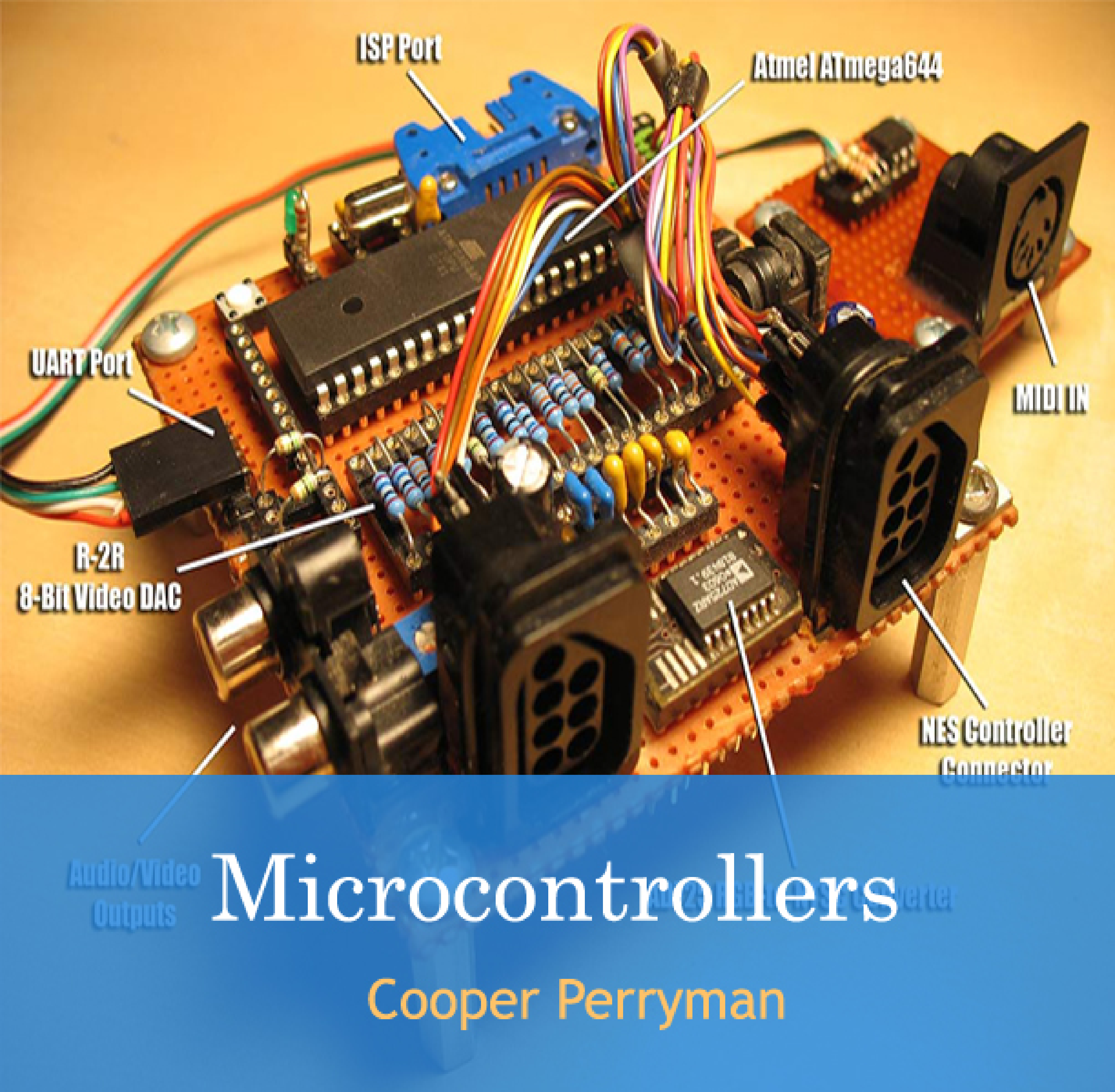


Microcontrollers

Cooper Perryman

First Edition, 2012

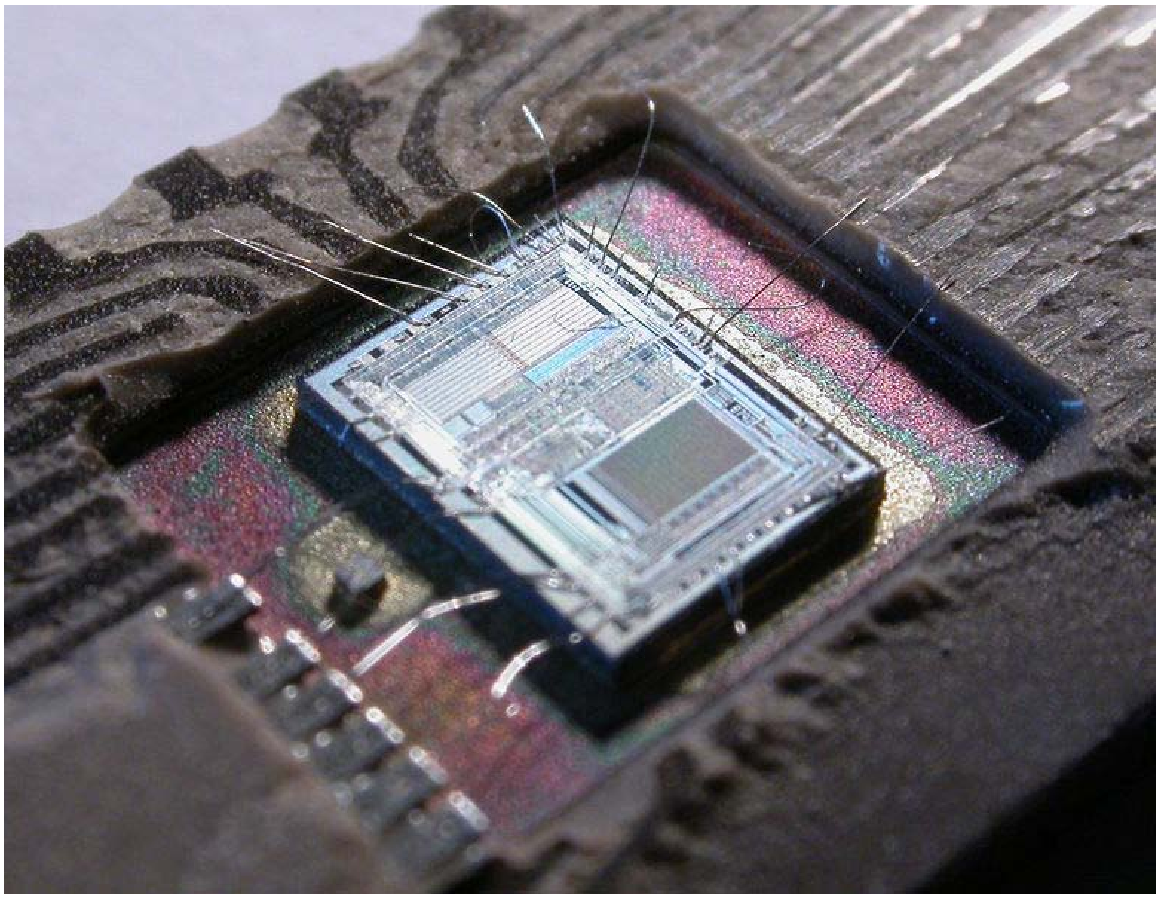
ISBN 978-81-323-3491-0

Published by:
Research World
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter-1

Microcontroller



The die from an Intel 8742, an 8-bit microcontroller that includes a CPU running at 12 MHz, 128 bytes of RAM, 2048 bytes of EPROM, and I/O in the same chip.

A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, and toys. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, for low power consumption (milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

Embedded design

A microcontroller can be considered a self-contained system with a processor, memory and peripherals and can be used as an embedded system. The majority of microcontrollers in use today are embedded in other machinery, such as automobiles, telephones, appliances, and peripherals for computer systems. These are called embedded systems. While some embedded systems are very sophisticated, many have minimal requirements for memory and program length, with no operating system, and low software complexity. Typical input and output devices include switches, relays, solenoids, LEDs, small or custom LCD displays, radio frequency devices, and sensors for data such as temperature, humidity, light level etc. Embedded systems usually have no keyboard, screen, disks, printers, or other recognizable I/O devices of a personal computer, and may lack human interaction devices of any kind.

Interrupts

Microcontrollers must provide real time (predictable, though not necessarily fast) response to events in the embedded system they are controlling. When certain events occur, an interrupt system can signal the processor to suspend processing the current instruction sequence and to begin an interrupt service routine (ISR, or "interrupt handler"). The ISR will perform any processing required based on the source of the interrupt before returning to the original instruction sequence. Possible interrupt sources are device dependent, and often include events such as an internal timer overflow, completing an analog to digital conversion, a logic level change on an input such as from a button being pressed, and data received on a communication link. Where power consumption is important as in battery operated devices, interrupts may also wake a microcontroller from a low power sleep state where the processor is halted until required to do something by a peripheral event.

Programs

Microcontroller programs must fit in the available on-chip program memory, since it would be costly to provide a system with external, expandable, memory. Compilers and assemblers are used to convert high-level language and assembler language codes into a compact machine code for storage in the microcontroller's memory. Depending on the device, the program memory may be permanent, read-only memory that can only be programmed at the factory, or program memory may be field-alterable flash or erasable read-only memory.

Other microcontroller features

Microcontrollers usually contain from several to dozens of general purpose input/output pins (GPIO). GPIO pins are software configurable to either an input or an output state. When GPIO pins are configured to an input state, they are often used to read sensors or external signals. Configured to the output state, GPIO pins can drive external devices such as LEDs or motors.

Many embedded systems need to read sensors that produce analog signals. This is the purpose of the analog-to-digital converter (ADC). Since processors are built to interpret and process digital data, i.e. 1s and 0s, they are not able to do anything with the analog signals that may be sent to it by a device. So the analog to digital converter is used to convert the incoming data into a form that the processor can recognize. A less common feature on some microcontrollers is a digital-to-analog converter (DAC) that allows the processor to output analog signals or voltage levels.

In addition to the converters, many embedded microprocessors include a variety of timers as well. One of the most common types of timers is the Programmable Interval Timer (PIT). A PIT may either count down from some value to zero, or up to the capacity of the count register, overflowing to zero. Once it reaches zero, it sends an interrupt to the processor indicating that it has finished counting. This is useful for devices such as thermostats, which periodically test the temperature around them to see if they need to turn the air conditioner on, the heater on, etc.

Time Processing Unit (TPU) is a sophisticated timer. In addition to counting down, the TPU can detect input events, generate output events, and perform other useful operations.

A dedicated Pulse Width Modulation (PWM) block makes it possible for the CPU to control power converters, resistive loads, motors, etc., without using lots of CPU resources in tight timer loops.

Universal Asynchronous Receiver/Transmitter (UART) block makes it possible to receive and transmit data over a serial line with very little load on the CPU. Dedicated on-chip hardware also often includes capabilities to communicate with other devices (chips) in digital formats such as I2C and Serial Peripheral Interface (SPI).

Higher integration

In contrast to general-purpose CPUs, micro-controllers may not implement an external address or data bus as they integrate RAM and non-volatile memory on the same chip as the CPU. Using fewer pins, the chip can be placed in a much smaller, cheaper package.

Integrating the memory and other peripherals on a single chip and testing them as a unit increases the cost of that chip, but often results in decreased net cost of the embedded system as a whole. Even if the cost of a CPU that has integrated peripherals is slightly more than the cost of a CPU and external peripherals, having fewer chips typically allows a smaller and cheaper circuit board, and reduces the labor required to assemble and test the circuit board.

A micro-controller is a single integrated circuit, commonly with the following features:

- central processing unit - ranging from small and simple 4-bit processors to complex 32- or 64-bit processors
- volatile memory (RAM) for data storage
- ROM, EPROM, EEPROM or Flash memory for program and operating parameter storage
- discrete input and output bits, allowing control or detection of the logic state of an individual package pin
- serial input/output such as serial ports (UARTs)
- other serial communications interfaces like I²C, Serial Peripheral Interface and Controller Area Network for system interconnect
- peripherals such as timers, event counters, PWM generators, and watchdog
- clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit
- many include analog-to-digital converters, some include digital-to-analog converters
- in-circuit programming and debugging support

This integration drastically reduces the number of chips and the amount of wiring and circuit board space that would be needed to produce equivalent systems using separate chips. Furthermore, on low pin count devices in particular, each pin may interface to several internal peripherals, with the pin function selected by software. This allows a part to be used in a wider variety of applications than if pins had dedicated functions. Micro-controllers have proved to be highly popular in embedded systems since their introduction in the 1970s.

Some microcontrollers use a Harvard architecture: separate memory buses for instructions and data, allowing accesses to take place concurrently. Where a Harvard architecture is used, instruction words for the processor may be a different bit size than the length of internal memory and registers; for example: 12-bit instructions used with 8-bit data registers.

The decision of which peripheral to integrate is often difficult. The microcontroller vendors often trade operating frequencies and system design flexibility against time-to-market requirements from their customers and overall lower system cost. Manufacturers have to balance the need to minimize the chip size against additional functionality.

Microcontroller architectures vary widely. Some designs include general-purpose microprocessor cores, with one or more ROM, RAM, or I/O functions integrated onto the package. Other designs are purpose built for control applications. A micro-controller instruction set usually has many instructions intended for bit-wise operations to make control programs more compact. For example, a general purpose processor might require several instructions to test a bit in a register and branch if the bit is set, where a micro-controller could have a single instruction to provide that commonly-required function.

Microcontrollers typically do not have a math coprocessor, so floating point arithmetic is performed by software.

Volumes

About 55% of all CPUs sold in the world are 8-bit microcontrollers and microprocessors. According to Semico, over four billion 8-bit microcontrollers were sold in 2006.

A typical home in a developed country is likely to have only four general-purpose microprocessors but around three dozen microcontrollers. A typical mid-range automobile has as many as 30 or more microcontrollers. They can also be found in many electrical devices such as washing machines, microwave ovens, and telephones.

A PIC 18F8720 **microcontroller** in an 80-pin TQFP package.

Manufacturers have often produced special versions of their microcontrollers in order to help the hardware and software development of the target system. Originally these included EPROM versions that have a "window" on the top of the device through which program memory can be erased by ultraviolet light, ready for reprogramming after a programming ("burn") and test cycle. Since 1998, EPROM versions are rare and have been replaced by EEPROM and flash, which are easier to use (can be erased electronically) and cheaper to manufacture.

Other versions may be available where the ROM is accessed as an external device rather than as internal memory, however these are becoming increasingly rare due to the widespread availability of cheap microcontroller programmers.

The use of field-programmable devices on a microcontroller may allow field update of the firmware or permit late factory revisions to products that have been assembled but not yet shipped. Programmable memory also reduces the lead time required for deployment of a new product.

Where hundreds of thousands of identical devices are required, using parts programmed at the time of manufacture can be an economical option. These "mask programmed" parts have the program laid down in the same way as the logic of the chip, at the same time.

Programming environments

Microcontrollers were originally programmed only in assembly language, but various high-level programming languages are now also in common use to target microcontrollers. These languages are either designed specially for the purpose, or versions of general purpose languages such as the C programming language. Compilers for general purpose languages will typically have some restrictions as well as enhancements to better support the unique characteristics of microcontrollers. Some microcontrollers have environments to aid developing certain types of applications. Microcontroller vendors often make tools freely available to make it easier to adopt their hardware.

Many microcontrollers are so quirky that they effectively require their own non-standard dialects of C, such as SDCC for the 8051, which prevent using standard tools (such as code libraries or static analysis tools) even for code unrelated to hardware features. Interpreters are often used to hide such low level quirks.

Interpreter firmware is also available for some microcontrollers. For example, BASIC on the early microcontrollers Intel 8052; BASIC and FORTH on the Zilog Z8 as well as some modern devices. Typically these interpreters support interactive programming.

Simulators are available for some microcontrollers, such as in Microchip's MPLAB environment and the Revolution Education PICAXE range. These allow a developer to analyze what the behavior of the microcontroller and their program should be if they were using the actual part. A simulator will show the internal processor state and also that of the outputs, as well as allowing input signals to be generated. While on the one hand most simulators will be limited from being unable to simulate much other hardware in a system, they can exercise conditions that may otherwise be hard to reproduce at will in the physical implementation, and can be the quickest way to debug and analyze problems.

Recent microcontrollers are often integrated with on-chip debug circuitry that when accessed by an in-circuit emulator via JTAG, allow debugging of the firmware with a debugger.

Types of microcontrollers

As of 2008 there are several dozen microcontroller architectures and vendors including:

- Parallax Propeller
- Freescale 68HC11 (8-bit)
- Intel 8051
- Silicon Laboratories Pipelined 8051 Microcontrollers
- ARM processors (from many vendors) using ARM7 or Cortex-M3 cores are generally microcontrollers
- STMicroelectronics STM8 (8-bit), ST10 (16-bit) and STM32 (32-bit)

- Atmel AVR (8-bit), AVR32 (32-bit), and AT91SAM (32-bit)
- Freescale ColdFire (32-bit) and S08 (8-bit)
- Hitachi H8, Hitachi SuperH (32-bit)
- Hyperstone E1/E2 (32-bit, First full integration of RISC and DSP on one processor core [1996])
- Infineon Microcontroller: 8, 16, 32 Bit microcontrollers for automotive and industrial applications
- MIPS (32-bit PIC32)
- NEC V850 (32-bit)
- PIC (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24)
- PowerPC ISE
- PSoC (Programmable System-on-Chip)
- Rabbit 2000 (8-bit)
- Texas Instruments Microcontrollers MSP430 (16-bit), C2000 (32-bit), and Stellaris (32-bit)
- Toshiba TLCS-870 (8-bit/16-bit)
- XMOS XCore XS1 (32-bit)
- Zilog eZ8 (16-bit), eZ80 (8-bit)

and many others, some of which are used in very narrow range of applications or are more like applications processors than microcontrollers. The microcontroller market is extremely fragmented, with numerous vendors, technologies, and markets. Note that many vendors sell (or have sold) multiple architectures.

Interrupt latency

In contrast to general-purpose computers, microcontrollers used in embedded systems often seek to optimize interrupt latency over instruction throughput. Issues include both reducing the latency, and making it be more predictable (to support real-time control).

When an electronic device causes an interrupt, the intermediate results (registers) have to be saved before the software responsible for handling the interrupt can run. They must also be restored after that software is finished. If there are more registers, this saving and restoring process takes more time, increasing the latency. Ways to reduce such context/restore latency include having relatively few registers in their central processing units (undesirable because it slows down most non-interrupt processing substantially), or at least having the hardware not save them all (this fails if the software then needs to compensate by saving the rest "manually"). Another technique involves spending silicon gates on "shadow registers": one or more duplicate registers used only by the interrupt software, perhaps supporting a dedicated stack.

Other factors affecting interrupt latency include:

- Cycles needed to complete current CPU activities. To minimize those costs, microcontrollers tend to have short pipelines (often three instructions or less), small write buffers, and ensure that longer instructions are continuable or restartable. RISC design principles ensure that most instructions take the same number of cycles, helping avoid the need for most such continuation/restart logic.

- The length of any critical section that needs to be interrupted. Entry to a critical section restricts concurrent data structure access. When a data structure must be accessed by an interrupt handler, the critical section must block that interrupt. Accordingly, interrupt latency is increased by however long that interrupt is blocked. When there are hard external constraints on system latency, developers often need tools to measure interrupt latencies and track down which critical sections cause slowdowns.
    - One common technique just blocks all interrupts for the duration of the critical section. This is easy to implement, but sometimes critical sections get uncomfortably long.
    - A more complex technique just blocks the interrupts that may trigger access to that data structure. This often based on interrupt priorities, which tend to not correspond well to the relevant system data structures. Accordingly, this technique is used mostly in very constrained environments.
    - Processors may have hardware support for some critical sections. Examples include supporting atomic access to bits or bytes within a word, or other atomic access primitives like the LDREX/STREX exclusive access primitives introduced in the ARMv6 architecture.
- Interrupt nesting. Some microcontrollers allow higher priority interrupts to interrupt lower priority ones. This allows software to manage latency by giving time-critical interrupts higher priority (and thus lower and more predictable latency) than less-critical ones.
- Trigger rate. When interrupts occur back-to-back, microcontrollers may avoid an extra context save/restore cycle by a form of tail call optimization.

Lower end microcontrollers tend to support fewer interrupt latency controls than higher end ones.

History

The first single-chip microprocessor was the 4-bit Intel 4004 released in 1971. With the Intel 8008 and more capable microprocessors available over the next several years.

These however all required external chip(s) to implement a working system, raising total system cost, and making it impossible to economically computerize appliances.

The first computer system on a chip optimized for control applications was the Intel 8048, released in 1975, with both RAM and ROM on the same chip. This chip would find its way into over one billion PC keyboards, and other numerous applications. At this time Intels President, Luke J. Valenter, stated that the (Microcontroller) was one of the most successful in the companies history, and expanded the division's budget over 25%.

Most microcontrollers at this time had two variants. One had an erasable EPROM program memory, which was significantly more expensive than the PROM variant which was only programmable once.

In 1993, the introduction of EEPROM memory allowed microcontrollers (beginning with the Microchip PIC16x84)) to be electrically erased quickly without an expensive package as required for EPROM, allowing both rapid prototyping, and In System Programming.

The same year, Atmel introduced the first microcontroller using Flash memory.

Other companies rapidly followed suit, with both memory types.

Cost has plummeted over time, with the cheapest 8-bit microcontrollers being available for under $0.25 in quantity (thousands) in 2009, and some 32-bit microcontrollers around $1 for similar quantities.

Nowadays microcontrollers are low cost and readily available for hobbyists, with large online communities around certain processors.

In the future, MRAM could potentially be used in microcontrollers as it has infinite endurance and its incremental semiconductor wafer process cost is relatively low.

Microcontroller embedded memory technology

Since the emergence of microcontrollers, many different memory technologies have been used. Almost all microcontrollers have at least two different kinds of memory, a non-volatile memory for storing firmware and a read-write memory for temporary data.

Data

From the earliest microcontrollers to today, six-transistor SRAM is almost always used as the read/write working memory, with a few more transistors per bit used in the register file. MRAM could potentially replace it as it is 4-10 times denser which would make it more cost effective.

In addition to the SRAM, some microcontrollers also have internal EEPROM for data storage; and even ones that do not have any (or not enough) are often connected to external serial EEPROM chip (such as the BASIC Stamp) or external serial flash memory chip.

A few recent microcontrollers beginning in 2003 have "self-programmable" flash memory.

Firmware

The earliest microcontrollers used hard-wired or mask ROM to store firmware. Later microcontrollers (such as the early versions of the Freescale 68HC11 and early PIC microcontrollers) had quartz windows that allowed ultraviolet light in to erase the EPROM.

The Microchip PIC16C84, introduced in 1993, was the first microcontroller to use EEPROM to store firmware

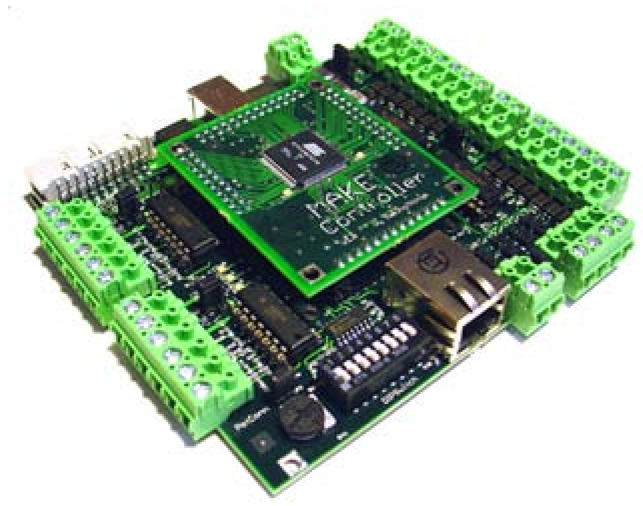
Also in 1993, Atmel introduced the first microcontroller using NOR Flash memory to store firmware.

PSoC microcontrollers, introduced in 2002, store firmware in SONOS flash memory.

MRAM could potentially be used to store firmware.

Chapter-2

Single-board Microcontroller



The Make Controller Kit

A **single-board microcontroller** is a microcontroller pre-built onto a single printed circuit board. This board provides all of the circuitry necessary for a useful control task: microprocessor, I/O circuits, clock generator, RAM, stored program memory and any support ICs necessary. The intention is that the board is immediately useful to an application developer, without needing to spend time and effort in developing the controller board.

As they are usually low-cost hardware, and have an especially low capital cost when starting out with their development, single-board microcontrollers have long been popular in education. They are also a popular means for experienced microcontroller developers to gain hands-on experience with a new processor family.

Origins of the single-board microcontroller

Single-board microcontrollers appeared in the late 1970s when the first generations of microprocessors, such as the 6502 and the Z80, made it practical to build an entire controller on a single board, and affordable to dedicate a processor chip to such a relatively minor task.

Processors of this era required a number of support chips in addition. RAM and EPROM were separate, often requiring memory management or refresh circuitry for dynamic memory as well. I/O processing might be carried out by a single chip such as the 8255, but frequently required several more chips.

The difference between a single-board microcontroller and a single-board computer is that the microcontroller has no aspirations beyond being a controller. The computer will have some facility for a user interface and bulk storage peripherals that the controller does not. In the 1970s these extra features were sparse and there was often little to distinguish the two roles. Some microcontrollers such as the KIM-1 / AIM-65 began as microcontrollers and were expanded over time and with additional boards to a system more like a microcomputer.

Compared to a microprocessor development board, the purpose of the single board microcontroller is still to be a controller first and foremost. The development board exists to showcase or to train on some particular processor family and this internal implementation is more important than the external function. Obviously there is a large overlap between all three systems.

Internal bus

The bus was universally a Von Neumann architecture, with program and data memory accessed by the same shared bus, even though they were frequently stored in fundamentally different types of memory: ROM for programs and RAM for data. This bus architecture was chosen to economise on the number of pins needed from the limited 40 available for the processor's ubiquitous dual-in-line IC package. When single-chip microcontrollers became available later on, the bus no longer needed to be exposed outside the package and so the Harvard architecture of separate program and data buses (both internal to the chip) became popular.

It was common to offer the internal bus through an expansion connector, or at least the space for such a connector to be soldered on. This was a low-cost option and offered the potential for expansion, even if it was rarely made use of. Typical expansions would be I/O devices, or memory expansion. It was unusual to add peripheral devices such as tape or disk storage, or even a CRT display.

I/O

I/O features on microcontrollers would comprise a few bits of bitwise digital I/O. The signal levels were low-powered and un-isolated, so additional signal conditioning or power outputs would be needed if real-world hardware was to be controlled. Rarely an analogue port (usually input only) might also be found. To control component costs, many boards were designed with extra hardware interface circuits but the components for these circuits weren't installed and the board was left bare. The circuit was only added as an option on delivery, or could be populated later.

It was common practice for boards to include "prototyping areas", areas of the board already laid out as a solderable breadboard area with the bus and power rails available, but without a defined circuit. Several controllers, particularly those intended for training, also included a pluggable re-usable breadboard for easy prototyping of extra I/O circuits that could be changed or removed for later projects.

