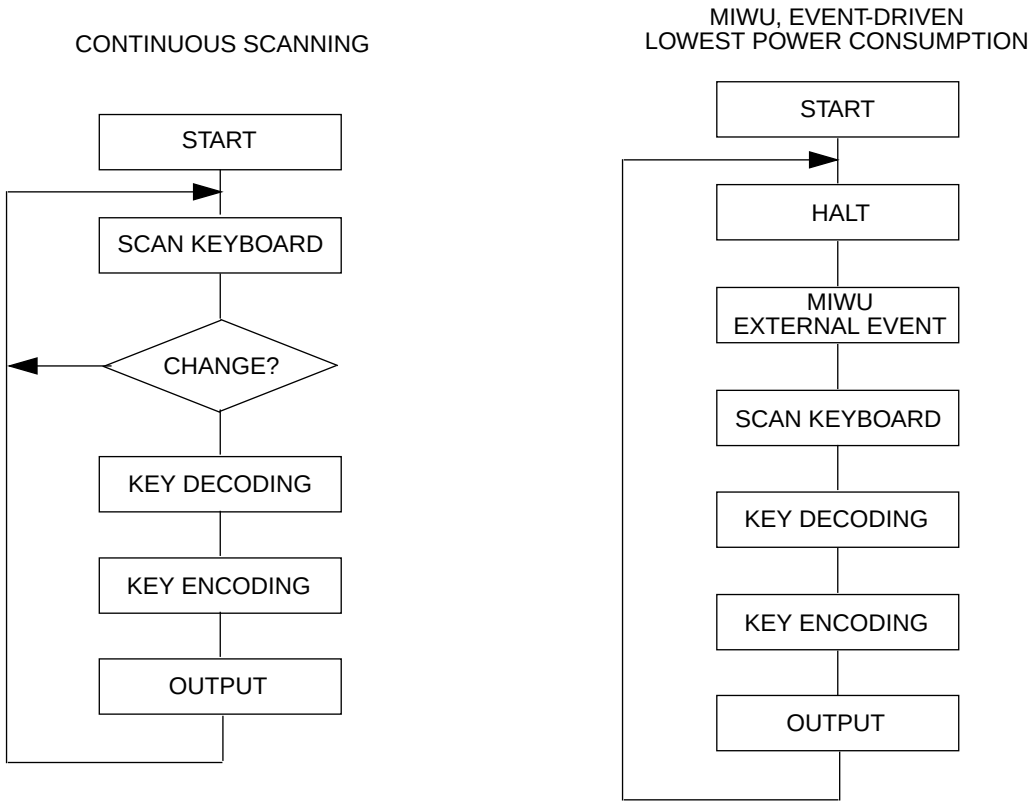


Figure 2-21 Keyboard Scanning

MIMW typically reduces current consumption down to less than 4 µA during quiescent states — compared to 6 µA of current consumption if code remained running. In addition, MIWU makes hardware design easier and more efficient by reducing component count, obviating the need for glue logic.

In a laptop keyboard application, for example, the L port is used as the keyboard input lines.

The microcontroller is in the HALT mode until a keystroke or data transmission -to-low transition on one of the lines, compelling the device to exit the HALT mode. The keyboard is than scanned to detect which key is pressed, and the appropriate key-code is sent to the computer. When the scan is complete, the microcontroller is switched back to the HALT mode (Figure 2-21).

With features such as Multi-Input Wake-up, HALT and IDLE, and low voltage and current drain capabilities, National's COP8SAx7 family of 8-bit microcontrollers is doing its part to reduce power consumption. And with "current" trends pointing to an environment where less is indeed more, that's putting real power back in the hands of today's embedded control system design.

The Multi-Input Wakeup feature utilizes the L Port. The user selects which particular L port bit (or combination of L Port bits) will cause the device to exit the HALT or IDLE modes. The selection is done through the register WKEN. The register WKEN is an 8-bit read/write register, which contains a control bit for every L port bit. Setting a particular WKEN bit enables a Wakeup from the associated L port pin.

The user can select whether the trigger condition on the selected L Port pin is going to be either a positive edge (low to high transition) or a negative edge (high to low transition). This selection is made via the register WKEDG, which is an 8-bit control register with a bit assigned to each L Port pin. Setting the control bit will select the trigger condition to be a negative edge on that particular L Port pin. Resetting the bit selects the trigger condition to be a positive edge. Changing an edge select entails several steps in order to avoid a Wakeup condition as a result of the edge change. First, the associated WKEN bit should be reset, followed by the edge select change in WKEDG. Next, the associated WKPND bit should be cleared, followed by the associated WKEN bit being re-enabled.

An example may serve to clarify this procedure. Suppose we wish to change the edge select from positive (low going high) to negative (high going low) for L Port bit 5, where bit 5 has previously been enabled for an input interrupt. The program would be as follows:

```
RBIT 5, WKEN; Disable MIWU
SBIT 5, WKEDG; Change edge polarity
RBIT 5, WKPND; Reset pending flag
SBIT 5, WKEN; Enable MIWU
```

If the L port bits have been used as outputs and then changed to inputs with Multi-Input Wakeup/Interrupt, a safety procedure should also be followed to avoid wakeup conditions. After the selected L port bits have been changed from output to input but before the associated WKEN bits are enabled, the associated edge select bits in WKEDG should be set or reset for the desired edge selects, followed by the associated WKPND bits being cleared.

This same procedure should be used following reset, since the L port inputs are left floating as a result of reset.

The occurrence of the selected trigger condition for Multi-Input Wakeup is latched into a pending register called WKPND. The respective bits of the WKPND register will be set on the occurrence of the selected trigger edge on the corresponding Port L pin. The user has the responsibility of clearing these pending flags. Since WKPND is a pending register for the occurrence of selected wakeup conditions, the device will not enter the HALT mode if any Wakeup bit is both enabled and pending. Consequently, the user must clear the pending flags before attempting to enter the HALT mode.

WKEN and WKEDG are all read/write registers, and are cleared at reset. WKPND register contains random value after reset.

2.13 INTERRUPTS

2.13.1 Introduction

The device supports eight vectored interrupts. Interrupt sources include Timer 1, Timer T0, Port L Wakeup, Software Trap, MICROWIRE/PLUS, and External Input.

All interrupts force a branch to location 00FF Hex in program memory. The VIS instruction may be used to vector to the appropriate service routine from location 00FF Hex.

The Software trap has the highest priority while the default VIS has the lowest priority.

Each of the six maskable inputs has a fixed arbitration ranking and vector.

Figure 2-22 shows the Interrupt block diagram.

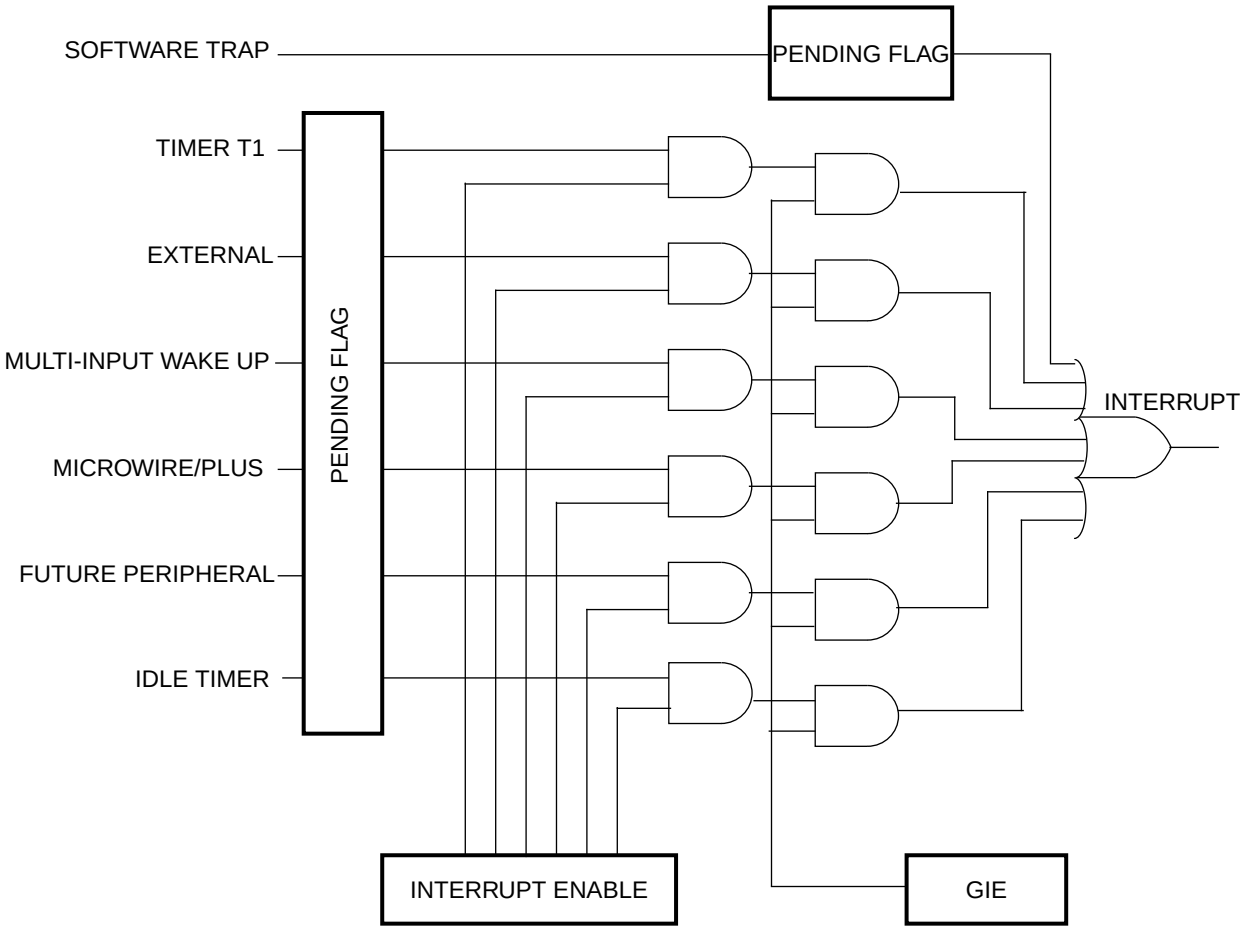


Figure 2-22 Interrupt Block Diagram

2.13.2 Maskable Interrupts

All interrupts other than the Software Trap are maskable. Each maskable interrupt has an associated enable bit and pending flag bit. The pending bit is set to 1 when the interrupt condition occurs. The state of the interrupt 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 are contained in 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 device is not processing a non-maskable 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.

An interrupt is an asynchronous event which may occur before, during, or after an instruction cycle. Any interrupt which occurs during the execution of an instruction is not acknowledged until the start of the next normally executed instruction is to be skipped, the skip is performed before the pending interrupt is acknowledged.

At the start of interrupt acknowledgment, 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(PC) is loaded with 00FF Hex, causing a jump to that program memory location.

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

If the user wishes to allow nested interrupts, the interrupts 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. The user must write the program in such a was as to prevent stack overflow, loss of saved context information, and other unwanted conditions.

The interrupt service routine stored at location 00FF Hex should use the VIS instruction to determine the cause of the interrupt, and jump to the interrupt handling routine corresponding to the highest priority enabled and active interrupt. Alternately, the user 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 user's program must decide which interrupt to service.

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 set 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.

2.13.3 VIS Instruction

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 interrupt vector table, directs the device 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. 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 interrupts sources are stored in a vector table.

The vector table may be as long as 32 bytes (maximum of 16 vectors) 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 to 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 Software Trap has the highest rand and its vector is always located at 0yFE and 0yFF. The number of interrupts which can become active defines the size of the table.

Table 2-5 shows the types of interrupts, the interrupt arbitration ranking, and the locations of the corresponding vectors in the vector table.

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

Table 2-5 Interrupt Vector Table

ARBITRATION RANKING	SOURCE	DESCRIPTION	VECTOR* ADDRESS (Hi-Low Byte)
(1) Highest	Software	INTR Instruction	0yFE - 0yFF
(2)	Reserved	Future	0yFC - 0yFD
(3)	External	G0	0yFA - 0yFB
(4)	Timer T0	Underflow	0yF8 - 0yF9
(5)	Timer T1	T1A/Underflow	0yF6 - 0yF7
(6)	Timer T1	T1B	0yF4 - 0yF5
(7)	MICROWIRE/PLUS	Busy Low	0yF2 - 0yF3
(8)	Reserved	Future	0yF0 - 0yF1
(9)	Reserved	Future	0yEE - 0yEF
(10)	Reserved	Future	0yEC - 0yED
(11)	Reserved	Future	0yEA - 0yEB
(12)	Reserved	Future	0yE8 - 0yE9
(13)	Reserved	Future	0yE6 - 0yE7
(14)	Reserved	Future	0yE4 - 0yE5
(15)	Port L/Wakeup	Port L Edge	0yE2 - 0yE3
(16) Lowest	Default	VIS Instruction Execution without any interrupts	0yE0 - 0yE1

*y is a variable which represents the VIS block. VIS and the vector table must be located in the same 256-byte block except if VIS is located at the last address of a block. In this case, the table must be in the next block.

Hex, then the vector location 0yFE and -0yFF should contain the data 03 and 10 Hex, respectively. When a Software Trap interrupt occurs and the VIS instruction is executed, the program jumps to the address specified in the vector table.

The interrupt sources in the vector table 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.

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 probably the result of an error. It can result from a change in the enable bits or pending flags prior to using the VIS instruction, or from inadvertent execution of the VIS command outside of the context of an interrupt. It is a good idea to make this vector point to the Software Trap interrupt service routine or some other error handling routine. A normal RETI instruction should not be used in any such routine because the stack might not contain a valid return address

To ensure reliable operation, the user should always use the VIS instruction to determine the source of an interrupt. Although it is possible to poll the pending bits to detect the source of an interrupt, this practice is not recommended. The use of polling allows the standard arbitration ranking to be altered, but the reliability of the interrupt system is compromised. The polling routine must individually test the enable and pending bits of each maskable interrupt. If a Software Trap interrupt should occur, it will be serviced last, even though it should have the highest priority. Under certain conditions, a Software Trap could be triggered but not serviced, resulting in an inadvertent "locking out" of all maskable interrupts by the Software Trap pending flag. Problems such as this can be avoided by using VIS instruction.

VIS Execution

When the VIS instruction is executed it activates the arbitration logic. The arbitration logic generates an even number between E0 and FE (E0, E2, E4, E6 etc...) depending on which active interrupt has the highest arbitration ranking at the time of the 1st cycle of VIS is executed. For example, if the software trap interrupt is active, FE is generated. If the external interrupt is active and the software trap interrupt is not, then FA is generated and so forth. If the only active interrupt is software trap, than E0 is generated. This number replaces the lower byte of the PC. The upper byte of the PC remains unchanged. The new PC is therefore pointing to the vector of the active interrupt with the highest arbitration ranking. This vector is read from program memory and placed into the PC which is now pointed to the 1st instruction of the service routine of the active interrupt with the highest arbitration ranking.

Figure 2-23 illustrates the different steps performed by the VIS instruction. Figure 2-24 shows a flowchart for the VIS instruction.

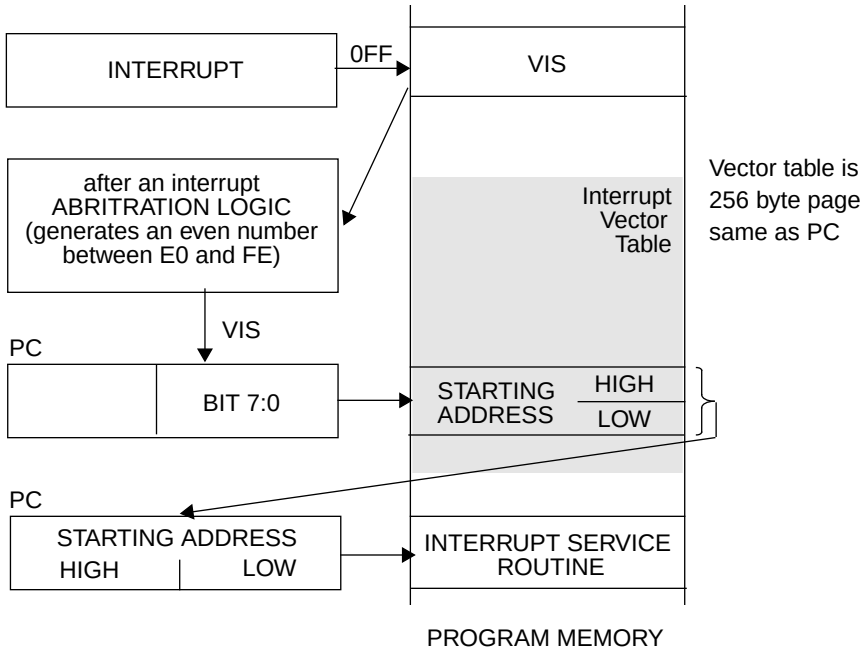


Figure 2-23 VIS Operation

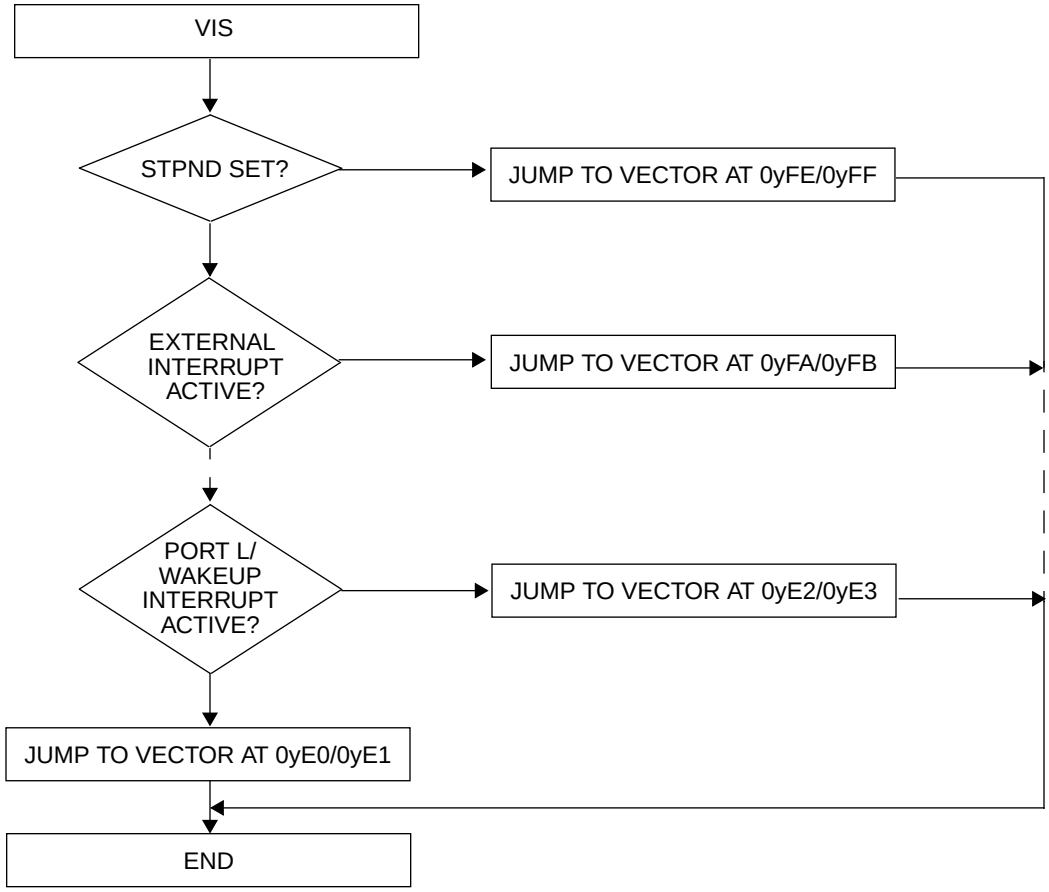


Figure 2-24 VIS Flow Chart

The non-maskable interrupt pending flag is cleared by the RPND (Reset Non-Maskable Pending Bit) instruction (under certain conditions) and upon RESET.

Programming Example: External Interrupt

```
            PSW             =00EF
            CNTRL           =00EE
            RBIT            0,PORTGC
            RBIT            0,PORTGD     ; G0 pin configured Hi-Z
            SBIT            IEDG, CNTRL  ; Ext interrupt polarity: falling edge
            SBIT            GIE, PSW     ; Set the GIE bit
            SBIT            EXEN, PSW    ; Enable the external interrupt
WAIT:       JP              WAIT         ; Wait for external interrupt
            .
            .
            .
            .=0FF                        ; The interrupt causes a
            VIS                          ; branch to address 0FF
                                         ; The VIS causes a branch to
                                         ; interrupt vector table
            .
            .
            .
            .=01FA                       ; Vector table (within 256 byte
            .ADDRW SERVICE               ; of VIS inst.) containing the ext
                                         ; interrupt service routine
            .
            .
            .
SERVICE:                                 ; Interrupt Service Routine
            RBIT, EXPND, PSW             ; Reset ext interrupt pend. bit
            .
            .
            .
            RET I                        ; Return, set the GIE bit
```

2.13.4 Non-maskable Interrupt

Pending Flag

There is a pending flag bit associated with the non-maskable interrupt, called STPND. This pending flag is not memory-mapped and cannot be accessed directly by the software.

The pending flag is reset to zero when a device Reset occurs. When the non-maskable interrupt occurs, the associated pending bit is set to 1. The interrupt service routine should contain an RPND instruction to reset the pending flag to zero. The RPND instruction always resets the STPND flag

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 or unused memory location returns zeros which is interpreted as the INTR instruction.

If the stack is popped beyond the allowed limit (address 02F or 06F 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. The GIE bit is not affected and the pending bit (not accessible by the user) is used to inhibit other interrupts and to direct the program to the ST service routine with the VIS instruction. 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 device Reset, but not necessarily performing all the same functions as a device 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 interrupt routine should record or indicate the context of the device so that the cause of the Software Trap can be determined.

If the user wishes to return to normal execution from the point at which the Software Trap was triggered, the user must first execute RPND, followed by RETSK rather tan RETI or RET. This is because the return address stored on the stack is the address of the INTR instruction that triggered the interrupt. The program must 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 RPND instruction in the service routine for the second Software Trap will reset the STPND flag; upon return to the first Software Trap routine, the STPND flag will have the wrong state. This will allow maskable interrupts to be acknowledged during the servicing of the first Software Trap. To avoid problems such as this, the user program should contain 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 user can use 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.

2.13.5 Port L Interrupts

Port L provides the user with an additional eight fully selectable, edge sensitive interrupts which are all vectored into the same service subroutine.

The interrupt from Port L shares logic with the wake up circuitry. The register WKEN allows interrupts from Port L to be individually enabled or disabled. The register WKEDG specifies the trigger condition to be either a positive or a negative edge. Finally, the register WKPND latches in the pending trigger conditions.

The GIE (Global Interrupt Enable) bit enables the interrupt function.

A control flag, LPEN, functions as a global interrupt enable for Port L interrupts. Setting the LPEN flag will enable interrupts and vice versa. A separate global pending flag is not needed since the register WKPND is adequate.

Since Port L is also used for waking the device out of the HALT or IDLE modes, the user can elect to exit the HALT or IDLE modes either with or without the interrupt enabled. If he elects to disable the interrupt, then the device will restart execution from the instruction immediately following the instruction that placed the microcontroller in the HALT or IDLE modes. In the other case, the device will first execute the interrupt service routine and then revert to normal operation. (See HALT MODE for clock option wakeup information.)

2.13.6 Interrupt Summary

The device 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 restart procedure.

2. Maskable interrupts, triggered by an on-chip peripheral block or an external device connected to the device. Under ordinary conditions, a maskable inter-

rupt will not interrupt any other interrupt routine in progress. A maskable interrupt routine in progress can be interrupted by the non-maskable interrupt request. A maskable interrupt routine should end with an RETI instruction.

2.14 WATCHDOG/CLOCK MONITOR

The devices contain a user selectable WATCHDOG and clock monitor. The following section is applicable only if WATCHDOG feature has been selected in the ECON register. The WATCHDOG is designed to detect the user program getting stuck in infinite loops resulting in loss of program control or “runaway” programs.

The WATCHDOG logic contains two separate service windows. While the user programmable upper window selects the Watchdog service time, the lower widow provides protection against an infinite program loop that contains the watchdog service instruction.

The COPSAx7 devices provide the added feature of a software trap that provides protection against stack overpops and addressing locations outside valid user program space.

The Clock Monitor is used to detect the absence of a clock or a very slow clock below a specified rate on the CKI pin.

The WATCHDOG consists of two independent logic blocks: WD UPPER and WD LOWER. WD UPPER establishes the upper limit on the service window and WD LOWER defines the lower limit of the service window.

Servicing the WATCHDOG consists of writing a specific value to a WATCHDOG Service Register named WDSVR which is memory mapped in the RAM. This value is composed of three fields, consisting of a 2-bit Window Select, a 5-bit Key Data field, and the 1-bit Clock Monitor Select field. Table 2-6 shows the WDSVR register.

Table 2-6 WATCHDOG Service Register (WDSVR)

Window Select		**Key Data**					**Clock Monitor**
X	X	0	1	1	0	0	Y
7	6	5	4	3	2	1	0

The lower limit of the service window is fixed at 256 instruction cycles. Bits 7 and 6 of the WDSVR register allow the user to pick an upper limit of the service window.

Table 2-7 shows the four possible combinations of lower and upper limits for the WATCHDOG service window. This flexibility in choosing the WATCHDOG service window prevents any undue burden on the user software.

Bits 5, 4, 3, 2 and 1 of the WDSVR register represent the 5-bit Key Data field. The key data is fixed at 01100. Bit 0 of the WDSVR Register is the Clock Monitor Select bit.

Table 2-7 WATCHDOG Service Window Select

WDSVR Bit 7	WDSVR Bit 6	Service Window (Lower-Upper Limits)
0	0	256–8k t_c Cycles
0	1	256–16k t_c Cycles
1	0	256–32k t_c Cycles
1	1	256–64k t_c Cycles

2.14.1 Clock Monitor

The Clock Monitor aboard the device can be selected or deselected under program control. The Clock Monitor is guaranteed not to reject the clock if the instruction cycle clock ($1/t_c$) is greater or equal to 10 kHz. This equates to a clock input rate on CKI of greater or equal to 100 kHz.

2.14.2 WATCHDOG/Clock Monitor Operation

The WATCHDOG is enabled by bit 2 of the ECON register. When this ECON bit is 0, the WATCHDOG is enabled and pin G1 becomes the WATCHDOG output with a weak pullup.

The WATCHDOG and Clock Monitor are disabled during reset. The device comes out of reset with the WATCHDOG armed, the WATCHDOG Window Select bits (bits 6, 7 of the WDSVR Register) set, and the Clock Monitor bit (bit 0 of the WDSVR Register) enabled. Thus, a Clock Monitor error will occur after coming out of reset, if the instruction cycle clock frequency has not reached a minimum specified value, including the case where the oscillator fails to start.

The WDSVR register can be written to only once after reset and the key data (bits 5 through 1 of the WDSVR Register) must match to be a valid write. This write to the WDSVR register involves two irrevocable choices: (i) the selection of the WATCHDOG service window (ii) enabling or disabling of the Clock Monitor. Hence, the first write to WDSVR Register involves selecting or deselecting the Clock Monitor, select the WATCHDOG service window and match the WATCHDOG key data. Subsequent writes to the WDSVR register will compare the value being written by the user to the WATCHDOG service window value and the key data (bits 7 through 1) in the WDSVR Register. Table 2-8 shows the sequence of events that can occur.

The user must service the WATCHDOG at least once before the upper limit of the service window expires. The WATCHDOG may not be serviced more than once in every lower limit of the service window. The user may service the WATCHDOG as many times as wished in the time period between the lower and upper limits of the service window. The first write to the WDSVR Register is also counted as a WATCHDOG service.

The WATCHDOG has an output pin associated with it. This is the WDOUT pin, on pin 1 of the port G. WDOUT is active low. The WDOUT pin has a weak pullup in the inactive

Table 2-8 WATCHDOG Service Actions

Key Data	Window Data	Clock Monitor	Action
Match	Match	Match	Valid Service: Restart Service Window
Don't Care	Mismatch	Don't Care	Error: Generate WATCHDOG Output
Mismatch	Don't Care	Don't Care	Error: Generate WATCHDOG Output
Don't Care	Don't Care	Mismatch	Error: Generate WATCHDOG Output

state. Upon triggering the WATCHDOG, the logic will pull the WDOUT (G1) pin low for an additional 16 tc–32 tc cycles after the signal level on WDOUT pin goes below the lower Schmitt trigger threshold. After this delay, the device will stop forcing the WDOUT output low.The WATCHDOG service window will restart when the WDOUT pin goes high.

A WATCHDOG service while the WDOUT signal is active will be ignored. The state of the WDOUT pin is not guaranteed on reset, but if it powers up low then the WATCHDOG will time out and WDOUT will go high.

The Clock Monitor forces the G1 pin low upon detecting a clock frequency error. The Clock Monitor error will continue until the clock frequency has reached the minimum specified value, after which the G1 output will go high following 16 t_c–32 t_c clock cycles. The Clock Monitor generates a continual Clock Monitor error if the oscillator fails to start, or fails to reach the minimum specified frequency. The specification for the Clock Monitor is as follows:

$1/t_c$ > 10 kHz—No clock rejection.

$1/t_c$ < 10 Hz—Guaranteed clock rejection.

2.14.3 WATCHDOG and Clock Monitor Summary

The following salient points regarding the WATCHDOG and CLOCK MONITOR should be noted:

- Both the WATCHDOG and CLOCK MONITOR detector circuits are inhibited during RESET.
- Following RESET, the WATCHDOG and CLOCK MONITOR are both enabled, with the WATCHDOG having he maximum service window selected.
- The WATCHDOG service window and CLOCK MONITOR enable/disable option can only be changed once, during the initial WATCHDOG service following RESET.
- The initial WATCHDOG service must match the key data value in the WATCHDOG Service register WDSVR in order to avoid a WATCHDOG error.
- Subsequent WATCHDOG services must match all three data fields in WDSVR in order to avoid WATCHDOG errors.

- The correct key data value cannot be read from the WATCHDOG Service register WDSVR. Any attempt to read this key data value of 01100 from WDSVR will read as key data value of all 0's.
- The WATCHDOG detector circuit is inhibited during both the HALT and IDLE modes.
- The CLOCK MONITOR detector circuit is active during both the HALT and IDLE modes. Consequently, the device inadvertently entering the HALT mode will be detected as a CLOCK MONITOR error (provided that the CLOCK MONITOR enable option has been selected by the program).
- With the single-pin R/C oscillator option selected and the CLKDLY bit reset, the WATCHDOG service window will resume following HALT mode from where it left off before entering the HALT mode.
- With the crystal oscillator option selected, or with the single-pin R/C oscillator option selected and the CLKDLY bit set, the WATCHDOG service window will be set to its selected value from WDSVR following HALT. Consequently, the WATCHDOG should not be serviced for at least 256 instruction cycles following HALT, but must be serviced within the selected window to avoid a WATCHDOG error.
- The IDLE timer T0 is not initialized with external RESET.
- The user can sync in to the IDLE counter cycle with an IDLE counter (T0) interrupt or by monitoring the T0PND flag. The T0PND flag is set whenever the twelfth bit of the IDLE counter toggles (every 4096 instruction cycles). The user is responsible for resetting the T0PND flag.
- A hardware WATCHDOG service occurs just as the device exits the IDLE mode. Consequently, the WATCHDOG should not be serviced for at least 256 instruction cycles following IDLE, but must be serviced within the selected window to avoid a WATCHDOG error.
- Following RESET, the initial WATCHDOG service (where the service window and the CLOCK MONITOR enable/disable must be selected) may be programmed anywhere within the maximum service window (65,536 instruction cycles) initialized by RESET. Note that this initial WATCHDOG service may be programmed within the initial 256 instruction cycles without causing a WATCHDOG error.

2.14.4 Detection of Illegal Conditions

The device can detect various illegal conditions resulting from coding errors, transient noise, power supply voltage drops, runaway programs, etc.

Reading of undefined ROM gets zeros. The opcode for software interrupt is 00. If the program fetches instructions from undefined ROM, this will force a software interrupt, thus signaling that an illegal condition has occurred.

The subroutine stack grows down for each call (jump to subroutine), interrupt, or PUSH, and grows up for each return or POP. The stack pointer is initialized to RAM location 06F Hex during reset. Consequently, if there are more returns than calls, the stack pointer will point to addresses 070 and 071 Hex (which are undefined RAM). Undefined RAM

from addresses 070 to 07F (Segment 0), and all other segments (i.e., Segments 4... etc.) is read as all 1's, which in turn will cause the program to return to address 7FFF Hex. This is an undefined ROM location and the instruction fetched (all 0's) from this location will generate a software interrupt signaling an illegal condition.

Thus, the chip can detect the following illegal conditions:

1. Executing from undefined ROM
2. Over “POP”ing the stack by having more returns than calls.

When the software interrupt occurs, the user can re-initialize the stack pointer and do a recovery procedure before restarting (this recovery program is probably similar to that following reset, but might not contain the same program initialization procedures). The recovery program should reset the software interrupt pending bit using the RPND instruction.

