The 8052-Basic Microcontroller

© by
Heinz Rongen
Forschungszentrum Jülich
Zentrallabor für Elektronik
52425 Jülich, Germany

email: H.Rongen@fz-juelich.de

Microcontroller Basics

This chapter introduces you to the world of microcontrollers, including definitions, some history, and a summary of what's involved in designing and building a microcontroller project.

What's a Microcontroller?

A microcontroller is a computer-on-a-chip, or, if you prefer, a single-chip computer. *Micro* suggests that the device is small, and *controller* tells you that the device might be used to control objects, processes, or events. Another term to describe a microcontroller is e*mbedded controller,* because the microcontroller and its support circuits are often built into, or embedded in, the devices they control. You can find microcontrollers in all kinds of things these days. Any device that measures, stores, controls, calculates, or displays information is a candidate for putting a microcontroller inside. The largest single use for microcontrollers is in automobiles ,just about every car manufactured today includes at least one microcontroller for engine control, and often more to control additional systems in the car. In desktop computers, you can find microcontrollers inside keyboards, modems, printers, and other peripherals. In test equipment, microcontrollers make it easy to add features such as the ability to store measurements, to create and store user routines, and to display messages and waveforms. Consumer products that use microcontrollers include cameras, video recorders, compact-disk players, and ovens. And these are just a few examples.

A microcontroller is similar to the microprocessor inside a personal computer. Examples of microprocessors include Intel's 8086, Motorola's 68000, and Zilog's Z80. Both microprocessors and microcontrollers contain a central processing unit, or CPU. The CPU executes instructions that perform the basic logic, math, and data-moving functions of a computer. To make a complete computer, a microprocessor requires memory for storing data and programs, and input/output (I/O) interfaces for connecting external devices like keyboards
and displays. In contrast, a microcontroller is a single-chip computer because it contains memory and I/O interfaces in addition to the CPU. Because the amount of memory and interfaces that can fit on a single chip is limited, microcontrollers tend to be used in smaller systems that require little more than the microcontroller and a few support components. Examples of popular microcontrollers are Intel's 8052 (including the 8052-BASIC, which is the focus of this book), Motorola's 68HC11, and Zilog's Z8.

A Little History

To understand how microcontrollers fit into the always-expanding world of computers, we need to look back to the roots of microcomputing. In its January 1975 issue, Popular Electronics magazine featured an article describing the Altair 8800 computer, which was the first microcomputer that hobbyists could build and program themselves. The basic Altair included no keyboard, video display, disk drives, or other elements we now think of as essential elements of a personal computer. Its

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

8080 microprocessor was programmed by flipping toggle switches on the front panel. Standard RAM was 256 bytes and a kit version cost $397 ($498 assembled). A breakthrough in the Altair's usability occurred when a small company called Microsoft offered a version of the BASIC programming language for it.
Of course, the computer world has changed a lot since the introduction of the Altair. Microsoft has become an enormous software publisher, and a typical personal computer now includes a keyboard, video display, disk drives, and Megabytes of RAM. What's more, there's no longer any need to build a personal computer from scratch, since mass production has drastically lowered the price of assembled systems. At most, building a personal computer now involves only installing assembled boards and other major components in an enclosure.
A personal computer like Apple's Macintosh or IBM's PC is a general-purpose machine, since you can use it for many applications—word processing, spreadsheets, computer-aided design, and more—just by loading the appropriate software from disk into memory.
Interfaces to personal computers are for the most part standard ones like those to video displays, keyboards, and printers.

But along with cheap, powerful, and versatile personal computers has developed a new interest in small, customized computers for specific uses. Each of these small computers is dedicated to one task, or a set of closely related tasks. Adding computer power to a device can enable it to do more, or do it faster, better, or more cheaply. For example, automobile engine controllers have helped to reduce harmful exhaust emissions. And microcontrollers inside computer modems have made it easy to add features and abilities beyond the basic computer-to-phone-line interface.
In addition to their use in mass-produced products like these, it's also become feasible to design computer power into one-of-a-kind projects, such as an environmental controller for a scientific study or an intelligent test fixture that ensures that a product meets its specifications before it's shipped to a customer. At the core of many of these specialized computers is a microcontroller. The computer's program is typically stored permanently in semiconductor memory such as ROM or EPROM. The interfaces between the microcontroller and the outside world vary with the application, and may include a small display, a keypad or switches, sensors, relays, motors, and so on. These small, special-purpose computers are sometimes called single-board computers*,* or SBCs. The term can be misleading, however, since the computer doesn't have to be on a single circuit board, and many types of computer systems, such as laptop and notebook computers, are now manufactured on a single board.