Communications and user interfaces

Communications interfaces were fairly uncommon, although several boards supported a serial port. This could either be used to integrate with a host computer when running, or just to download programs when programming.

Microcomputers of this period usually supported the Kansas City or CUTS tape interface and a few even had the ability to use floppy disk storage. There could also be user interface connections for keyboards, CRT screen displays or terminals. These were not found on microcontrollers.

Programming

Many of the earliest systems had no internal facility for programming at all, and relied on a separate "host" system. This programming was typically in assembly language, sometimes C or even PL/M, and then cross-assembled or cross-compiled on the host.

EPROM burning

The completed object code from the host system (usually in Intel HEX format) would then be "burned" onto an EPROM with an EPROM programmer, this EPROM then being physically plugged into the board. As the EPROM would be removed and replaced many times during program development, it was usual to provide a ZIF socket to avoid wear or damage. As "washing" an EPROM with a UV eraser takes a considerable time, it was also usual for a developer to have several EPROMs in circulation at any one time.

Keypad monitors

KIM-1 6502-based single-board computer (1975)

Where the single-board controller formed the entire development environment (typically in education) the board might also be provided with a simple hexadecimal keypad, calculator-style LED display and a "monitor" program set permanently in ROM. This monitor allowed machine code programs to be entered directly through the keyboard and held in RAM. These programs were in machine code, not even in assembly language, and were assembled by hand on paper first. It's arguable as to which process was more time-consuming and error prone: assembling by hand, or keying byte-by-byte.

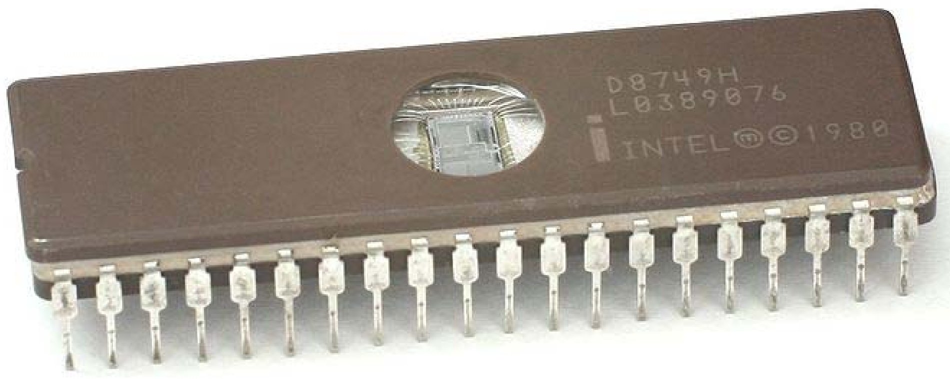
Single-board "keypad and calculator display" microcontrollers of this type were very similar to some low-end microcomputers of the time, such as the KIM-1 or the Microprofessor I. Some of these microprocessor "trainer" systems are still in production today, as a very low-cost introduction to microprocessors at the hardware programming level.

Hosted development

When the cheap desktop PC appeared in the mid-1980s, there was a shift to hosted development but without the need for EPROM burning. Hardware was now cheaper and RAM capacity had expanded such that it was possible to download the program through the serial port and hold it in RAM. This massive reduction in the cycle time to test a new version of a program gave an equally large boost in development speed.

This program memory was still volatile and would be lost if power was turned off. Flash memory was not yet available at a viable price. As a completed controller project usually required to be non-volatile, the final step in a project was often to burn an EPROM again.

Single-chip microcontrollers



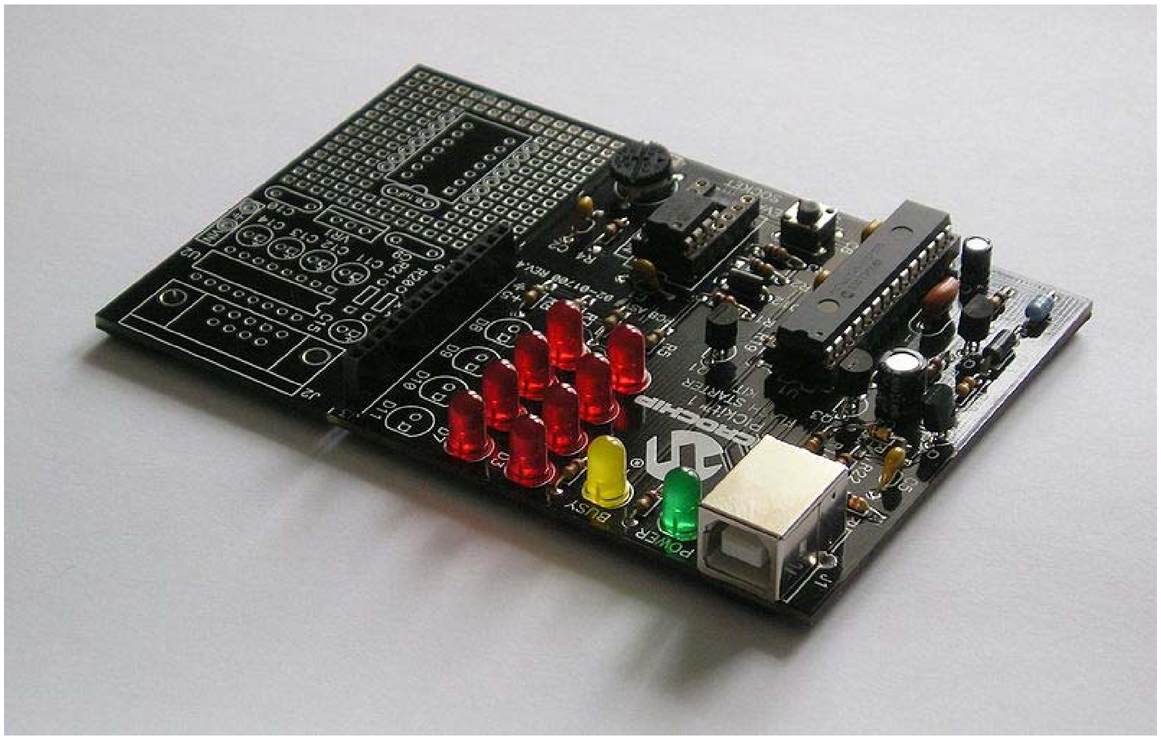
A 8048-family microcontroller with on-board UV EPROM, the 8749

With the development of single-chip microcontrollers such as the 8748, it became possible to combine most of the features of the previous board into a single IC package. Often only the clock generator remained separate, as it remained easier to use a quartz crystal oscillator than to integrate a stable clock circuit into a low-cost IC.

Single-chip microcontrollers integrate their necessary memory (both RAM and ROM) on-package and so do not need to expose their bus through the IC package's pins. These pins are thus freed-up for I/O lines. Both of these changes make the design of a single-board microcontroller simpler, but also less necessary. Single-board development and training systems remained popular, but there was now less need to provide single-board systems as embeddable components for production systems.

Program memory

For production use as embedded systems, the on-board ROM would be either mask programmed at the chip factory or one-time programmed (OTP) by the developer as a PROM. PROMs often used the same UV EPROM technology for the chip, but in a cheaper package without the transparent erasure window. During program development it was still necessary to burn EPROMs, this time the entire controller IC, and so ZIF sockets would be provided.



A development board for a PIC family device

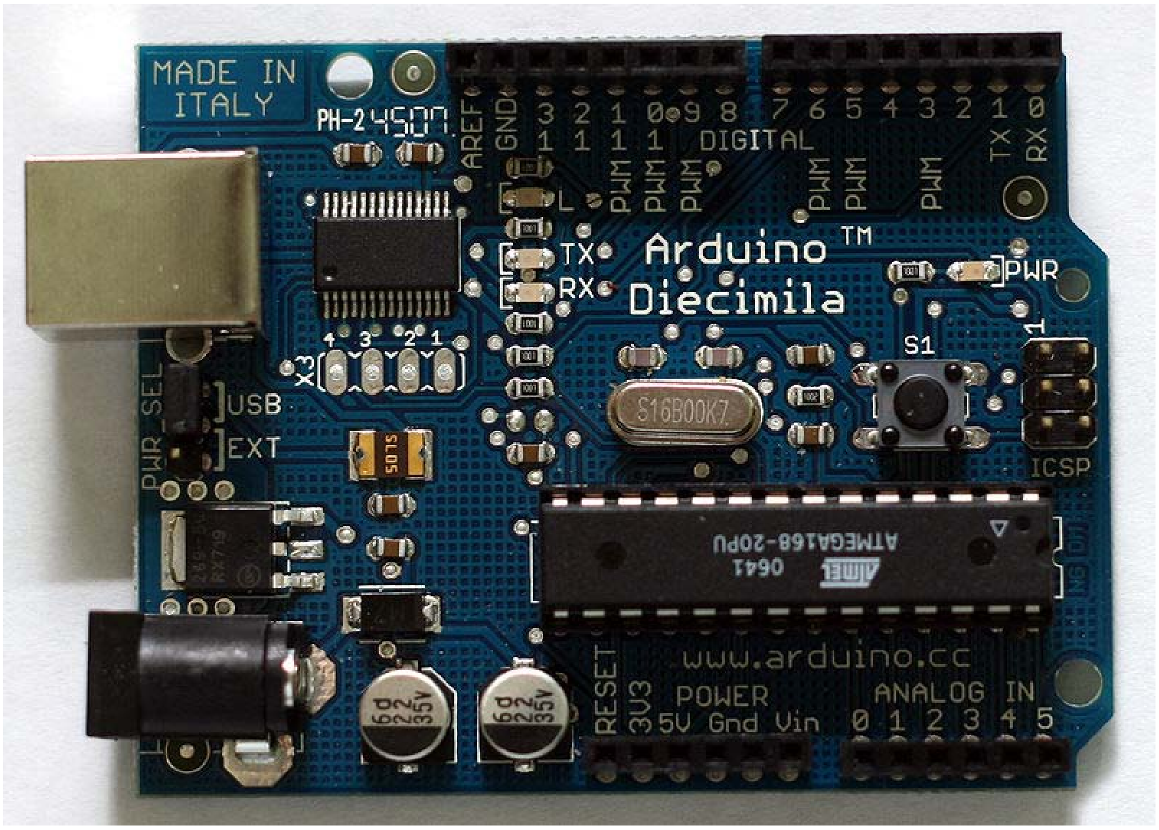
With the development of affordable EEPROM, EAROM and eventually flash memory, it became practical to attach the controller permanently to the board and to download program code to it through a serial connection to a host computer. This was termed "in-circuit programming". Erasure of old programs was carried out by either over-writing them with a new download, or bulk erasing them electrically (for EEPROM) which was slower, but could be carried out in-situ.

The main function of the controller board was now to carry the support circuits for this serial interface, or USB on later boards. As a further convenience feature during development, many boards also carried low-cost features like LED monitors of the I/O lines or reset switches mounted on-board.

- 8748

- PIC
- Atmel AVR

Single-board microcontrollers today



Arduino Diecimila

Dwengo board

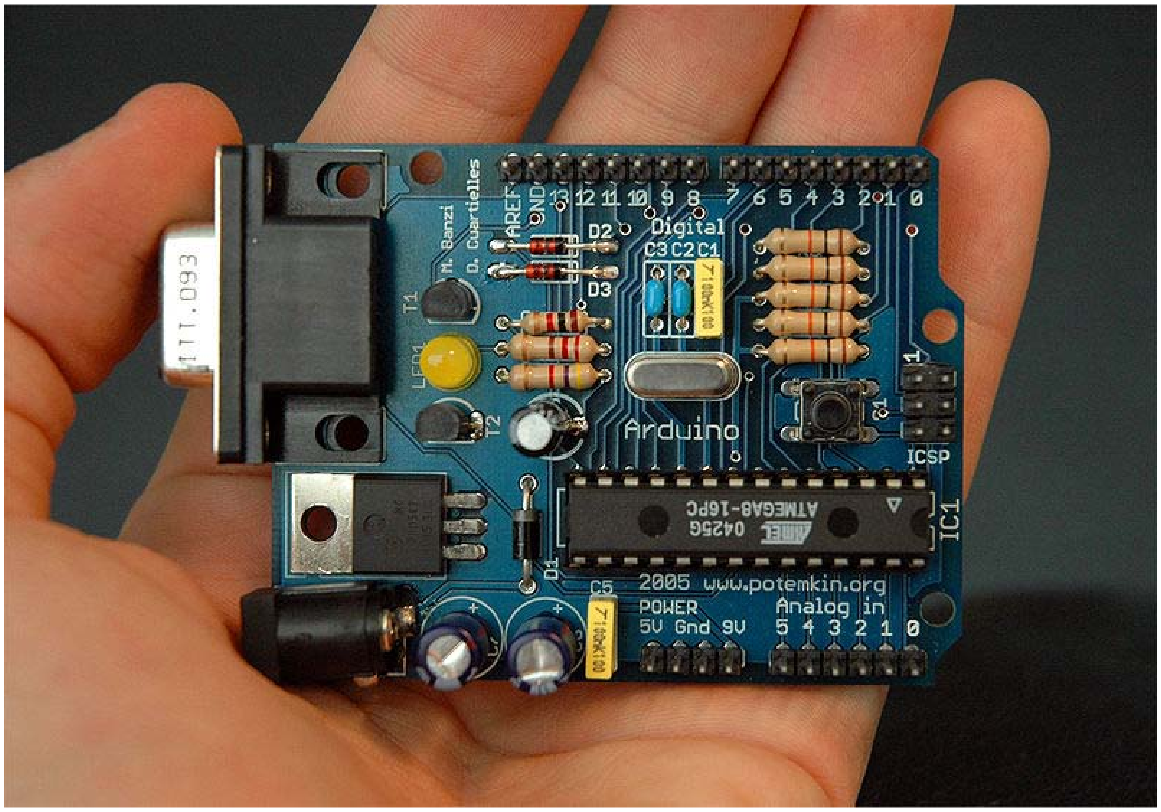
Microcontrollers are now cheap and simple to design circuit boards for. Development host systems are also cheap, especially when using open source software. Higher level programming languages abstract details of the hardware, making differences between specific processors less obvious to the application programmer. Rewritable flash memory has replaced slow programming cycles, at least during program development. Accordingly almost all development now is based on cross-compilation from personal computers and download to the controller board through a serial-like interface, usually appearing to the host as a USB device.

The original market demand of a simplified board implementation is no longer so relevant to microcontrollers. Single-board microcontrollers are still important, but have shifted their focus to:

- Easily accessible platforms aimed at traditionally "non-programmer" groups, such as artists. These controllers may be embedded to form part of a physical computing project. Popular choices for this work are the Arduino, Dwengo or the Wiring project.
- Technology demonstrator boards for innovative processors or peripheral features:
    - AVR Butterfly
    - Parallax Propeller

Chapter-3

Arduino



Arduino compared to a human hand

Arduino is an open-source single-board microcontroller, designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language and the boot loader that runs on the board.

Arduino hardware is programmed using a Wiring-based language (syntax + libraries), similar to C++ with some simplifications and modifications, and a Processing-based IDE.

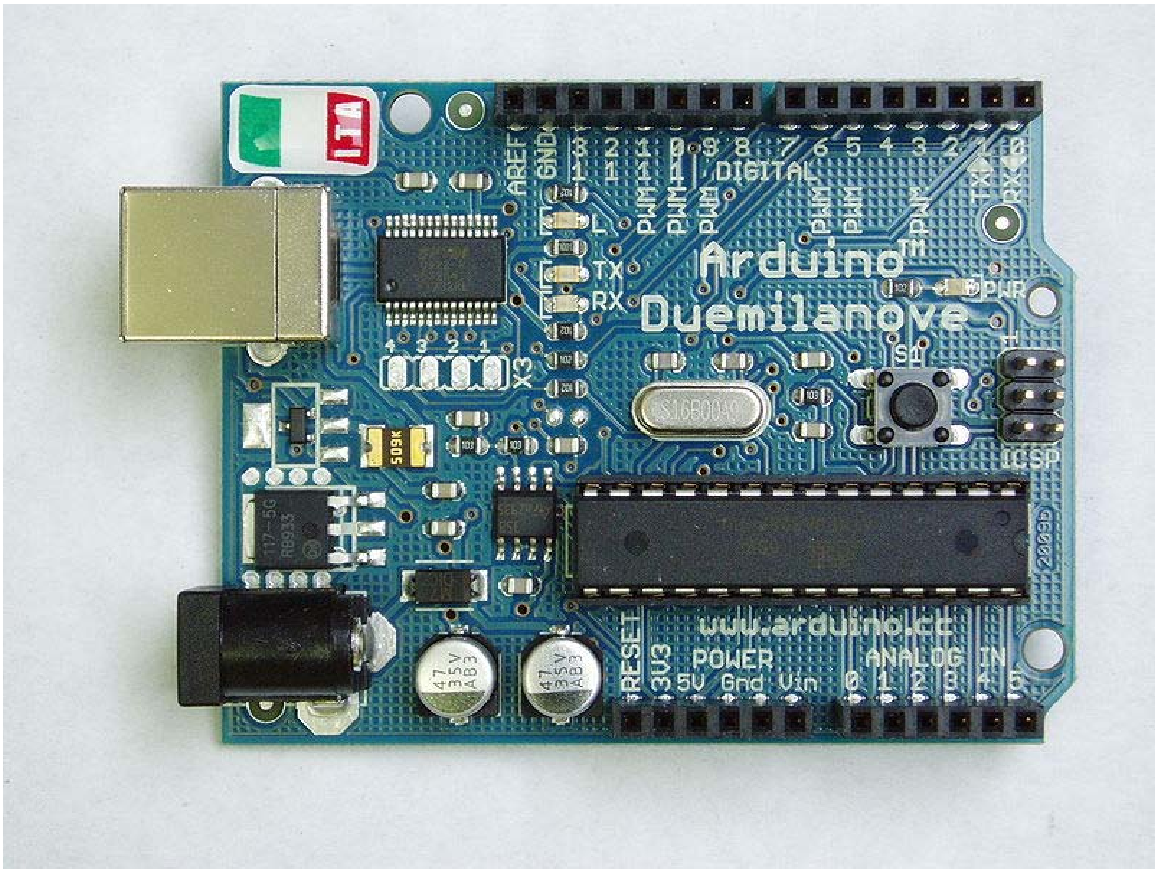
Currently shipping versions can be purchased pre-assembled; hardware design information is available for those who would like to assemble an Arduino by hand. Additionally, variations of the Italian-made Arduino—with varying levels of compatibility—have been released by third parties.

The Arduino project received an honorary mention in the Digital Communities category at the 2006 Prix Ars Electronica.

The project began in Ivrea, Italy in 2005 to make a device for controlling student-built interaction design projects less expensively than other prototyping systems available at the time. As of February 2010 more than 120,000 Arduino boards had been shipped. Founders Massimo Banzi and David Cuartielles named the project after a local bar named Arduino. The name is an Italian masculine first name, meaning "strong friend". The English pronunciation is "Hardwin", a namesake of Arduino of Ivrea.

Platform

Hardware



An official Arduino Duemilanove (rev 2009b).

An Arduino board consists of an 8-bit Atmel AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits. An important aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules (known as shields). Official Arduinos have used the megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, and ATmega1280. A handful of other processors have been used by Arduino compatibles. Most boards include a 5 volt linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a bootloader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external chip programmer.

At a conceptual level, when using the Arduino software stack, all boards are programmed over an RS-232 serial connection, but the way this is implemented varies by hardware version. Serial Arduino boards contain a simple inverter circuit to convert between RS-232-level and TTL-level signals. Current Arduino boards are programmed via USB, implemented using USB-to-serial adapter chips such as the FTDI FT232. Some variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods. (When used with traditional microcontroller tools instead of the Arduino IDE, standard AVR ISP programming is used.)

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, now superseded by the Duemilanove, for example, provides 14 digital I/O pins, six of which can produce PWM signals, and six analog inputs. These pins are on the top of the board, via female 0.1 inch headers. Several plug-in application "shields" are also commercially available.

The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards provide male header pins on the underside of the board to be plugged into solderless breadboards.

Sortable table

Arduino	**Processor**	**Flash KiB**	**EEPROM KiB**	**SRAM KiB**	**Digital I/O pins**	**...with PWM**	**Analog input pins**	**Dimensions (inches)**	**Dimensions (mm)**
Diecimila	ATmega168	16	0.5	1	14	6	6	2.7"x2.1"	68.6mmx53.3 mm
Duemilanove	ATmega168/328	16	0.5	1	14	6	6	2.7"x2.1"	68.6mmx53.3 mm
Uno	ATmega328P	32	1	2	14	6	6	2.7"x2.1"	68.6mmx53.3 mm
Mega	ATmega1280	128	4	8	54	14	16	4"x2.1"	101.6mmx53.3mm
Fio	ATmega328P	32	1	2	14	6	8	1.1"x1.6"	27.9mmx40.6 mm

Mega2560	ATmega2560	256	4	8	54	14	16	4"x2.1"	101.6mmx53.3mm

Software

The Arduino IDE is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the *Wiring* project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit makefiles or run programs on the command line.

The Arduino IDE comes with a C/C++ library called "Wiring" (from the project of the same name), which makes many common input/output operations much easier. Arduino programs are written in C/C++, although users only need define two functions to make a runnable program:

- setup() – a function run once at the start of a program that can initialize settings
- loop() – a function called repeatedly until the board powers off

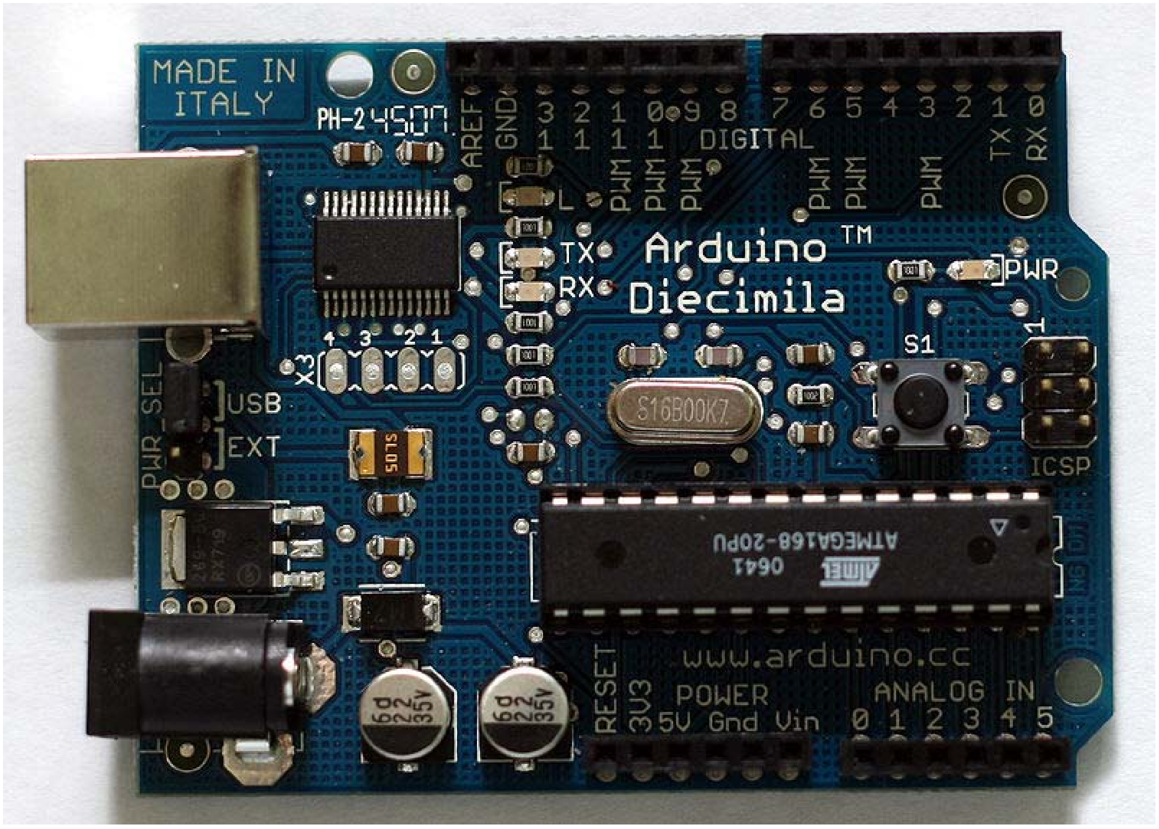
A typical first program for a microcontroller simply blinks a LED (light-emitting diode) on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13

void setup () {
    pinMode (LED_PIN, OUTPUT);     // enable pin 13 for digital output
}