2.15 MICROWIRE/PLUS

MICROWIRE/PLUS is a serial SPI compatible synchronous communications interface. The MICROWIRE/PLUS capability enables the device to interface with MICROWIRE/PLUS or SPI peripherals (i.e. A/D converters, display drivers, EEPROMs etc.) and with other microcontrollers which support the MICROWIRE/PLUS or SPI interface. It consists of an 8-bit serial shift register (SIO) with serial data input (SI), serial data output (SO) and serial shift clock (SK). Figure 2-25 shows a block diagram of the MICROWIRE/PLUS logic.

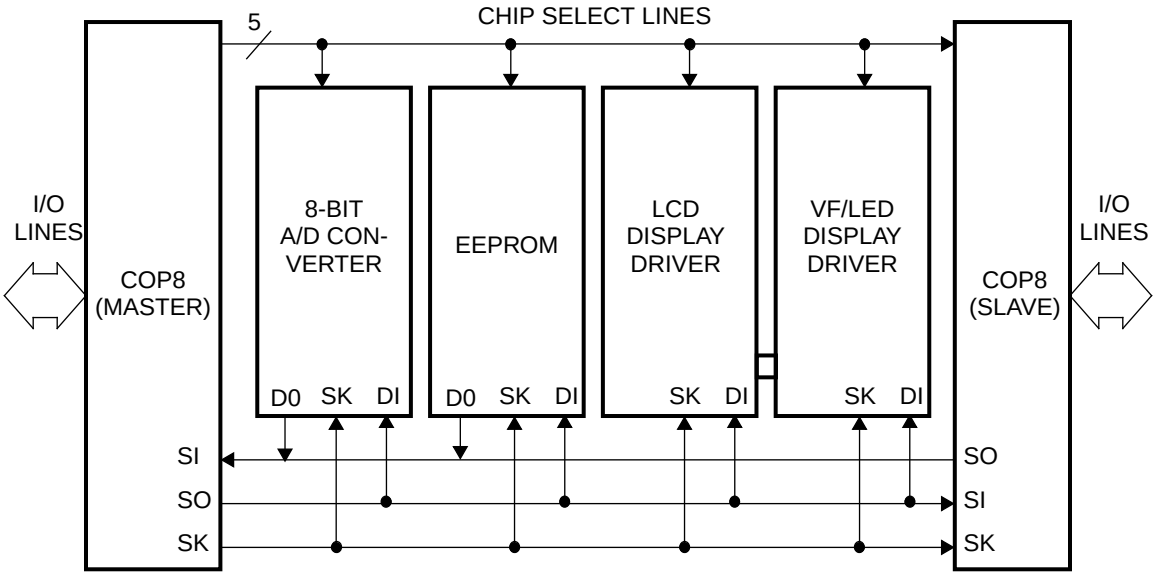


Figure 2-25 MICROWIRE/PLUS Application

The shift clock can be selected from either an internal source or an external source. Operating the MICROWIRE/PLUS arrangement with the internal clock source is called the Master mode of operation. Similarly, operating the MICROWIRE/PLUS arrangement with an external shift clock is called the Slave mode of operation.

The CNTRL register is used to configure and control the MICROWIRE/PLUS mode. To use the MICROWIRE/PLUS, the MSEL bit in the CNTRL register is set to one. In the master mode, the SK clock rate is selected by the two bits, SL0 and SL1, in the CNTRL register. Table 2-9 details the different clock rates that may be selected.

Table 2-9 MICROWIRE/PLUS Master Mode Clock Select

SL1	SL0	SK period
0	0	2 X t_c
0	1	4 X t_c
1	x	8 X t_c

Where t_c is the instruction cycle clock

2.15.1 MICROWIRE/PLUS Operation

Setting the BUSY bit in the PSW register causes the MICROWIRE/PLUS to start shifting the data. It gets reset when eight data bits have been shifted. The user may reset the BUSY bit by software to allow less than 8 bits to shift. If enabled, an interrupt is generated when eight data bits have been shifted. The device may enter the MICROWIRE/PLUS mode either as a Master or as a Slave. Figure 2-25 shows how two microcontroller devices and several peripherals may be interconnected using the MICROWIRE/PLUS arrangements.

WARNING

The SIO register should only be loaded when the SK clock is in the idle phase. Loading the SIO register while the SK clock is in the active phase, will result in undefined data in the SIO register.

Setting the BUSY flag when the input SK clock is in the active phase while in the MICROWIRE/PLUS is in the slave mode may cause the current SK clock for the SIO shift register to be narrow. For safety, the BUSY flag should only be set when the input SK clock is in the idle phase.

MICROWIRE/PLUS Master Mode Operation

In the MICROWIRE/PLUS Master mode of operation the shift clock (SK) is generated internally. The MICROWIRE Master always initiates all data exchanges. The MSEL bit in the CNTRL register must be set to enable the SO and SK functions onto the G Port. The SO and SK pins must also be selected as outputs by setting appropriate bits in the Port G configuration register. In the slave mode, the shift clock stops after 8 clock pulses. Table 2-10 summarizes the bit settings required for Master mode of operation.

Table 2-10 MICROWIRE/PLUS Mode Settings

This table assumes that the control flag MSEL is set.

G4 (SO) Config. Bit	G5 (SK) Config. Bit	G4 Fun.	G5 Fun.	Operation
1	1	SO	Int. SK	MICROWIRE/PLUS Master
0	1	TRI-STATE	Int. SK	MICROWIRE/PLUS Master
1	0	SO	Ext. SK	MICROWIRE/PLUS Slave
0	0	TRI-STATE	Ext. SK	MICROWIRE/PLUS Slave

MICROWIRE/PLUS Slave Mode Operation

In the MICROWIRE/PLUS Slave mode of operation the SK clock is generated by an external source. Setting the MSEL bit in the CNTRL register enables the SO and SK functions onto the G Port. The SK pin must be selected as an input and the SO pin is selected as an output pin by setting and resetting the appropriate bits in the Port G configuration register. Table 2-10 summarizes the settings required to enter the Slave mode of operation.

The user must set the BUSY flag immediately upon entering the Slave mode. This ensures that all data bits sent by the Master is shifted properly. After eight clock pulses the BUSY flag is clear, the shift clock is stopped, and the sequence may be repeated.

Alternate SK Phase Operation and SK Idle Polarity

The device allows either the normal SK clock or an alternate phase SK clock to shift data in and out of the SIO register. In both the modes the SK idle polarity can be either high or low. The polarity is selected by bit 5 of Port G data register. In the normal mode data is shifted in on the rising edge of the SK clock and the data is shifted out on the falling edge of the SK clock. The SIO register is shifted on each falling edge of the SK clock. In the alternate SK phase operation, data is shifted in on the falling edge of the SK clock and shifted out on the rising edge of the SK clock. Bit 6 of Port G configuration register selects the SK edge.

A control flag, SKSEL, allows either the normal SK clock or the alternate SK clock to be selected. Resetting SKSEL causes the MICROWIRE/PLUS logic to be clocked from the normal SK signal. Setting the SKSEL flag selects the alternate SK clock. The SKSEL is mapped into the G6 configuration bit. The SKSEL flag will power up in the reset condition, selecting the normal SK signal.

Table 2-11 MICROWIRE/PLUS Shift Clock Polarity and Sample/Shift Phase

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

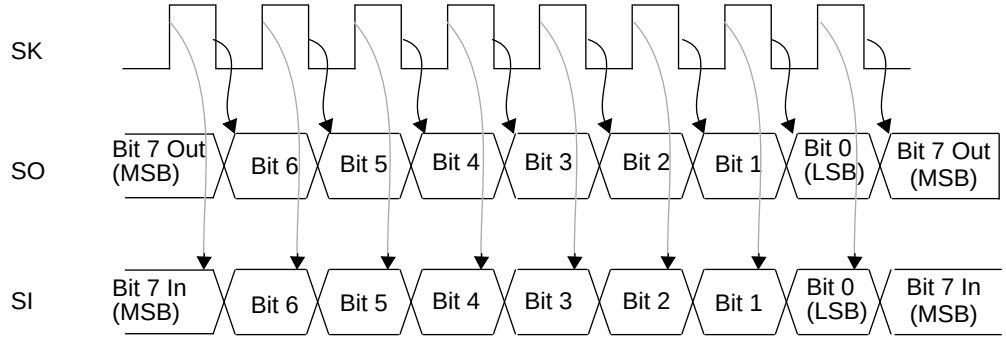


Figure 2-26 MICROWIRE/PLUS SPI Mode Interface Timing, Normal SK Mode, SK Idle Phase being Low

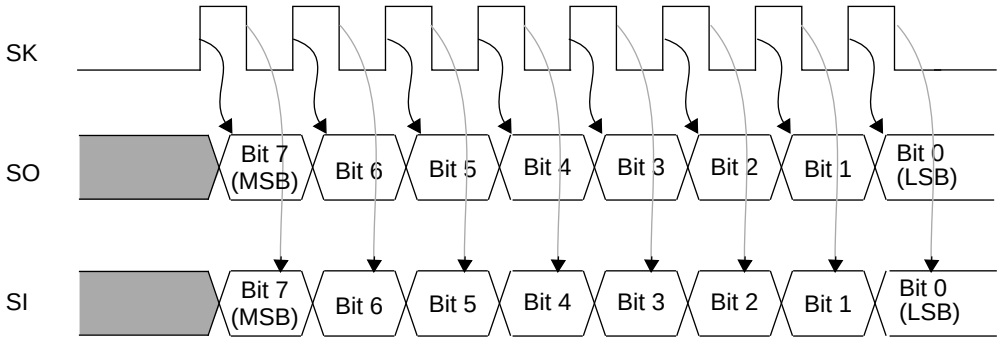


Figure 2-27 MICROWIRE/PLUS SPI Mode Interface Timing, Alternate SK Mode, SK Idle Phase being Low

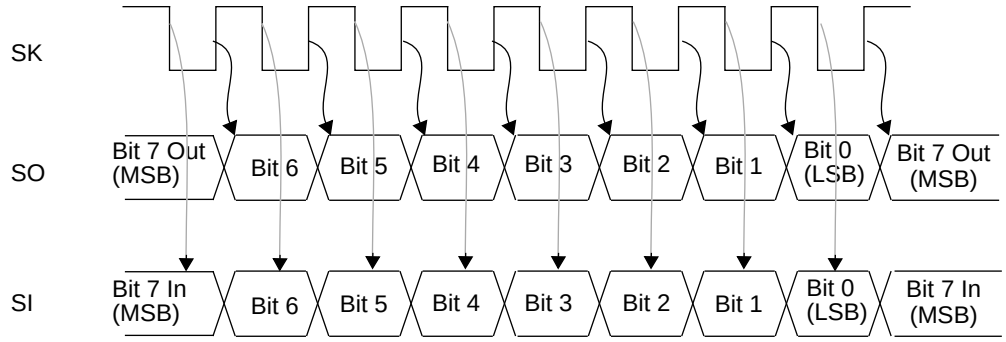


Figure 2-28 MICROWIRE/PLUS SPI Mode Interface Timing, Alternate SK Mode, SK Idle Phase being High

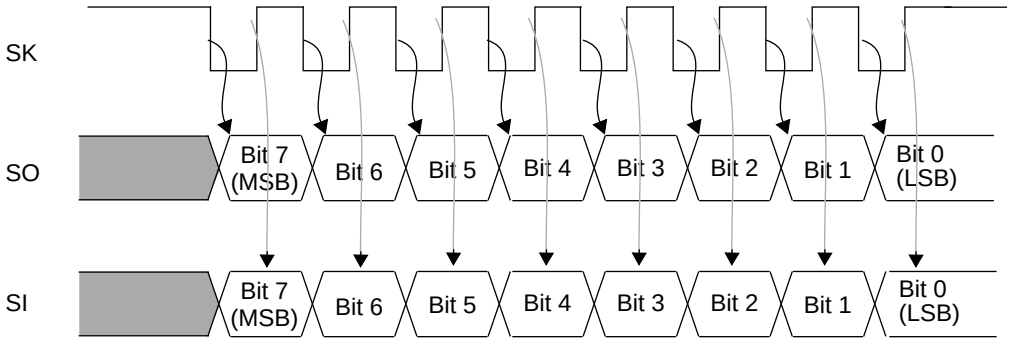


Figure 2-29 MICROWIRE/PLUS SPI Mode Interface Timing, Normal SK Mode, SK Idle Phase being High

2.16 MEMORY MAP

All RAM, ports and registers (except A and PC) are mapped into data memory address space.

RAM Select	Address ADD REG	Contents
64 On-chip RAM Bytes. Selected by ECON Register	00 to 2F	On-chip RAM (48 Bytes)
	30 to 7F	Unused RAM (Reads as all ones)
128 On-chip RAM Bytes.	00 to 6F	On-chip RAM (112 Bytes)
	70 to 7F	Unused RAM (Reads as all ones)
	94	Port F Data Register
	95	Port F Configuration Register
	96	Port F Input Pins (read only)
	97	Reserved
	C7	WATCHDOG Service Register (Reg:WDSVR)
	C8	MIWU Edge Select Register (Reg:WKEDG)
	C9	MIWU Enable Register (Reg:WKEN)
	CA	MIWU Pending Register (Reg:WKPND)
	CB to CF	Reserved
	D0	Port L Data Register
	D1	Port L Configuration Register
	D2	Port L Input Pins (Read Only)
	D3	Reserved
	D4	Port G Data Register
	D5	Port G Configuration Register
	D6	Port G Input Pins (Read Only)
	D7	Reserved
	D8	Port C Data Register
	D9	Port C Configuration Register
	DA	Port C Input Pins (Read Only)

RAM Select	Address ADD REG	Contents
	DB	Reserved
	DC	Port D
	DD to DF	Reserved
	E0 to E5	Reserved
	E6	Timer T1 Autoload Register T1RB Lower Byte
	E7	Timer T1 Autoload Register T1RB Upper Byte
	E8	ICNTRL 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
	F0 to FB	On-Chip RAM Mapped as Registers
	FC	X Register
	FD	SP Register
	FE	B Register
	FF	Segment Register

Reading any undefined memory location in the address range of 0080H–00FFH will return undefined data.

2.17 INSTRUCTION SET

2.17.1 Introduction

This section defines the instruction set of the COPSAx7 Family members. It contains information about the instruction set features, addressing modes and types.

2.17.2 Instruction Features

The strength of the instruction set is based on the following features:

- Mostly single-byte opcode instructions 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 that 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.

2.17.3 Addressing Modes

The 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.

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

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

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.

2.17.4 Instruction Types

The instruction set contains a 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.

Jump Relative (JP)

Jump Absolute (JMP)

Jump Absolute Long (JMPL)

Jump Indirect (JID)

Jump to Subroutine (JSR)

Jump to Subroutine Long (JSRL)

Return from Subroutine (RET)

Return from Subroutine and Skip (RETSK)

Return from Interrupt (RETI)

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 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)

NOTE: The VIS 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 the program memory into the program counter (PC) in order to jump to the associated interrupt service routine.

2.18 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 COP8 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 memory at addresses 00F0-00FF.
REG#	# the of 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.

2.18.1 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

2.18.2 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.#

2.18.3 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

2.18.4 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.#

2.18.5 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

2.18.6 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

2.18.7 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

2.18.8 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#)

2.18.9 IFBIT — Test Bit

Syntax: a)IFBIT #,[B]

b)IFBIT #,MD

c)IFBIT #,A

Description: The selected bit (# = 0 to 7, with 7 being the high-order bit) 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

2.18.10 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#

2.18.11 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

2.18.12 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.#

2.18.13 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

2.18.14 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

2.18.15 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

2.18.16 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

2.18.17 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 COP888 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. The contents of PCU (Upper 7 bits of PC) are 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 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) 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

2.18.18 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 program memory (vector table and the new instruction) will be fetched from the next page of 256 bytes.

Operation: PCL <- A

PCL <- Program Memory (PCU,A)

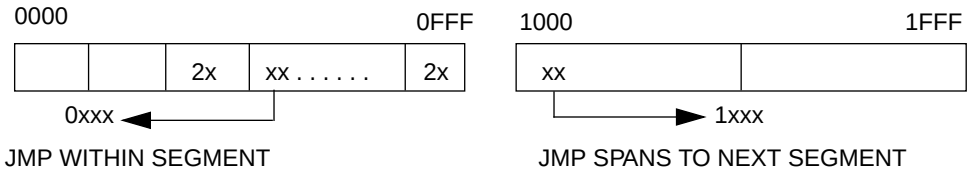
Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JID	Indirect	3	1	A5

2.18.19 JMP — Jump Absolute

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.

The address range is 15 bits. The JMP instruction address contents is only 12 bits. Resolution of the high 3 bits of the address is to the 4K memory segment containing the **low byte of the instruction**. The following diagram illustrates:



If instruction byte 2 is in 4K segment 0, the jump address is in segment 0.

If instruction byte 2 is in 4K segment 1, the jump address is in segment 1.

Operation: PC11-8 <- HIADDR (HIGH NIBBLE OF SECOND BYTE OF INSTRUCTION, 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

2.18.20 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

2.18.21 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)

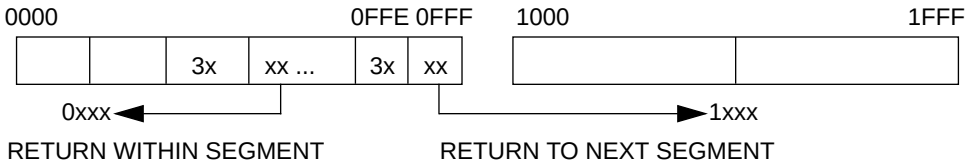
Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JP DISP	Relative	3	1	0, 1, E, F + DISP #

2.18.22 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.

The address range is 15 bits. The JSR instruction address contents is only 12 bits. Resolution of the high 3 bits of the address is to the 4K memory segment containing the **high byte of the return address**. A JSR contained in the last 2 bytes of a 4K segment will jump to the next segment. The following diagram illustrates:



If the return address is in 4K segment 0, the jump address is in segment 0.

If the return address is in 4K segment 1, the jump address is in segment 1.

Operation: [SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: SET UP FOR NEXT STACK REFERENCE

PC11-8 <- HIADDR (HIGH NIBBLE OF RETURN ADDRESS, LOW NIBBLE OF FIRST BYTE INSTRUCTION)

PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSR ADDR	Absolute	5	2	3HIADDR/LOADDR

2.18.23 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

2.18.24 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 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 <- Program Memory (PCU, A)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LAID	Indirect	3	1	A4

2.18.25 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

2.18.26 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.#

2.18.27 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.#

2.18.28 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.#

2.18.29 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

2.18.30 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

2.18.31 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

2.18.32 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

2.18.33 RBIT — Reset Memory Bit

Syntax: a)RBIT #,[B]

b)RBIT #,MD

Description: The selected bit (# = 0 to 7, with 7 being the high-order bit) 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+#)

2.18.34 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

2.18.35 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

2.18.36 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

2.18.37 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

2.18.38 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

2.18.39 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

ELSE STPND <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RPND	Implicit	1	1	B5

2.18.40 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

2.18.41 SBIT — Set Memory Bit

Syntax: a)SBIT #,[B]

b)SBIT #,MD

Description: The selected bit (# = 0 to 7, with 7 being the high-order bit) 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+#)

2.18.42 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

2.18.43 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: $A \leftarrow A + \overline{VALUE} + C$

C <- ABSENCE OF BYTE BORROW

HC <- ABSENCE OF LOW NIBBLE HALF 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

2.18.44 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

2.18.45 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 <- Program Memory (PCU,VA)

PCL <- Program Memory (PCU,VA+1)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
VIS	Implicit	5	1	B4

2.18.46 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

2.18.47 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

2.18.48 Register and Symbol Definition

The following abbreviations represent the nomenclature used in the instruction description and the COP8 cross-assembler

Registers	
A	8-Bit Accumulator Register
B	8-Bit Address Register
X	8-Bit Address Register
SP	8-Bit Stack Pointer Register
PC	15-Bit Program Counter Register
PU	Upper 7 Bits of PC
PL	Lower 8 Bits of PC
C	1 Bit of PSW Register for Carry
HC	1 Bit of PSW Register for Half Carry
GIE	1 Bit of PSW Register for Global Interrupt Enable
VU	Interrupt Vector Upper Byte
VL	Interrupt Vector Lower Byte
Symbols	
[B]	Memory Indirectly Addressed by B Register
[X]	Memory Indirectly Addressed by X Register
MD	Direct Addressed Memory
Mem	Direct Addressed Memory or [B]
Meml	Direct Addressed Memory or [B] or Immediate Data
Imm	8-Bit Immediate Data
Reg	Register Memory: Addresses F0 to FF (Includes B, X and SP)
Bit	Bit Number (0 to 7)
←	Loaded with
↔	Exchanged with

2.18.49 Instruction Set Summary

ADD	A,Meml	ADD	A ⟵ A + Meml
ADC	A,Meml	ADD with Carry	A ⟵ A + Meml + C, C ⟵ Carry, HC ⟵ Half Carry
SUBC	A,Meml	Subtract with Carry	A ⟵ A - Meml + C, C ⟵ Carry, HC ⟵ Half Carry
AND	A,Meml	Logical AND	A ⟵ A and $\overline{\text{Meml}}$
ANDSZ	A,Imm	Logical AND Immed., Skip if Zero	Skip next if (A and Imm) = 0
OR	A,Meml	Logical OR	A ⟵ A or Meml
XOR	A,Meml	Logical EXclusive OR	A ⟵ A xor Meml
IFEQ	MD,Imm	IF EQual	Compare MD and Imm, Do next if MD = Imm
IFEQ	A,Meml	IF EQual	Compare A and Meml, Do next if A = Meml
IFNE	A,Meml	IF Not Equal	Compare A and Meml, Do next if A ≠ Meml
IFGT	A,Meml	IF Greater Than	Compare A and Meml, Do next if A > Meml
IFBNE	#	If B Not Equal	Do next if lower 4 bits of B ≠ Imm
DRSZ	Reg	Decrement Reg., Skip if Zero	Reg ⟵ Reg - 1, Skip if Reg = 0
SBIT	#,Mem	Set BIT	1 to bit, Mem (bit = 0 to 7 immediate)
RBIT	#,Mem	Reset BIT	0 to bit, Mem
IFBIT	#,Mem	IF BIT	If bit #,A or Mem is true do next instruction
RPND		Reset PeNDing Flag	Reset Software Interrupt Pending Flag
X	A,Mem	EXchange A with Memory	A ⟷ Mem
X	A,[X]	EXchange A with Memory [X]	A ⟷ [X]
LD	A,Meml	LoaD A with Memory	A ⟵ Meml
LD	A,[X]	LoaD A with Memory [X]	A ⟵ [X]
LD	B,Imm	LoaD B with Immed.	B ⟵ Imm
LD	Mem,Imm	LoaD Memory Immed	Mem ⟵ Imm
LD	Reg,Imm	LoaD Register Memory Immed.	Reg ⟵ Imm
X	A, [B ±]	EXchange A with Memory [B]	A ⟷ [B], (B ⟵ B ±1)
X	A, [X±]	EXchange A with Memory [X]	A ⟷ [X], (X ⟵X±1)
LD	A, [B±]	LoaD A with Memory [B]	A ⟵ [B], (B ⟵ B ±1)
LD	A, [X±]	LoaD A with Memory [X]	A ⟵ [X], (X ⟵ X±1)
LD	[B±],Imm	LoaD Memory [B] Immed.	[B] ⟵ Imm, (B ⟵ B±1)
CLR	A	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 of A (follows ADC, SUBC)
RRC	A	Rotate A Right thru C	C ⟶ A7 ⟶ ... ⟶ A0 ⟶ C
RLC	A	Rotate A Left thru C	C ⟵ A7 ⟵... ⟵ A0 ⟵ C, HC ⟵ A0
SWAP	A	SWAP nibbles of A	A7...A4 ⟷ A3...A0
SC		Set C	C ⟵ 1, HC ⟵ 1
RC		Reset C	C ⟵ 0, HC ⟵ 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 Routine	PU ⟵ [VU], PL⟵[VL]
JMPL	Addr.	Jump absolute Long	PC ⟵ ii (ii = 15 bits, 0 to 32k)
JMP	Addr.	Jump absolute	PC9...0 ⟵ i (i = 12 bits)
JP	Disp.	Jump relative short	PC ⟵ PC + r (r is -31 to +32, except 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, PC9...0 ⟵ i
JID		Jump InDirect	PL⟵ROM (PU,A)
RET		RETurn from subroutine	SP + 2, PL⟵ [SP], PU⟵ [SP-1]
RETSK		RETurn and SKip	SP + 2, PL⟵ [SP],PU⟵ [SP-1], skip next instruction
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

2.18.50 Instruction Execution Time

Most instructions are single byte (with immediate addressing mode instructions taking two bytes).

Most single byte instructions take one cycle time to execute.

Skipped instructions require x number of cycles to be skipped, where x equals the number of bytes in the skipped instruction opcode.

See the BYTES and CYCLES per INSTRUCTION table for details.

Bytes and Cycles per Instruction

The following table shows the number of bytes and cycles for each instruction in the format of byte/cycle.

Arithmetic and Logic Instructions

	[B]	Direct	Immed.
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
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

Instructions Using A & C

CLRA	1/1
INCA	1/1
DECA	1/1
LAID	1/3
DCORA	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

Transfer of Control Instructions

JMPL	3/4
JMP	2/3
JP	1/3
JSRL	3/5
JSR	2/5
JID	1/3
VIS	1/5
RET	1/5
RETSK	1/5
RETI	1/5
INTR	1/7
NOP	1/1

Memory Transfer Instructions

	Register Indirect		Direct	Immed.	Register Indirect Auto Incr & Decr		
	[B]	[X]			[B+, B-]	[X+, X-]	
X A,*	1/1	1/3	2/3		1/2	1/3	
LD A,*	1/1	1/3	2/3	2/2	1/2	1/3	
LD B,Imm				1/1			(If B < 16)
LD B,Imm				2/2			(If B > 15)
LD Mem,Imm	2/2		3/3		2/2		
LD Reg,Imm			2/3				
IFEQ MD,Imm			3/3				

* => Memory location addressed by B or X or directly.

2.18.51 Opcode Table

UPPER NIBBLE	LOWER NIBBLE																
		D	C	B	A	9	8	7	6	5	4	3	2	1	0		
F	E	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-15	JP-31	LD 0F1,#i	DRSZ 0F1	*	SC	SUBC A,#i	SUBC A,[B]	IFBIT 1,[B]	*	LD B,#0E	IFBNE 1	JSR x100-x1FF	JMP x100-x1FF	JP+18	JP + 2	1	
JP-14	JP-30	LD 0F2,#i	DRSZ 0F2	X A,[X+]	X A,[B+]	IFEQ A,#i	IFEQ A,[B]	IFBIT 2,[B]	*	LD B,#0D	IFBNE 2	JSR x200-x2FF	JMP x200-x2FF	JP+19	jp + 3	2	
JP-13	JP-29	LD 0F3,#i	DRSZ 0F3	X A,[X-]	X A,[B-]	IFGT A,#i	IFGT A,[B]	IFBIT 3,[B]	*	LD B,#0C	IFBNE 3	JSR x300-x3FF	JMP x300-x3FF	JP+20	JP + 4	3	
JP-12	JP-28	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-11	JP-27	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-10	JP-26	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-9	JP-25	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-8	JP-24	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-7	JP-23	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-6	JP-22	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-5	JP-21	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-4	JP-20	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-3	JP-19	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-2	JP-18	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-1	JP-17	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	
JP-0	JP-16																

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

2.19 PROGRAMMING EXAMPLES

This section is intended to be an overview of programming examples. For more detailed and varied programming examples, refer to the Microontroller COP8 Databook.

2.19.1 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.

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
```

2.19.2 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
```

2.19.3 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.

```
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
```

2.19.4 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.

```
DIVIDE (16÷16) 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.

```
DIVIDE (24÷8) 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
```

2.20 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	7V
Voltage at Any Pin	–0.6V to V_{CC} +0.6V
ESD Protection Level	2KV (Human Body Model)
Total Current into V_{CC} Pin (Source)	80 mA
Total Current out of GND Pin (Sink)	100 mA
Storage Temperature Range	–65˚C to +140˚C

NOTE: *Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

2.20.1 DC Electrical Characteristics (0˚C ≤ T_A ≤ +70˚C unless otherwise specified)

Parameter	Conditions	Min	Typ	Max	Units
Operating Voltage		2.7		5.5	V
Power Supply Rise Time (On-Chip Power-on Reset Selected)		10 ns		50 ms	V
Power Supply Ripple (Note 1)	Peak-to-Peak			0.1 V_{CC}	V
Supply Current (Note 2)					
CKI = 10 MHz	V_{CC} = 5.5V, t_c = 1 µs			6	mA
CKI = 4 MHz	V_{CC} = 4.5V, t_c = 2.5 µs			2.1	mA
HALT Current WATCHDOG Disable(Note 3)	V_{CC} = 5.5V, CKI = 0 MHz		<4	8	µA
IDLE Current (Note 2)					
CKI = 10 MHz	V_{CC} = 5.5V, t_c = 1 µs			1.5	mA
CKI = 4 MHz	V_{CC} = 4.5V, t_c = 2.5 µs			0.8	mA
Input Levels (V_{IH}, V_{IL})					
RESET					
Logic High		0.8 V_{CC}			V
Logic Low				0.2 V_{CC}	V
CKI, All Other Inputs					
Logic High		0.7 V_{CC}			V
Logic Low				0.2 V_{CC}	V
Value of the Internal Bias Resistor for the Crystal/ Resonator Oscillator		0.5	1.0	2.0	MΩ
CKI Resistance to V_{CC} or GND when R/C oscillator is selected	V_{CC} = 5.5V	5	8	11	kΩ
Hi-Z Input Leakage (Same as TRI-STATE output)	V_{CC} = 5.5V	–2		+2	µA
Input Pullup Current	V_{CC} = 5.5V, V_{IN} = 0V	–40		–250	µA
G and L Port Input Hysteresis		0.25 V_{CC}			V
Output Current Levels					
D Outputs					
Source	V_{CC} = 4.5V, V_{OH} = 3.3V	–0.4			mA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–0.2			mA
Sink	V_{CC} = 4.5V, V_{OL} = 1.0V	10			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	2			mA
L Port					
Source (Weak Pull-Up)	V_{CC} = 4.5V, V_{OH} = 2.7V	–10		–110	µA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–2.5		–33	µA

Parameter	Conditions	Min	Typ	Max	Units
Source (Push-Pull Mode)	V_{CC} = 4.5V, V_{OH} = 3.3V	–0.4			mA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–0.2			mA
Sink (L0-L3, Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 1.0V	10			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	2			mA
Sink (L4-L7, Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 0.4V	1.6			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	0.7			mA
All Others					
Source (Weak Pull-Up Mode)	V_{CC} = 4.5V, V_{OH} = 2.7V	–10		–110	µA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–2.5		–33	µA
Source (Push-Pull Mode)	V_{CC}= 4.5V, V_{OH} = 3.3V	–0.4			mA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–0.2			mA
Sink (Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 0.4V	1.6			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	0.7			mA
Allowable Sink Current per Pin (Note 6)					
D Outputs and L0 to L3				15	mA
All others				3	mA
Maximum Input Current without Latchup (Note 4)				±200	mA
RAM Retention Voltage, Vr		2.0			V
V_{CC} rise time from a $V_{CC} \geq$ 2.0V		1.2			µs
Input Capacitance	(Note 6)			7	pF
Load Capacitance on D2	(Note 6)			1000	pF

2.20.2 AC Electrical Characteristics (0˚C ≤ T_A ≤ +70˚C unless otherwise specified)

Parameter	Conditions	Min	Typ	Max	Units
Instruction Cycle Time (t_c)					
Crystal/Resonator, External	4.5V ≤ V_{CC} ≤ 5.5V	1.0		DC	µs
	2.7V ≤ V_{CC} < 4.5V	2.0		DC	µs
Internal R/C Oscillator	4.5V ≤ V_{CC} ≤ 5.5V		2.0		µs
	2.7V ≤ V_{CC} < 4.5V		TBD		µs
R/C Oscillator Frequency Variation	4.5V ≤ V_{CC} ≤ 5.5V			±35	%
(Note 6)	2.7V ≤ V_{CC} < 4.5V			TBD	%
External CKI Clock Duty Cycle (Note 6)	fr = Max	45		55	%
Rise Time (Note 6)	fr = 10 MHz Ext Clock			12	ns
Fall Time (Note 6)	fr = 10 MHz Ext Clock			8	ns
Inputs					
t_{SETUP}	4.5V ≤ V_{CC} ≤ 5.5V	200			ns
	2.7V ≤ V_{CC} < 4.5V	500			ns
t_{HOLD}	4.5V ≤ V_{CC} ≤ 5.5V	60			ns
	2.7V ≤ V_{CC} < 4.5V	150			ns
Output Propagation Delay (Note 5)	R_L = 2.2k, C_L = 100 pF				
t_{PD1}, t_{PD0}					
SO, SK	4.5V ≤ V_{CC} ≤ 5.5V			0.7	µs
	2.7V ≤ V_{CC} < 4.5V			1.75	µs
All Others	4.5V ≤ V_{CC} ≤ 5.5V			1.0	µs
	2.7V ≤ V_{CC} < 4.5V			2.5	µs
MICROWIRE Setup Time (t_{UWS}) (Note 5)		20			ns
MICROWIRE Hold Time (t_{UWH}) (Note 5)		56			ns
MICROWIRE Output Propagation Delay (t_{UPD})				220	ns
MICROWIRE Maximum Shift Clock					
Master Mode				500	kHz
Slave Mode				1	MHz
Input Pulse Width (Note 6)					
Interrupt Input High Time		1			t_c
Interrupt Input Low Time		1			t_c
Timer Input High Time		1			t_c
Timer Input Low Time		1			t_c
Reset Pulse Width		1			µs

t_C = Instruction cycle time (Clock Input frequency divided by 10)

Note 1: Maximum rate of voltage change must be < 0.5 V/ms.

Note 2: Supply and IDLE currents are measured with CKI driven with a square wave Oscillator, CKO driven 180˚ out of phase with CKI, inputs connected to V_{CC} and outputs driven low but not connected to a load.

Note 3: The HALT mode will stop CKI from oscillating in the R/C and the Crystal configurations. In the R/C configuration, CKI is forced high internally. In the crystal or external configuration, CKI is TRI-STATE. Measurement of I_{DD} HALT is done with device neither sourcing nor sinking current; with L, F, C, G0, and G2-G5 programmed as low outputs and not driving a load; all outputs programmed low and not driving a load; all inputs tied to V_{CC}; WATCHDOG and clock monitor disabled. Parameter refers to HALT mode entered via setting bit 7 of the G Port data register.

Note 4: Pins G6 and $\overline{\text{RESET}}$ are designed with a high voltage input network. These pins allow input voltages > V_{CC} and the pins will have sink current to V_{CC} when biased at voltages > V_{CC} (the pins do not have source current when biased at a voltage below V_{CC}). The effective resistance to V_{CC} is 750Ω (typical). These two pins will not latch up. The voltage at the pins must be limited to < 14 Volts. WARNING: Voltages in excess of 14 volts will cause damage to the pins. This warning excludes ESD transients.

Note 5: The output propagation delay is referenced to the end of the instruction cycle where the output change occurs.

Note 6: Parameter characterized but not tested.

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	7V
Voltage at Any Pin	–0.6V to V_{CC} +0.6V
ESD Protection Level	2KV (Human Body Model)
Total Current into V_{CC} Pin (Source)	80 mA
Total Current out of GND Pin (Sink)	100 mA
Storage Temperature Range	–65˚C to +140˚C

NOTE: *Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

2.20.3 DC Electrical Characteristics (–40˚C ≤ TA ≤ +85˚C unless otherwise specified)

Parameter	Conditions	Min	Typ	Max	Units
Operating Voltage		2.7		5.5	V
Power Supply Rise Time (On-Chip Power-on Reset Selected)		10 ns		50 ms	
Power Supply Ripple (Note 1)	Peak-to-Peak			0.1 V_{CC}	V
Supply Current (Note 2)					
CKI = 10 MHz	V_{CC} = 5.5V, t_c = 1 µs			6.0	mA
HALT Current WATCHDOG Disable (Note 3)	V_{CC} = 5.5V, CKI = 0 MHz		<4	10.0	µA
IDLE Current (Note 2)					
CKI = 10 MHz	V_{CC} = 5.5V, t_c = 1 µs			1.5	mA
Input Levels (V_{IH}, V_{IL})					
RESET					
Logic High		0.8 V_{CC}			V
Logic Low				0.2 V_{CC}	V
CKI, All Other Inputs					
Logic High		0.7 V_{CC}			V
Logic Low				0.2 V_{CC}	V
Value of the Internal Bias Resistor for the Crystal/ Resonator Oscillator		0.5	1.0	2.0	MΩ
CKI Resistance to V_{CC} or GND when R/C oscillator is selected	V_{CC} = 5.5V	5	8	11	kΩ
Hi-Z Input Leakage (Same as TRI-STATE output)	V_{CC} = 5.5V	–2		+2	µA
Input Pullup Current	V_{CC} = 5.5V, V_{IN} = 0V	–40		–250	µA
G and L Port Input Hysteresis		0.25 V_{CC}			V
Output Current Levels					
D Outputs					
Source	V_{CC} = 4.5V, V_{OH} = 3.3V	–0.4			mA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–0.2			mA
Sink	V_{CC} = 4.5V, V_{OL} = 1.0V	10			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	2			mA
L Port					
Source (Weak Pull-Up)	V_{CC} = 4.5V, V_{OH} = 2.7V	–10.0		–110	µA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–2.5		–33	µA
Source (Push-Pull Mode)	V_{CC} = 4.5V, V_{OH} = 3.3V	–0.4			mA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–0.2			mA
Sink (L0-L3, Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 1.0V	10.0			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	2			mA
Sink (L4-L7, Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 0.4V	1.6			mA

Parameter	Conditions	Min	Typ	Max	Units
	V_{CC} = 2.7V, V_{OL} = 0.4V	0.7			mA
All Others					
Source (Weak Pull-Up Mode)	V_{CC} = 4.5V, V_{OH} = 2.7V	–10.0		–110	µA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–2.5		–33	µA
Source (Push-Pull Mode)	V_{CC}= 4.5V, V_{OH} = 3.3V	–0.4			mA
	V_{CC} = 2.7V, V_{OH} = 1.8V	–0.2			mA
Sink (Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 0.4V	1.6			mA
	V_{CC} = 2.7V, V_{OL} = 0.4V	0.7			mA
Allowable Sink Current per Pin (Note 6)					
D Outputs and L0 to L3				15	mA
All others				3	mA
Maximum Input Current without Latchup (Note 4)				±200	mA
RAM Retention Voltage, Vr		2.0			V
V_{CC} rise time from a $V_{CC} \geq$ 2.0V		1.2			µs
Input Capacitance	(Note 6)			7	pF
Load Capacitance on D2	(Note 6)			1000	pF

2.20.4 AC Electrical Characteristics ($-40^{\circ}C \le T_A \le +85^{\circ}C$ unless otherwise specified)

Parameter	Conditions	Min	Typ	Max	Units
Instruction Cycle Time (t_c)					
Crystal/Resonator, External	$4.5V \le V_{CC} \le 5.5V$	1.0		DC	µs
	$2.7V \le V_{CC} < 4.5V$	2.0		DC	µs
Internal R/C Oscillator	$4.5V \le V_{CC} \le 5.5V$		2.0		µs
	$2.7V \le V_{CC} < 4.5V$		TBD		µs
R/C Oscillator Frequency Variation	$4.5V \le V_{CC} \le 5.5V$			±35	%
(Note 6)	$2.7V \le V_{CC} < 4.5V$			TBD	%
External CKI Clock Duty Cycle (Note 6)	fr = Max	45		55	%
Rise Time (Note 6)	fr = 10 MHz Ext Clock			12	ns
Fall Time (Note 6)	fr = 10 MHz Ext Clock			8	ns
Inputs					
t_{SETUP}	$4.5V \le V_{CC} \le 5.5V$	200			ns
	$2.7V \le V_{CC} < 4.5V$	500			ns
t_{HOLD}	$4.5V \le V_{CC} \le 5.5V$	60			ns
	$2.7V \le V_{CC} < 4.5V$	150			ns
Output Propagation Delay (Note 5)	R_L = 2.2k, C_L = 100 pF				
t_{PD1}, t_{PD0}					
SO, SK	$4.5V \le V_{CC} \le 5.5V$			0.7	µs
	$2.7V \le V_{CC} < 4.5V$			1.75	µs
All Others	$4.5V \le V_{CC} \le 5.5V$			1.0	µs
	$2.7V \le V_{CC} < 4.5V$			2.5	µs
MICROWIRE Setup Time (t_{UWS}) (Note 5)		20			ns
MICROWIRE Hold Time (t_{UWH}) (Note 5)		56			ns
MICROWIRE Output Propagation Delay (t_{UPD})				220	ns
MICROWIRE Maximum Shift Clock					
Master Mode				500	kHz
Slave Mode				1	MHz
Input Pulse Width (Note 6)					
Interrupt Input High Time		1			t_c
Interrupt Input Low Time		1			t_c
Timer 1,2, 3 Input High Time		1			t_c
Timer 1,2, 3 Input Low Time		1			t_c
Reset Pulse Width		1			µs

t_C = Instruction cycle time (Clock Input frequency divided by 10)

Note 1: Maximum rate of voltage change must be < 0.5 V/ms.

Note 2: Supply and IDLE currents are measured with CKI driven with a square wave Oscillator, CKO driven 180˚ out of phase with CKI, inputs connected to V_{CC} and outputs driven low but not connected to a load.

Note 3: The HALT mode will stop CKI from oscillating in the R/C and the Crystal configurations. In the R/C configuration, CKI is forced high internally. In the crystal or external configuration, CKI is TRI-STATE. Measurement of I_{DD} HALT is done with device neither sourcing nor sinking current; with L, F, C, G0, and G2-G5 programmed as low outputs and not driving a load; all outputs programmed low and not driving a load; all inputs tied to V_{CC}; clock monitor disabled. Parameter refers to HALT mode entered via setting bit 7 of the G Port data register.

Note 4: Pins G6 and $\overline{\text{RESET}}$ are designed with a high voltage input network. These pins allow input voltages > V_{CC} and the pins will have sink current to V_{CC} when biased at voltages > V_{CC} (the pins do not have source current when biased at a voltage below V_{CC}). The effective resistance to V_{CC} is 750Ω (typical). These two pins will not latch up. The voltage at the pins must be limited to < 14 Volts.
WARNING: Voltages in excess of 14 volts will cause damage to the pins. This warning excludes ESD transients.

Note 5: The output propagation delay is referenced to the end of the instruction cycle where the output change occurs.

Note 6: Parameter characterized but not tested.

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	7V
Voltage at Any Pin	–0.6V to V_{CC} +0.6V
ESD Protection Level	2KV (Human Body Model)
Total Current into V_{CC} Pin (Source)	80 mA
Total Current out of GND Pin (Sink)	100 mA
Storage Temperature Range	–65˚C to +140˚C

NOTE: *Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

2.20.5 DC Electrical Characteristics (–40˚C ≤ T_A ≤ +125˚C unless otherwise specified)

Parameter	Conditions	Min	Typ	Max	Units
Operating Voltage		4.5		5.5	V
Power Supply Rise Time (On-Chip Power-on Reset Selected)		10 ns		50 ms	
Power Supply Ripple (Note 1)	Peak-to-Peak			0.1 V_{CC}	V
Supply Current (Note 2)					
CKI = 10 MHz	V_{CC} = 5.5V, t_c = 1 µs			6.0	mA
HALT Current WATCHDOG Disable (Note 3)	V_{CC} = 5.5V, CKI = 0 MHz		<10	30	µA
IDLE Current (Note 2)					
CKI = 10 MHz	V_{CC} = 5.5V, t_c = 1 µs			1.5	mA
Input Levels (V_{IH}, V_{IL})					
RESET					
Logic High		0.8 V_{CC}			V
Logic Low				0.2 V_{CC}	V
CKI, All Other Inputs					
Logic High		0.7 V_{CC}			V
Logic Low				0.2 V_{CC}	V
Value of the Internal Bias Resistor for the Crystal/Resonator Oscillator		0.5	1.0	2.0	MΩ
CKI Resistance to V_{CC} or GND when R/C oscillator is selected	V_{CC} = 5.5V	5	8	11	kΩ
Hi-Z Input Leakage (Same as TRI-STATE output)	V_{CC} = 5.5V	–5		+5	µA
Input Pullup Current	V_{CC} = 5.5V, V_{IN} = 0V	–35		–400	µA
G and L Port Input Hysteresis		0.25 V_{CC}			V
Output Current Levels					
D Outputs					
Source	V_{CC} = 4.5V, V_{OH} = 3.3V	–0.4			mA
Sink	V_{CC} = 4.5V, V_{OL} = 1.0V	9			mA
L Port					
Source (Weak Pull-Up)	V_{CC} = 4.5V, V_{OH} = 2.7V	–9.0		–140	µA
Source (Push-Pull Mode)	V_{CC} = 4.5V, V_{OH} = 3.3V	–0.4			mA
Sink (L0-L3, Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 1.0V	9.0			mA
Sink (L4-L7, Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 0.4V	1.4			mA
All Others					
Source (Weak Pull-Up Mode)	V_{CC} = 4.5V, V_{OH} = 2.7V	–9.0		–140	µA
Source (Push-Pull Mode)	V_{CC}= 4.5V, V_{OH} = 3.3V	–0.4			mA
Sink (Push-Pull Mode)	V_{CC} = 4.5V, V_{OL} = 0.4V	1.4			mA

Parameter	Conditions	Min	Typ	Max	Units
Allowable Sink Current per Pin (Note 6)					
D Outputs and L0 to L3				15	mA
All others				3	mA
Maximum Input Current without Latchup (Note 4)				±200	mA
RAM Retention Voltage, Vr		2.0			V
V_{CC} rise time from a $V_{CC} \geq 2.0V$		1.2			µs
Input Capacitance	(Note 6)			7	pF
Load Capacitance on D2	(Note 6)			1000	pF

2.20.6 AC Electrical Characteristics (–40˚C ≤ T_A ≤ +125˚C unless otherwise specified)

Parameter	Conditions	Min	Typ	Max	Units
Instruction Cycle Time (t_c)					
Crystal/Resonator, External	4.5V ≤ V_{CC} ≤ 5.5V	1.0		DC	µs
Internal R/C Oscillator	4.5V ≤ V_{CC} ≤ 5.5V		2.0		µs
R/C Oscillator Frequency Variation (Note 6)	4.5V ≤ V_{CC} ≤ 5.5V			TBD	%
External CKI Clock Duty Cycle (Note 6)	fr = Max	45		55	%
Rise Time (Note 6)	fr = 10 MHz Ext Clock			12	ns
Fall Time (Note 6)	fr = 10 MHz Ext Clock			8	ns
Inputs					
t_{SETUP}	4.5V ≤ V_{CC} ≤ 5.5V	200			ns
t_{HOLD}	4.5V ≤ V_{CC} ≤ 5.5V	60			ns
Output Propagation Delay (Note 5)	R_L = 2.2k, C_L = 100 pF				
t_{PD1}, t_{PD0}					
SO, SK	4.5V ≤ V_{CC} ≤ 5.5V			0.7	µs
All Others	4.5V ≤ V_{CC} ≤ 5.5V			1.0	µs
MICROWIRE Setup Time (t_{UWS}) (Note 5)		20			ns
MICROWIRE Hold Time (t_{UWH}) (Note 5)		56			ns
MICROWIRE Output Propagation Delay (t_{UPD})				220	ns
MICROWIRE Maximum Shift Clock					
Master Mode				500	kHz
Slave Mode				1	MHz
Input Pulse Width (Note 6)					
Interrupt Input High Time		1			t_c
Interrupt Input Low Time		1			t_c
Timer 1, 2, 3 Input High Time		1			t_c
Timer 1, 2, 3 Input Low Time		1			t_c
Reset Pulse Width		1			µs

t_C = Instruction cycle time (Clock Input frequency divided by 10)
Note 1: Maximum rate of voltage change must be < 0.5 V/ms.
Note 2: Supply and IDLE currents are measured with CKI driven with a square wave Oscillator, CKO driven 180˚ out of phase with CKI, inputs connected to V_{CC} and outputs driven low but not connected to a load.
Note 3: The HALT mode will stop CKI from oscillating in the R/C and the Crystal configurations. In the R/C configuration, CKI is forced high internally. In the crystal or external configuration, CKI is TRI-STATE. Measurement of I_{DD} HALT is done with device neither sourcing nor sinking current; with L, F, C, G0, and G2-G5 programmed as low outputs and not driving a load; all outputs programmed low and not driving a load; all inputs tied to V_{CC}; clock monitor disabled. Parameter refers to HALT mode entered via setting bit 7 of the G Port data register.
Note 4: Pins G6 and $\overline{\text{RESET}}$ are designed with a high voltage input network. These pins allow input voltages > V_{CC} and the pins will have sink current to V_{CC} when biased at voltages > V_{CC} (the pins do not have source current when biased at a voltage below V_{CC}). The effective resistance to V_{CC} is 750Ω (typical). These two pins will not latch up. The voltage at the pins must be limited to < 14 Volts. WARNING: Voltages in excess of 14 volts will cause damage to the pins. This warning excludes ESD transients.
Note 5: The output propagation delay is referenced to the end of the instruction cycle where the output change occurs.
Note 6: Parameter characterized but not tested.

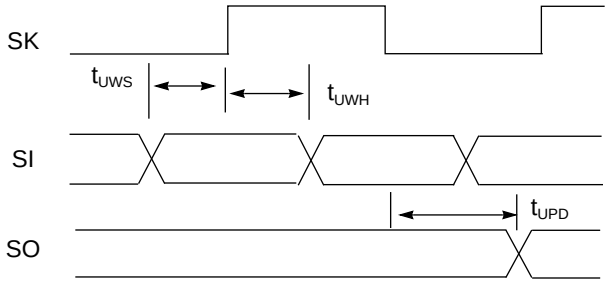


Figure 2-30 MICROWIRE/PLUS Timing

2.21 ESD/EMI CONSIDERATIONS

National's patent ESD protection and EMI reduction circuits are implemented on device to address ESD/EMI problems.

2.22 INPUT PROTECTION

The COP8SAx7 input pins have internal circuitry for protection from ESD. The internal circuitry is shown in Figure 2-31.

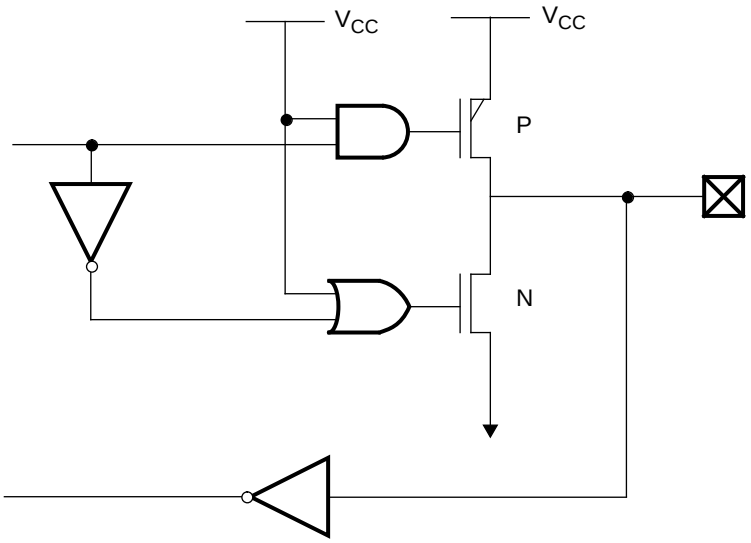


Figure 2-31 Ports L/C/G/F Input Protection (Except G6)

The input protection circuitry is implemented with the P_channel transistors. The equivalent diode circuit is shown in Figure 2-32.

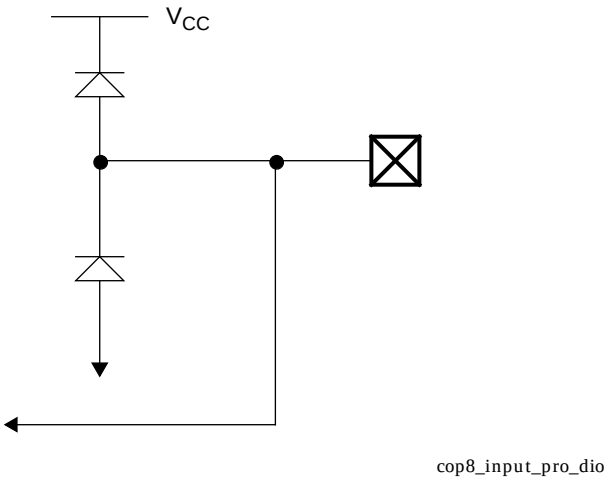


Figure 2-32 Diode Equivalent of Input Protection

National's patented Merrill Clamp ESD protection circuitry is implemented on device to direct the current resulting from an ESD pulse on an input pin, onto the V_{CC} and GND bus. Figure 2-33 shows the on-chip detection/protection circuit. The circuitry is designed to meet ESD protection goal of at least 2000 volts as measured using Human Body Model.

There are two Merrill Clamp blocks on each die. The Merrill Clamp block is used to help protect the chip from ESD events. The Merrill Clamp senses a fast rise-time on the V_{CC} supply and shorts the V_{CC} line to the GND with a transistor. A Merrill Resistor block is used to drain off any charge on V_{CC} between ESD events. The Merrill Resistor block contains only a 4 MΩ resistor which is connected between the V_{CC} supply line and the GND return line.

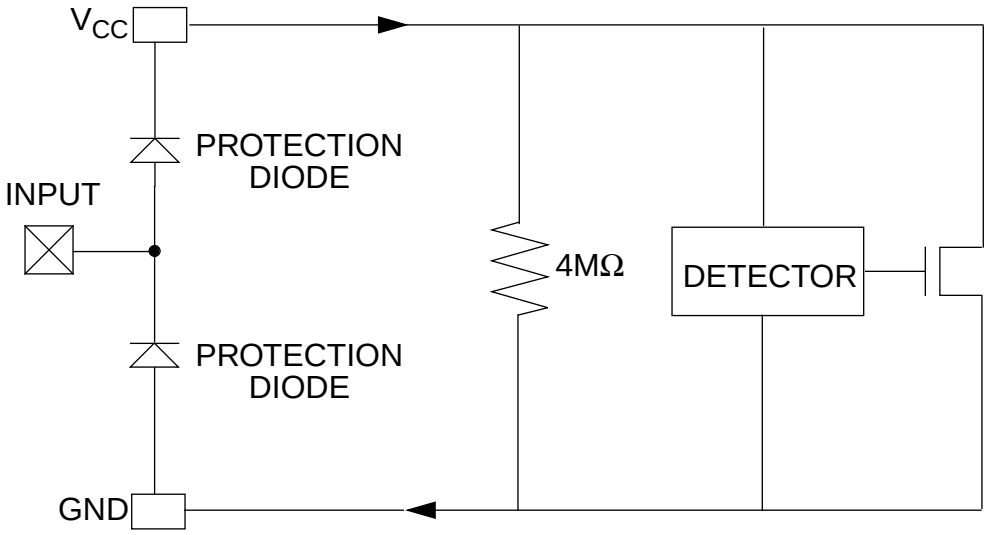


Figure 2-33 On-Chip ESD Detection/Protection Circuit

2.23 ELECTROMAGNETIC INTERFERENCE (EMI) CONSIDERATIONS

2.23.1 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).

2.23.2 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]

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 2-12. 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.

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,

1. "FACT™ Advanced CMOS Logic Databook", National Semiconductor, 1993

Table 2-12 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

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.

2.23.3 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.

2.23.4 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.

2.23.5 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.

2.23.6 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.

2.23.7 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.

2.24 EMI REDUCTION ON THE COP8SAx7

The COP8SAx7 devices products incorporate circuity that guards against electromagnetic interference — an increasing problem in today's microcontroller board designs. National's patented EMI reduction technology offers low EMI emissive clock circuitry, EMI-optimized pinouts, gradual turn-on output drivers (GTOs) and an on-chip choke device to help circumvent many of the EMI issues influencing embedded control designs. National has achieved 15-20 dB reduction in EMI transmissions when designs have incorporated its patented EMI reducing circuitry (Figure 2-34).

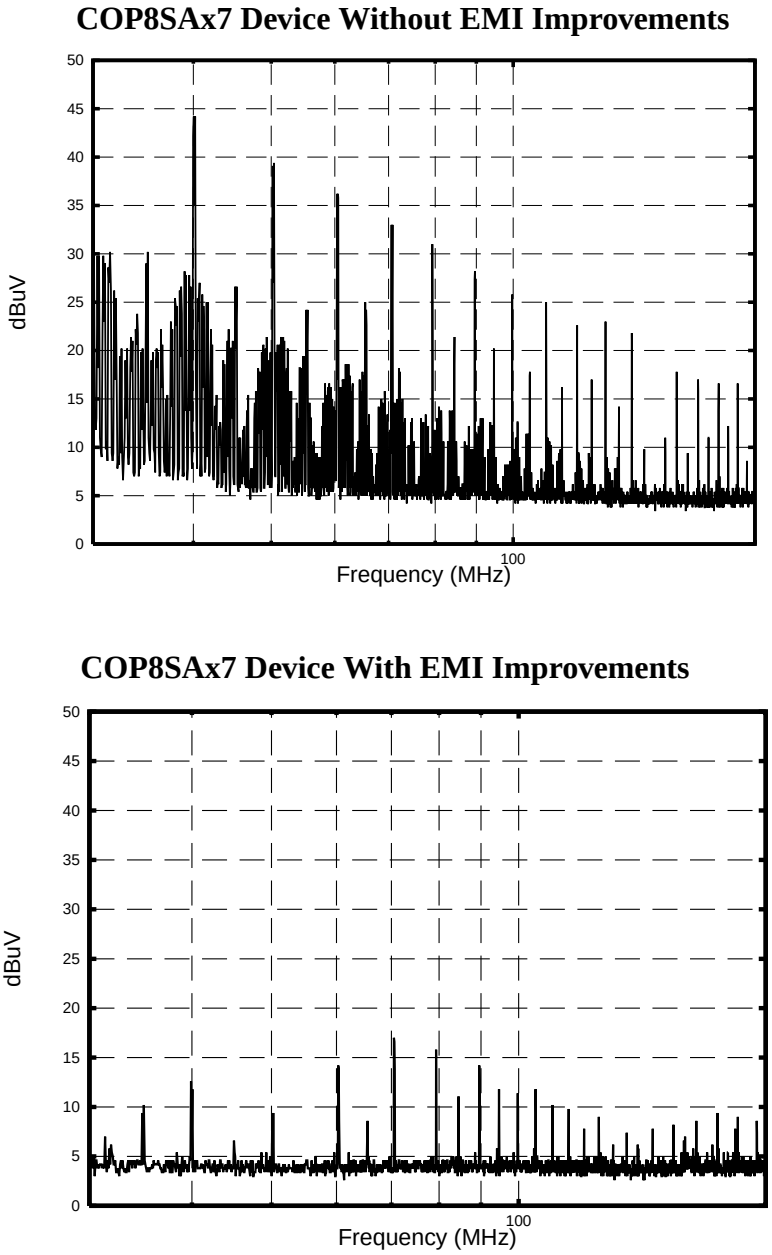


Figure 2-34 EMI Improvements

2.24.1 Silicon Design Changes to Achieve Low EMI

National's design goal was to reshape the IC pin current waveforms from their original values. A second goal was to provide separate power and ground bus systems on-chip for each of the following groups: Chip digital logic, Chip I/O buffers, and if present a chip A/D converter (or other analog intensive sections).

Inspection of these goals demanded the need to divide the chip into separate areas: the nucleus, the perimeter and the analog. Also the partitioning of the nucleus (containing the digital logic functions) and the perimeter (containing digital I/O and oscillator functions) allowed the use of an on-chip device to choke the nucleus' current, thereby wave shaping the current. The current choke, however, causes the nucleus V_{CC} voltage to dip by 0.5 to 1 Volt at each clock edge. These V_{CC} voltage swings were confined to within the chip's nucleus where the circuitry (digital logic) is tolerant and no significant radiation antennae exist.

On-chip level shifting circuitry was inserted as interface between the nucleus and the perimeter. It allowed the nucleus to operate at a lower V_{CC} than the perimeter.

Another challenge was to reduce EMI from the chip's output drivers. CMOS output drivers turn on fast, in about 1nS. This causes 2 problems. The first is called 'shoot through' current which flows from DV_{CC} to DGND when both the pull up and the pull down drivers are on and the output load is small. The second is due to output current magnitude increasing to maximum in about 1 nS when the output changes state. Both problems were solved by using fast turn off, gradual turn on device drivers.

Figure 2-35 illustrates, in block diagram form, how the chip is partitioned for EMC. The power/ground pads named DV_{CC} (Driver V_{CC}), LV_{CC} (Logic V_{CC}),V_{CC} (a global V_{CC} to most of the chip), GND (a global GND to most of the chip) and DGND (driver ground) are all available). To minimize package pin count, DV_{CC} and LV_{CC} is bonded to the same pin, GND & DGND is also bonded to the same pin, and the V_{CC} pad is not bonded.

2.24.2 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.

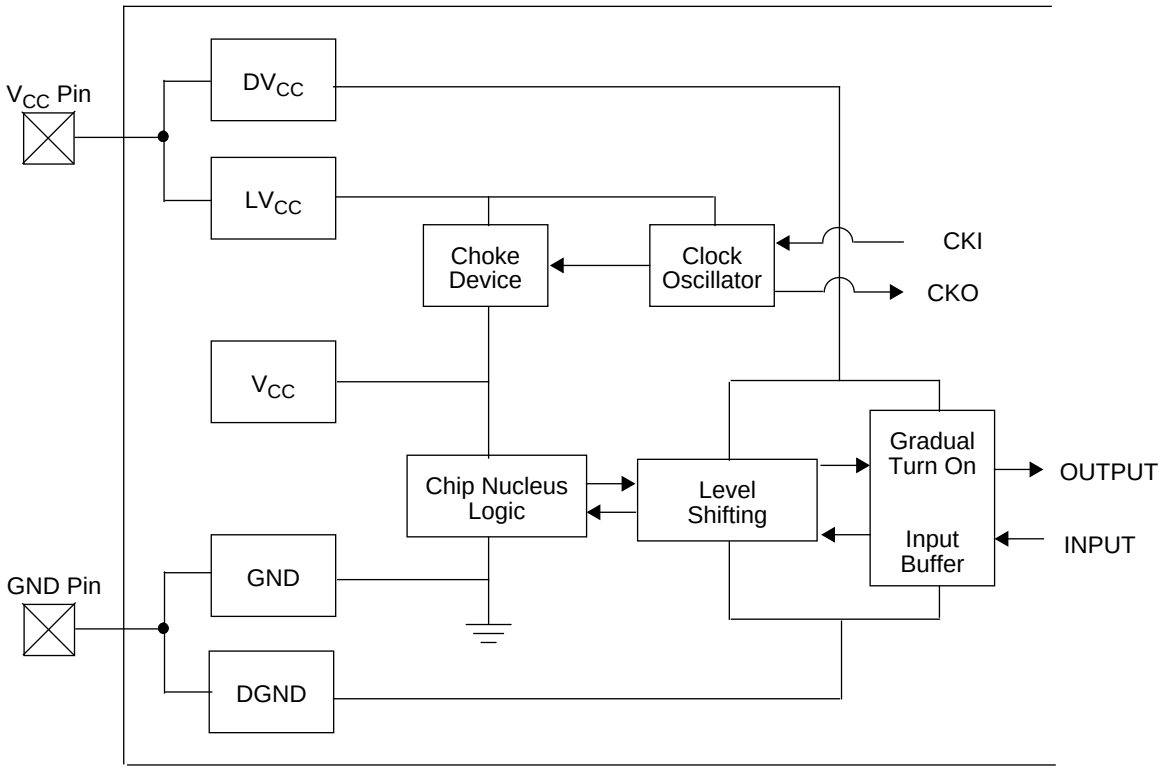


Figure 2-35 Block diagram of EMI Circuitry

Chapter 3

DEVELOPMENT SUPPORT

3.1 SUMMARY

- iceMASTER: IM-COP8/400 -- Full featured in-circuit emulation for all COP8 products. A full set of COP8 Basic and Feature Family device and package specific probes are available.
- COP8 Debug Module: Moderate-cost in-circuit emulation and development programming unit.
- COP8 Evaluation & Programming Unit: EPU-COP888GG -- low-cost in-circuit simulation and development programming unit.
- Assembler: COP8-DEV-IBMA. A DOS installable cross development Assembler, Linker, Librarian and Utility Software Development Tool Kit.
- C Compiler: COP8C. A DOS installable cross development Software Tool Kit.
- OTP/EPROM Programmer Support: Covering needs from engineering prototype, pilot production, to full production environments.
- In-factory Programming Support: Covering high volume production OTP programmed devices.

3.2 iceMASTER (IM) IN-CIRCUIT EMULATION

The iceMASTER IM-COP8/400 is a full featured, PC based, in-circuit emulation tool developed and marketed by MetaLink Corporation to support the whole COP8 family of products. National and National authorized Distributors are resale vendors for these products.

See Figure 3-1 for the COP8 iceMASTER configuration.

The iceMASTER IM-COP8/400 with its device specific COP8 Probe provides a rich feature set for developing, testing, and maintaining the product:

- Real-time in-circuit emulation; full 2.7-5.5V operation range, full DC-10MHz clock. Chip options are programmable or jumper selectable.
- Direct connection to application board by package-compatible socket or surface mount assembly.
- Full 32K byte of loadable programming space that overlays (replaces) the on-chip ROM or EPROM. On-chip RAM and I/O blocks are used directly or re-created on the probe as necessary.

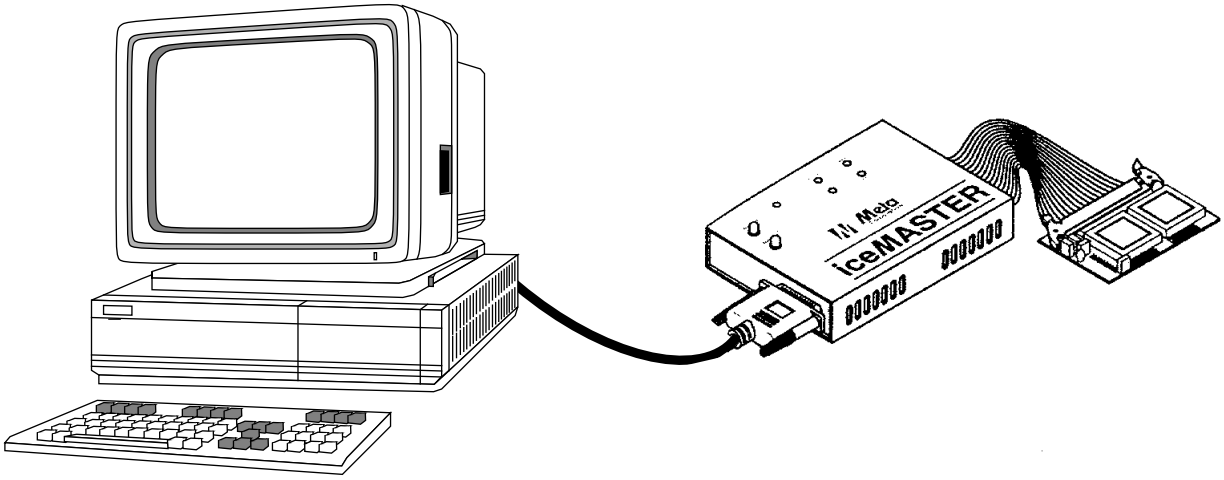


Figure 3-1 COP8 iceMASTER Environment

- Full 4k frame synchronous trace memory. Address, instruction, and eight unspecified, circuit connectable trace lines. Display can be high-level language source (e.g., C source), assembly, or mixed.
- A full 64k hardware configurable break, trace on, trace off, and pass count increment events.
- Tool set integrated interactive symbolic debugger — supports both assembler (COFF) and C Compiler (.COD) linked object formats.
- Real time performance profiling analysis; selectable bucket definition.
- Watch windows, content updated automatically at each execution break.
- Instruction-by-instruction memory/register changes displayed in source window when in single step operation.
- Single base unit and debugger software reconfigurable to support the entire COP8 family; only the probe personality needs to change. Debugger software is processor customized, and reconfigured from a master model file.
- Processor-specific symbolic display of registers and bit level assignments, configured from master model file.
- Halt/Idle mode notification.
- On-line HELP.
- Includes a copy of COP8-DEV-IBMA assembler and linker SDK.

IM Order-Information

Base Unit	
IM-COP8/400-1	iceMASTER base unit, 110V Power Supply
IM-COP8/400-2	iceMASTER base unit, 220V Power Supply
iceMASTER Probe, COPSAx7	
COP8SA-IM44V	44 PLCC, 2.7 – 5.5V
COP8SA-IM40N	40 DIP, 2.7 – 5.5V
COP8SA-IM28N	28 DIP, 2.7 – 5.5V
COP8SA-IM20N	20 DIP, 2.7 – 5.5V
COP8SA-IM16N	16 DIP, 2.7 – 5.5V
Optional Surface Mount Adapter Kits	
MHW-COP8/44P-Q	44 PLCC to 44 PQFP SM Adapter
MHW-SOIC-28	28 DIP to 28 SOIC SM Adapter
MHW-SOIC-20	20 DIP to 20 SOIC SM Adapter
MHW-SOIC-16	16 DIP to 16 SOIC SM Adapter
e.g., Target package is 44P; order: IM-COP8/400-1, COP8SA-IM44V and MHW-COP8/44V-P.	

3.3 iceMASTER DEBUG MODULE (DM)

The iceMASTER Debug Module is a PC based, combination in-circuit emulation tool and COP8 based OTP/EPROM programming tool developed and marketed by MetaLink Corporation to support the whole COP8 family of products. National and National authorized Distributors are resale vendors for these products.

See Figure 3-2 for the iceMASTER Debug Module configuration.

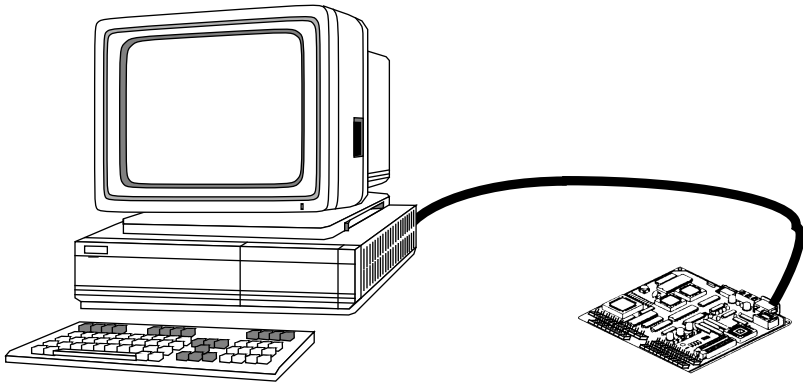


Figure 3-2 COP8-DM Environment

The iceMASTER Debug Module is a moderate-cost development tool. It has the capability of in-circuit emulation for a specific COP8 microcontroller and in addition serves as a programming tool for COP8 OTP and EPROM product families. It offers the following features:

- Real-time in-circuit emulation; full operating voltage range operation, full DC-10MHz clock.
- All processor I/O pins can be cabled to an application development board with package-compatible cable to socket and surface mount assembly.
- Full 32 kbyte of loadable programming space that overlays (replaces) the on-chip ROM or EPROM. On-chip RAM and I/O blocks are used directly or re-created as necessary.
- 100 frames of synchronous trace memory. The display can be high-level language source (C source), assembly, or mixed. The most recent history prior to a break is available in the trace memory.
- Configured break points; uses INTR instruction which is modestly intrusive.
- Software-only supported features are selectable.
- Tool set integrated interactive symbolic debugger — supports both assembler (COFF) and C Compiler (.COD) SDK linked object formats.
- Instruction-by-instruction memory/register changes displayed when in single step operation.
- Processor-specific symbolic display of registers and bit level assignments, configured from master model file.
- Halt/Idle mode notification.
- Programming menu supports full product line of programmable OTP and EPROM COP8 products. Program data is taken directly from the overlay RAM. Programming of 44 PQFP parts requires external programming adapters.
- Includes wallmount power supply
- On-board VPP generator from 5V input or connection to external supply supported. Requires VPP level adjustment per the family programming specification (correct level is provided on an on-screen pop-down display).
- Includes a copy of COP8-DEV-IBMA assembler snf linker SDK.

DM Order-Information

Debug Module Unit	
COP8SA-DM	
Cable Adapters, requires one for emulation	
DM-COP8/44P	44 PLCC
DM-COP8/40D	40 DIP
DM-COP8/28D	28 DIP
DM-COP8/20D	20 DIP
DM-COP8/16D	16 DIP
Optional Surface Mount Adapter Kits	
MHW-COP8/44P-Q	44 PLCC to 44 PQFP
DM-COP8/28D-SO	28 DIP to 28 SOIC
DM-COP8/20D-SO	20 DIP to 20 SOIC
DM-COP8/16D-SO	16 DIP to 16 SOIC
Optional Programming Adapters	
COP8-PGMA-44Q	44 PQFP

3.4 iceMASTER EVALUATION PROGRAMMING UNIT (EPU)

The iceMASTER-COP8 EPU is a PC based, in-circuit simulation tool to support the feature family COP8 products.

See Figure 3-3 for the iceMASTER COP8SA-EPU configuration.

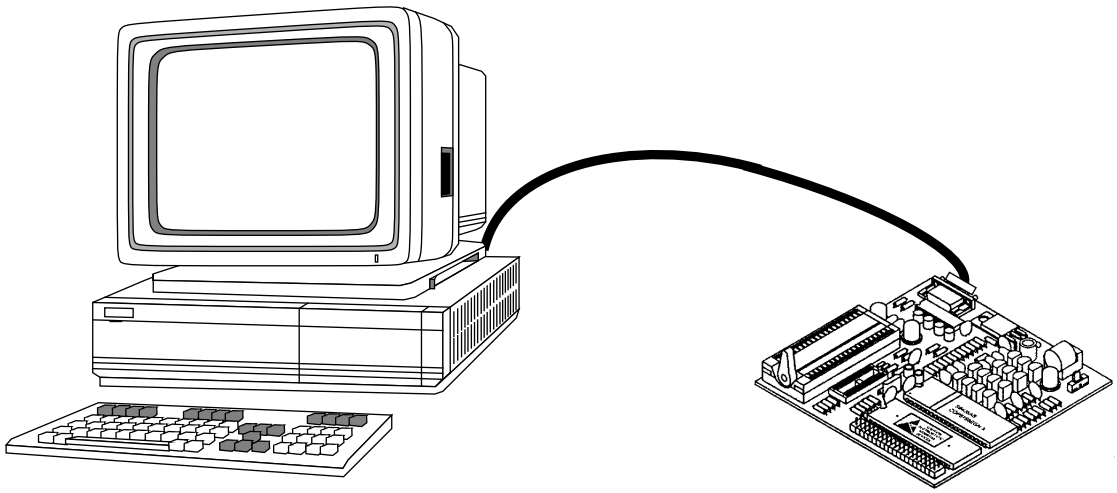


Figure 3-3 EPU-COP8 Tool Environment

The simulation capability is a very low-cost means of evaluating the general COP8 architecture. In addition, the EPU has programming capability, with added adapters, for programming the whole COP8 product family of OTP and EPROM products. The product includes the following features:

- Non-real-time in-circuit simulation. Program overlay memory is PC resident; instructions are downloaded over RS-232 as executed. Approximate performance is 20KHz.
- Includes a 40-pin DIP cable adapter. Other target packages are not supported. All processor I/O pins are cabled to the application development environment.
- Full 32 kbyte of loadable programming space that overlays (replaces) the on-chip ROM or EPROM. On-chip RAM and I/O blocks are used directly or re-created as necessary.
- On-chip timer and watchdog execution are not well synchronized to the instruction simulation.
- 100 frames of synchronous trace memory. The display can be high-level language-source (e.g., C source), assembly, or mixed. The most recent history prior to a break is available in the trace memory.
- Up to eight software configured break points; uses INTR instruction which is modestly intrusive.
- Common look-feel debugger software across all MetaLink products — only supported features are selectable.
- Tool set integrated interactive symbolic debugger — supports both assembler (COFF) and C Compiler (.COD) SDK linked object formats.
- Instruction-by-instruction memory/register changes displayed when in single step operation.
- Processor-specific symbolic display of registers and bit level assignments, configured from master model file.
- Halt/Idle mode notification. Restart requires special handling.
- Programming menu supports full product line of programmable OTP and EPROM COP8 products. Only a 40 ZIF socket is available on the EPU unit. Adapters are available for other part package configurations.
- Integral wall mount power supply provides 5V and develops the required VPP to program parts.
- Includes a copy of COP8-DEV-IBMA assembler, linker SDK.

3.4.1 Getting Started With the EPU

Installing the EPU Software

You must have at least 1700 kilobytes of free disk space on your PC to install the EPU host software. To install this software type the following from the DOS command line:

C:\>**a:\install a: c: \epu-cop8** or

C:\>**x:\install** (Where x is either A or B)

The install function will prompt for paths.

Installing the Assembler Software

Place the COP8-DEV-IBMA disk into a floppy drive. Enter:

C:\>**x:\install** (Where x is either A or B)

Install will prompt for the drives letter of the source drive. Enter if other than a:. It will then prompt for the directory to install the files. If other than c:\cop enter the correct path. After confirming there is room for the files, INSTALL copies them to the installation directory.

Pay particular attention to the notes on editing your CONFIG.SYS and AUTOEXEC.BAT files. You must properl.y set up your PATH and environment variables. If you need to redisplay the notes, enter TYPE ASMREAD.ME

Installing the Hardware

Attach the serial cable provided to either COM1 or COM2 of the PC and connect the other end to the EPU. Plug in the AC adapter provided into the AC outlet. Connect the output of the adapter to the DC jack provided on the EPU. In order to observe the I/Os the target cable may be plugged into the header on the EPU. *CAUTION:* Failure to remove the conductive foam from the target cable may result in improper operation of the unit.

Assembling and Running a Program on the EPU

Assume that the EPU and Assembler software have been installed in path: c:\epu-cop8\. Then use the following to assemble a program.

To assemble a program. The various flags are explained in the COP8 Assembler/Linker/ Librarian Users Manual.

C:\EPU-COP8>**asmcop /1 /sy sample /PW=132**

To link the program

C:\EPU-COP8>**lncop sample /fo=h**

To load it into the EPU

C:\EPU-COP8>**epu888gg sample**

Programming a Sample Device

Ensure that the jumper block is NOT installed on the EPU board at J4 - this conditions the programming socket pins for the COP8SAx algorithm. The sample devices shipped with the EPU are 40-pin DIP packages; the EPU will program all COP8SAx DIP packages, directly. Other packages require a programming adaptor. To enable the programming algorithm, start the EPU software and select the MISC menu box. The pop-down menu can then be used to start the programming functions. The memory used by the EPU simulation function is the same as the memory used to contain the code for programming - its easy to develop code, debug and prove correctness, then program a sample part for hardware testing.

EPU Order-Information

Evaluation Programming Unit	
COP8SA-EPU	Evaluation Programming Unit with debugger and programmer control software with 40 pin ZIF programming socket.
Optional Programming Adapters	
COP8-PGMA-44P-Q	44 PLCC & 44 PQFP
COP8-PGMA-28 SO	28, 20 and 16 SOIC
COP8-PGMA-878x	COP8782, COP8781, COP8780

3.5 COP8 ASSEMBLER/LINKER SOFTWARE DEVELOPMENT TOOL KIT

National Semiconductor offers a relocatable COP8 macro cross assembler, linker, librarian, and utility software development tool kit. Features are summarized as follows:

- Basic and Feature Family instruction set by "device" type.
- Nested macro capability.
- Extensive set of assembler directives.
- Supported on PC/DOS platform.
- Generates National standard COFF output files.
- Integrated Linker and Librarian.
- Integrated utilities to generate ROM code file outputs.
- DUMPCOFF utility.

This product is integrated as a part of MetaLink tools as a development kit, fully supported by the MetaLink debugger. It may be ordered separately or bundled with the MetaLink products at no additional cost.

Assembler/Linker Order-Information

Assembler SDK:	
COP8-DEV-IBMA	Assembler SDK on installable 3.5" PC®/DOS floppy disk drive format. Periodic upgrades and most recent version is available on National's BBS and the Internet.

3.6 COP8 C COMPILER

A C Compiler is developed and marketed by Byte Craft Limited. The COP8C compiler is a fully integrated development tool specifically designed to support the compact embedded configuration of the COP8 family of products.

Features are summarized as follows:

- ANSI C with some restrictions and extensions that optimize development for the COP8 embedded application.
- BITS data type extension. Register declaration #pragma with direct bit level definitions.
- C language support for interrupt routines.
- Expert system, rule-based code generation and optimization.
- Performs consistency checks against the architectural definitions of the target COP8 device.
- Generates program memory code.
- Supports linking of compiled object or COP8 assembled object formats.
- Global optimization of linked code.
- Symbolic debug load format fully source-level supported by the MetaLink debugger.

3.7 INDUSTRY WIDE OTP / EPROM PROGRAMMING SUPPORT

Programming support, in addition to the MetaLink development tools, is provided by a full range of independent approved vendors to meet the needs from the engineering laboratory to full production.

Approved List:

Manufacturer	North/South America	Europe	Asia/Japan
BP Microsystems (USA) www.bpmicro.com sales@bpmicro.com	US: Houston, TX 800-225-2102 713-688-4600 fax: 713-688-0920 bbx: 713-688-9283	GER: Penzberg (HT Eurep): 88-56-932616 fax: 88-56-932624 UK: North Ants (Direct Insight): 1280-700-262 fax: 1280-700-577	HK: (Tektron): 2388-0629 fax: 2780-5871 TAI: Taipei (Jeritronics): 2-585-1636 fax: 2-586-4736
Data I/O (USA) www.data-io.com sales@data-io.com techhelp@data-io.com	US: Redmond, WA 800-426-1045 206-881-6444 fax: 206-882-1043 bbs: 206-882-3211 CAN: 905-678-0761 fax: 905-678-7306	UK: Wokingham, Berks (0) 734-440011 fax: (0) 734-448700 GER: Graefelfing fax: (0) 89-8585810	JAP: Tokyo 3-3779-2161 fax: 3-3779-2203 ASIA: Rest of World Contact USA: 206-867-6919 fax: 206-882-1043
HI - LO (Taiwan) www.hilosystems.com.tw	US: Fremont, CA 510-623-8860 bbs: 510-623-0430	SWZ: Schwerzenbach 1-825-5777 fax: 1-825-5661 SWD: Frolunda, 9 31-49-250 fax: 31-491215 UK: Arkley, Barnet, Herts. 181-441-3890 fax: 181-441-1843 GER: Detmold 5232-8131 fax: 5232-86197 FRA: Epinay sur seine 1-48418036 fax: 1-48410223 ITA: Bologna 51-532119 fax: 51-601-0076	TAI: Taipei 2-764-0215 fax: 2-756-6403 bbs: 2-769-0881 hilosys@fit.ivnet.com.tw
ICE Technology (UK) www.icetech.com	US: Henderson, NC 800-624-8949 919-693-0679 fax: 919-693-0681	UK: Penistone, S.York (0) 1226-767404 fax: (0) 1226-370434 bbs: (0) 1226-761181 sales@icetech.com GER: Magnadata: 60-82-742-1515; 60-82-3448	JAP: Tokyo 3-5386-5501 fax: 3-5386-5503 HK: 2891-3673 fax: 2834-9748 SG: 483-1691 fax: 483-1692
MetaLink (USA)	US: Chandler, AZ 800-638-2423 602-926-0797 fax: 602-926-1198	GER: (Kirchseeon) 80-91-5696-0 fax: 80-91-2386	HK: (National) 2737-1600 fax: 2736-9960
Needhams (USA)	US: Sacramento, CA 916-924-8037 fax: 916-924-8065	Contact USA	Contact USA

Manufacturer	North/South America	Europe	Asia/Japan
Systems General (Taiwan)	US: Milpitas, CA 800-967-4776 408-263-6667 fax: 408-262-9220 bbs: 408-262-6438	FRA: (Microprossy): 20-151133 fax: 20-151166	TAI: Taipei 2-918-3005 fax: 2-911-1283 bbs: 2-918-1076 JAP: Tokyo 3-3441-1510 fax: 3-3441-7185
Xeltek (USA) www.xeltek.com info@xeltek.com	US: Sunnyvale, CA 408-524-1929 fax: 408-245-7084 bbs: 408-245-7082	GER: 20-41-684758	SING: (Test & Measurement) 280-4611 fax: 280-4677

3.8 AVAILABLE LITERATURE

For more information, please see the COP8 Basic Family User's Manual, Literature Number 620895, COP8 Feature Family User's Manual, Literature Number 620897 and National's Family of 8-bit Microcontrollers COP8 Selection Guide, Literature Number 630006.

3.9 DIAL-A-HELPER SERVICE

Dial-A-Helper is a service provided by the Microcontroller Applications group. The Dial-A-Helper is an Electronic Information System that may be accessed as a Bulletin Board System (BBS) via data modem, as an FTP site on the Internet via standard FTP client application, or as an FTP site on the Internet using a standard Internet browser such as Netscape or Mosaic.

The Dial-A-Helper system provides access to an automated information storage and retrieval system. The system capabilities include a MESSAGE SECTION (electronic mail, when accessed as a BBS) for communications to and from the Microcontroller Applications Group and a FILE SECTION which consists of several file areas where valuable application software and utilities can be found.

3.10 DIAL-A-HELPER BBS VIA A STANDARD MODEM

Modem:	CANADA/U.S.:	(800) NSC-MICRO (800) 672-6427	
	EUROPE:	(+49) 0-814-135 13 32	
	Baud:	14.4k	
	Set-Up:	Length:	8-Bit
		Parity:	None
		Stop Bit:	1
	Operation:	24 Hours, 7 Days	

Dial-A-Helper via FTP

ftp nscmicro.nsc.com

user: anonymous

password: username@yourhost.site.domain

Dial-A-Helper via a WorldWide Web Browser

ftp://nscmicro.nsc.com

3.11 NATIONAL SEMICONDUCTOR ON THE WORLDWIDE WEB

See us on the WorldWide Web at: http://www.national.com

3.12 CUSTOMER RESPONSE CENTER

Complete product information and technical support is available from National's customer response centers. Please see back cover for telephone number in your area.

Chapter 4

COP8SAx7 APPLICATION IDEAS

This chapter describes several application examples using the COP8SAx7 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:

4.1 TESTING A REMOTE NORMALLY OPEN SWITCH FOR CONNECTION

Some applications using remote normally open switches for sensors call for knowing whether the sensor is connected. Usually one cannot tell the difference between a normally open switch and a disconnected one. However, a 10k resistor is connected across the switch, one can measure the resistance to determine whether the switch is connected. Usually this requires a linear device such as an additional transistor.

If the switch is connected, a programmable I/O port on a microcontroller that can be placed in the TRI-STATE mode can detect the presence of the 10k resistor without additional parts. The following shows an example of this technique using a COP8SAx7.

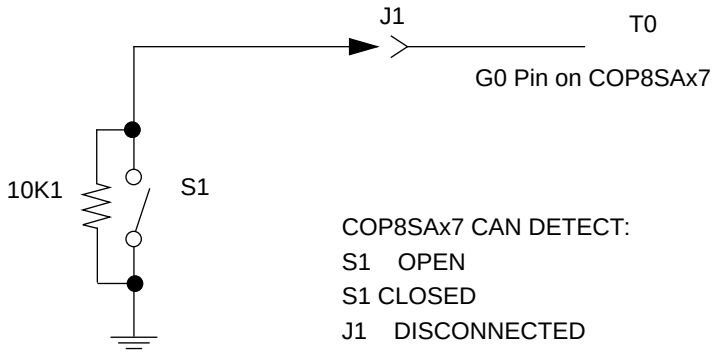


Figure 4-1 Test Circuit

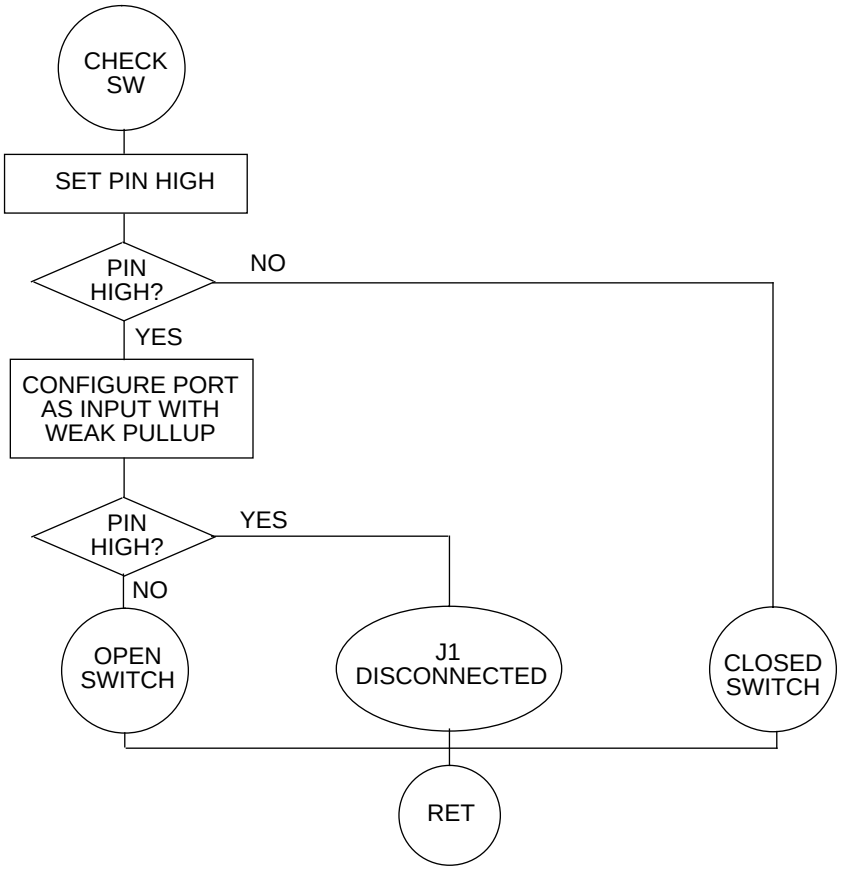


Figure 4-2 Flow Chart