void loop () {
    digitalWrite (LED_PIN, HIGH);  // turn on the LED
    delay (1000);                  // wait one second (1000
milliseconds)
    digitalWrite (LED_PIN, LOW);   // turn off the LED
    delay (1000);                  // wait one second
}
```

The above code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program.

The Arduino IDE uses the GNU toolchain and AVR Libc to compile programs, and uses avrdude to upload programs to the board.

Official hardware



The Arduino Diecimila

The original Arduino hardware is manufactured by the Italian company Smart Projects. Some Arduino-branded boards have been designed by the American company SparkFun Electronics.

Thirteen versions of the Arduino hardware have been commercially produced to date:

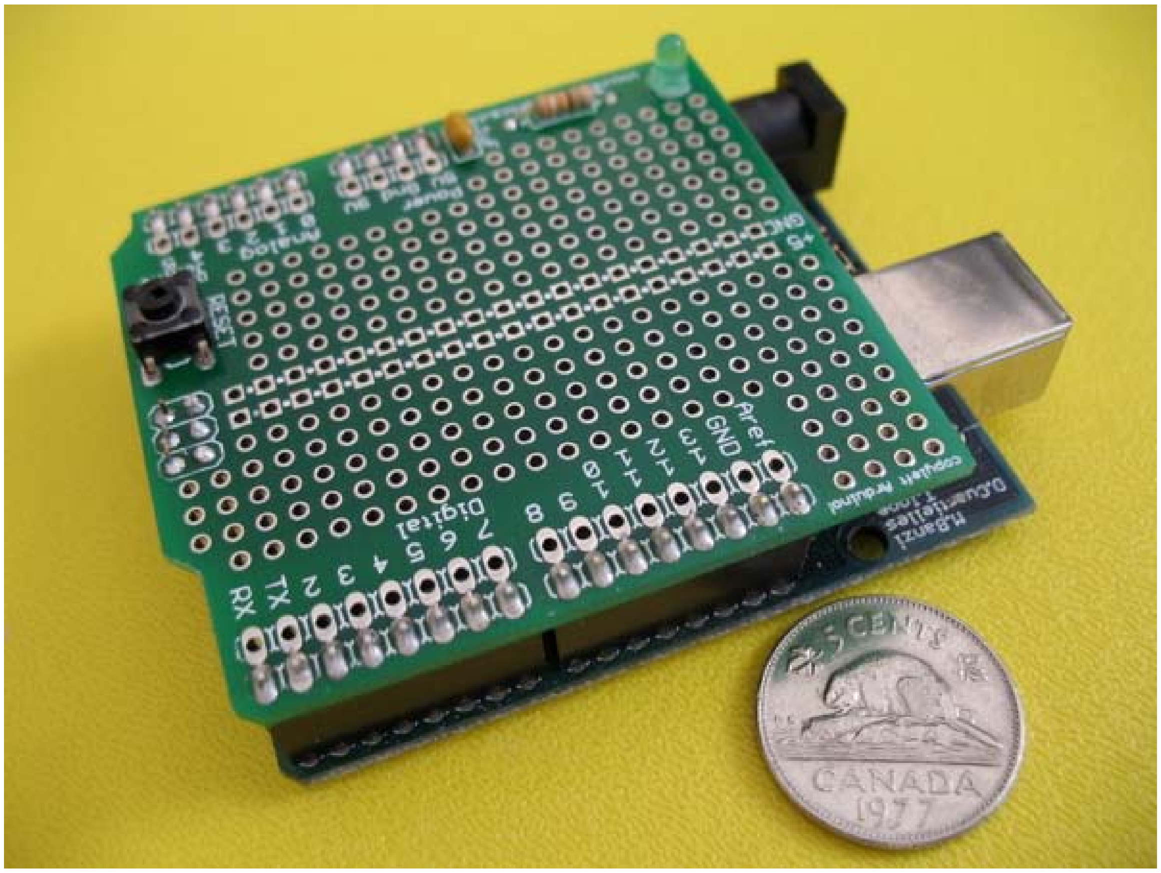
1. The Serial Arduino, programmed with a DE-9 serial connection and using an ATmega8
2. The Arduino Extreme, with a USB interface for programming and using an ATmega8
3. The Arduino Mini, a miniature version of the Arduino using a surface-mounted ATmega168
4. The Arduino Nano, an even smaller, USB powered version of the Arduino using a surface-mounted ATmega168 (ATmega328 for newer version)
5. The LilyPad Arduino, a minimalist design for wearable application using a surface-mounted ATmega168
6. The Arduino NG, with a USB interface for programming and using an ATmega8
7. The Arduino NG plus, with a USB interface for programming and using an ATmega168

8. The Arduino Bluetooth, with a Bluetooth interface for programming using an ATmega168
9. The Arduino Diecimila, with a USB interface and utilizes an ATmega168 in a DIL28 package (pictured)
10. The Arduino Duemilanove ("2009"), using the ATmega168 (ATmega328 for newer version) and powered via USB/DC power, switching automatically
11. The Arduino Mega1280, using a surface-mounted ATmega1280 for additional I/O and memory.
12. The Arduino Uno, uses the same ATmega328 as late-model Duemilanove, but whereas the Duemilanove used an FTDI chipset for USB, the Uno uses an ATmega8U2 programmed as a serial converter.
13. The Arduino Mega2560, uses a surface-mounted ATmega2560, bringing the total memory to 256 kB. It also incorporates the new ATmega8U2 USB chipset.

Open hardware and open source

The Arduino hardware reference designs are distributed under a Creative Commons Attribution Share-Alike 2.5 license and are available on the Arduino Web site. Layout and production files for some versions of the Arduino hardware are also available. The source code for the IDE and the on-board library are available and released under the GPLv2 license.

Accessory hardware



A prototyping shield, mounted on an Arduino

Arduino and Arduino-compatible boards make use of *shields*, which are printed circuit boards that sit atop an Arduino, and plug into the normally supplied pin-headers. These are expansions to the base Arduino. There are many functions of shields, from motor controls, to breadboarding (prototyping).

For example:

- Arduino Ethernet Shield
- XBee Shield
- TouchShield from Liquidware
- Datalog Shield: RTC, SD card storage, temperature sensing, etc. From NuElectronics
- USB Host Shield from Circuits@Home
- Cosmo WiFi Connect from JT5

Arduino-compatible boards

Although the hardware and software designs are freely available under copyleft licenses, the developers have requested that the name "Arduino" be exclusive to the official product and not be used for derivative works without permission. The official policy document on the use of the Arduino name emphasizes that the project is open to incorporating work by others into the official product.

As a result of the protected naming conventions of the Arduino, a group of Arduino users forked the Arduino Diecimila, releasing an equivalent board called Freeduino. The name "Freeduino" is not trademarked and is free to use for any purpose.

Several Arduino-compatible products commercially released have avoided the "Arduino" name by using "-duino" name variants.

Arduino footprint-compatible boards

Example of a Arduino-compatible board: the Freetronics TwentyTen

The following boards are fully or almost fully compatible with both the Arduino hardware and software, including being able to accept "shield" daughterboards.

- The 'Freeduino SB', manufactured and sold as a mini-kit by Solarbotics Ltd..
- The 'Cosmo Black Star', manufactured and sold by JT5.
- The 'Freeduino MaxSerial', a board with a standard DE-9 serial port. It was manufactured and sold assembled or as a kit by Fundamental Logic until May 2010.
- The 'Freeduino Through-Hole', a board that avoids surface-mount soldering, manufactured and sold as a kit by NKC Electronics.
- The 'Illuminato Genesis', a board that uses an ATmega644 instead of an ATmega168. This provides 64 kB of flash, 4 kB of RAM and 42 general I/O pins. Hardware and firmware are open source.
- The 'metaboard', a board that is designed to have a very low complexity and thus a very low price. Hardware and firmware are open source. It was developed by Metalab, a hackerspace in Vienna.
- The 'Seeeduino', derived from the Diecimila.
- The 'eJackino', kit by CQ publisher in Japan. Similar to Seeeduino, eJackino can use Universal boards as Shields. On back side, there is a "Akihabara station" silk, just like Italia on Arduino.
- The 'Japanino' is a kit by Otonano Kagaku publisher in Japan. The board an a POV kit was included in Vol. 27 of the eponymous series. An ATmega168 powers it. It is unique in having a regular size USB A connector.
- The 'Wiseduino' is an Arduino-compatible microcontroller board, which includes a DS1307 real-time clock (RTC) with backup battery, a 24LC256 EEPROM chip and a connector for XBee adapter for wireless communication.
- The 'TwentyTen' is an Arduino-compatible microcontroller board, based on the Duemilanove, with some improvements, including a prototyping area, rearranged LEDs, mini-USB connector, and altered pin 13 circuitry so that the LED and resistor do not interfere with pin function when acting as an input.
- The 'Volksduino' is a low cost, high power, shield compatible, complete Arduino-compatible board kit. Based on the Duemilanove, it comes with a 5 V / 1 A voltage regulator and has an option for a 3.3 V regulator. The Volksduino was designed by Applied Platonics to have a low component count and to be "easy for anyone of any age to put together".

- The 'ZArdino' is a South African Arduino-compatible board derived from the Duemilanove, features mostly through hole construction with the exception of the SMD FT232RL IC, power selection switches, option for a phoenix power connector instead of DC jack, extra I/O pads for using Veroboard as shields, designed to be easy to construct in countries where exotic components are hard to find.

- The 'Zigduino', is an Arduino using the ATmega128RFA1 to integrate Zigbee (IEEE 802.15.4). It can be used with other 802.15.4 network standards as well as ZigBee. The board is the same shape as the Duemilanove and includes an external RPSMA jack on the side of the board opposite the power jack. It is compatible with shields that work with other 3.3 V boards. Due for release Q1 2011 and now taking reservations for the first production run.

- The 'InduinoX' is an Arduino using the ATmega168 and designed for training. It includes on board peripherals such as an RGB LED, switches, and IR Tx/Rx. It was developed and marketed by Simplelabs.

Special purpose Arduino-compatible boards

Special purpose Arduino-compatible boards add additional hardware optimised for a specific application. It is kind of like having an Arduino and a shield on a single board. Some are shield compatible, others are not.

- The "DFRobotShop Rover" is a versatile, minimalist tracked platform based on the Arduino Duemilanove. The PCB incorporates an ATmega328 chip with Arduino bootloader, as well as a dual H-bridge and additional prototyping space and headers. The PCB is compatible with many shields, though four digital pins are used when operating the motor controller. In addition to this, there is an onboard voltage regulator, additional LEDs, a temperature sensor, and a light sensor. The DFRobotShop Rover kit includes a twin motor gearbox, tracks, and minimalist frame to create a tracked mobile robot.

- The "Lightuino", a Arduino-compatible shield that drives LEDs (70 constant-current channels) and LED matrices (1100 LEDs). It also features an adjustable voltage regulator to power the LEDs, an ambient light sensor to decide when to turn "on", and an IR receiver to control your project.

- The "ArduPilot", an Arduino-compatible board designed for auto-piloting and autonomous navigation of aircraft, cars, and boats. It uses GPS for navigation and thermopile sensors or an IMU for stabilization.

Arduino-compatible boards with software-compatibility only

These boards are compatible with the Arduino software but do not accept standard shields. They have different connectors for power and I/O, such as a series of pins on the underside of the board for use with breadboards for easy prototyping, or more specific connectors. One of the important choices made by Arduino-compatible board designers is whether or not to include USB circuitry in the Arduino-compatible board. It is easy to put that circuitry in the cable between development PC and board, thus making each instance of the board less expensive. For many Arduino tasks, the USB circuitry is redundant once the device has been programmed.

- The "Ardweeny" - an inexpensive, even more compact breadboardable device from Solarbotics.
- The "Bare Bones Board" (BBB) and "Really Bare Bones Board" (RBBB) by Modern Device - compact inexpensive Arduino-compatible boards suitable for breadboarding.
- The "Boarduino" - an inexpensive Arduino-Diecimila-compatible board made for breadboarding, produced by Adafruit.
- The "Breaduino" - a complete, very low cost Arduino-compatible kit made to be assembled entirely on a breadboard, made by Applied Platonics.
- The "Diavolino" - another Arduino compatible board from Evil Mad Scientist Laboratories.
- The "DragonFly", a compact board with Molex connectors, aimed at environments where vibration could be an issue. DragonFly features the ATmega1280 and have all 86 IO lines pinned out to connectors.
- The "Femtoduino" - an ultra-small (20.7x15.2 mm) Arduino compatible board designed by Fabio Varesano. Femtoduino is currently the smallest Arduino compatible board available.
- The "iDuino", a USB board for breadboarding, manufactured and sold as a kit by Fundamental Logic.
- The "JeeNode" is a low-cost, low-size, radio-enabled Arduino-compatible board. Based on RBBB with a HopeRF RFM12B wireless module but with a modular approach to I/O interfaces.
- The "LEDuino", a board with enhanced I²C, DCC decoder and CAN-bus interfaces. Manufactured using surface mount and sold assembled by Siliconrailway.
- The "NB1A", is an Arduino-compatible board that includes a battery backed up real-time clock and a four channel DAC. Most Arduino-compatible boards require an additional shield for these resources.
- The "NB2A", is an Sanguino-compatible board that includes a battery backed up real-time clock and a two channel DAC. Sanguino's feature the ATmega644P, which has additional memory, I/O lines and a second UART.
- The "Nymph", a compact board with Molex connectors, aimed at environments where vibration could be an issue. Nymph features the ATmega328P.
- The "Oak Micros om328p" - an Arduino Duemilanove compacted down to a breadboardable device (36 mm x 18 mm) that can be inserted into a standard 600 mil 28-pin socket, with USB capability, ATmega328P, and 6 onboard LEDs.
- The "Rainbowduino", is an Arduino-compatible board designed specifically for driving LEDs. It is generally used to drive an 8x8 RGB LED matrix using row scanning, but can be used for other things.

- The "Roboduino", designed for robotics. All of its connections have neighboring power buses that accommodate servos and sensors. Additional headers for power and serial communication are also provided. It was developed by Curious Inventor, LLC.
- The "Sanguino" - An open source enhanced Arduino-compatible board that uses an ATMega644P instead of an ATMega168. This provides 64 kB of flash, 4 kB of RAM and 32 general I/O pins in a 40 pin DIP device. It was developed with the RepRap Project in mind.
- The "Seeeduino Mega", is an Arduino-Mega-compatible board with 16 extra I/O Pins.
- The "Stickduino", similar to a USB key.
- The "Wireless Widget", is a compact (35 mm x 70 mm), low voltage, battery powered Arduino-compatible board with onboard wireless capable of ranges up to 120 m. The Wireless Widget was designed for both portable and low cost Wireless sensor network applications.
- The "Teensy and Teensy++" - a pair of boards from PJRC.com that run most Arduino sketches using the Teensyduino software add-on to the Arduino IDE.
- The "ZB1", is an Arduino-compatible board that includes a Zigbee radio (XBee). The ZB1 can be powered by USB, a wall adapter or an external battery source. It is designed for low-cost Wireless sensor network applications.

Non-ATmega boards

The following boards accept Arduino "shield" daughter boards but do not use ATmega micro-controllers, and are therefore incompatible with the Arduino IDE.

- The "Amicus18", The Amicus18 is an embedded system platform based on PIC architecture (18F25K20). Can be programmed with any programming language, though the Amicus IDE is completely free and extremely powerful.
- The "PROplus", ARM 100 MHz Cortex M3 and ARM7TDMI-based shield-compatible boards from Coridium, programmable in BASIC or C.
- The "Cortino", a development system for the 32-bit ARM Cortex M3 Microprocessor.
- The "Pinguino", is a board based on a PIC microcontroller, with native USB support and compatibility with the Arduino programing language plus an IDE built with Python and sdcc as compiler.
- The "Unduino", is a board based on the dsPIC33FJ128MC202 microcontroller, with integrated motor control peripherals.
- The "Leaflabs Maple", a 72 MHz 32-bit ARM Cortex M3 micro-controller with USB support, compatibility with Arduino shields, and 39 GP I/O pins. Programmable with the open source Maple IDE (which is a branch of the Arduino library) or low-level native code (with support from the libmaple C library).
- The "Netduino", a 48 MHz 32-bit ARM7 micro-controller board with support for the .NET Micro Framework. Pin compatible with Arduino shields although drivers are required for some shields.

- The "Vinculo", a USB development board for the FTDI Vinculum II microcontroller.
- The "FEZ Domino" and "FEZ Panda", 72 MHz 32-bit ARM (GHI Electronics USBizi chips) micro-controller boards with support for the .NET Micro Framework. Pin compatible with Arduino shields, although drivers are required for some shields.

Development team

The core Arduino developer team is composed of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, David Mellis and Nicholas Zambetti. Massimo Banzi was interviewed on the March 21st, 2009 episode (Episode 61) of FLOSS Weekly on the TWiT.tv network, in which he discussed the history and goals of the Arduino project.

Chapter-4

BASIC Stamp and Freescale 68HC11

BASIC Stamp

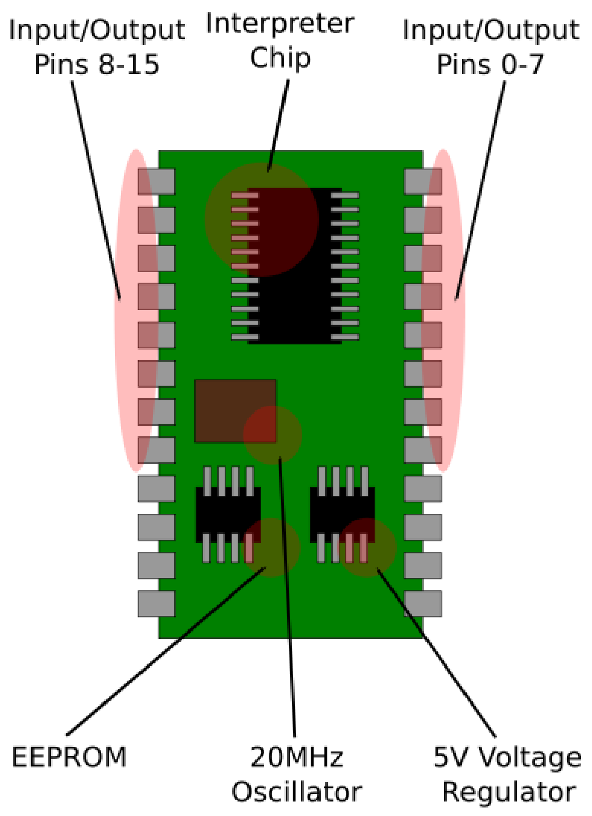


Diagram of BASIC Stamp

The **BASIC Stamp** is a microcontroller with a small, specialized BASIC interpreter (PBASIC) built into ROM. It is made by Parallax, Inc. and has been popular with electronics hobbyists since the early 1990s due to its low threshold of learning and ease of use (due to its simple BASIC language).

Technical specifications

Although the BASIC Stamp has the form of a DIP chip, it is in fact a small Printed Circuit Board that contains the essential elements of a microprocessor system:

- A Microcontroller containing the CPU, a built in ROM containing the BASIC interpreter, and various peripherals
- Memory (an i^2C EEPROM)
- A clock, usually in the form of a ceramic resonator
- A power supply
- External input and output

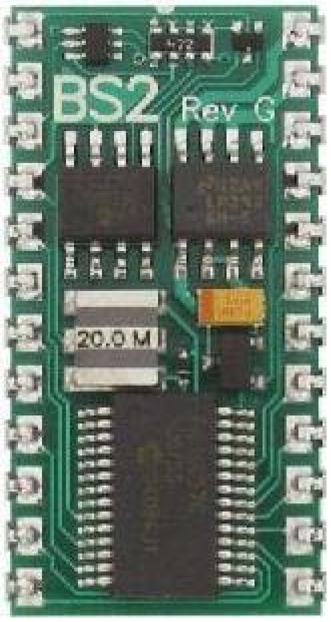
The end result is that a hobbyist can connect a 9 V battery to a BASIC Stamp and have a complete system. A connection to a personal computer allows the programmer to download software to the BASIC Stamp, which is stored in the onboard non-volatile memory device: it remains programmed until it is erased or reprogrammed, even when the power is removed.

Programming

The BASIC Stamp is programmed in a variant of the BASIC language, called PBASIC. PBASIC incorporates common microcontroller functions, including PWM, serial communications, I^2C and 1-Wire communications, communications with common LCD driver circuits, hobby servo pulse trains, pseudo-sine wave frequencies, and the ability to time an RC circuit which may be used to detect an analog value.

Once the program has been written, it is tokenized and sent to the chip through a serial cable.

Versions



The BASIC Stamp 2

There are currently four variants of the interpreter:

1. BASIC Stamp 1 (BS1),
2. BASIC Stamp 2 (BS2), with six sub-variants:
    1. BS2e
    2. BS2sx
    3. BS2p24
    4. BS2p40
    5. BS2pe
    6. BS2px
3. Javelin Stamp
4. Spin Stamp.

The BS2 sub-variants feature more memory, faster execution speed, additional specialized PBASIC commands, extra I/O pins, etc, in comparison to the original BS2 model. While the BS1 and BS2 use a PIC, the remaining BASIC Stamp 2 variants use a Parallax SX processor.

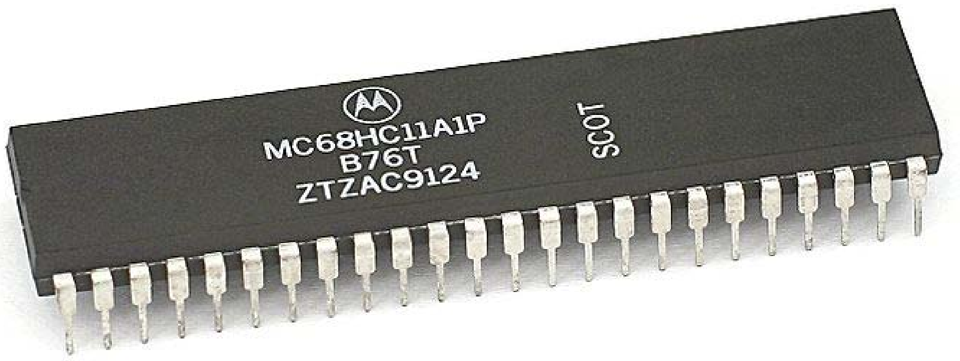
The third variant is the Javelin Stamp. This module uses a subset of Sun Microsystems' Java programming language instead of Parallax's PBASIC. It does not include any networking facilities.

The fourth variant is the Spin Stamp. The module is based on the Parallax Propeller and therefore uses the SPIN programming language instead of PBASIC.

A number of companies now make "clones" of the BASIC Stamp with additional features, such as faster execution, analog-to-digital converters and hardware-based PWM which can run in the background. However, while many use the same pinout as the BASIC Stamp in order to be hardware-compatible in larger-scale projects, they are not necessarily software-compatible.

The Parallax Propeller is gradually accumulating software libraries which give it similar functionality to the BASIC Stamp, however there is no uniform list of which PBASIC facilities now have Spin equivalents.

Freescale 68HC11



Motorola MC68HC11, plastic DIP.

The MC68HC11A8 is available in a 48-pin dual in-line package (DIP), as well as the 52-pin plastic leaded chip carrier (PLCC) as shown above.

The **68HC11** (**6811** or **HC11** for short) is an 8-bit microcontroller (µC) family introduced by Motorola in 1985. Now produced by Freescale Semiconductor, it descended from the Motorola 6800 microprocessor. It is a CISC microcontroller. The 68HC11 devices are more powerful and more expensive than the 68HC08 microcontrollers, and are used in barcode readers, hotel card key writers, amateur robotics, and various other embedded systems. The MC68HC11A8 was the first MCU to include CMOS EEPROM.

Internally, the HC11 instruction set is upward compatible with the 6800, with the addition of a Y index register. (Instructions using the Y register have opcodes prefixed with the byte 0x18). It has two eight-bit accumulators, A and B, two sixteen-bit index registers, X and Y, a condition code register, a 16-bit stack pointer, and a program counter. In addition, some instructions treat the A and B registers as a combined 16-bit D register.

The standard bootloader for the HC11 family is called BUFFALO, "Bit User Fast Friendly Aid to Logical Operation" (a BUFFALO prompt seen on the serial port at bootup is a sign that a board's flash memory has been erased). Not all HC11 models come with the BUFFALO bootloader. The 68HC11A0 and A1 do not but the A8 does.

Different versions of the HC11 have different numbers of external ports, labeled alphabetically. The most common version has five ports, A, B, C, D, and E, but some have as few as 3 ports (version D3). Each port is eight-bits wide except for D, which is six bits (in some variations of the chip, D also has eight bits). It can be operated with an internal program and RAM (1 to 768 bytes) or an external memory of up to 64 kilobytes. With external memory, B and C are used as address and data bus. In this mode, port C is multiplexed to carry both the lower byte of the address and data.