```
GDATA   = 0D4
GCONF   = 0D5
GPINS   = 0D6
DPORT   = 0DC

CHECK:  SBIT    0, GCONF        ; Set G0 High
        SBIT    0, GDATA
        IFBIT   0, GPINS        ; Test G0. High?
        JP      $1              ; Yes
SCLS:   NOP                     ; No, close switch
        SBIT    0, DPORT
        RBIT    1, DPORT
        JP      CHECK
$1      RBIT    0, GCONF        ; Configure G0 as input with weak pullup
        IFBIT   0, GPINS        ; Test Go. High?
        JP      OPEN            ; Yes J1 disconnected
NORM:   NOP                     ; No, open switch
        SBIT    0, DPORT
        RBIT    1, DPORT
        JP      CHECK
OPEN:   NOP
        RBIT    0, DPORT
        SBIT    1, DPORT
        JP      CHECK
```

4.2 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 COP888 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.

4.2.1 MICROWIRE/PLUS Master/Slave Protocol

This section gives an example of a MICROWIRE/PLUS master/slave protocol, the slave mode operating procedure for the example protocol, and a timing illustration of the example 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 4-3).

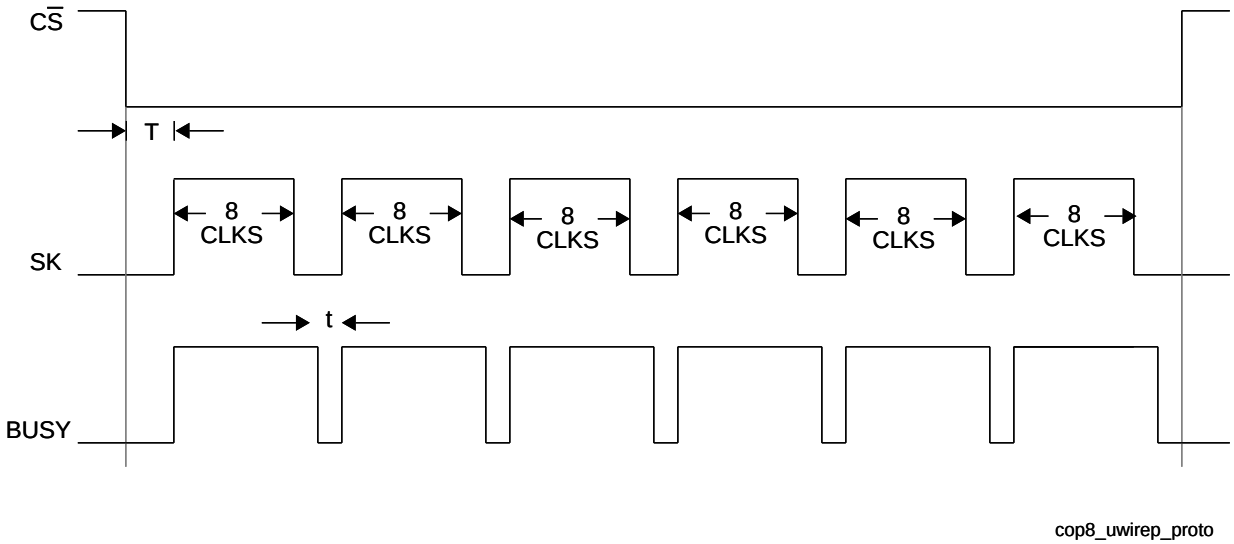


Figure 4-3 MICROWIRE/PLUS Sample Protocol Timing

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.
7. Repeat steps 3 through 6. The user must ensure step 3 is performed within *t-time* (refer to Figure 4-3) as agreed upon in the protocol.

4.2.2 NM93C06-COP8SAx7 Interface

This example shows the COP8SAx7 interface to a NM93C06, a 256-bit E^2PROM, using the MICROWIRE/PLUS interface. The pin connections for the interface is shown in Figure 4-4. Some notes on the NM93C06 interface requirements are:

1. The SK clock frequency should be less than 250 KHz. The SK clock should be configured for standard SK mode.

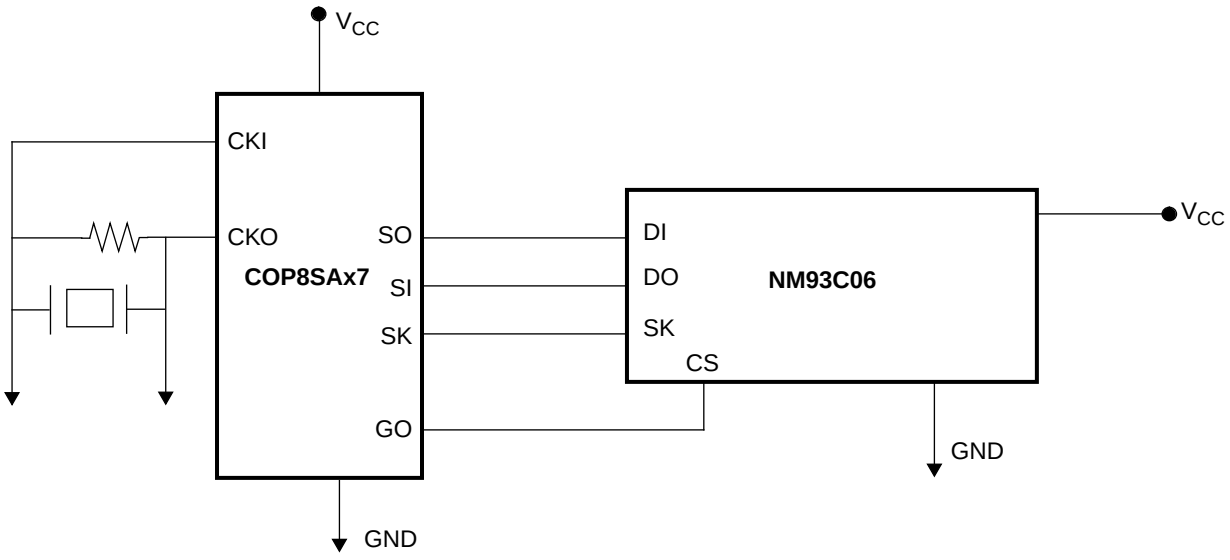


Figure 4-4 NM93C06-COP8SAx7 Interface

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.

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 have 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 NM93C06. 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 data sheet. A source listing of the software used to interface the NM93C06 with the COP8SAx7 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 NM93C06
        RDATL  = NMCMEM+1   ;Lower byte of NM93C06 register data read
        RDATH  = NMCMEM+2   ;Upper byte of NM93C06 register data read
        WDATL  = NMCMEM+3   ;Lower byte of data to be written to NM93C06 register
        WDATH  = NMCMEM+4   ;Upper byte of data to be written to NM93C06 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 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 NM93C06 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 NM93C06. 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 NM93C06
;
EWDS:   LD      SNDBUF,#0
        LD      FLAGS,#0
        JSR     INIT
        RET
```

```
;This routine erases all registers of the NM93C06
;
ERAL:    LD       SNDBUF,#020
         LD       FLAGS,#0
         JSR      INIT
         RET

;This routine reads the contents of a NM93C06 register. The NM93C06 address is
;specified in the lower nibble of location “ADRESS.” The upper nibble should be set
;to zero. The 16-bit contents of the NM93C06 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 NM93C06 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 NM93C06
         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
```

4.3 TIMER APPLICATIONS

This section describes several 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.

4.4 TIMER PWM APPLICATIONS

The 16-bit timer can be used in any of the following applications:

4.4.1 Rudimentary D-A Converter

The first example illustrates how a rudimentary D-A converter may be built by using the timer. By controlling the values in the register the user can control the frequency and duty cycle of the resultant waveform. The voltage at the capacitor is directly proportional to the duty cycle. Increasing the T_{ON} value while holding the frequency constant serves to increase the analog voltage, whereas increasing the T_{OFF} value while holding the frequency constant serves to decrease the analog voltage.

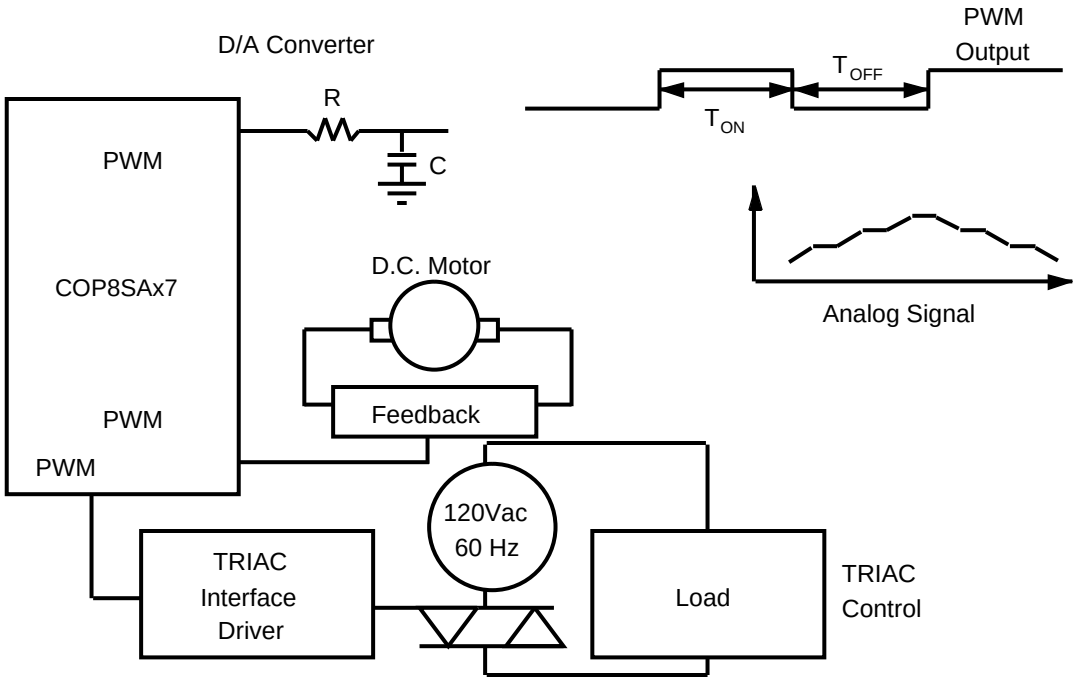


Figure 4-5 Timer PWM Applications

4.4.2 PWM Motor Control

In the second example, the PWM output provides servo control for the DC motor. By varying the ON and OFF times (varying duty cycle) and varying the cycle time, the DC motor speed is controlled.

The duty cycle of the PWM output controls speed of a DC motor. A high output (100% Duty Cycle) causes the motor to run at maximum speed, while a low output (0% Duty Cycle) turns the motor off.

PWM Motor Control Code

```
             UPTML0       =   020
             UPTMHI       =   021
             DNTMLO       =   022
             DNTMHI       =   023

             T1L          =   0C0
             T1H          =   0C1
             T1RALO       =   0C2
             T1RAHI       =   0C3
             T1RBLO       =   0C4
             T1RBHI       =   0C5
             T1CNTRL      =   0C6

             PORTLD       =   0D0
             PORTLC       =   0D1

INITIALIZATION:                          BYTES/CYCLES

T1INIT:      LD           PORTLC,#0FF          3/3
             LD           PORTLD,#0            3/3
             LD           A, DNTMLO            2/3
             X            A, T1L               2/3
             LD           A, DNTMHI            2/3
             X            A, T1H               2/3
             JSR          UPDATE               2/5
                                              16/23
                                              11/46
             TOTAL INITIALIZATION TIME    =    27/69
CHANGE:

PWMCHG:      _____        Set up times for new Duty
             _____        Cycle in UPTMLO, UPTMHI,
             _____        DNTMLO, DNTMHI
             _____
             LD           T@CNTRL, #0B5   3/3

INTERRUPT:
             . = 0FF (Interrupt Address)

INTR:        VIS              1/7
T2INTR:      JSR UPDATE       2/5
             RETI             1/5
                              4/17
Both the T1RA and T1RB Timer Interrupts should be
programmed to vector to T1TNTR with the VIS instruction.
```

```
      UPDATE SUBROUTINE:        Bytes/Cycles

UPDATE:   LD      B,#T1RALO          2/2
          LD      X,#UPTMLO          2/3
UPDLUP:   LD      A,[X+]             1/3
          X       A,[B+]             1/2
          IFBNE   #6                 1/1
          JP      UPDLUP            1/3-1
          LD      [B],#0B0           2/2
          RET                        1/5

                                    11/46

                                     4/17
                                    11/46
                                    _____
          TOTAL INTERRUPT TIME    = 15/63
          TOTAL BYTES = 16 +3 +    +15 = 34 +
                 INTIT/CHANGE/INTR
```

4.4.3 AC Motor TRIAC Control

The third example is a dimmer application or AC motor control application.

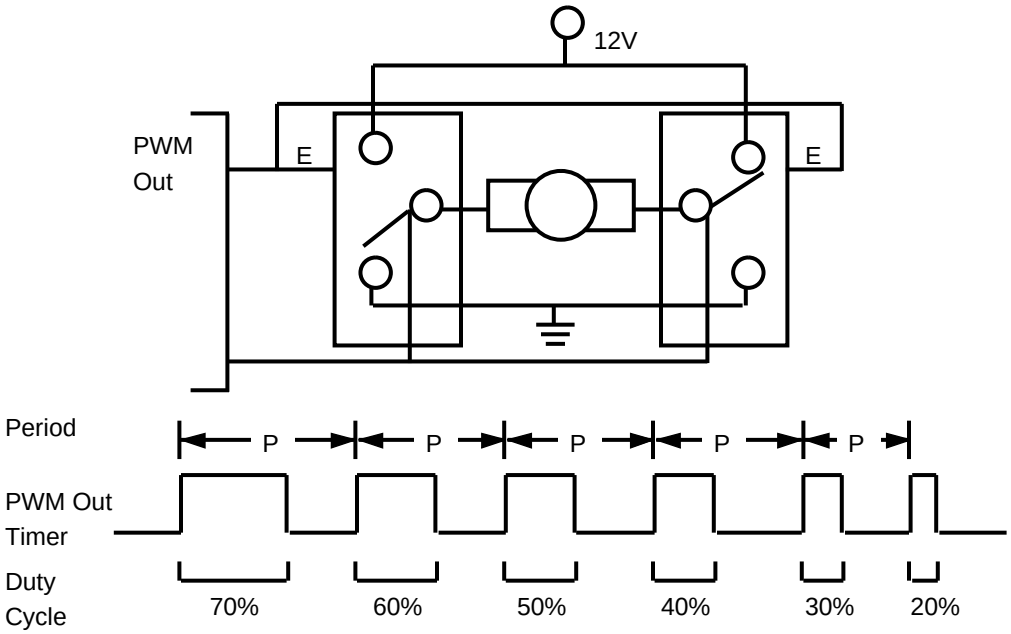


Figure 4-6 PWM Motor Control

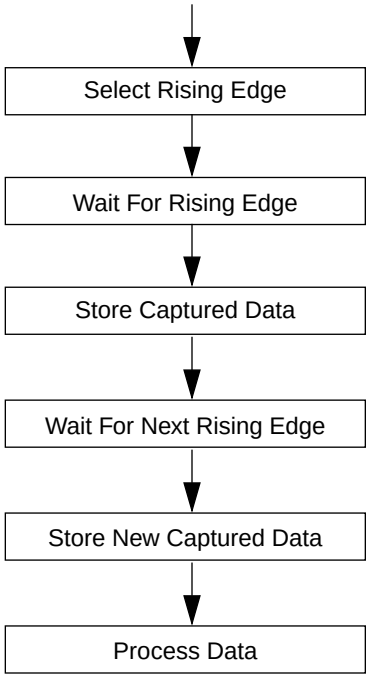
4.4.4 Timer Capture Example

The Timer Input Capture Mode can be used to measure the time between events. The simple block diagram in Figure 4-7 shows how the COP8SAx7 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.

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 VIS instruction vectors the program to interrupt service routine for the timer. 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 a display unit using the MICROWIRE/PLUS interface and a display driver.

NOTE: The T1B input may be used simultaneously to measure the time between different events.



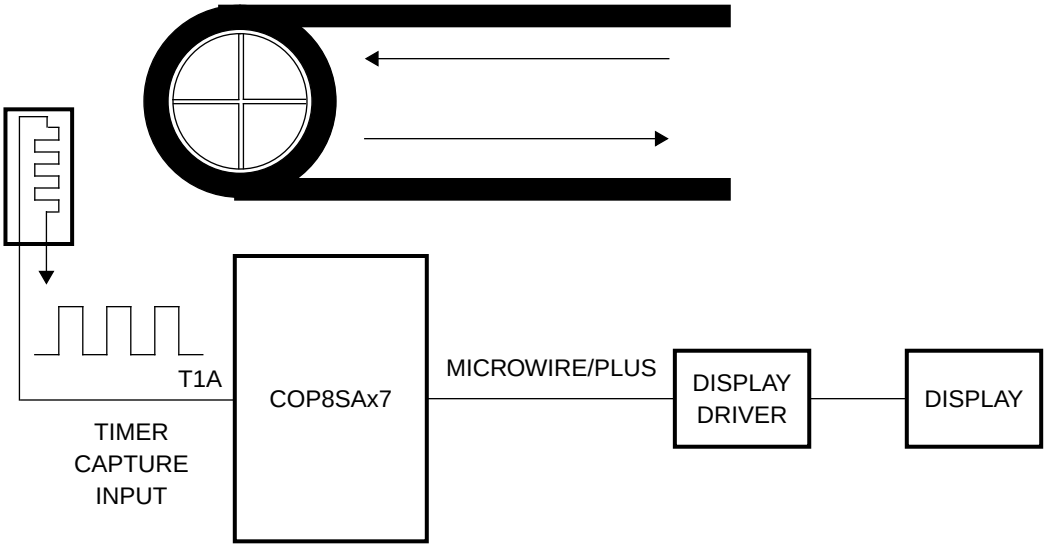


Figure 4-7 Timer Capture Application

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
```

4.4.5 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
```

4.5 TRIAC CONTROL

The COP8SAx7 family of 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 COP8SAx7 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.

COP8SAx7 program 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 COP8SAx7'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
```

4.6 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 microcontroller 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 4-8. 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.

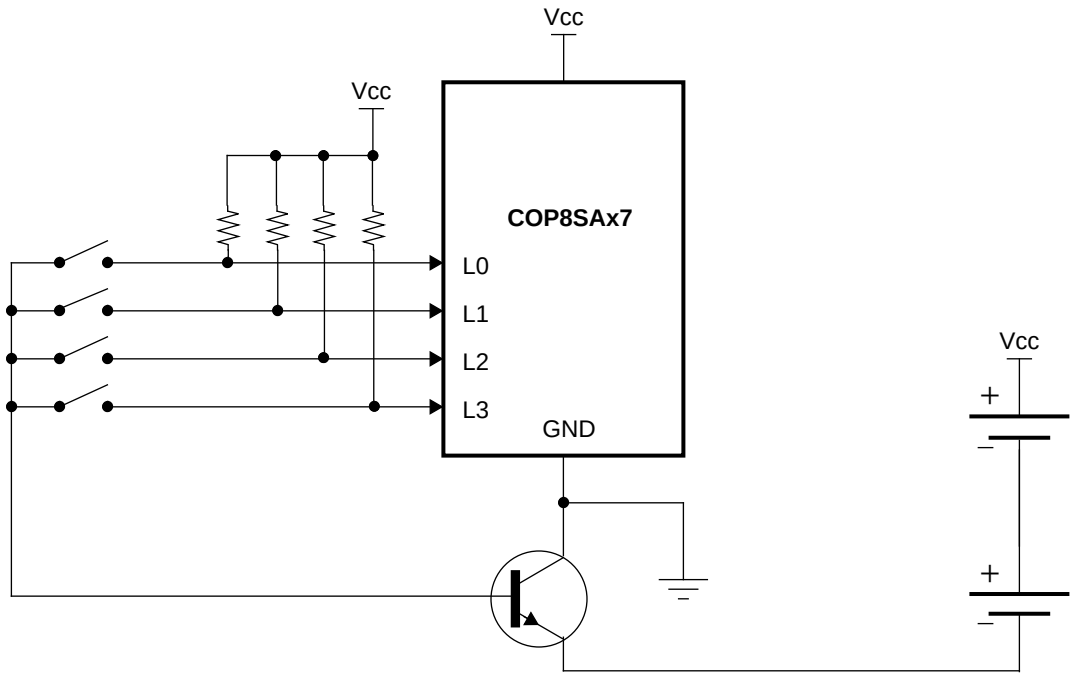


Figure 4-8 Power Wakeup Using An NPN Transistor

An alternative technique is shown in Figure 4-9. 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 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.

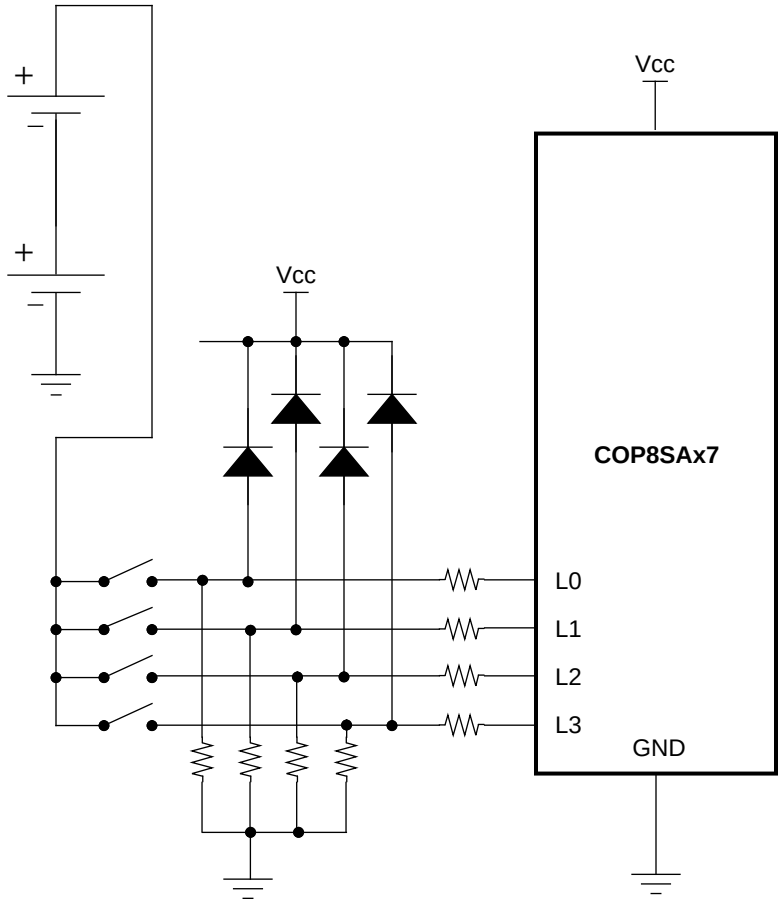


Figure 4-9 Power Wakeup Using Diodes And Resistors

4.7 BATTERY-POWERED WEIGHT MEASUREMENT

Figure 4-9 shows the block diagram of a simple weight scale application. This implementation of weight measurement may be used with the COP8SAx7 device which 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 50% duty cycle signal is generated for the audible tone. A 24-segment LCD display indicates the weight to the user. Four switch inputs are used for configuring the scale.

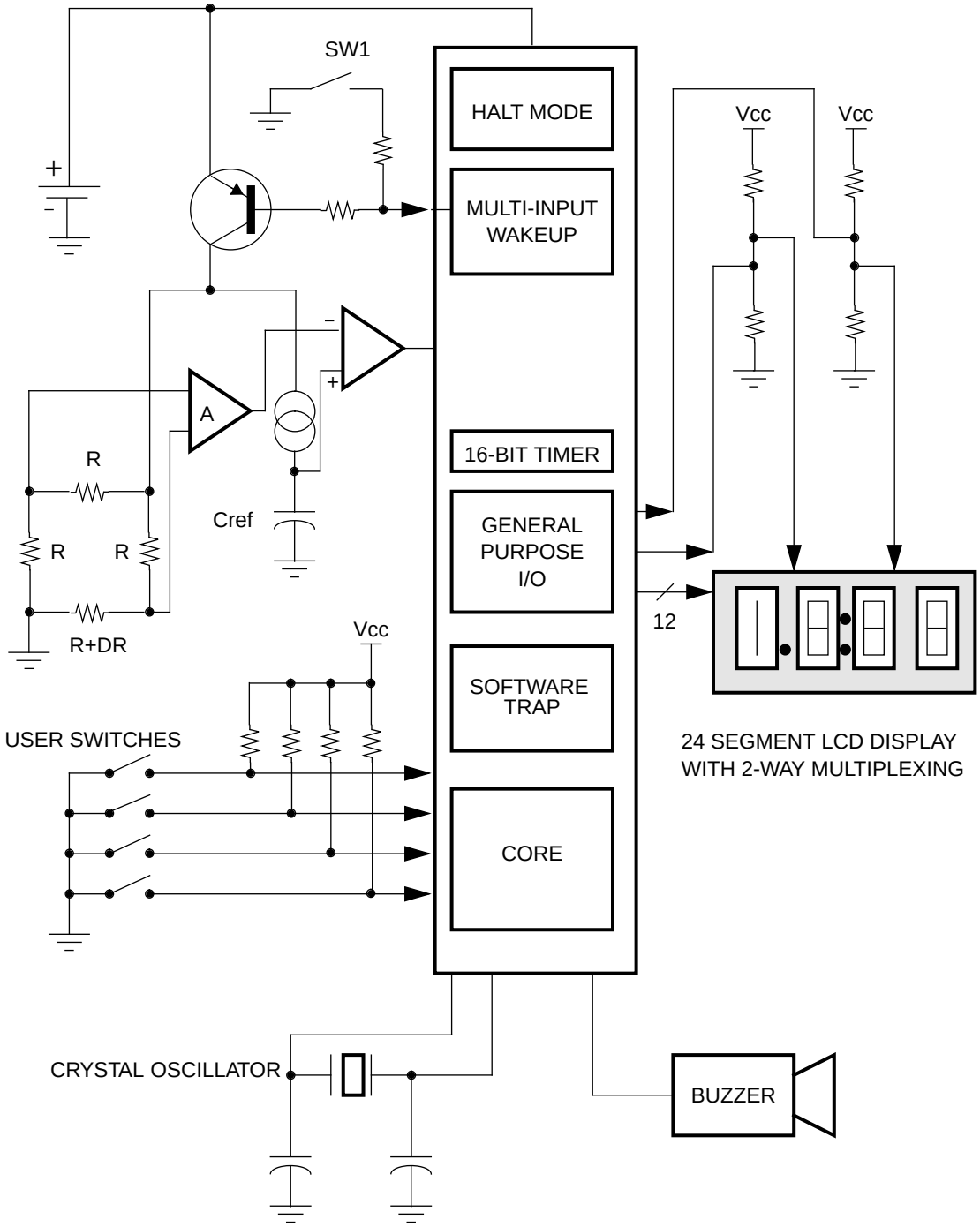


Figure 4-10 Battery-powered Weight Measurement

If the application is not in use, the COP8SAx7 is held in HALT mode. As soon as a weight is applied to the system, SW1 closes and the device 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 on for a programmed period of time even when the switch is open again. The measurement and display are then performed. After completing the measurement and display routines, the COP8SAx7 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. An internal power-on reset circuit is used in this application.

4.8 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 COP8SAx7 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
```

4.9 INDUSTRIAL TIMER

Figure 4-11 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.

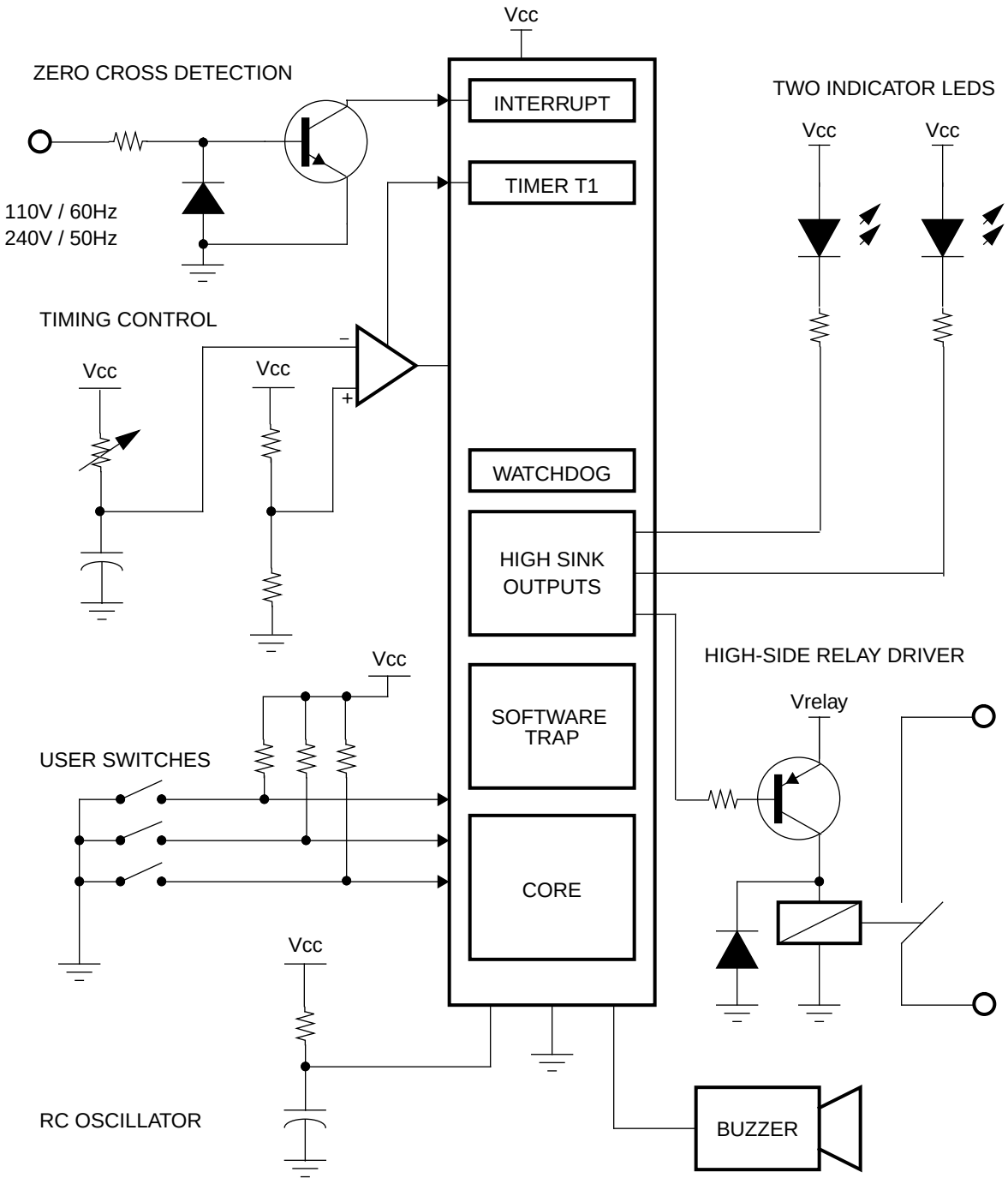


Figure 4-11 Industrial Timer Application

The A/D conversion routine used by this industrial timer is based on the single slope technique. 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 COP8SAx7 can identify the switch positions if the difference in each resistance for each position is greater than the inaccuracy in measuring the absolute resistance.

4.10 COP8SAA7 ELECTRONIC KEY APPLICATIONS

Remote door control is becoming very popular in automobiles and in private homes. In automobiles, the simplest application controls door locks, but more sophisticated uses include comfort control and turning burglar alarms on and off. A combination of the auto and home applications will allow a single electronic key to control automobile doors and burglar alarm, home burglar alarm, garage door opener, and home lighting. A hand-held, reliable electronic key capable of controlling all these functions must be contained in a small package that consumes very little power.

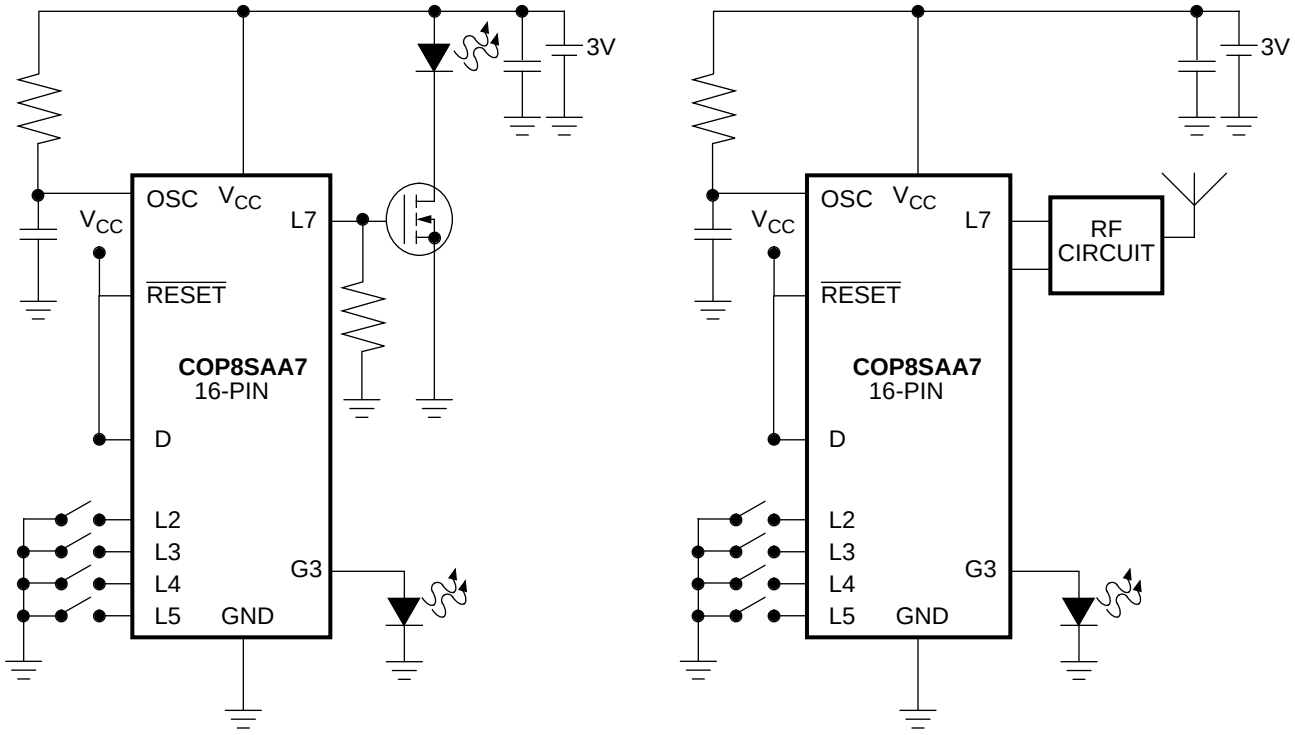


Figure 4-12 Transmitter in Single Cell Operation

COP8SAA7 has the basic requirements to meet the needs of these applications:

- A 16 pin small-outline (SO) package
- A permanently powered HALT mode that consumes typically less than 4 µA
- Reverse battery protection
- Fixed code/rolling code capabilities
- Four channels
- Single-chip solution
- End-of-line programming
- 1K bytes ROM and 64 bytes RAM
- WATCHDOG circuitry

- Hard-wired wakeup signal
- Multi-input wakeup signal

4.10.1 Typical Applications

- Burglar alarms
- Gate and garage door openers
- Central locking
- IR/RF transmission

4.10.2 Flexibility

The COP8SAA7 can operate in both RF and IR applications and fixed and rolling code implementations with or without external EEPROM.

4.10.3 Low Cost

A minimum number of external components are required. Single-cell operation for fixed-code applications is possible, and reverse battery protection prevents damage.

4.10.4 Small Transmitter Size

The device is contained in a small 16-pin SO package.

4.10.5 Low-Cost Version for Rolling Code

This type of application should be implemented with EEPROM, which contains basic code, actual code, and some diagnostic and correctional data. Because the COP8SAA7 does not require a large number of external components, external memory contained in an SO package will not significantly impact the overall size of the system. Figure 4-13 shows a low-cost version of a rolling code IR transmitter implemented with the COP8SAA7 and National Semiconductor's NM93C06 EEPROM.

4.10.6 Receiver Circuit

The receiver circuit usually has additional control functions, which require a large amount of software. In some cases, EEPROM is required for diagnostics.

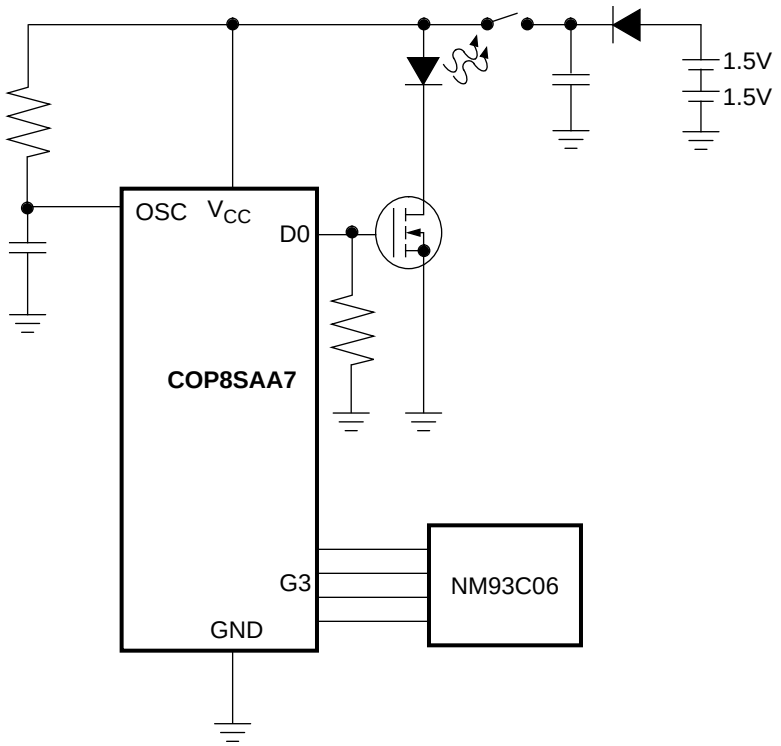


Figure 4-13 Rolling Code IR Transmitter Using External EEPROM

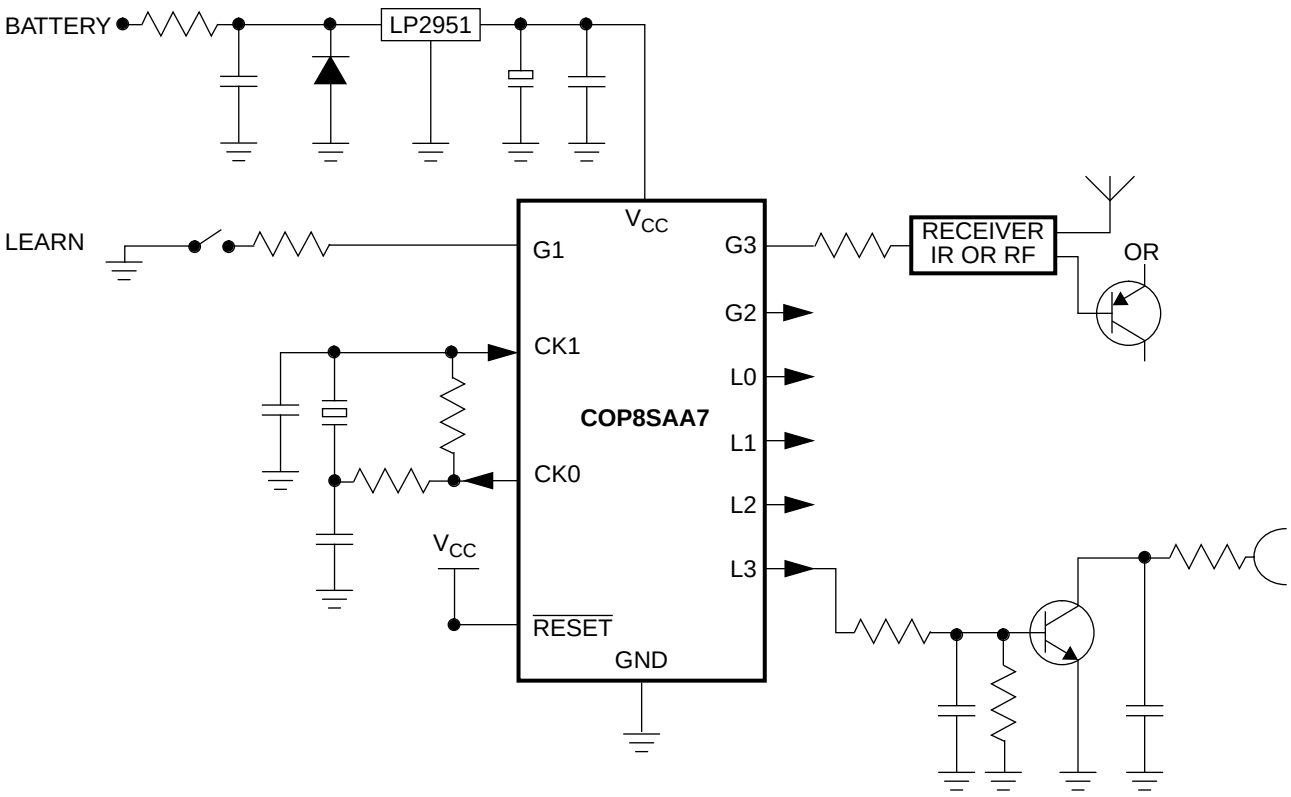


Figure 4-14 Rolling Code IR/RF Transmitter Using One-Chip EEPROM

4.11 COP8SAX7 DIRECT LED DISPLAY DRIVE APPLICATION

Because of its enhanced I/O structure, the COP8SAx7 can be used to directly drive up to 32 LED segments on the 28-pin device. The device features four high-current outputs (L0-L3). Each output can sink at least 15 mA. The 32 LED segments can be driven in multiplexed mode by connecting the L0-L3 lines to the common cathode of the eight-segment LED display. Eight additional outputs are connected to the anodes to select the specific LED segment. The 40/44 pin devices can drive up to 64 LED segments.