A MC68HC24 port replacement unit is available for the HC11. When placed on the external address bus, it replicates the original functions of B and C. Port A has input capture, output compare, pulse accumulator, and other timer functions; port D has serial I/O, and port E has an analog to digital converter (ADC).

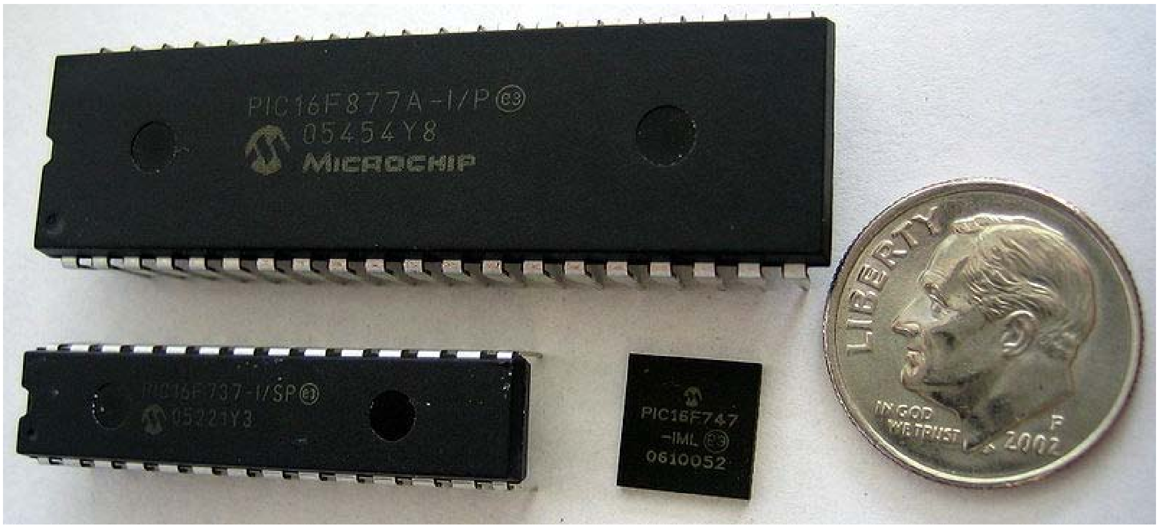
In the early 1990s Motorola produced an evaluation board kit for the 68HC11 with several UARTs, RAM, and an EPROM. The cost of the evaluation kit was $68.11.

The Freescale 68HC12 is an enhanced 16-bit version of the 68HC11.

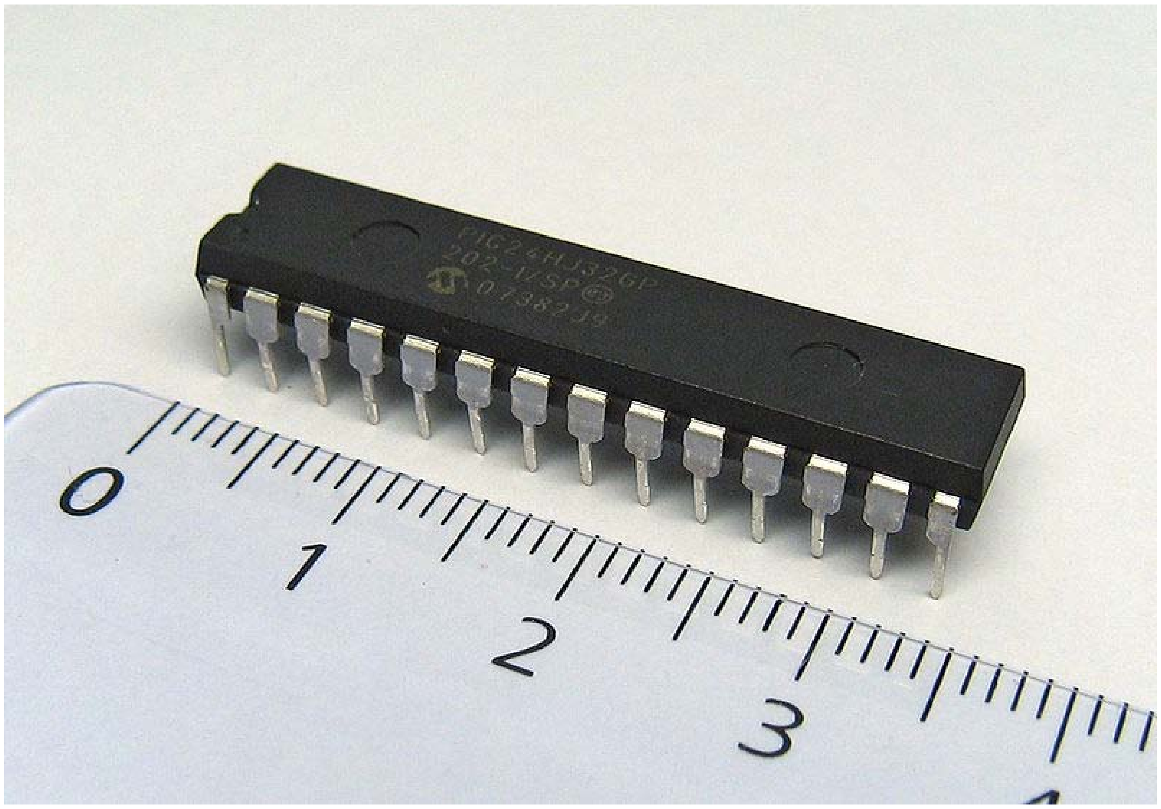
The Freescale 68HC16 microcontroller is intended as a 16-bit mostly software compatible upgrade of the 68HC11.

Chapter-5

PIC Microcontroller



PIC microcontrollers in DIP and QFN packages



16-bit 28-pin PDIP PIC24 microcontroller next to a metric ruler

PIC is a family of Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1640 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "**Peripheral Interface Controller**".

PICs are popular with both industrial developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

Microchip announced on February 2008 the shipment of its six billionth PIC processor.

Core architecture

The PIC architecture is characterized by its multiple attributes:

- Separate code and data spaces (Harvard architecture) for devices other than PIC32, which has a Von Neumann architecture.
- A small number of fixed length instructions
- Most instructions are single cycle execution (2 clock cycles), with one delay cycle on branches and skips

- One accumulator (W0), the use of which (as source operand) is implied (i.e. is not encoded in the opcode)
- All RAM locations function as registers as both source and/or destination of math and other functions.
- A hardware stack for storing return addresses
- A fairly small amount of addressable data space (typically 256 bytes), extended through banking
- Data space mapped CPU, port, and peripheral registers
- The program counter is also mapped into the data space and writable (this is used to implement indirect jumps).

There is no distinction between memory space and register space because the RAM serves the job of both memory and registers, and the RAM is usually just referred to as the register file or simply as the registers.

Data space (RAM)

PICs have a set of registers that function as general purpose RAM. Special purpose control registers for on-chip hardware resources are also mapped into the data space. The addressability of memory varies depending on device series, and all PIC devices have some banking mechanism to extend addressing to additional memory. Later series of devices feature move instructions which can cover the whole addressable space, independent of the selected bank. In earlier devices, any register move had to be achieved via the accumulator.

To implement indirect addressing, a "file select register" (FSR) and "indirect register" (INDF) are used. A register number is written to the FSR, after which reads from or writes to INDF will actually be to or from the register pointed to by FSR. Later devices extended this concept with post- and pre- increment/decrement for greater efficiency in accessing sequentially stored data. This also allows FSR to be treated almost like a stack pointer (SP).

External data memory is not directly addressable except in some high pin count PIC18 devices.

Code space

The code space is generally implemented as ROM, EPROM or flash ROM. In general, external code memory is not directly addressable due to the lack of an external memory interface. The exceptions are PIC17 and select high pin count PIC18 devices.

Word size

All PICs handle (and address) data in 8-bit chunks. However, the unit of addressability of the code space is not generally the same as the data space. For example, PICs in the baseline and mid-range families have program memory addressable in the same wordsize as the instruction width, i.e. 12 or 14 bits respectively. In contrast, in the PIC18 series, the program memory is addressed in 8-bit increments (bytes), which differs from the instruction width of 16 bits.

In order to be clear, the program memory capacity is usually stated in number of (single word) instructions, rather than in bytes.

Stacks

PICs have a hardware call stack, which is used to save return addresses. The hardware stack is not software accessible on earlier devices, but this changed with the 18 series devices.

Hardware support for a general purpose parameter stack was lacking in early series, but this greatly improved in the 18 series, making the 18 series architecture more friendly to high level language compilers.

Instruction set

A PIC's instructions vary from about 35 instructions for the low-end PICs to over 80 instructions for the high-end PICs. The instruction set includes instructions to perform a variety of operations on registers directly, the accumulator and a literal constant or the accumulator and a register, as well as for conditional execution, and program branching.

Some operations, such as bit setting and testing, can be performed on any numbered register, but bi-operand arithmetic operations always involve W (the accumulator), writing the result back to either W or the other operand register. To load a constant, it is necessary to load it into W before it can be moved into another register. On the older cores, all register moves needed to pass through W, but this changed on the "high end" cores.

PIC cores have skip instructions which are used for conditional execution and branching. The skip instructions are 'skip if bit set' and 'skip if bit not set'. Because cores before PIC18 had only unconditional branch instructions, conditional jumps are implemented by a conditional skip (with the opposite condition) followed by an unconditional branch. Skips are also of utility for conditional execution of any immediate single following instruction.

The 18 series implemented shadow registers which save several important registers during an interrupt, providing hardware support for automatically saving processor state when servicing interrupts.

In general, PIC instructions fall into 5 classes:

1. Operation on working register (WREG) with 8-bit immediate ("literal") operand. E.g. `movlw` (move literal to WREG), `andlw` (AND literal with WREG). One

instruction peculiar to the PIC is `retlw`, load immediate into WREG and return, which is used with computed branches to produce lookup tables.
2. Operation with WREG and indexed register. The result can be written to either the Working register (e.g. `addwf` *`reg`*`,w`). or the selected register (e.g. `addwf` *`reg`*`,f`).
3. Bit operations. These take a register number and a bit number, and perform one of 4 actions: set or clear a bit, and test and skip on set/clear. The latter are used to perform conditional branches. The usual ALU status flags are available in a numbered register so operations such as "branch on carry clear" are possible.
4. Control transfers. Other than the skip instructions previously mentioned, there are only two: `goto` and `call`.
5. A few miscellaneous zero-operand instructions, such as return from subroutine, and `sleep` to enter low-power mode.

Performance

The architectural decisions are directed at the maximization of speed-to-cost ratio. The PIC architecture was among the first scalar CPU designs, and is still among the simplest and cheapest. The Harvard architecture—in which instructions and data come from separate sources—simplifies timing and microcircuit design greatly, and this benefits clock speed, price, and power consumption.

The PIC instruction set is suited to implementation of fast lookup tables in the program space. Such lookups take one instruction and two instruction cycles. Many functions can be modeled in this way. Optimization is facilitated by the relatively large program space of the PIC (e.g. 4096 x 14-bit words on the 16F690) and by the design of the instruction set, which allows for embedded constants. For example, a branch instruction's target may be indexed by W, and execute a "RETLW" which does as it is named - return with literal in W.

Execution time can be accurately estimated by multiplying the number of instructions by two cycles; this simplifies design of real-time code. Similarly, interrupt latency is constant at three instruction cycles. External interrupts have to be synchronized with the four clock instruction cycle, otherwise there can be a one instruction cycle jitter. Internal interrupts are already synchronized. The constant interrupt latency allows PICs to achieve interrupt driven low jitter timing sequences. An example of this is a video sync pulse generator. This is no longer true in the newest PIC models, because they have a synchronous interrupt latency of three or four cycles.

Advantages

The PIC architectures have these advantages:

- Small instruction set to learn
- RISC architecture
- Built in oscillator with selectable speeds

- Inexpensive microcontrollers
- Wide range of interfaces including I2C, SPI, USB, USART, A/D, programmable Comparators, PWM, LIN, CAN, PSP, and Ethernet

Limitations

The PIC architectures have these limitations:

- One accumulator
- Register-bank switching is required to access the entire RAM of many devices
- Operations and registers are not orthogonal; some instructions can address RAM and/or immediate constants, while others can only use the accumulator

The following limitations have been addressed in the **PIC18** series, but still apply to earlier cores:

- Stack:

1. The hardware call stack is not addressable, so preemptive task switching cannot be implemented
2. Software-implemented stacks are not efficient, so it is difficult to generate reentrant code and support local variables

With paged program memory, there are two page sizes to worry about: one for CALL and GOTO and another for computed GOTO (typically used for table lookups). For example, on PIC16, CALL and GOTO have 11 bits of addressing, so the page size is 2048 instruction words. For computed GOTOs, where you add to PCL, the page size is 256 instruction words. In both cases, the upper address bits are provided by the PCLATH register. This register must be changed every time control transfers between pages. PCLATH must also be preserved by any interrupt handler.

Compiler development

While several commercial compilers are available, in 2008, Microchip released their own C compilers, C18 and C30, for the line of 18F 24F and 30/33F processors. By contrast, Atmel's AVR microcontrollers—which are competitive with PIC in terms of hardware capabilities and price, but feature a more traditional instruction set—have long been supported by the GNU C Compiler.

The easy to learn RISC instruction set of the PIC assembly language code can make the overall flow difficult to comprehend. Judicious use of simple macros can increase the readability of PIC assembly language. For example, the original Parallax PIC assembler ("SPASM") has macros which hide W and make the PIC look like a two-address machine. It has macro instructions like "`mov b, a`" (move the data from address *a* to address *b*) and "`add b, a`" (add data from address *a* to data in address *b*). It also hides the skip instructions by providing three operand branch macro instructions such as "`cjne a, b, dest`" (compare *a* with *b* and jump to *dest* if they are not equal).

Family core architectural differences

Baseline core devices

These devices feature a 12-bit wide code memory, a 32-byte register file, and a tiny two level deep call stack. They are represented by the PIC10 series, as well as by some PIC12 and PIC16 devices. Baseline devices are available in 6-pin to 40-pin packages.

Generally the first 7 to 9 bytes of the register file are special-purpose registers, and the remaining bytes are general purpose RAM. If banked RAM is implemented, the bank number is selected by the high 3 bits of the FSR. This affects register numbers 16–31; registers 0–15 are global and not affected by the bank select bits.

The ROM address space is 512 words (12 bits each), which may be extended to 2048 words by banking. `CALL` and `GOTO` instructions specify the low 9 bits of the new code location; additional high-order bits are taken from the status register. Note that a CALL instruction only includes 8 bits of address, and may only specify addresses in the first half of each 512-word page.

The instruction set is as follows. Register numbers are referred to as "f", while constants are referred to as "k". Bit numbers (0–7) are selected by "b". The "d" bit selects the destination: 0 indicates W, while 1 indicates that the result is written back to source register f.

12-bit PIC instruction set

Opcode (binary)	Mnemonic	Description
`0000 0000 0000`	NOP	No operation
`0000 0000 0010`	OPTION	Load OPTION register with contents of W
`0000 0000 0011`	SLEEP	Go into standby mode
`0000 0000 0100`	CLRWDT	Reset watchdog timer
`0000 0000 01ff`	TRIS f	Move W to port control register (f=1..3)
`0000 001 fffff`	MOVWF f	Move W to f
`0000 010 xxxxx`	CLRW	Clear W to 0 (a.k.a CLR x, W)
`0000 011 fffff`	CLRF f	Clear f to 0 (a.k.a. CLR f, F)
`0000 10d fffff`	SUBWF f, d	Subtract W from f (d = f − W)
`0000 11d fffff`	DECF f, d	Decrement f (d = f − 1)
`0001 00d fffff`	IORWF f, d	Inclusive OR W with F (d = f OR W)
`0001 01d fffff`	ANDWF f, d	AND W with F (d = f AND W)
`0001 10d fffff`	XORWF f, d	Exclusive OR W with F (d = f XOR W)

`0001 11d fffff`	ADDWF f, d	Add W with F (d = f + W)
`0010 00d fffff`	MOVF f, d	Move F (d = f)
`0010 01d fffff`	COMF f, d	Complement f (d = NOT f)
`0010 10d fffff`	INCF f, d	Increment f (d = f + 1)
`0010 11d fffff`	DECFSZ f, d	Decrement f (d = f − 1) and skip if zero
`0011 00d fffff`	RRF f, d	Rotate right F (rotate right through carry)
`0011 01d fffff`	RLF f, d	Rotate left F (rotate left through carry)
`0011 10d fffff`	SWAPF f, d	Swap 4-bit halves of f (d = f<<4 | f>>4)
`0011 11d fffff`	INCFSZ f, d	Increment f (d = f + 1) and skip if zero
`0100 bbb fffff`	BCF f, b	Bit clear f (Clear bit b of f)
`0101 bbb fffff`	BSF f, b	Bit set f (Set bit b of f)
`0110 bbb fffff`	BTFSC f, b	Bit test f, skip if clear (Test bit b of f)
`0111 bbb fffff`	BTFSS f, b	Bit test f, skip if set (Test bit b of f)
`1000 kkkkkkkk`	RETLW k	Set W to k and return
`1001 kkkkkkkk`	CALL k	Save return address, load PC with k
`101k kkkkkkkk`	GOTO k	Jump to address k *(9 bits)*
`1100 kkkkkkkk`	MOVLW k	Move literal to W (W = k)
`1101 kkkkkkkk`	IORLW k	Inclusive or literal with W (W = k OR W)
`1110 kkkkkkkk`	ANDLW k	AND literal with W (W = k AND W)
`1111 kkkkkkkk`	XORLW k	Exclusive or literal with W (W = k XOR W)

Mid-range core devices

These devices feature a 14-bit wide code memory, and an improved 8 level deep call stack. The instruction set differs very little from the baseline devices, but the increased opcode width allows 128 registers and 2048 words of code to be directly addressed. The mid-range core is available in the majority of devices labeled PIC12 and PIC16.

The first 32 bytes of the register space are allocated to special-purpose registers; the remaining 96 bytes are used for general-purpose RAM. If banked RAM is used, the high 16 registers (0x70–0x7F) are global, as are a few of the most important special-purpose registers, including the STATUS register which holds the RAM bank select bits. (The other global registers are FSR and INDF, the low 8 bits of the program counter PCL, the PC high preload register PCLATH, and the master interrupt control register INTCON.)

The PCLATH register supplies high-order instruction address bits when the 8 bits supplied by a write to the PCL register, or the 11 bits supplied by a `GOTO` or `CALL` instruction, is not sufficient to address the available ROM space.

14-bit PIC instruction set

Opcode (binary)	**Mnemonic**	**Description**
00 0000 0000 0000	NOP	No operation
00 0000 0000 1000	RETURN	Return from subroutine, W unchanged
00 0000 0000 1001	RETFIE	Return from interrupt
00 0000 0110 0010	OPTION	Write W to OPTION register
00 0000 0110 0011	SLEEP	Go into standby mode
00 0000 0110 0100	CLRWDT	Reset watchdog timer
00 0000 0110 01ff	TRIS f	Write W to tristate register f
00 0000 1 fffffff	MOVWF f	Move W to f
00 0001 0 xxxxxxx	CLRW	Clear W to 0 (W = 0)
00 0001 1 fffffff	CLRF f	Clear f to 0 (f = 0)
00 0010 d fffffff	SUBWF f, d	Subtract W from f (d = f − W)
00 0011 d fffffff	DECF f, d	Decrement f (d = f − 1)
00 0100 d fffffff	IORWF f, d	Inclusive OR W with F (d = f OR W)
00 0101 d fffffff	ANDWF f, d	AND W with F (d = f AND W)
00 0110 d fffffff	XORWF f, d	Exclusive OR W with F (d = f XOR W)
00 0111 d fffffff	ADDWF f, d	Add W with F (d = f + W)
00 1000 d fffffff	MOVF f, d	Move F (d = f)
00 1001 d fffffff	COMF f, d	Complement f (d = NOT f)
00 1010 d fffffff	INCF f, d	Increment f (d = f + 1)
00 1011 d fffffff	DECFSZ f, d	Decrement f (d = f − 1) and skip if zero
00 1100 d fffffff	RRF f, d	Rotate right F (rotate right through carry)
00 1101 d fffffff	RLF f, d	Rotate left F (rotate left through carry)
00 1110 d fffffff	SWAPF f, d	Swap 4-bit halves of f (d = f<<4 | f>>4)
00 1111 d fffffff	INCFSZ f, d	Increment f (d = f + 1) and skip if zero
01 00 bbb fffffff	BCF f, b	Bit clear f (Clear bit b of f)
01 01 bbb fffffff	BSF f, b	Bit set f (Set bit b of f)
01 10 bbb fffffff	BTFSC f, b	Bit test f, skip if clear (Test bit b of f)
01 11 bbb fffffff	BTFSS f, b	Bit test f, skip if set (Test bit b of f)
10 0 kkkkkkkkkkk	CALL k	Save return address, load PC with k
10 1 kkkkkkkkkkk	GOTO k	Jump to address k (11 bits)
11 00xx kkkkkkkk	MOVLW k	Move literal to W (W = k)
11 01xx kkkkkkkk	RETLW k	Set W to k and return

`11 1000 kkkkkkkk`	IORLW k	Inclusive or literal with W (W = k OR W)
`11 1001 kkkkkkkk`	ANDLW k	AND literal with W (W = k AND W)
`11 1010 kkkkkkkk`	XORLW k	Exclusive or literal with W (W = k XOR W)
`11 110x kkkkkkkk`	SUBLW k	Subtract W from literal (W = k − W)
`11 111x kkkkkkkk`	ADDLW k	Add literal to W (W = k + W)

Enhanced Mid-range core devices

Enhanced Mid-range core devices introduce a deeper hardware stack, additional reset methods, 14 additional instructions and 'C' programming language optimizations.

PIC17 high end core devices

The 17 series never became popular and has been superseded by the PIC18 architecture. It is not recommended for new designs, and availability may be limited.

Improvements over earlier cores are 16-bit wide opcodes (allowing many new instructions), and a 16 level deep call stack. PIC17 devices were produced in packages from 40 to 68 pins.

The 17 series introduced a number of important new features:

- a memory mapped accumulator
- read access to code memory (table reads)
- direct register to register moves (prior cores needed to move registers through the accumulator)
- an external program memory interface to expand the code space
- an 8-bit x 8-bit hardware multiplier
- a second indirect register pair
- auto-increment/decrement addressing controlled by control bits in a status register (ALUSTA)

PIC18 high end core devices

Microchip introduced the PIC18 architecture in 2000. Unlike the 17 series, it has proven to be very popular, with a large number of device variants presently in manufacture. In contrast to earlier devices, which were more often than not programmed in assembly, C has become the predominant development language .

The 18 series inherits most of the features and instructions of the 17 series, while adding a number of important new features:

- much deeper call stack (31 levels deep)
- the call stack may be read and written
- conditional branch instructions

- indexed addressing mode (PLUSW)
- extending the FSR registers to 12 bits, allowing them to linearly address the entire data address space
- the addition of another FSR register (bringing the number up to 3)

The auto increment/decrement feature was improved by removing the control bits and adding four new indirect registers per FSR. Depending on which indirect file register is being accessed it is possible to postdecrement, postincrement, or preincrement FSR; or form the effective address by adding W to FSR.

In more advanced PIC18 devices, an "extended mode" is available which makes the addressing even more favorable to compiled code:

- a new offset addressing mode; some addresses which were relative to the access bank are now interpreted relative to the FSR2 register
- the addition of several new instructions, notable for manipulating the FSR registers.