Brightness can be improved by reducing the number of muxed lines to three. This still allows users to drive 24 segments, enough for most applications in the appliance market. The core timer (T1) can be used to refresh the display. An external R/C clock generating an interrupt, or the 50/60-Hz frequency of the main power supply, can also be used to refresh the display.

In addition to the pins used for the display drive, additional I/O pins are available for application-specific I/Os. These pins can also be used in conjunction with MICROWIRE/PLUS peripherals, including A/D converters, zero cross detectors with built-in comparator, and serial EEPROM.

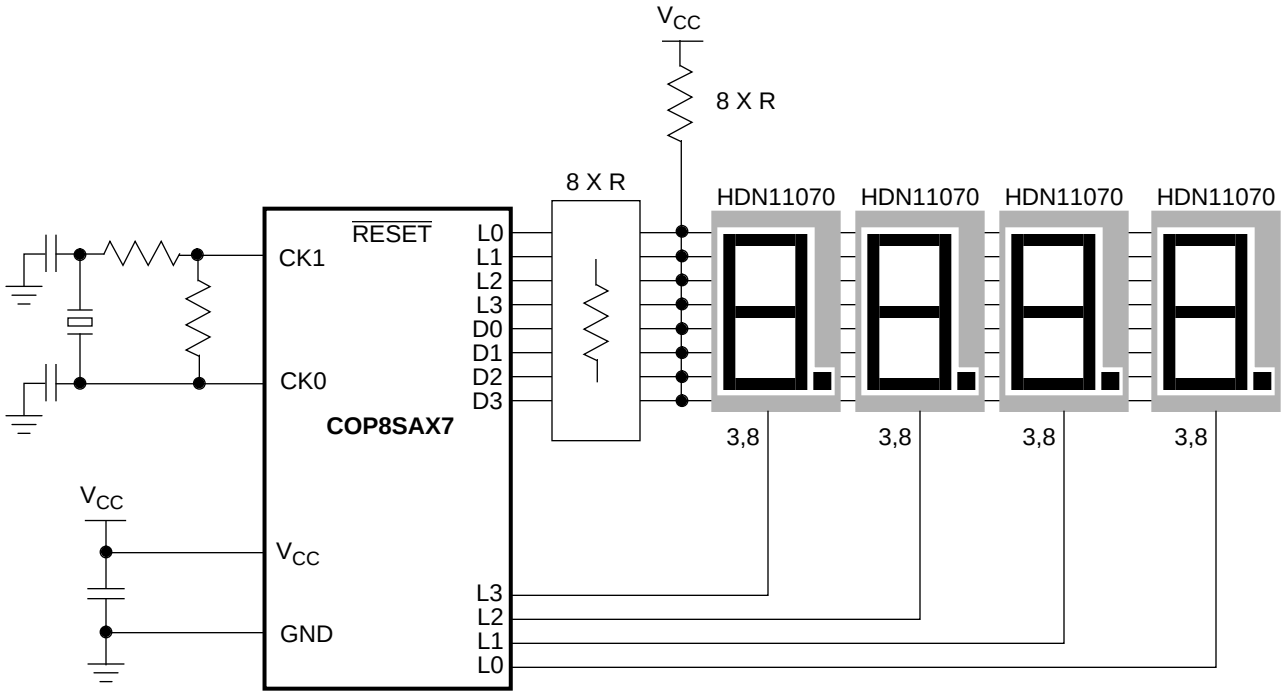


Figure 4-15 LED Direct Drive Using COP8SAx7

4.11.1 Improved Brightness

Another way to improve the brightness of the LED display is to use one additional transistor per muxed line. The muxed line can be any output of the controller, which frees the L0-L3 lines for other purposes (LED drive, triac drive, etc.).

Additional features:

- HALT mode (typically less than 4 µA)
- 16-bit Multi-function timer with two auto-reload registers
- Programmable I/Os
- Multi-input wakeup
- WATCHDOG circuitry
- Software trap interrupt
- Schmitt trigger inputs
- Efficient table look-up capability
- On-chip R/C oscillator
- Internal power-on reset
- MICROWIRE/PLUS Serial Interface

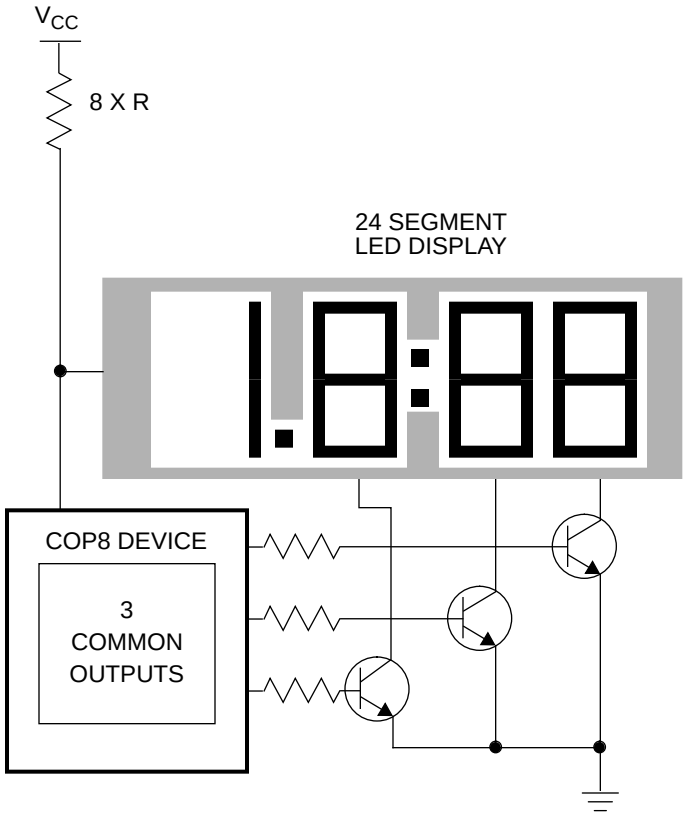


Figure 4-16 LED Drive

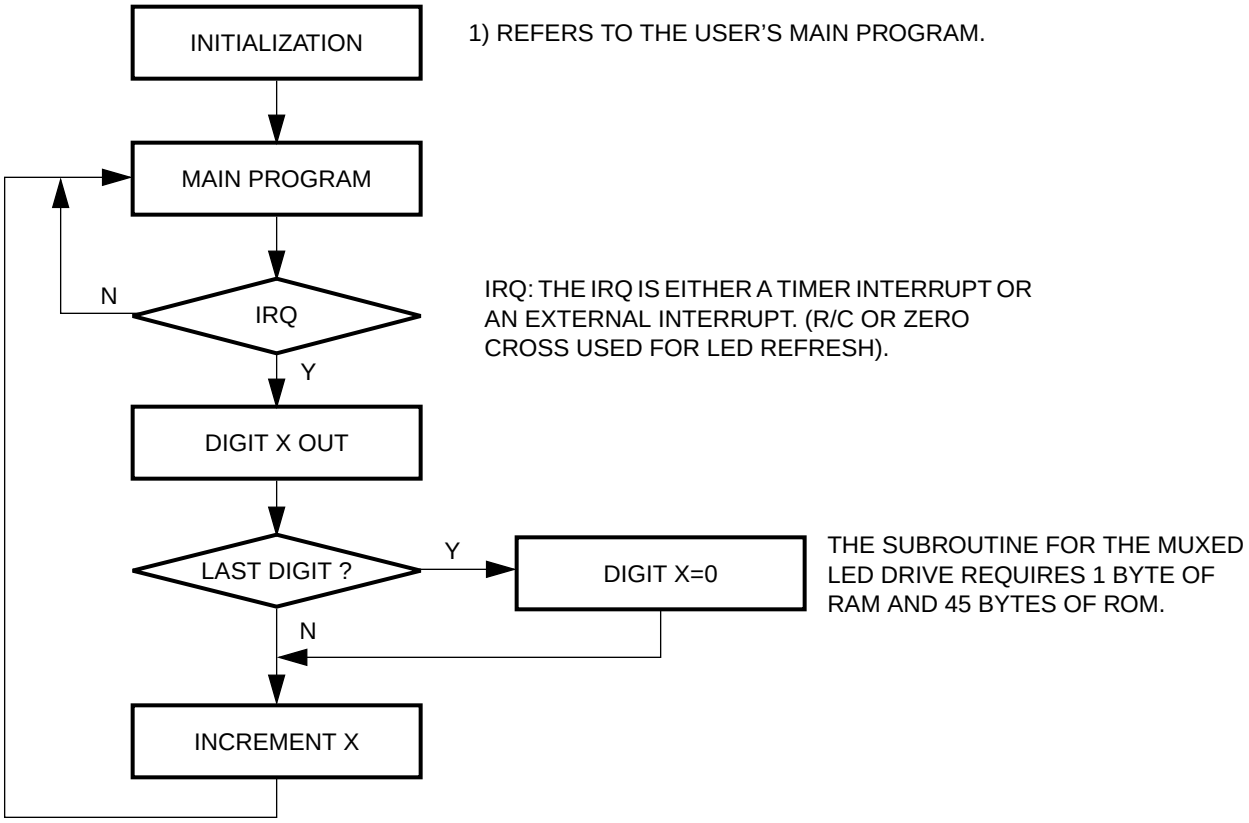


Figure 4-17 Four-Way Multiplexed Direct LED Drive

4.12 CORDLESS PHONE APPLICATION

The COP8 family meets the key requirements of cordless phone applications by providing cost-effective and versatile solutions. The COP8SAx7 device supports key functions of the cordless phone's two-piece set (a portable handset unit and a base station). The set performs standard telephone functions such as network signaling, tone dialing (DTMF), security, last number redial, alerting, and full duplex voice communication. The base station connects to the telephone network and provides an RF linkage to the portable handset using one of the 10 FCC-defined channels in the 46/49 MHz cordless telephone bands. Figures 4-18 and 4-19 show the COP8SAB7 used in a typical cordless phone application.

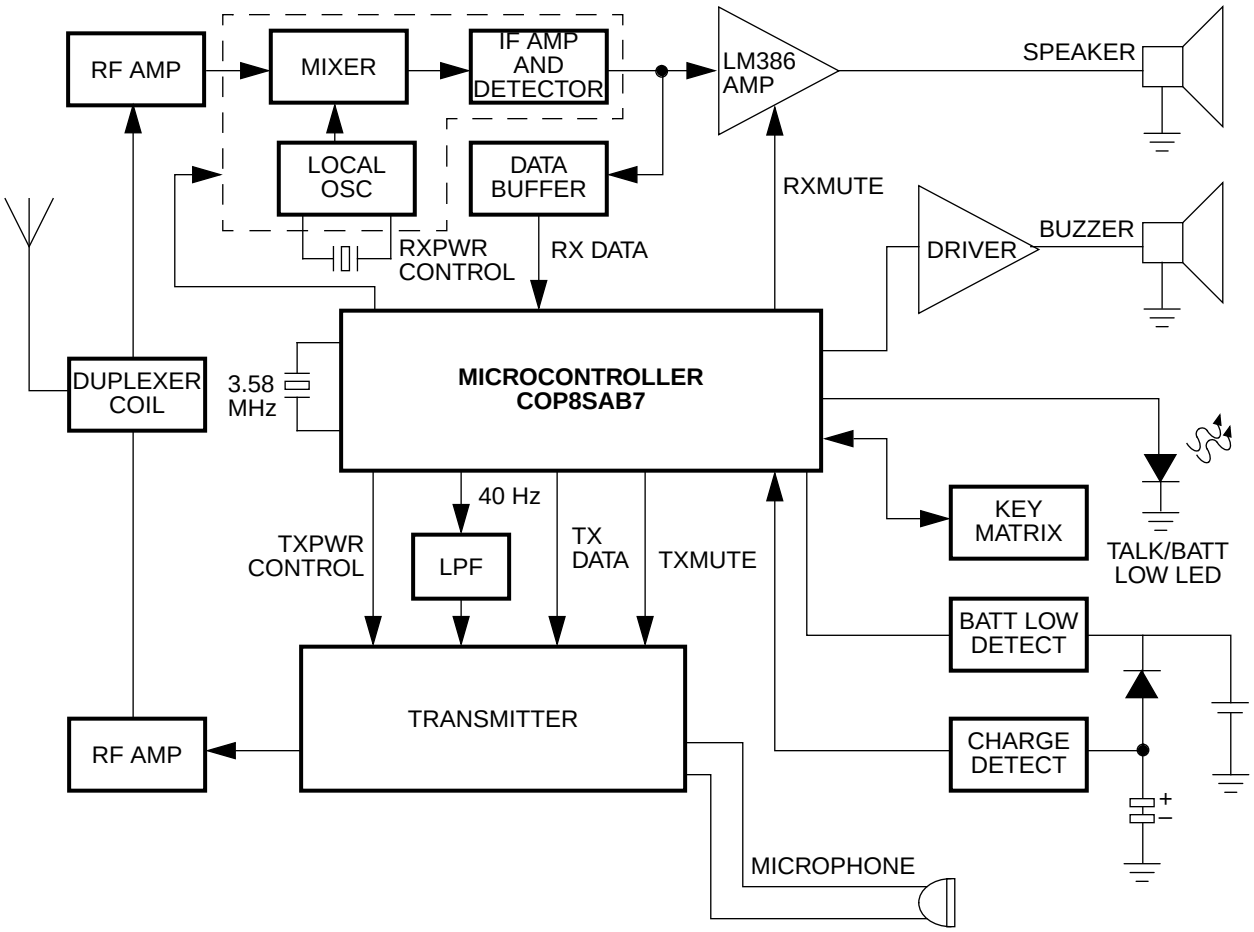


Figure 4-18 Handset Block Diagram

The COP87SAx7 features suited for this application include:

- Low operating voltage (2.7V to 5.5V)
- Low current drain
- HALT mode (typically less than 4 µA)
- 6-bit programmable timer
- Efficient instruction set (efficient table look-up instruction)

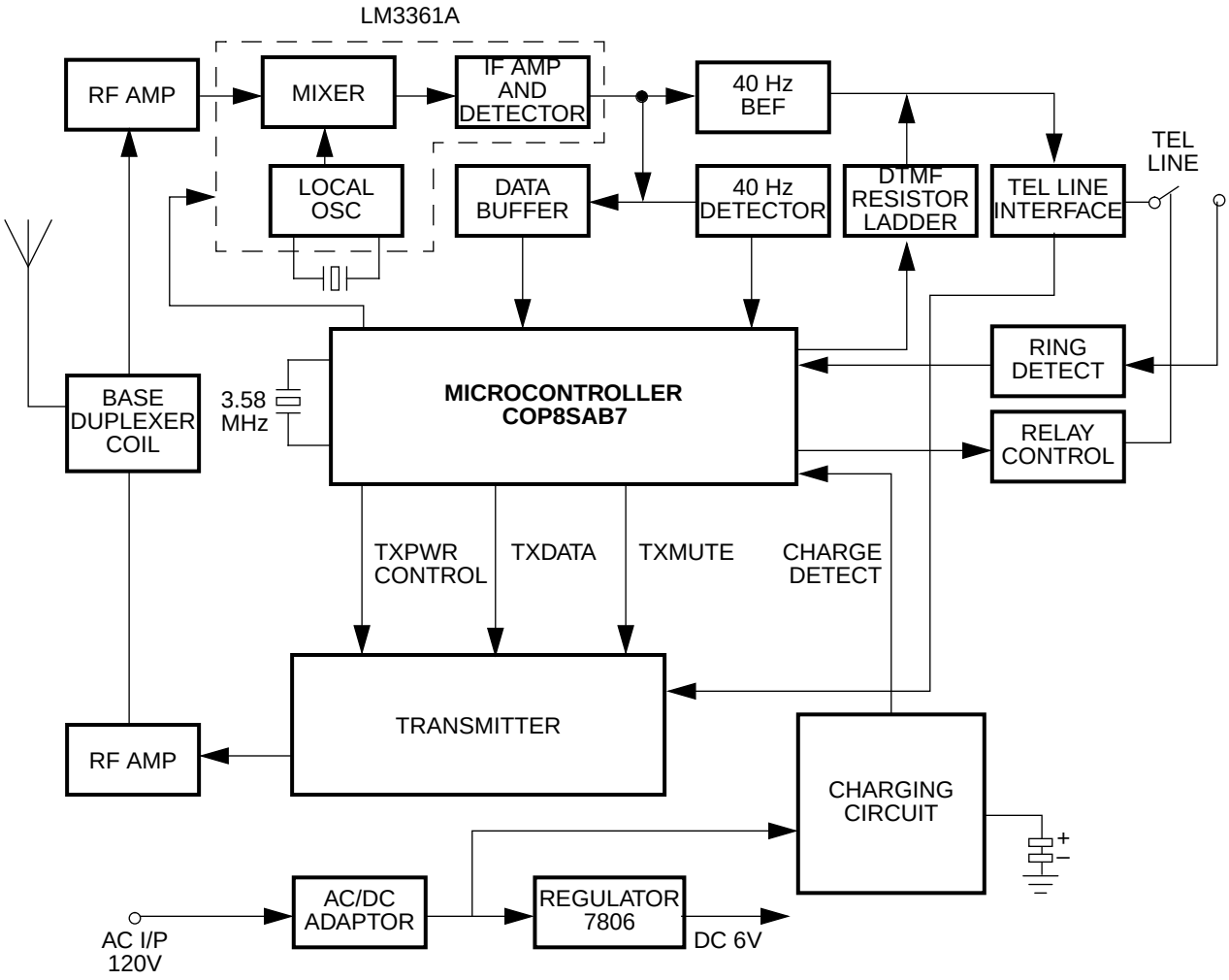


Figure 4-19 Base Block Diagram

- Programmable I/O
- LED direct drive
- Sufficient I/O in a small package
- Schmitt trigger inputs
- Smooth migration from a smaller to a larger device

4.12.1 Typical Application Requirements

Tone Dialing

The DTMF function can be accomplished in software and a 6-bit R/2R resistor D/A network. The program size including tables can be approximately 300 bytes of COP8 code. This software approach can guarantee a frequency tolerance of less than 1.5% and harmonic distortion of less than 27 dB.

Battery Saving Functions

The power saving HALT mode (typicallt < 4 μA) can be used to efficiently manage the life of the handset battery. In the “OFF” mode, both the receiver and transmitter in the handset are turned off. The microcontroller will go into HALT mode. In the “STAND BY” mode, the microcontroller should periodically turn on the handset receiver for a short time duration to monitor any incoming signals.

RF Transmission and Digital Security Code Generation/Transmission

Frequency Modulation, data transmission, and security code generation/transmission can be accomplished with the help of multi-function 16-bit timer and the hardware interrupt capability. The l6-bit timer can generate accurate control timing and the desired tone output. It is important to note that the same device can be used in the base as in the handset.

National offers a range of COMBO, EEPROM, Linear Audio, and 8-bit microcontroller products which provide a complete low-power and low-voltage solution.

4.13 COP8SAC7 BASED AUTOMATED SECURITY/MONITORING APPLICATIONS

The architecture, features, and flexibility of the COP8SAx7 family of microcontrollers provide cost-effective solutions for security/monitoring by eliminating external components from the circuit. Figure 4-20 demonstrates an application example of a security/monitoring system using the COP8SAC7. The application also illustrates interfacing the device to a number of specialized peripherals using a minimum number of I/O lines.

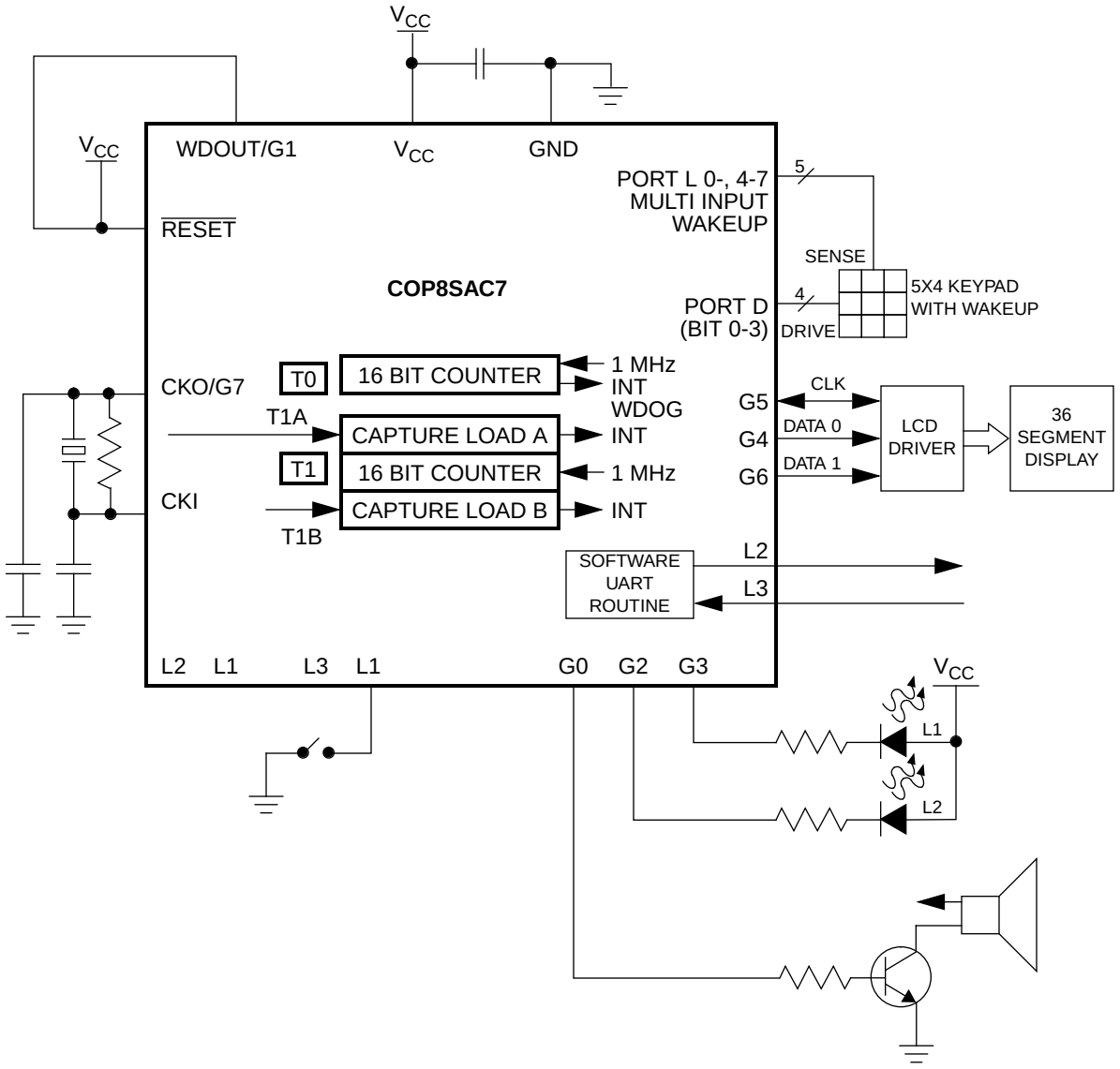


Figure 4-20 Example of a Security/Monitoring System

COP8SAC7 features suited for these applications include:

- Low-power HALT and IDLE
- 16-bit timers, each with two 16-bit registers supporting processor independent PWM mode, external event counter mode, or input capture mode
- MICROWIRE/PLUS serial communication

- Multi-Input Wakeup/interrupts
- Powerful table lookup and indirect addressing modes
- Output pins have direct drive capability
- Fully static HALT mode with hardware restart capability
- WATCHDOG timer
- Programmable I/O
- 254 consecutive bytes of table lookup data in one block
- 10/14 multi-source vectored interrupts
- Schmitt trigger inputs
- LCD arithmetic instructions

The COP8SAC7 is the heart of the system that provides the processing power to scan the keypad, service the receiver interrupts, update the real-time clock, serially communicate with the display unit and data storage unit, and activate the auto-dialer unit. System capabilities may be enhanced or scaled down by simply changing the processor's algorithm.

4.13.1 Typical Application Requirements

Keypad Scanning

The keyboard is organized as a matrix. The L port (Multi-Input Wakeup capability) is configured as an input with weak pull-ups, and is used to sense the keypad inputs. Most of the time the chip is in the current-saving HALT mode (Idd < 4 µA). Any keystroke can create a high to low transition one of the L lines, which wakes up the microcontroller from HALT mode. After returning from the HALT mode, the keypad is scanned in order to detect which key is pressed. This event-driven keypad scanning technique results in lowest possible current consumption because the HALT mode is used between successive keystrokes.

Display Terminal Unit

The display terminal unit can interface with the COP8SAC7 through the software UART. The COP8SAC7 is interrupted by the terminal and the microcontroller decodes the character sent and services the corresponding request. The terminal keyboard can be used to program the telephone numbers to be dialed by the auto-dialer unit. The real-time clock is displayed on the terminal screen.

LCD Display Unit

The LCD display unit can be used to display the time and date information. The microcontroller can communicate with the LCD display driver serially using the MICROWIRE/PLUS serial interface.

Time Keeping

The time keeping routine is the most important software routine and is executed irrespective of the other modules being executed. The program uses the IDLE timer T0 for this purpose. The IDLE Timer is a 16-bit timer and runs continuously at a fixed rate of the instruction cycle clock. The IDLE Timer counter is not memory mapped and consequently, the user can not read or write to it. The toggling of the twelfth bit of the IDLE counter can be programmed to generate an interrupt. The interrupt is generated every 4 ms at the maximum instruction cycle clock rate of 1 µS. The software uses this interrupt to update counters in Data Memory for time keeping. The time keeping routine then sets a flag to update the display which is then used by the main program.

4.14 COP8SAC7 KEYBOARD APPLICATIONS

The COP8SAC7 microcontroller can be used to cost-effectively provide a solution for a laptop keyboard. With only a few passive components, the device can accomplish the complete keyboard scan function. The same principles can be applied to all types of keyboards (i.e. desktop keyboards) or data input devices. With low power consumption being the most important design criteria, the device offers the power saving Multi-Input Wakeup mode.

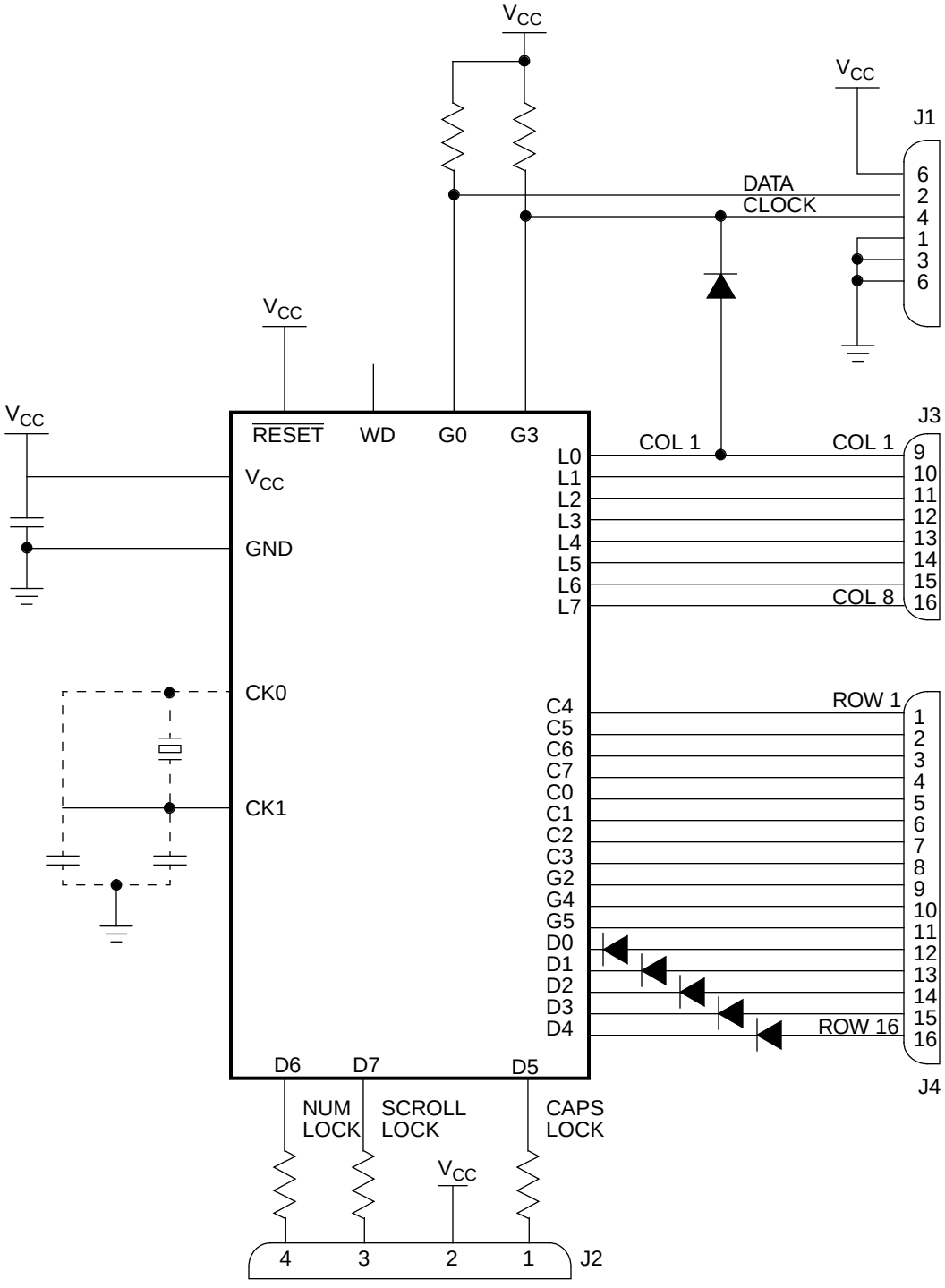


Figure 4-21 Laptop/Notebook Keyboard Schematics

The device features suited for these applications include:

- Single Chip solution
- Low cost on-chip R/C oscillator (or optional ceramic oscillator)
- Output pins have LED direct drive capability
- Internal power-on reset
- Programmable I/O with on-chip pull-ups
- Powerful table lookup and indirect addressing modes
- Fully static HALT mode with hardware restart capability
- Multi-Input Wakeup
- WATCHDOG timer
- 254 consecutive bytes of table lookup data in one block
- Multi-source interrupt (external interrupt with selectable edge, timer interrupt with selectable edge, and software interrupt)
- Schmitt trigger inputs

4.14.1 Typical Application Requirements

Keyboard Scanning

The keyboard is organized as an 8 input by 16 output matrix. Software key rollover eliminates the need for decoupling diodes in the 16 by 8 key matrix. The COP8SAC7's L port is configured as a weak pull-up input port, thus allowing the use of the Multi-Input Wakeup feature. Most of the time the chip is in the current saving HALT mode (<4 µA) Any keystroke or a data transmission from the computer will create a high-to-low transition one of the L lines, which wakes up the microcontroller from HALT mode. After returning from the HALT mode, the keyboard is scanned in order to detect which key is pressed and the appropriate key code is sent to the computer. This event-driven keyboard scanning technique results in lowest possible current consumption because the HALT mode is used between successive keystrokes.

LED Direct Drive

Three LEDs are driven directly by three of the device high sink D-lines (15 mA sink capability), thus eliminating the need for additional LED drivers or transistors.

4.14.2 Typical Applications

- Security alarm/monitors
- Appliance control
- Laptop/notebook/desktop keyboards

More Information

A complete keyboard solution is described in Application Note AN-734.

4.15 COP8SAA7 CLOSED LOOP TEMPERATURE CONTROL APPLICATIONS

Closed loop air control is the primary function of air-conditioning/heating systems in automobiles.

4.15.1 Primary System considerations:

- Low component cost
- Low power
- Small physical size
- Low EMI

The COP8SAA7 features suited for this application include:

- 16-bit PWM timer
- Enhanced outputs for LED and LCD display drives
- Additional hardware interrupts
- High-current programmable outputs
- Software trap interrupts
- Schmitt trigger inputs
- MICROWIRE/PLUS

The COP8SAA7 based Closed Loop Temperature control system, shown in Figure 4-22, performs the following:

- Measures the user-set potentiometer with a simple R/D converter
- Measures the actual temperature with a successive approximation A/D conversion using the PWM timer
- Controls the air mix motor speed with a PWM timer
- Reads the keyboard with standard keyboard scanning routines
- Controls the LED driver, solenoid driver (for A/C) and EEPROM through the MICROWIRE/PLUS serial interface.

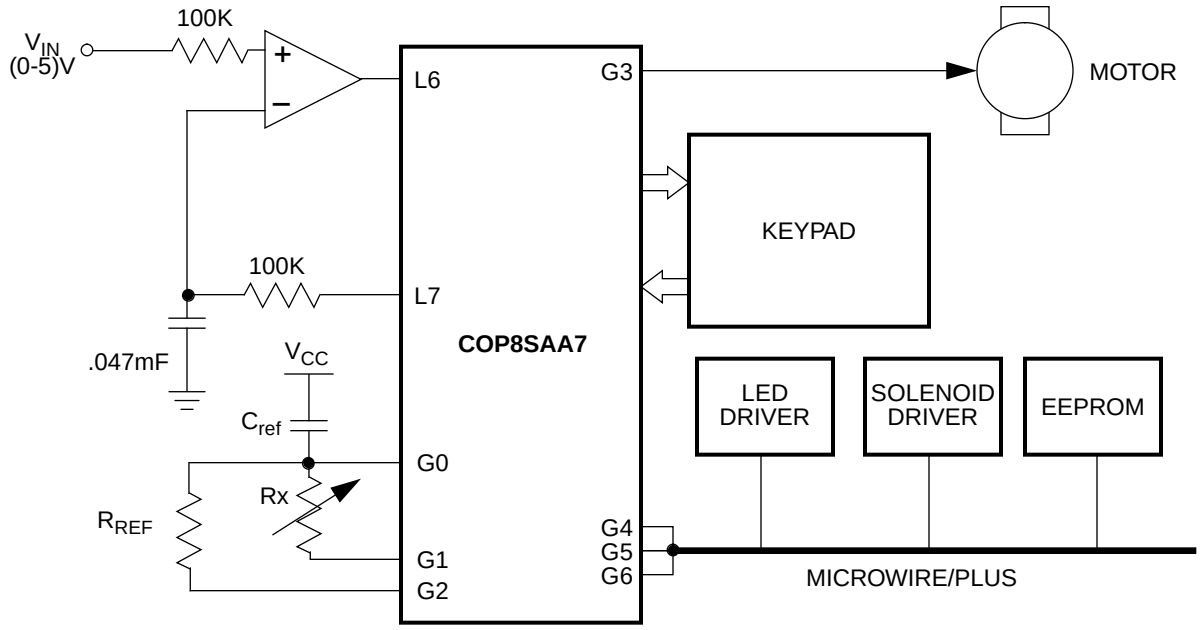


Figure 4-22 Automotive Closed Loop Air Control

4.15.2 Typical Requirements for Motor Control Systems

Single Slope A/D or V/F Conversion

Users can build a high-resolution A/D converter by using an external comparator with the on-chip 16-bit timer.

Small LED or LCD Display Units

Eight high-current outputs, drive to rail capability, programmable I/Os, enhanced table lookup capability, and two pointer registers support the installation of small two or three way multiplexed display panels. The COP8SAA7 can communicate with the LCD/LED display driver serially using the MICROWIRE/PLUS serial interface.

4.16 AUTOMATIC WASHING MACHINE

The COP8SAC7 can be used as a key control element in an automatic washing machine control module. The device can improve reliability, functionality, and bring specific features to the application.

The COP8SAC7 incorporates features to meet the basic requirements of the appliance market:

- 16-bit multi-function timer with two auto-reload registers
- Enhanced outputs for LED and LCD display drives
- High-current programmable outputs
- Schmitt trigger inputs
- WATCHDOG/clock monitor circuitry
- Software trap interrupt
- Additional hardware interrupts
- MICROWIRE/PLUS serial interface

Figure 4-23 shows the block diagram of an automatic washing machine control unit.

Following is the description of main functions:

4.16.1 Reliability and Safety Features

The WATCHDOG circuitry prevents an application program from continually repeating an infinite loop. The Software Trap (ST) reports corruption of the program counter (over popping of the stack, for example) by initiating a non-maskable interrupt.

4.16.2 LED or LCD Display Units

Eight high-current outputs, drive-to-rail capability, programmable I/Os, powerful table look-up capability, and two pointer registers support the installation of multiplexed display panels. The configuration shown in Figure 4-24 supports 24 LED segments, two digital LED blocks, and an 11-key display and scan function.

4.16.3 Zero Cross Detection

The G0 external interrupt pin with programmable edge polarity allows installation of a simple time-base counter, using the “main power” zero crossing.

4.16.4 Other I/O Functions

Pins F2 and F3 are used for detection of water-in, drain, unbalance, and upper cover sensor circuit. Pin F4 is used to monitor motor loading to judge the weight of the material being washed and select proper rinse and spin procedures.

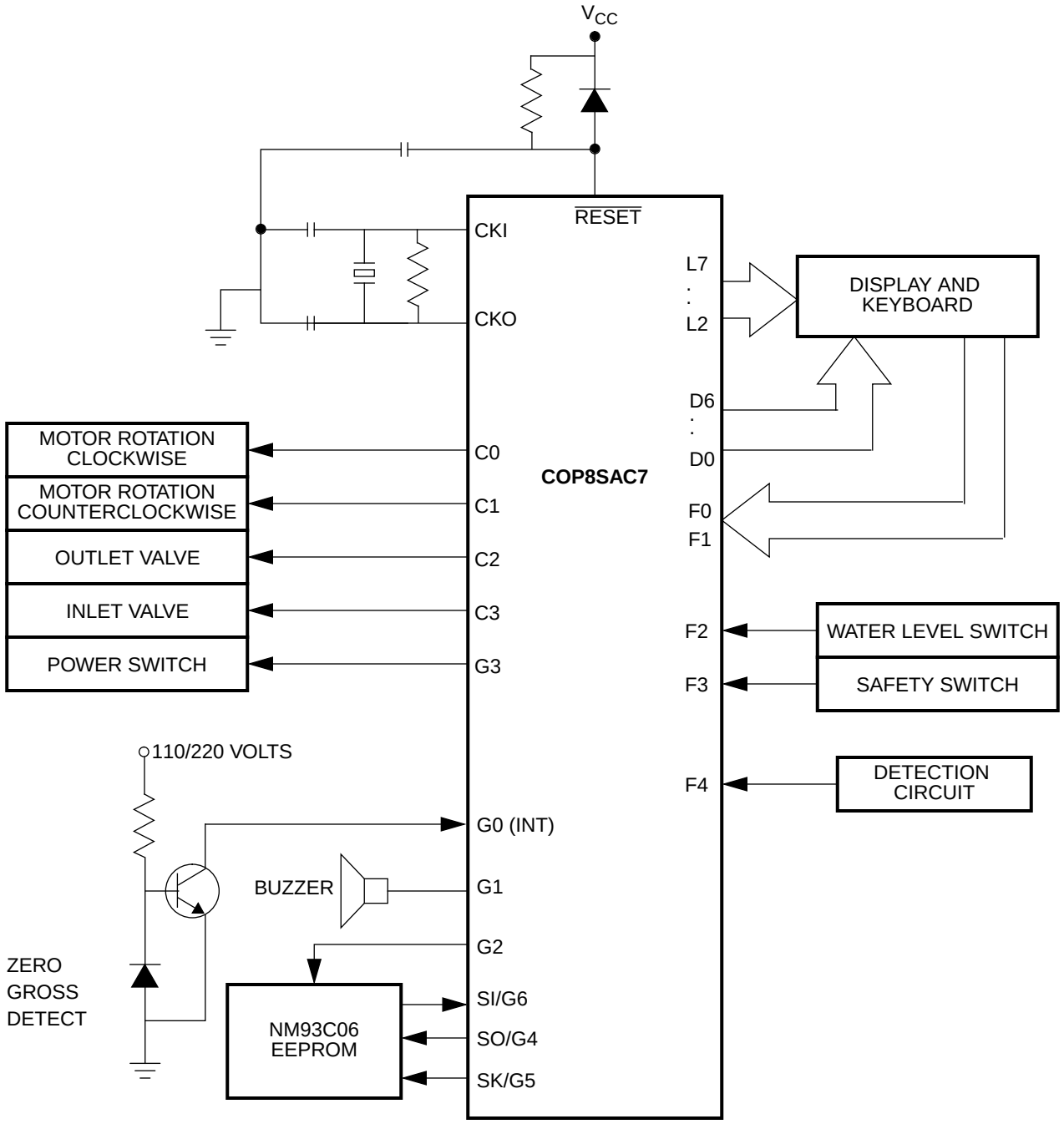


Figure 4-23 Automatic Washing Machine Control Model Using COP8SAC7

Pins C0-C3 and G3 provide the output drive for the triacs used to control the motor, outlet valve, inlet valve, and power switch.

4.16.5 External EEPROM Interface

The MICROWIRE/PLUS serial interface can be used to interface the device to an external 256-bit EEPROM C93CO6). The external EEPROM is used to store the full wash procedure last used so the washing machine uses the same procedure after power up.

4.16.6 Software Considerations

The application program can select or preset washing procedures not only before the start of a wash cycle but also during the washing cycle. This means the program is not a simple sequential control program but a real-time alterable program.

Figure 4-24 and 4-25 show the program flowcharts.

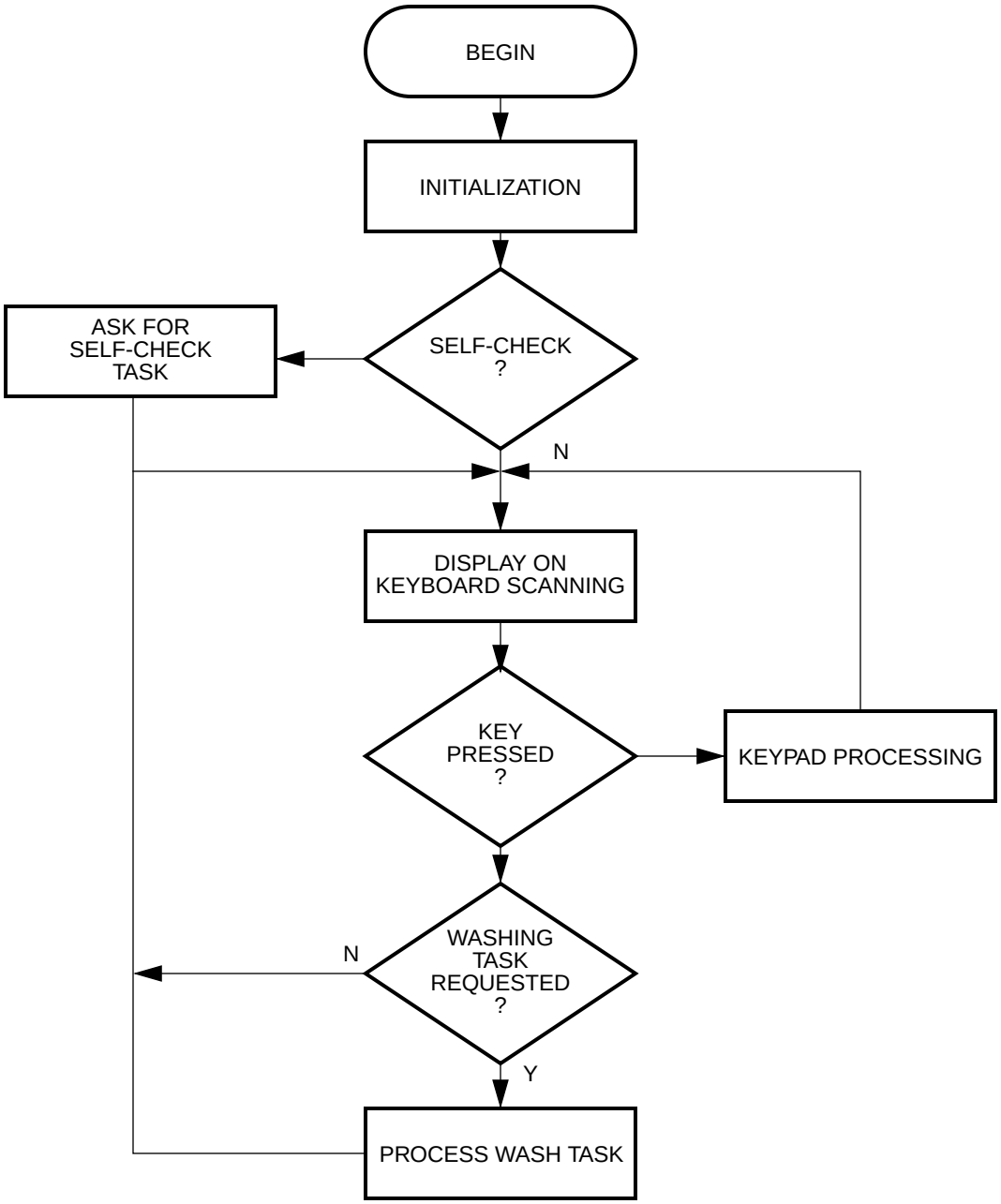


Figure 4-24 Main Program Flow

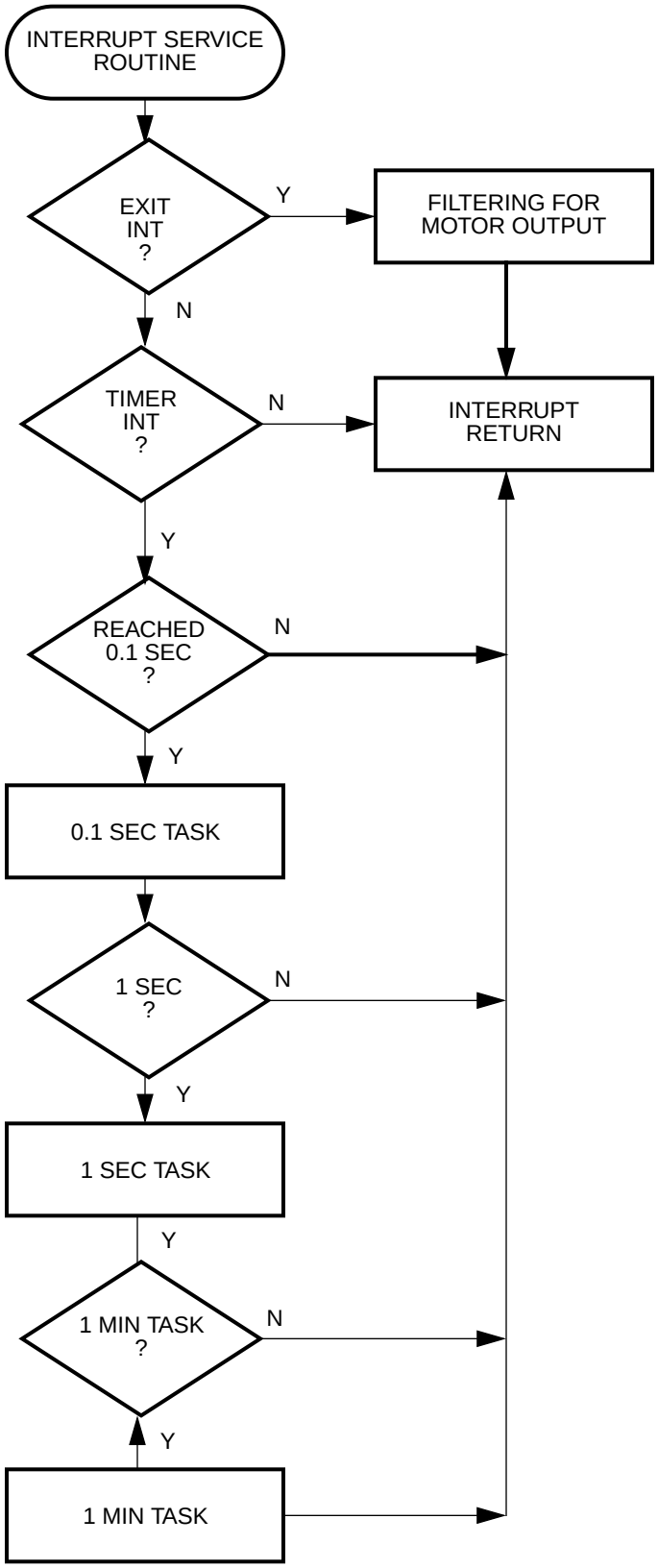


Figure 4-25 Interrupt Routine Flow

4.17 AIR CONDITIONER CONTROLLER

The COP8SAB7 can be used to support functions such as sensing, cooling, and dehumidifying in a complete air conditioning system. The device provides over-voltage and under-voltage control and protection for the compressor and fans.

Figure 4-26 shows the block diagram of an air conditioning system using the COP8SAB7.

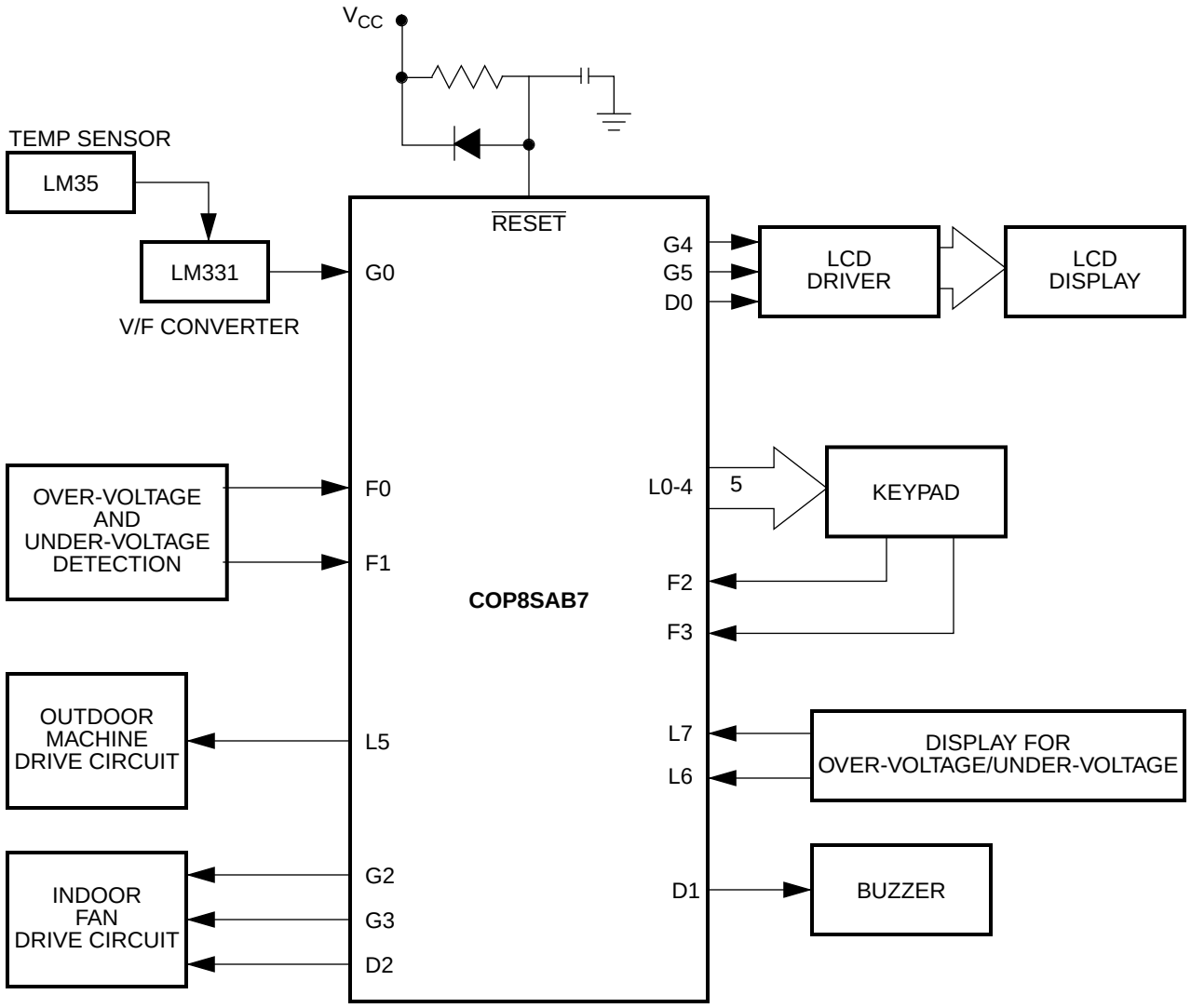


Figure 4-26 Block Diagram of Air Conditioning Control Module

4.17.1 Temperature Detection

The LM35 temperature sensor is used to generate the input voltage for the LM331 V/F converter. Based on the input voltage, LM331 presents the proportional output frequency to the G0/INT pin of the COP8SAB7. The program counts the number of interrupts to calculate the frequency and translates the frequency to the appropriate temperature value.

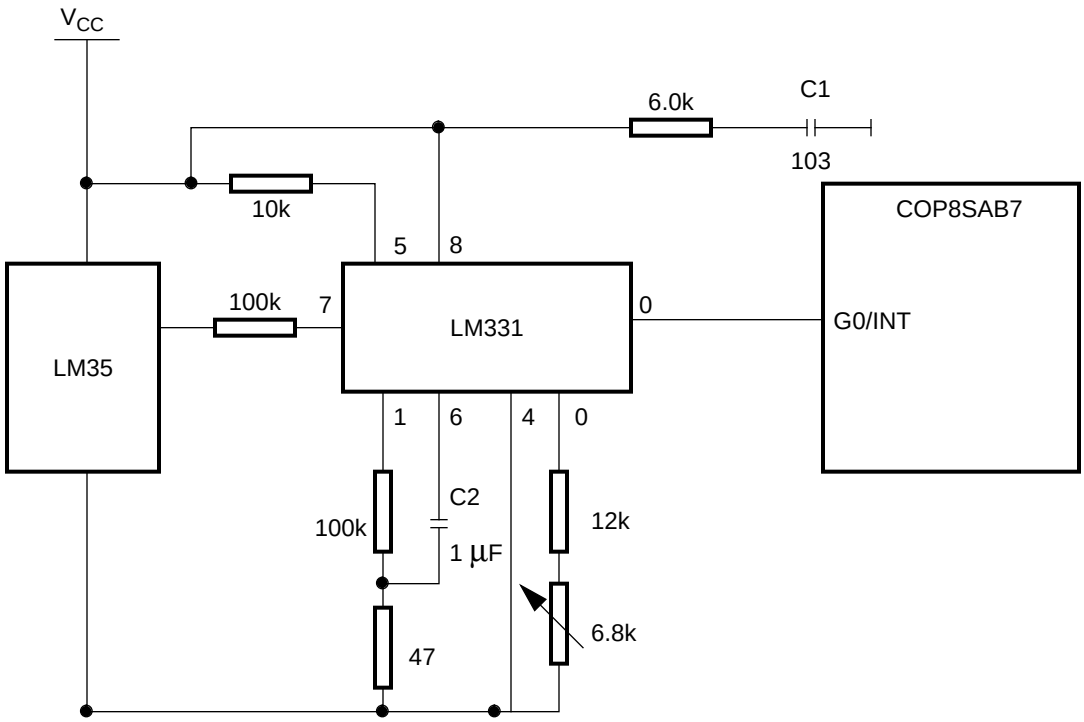


Figure 4-27 Temperature Detection Circuit

4.17.2 Keypad Scanning

Figure 4-28 shows the keypad organization. Pins L0-L4 are normally output high. The input pins F2 and F3 are low when no key is pressed and high when a key is pressed.

4.17.3 Over-Voltage and Under-Voltage Detection

The status of line voltage range is read through the F0 and F1 pins. Reading lows at these pins indicate a line voltage out of normal range. Reading highs indicate the line voltage is within the normal range. See Figure 4-29.

4.17.4 Drive Circuits for Fan, Compressor and Buzzer

Figure 4-30 shows the appropriate drive circuitry. Relay J controls the compressor switch. Three triacs are used to control the fan in high, medium, and low states. Figure 4-29 shows only one triac. The MOC3041 is an optoelectric isolation driver with zero cross trigger. When one key is pressed, the D pin outputs a positive pulse to activate the buzzer FM0.

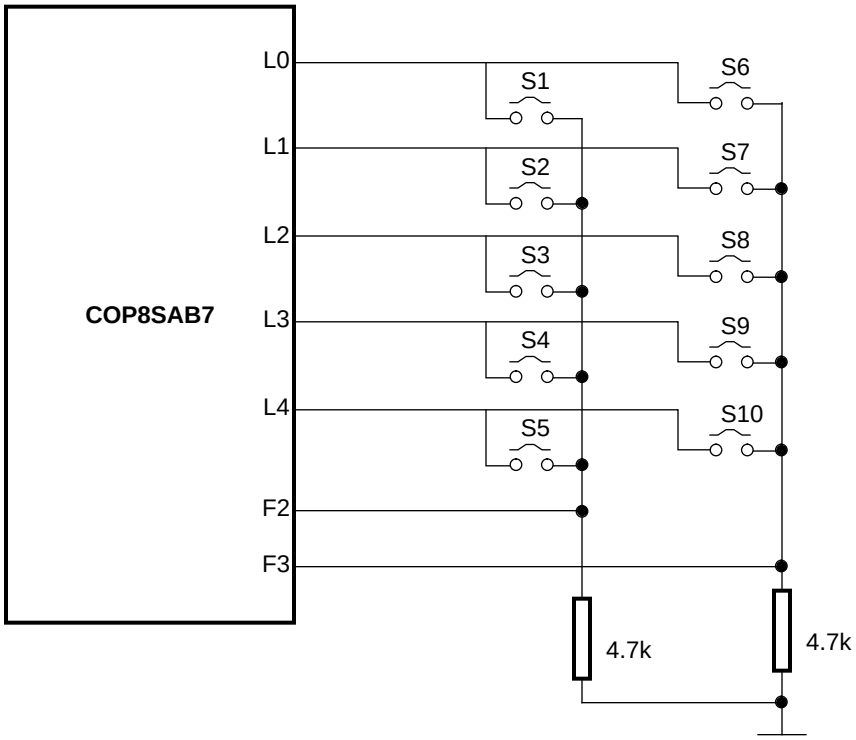


Figure 4-28 Keypad Scanning

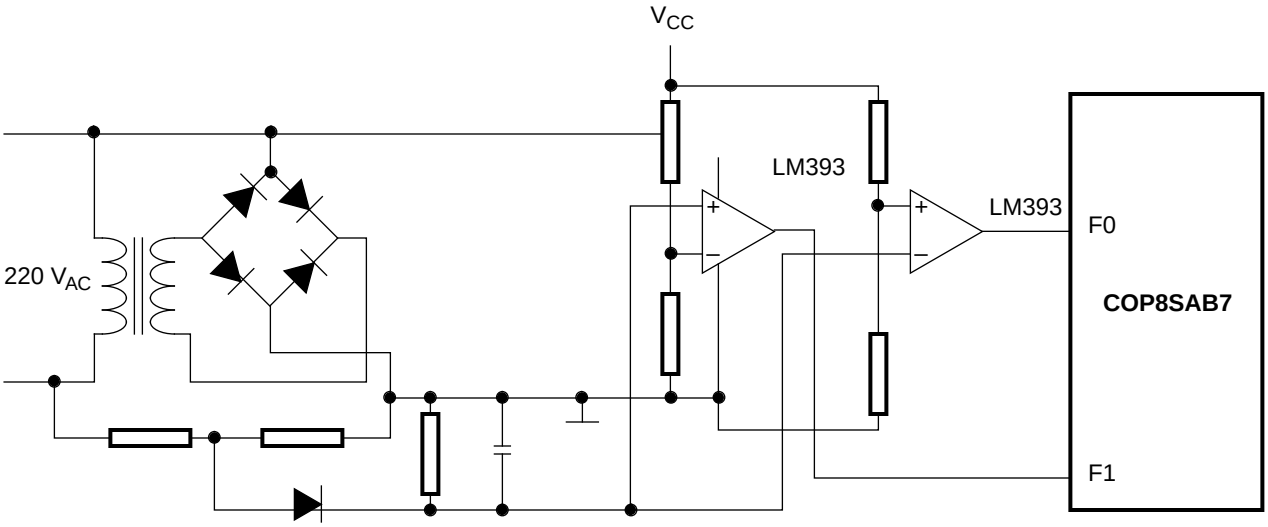


Figure 4-29 Over-Voltage and Under-Voltage Detection Circuit

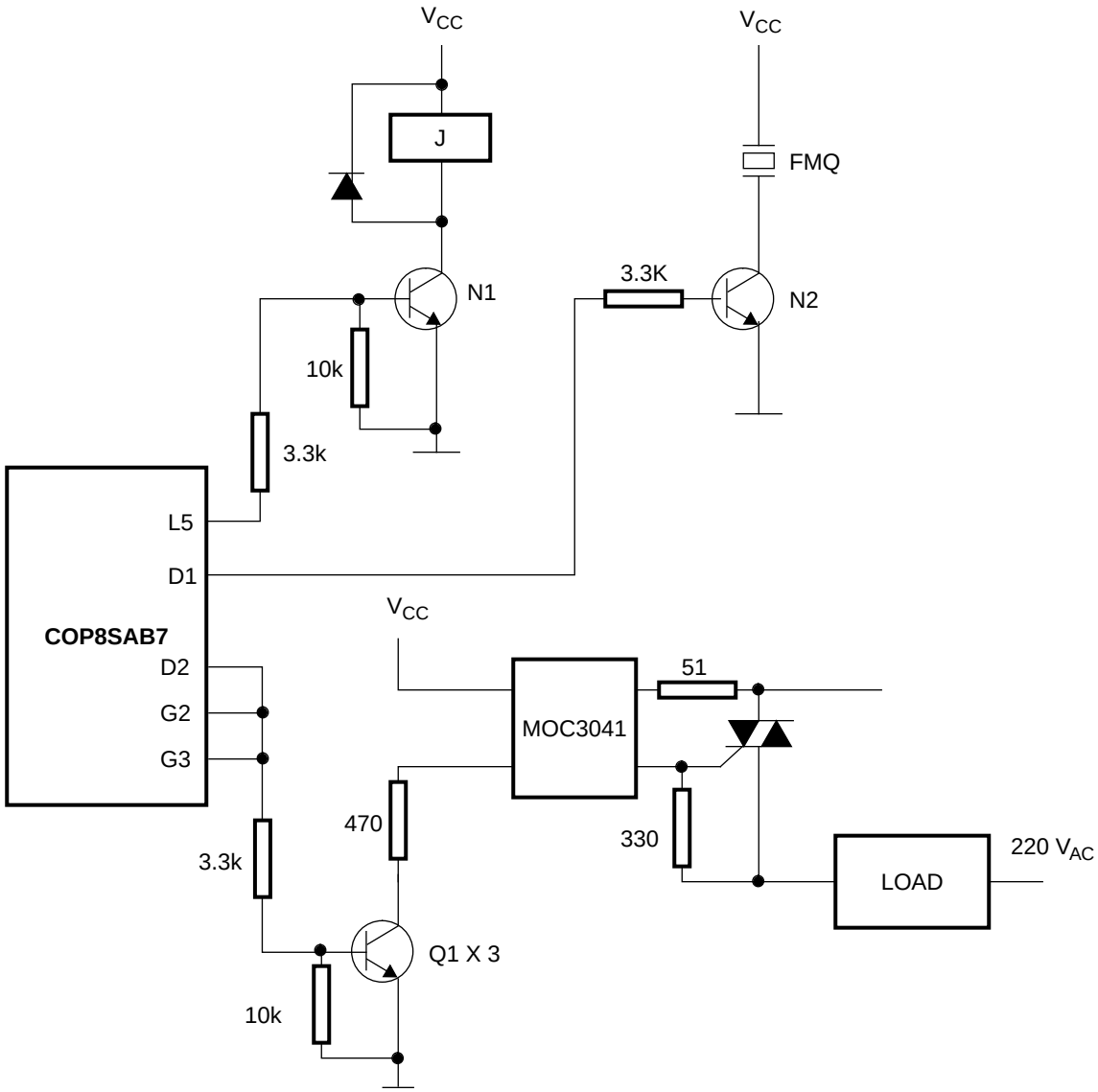
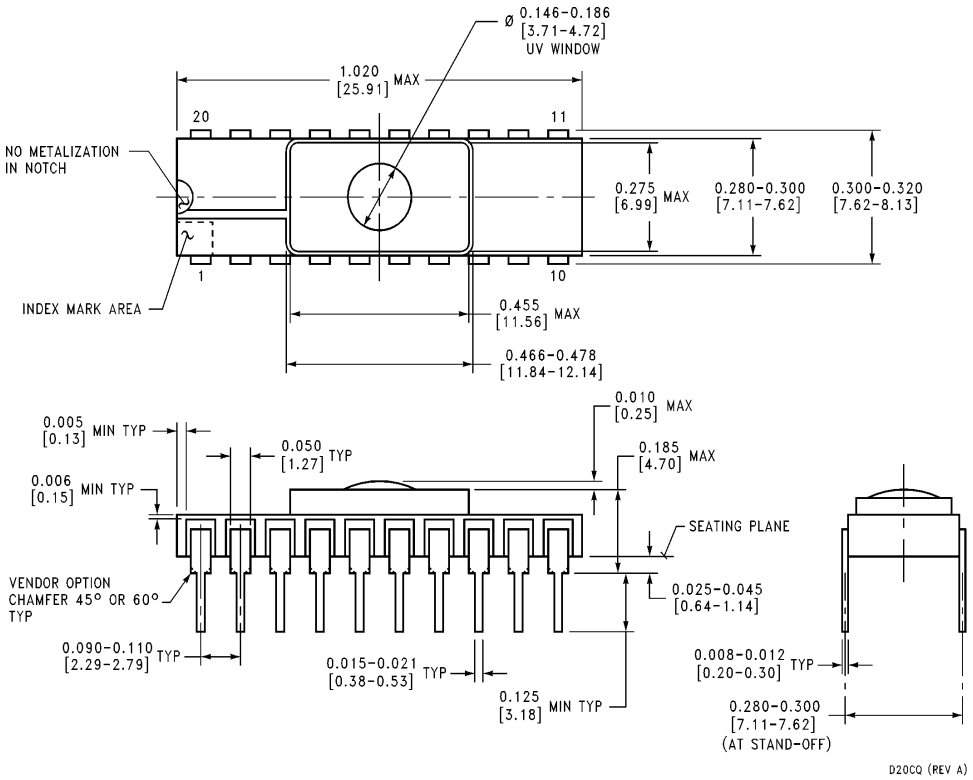