These changes were primarily aimed at improving the efficiency of a data stack implementation. If FSR2 is used either as the stack pointer or frame pointer, stack items may be easily indexed—allowing more efficient re-entrant code. Microchip's MPLAB C18 C compiler chooses to use FSR2 as a frame pointer.

PIC24 and dsPIC 16-bit microcontrollers

In 2001, Microchip introduced the dsPIC series of chips, which entered mass production in late 2004. They are Microchip's first inherently 16-bit microcontrollers. PIC24 devices are designed as general purpose microcontrollers. dsPIC devices include digital signal processing capabilities in addition.

Architecturally, although they share the PIC moniker, they are very different from the 8-bit PICs. The most notable differences are:

- they feature a set of 16 working registers (W0-W15)
- they fully support a stack in RAM, and do not have a hardware stack
- bank switching is not required to access RAM or special function registers
- data stored in program memory can be accessed directly using a feature called Program Space Visibility
- interrupt sources may be assigned to distinct handlers using an interrupt vector table

Some features are:

- hardware MAC (multiply-accumulate)
- barrel shifting
- bit reversal

- (16×16)-bit single-cycle multiplication and other DSP operations
- hardware divide assist (19 cycles for 16/32-bit divide)
- hardware support for loop indexing
- Direct memory access

dsPICs can be programmed in C using a variant *[What Variant ?]* of gcc.

PIC32 32-bit microcontrollers

In November 2007 Microchip introduced the new PIC32MX family of 32-bit microcontrollers. The initial device line-up is based on the industry standard MIPS32 M4K Core. The device can be programmed using the Microchip MPLAB C Compiler for PIC32 MCUs, a variant of the GCC compiler. The first 18 models currently in production (PIC32MX3xx and PIC32MX4xx) are pin to pin compatible and share the same peripherals set with the PIC24FxxGA0xx family of (16-bit) devices allowing the use of common libraries, software and hardware tools.

The PIC32 architecture brings a number of new features to Microchip portfolio, including:

- The highest execution speed 80 MIPS (120+ Dhrystone MIPS @ 80 MHz)
- The largest flash memory: 512 kByte
- One instruction per clock cycle execution
- The first cached processor
- Allows execution from RAM
- Full Speed Host/Dual Role and OTG USB capabilities
- Full JTAG and 2 wire programming and debugging
- Real-time trace

Device variants and hardware features

PIC devices generally feature:

- Sleep mode (power savings).
- Watchdog timer.
- Various crystal or RC oscillator configurations, or an external clock.

Variants

Within a series, there are still many device variants depending on what hardware resources the chip features.

- General purpose I/O pins.
- Internal clock oscillators.
- 8/16/32 Bit Timers.
- Internal EEPROM Memory.

- Synchronous/Asynchronous Serial Interface USART.
- MSSP Peripheral for I²C and SPI Communications.
- Capture/Compare and PWM modules.
- Analog-to-digital converters (up to ~1.0 MHz).
- USB, Ethernet, CAN interfacing support.
- External memory interface.
- Integrated analog RF front ends (PIC16F639, and rfPIC).
- KEELOQ Rolling code encryption peripheral (encode/decode)
- And many more.

Trends

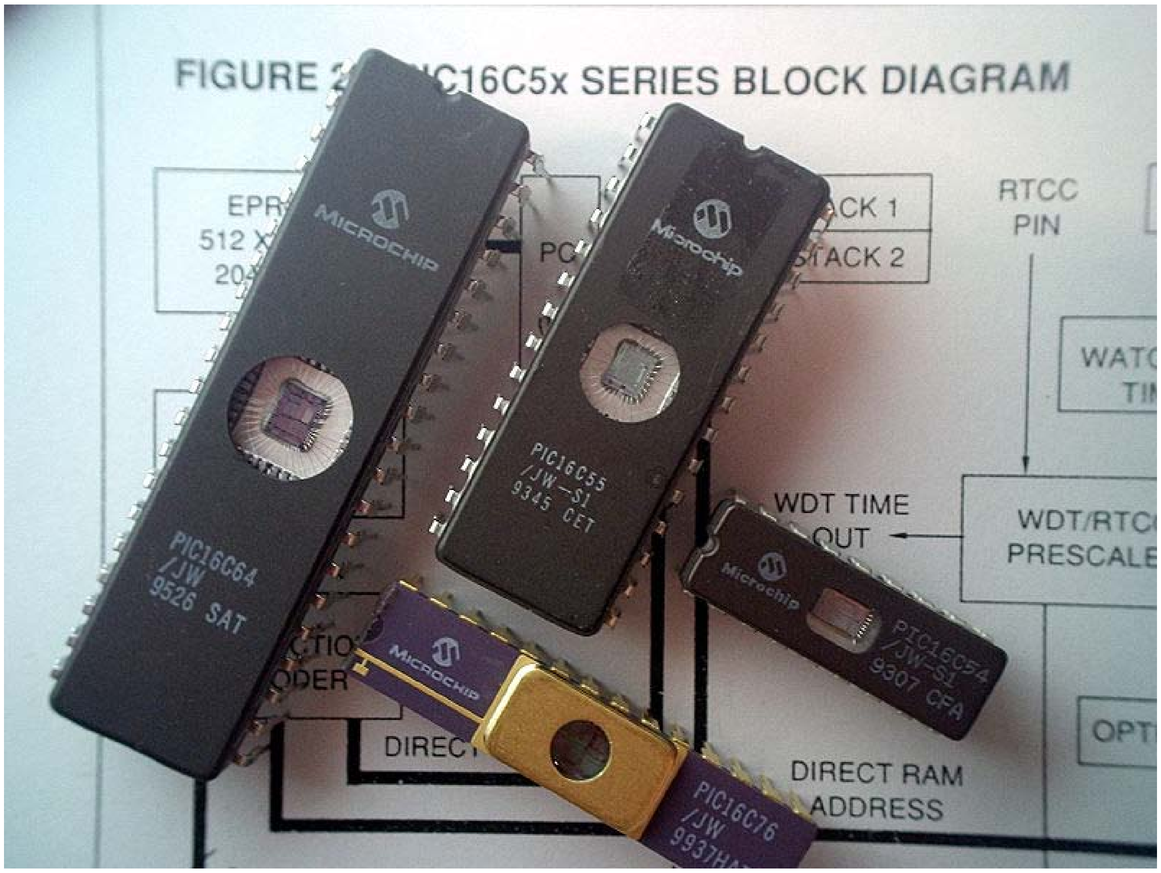
The first generation of PICs with EPROM storage are almost completely replaced by chips with Flash memory. Likewise, the original 12-bit instruction set of the PIC1650 and its direct descendants has been superseded by 14-bit and 16-bit instruction sets. Microchip still sells OTP (one-time-programmable) and windowed (UV-erasable) versions of some of its EPROM based PICs for legacy support or volume orders. The Microchip website lists PICs that are not electrically erasable as OTP despite the fact that UV erasable windowed versions of these chips can be ordered.

History

The original PIC was built to be used with General Instruments' new 16-bit CPU, the CP1600. While generally a good CPU, the CP1600 had poor I/O performance, and the 8-bit PIC was developed in 1975 to improve performance of the overall system by offloading I/O tasks from the CPU. The PIC used simple microcode stored in ROM to perform its tasks, and although the term was not used at the time, it shares some common features with RISC designs.

In 1985, General Instruments spun off their microelectronics division and the new ownership cancelled almost everything — which by this time was mostly out-of-date. The PIC, however, was upgraded with internal EPROM to produce a programmable channel controller and today a huge variety of PICs are available with various on-board peripherals (serial communication modules, UARTs, motor control kernels, etc.) and program memory from 256 words to 64k words and more (a "word" is one assembly language instruction, varying from 12, 14 or 16 bits depending on the specific PIC micro family).

PIC and PICmicro are registered trademarks of Microchip Technology. It is generally thought that PIC stands for **Peripheral Interface Controller**, although General Instruments' original acronym for the initial PIC1640 and PIC1650 devices was "**Programmable Interface Controller**". The acronym was quickly replaced with "**Programmable Intelligent Computer**".



Various older (EPROM) PIC microcontrollers

The Microchip 16C84 (PIC16x84), introduced in 1993 , was the first Microchip CPU with on-chip EEPROM memory. This electrically-erasable memory made it cost less than CPUs that required a quartz "erase window" for erasing EPROM.

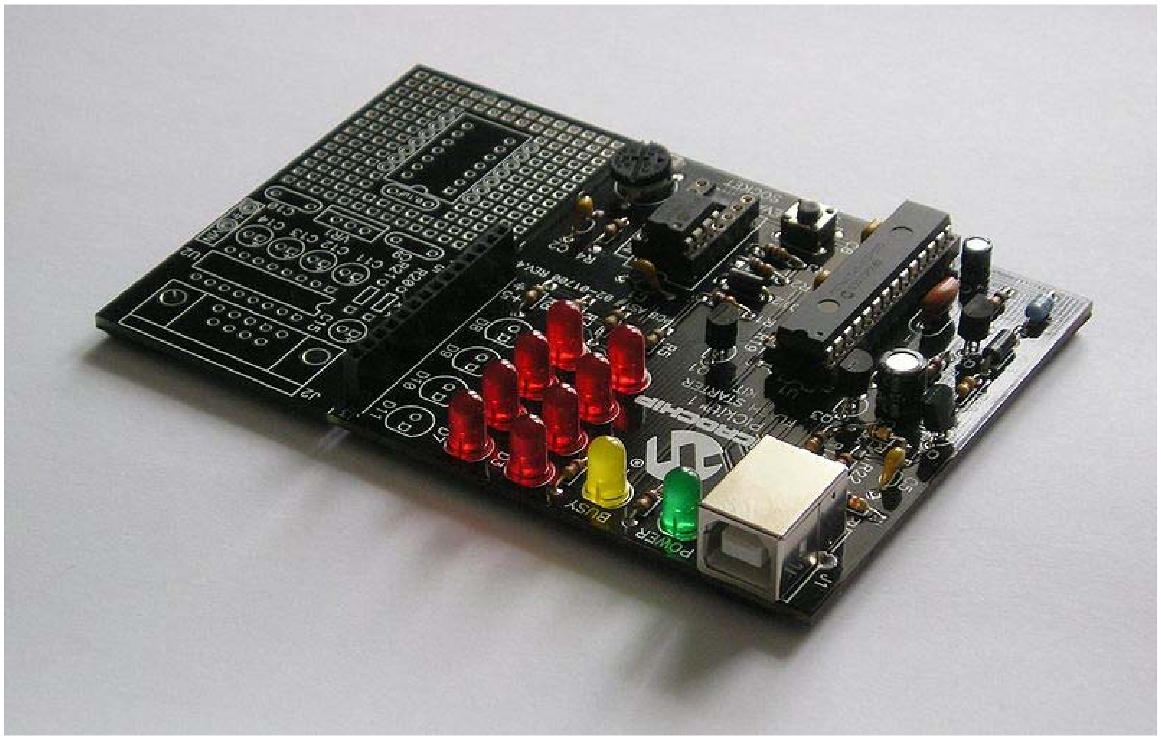
Development tools

Microchip provides a freeware IDE package called MPLAB, which includes an assembler, linker, software simulator, and debugger. They also sell C compilers for the PIC18 and dsPIC which integrate cleanly with MPLAB. Free student versions of the C compilers are also available with all features. But for the free versions, optimizations will be disabled after 60 days.

Several third parties make C language compilers for PICs, many of which integrate to MPLAB and/or feature their own IDE. A fully featured compiler for the PICBASIC language to program PIC microcontrollers is available from meLabs, Inc.

Development tools are available for the PIC family under the GPL or other free software or open sources licenses.

Device programmers


A development board for low pin-count MCU, from Microchip

Devices called "programmers" are traditionally used to get program code into the target PIC. Most PICs that Microchip currently sell feature ICSP (In Circuit Serial Programming) and/or LVP (Low Voltage Programming) capabilities, allowing the PIC to be programmed while it is sitting in the target circuit. ICSP programming is performed using two pins, clock and data, while a high voltage (12V) is present on the Vpp/MCLR pin. Low voltage programming dispenses with the high voltage, but reserves exclusive use of an I/O pin and can therefore be disabled to recover the pin for other uses (once disabled it can only be re-enabled using high voltage programming).

There are many programmers for PIC microcontrollers, ranging from the extremely simple designs which rely on ICSP to allow direct download of code from a host computer, to intelligent programmers that can verify the device at several supply voltages. Many of these complex programmers use a pre-programmed PIC themselves to send the programming commands to the PIC that is to be programmed. The intelligent type of programmer is needed to program earlier PIC models (mostly EPROM type) which do not support in-circuit programming.

Many of the higher end flash based PICs can also self-program (write to their own program memory). Demo boards are available with a small bootloader factory programmed that can be used to load user programs over an interface such as RS-232 or USB, thus obviating the need for a programmer device. Alternatively there is bootloader firmware available that the user can load onto the PIC using ICSP. The advantages of a bootloader over ICSP is the far superior programming speeds, immediate program execution following programming, and the ability to both debug and program using the same cable.

Microchip PICSTART Plus programmer

Programmers/debuggers are available directly from Microchip. Third party programmers range from plans to build your own, to self-assembly kits and fully tested ready-to-go units. Some are simple designs which require a PC to do the low-level programming signalling (these typically connect to the serial or parallel port and consist of a few simple components), while others have the programming logic built into them (these typically use a serial or USB connection, are usually faster, and are often built using PICs themselves for control). The major problem of home-made or very simple programmers is that these programmers do not comply with programming specifications and this can cause premature loss of data in the flash or EEPROM.

Debugging

Software emulation

Commercial and free emulators exist for the PIC family processors.

In-circuit debugging

Later model PICs feature an ICD (in-circuit debugging) interface, built into the CPU core. ICD debuggers (MPLAB ICD2 and other third party) can communicate with this interface using three lines. This cheap and simple debugging system comes at a price however, namely limited breakpoint count (1 on older pics 3 on newer PICs), loss of some IO (with the exception of some surface mount 44-pin PICs which have dedicated lines for debugging) and loss of some features of the chip. For small PICs, where the loss of IO caused by this method would be unacceptable, special headers are made which are fitted with PICs that have extra pins specifically for debugging.

In-circuit emulators

Microchip offers three full in circuit emulators: the MPLAB ICE2000 (parallel interface, a USB converter is available); the newer MPLAB ICE4000 (USB 2.0 connection); and most recently, the REAL ICE. All of these ICE tools can be used with the MPLAB IDE for full source-level debugging of code running on the target.

The ICE2000 requires emulator modules, and the test hardware must provide a socket which can take either an emulator module, or a production device.

The REAL ICE connects directly to production devices which support in-circuit emulation through the PGC/PGD programming interface, or through a high speed connection which uses two more pins. According to Microchip, it supports "most" flash-based PIC, PIC24, and dsPIC processors.

The ICE4000 is no longer directly advertised on Microchip's website, and the purchasing page states that it is not recommended for new designs.

PIC clones

Third party manufacturers make compatible products.

PICKit 2 open source structure and clones

PICKit 2 has been an interesting PIC programmer from Microchip. It can program all PICs and debug most of the PICs (as of May-2009, only the PIC32 family is not supported for MPLAB debugging). Ever since its first releases, all software source code (firmware, PC application) and hardware schematic are open to the public. This makes it relatively easy for an end user to modify the programmer for use with a non-Windows operating system such as Linux or Mac OS. In the mean time, it also creates lots of DIY interest and clones. This open source structure brings many features to the PICKit 2 community such as Programmer-to-Go, the UART Tool and the Logic Tool, which have been contributed by PICKit 2 users. Users have also added such features to the PICKit 2 as 4MB Programmer-to-go capability, USB buck/boost circuits, RJ12 type connectors and others.

Part number suffixes

The F in a name generally indicates the PICmicro uses flash memory and can be erased electronically. A C generally means it can only be erased by exposing the die to ultraviolet light (which is only possible if a windowed package style is used). An exception to this rule is the PIC16C84 which uses EEPROM and is therefore electrically erasable.

Chapter-6

Parallax Propeller

Parallax Propeller chip

The **Parallax P8X32 Propeller**, introduced in 2006, is a multi-core architecture parallel microcontroller with eight 32-bit RISC CPU cores.

The Parallax Propeller microcontroller, Propeller Assembly language, and Spin interpreter were designed by one person, Parallax's co-founder and president Chip Gracey. The Spin Programming language and "Propeller Tool" integrated development environment were designed by Chip Gracey and Parallax's software engineer Jeff Martin. The Propeller is known for being easy to program.

Multi-core architecture

Each of the eight 32-bit cores (called a **cog**) has an elementary CPU (the ALU of which does not directly support division) and access to 512 32-bit long words (2 KB) of instructions and data. Self-modifying code is possible and is used internally, for example by an instruction that is used to create a subroutine call/return mechanism without the need for a stack. Access to shared memory (32 KB RAM; 32 KB ROM) is controlled in round-robin fashion by an internal bus controller called the **hub**. Each cog also has access to two dedicated hardware counters and two special "video registers" for use in generating PAL, NTSC, VGA, servo-control, or other timing signals.

Speed and power management

The Propeller can be clocked using either an internal, on-chip oscillator (providing a lower total parts count, but sacrificing some accuracy and thermal stability) or an external crystal or resonator (providing higher maximum speed with greater accuracy at an increased total cost). Only the external oscillator may be run through an on-chip PLL clock multiplier, which may be set at 1x, 2x, 4x, 8x, or 16x.

Both the on-board oscillator frequency (if used) and the PLL multiplier value may be changed at run-time. If used correctly, this can improve power efficiency; for example, the PLL multiplier can be decreased before a long "no operation" wait required for timing purposes, then increased afterwards, causing the processor to use less power. However, the utility of this technique is limited to situations where no other cog is executing timing-dependent code (or is carefully designed to cope with the change), since the effective clock rate is common to all cogs.

The effective clock rate ranges from 32 kHz up to 80 MHz (with the exact values available for dynamic control dependent on the configuration used, as described above). When running at 80 MHz, the proprietary interpreted **Spin programming language** executes approximately 80,000 instruction-tokens per second on each core, giving 8 times 80,000 for 640,000 high level instructions per second. Most machine-language instructions take 4 clock-cycles to execute, resulting in 20 MIPS per cog, or 160 MIPS in total for an 8-cog Propeller.

In addition to lowering the clock rate to that actually required, power consumption can be reduced by turning off cogs (which then use very little power), and by reconfiguring I/O pins which are not needed, or can be safely placed in a high-impedance state ("tristated"), as inputs. Pins can be reconfigured dynamically, but again, the change applies to all cogs, so synchronization is important for certain designs. (Some protection is available for situations where one core attempts to use a pin as an output while another attempts to use it as an input; this is explained in Parallax's technical reference manual.)

On-board peripherals

Each cog has access to some dedicated counter/timer hardware, and a special timing signal generator intended to simplify the design of video output stages, such as composite PAL or NTSC displays (including modulation for broadcast) and VGA monitors. Parallax thus makes sample code available which can generate video signals (text and somewhat low-resolution graphics) using a minimum parts count consisting of the Propeller, a crystal oscillator, and a few resistors to form a crude DAC. The frequency of the oscillator is important, as the correction ability of the video timing hardware is limited to the clock rate. It is possible to use multiple cogs in parallel to generate a single video signal. More generally, the timing hardware can be used to implement various pulse-width modulated (PWM) timing signals.

ROM extensions

In addition to the Spin interpreter and a bootloader, the built-in ROM provides some data which may be useful for certain sound, video, or mathematical applications:

- a bitmap font is provided, suitable for typical character generation applications (but not customizable);
- a logarithm table (base 2, 2048 entries);
- an antilog table (base 2, 2048 entries); and
- a sine table (16-bit, 2049 entries).

The math extensions are intended to help compensate for the lack of a floating-point unit as well as more primitive missing operations, such as multiplication and division (this is masked in Spin but is a limitation for assembly language routines). The Propeller is a 32-bit processor, however, and these tables may not have sufficient accuracy for higher-precision applications.

Built in SPIN byte code interpreter

Spin is a multitasking high level computer programming language created by Parallax's Chip Gracey, who also designed the Propeller microcontroller on which it runs, for their line of Propeller microcontrollers.

Spin code is written on the Propeller Tool, a GUI oriented software development platform written for Windows XP. This compiler converts the Spin code into bytecodes that can be loaded (with the same tool) into the main 32 KB RAM, and optionally into the serial boot FLASH EEPROM, of the Propeller chip. After booting the propeller a bytecode interpreter is copied from the built in ROM into the 2 KB RAM of the primary COG. This COG will then start interpreting the bytecodes in the main 32 KB RAM. More than one copy of the bytecode interpreter can run in other COGs, so several Spin code threads can run simultaneously. Within a Spin code program, assembler code program(s) can be "inline" inserted. These assembler program(s) will then run on their own COG's.

Like Python, Spin uses indentation/whitespace, rather than curly braces or keywords, to delimit blocks.

The Propeller's interpreter for its proprietary multi-threaded SPIN computer language is a byte code interpreter. This interpreter decodes strings of instructions, one instruction per byte, from user code which has been edited, compiled, and loaded onto the Propeller from within a purpose-specific IDE. This IDE, which Parallax simply calls "The Propeller tool", is intended for use under the Windows operating system.

The SPIN language is a high level language. Because it is interpreted in software, it runs slower than pure Propeller assembler but can be more space-efficient (Propeller assembly opcodes are 32 bits long; SPIN directives are 8 bits long, which may be followed by a number of 8-bit bytes to specify how that directive operates). SPIN also allows users to avoid significant memory segmentation issues that must be considered for assembly code.

Mixing SPIN and assembly code is straightforward; SPIN is more appropriate for high-level logic, while assembly routines may be required for I/O routines that require exact timing.

At startup, a copy of the byte code interpreter (less than 2 KB in size), will be copied into the dedicated RAM of a cog and will then start interpreting byte code in the main 32 KB RAM. Additional cogs can be started from that point, loading a separate copy of the interpreter into the new cog's dedicated RAM (a total of eight interpreter threads can, therefore, run simultaneously). Notably, this means that at least a minimal amount of startup code *must* be SPIN code, for all Propeller applications.

Syntax

The syntax of Spin can be broken down into blocks. The blocks are as following.

VAR Holds global variables

CON Holds program constants

PUB Holds code for a public subroutine

PRI Holds code for a private subroutine

OBJ Holds code for objects

Example keywords

reboot: causes the microcontroller to reboot

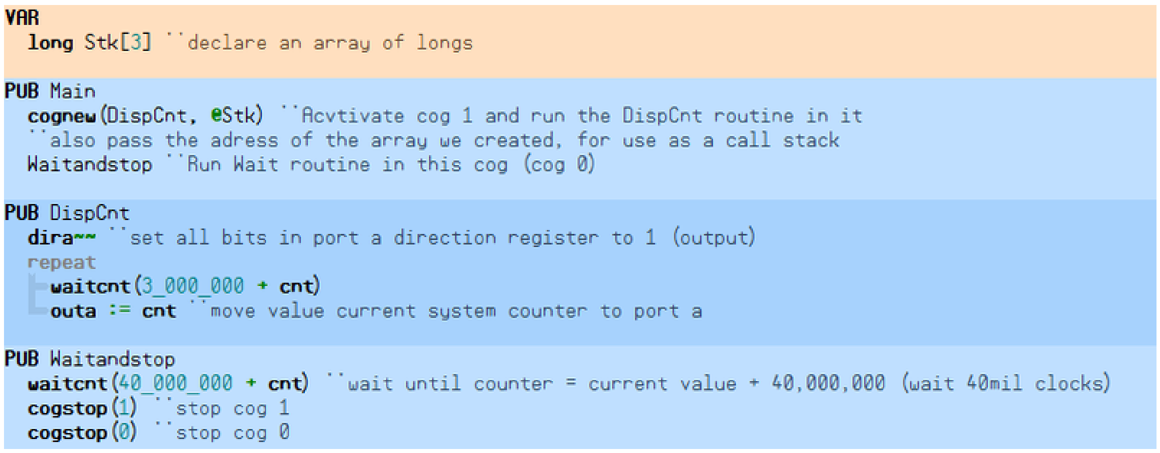
waitcnt: wait for the system counter to equal or exceed a specified value

waitvid: Waits for a video timing event before outputting video data to I/O pins.

coginit: starts a processor on a new task

Example program

An example program, (as it would appear in the "Propeller Tool" editor) which outputs the current system counter every 3,000,000 cycles, then is shut down by another cog after 40,000,000 cycles:



```
VAR
  long Stk[3] ''declare an array of longs

PUB Main
  cognew(DispCnt, @Stk) ''Acvtivate cog 1 and run the DispCnt routine in it
  ''also pass the adress of the array we created, for use as a call stack
  Waitandstop ''Run Wait routine in this cog (cog 0)

PUB DispCnt
  dira~~ ''set all bits in port a direction register to 1 (output)
  repeat
    waitcnt(3_000_000 + cnt)
    outa := cnt ''move value current system counter to port a

PUB Waitandstop
  waitcnt(40_000_000 + cnt) ''wait until counter = current value + 40,000,000 (wait 40mil clocks)
  cogstop(1) ''stop cog 1
  cogstop(0) ''stop cog 0
```

The Parallax Propeller is gradually accumulating software libraries which give it similar functionality to Parallax's older BASIC Stamp product; however there is no uniform list of which PBASIC facilities now have Spin equivalents.

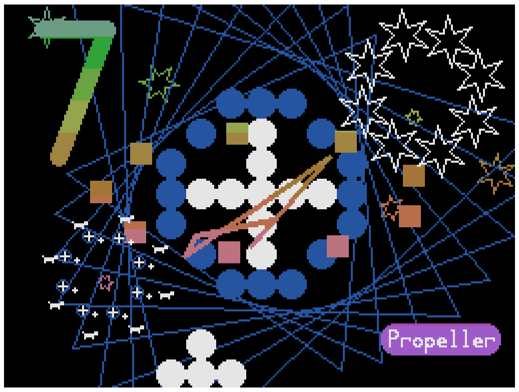
Package and I/O

The initial version of the chip (called the P8X32) provides one 32 bit port in a 40-pin 0.6" DIP, 44-pin LQFP, or QFN package. Of the 40 available pins, 32 are used for I/O, four for power and ground pins, two for an external crystal (if used), one to enable brownout-detection, and one for reset.

All 8 cores can access the 32-bit port (designated "A"; there is currently no "B") simultaneously. A special control mechanism is used to avoid I/O conflicts if one core attempts to use an I/O pin as an output while another tries to use it as input. Any of these pins can be used for the timing or pulse-width modulation output techniques described above.

Parallax has stated that it expects later versions of the Propeller to offer more I/O pins and/or more memory.

Virtual I/O devices



Screen capture of the graphics demo that Parallax created to demonstrate the NTSC library

The Propeller's designers designed it around the concept of "virtual I/O devices". For example, the "HYDRA Game Development Kit", (a computer system designed for hobbyists, to learn to develop "retro-style" video games) uses the built in character generator and video support logic to generate a virtual Video display generator that outputs VGA colour pictures, PAL/NTSC compatible colour pictures or broadcast RF video+audio in software.

The screen capture displayed here was made using a software "virtual display driver" that sends the pixel data over a serial link to a PC.

Software libraries are available to implement several I/O devices ranging from simple UARTs and Serial I/O interfaces such as SPI, I2C and PS/2 compatible serial mouse and keyboard interfaces, motor drivers for robotic systems, MIDI interfaces and LCD controllers.

Dedicated cores instead of interrupts

The design philosophy of the Propeller is that a hard real-time, multi-core architecture negates the need for dedicated interrupt hardware and support in assembly. In traditional CPU architecture external interrupt lines are fed to an on-chip interrupt controller and serviced by one or more interrupt service routines. When an interrupt occurs the interrupt controller suspends normal CPU processing and saves internal state (typically on the stack) then vectors to the designated interrupt service routine. After handling the interrupt the service routine executes a "return from interrupt" instruction which restores the internal state and resumes CPU processing.

To handle an external signal promptly on the Propeller, any one of the 32 I/O lines is configured as an input. A cog is then configured to wait for a transition (either positive or negative edge) on that input using one of the two counter circuits available to each cog. While waiting for the signal, the cog operates in low-power mode, essentially sleeping.

Extending this technique, a Propeller can be set up to respond to eight independent "interrupt" lines with essentially zero handling delay. Alternately, a single line can be used to signal the "interrupt" and then additional input lines can be read to determine the nature of the event. The code running in the other cores is not affected by the interrupt handling cog. Unlike a traditional multitasking single-processor interrupt architecture, the signal response timing remains predictable , and indeed using the term "interrupt" in this context can cause confusion, since this functionality can be more properly thought of as polling with a zero loop time.

Boot mechanism

On power up, brownout detection, software reset, or external hardware reset, the Propeller will load a machine-code boot routine from the internal ROM into the RAM of its first (primary) cog and execute it. This code emulates an I^2C interface in software, temporarily using two I/O pins for the needed serial clock and data signals to load user code from an external I^2C EEPROM.

Simultaneously, it emulates a serial port, using two other I/O pins that can be used to upload software directly to RAM (and optionally to the external EEPROM). If the Propeller does not see any commands from the serial port, it will load the user program (the entry code of which must be written in SPIN, as described above) from the serial EEPROM into the main 32K RAM. After that it will load the SPIN interpreter from its built-in ROM into the dedicated RAM of its first cog, thereby overwriting most of the bootloader.

Regardless of how the user program is loaded, execution starts by interpreting initial user bytecode with the SPIN interpreter running in the primary cog. After this initial SPIN code runs, the application can turn on any unused cog to start a new thread, and/or start assembler code routines.

External persistent memory

The Propeller boots from an external serial EEPROM; once the boot sequence completes, this device may be accessed as an external peripheral.

C compiler

There is also a C compiler available from ImageCraft, the ICCV7 for Propeller. It supports the 32K Large Memory Model, to bypass the 2K limitation per cog, and is typically 5 to 10 times faster than standard SPIN code. A free ANSI C compiler called Catalina is also available. It is based on LCC.

Propeller and Java

There is also a movement in place to put the JVM on Propeller. A compiler, debugger, and emulator are being developed.

Future versions

Parallax is currently building a new Propellerwith cogs that each will run at about 160 MIPS, whereas the current Propeller's cogs each run at around 20 MIPS. The improved performance would result from a maximum clock speed increase to 160 MHz (from 80 MHz) and an architecture that pipelines instructions, achieving an average execution of nearly one instruction per clock cycle (approx. 4x increase).

Chapter-7

Intel MCS-51 and Intel MCS-48

Intel MCS-51

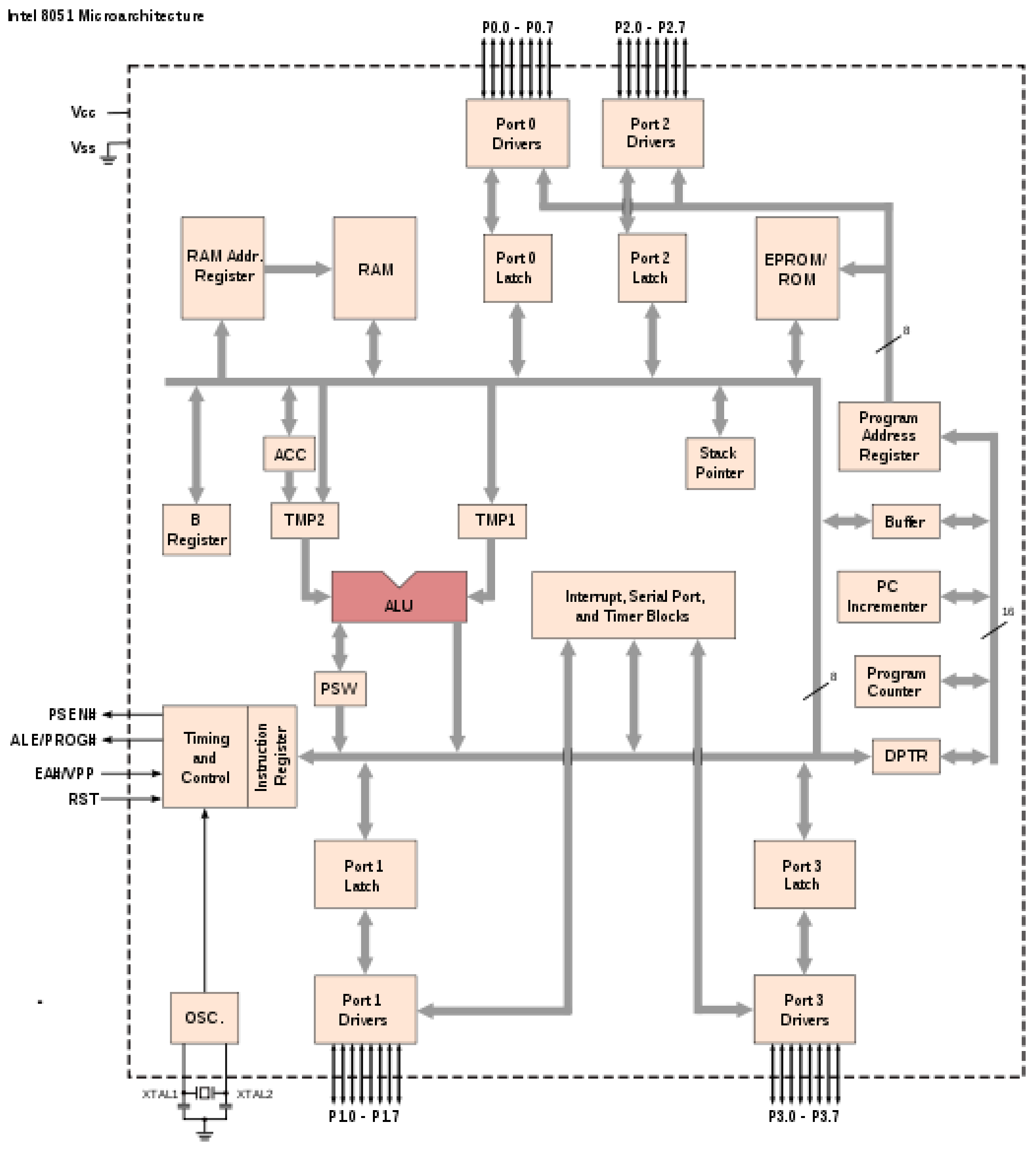
Intel P8051 microcontroller.

SAB-C515-LN by Infineon is based on the 8051

The **Intel MCS-51** is a Harvard architecture, single chip microcontroller (µC) series which was developed by Intel in 1980 for use in embedded systems. Intel's original versions were popular in the 1980s and early 1990s, but has today largely been superseded by a vast range of faster and/or functionally enhanced 8051-compatible devices manufactured by more than 20 independent manufacturers including Atmel, Infineon Technologies (formerly Siemens AG), Maxim Integrated Products (via its Dallas Semiconductor subsidiary), NXP (formerly Philips Semiconductor), Nuvoton (formerly Winbond), ST Microelectronics, Silicon Laboratories (formerly Cygnal), Texas Instruments, Ramtron International, Silicon Storage Technology, and Cypress Semiconductor.

Intel's original MCS-51 family was developed using NMOS technology, but later versions, identified by a letter C in their name (e.g., 80C51) used CMOS technology and were less power-hungry than their NMOS predecessors. This made them more suitable for battery-powered devices.

Important features and applications



i8051 microarchitecture.

The 8051 architecture provides many functions (CPU, RAM, ROM, I/O, interrupt logic, timer, etc.) in a single package

- 8-bit ALU, Accumulator and 8-bit Registers; hence it is an 8-bit microcontroller
- 8-bit data bus – It can access 8 bits of data in one operation
- 16-bit address bus – It can access 2^{16} memory locations – 64 KB (65536 locations) each of RAM and ROM
- On-chip RAM – 128 bytes (data memory)
- On-chip ROM – 4 kByte (program memory)
- Four byte bi-directional input/output port

- UART (serial port)
- Two 16-bit Counter/timers
- Two-level interrupt priority
- Power saving mode (on some derivatives)

A particularly useful feature of the 8051 core is the inclusion of a boolean processing engine which allows bit-level boolean logic operations to be carried out directly and efficiently on internal registers and RAM. This feature helped cement the 8051's popularity in industrial control applications. Another valued feature is that it has four separate register sets, which can be used to greatly reduce interrupt latency compared to the more common method of storing interrupt context on a stack.

The MCS-51 UARTs make it simple to use the chip as a serial communications interface. External pins can be configured to connect to internal shift registers in a variety of ways, and the internal timers can also be used, allowing serial communications in a number of modes, both synchronous and asynchronous. Some modes allow communications with no external components. A mode compatible with an RS-485 multi-point communications environment is achievable, but the 8051's real strength is fitting in with existing ad-hoc protocols (e.g., when controlling serial-controlled devices).

Once a UART, and a timer if necessary, have been configured, the programmer needs only to write a simple interrupt routine to refill the *send* shift register whenever the last bit is shifted out by the UART and/or empty the full *receive* shift register (copy the data somewhere else). The main program then performs serial reads and writes simply by reading and writing 8-bit data to stacks.

MCS-51 based microcontrollers typically include one or two UARTs, two or three timers, 128 or 256 bytes of internal data RAM (16 bytes of which are bit-addressable), up to 128 bytes of I/O, 512 bytes to 64 kB of internal program memory, and sometimes a quantity of extended data RAM (ERAM) located in the external data space. The original 8051 core ran at 12 clock cycles per machine cycle, with most instructions executing in one or two machine cycles. With a 12 MHz clock frequency, the 8051 could thus execute 1 million one-cycle instructions per second or 500,000 two-cycle instructions per second. Enhanced 8051 cores are now commonly used which run at six, four, two, or even one clock per machine cycle, and have clock frequencies of up to 100 MHz, and are thus capable of an even greater number of instructions per second. All SILabs, some Dallas and a few Atmel devices have single cycle cores.

Common features included in modern 8051 based microcontrollers include built-in reset timers with brown-out detection, on-chip oscillators, self-programmable Flash ROM program memory, bootloader code in ROM, EEPROM non-volatile data storage, I^{2}C, SPI, and USB host interfaces, CAN or LIN bus, PWM generators, analog comparators, A/D and D/A converters, RTCs, extra counters and timers, in-circuit debugging facilities, more interrupt sources, and extra power saving modes.

Memory Architecture

The MCS-51 has four distinct types of memory – internal RAM, special function registers, program memory, and external data memory.

Internal RAM (IRAM) is located from address 0 to address 0xFF. IRAM from 0x00 to 0x7F can be accessed directly, and the bytes from 0x20 to 0x2F are also bit-addressable. IRAM from 0x80 to 0xFF must be accessed indirectly, using the @R0 or @R1 syntax, with the address to access loaded in R0 or R1.

Special function registers (SFR) are located from address 0x80 to 0xFF, and are accessed directly using the same instructions as for the lower half of IRAM. Some of the SFR's are also bit-addressable.

Program memory (PMEM, though less common in usage than IRAM and XRAM) is located starting at address 0. It may be on- or off-chip, depending on the particular model of chip being used. Program memory is read-only, though some variants of the 8051 use on-chip flash memory and provide a method of re-programming the memory in-system or in-application. Aside from storing code, program memory can also store tables of constants that can be accessed by MOVC A, @DPTR, using the 16-bit special function register DPTR.

External data memory (XRAM) also starts at address 0. It can also be on- or off-chip; what makes it "external" is that it must be accessed using the MOVX (Move eXternal) instruction. Many variants of the 8051 include the standard 256 bytes of IRAM plus a few KB of XRAM on the chip. If more XRAM is required by an application, the internal XRAM can be disabled, and all MOVX instructions will fetch from the external bus.

Programming

There are various high-level programming language compilers for the 8051. Several C compilers are available for the 8051, most of which feature extensions that allow the programmer to specify where each variable should be stored in its six types of memory, and provide access to 8051 specific hardware features such as the multiple register banks and bit manipulation instructions. There are many commercial C compilers. SDCC is a popular open source C compiler. Other high level languages such as Forth, BASIC, Pascal/Object Pascal, PL/M and Modula-2 are available for the 8051, but they are less widely used than C and assembly.

Because IRAM, XRAM, and PMEM all have an address 0, C compilers for the 8051 architecture provide compiler-specific pragmas or other extensions to indicate where a particular piece of data should be stored (i.e. constants in PMEM or variables needing fast access in IRAM). Since data could be in one of three memory spaces, a mechanism is usually provided to allow determining to which memory a pointer refers, either by constraining the pointer type to include the memory space, or by storing metadata with the pointer.

Instruction set

The MCS-51 instruction set offers several addressing modes, including

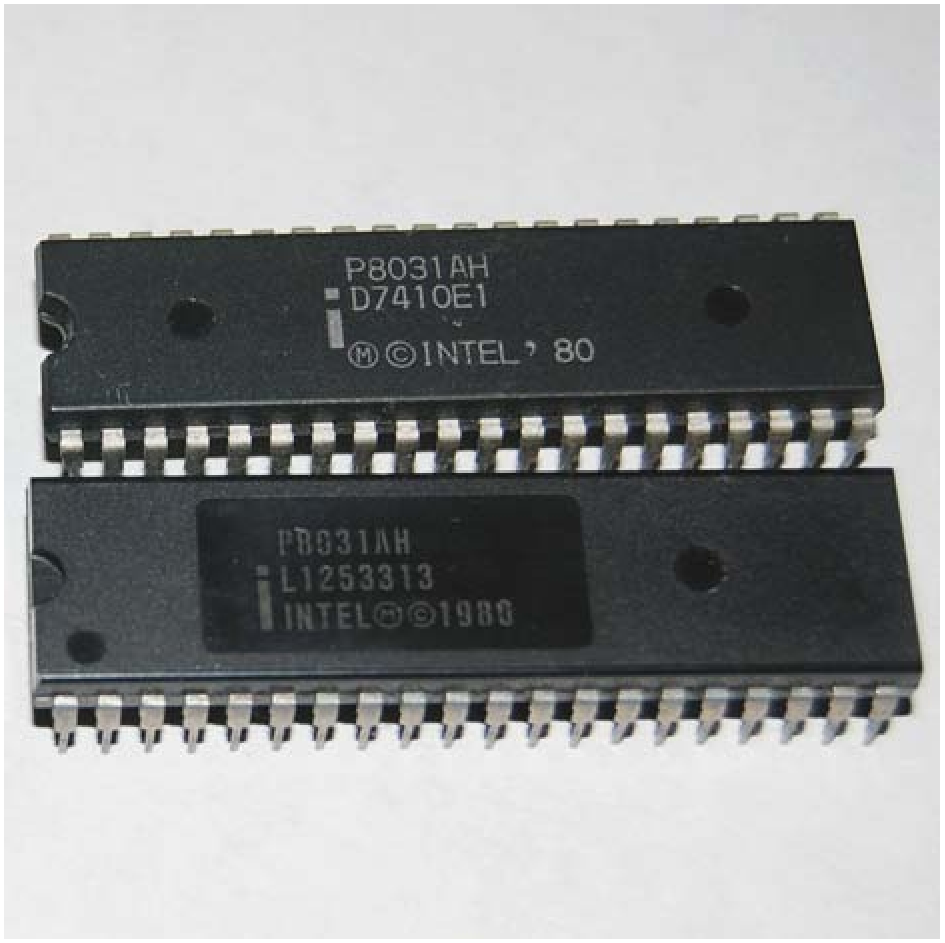
- direct register, using ACC (the accumulator) and R0-R7
- direct memory, which access the internal RAM or the SFR's, depending on the address
- indirect memory, using R0, R1, or DPTR to hold the memory address. The instruction used may vary to access internal RAM, external RAM, or program memory.
- individual bits of a range of IRAM and some of the SFR's

Many of the operations allow any addressing mode for the source or the destination, for example, MOV 020h, 03fh will copy the value in memory location 0x3f in the internal RAM to the memory location 0x20, also in internal RAM.

Because the 8051 is an accumulator-based architecture, all arithmetic operations must use the accumulator, e.g. ADD A, 020h will add the value in memory location 0x20 in the internal RAM to the accumulator.

One does not need to master these instructions to program the 8051. With the availability of good quality C compilers, including open source SDCC, virtually all programs can be written with high-level language.

Related processors



Intel 8031 processors

The 8051's predecessor, the 8048, was used in the keyboard of the first IBM PC, where it converted keypresses into the serial data stream which is sent to the main unit of the computer. The 8048 and derivatives are still used today for basic model keyboards.

The *8031* was a cut down version of the original Intel 8051 that did not contain any internal program memory (ROM). To use this chip, external ROM had to be added containing the program that the 8031 would fetch and execute. An 8051 chip could be sold as a ROM-less 8031, as the 8051's internal ROM is disabled by the normal state of the EA pin in an 8031-based design. A vendor might sell an 8051 as an 8031 for any number of reasons, such as faulty code in the 8051's ROM, or simply an oversupply of 8051's and undersupply of 8031's.

The *8052* was an enhanced version of the original 8051 that featured 256 bytes of internal RAM instead of 128 bytes, 8 KB of ROM instead of 4 KB, and a third 16-bit timer. The *8032* had these same features except for the internal ROM program memory. The 8052 and 8032 are largely considered to be obsolete because these features and more are included in nearly all modern 8051 based microcontrollers.

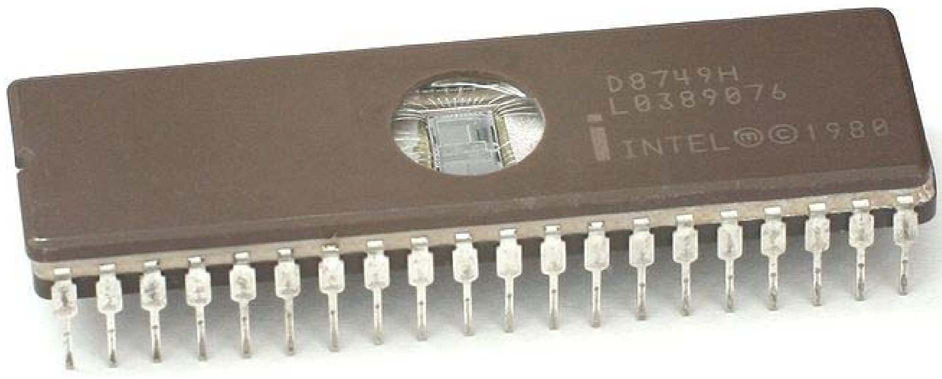
Intel discontinued its MCS-51 product line in March 2007, however there are plenty of enhanced 8051 products or silicon IP added regularly from other vendors.

The 80C537 and 80C517 are CMOS versions, designed for the automotive industry. Enhancements mostly new peripheral features and expanded arithmetic instructions. The 80C517 has fail save mechanisms, analog signal processing facilities and timer capabilities and 8 KB on-chip program memory. Other features include:

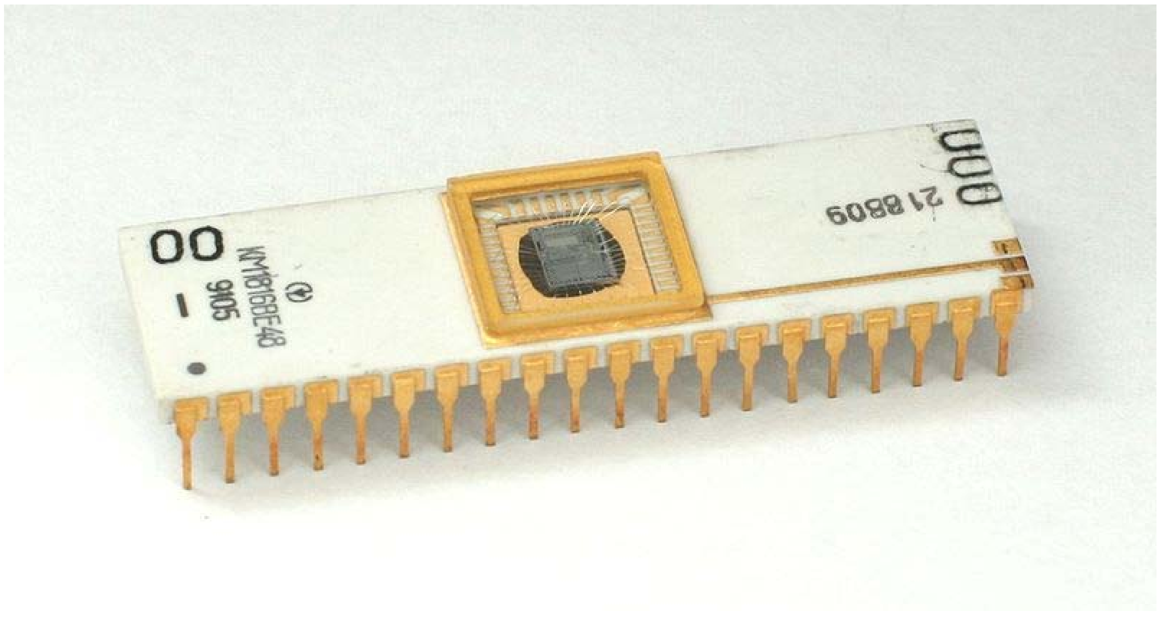
- 256 byte on-chip RAM
- 256 directly addressable bits
- External program and data memory expandable up to 64 KB
- 8-bit A/D converter with 12 multiplexed inputs
- Arithmetic unit can make division, multiplication, shift and normalize operations
- Eight data pointers instead of one for indirect addressing of program and external data memory
- Extended watchdog facilities
- Nine ports
- Two full-duplex serial interfaces with own baud rate generators
- Four priority level interrupt systems, 14 interrupt vectors
- Three power saving modes

Intel MCS-48

Intel 8048 microcontroller



The 8749 with UV EPROM



USSR KM1816BE48, an Intel 8748 clone.

The **MCS-48** microcontroller (µC) series, Intel's first microcontroller, was originally released in 1976. Its first members were 8048, 8035 and 8748.

The MCS-48 series has a Modified Harvard architecture, with internal or external program ROM and 64–256 bytes of internal (on-chip) RAM. The I/O is mapped into its own address space, separate from programs and data. The 8048 is probably the most prominent member of Intel's MCS-48 family of microcontrollers. It was inspired by, and is somewhat similar to, the Fairchild F8 microprocessor.

Though the MCS-48 series was eventually replaced by the very popular Intel MCS-51, even at the turn of the millennium it remains quite popular, due to its low cost, wide availability, memory efficient one-byte instruction set, and mature development tools. Because of this it is much used in high-volume consumer electronics devices such as TV sets, TV remotes, toys, and other gadgets where cost-cutting is essential.

Variants

The *8049* has 2 KB of masked ROM (the *8748* and *8749* had EPROM) that can be replaced with a 4 KB external ROM, as well as 128 bytes of RAM and 27 I/O ports. The µC's oscillator block divides the incoming clock into 15 internal phases, thus with its 11 MHz max. crystal one gets 0.73 MIPS (of one-clock instructions). Some instructions are single byte/cycle ones, but a large amount of opcodes need two cycles and/or two bytes, so the raw performance would be closer to 0.5 MIPS.

Another variant, the ROM-less *8035*, was used in Nintendo's arcade game Donkey Kong. Although not being a typical application for a microcontroller, its purpose was to generate the background music of the game.

The Intel *8748* has on-chip clock oscillator, 2 8-bit timers, 27 I/O ports, 64 bytes of RAM and 1 KB of EPROM. A version with 2 KB EPROM and 128 bytes RAM was also available under the *8749* number.

Device	Internal	Memory	Remarks
8020	1K x 8 ROM	64 x 8 RAM	Subset of 8048, 20 pins, Only 13 I/O Lines
8021	1K x 8 ROM	64 x 8 RAM	Subset of 8048, 28 pins ,21 I/O Lines
8022	2K x 8 ROM	64 x 8 RAM	Subset of 8048, A/D-converter
8035	none	64 x 8 RAM	
8039	none	128 x 8 RAM	
8040	none	256 x 8 RAM	
8048	1K x 8 ROM	64 x 8 RAM	
8049	2K x 8 ROM	128 x 8 RAM	
8050	4K x 8 ROM	256 x 8 RAM	
8748	1K x 8 EPROM	64 x 8 RAM	
8749	2K x 8 EPROM	128 x 8 RAM	
8648	1K x 8 OTP EPROM	64 x 8 RAM	Factory OTP EPROM
Device	Internal	Memory	Remarks
8041	1K x 8 ROM	64 x 8 RAM	Universal Peripheral Interface (UPI)
8041AH	1K x 8 ROM	128 x 8 RAM	UPI
8741A	1K x 8 EPROM	64 x 8 RAM	UPI, EPROM version of 8041
8741AH	1K x 8 OTP EPROM	128 x 8 RAM	UPI, OTP EPROM version of 8041AH
8042AH	2K x 8 ROM	256 x 8 RAM	UPI
8742	2K x 8 EPROM	128 x 8 RAM	UPI, EPROM version

```
8742AH  2K x 8 OTP EPROM  256 x 8 RAM   UPI, OTP EPROM version of
8042AH
```

Uses

The 8048 was used in the Magnavox Odyssey² video game console, the Korg Trident series, Roland Jupiter-4 and Roland ProMars analog synthesizers.

The original IBM PC keyboard used an 8048 as its internal microcontroller. The PC AT replaced the PC's Intel 8255 peripheral interface chip at I/O port addresses `0x60-63` with an 8042 accessible through port addresses `0x60` and `0x64`. As well as managing the keyboard interface the 8042 controlled the A20 line of the AT's Intel 80286 CPU, and could be commanded by software to reset the 80286 (unlike the 80386 and later processors, the 80286 had no way of switching from protected mode back to real mode except by being reset). Later PC compatibles integrate the 8042's functions into their super I/O devices.

Chapter-8

PICAXE

PICAXE is the name of a UK-sourced microcontroller system based on a range of Microchip PICs. There are 13 PICAXE variants of differing pin counts from 8 to 40 pins. Initially marketed for use in education and by electronics hobbyists, they are also used in commercial and technical fields, including rapid prototype development. All use pre-loaded factory bootstrap interpretation code to allow user generated programs to be downloaded using a simple USB or RS-232 serial connection.

Hardware

General

Based upon a variety of Microchip PICs, the chips come in a number of DIP footprints, from an 8-pin, 128-byte program capacity device through to a 40-pin, 4096-byte program capacity device. With their DIP footprint they are suited for use with solderless breadboards and more traditional PCBs designs, although surface mount versions are available.

The 8-pin PICAXE-08M is priced below GB£2 and is often chosen as an entry-level option as it lends itself to easy prototyping.

The current recommended parts, as of November 2009, are 08M, 14M, 18M, 20M, 18X, 28X1, 40X1, 20X2, 28X2, 40X2.

As of 12 August 2010, Rev Ed has released the 18M2. This chip is promoted as a replacement for the 18X, 18M.

The 28, 28A, 28X and 40X have been discontinued as they have been superseded by the X1 parts. The 18 and 18A have been discontinued as they have been superseded by the 18M. The 08, 18, 18A are still available but are not recommended for new designs.

All current devices use an internal 4/8 MHz oscillator and hence require few components to create a basic hardware platform. The X1 and X2 parts also have the ability to operate from 31 kHz to 8 MHz in 7 steps using internal low frequency oscillators in addition to the internal resonator.

The X2 parts can also operate at higher speeds from the internal oscillators. The X1 and X2 parts can also use an external resonator for up to 20 MHz (X1 parts) and up to 64 MHz for the X2 parts.

The M2 Series PICAXE Chips

Rev Ed is releasing a new M2 series of chips. The 18M2 is the first to be released which was available from 12 August 2010. The 14M2 and 20M2 will be available at a future time.

Features of the 18M2 include:

- Almost every pin is individually configurable, so, for instance, the 18M2 can now have 13 outputs instead of 8. Therefore M2 chips can be used in either the ‘traditional’ fixed pin format or the new flexible ‘user configured’ format.
- Many extra ADC channels are now also available on other pins, with new support for touch sensor inputs.
- The reset and serial input pins can now be used as 2 extra input pins, giving 2 more general purpose input pins. New software ‘reset’ command if required.
- It can now run four separate tasks in parallel, but limited to a single core. This allows BASIC programs to execute diferent tasks at the same time, as well as having four separate starts in the Logicator flowcharting program.
- The 18M2 device is meant to replace the older 18, 18A, 18M and 18X parts.
- It gives eight times the equivalent memory capacity (2048 bytes, up to 1800 lines of program) at the same price as the older 256 byte 18A/18M, this is due to the effect of the Moore's law, which states that the amount of transistor that can be placed inside an integrated circuit is doubled every two years.
- The 18M2 incorporates I^2C commands (but not 1-wire) for use with an external RTC, memory and other chips.
- Fully backwards compatible with all existing 18 pin PICAXE project boards and programs written for any older 18 pin PICAXE part.
- It can operate at 1.8V, saving considerable space and cost in portable designs. However, being this technology relatively new to the hobbyist market, most of the parts use 3.3V or 5V for operation.
- 28 general purpose byte variables, as opposed to the 14 previously available on older parts, with up to a total of 256 bytes of RAM.
- New ‘time’ variable counts elapsed seconds.
- 256 bytes of non-volatile data EEPROM memory.
- Faster internal resonator (up to 32 MHz) means up to 8x faster program processing.

- Full support for common features such as ring tone tunes, servos, digital temperature sensors and infra-red input and output on any pin, most of these using inbuilt BASIC program commands.
- Full support for advanced features like 5 bit DAC, SR latch, hardware serial commands (for much faster baud rates) and PWM control of motors.

Minimal configurations

All current devices require nothing more than the connection of power and configuration of the Serial In pin used for downloading.

The 18, 28 and 40-pin devices (but not the 18M2) requires a pull-up resistor from the Reset pin. The early PICAXE 28 and 40X chips require the connection of a resonator or crystal.

Power supply requirements

Power supply voltages are very flexible allowing operation from batteries or regulated power supplies. Most of the PICAXE range is nominally 4.5 Vdc to 5 Vdc but can operate down to approximately 3 Vdc. The X2 range also includes special low power (1.8 Vdc to 3.3 Vdc) variants.

The new M2 series PICAXE chips can operate from 1.8 Vdc to 5 Vdc

Low power modes

There are a number of commands (SLEEP, NAP, HIBERNATE, DOZE) to put the device into low power operating modes in order to conserve power and extend operational lifetime when powered by batteries.

Many of the variants have controllable clocks, allowing for operation below their nominal operating frequencies, to achieve minimal lower power consumption. Execution speed can be controlled by the user program.

Non-volatile data storage

All devices contain non-volatile data memory which allows data to be stored and recovered when power is removed. Access to the non-volatile data memory is through the use of the READ and WRITE commands and, on the 28A, the READMEM and WRITEMEM commands can also be used.

The amount of non-volatile data memory available depends upon the device:

- The 08 and 18 have 128 bytes of EEPROM which is used to store both downloaded program and for non-volatile data memory use.

- The 08M, 14M, 18M, 20M have 256 bytes of EEPROM which is used to store both downloaded program and for non-volatile data memory use.
- The 18A, 18X snd 18M2 have 256 bytes of EEPROM exclusively available for non-volatile data memory use.
- The 28 has 64 bytes of EEPROM exclusively available for non-volatile data memory use.
- The 28A has 64 bytes of EEPROM exclusively available for non-volatile data memory use and also has 256 bytes of Flash memory which may be used for non-volatile data storage using the READMEM and WRITEMEM commands.
- The 28X and 40X have 128 bytes of EEPROM exclusively available for non-volatile data memory use.
- The 28X1 and 40X1 have 4096 bytes of program space, double the variable memory and athe addition of 128 bytes of scratchpad memory. These also many new features for IO etc.
- The 20X2, 28X2 and 40X2 a slightly different arrangement with up to 4 internal program memory slots, 32 external memory slots, 1024 bytes of scratchpad memory and more versatile bidirectional IO.

PICAXE chip factory marking/identification

- PICAXE-08 PIC12F629
- PICAXE-08M PIC12F683
- PICAXE-14M PIC16F684
- PICAXE-18M PIC16F819
- PICAXE-18X PIC16F88
- PICAXE-18M2 PICAXE 18M2
- PICAXE-20M PIC16F677
- PICAXE-20X2 PIC18F14K22
- PICAXE-28X1 PIC16F886
- PICAXE-28X2 PIC18F2520 (3V version PIC18F25K20)
- PICAXE-40X1 PIC16F887
- PICAXE-40X2 PIC18F4520 (3V Version PIC18F45K20)
- PICAXE-18 PIC16F627(A) Superseded by 18M
- PICAXE-18A PIC16F819 Superseded by 18M
- PICAXE-28A PIC16F872 Superseded by 28X1
- PICAXE-28X PIC16F873A Superseded by 28X1
- PICAXE-40X PIC16F874A Superseded by 40X1

Note that the PICAXE-18M2 is a new custom part factory manufactured by Microchip Inc. for Revolution Education and so is factory engraved with the full PICAXE-18M2 name. This was done to avoid confusion in educational environments.

Software

All programming development is done using the Programming Editor or the AXEpad editor. These are both free software downloads supporting the entire development process from source code editing to program downloading. Microsoft (Windows 98 and later), Linux and Macintosh operating systems are supported. Historically a serial port was required to download programs, however a USB download cable is now available from the developers. The cable incorporates a USB-serial adapter moulded into its plug, eliminating the need for a separate adapter for users of modern USB-only PCs.

The Programming Editor and AXEpad also include a number of Wizards which aid in program and project development.

Editing can be done in plain text or RTF with color syntax highlighting. Source code can be created outside the Programming Editor and imported for downloading. Programs can be created in a Basic-like language or by using a visual flowcharting tool. The Programming Editor also supports PICmicro Assembly Language programming.

Simulator

The Programming Editor software includes a comprehensive line by line on-screen simulation of the BASIC program. This enables users to step through the program on-screen, to watch the program in operation, and help identify any programming errors.

Tune wizard

This Wizard is used to create TUNE commands which can be used with the 08M, 14M, 18M, 20M, 28X1 and 40X1. The Wizard includes the ability to import suitable mobile telephone ring tones and convert them to appropriate TUNE commands.

Serial LCD CGRAM wizard

This Wizard provides a visual means to create custom defined characters to be used with serial LCD displays. The desired character is specified by selecting pixels which will be displayed on a 7x5 grid and the necessary command to program that character within the LCD is generated.

Datalogger wizard

This Wizard is used to generate datalogging programs.

There is a Datalogger sub-wizard to take the current PC time and create a small program to set the time in a DS1307 (or compatible) RTC. The wizard will then download to the PICAXE chip and run the program. While ostensibly for the AXE110 datalogger this will also work with other i2c enabled PICAXE chips.

PICAXE connect wizard

This Wizard allows the configuration of MaxStream XBee (ZigBee) modules to be configured for use.

PICAXE net server wizard

This Wizard allows configuration of the PICAXE.net Web Server product. The Wizard provides the ability to create and upload page images and to upgrade the web server firmware.

Programming language

The programming language is BASIC-like and very similar to that used by the Basic Stamp 1 (BS1). Most programs written for the BS1 should be easily convertible for use. The most notable difference is that the BS1's POT command has been replaced by the READADC command which allows an analogue voltage to be read directly. READADC10 allows analogue inputs to be read to 10-bit resolution.

The programming language includes high-level support for underlying processor capabilities and additional functionality for various devices. All variants support a common core programming command set but not all support all commands.

Variables

The programming language provides 14 byte variables on smaller PICAXE chips, 28 byte variables for X1 and M2 parts and 56 byte variables for the latest X2 parts through the Programming Editor. This memory can be manipulated in measures of bits (as bit0-15 for smaller parts at bit0-31 for larger parts), bytes (b0-b13, b27 or b55), and 16-bit words (w0-6, w0-13 and w0-27). The bit, byte, and word variables all overlap in the same memory area. Every two byte variables overlap with a word variable, so b0 & b1 make up w0 (where b1 is the most significant byte), likewise b2 & b3 compose w1 (where b3 is the most significant byte), and so on. The bit values bit0 to bit7 overlap b0, and bit8 to bit15 overlap b1, etc (which also covers w0). This can be used to carve apart variables, or simply make the most efficient use of the limited memory given.

Variables can be given meaningful names (aliases) for use within a program through use of the SYMBOL directive. The SYMBOL directive can also be used to create named constants.

In addition to the pre-defined variables, all but the 08 have access to the some of the internal SFR (Special Function Registers) and General Purpose Registers of the PICmicro they are based upon, allowing many of the unused Registers to be used as Random Access Memory (RAM) during execution. This can be used for temporary storage of variable values (using PEEK and POKE) and allows re-use of variables within subroutines and other code sections. The smaller PICAXE parts have 48 bytes free for use, while the other parts have access to greater number of registers enhanced parts as 95/96 bytes for the 18X, 28X1 and 40X1, 112 bytes for the 28X and 40X, 72 bytes for the 20X2 and 200 bytes for the 28X2/40X2.

The PEEK and POKE commands can also be used to implement byte arrays and software stacks.

The 20X2, 28X1, 40X1 parts also have 128 bytes of 'scratchpad' memory available while the 28X2 and 40X2 parts have 1024 bytes of ‘scratchpad’ memory. The scratchpad memory can be accessed directly (PUT and GET commands) or indirectly via the scratchpad pointer '@ptr'.

The READ and WRITE commands, which allow data to be stored in and retrieved from non-volatile EEPROM memory, can be used to retain data and settings while powered down.

Arithmetical manipulation

All arithmetical operations are performed using 16-bit, unsigned, positive only operations. Variables are expanded as required to 16-bits on use by leading zero padding and results of processing are stored by truncating the resultant value to an appropriate number of bits before storage. A byte variable will have the eight least significant bits of the result stored, a bit variable will be set to the least significant bit value of the result.

Note that when using byte size variables, the internal maths are done using 16-bits so intermediate results on a line can exceed the maximum byte value of 255 but must not exceed 65535 otherwise erroneous results will occur.

Care must therefore be taken when manipulating variables and values to consider the effect of wrap-round, underflow and overflow. In particular it must be noted that a value can never be less than zero, and a byte value can never exceed 255. These issues must be considered in the implementation of FOR-NEXT loops and other looping constructs, which may, under some circumstances, never meet their terminating conditions. There is no facility to automatically report or indicate wrap-round, underflow or overflow.

All arithmetical expressions in assignments (LET) are evaluated in a strictly left-to-right manner. There is no operator precedence. Note that the keyword "LET" is optional.

In the future, the use of parenthesis for maths precedence to control calculations will be possible on the X1 and X2 parts. This is currently awaiting an update of the Programming Editor by the PICAXE makers, Rev Ed. The intention to include parenthesis was made over a year ago in the yahoo picaxe forum but to date there has been no mention of any intention.

The 28X1 and 40X1 parts have additional unary maths functions which include: Sin, Cos, Sqr, Inv, NCD, DCD, BintoBCD and BCDtoBin functions. When unary maths functions are used, they MUST be the first command on a program line. The unary functions can be followed by additional "normal" maths functions on the same line. For example:

```
  LET b1 = COS 30 + 55 / 10
```

is valid

Users should check picaxe documents before using COS, SIN etc to be sure that the table lookpup method that it uses is accurate for their purposes.

Program flow control

A number of control constructs are provided to handle program flow -

- GOTO - Redirects program execution to another location in the program.
- IF-THEN - Not the usual "IF *condition* THEN *statement*". The only *statement* IF/THEN takes is a Label: , which it does a "GOTO" when the *condition* is true. IF/THEN falls to the next line when the *condition* is false.
- BRANCH - Conditionally redirects program execution to one of a number of locations in the program (same as ON-GOTO).
- GOSUB - Redirects program execution to another location in the program, then continues after the GOSUB command following the execution of a RETURN in the subroutine.
- RETURN - Returns program execution to after the most recent GOSUB.
- SETINT - Enables polled interrupts and automatically redirects program execution to an interrupt handler when an interrupt is detected.
- SETINTFLAGS - Enables polled interrupts on certain flag-byte conditions.

```
For both SETINT and SETINTFLAGS, program execution continues from where
the program was diverted from upon execution
of a RETURN within the interrupt handler.
```

- FOR-NEXT - Repeats a section of code while a variable value is within a specified range.

A number of block structured constructs and constructs found in other BASIC language dialects exist -

- IF-THEN-ELSE-ENDIF - Selects one of two paths of execution dependent upon conditions. Also supports an ELSEIF clause.
- SELECT-CASE-ENDSELECT - Selects one of a number of paths of execution depending upon value of a variable matched against a list of conditions.
- DO-LOOP - Conditionally repeats a section of code. Allows construction of WHILE-DO and REPEAT-UNTIL loops.
- ON-GOTO - Conditionally redirects program execution to one of a number of locations in the program.
- ON-GOSUB - Conditionally calls one of a number of subroutines in the program.

Program size

The size of program permitted is dictated by the on-chip EEPROM or Flash capacity.

- The 08 and 18 have 128 bytes of program memory allowing programs of approximately 40 lines of source code.
- The 08M, 18A, 18M, 28 and 28A have 256 bytes of program memory allowing programs of approximately 80 lines of source code.
- the 18M2 has 2048 bytes allowing 800 to 1800 lines of of source code
- The X-range have 2048 bytes of program memory allowing approximately 600 lines of source code.
- The X1 parts have 4096 bytes of program memory allowing approximately 800 to 1800 lines of source code.
- The X2 parts are based upon "slots" for program space where each "slot" has 4096 bytes of program memory available.

The 20X2 has one internal slot (#0) and can use one external slot (#4) in an i2c EEPROM (24LC128 or larger).

The 28X2 and 40X2 parts have 4 internal slots (#0 to #3) and can access three external slots (#4 to #7) in an i2c EEPROM (24LC128 or larger) allowing 2000 to 3200 lines of source code over the total of 4 internal program "slots".

Because the program downloaded is stored as variable length tokens, it is not possible to easily predict the size of program which will be generated from any given source code. In particular, the program size will vary depending upon what value constants are used and which input and output pins are referenced in various commands. The Programming Editor does however provide a Syntax Check function which allows the size of program generated to be determined without having to download the program. This also allows programs to be developed and checked even without access to target hardware.

The variable length size of tokens, coupled with the inability to predict their alignment within the program memory also means that it is not possible to accurately predict the execution speed of any particular command, although typically this will be a minimum of 250 microseconds at normal 4 MHz operating speed. To date there has been no public acknowledgment by Revolution Education as to the "specific" speed of execution of their commands, and how their system may compare to compiling basic into hex directly. This makes it difficult for users to determine how well a PICAXE system compares to other forms of system design.

Subroutine nesting

For non “X” part PICAXE chips, a maximum of 16 subroutine call statements (GOSUB) is supported within a program (15 if interrupts are supported). The X, X1 and X2 ranges support up to 256 GOSUB statements.

Subroutines can be nested to a depth of 4 levels for most PICAXE chips and to a depth of 8 levels for the X1 and X2 parts. One level of depth must be reserved for interrupt handler use when interrupts are enabled.

Illusion of limited capabilities

Although these are constrained devices, with a limited number of variables, limitations in the programming command structures and, on lower-end devices, limited program memory, these limitations have not prevented many successful projects and applications from emerging. The PICAXE is increasingly being used in (and to support) many commercial products.

Cost Effectiveness

Provided as a range of devices, each offering differing capabilities and functionality, projects and designs can be tailored to meet requirements such as cost.

Users should be aware that the cost of a PICAXE, whilst being claimed to be cheap, may not in the long run be cheaper than simply programming blank PIC processors via hardware such as PicStart plus or via a home made programmer. A quick examination of the costs of PICAXE chips compared to the price of blank PICs from Microchip demonstrates this. Using multiple devices also has many advantages in system design and modularisation.

For users unfamiliar with microcontroller developments and related tool chains, the rapid development nature of the PICAXE, simplicity of use and minimal learning curve may mitigate extra cost, and this can be particularly true for users who are undertaking single development projects.

Points of Interest

PEEK and READ

Whereas most internal processing is done using 16-bits, a value read using the PEEK or READ commands is returned as an 8-bit value, and when stored in a word variable, only the least significant bits of that word will be altered. The addresses reachable by peek or limited by picaxe to only the lowest data bank and so users cannot read the entire data area.

RANDOM

The RANDOM command provides pseudo-random number generator functionality which updates a variable to a new value when it is used. The new value is determined by the existing value of the variable when the RANDOM command is used.

Because all variables are initialised to zero when power is applied or a reset occurs, the variables to be used with RANDOM should be seeded first to prevent the same sequence of random numbers being generated whenever the program is started. Seeding can be done by storing and updating a seeding value within non-volatile EEPROM memory.

A mechanism to avoid seeding is to repeatedly execute the RANDOM command (such as when waiting for an input condition) so the value it has generated will be unpredictable when it is used later. For the X1 and X2 parts, an alternative to repeated execution of the RANDOM command is to set the timer running and then use the timer variable to ‘seed’ the random command. This will give much better results.

Although RANDOM can be used with a byte variable, because such a byte value is expanded to 16-bits before the randomising function is applied, the sequence of 'random results' will be very short and produce only a limited set of values.

MIN and MAX

The MIN and MAX operators using in assignments are 'limiting operations', ensuring that the result of the expression evaluated to that point never falls below or exceeds a specified value respectively.

Although somewhat counter-intuitive, MIN can also be thought of as returning the highest of two values, and MAX can be thought of as returning the lowest of the two.

Programming interface

Programs are downloaded from the Programming Editor using a serial link, either a physical serial port or a USB to serial adapter. USB serial ports must be able to support RS232 break signalling for successful use.

The basic programming interface consists of just two resistors and does not need RS232 level converters.

Debugging facilities

- If spare output lines are available, these can be used to control LEDs or Piezo sounders to visually or audibly indicate the program's execution reaching the points where such commands are placed. The Serial Output command (SEROUT) can be used to send execution progress indicators and variable content information to a PC or other terminal device.
- Many of the variants support the SERTXD command which is equivalent to using the SEROUT command to return information about program execution but it uses the Serial Out line of the download interface alleviating the need to use a Digital Output line for this purpose.
- The DEBUG command can be used with the Debug Monitoring facilities of the Programming Editor, the contents of variables when the DEBUG command is executed can be observed.

Interfacing

Being based upon the PICmicro, the PICAXE has great versatility in interfacing. Most variants support the on-chip hardware of the underlying PICmicro.

Digital outputs

Digital Outputs can each sink and source around 20mA and are capable of driving LEDs and other small loads directly. Be aware that while each output can handle 20mA there are limits for each port and for the entire chip that typically prevents 20mA being used on every output. For some PICAXE chips, the port and total limit is around 95mA while for others the limit can be around 200mA. A review of the datasheet for the core PICmicro chip is recommended.

Digital inputs

Most Digital Inputs are protected by diode clamps to the power rails, which as well as offering good ESD protection, allows interfacing to high voltage signals often with little more than a current limiting resistor being required.

This is particularly useful for interfacing to PCs, PDAs and terminals where a complete serial interface can be implemented using just one resistor. This allows for low-cost designs which do not necessitate the use of RS232 level conversion circuits, although such converters can be used to provide a more traditional interface if desired.

The voltage required to make a 1 or 0 on each type of pin is listed below.

TTL (Vsupply > 4.5V)

Vih : >= 2.0V Vil : <= 0.8V

TTL (Vsupply <= 4.5V)

Vih : >= 0.25 * Vsupply + 0.8V Vil : <= 0.15 * Vsupply

Schmitt Trigger (ST)

Vih : >= 0.8 * Vsupply (>= 4V @ 5V) Vil : <= 0.2 * Vsupply (<= 1V @ 5V)

Bi-directional inputs and outputs

The 18, 18A, 18M, 18X, 20M, 28 and 28A have fixed direction Input and Output pins. That is to say that every pin is pre-defined as either an input or an output and this use cannot be explicitly changed under program control.

The 08 and 08M have three bi-directional pins which can be either input or output pins and can be explicitly set as such and changed under program control. The 14M has nine bi-directional pins when used in its advanced configuration.

The 28X/X1 and 40X/X1 also have eight 'PORTC' lines which can be made inputs or outputs under program control. The 28X/X1 and 40X/X1 have another eight "PortB' lines which are always defined as output. Some of the additional pins/legs on the 40X/X1 chips provide a further eight 'PortD' lines which are always set as inputs.

At the start of program execution (after power is applied or reset occurs) all of the bi-directional pins will automatically be set as inputs and only become outputs when instructed by the executing program.

Because it is not always clear what program has previously been downloaded into a PICAXE which supports bi-directional pins, care must be taken when inserting those into alternative hardware. It is recommended that any existing program be erased before it is inserted into hardware different to that which it was previously used with. Inadvertently, or deliberately, making an input line an output line may have damaging effects upon the chip itself and any hardware connected to those pins.

Interrupts

The 08M, 18A, 18M and all X parts support hardware interrupting on a pattern match with the input pins. But this isn't actually the case in practice. According to picaxe documents, the picaxe "polls" the inputs and responds to the change in pin status via firmware. Users should be aware of the difference between programming a PIC microcontroller to respond to changes in the PORTB input register hardware and any claims that Picaxe supports "Hardware Interrupts" using firmware to simulate the same.

Analogue inputs

Analogue input is supported across the entire range. The 08 and 18 support only coarse, low-resolution analogue inputs which is delivered as one of 16 8-bit levels. The others support full 256-level, 8-bit analogue input, and some offer 1024-level, 10-bit inputs.

The number of analogue inputs depends on the actual device, and in some cases the analogue inputs and digital inputs share the same pins, allowing them to be used as either analogue or digital inputs, and with some design, both.

Analogue outputs

Unlike PICs that are capable of producing an analogue output voltage via its internal ladder (requiring external hardware to amplify the signal), there is no ability to generate an analogue output voltage directly, however analogue voltages can be produced by using PWM output capabilities plus the addition of suitable circuitry.

The new 18M2 includes one DAC with a 5-bit resolution giving 32 discrete voltage steps at the DAC output pin.

Dallas/Maxim one-wire device/network connectivity

The firmware supports interfacing with the Dallas/Maxim type 1-Wire devices, including the iButton range of sensors and logger devices together with various 1-Wire chips such as the DS18B20 temperature Sensor, DS243x series EEPROMs, DS2406, DS2408 and DS2413 programmable IO chips with 2, 8 and 2 channels respectively and many more 1-Wire devices.

The firmware for virtually all PICAXE chips provides the ability, through the READOWSN command, to read the unique serial number of any single 1-Wire device connected to a PICAXE input pin, whether or not the particular PICAXE chip firmware includes the additional commands to use that 1-Wire device's capabilities.

The READTEMP and READTEMP12 commands available on virtually all PICAXE chips provides the ability to easily read the temperature from a single DS18B20 temperature sensor connected to a PICAXE input pin.

The OWIN and OWOUT commands which are available only in the firmware of the more recently produced PICAXE chips, such as the X1 and X2 parts, enables communications and use of the capabilities of virtually all available 1-Wire devices including 1-Wire networks comprising many devices connected to a single PICAXE input pin.

I²C

All of the X Parts (18X, 20X2, 28X, 28X1, 28X2, 40X, 40X1 and 40X2) have the capability to use the in-built I²C commands for reading and writing to most I²C devices.

The new 18M2 chip also has i2c communications capability.

Prior to firmware revision 8.6 on the 18X, and 7.7 on the 28X and 40X, the device being accessed must require at least two data bytes (3 bytes total including the address byte) to be able to be used with the built-in commands. Starting with the above mentioned firmware revisions, and the latest editor software, devices that only require two bytes (an address byte and a data byte) can be used.

Some limitations with the firmware controlled I²C functions are that there is no control beyond read and write commands. Protocol options like repeated start, early stop, attention are not supported. By bit-banging the signals, however, the entire gamut of I²C options can be used, but this requires an in depth familiarity with the I²C protocol itself.

The X1 and X2 parts can also be operated as I²C slave devices.

Infrared

For the non X1/X2 parts, there are three commands to support infrared interfacing -

- INFRAIN - Allows reception of infrared signals which use Sony's SIRC protocol. This command is tailored to supporting key press codes which would be sent from a TV remote control and supports only a subset of possible remote control commands.
- INFRAIN2 - Allows reception of infrared signals which use Sony's SIRC protocol. This command allows the reception of any remote control command.
- INFRAOUT - Allows the generation of an infrared signal using Sony's SIRC protocol.

The INFRAIN and INFRAIN2 commands require that an external infrared sensor (TSOP18) is used to demodulate the incoming infrared signal for processing. INFRAOUT can be used to drive an infrared LED (and optional visible light LED) directly.

Even without the INFRAIN and INFRAIN2 commands, infrared signals which use Sony's SIRC protocol can be read by means of bit-banging; sampling the signal provided by the infrared sensor and determining the remote control command sent. This technique is also applicable for decoding infrared signals sent using other protocols, but is not suitable for all such protocols.

For the 28X1 and 40X1 parts, there are two different commands (keywords) with slightly different functionality:

- IRIN - This command is similar to the 'infrain2' command found on other PICAXE devices, but can be used on any input pin. There is also an optional timeout feature
- IROUT - This command is similar to the 'infraout' command found on other PICAXE devices, but can be used on any output pin.

LCD

All variants can interface to serially controlled LCDs using the SEROUT command providing they use a baud rate which is supported.

All but the 08 and 08M can directly control Hitachi HD44780 and similar LCDs operating in parallel, 4-bit mode. The 08 and 08M have too few output lines to support direct interfacing but can connect to such displays through additional interface circuitry, for example by using shift registers. Such interfacing schemes can always be used instead of direct parallel connections.

PC keyboard

The 18A, 18M, 28A and the X-range all provide the ability to interface to a PC keyboard using the KEYIN and KEYLED commands.

These two commands respectively determine which key was pressed on a PC keyboard through its scan code and control the setting of the keyboard LEDs. The KEYLED command can be used to flash the keyboard LEDs to indicate when a key press has been received.

PWM output

The 08 and 08M support the PWM command and the 08M, 18M and the X-range support the PWMOUT command.

The PWM command provides for a burst of a pulse-width modulated signal to be generated which may be used to charge a simple Resistor-Capacitor circuit to generate an analogue voltage output. PWM can be used with any Digital Output pin.

The PWMOUT command allows a pulse-width modulated signal to be continually generated while execution continues, which may be used to generate analogue voltages, as with PWM, but can also be used for speed control of motors and brightness control of visual indicators. PWMOUT can only be used with certain Digital Output pins. The 08M and 18X provide only a single PWMOUT output, while the 28X and 40X provide two.

For the 14M, the 28X1 and 40X1 parts, there are different PWM commands.

- HPWM - (which stands for hardware PWM) allows selection of the mode, polarity, setting period and duty. Hardware PWM is an advanced method of motor control using PWM methods. HPWM can provide up to 4 separate pins with a PWM output as defined by the PIC microcontroller’s internal pwm hardware.

hpwm can be used instead of, not at the same time as, the pwmout command on 2. However pwmout on 1 can be used simultaneously if desired.

HPWMDUTY – On the X1 and X2 parts only, the hpwmduty command can be used to alter the hpwm duty cycle without resetting the internal timer (as occurs with a hpwm command). A hpwm command must be issued before this command will function.

Serial

User programmed serial input and output is implemented in the firmware, allowing the serial communications to be received on any Digital Input pin and transmitted on any Digital Output Pin. The serial interface supports a variety of baud rates between 300 and 4800 baud when operating at 4 MHz, up to 9600 baud at 8 MHz (M, A, and X parts), and up to 19,200 baud at 16 MHz (28X and 40X only).

The X1 and X2 parts also support background serial in, and much faster serial out bauds on the hardware serial pins.

On the 18X, 28X and 40X access to the on-chip AUSART through the use of PEEK and POKE allows bytes to be sent and received at higher baud rates. Those which allow access to the AUSART are capable of transmitting MIDI compatible data and potentially DMX-512 data streams.

Servo control

All except the 08 and 18 can simultaneously support the control of multiple servos connected to any of its Digital Output pins.

Up to eight servos can be controlled directly using the SERVO command. Once the servo positions have been specified, the required signal stream for each servo is sent while program execution continues.

Using SERVO has a severe affect on other time based commands, eg pause/sound. Users should be aware that the PICAXE does not take into consideration the servo pulse length delay when processing delay loops so any time critical functions are bound to be longer than specified in code. A quick test to enable several servos, and then try to compute a valid delay should confirm this.

Sound and tune control

The SOUND command allows tone generation on any of the Digital Outputs. Piezo sounders can be directly connected to the Digital Output pins and loudspeakers can be driven through a simple amplifier circuit.

The 08M, 14M, 18M, 20M, 20X2, 28X1, 28X2, 40X1 and 40X2 have the PLAY command which allows the playing of between one and four pre-programmed tunes depending upon which PICAXE chip is used. For the 08M only, there is capability to optionally flash LEDs in step with the tune's beat.

The next command available for the 08M, 14M, 18M, 20M, 20X2, 28X1, 28X2, 40X1 and 40X2 is the TUNE command. Whereas the SOUND command provides for the frequency of a tone (or white noise) and its duration to be specified, the TUNE command allows the tone to be user specified in terms of a musical note and the tempo of the tune to be specified.

The Programming Editor includes a Wizard to convert suitable mobile phone ringtones to an equivalent TUNE command with the necessary data.

SPI

Native support for SPI and similar interfacing is provided on the X1 and X2 and can be implemented by direct control of the digital output lines, using a technique known as bit-banging.

Chapter-9

Intel i960 and MIDIbox

Intel i960

Intel i960

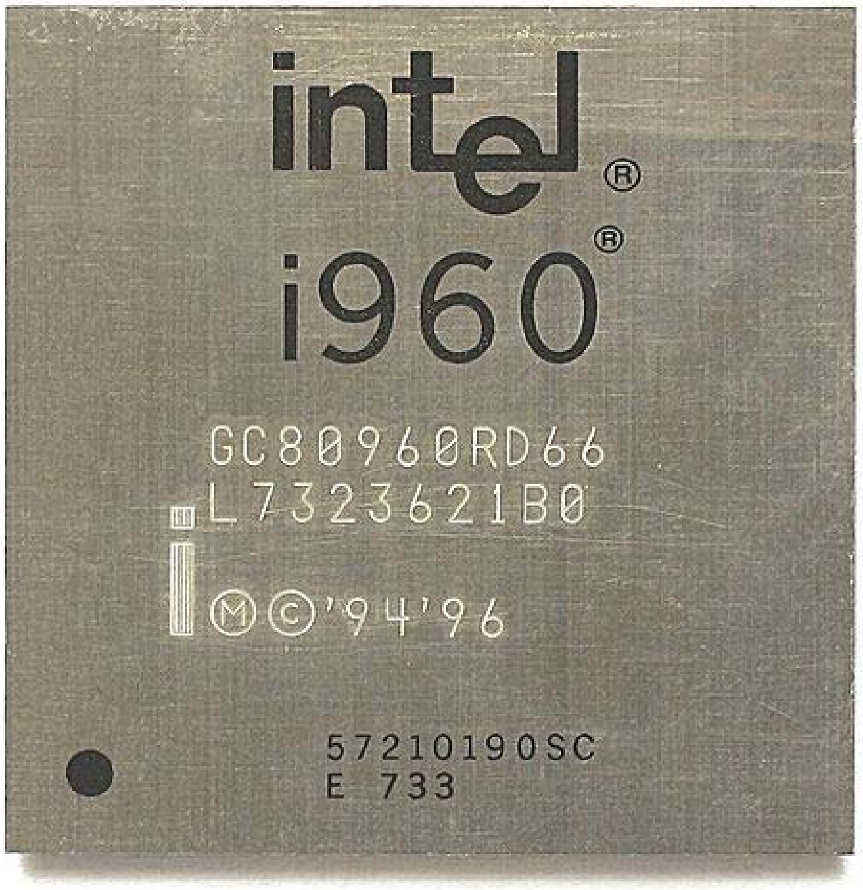
Intel i960HA microprocessor

Produced	From 1984 to late 1990s
Common manufacturer(s)	Intel
Max. CPU clock rate	10 MHz to 100 MHz
Cores	1

Intel's **i960** (or **80960**) was a RISC-based microprocessor design that became popular during the early 1990s as an embedded microcontroller, becoming a best-selling CPU in that field, along with the competing AMD 29000. In spite of its success, Intel dropped i960 marketing in the late 1990s as a side effect of a settlement with DEC in which Intel received the rights to produce the StrongARM CPU. The processor continues to be used in a few military applications.

Origin

Intel N80960SA (PLCC Package)



Intel GC80960RD66 (BGA Package)

Intel GC80960RN, sSpec: SL3YW, BGA Package

Intel FC80960HD66 (PQFP Package)

The i960 design was started as a response to the failure of Intel's iAPX 432 design of the early 1980s. The iAPX 432 was intended to directly support high-level languages that supported tagged, protected, garbage-collected memory — such as Ada and Lisp — in hardware. Because of its instruction-set complexity, its multi-chip implementation, and design flaws, the iAPX 432 was very slow in comparison to other processors of its time.

In 1984 Intel and Siemens started a joint project, ultimately called BiiN, to create a high-end fault-tolerant object-oriented computer system programmed entirely in Ada. Many of the original i432 team members joined this project, though a new lead architect was brought in from IBM, Glenford Myers. The intended market for the BiiN systems were high-reliability computer users such as banks, industrial systems and nuclear power plants.

Intel's major contribution to the BiiN system was a new processor design, influenced by the protected-memory concepts from the i432. The new design included a number of features to improve performance and avoid problems that had led to the downfall of the i432, which resulted in the i960 design. The first 960 processors entered the final stages of design, known as *taping-out*, in October 1985 and were sent to manufacturing that month, with the first working chips arriving in late 1985 and early 1986.

The BiiN effort eventually failed, due to market forces, and the 960MX was left without a use. Myers attempted to save the design by outlining several subsets of the full capability architecture created for the BiiN system. He tried to convince Intel management to market the i960 (then still known as the "P7") as a general-purpose processor, both in place of the Intel 80286 and i386 (which taped-out the same month as the first i960), as well as the emerging RISC market for Unix systems, including a pitch to Steve Jobs's for use in the NeXT system. Competition within and outside of Intel came not only from the i386 camp, but also from the i860 processor, yet another RISC processor design emerging within Intel at the time. Myers was unsuccessful at convincing Intel management to support the i960 as a general-purpose or Unix processor, but the chip found a ready market in early high-performance 32-bit embedded systems.

Architecture

To avoid the performance issues that plagued the i432, the central i960 instruction-set architecture was a RISC design, only implemented in full in the **i960MX**, and the memory subsystem was made 33-bits wide — for a 32-bit word and a "tag" bit to indicate protected memory. In many other ways the i960 followed the original Berkeley RISC design, notably in its use of register windows, an implementation-specific number of caches for the per-subroutine registers, allowing for fast routine calls. The competing Stanford University design, commercialized as MIPS, did not use this system, relying on the compiler to generate optimal subroutine call and return code instead. Unlike the i386, but in common with most 32-bit designs, the i960 has a flat 32-bit memory space, with no memory segmentation. The i960 architecture also anticipated a superscalar implementation, with instructions being simultaneously dispatched to more than one unit within the processor.

i960 variants

The "full" **i960MX** was never released for the non-military market, but the otherwise identical **i960MC** was used in high-end embedded applications. The i960MC included all of the features of the original BiiN system, but these were simply not mentioned in the literature, leading many to wonder why the i960MC was so large and had so many pins labeled "no connect".

80960Kx

A version of the RISC core without memory management or an FPU became the **i960KA**, and the RISC core with the FPU became the **i960KB**. The versions were, however, all identical internally — only the labeling was different. This meant the CPUs were much larger than necessary for the "actually supported" feature sets, and as a result, more expensive to manufacture than they needed to be.

The **i960KA** became successful as a low-cost 32-bit processor for the laser-printer market, as well as for early graphics terminals and other embedded applications. Its success paid for future generations, which removed the complex memory sub-system.

80960Cx

The **i960CA**, first announced in July 1989, was the first pure RISC implementation of the i960 architecture. It featured a newly-designed superscalar RISC core and added an unusual addressable on-chip cache, but lacked an FPU and MMU, as it was intended for high-performance embedded applications. The i960CA is widely considered to have been the first single-chip superscalar RISC implementation. The C-series only included one ALU, but could dispatch and execute an arithmetic instruction, a memory reference, and a branch instruction at the same time, and sustain two instructions per cycle under certain circumstances. The first versions released ran at 33 MHz, and Intel promoted the chip as capable of 66 MIPS. The i960CA microarchitecture was designed in 1987–1988 and formally announced on September 12, 1989. Later, the *i960CF* included a floating-point unit, but continued to omit an MMU.

80960Jx

The **80960Jx** is a processor for embedded applications. It features 32-bit multiplexed address/data bus, instruction and data cache, 1K on-chip RAM, interrupt controller and two independent 32-bit timers. The 80960Jx's testability features included ONCE (on-circuit emulation) mode and boundary scan (JTAG).

80960VH

Announced October 1998 i960VH Embedded-PCI processor featured 32-bit 33MHz PCI bus and 100MHz i960JT processor core. The core also featured 16KB of instruction cache, 4KB of data cache and 1KB of built-in RAM. Other core features included two 32-bit timers, programmable interrupt controller, I²C interface and a two-channel DMA controller.

Demise

Intel attempted to bolster the i960 in the I/O device controller market with the I2O standard, but this had little success and the design work was eventually ended. By the mid 1990s its price/performance ratio had fallen behind competing chips of more recent design, and Intel never produced a reduced power-consumption version that could be used in battery-powered systems.

In 1990 the i960 team was redirected to be the "second team" working in parallel on future i386 implementations — specifically the P6 processor, which later became the Pentium Pro. The i960 project was sent to another, smaller development team, essentially ensuring its ultimate demise.

Current status

Because of its high performance in calculating XOR values, the Intel 960 processor family is often used to control higher-end, RAID capable SCSI disk array host adapter cards. An i960RS chip also powers Adaptec's AAR-2400A controller, which uses four commodity parallel ATA drives to build an affordable RAID-5 protected fault tolerant storage system for small PC servers and workstations.

The Intel 960 was also used in some Brocade Fibre Channel switches to run Fabric OS.

The Intel 960 architecture is also used in slot machines. Currently they are found in IGT's Stepper S2000 family and i960 video family. It was also used as the main CPU of Sega's famous Model 2 series of arcade boards.

The indigenously developed Indian HAL Tejas light combat aircraft's MMR (multi-mode radar) is said to use the i960. Full adoption of the HAL Tejas into Indian Air Force service might only occur around 2010.

The Indian Space Research Organisation is said to use the chip in its on board computers in its launch vehicles.

Intel 960 processor is also used in Automatic Radar Plotting Aid (ARPA) interfacing boards of radars from Kelvin Hughes.

It was also used on some HP X-Terminals.

Some SATA RAID controllers use Intel 80303 IOP (Intelligent I/O Processor) which integrates PCI-to-PCI bridge, memory controller and 80960JT-100 CPU core.

MIDIbox

The **MIDIbox** project is an open source modular DIY framework (hardware and software) MIDI platform built around the PIC family of microcontrollers (specifically the PIC18F452, PIC16F88, PIC18F4620 and PIC18F4685) and recently with STM32 32-bit ARM Cortex too. It can be used to build hardware MIDI control units for various synthesizers, multi-track recording software, and other MIDI devices; as well as stand-alone synthesizers, sequencers and other projects.

History

The MIDIbox Hardware Platform is the continuation of Thorsten Klose's earlier work on MIDI controllers. Designs are based around a standardized environment of reusable and exchangeable modules. Soon after the release of the first modules, a small group of enthusiasts formed which grew into a thriving open source development community.

The MIDIbox Hardware Platform (MBHP)

The focus of the platform is on well defined and documented modules based on small, uncomplicated circuits to allow for amateur assembly. These modules are then assembled into a complete project. All boards can be made as single-layer PCBs, and prototype boards designed with a freeware CAD program. Almost all components are through-hole for easier assembly.

The MIDIbox hardware platform runs its own open-source operating system: MIOS (MIDIbox Operating System), written in PIC assembly language for speed and accuracy. There is a C wrapper layer available for simplified coding. MIOS is designed and documented to allow simple reconfiguration, adaptation and extension by hobbyists and enthusiasts.

The modules

Currently there are about 15 separate modules available:

Microcontroller modules

- Core Module
- PIC Programmer Modules like an actual PIC-Burner or the JDM Module

Input modules

- AIN Module Analog Input (0-5V)
- DIN Module Digital Input (ON/OFF)

Output modules

- DOUT Module Digital Output (eg. LED ON/OFF)
- LCD Module Liquid Crystal Display
- AOUT Module Analog Out to output Voltages (for Controls)

Sound modules

- SID Module for the MOS Technology SID (as found in the Commodore 64)
- OPL3 Module for the FM-Chips YMF262 and YAC512
- IIC SpeakJet Module for the SpeakJet SoundChip

Memory expansion modules

- BankStick 32k / 64k Memory module

MIDI I/O modules

- LTC Module MIDI LED Indicators + 1 MIDI-Out + 1 Thru (+ 1 optional to-COM-Port)
- USB Modules PC/USB Interface

Miscellaneous modules

- MF Module to control Motorfaders
- IIC Modules to communicate to other (Microcontroller-)Devices via I2C

The MIDIbox Operating System (MIOS)

The MIDIbox Operating System (MIOS) has been developed to allow the design of flexible MIDI controller applications. MIOS adheres to a non-commercial, open platform as fundamental to the exchange of ideas and personal adaptations which are not possible with commercial controllers.

Most controllers built by the community are based on existing documented designs, and begin life with the feature set provided by the existing firmware. End users can enhance their devices with exchangeable program code, and customize them to suit their host application, synthesizer or other MIDI device. Users can also customize to suit their own preferred workflow, or design a new project from scratch.

Application source code, module schematics and PCB layouts are available free for non-commercial use as templates for modifications and improvements. Thus MIOS and the Hardware Platform allow an easy entry to hobbyist microcontroller development, while making possible applications outside the realms of the commercial, mainstream MIDI market.

MIOS was licensed under the GPL until version 1.8. Later versions now require Thorsten Klose's permission for commercial use.

Specifications

The operating system consist of a kernel that provides user hooks to hardware and software events, and functions for interaction with Hardware Platform modules.

One core module with a PIC18F452 microcontroller can handle

- up to 128 digital inputs
- up to 128 digital outputs
- up to 64 analog inputs
- character and graphical LCDs
- up to 8 BankSticks (I2C EEPROMs)
- one MIDI In and one MIDI Out, or an RS232 serial COM port

Background drivers are available for the following control tasks:

- MIDI I/O processing
- Bootstrap loader
- Analog conversion of up to 64 pots, faders or other analog sources with a 10-bit resolution
- Motor handling for up to 8 motorized moving faders with a 10-bit resolution
- Handling of up to 64 rotary encoders
- Handling of up to 128 buttons, touch sensors or similar digital input devices
- Handling of up to 128 LEDs, relays, Digital-Analog-Converters or similar output devices. In multiplex mode a nearly unlimited number of LEDs, LED rings and LED digits can be driven
- Read/Write from/to EEPROM, Flash, and BankStick
- Linking PIC18F Core modules via MIDIbox Link

The whole operating system has been written in assembly language and has been optimized for speed. MIOS currently uses 8k of program memory and 640 bytes of RAM.

Only 75 µs is required to read 128 digital input pins and to write to 128 output pins. 16 rotary encoders are handled within 100 µs. Analog inputs are scanned in the background every 200 µs; changes larger than a definable minimum range trigger a user hook.

Up to 256 MIDI events can trigger dedicated functions; processing of the event list requires about 300 µS. MIDI events can also be processed by a user routine for sysex parsing or similar jobs. A user timer is available for time triggered code.

Support for other high-level languages apart from C is possible.

MIOS hardware

MIOS is a dedicated operating system for the Microchip Technology PIC18F452 microcontroller. This PIC is pin compatible to the PIC16F877 which was used in early

MIDIbox projects. Thus it is backwards compatible with older MIDIbox Core modules, with one board modification.

The PIC18F452 features more internal flash, much more internal RAM, some new instructions and a better system architecture. It is available for the same price as the PIC16F877 in most countries.

Complete solutions

At this point there are 11 fully-documented projects available, as well as a large number of user projects generated by the community. The official projects are as follows:

- **MIDIbox SEQ V3:**

16 Track Live Step and Morph Sequencer + advanced Arpeggiator

- **MIDIbox SID V1:**

Hardware MIDI-controllable Synthesizer based on the MOS Technology SID (MOS6581) sound chip as shipped with the Commodore 64/128

- **MIDIbox FM V1:**

Hardware synthesizer based on the Yamaha YMF262 sound chip (also known as OPL3) for generating the famous FM sounds known from Soundblaster (compatible) soundcards of the early 90s

- **MIDI Merger V1:**

Merges two separate MIDI inputs to a single output

- **MIDI Router V1:**

Routes various MIDIboxes to a single MIDI port

- **MIDI Processor:**

Provides basic functionality to receive and transmit MIDI events

- **MIDIbox CV**

Provides CV and gate outputs to drive voltage controlled devices such as analog modular synthesizers

- **MIDIbox 64:**

Full-fledged 64 channel MIDI controller

- **MIDIbox 64E V2:**

Extended version of the MIDIbox 64

- **MIDIO128 V2:**

The MIDIO128 interface is used to drive up to 128 digital output pins and to react on up to 128 digital input pins via MIDI

- **MIDIbox LC V1:**

Alternative to the MIDIbox 64/64E

- **MIDImon V2:**

Reports events, which are transmitted over the MIDI cable, in a readable form

Chapter-10

List of Common Microcontrollers

This is a **list of common microcontrollers** listed by brand.

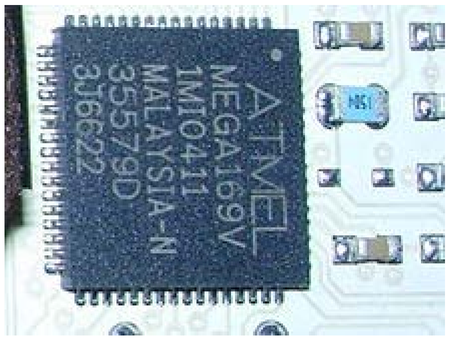
AMCC

Until May 2004, these µCs were developed and marketed by IBM, whose 4xx family was sold to Applied Micro Circuits Corporation.

- PowerPC 403
  - PPC 403GCX
- PowerPC 405
  - PPC 405EP
  - PPC 405GP/CR
  - PPC 405GPr
  - PPC NPe405H/L
- PowerPC 440
  - PPC 440GP
  - PPC 440GX
  - PPC 440EP/EPx/GRx
  - PPC 440SP/SPe

Altera

- Nios II 32-bit configurable soft microprocessor
- Nios 16-bit configurable soft processor



Atmel ATmega169 (64-pin MLF).

Analog Devices

- Blackfin
- Super Harvard Architecture Single-Chip Computer (SHARC)
- TigerSHARC
- ADSP-21xx digital signal processor
- MicroConverter Family - ARM7 and 8052 cores

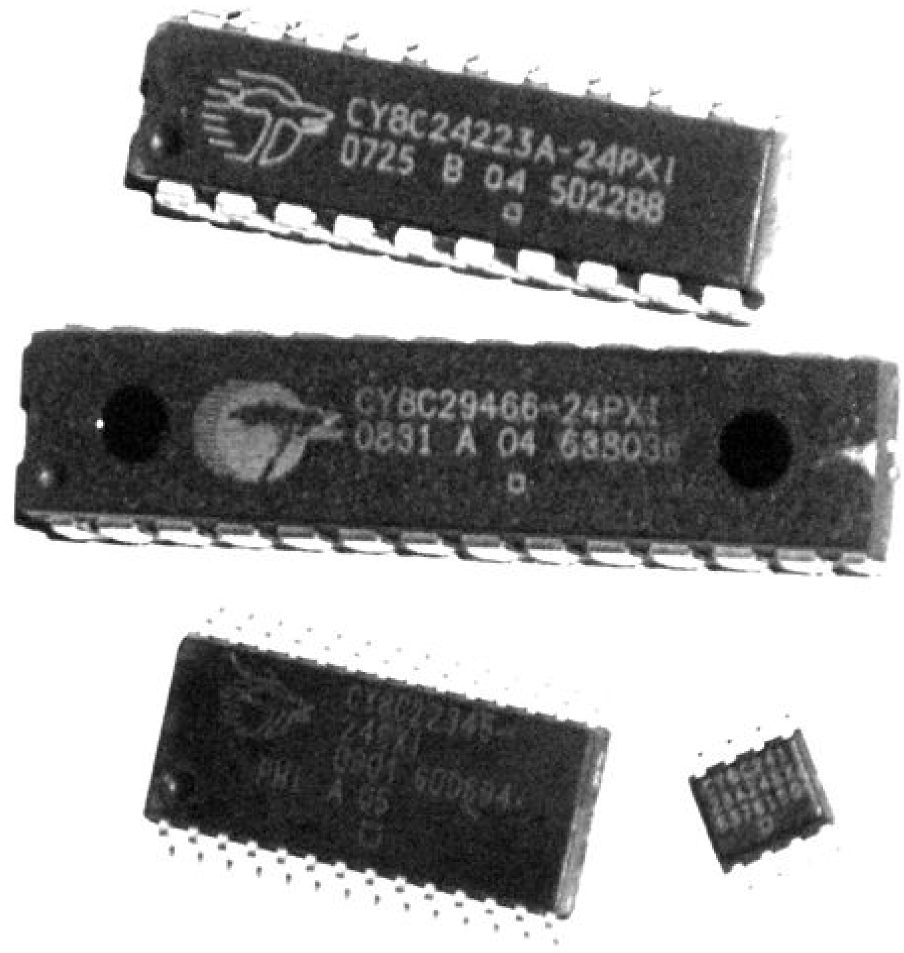
Atmel

- AT89 series (Intel 8051 architecture)
- AT90, ATtiny, ATmega, ATxmega series (AVR architecture) (Atmel Norway design)
- AT91SAM (ARM architecture)
- AVR32 (32-bit AVR architecture) (Atmel Norway design)
- MARC4

Charmed Labs

- Qwerk
- XPort

Cypress Semiconductor



Cypress PsoC chips

- CY8C2xxxx (PSoC1) CPU M8C
- CY8C3xxxx (PSoC3) CPU 8051
- CY8C5xxxx (PSoC5) CPU ARM Cortex M3

Psoc (Programmable system on CHIP)

Dallas Semiconductor

- 8051 Family

- MAXQ RISC Family
- Secure Micros Family

ELAN Microelectronics Corp.

ELAN Microelectronics Corporation is an IC designer and provider of 8-bit microcontrollers and PC Peripheral ICs. Situated at the Hsinchu Science Park, the Silicon Valley of Taiwan, ELAN's microcontroller product range includes the following:

- EM78PXXX Low Pin-Count MCU Family
- EM78PXXXN GPIO Type MCU Family
- EM78PXXXN ADC Type MCU Family

Energy Micro AS

Energy Micro AS provides low energy 32-bit microcontrollers using an ARM Cortex-M3 processor. The semiconductor company is situated in Oslo, Norway. The EFM32 products consists of:

- *Standard microcontrollers*
- *Application specific microcontrollers*
- *Custom microcontrollers*

EPSON Semiconductor

- 4-bit Microcomputers S1C60/62/63 family
- 8-bit Microcomputers S1C88 family
- 16-bit Microcomputers S1C17 family
- 32-bit Microcomputers S1C33 family

Freescale Semiconductor

Until 2004, these µCs were developed and marketed by Motorola, whose semiconductor division was spun off to establish Freescale.

- 8-bit
  - 68HC05 (CPU05)
  - 68HC08 (CPU08)
  - 68HC11 (CPU11)
- 16-bit
  - 68HC12 (CPU12)
  - 68HC16 (CPU16)
  - Freescale DSP56800 (DSPcontroller)
- 32-bit
  - Freescale 683XX
  - M·CORE

    - MPC500
    - MPC 860 (PowerQUICC)
    - MPC 8240/8250 (PowerQUICC II)
    - MPC 8540/8555/8560 (PowerQUICC III)

Fujitsu

- F²MC Family (8/16 bit)
- FR Family (32 bit)
- FR-V Family (32 bit RISC)

Holtek

Holtek Semiconductor is a Taiwan-based designer of 8-bit microcontrollers and peripheral products. Located in the *Hsinchu Science Park* (), the company's product range includes the following microcontroller device series:

- HT48FXX Flash I/O type series
- HT48RXX I/O type series
- HT46RXX A/D type series
- HT49RXX LCD type series
- HT82XX Computer Peripheral series
- HT95XX Telecom Peripheral series
- HT86XX Voice series

Infineon

- 8-bit
    - XC800 family
    - C500/C800 family
- 16-bit
    - XE166
    - C166 family
    - C167 family
    - XC2000 family
- 32-bit
    - TRICORE family

Intel

→ *List of Intel microprocessors#Microcontrollers*

- 8-bit
    - MCS-48 (8048 family – also incl. 8035, 8038, 8039, 8040, 8X42, 8X49, 8050; X=0 or 7)
    - MCS-51 (8051 family – also incl. 8X31, 8X32, 8X52; X=0, 3, or 7)

    - 8xC251
- 16-bit
    - MCS-96 (8096 family – also incl. 8061)
    - Intel MCS-296

Lattice Semiconductor

- Mico8 8 bit soft microprocessor
- Mico32 32 bit soft microprocessor

Microchip Technology

Microchip produces microcontrollers with 3 very different architectures:

8-bit (8 bit data bus) PICmicro, with a single accumulator (8 bits):

- PIC10 and PIC12: 12-bit instruction words
- PIC16 series: 14-bit instruction words, one address pointer ("indirect register pair")
    - PIC16F628 (Replacement for very popular but discontinued PIC16F84)
    - PICAXE
- PIC18 series: 16-bit instruction words, three address pointers ("indirect register pairs")

16-bit (16-bit data bus) microcontrollers, with 16 general-purpose registers (each 16-bit)

- PIC24: 24-bit instruction words
- dsPIC: based on PIC24, plus DSP functions, such as a single-cycle MAC (multiply-accumulate) into two 40-bit accumulators.

32-bit (32 bit data bus) microcontrollers:

- PIC32MX series: 32 bit instructions, uses the MIPS architecture

National Semiconductor

- COP400 (4-bit)
- COP8
- CR16
- SC/MP

NEC

- 17K
- V25
- 75X

- 78K
- V850

Parallax

- SX
  - **These were formerly made by Ubicom. The SX die is still manufactured by Ubicom, who send it to Parallax for packaging**
  - SX-18, 20, 28, 48 and 52 versions (Note that the SX-18 and SX-52 have been discontinued)
  - Parallax's SX series is an 8 bit microcontroller which has unusually high speed, up to 75 MHz (75 MIPS), and a high degree of flexibility. Andre LaMothe has proven that the SX-52 can actually be clocked to 80 MHz (80 MIPS) even though the specs say 75 MHz is the maximum. He has used the SX-52 in thousands of XGameStation development computers all running at 80 MHz. Some users have referred to these microcontrollers as PICs on steroids. While Parallax's SX micros are limited in variety, their high speed and additional resources allow programmers to create 'virtual devices', including complete video controllers, as required. Refer to Parallax's Web site for information, as they are the sole distributor of these devices.
- Propeller
  - The Propeller is a 8-core 32-Bit microcontroller with 32kB internal RAM.

NXP Semiconductors

- 8-bit
  - 80C51
- 16-bit
  - XA
- 32-bit
  - ARM7
    - LPC2000
  - ARM9
    - LPC3000
  - ARM Cortex-M4
    - LPC4300
  - ARM Cortex-M3
    - LPC1700/LPC1300/LPC1800
  - ARM Cortex-M0
    - LPC1100/LPC1200

Rabbit Semiconductor

- Rabbit 2000
- Rabbit 3000

- Rabbit 4000

Renesas Electronics

Renesas is a joint venture of Hitachi and Mitsubishi Electric. In April 2010 Renesas Technology and NEC Electronics merged to form Renesas Electronics.

- 4-bit
  - 720
- 8-bit
  - 78K
  - SLP
  - 740
- 16-bit
  - M16C
  - H8
  - R8C
- 32-bit
  - SuperH
  - V850
  - RX

SiLabs

Manufactures a line of 8051-compatible microcontrollers, notable for high speeds (50-100 MIPS) and large memories in relatively small package sizes. A free IDE is available that supports the USB-connected ToolStick line of modular prototyping boards. These microcontrollers were originally developed by Cygnal.

- C8051F300
  - QFN11 package (3x3 mm), 25 MIPS, 8kB Flash Memory, 256B RAM, 8 I/O, UART, SMBus, 3 timers, 8 bit 8 ch 500kbs ADC, temperature sensor, Comparator.
- C8051F120
  - TQFP100 package, 128k Flash, 8448B RAM, 64 I/O, 2 UARTS, SMBus, SPI, 5 timers, 12 bit 8ch ADC, 8 bit 8ch ADC, 12 bit 2ch DAC, temperature sensor, 2 comparators, 16x16MAC.

Silicon Motion

- SM2XX Family - Flash Memory Card Controllers
- SM321 - USB 2.0
- SM323 - USB 2.0
- SM323E - USB 2.0
  - Silicon Motion's SM321E and SM324 controllers support SLC and MLC NAND flash from Samsung, Hynix, Toshiba and ST Micro as well as

    - flash products from Renesas, Infineon and Micron. The SM321E is available in a 48-pin LQFP package and a 44-pin LGA package. The SM321E supports up to 4 SLC or MLC NAND flash chips with 4 bytes / 528 bytes ECC
- SM324 - USB 2.0
    - Supports dual-channel data transfer at read speeds of 233x (35 MB/s) and write speeds of 160x (24 MB/s), making it the fastest USB 2.0 flash disk controller in the market. The SM324 also has serial peripheral interface (SPI) which allows for not only Master and Slave modes, but the flexibility to develop more functionality into USB flash disk (UFD) products such as GPS, fingerprint sensor, Bluetooth and memory-capacity display. The SM324 is available in a 64-pin LQFP package. The SM324 supports 8 SLC or MLC NAND flash chips with 4 bytes / 528 bytes ECC.
- SM330 - USB 2.0
- SM501 - Mobile Graphics
- SM712 - Mobile Graphics
- SM722 - Mobile Graphics
- SM340 - MP3/JPEG
- SM350 - MP3/JPEG
- SM370 - Image processing

Sony

- SPC700/700αII Series
- SPC900 Series
- SPC970 Series
- SR11 Series

STMicroelectronics

- ST6 (8 bit)
- ST7 (8 bit)
- STM8 (8 bit), STM MCU Pages, Extra info concerning STM8 family.
- µPSD (8032 - 8 bit)
- ST10 (16 bit)
- STM32 (ARM Cortex M3 - 32 bit), STM MCU Pages, Extra info concerning STM32 family.
- STR7 (ARM7TDMI - 32 bit)
- STR9 (ARM966E-S - 32 bit)

Texas Instruments

- TMS370 (8-bit)
- MSP430 (16-bit, Ultra-low-power)
- TMS320F28xx (32-bit)
- C2000 (32-bit, Real-time control)

- Stellaris (32-bit, ARM Cortex-M3)
- TMS570 (32-bit RISC, ARM Cortex-R4)

Toshiba

- TLCS-47 (4-bit)
- TLCS-870 (8-bit CISC)
- TLCS-900 (16 and 32-bit CISC)
- TX19A (32-bit RISC)

Ubicom

- IP2022
    - Ubicom's IP2022 is a high performance (120 MIPS) 8-bit microcontroller. Features include: 64k flash code memory, 16kB PRAM (fast code and packet buffering), 4kB data memory, 8-channel A/D, various timers, and on-chip support for Ethernet, USB, UART, SPI and GPSI interfaces.
- IP3022
    - IP3022 is Ubicom's latest high performance 32bit processor running at 250 MHz featuring 8 hardware threads (barrel processor). It is specifically targeted at Wireless Routers.

Xemics

- XE8000 8-bit microcontroller family

Xilinx

- Microblaze 32 bit soft microprocessor
- Picoblaze 8 bit soft microprocessor

XMOS

- XCore XS1 32-bit, multithreaded, event driven micro

ZiLOG

Zilog's (primary) microcontroller families, in chronological order:

- *Older:*
    - Zilog Z8 - 8-bit Harvard architecture ROM / EPROM / OTP microcontroller with on-chip SRAM.
    - Zilog Z180 - Z80 based microcontroller; on-chip peripherals; external memory; 1 MB address space.
- *Newer:*

- Zilog eZ8 - Better pipelined Z8 (2-3 times as clock cycle efficient as original Z8) with on-chip flash memory and SRAM.
- Zilog eZ80 - Fast 8/16/24-bit Z80 (3-4 times as cycle efficient as original Z80) with flash, SRAM, peripherals; linear addressing of 16 MB.
- Zilog Z16 - Fast 8/16/32-bit CPU with compact object code; 16 MB (4GB possible) addressing range; flash, SRAM, peripherals, on chip.