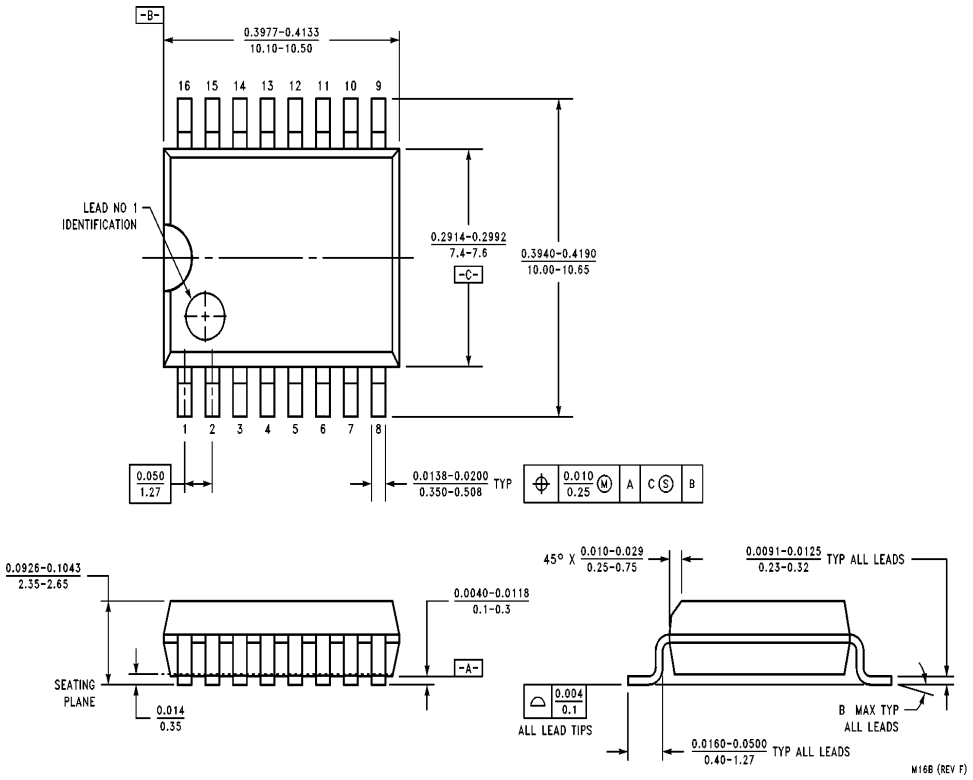
Figure 4-30 Drives Circuits for Fan, Compressor and Buzzer

Appendix A

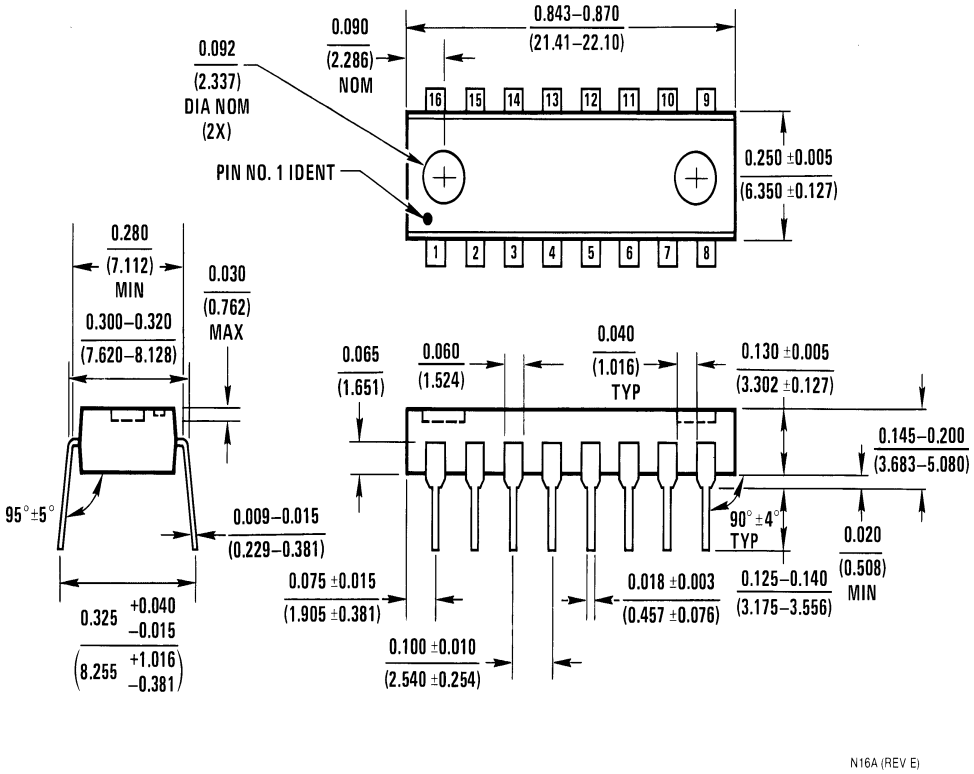
PHYSICAL DIMENSIONS



20-Lead Hermetic Dual-In-Line Packge, EPROM (D)
Order Number COP8SAC720Q9
See NS Package Number D20CQ

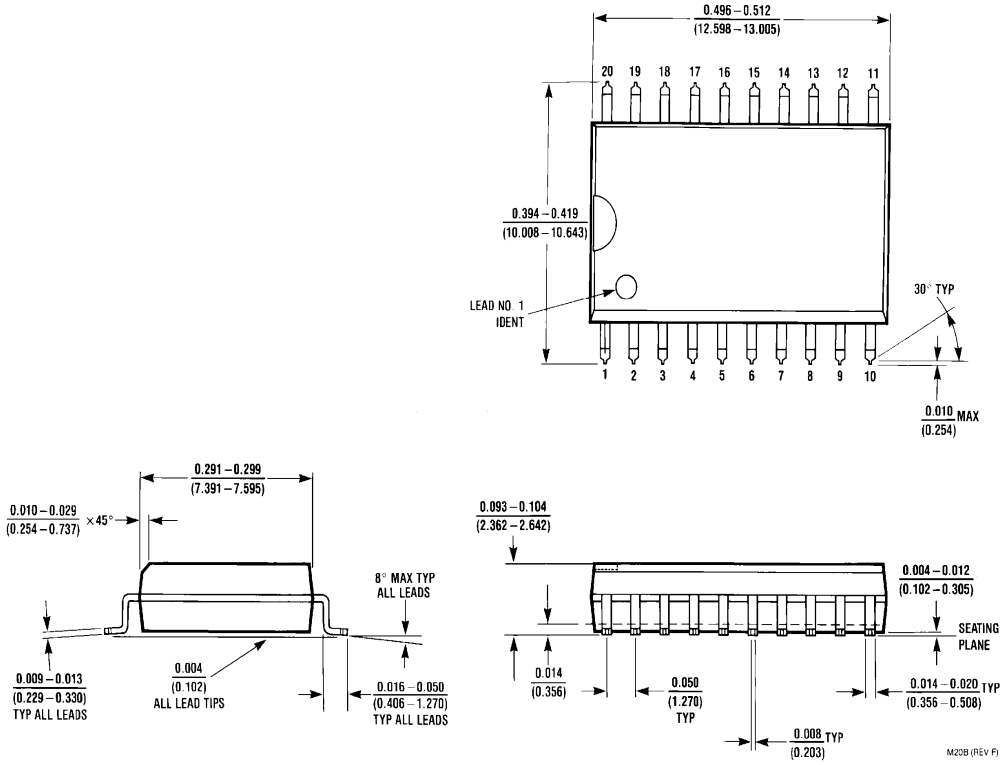


Molded Small Outline Package (WM)
Order Number COP8SAA716M8, or COP8SAA716M9
See NS Package Number M16B

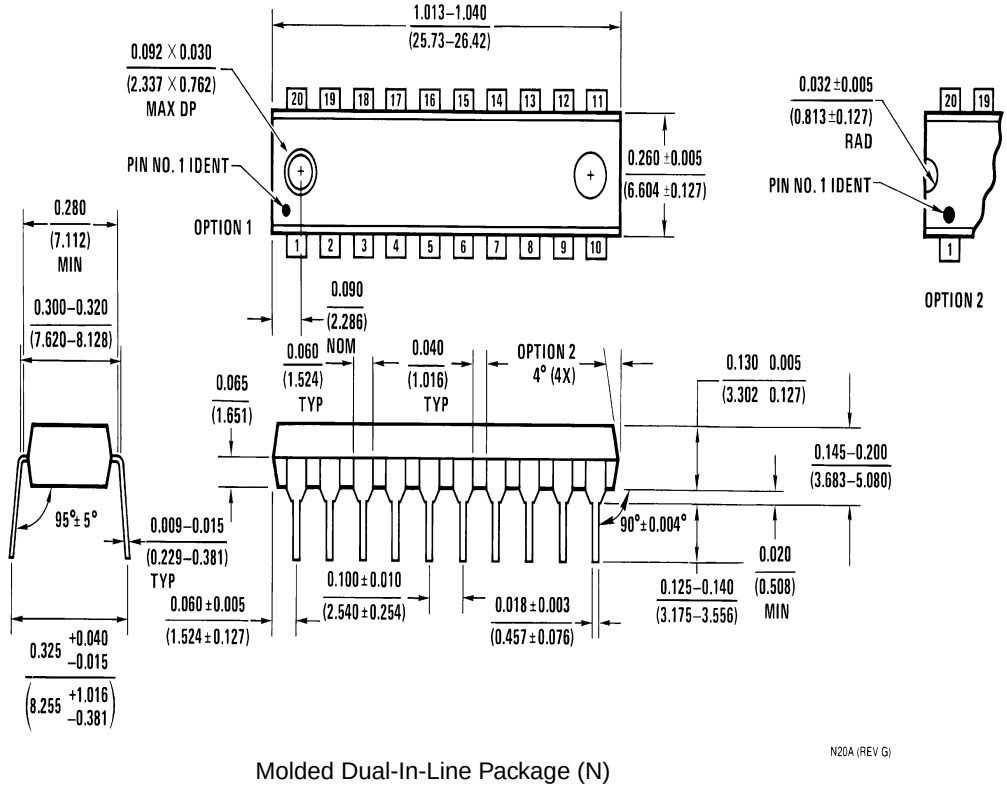


Molded Dual-In-Line Package (N)
Order Number COP8SAA716N8, or COP8SAA716N9
See NS Package Number N16A

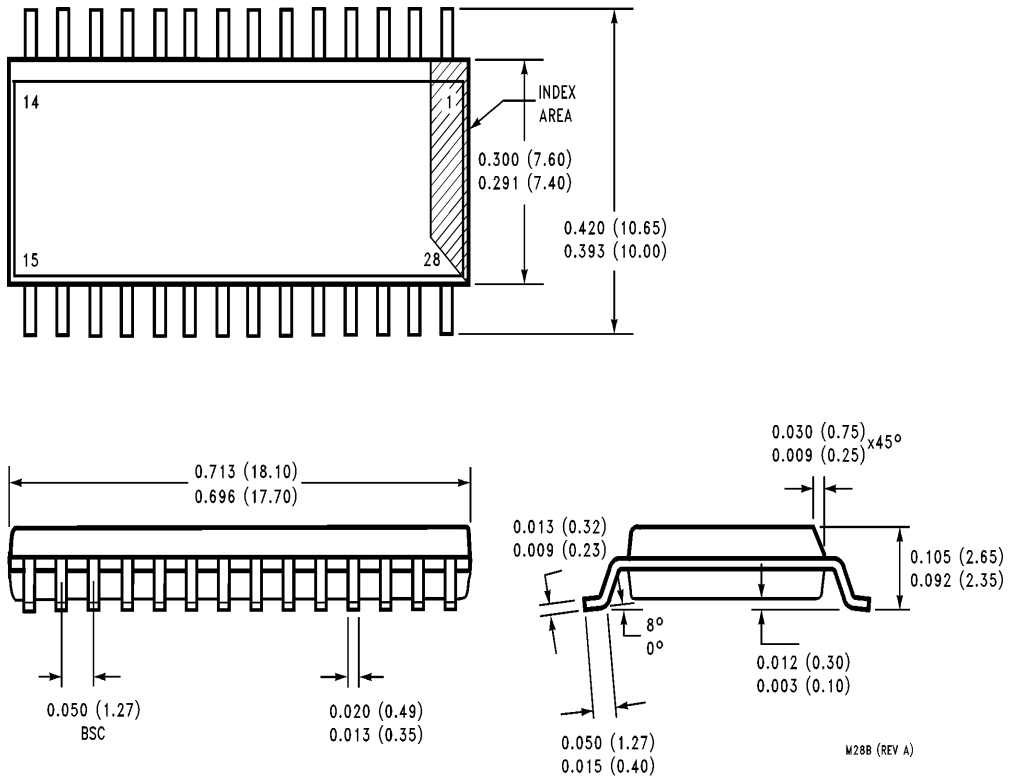
A-2 PHYSICAL DIMENSIONS



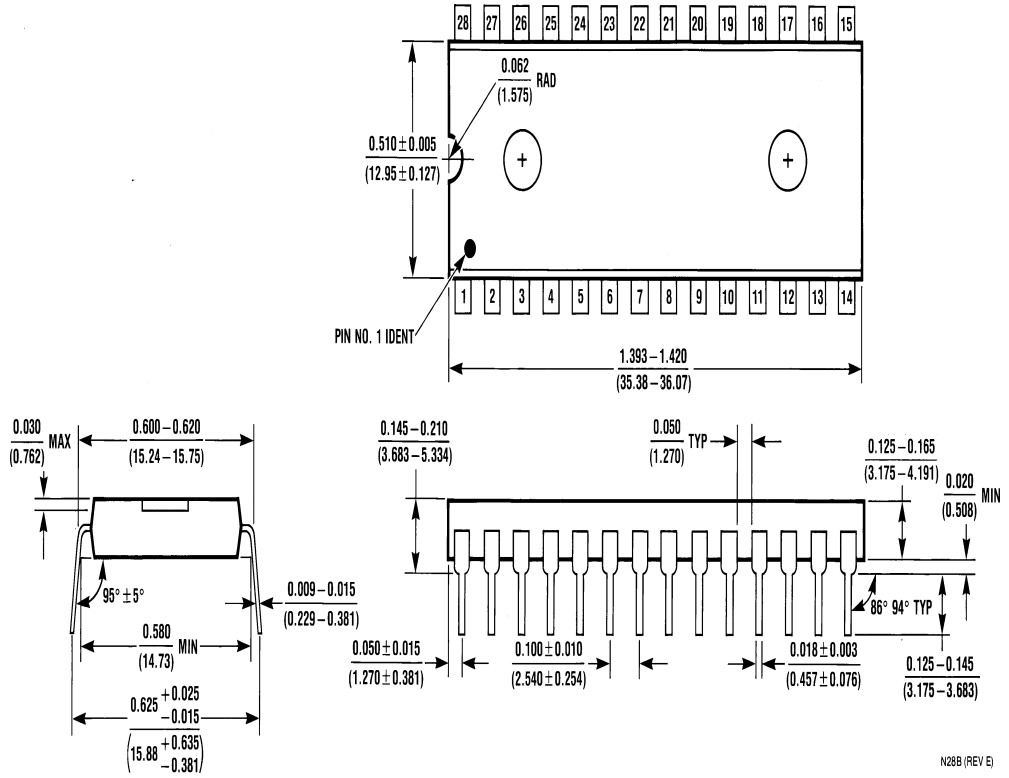
Molded SO Wide Body Package (WM)
Order Number COP8SAA720M9, COP8SAB720M9, COP8SAC720M9
COP8SAA720M8, COP8SAB720M8, or COP8SAC720M8
See NS Package Number M20B



Molded Dual-In-Line Package (N)
Order Number COP8SAA720N9, COP8SAB720N9, COP8SAC720N9,
COP8SAA720N8, COP8SAB720N8, or COP8SAC720N8,
See NS Package Number N20A

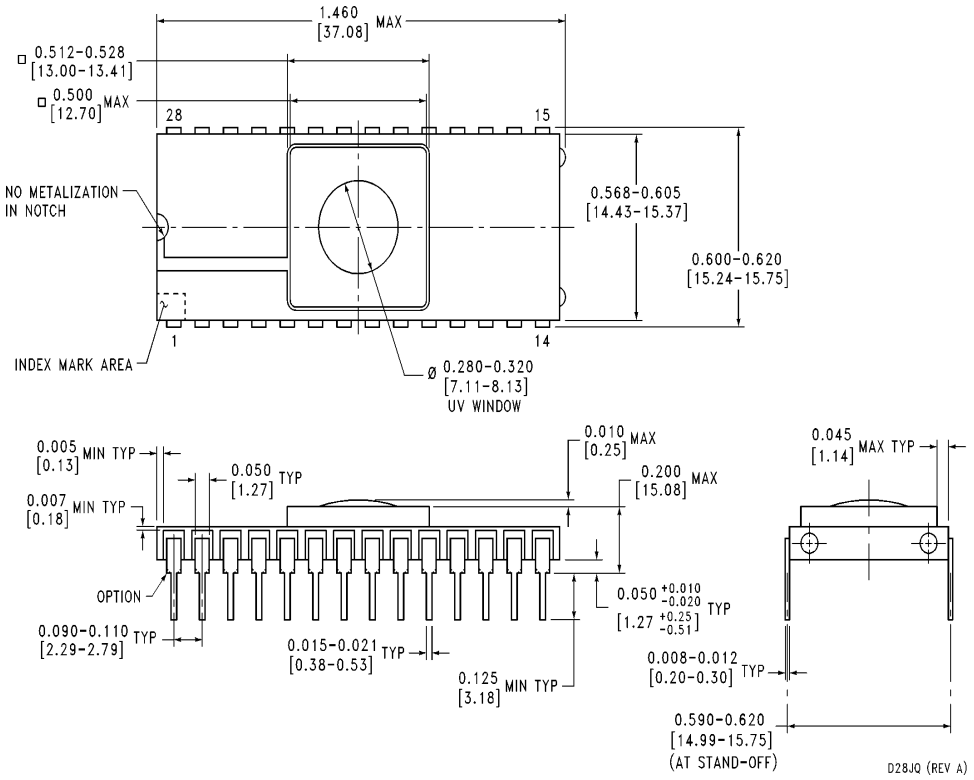


Molded SO Wide Body Package (WM)
Order Number COP8SAA728M9, COP8SAB728M9, COP8SAC728M9,
COP8SAA728M8, COP8SAB728M8, or COP8SAC728M8
See NS Package Number M28B

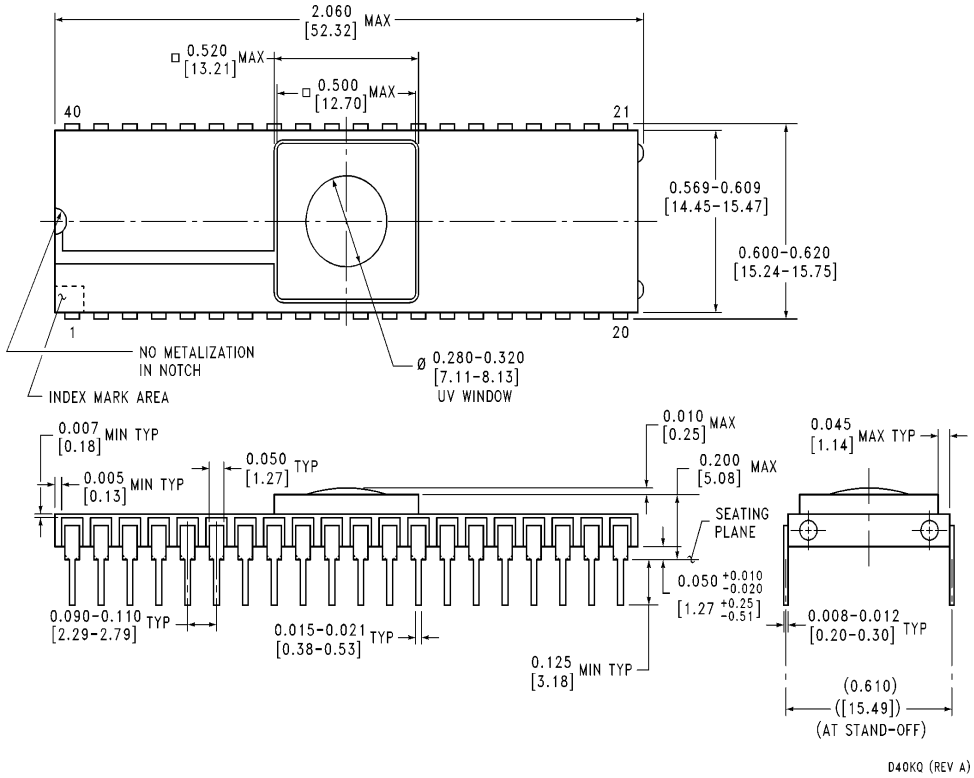


Molded Dual-In-Line Package (N)
Order Number COP8SAA728N9, COP8SAB728N9, COP8SAC728N9,
COP8SAA728N8, COP8SAB728N8, or COP8SAC728N8
See NS Package Number N28B

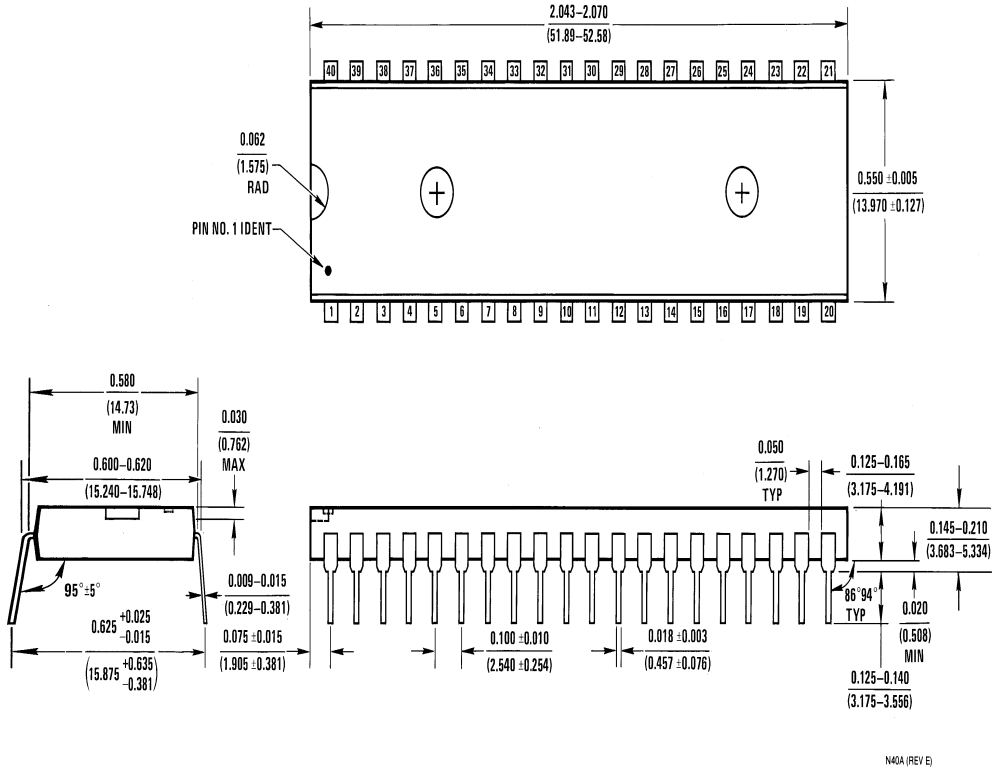
A-4 PHYSICAL DIMENSIONS



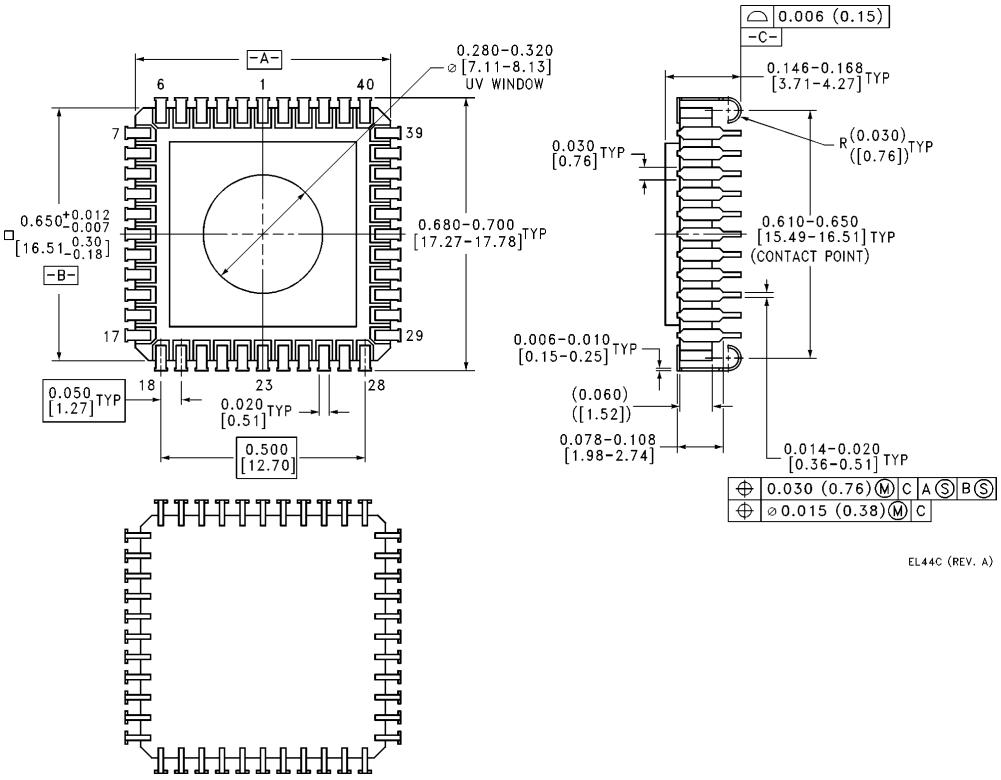
28-Lead Hermetic Dual-In-Line Package EPROM (D)
Order Number COP8SAC728Q9
See NS Package Number D28JQ



40-Lead Hermetic DIP EPROM (D)
Order Number COP8SAC740Q9
See NS Package Number D40KQ

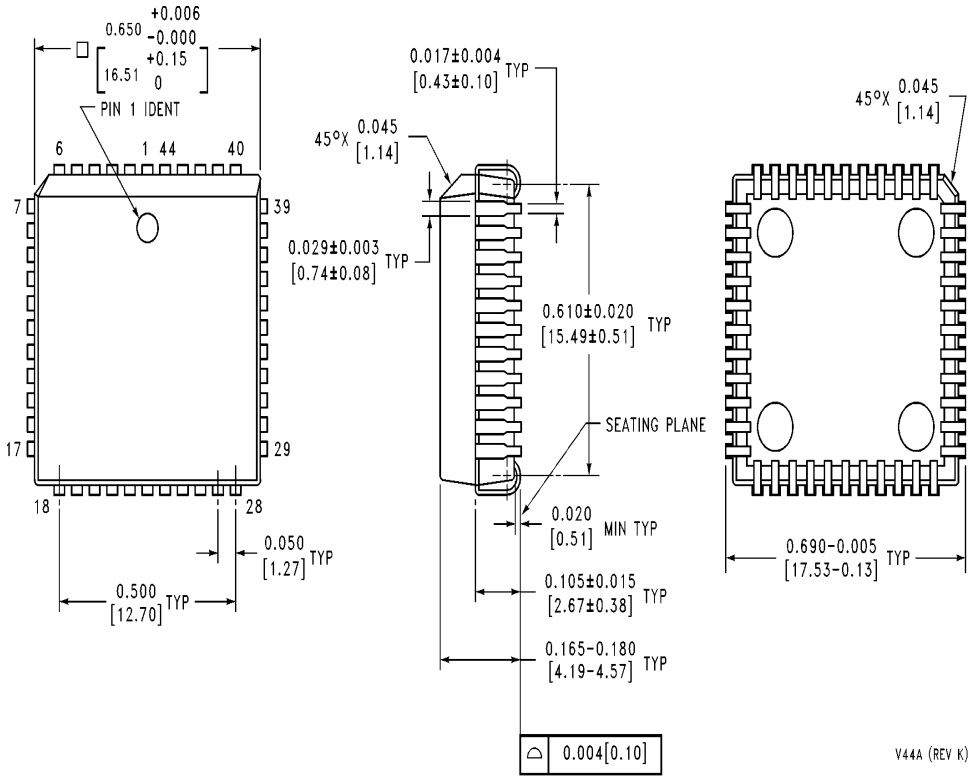


Molded Dual-In-Line Package (N)
Order Number COP8SAC740N9, COP8SAC740N8, or COP8SAC740N6
See NS Package Number N40A

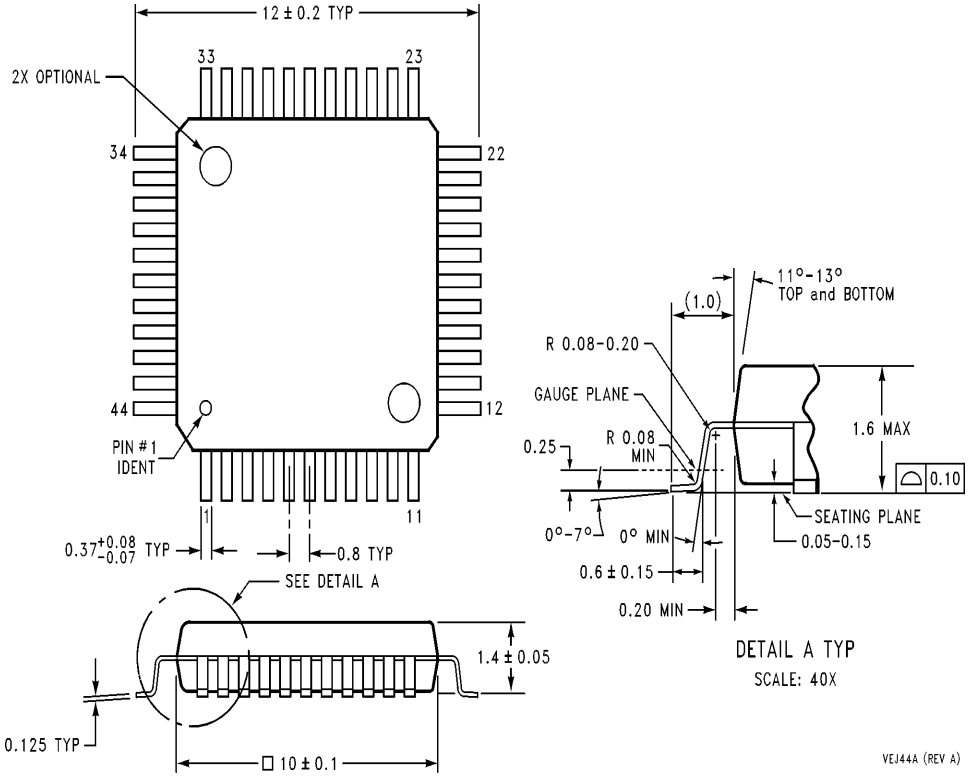


44-Lead EPROM Leaded Chip Carrier (EL)
Order Number COP8SAC744Q9
See NS Package Number EL44C

A-6 PHYSICAL DIMENSIONS



Molded Dual-In-Line Package (N)
Order Number COP8SAC744V9, COP8SAC744V8, or COP8SAC744V6
See NS Package Number V44A



Plastic Leaded Chip Carrier (V)
Order Number COP8SAC7VEJ9, or COP8SAC7VEJ8
See NS Package Number VEJ44A

INDEX