New Tools

To design and build a computer-controlled device, you need skills in both circuit design and software programming. The good news is that a couple of recent advances have simplified the tasks involved.
One is the introduction of microcontrollers themselves, since they contain all of the elements of a computer on a single chip. Using a microcontroller can reduce the

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

number of components and thus the amount of design work and wiring required for a project. The 8052-BASIC microcontroller even includes its own programming language, called BASIC-52. The other development is personal computers themselves. A desktop computer can help tremendously by serving as a *host system* for writing and testing programs. As you are developing a project, you can use a serial link to connect the host system to a *target system,* which contains the microcontroller circuits you are testing. You can then use the personal computer's keyboard, video display, disk drives, and other resources for writing and testing programs and transferring files between the two systems.

The 8052-BASIC chip described in this book is perfect for many simpler applications, especially control and monitoring tasks. Because the chip is easy to use, it's a good way to learn about microcontrollers and computers in general. Although you can't do the most complex projects with it, you can do a lot, at low cost and without a lot of hassle.

Choosing a chip.

Does it matter which microcontroller chip you use? All microcontrollers contain a CPU, and chances are that you can use any of several devices for a specific project. Within each device family, you'll usually find a selection of family members, each with different combinations of options. For example, the 8052-BASIC is a member of the 8051 family of microcontrollers, which includes chips with program memory in ROM or EPROM, and with varying amounts of RAM and other features. You select the version that best suits your system's requirements.

Microcontrollers are also characterized by how many bits of data they process at once, with a higher number of bits generally indicating a faster or more powerful chip. Eight-bit chips are popular for simpler designs, but 4-bit, 16-bit, and 32-bit architectures are also available. The 8052-BASIC is an 8-bit chip. Power consumption is another consideration, especially for battery-powered systems. Chips manufactured with CMOS processes usually have lower power consumption than those manufactured with NMOS processes. Many CMOS devices have special standby or "sleep" modes that limit current consumption to as low as a few microamperes when the circuits are inactive. Using these modes, a data logger can reduce its power consumption between samples, and power up only when it's time to take data. The 8052-BASIC chip is available in both NMOS and CMOS versions. The original 8052-BASIC was an NMOS chip, offered directly from Intel. (Intel's term for its NMOS process is HMOS.) Although Intel never offered a CMOS version directly, Micromint became a source by ordering a batch of CMOS 8052's with the BASIC-52 programming language in ROM. The CMOS version, the 80C52-BASIC, has maximum power consumption of 30 milliamperes, compared to 175 milliamperes for the NMOS 8052-BASIC. All microcontrollers have a defined instruction set, which consists of the binary words that cause the CPU to carry out specific operations. For example, the instruction *0010 0110* tells an 8052 to add the values in two locations.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

The binary instructions are also known as operation codes, or opcodes for short. The opcodes perform basic functions like adding, subtracting, logic operations, moving and copying data, and controlling program branching. Control circuits often require reading or changing single bits of input or output, rather than reading and writing a byte at a time. For example, a microcontroller might use the eight bits of an output port to switch power to eight sockets. If each socket must operate independently of the others, a way is needed to change each bit without affecting the others. Many microcontrollers include bit-manipulation (also called Boolean) opcodes that easily allow programs to set, clear, compare, copy, or perform other logic operations on single bits of data, rather than a byte at a time.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

Options for storing programs.

Another consideration in circuit design is how to store programs. Instead of using disk storage, most microcontroller circuits store their programs on-chip. For one-of-kind projects or small-volume production, EPROM has long been the most popular method of program storage. Besides EPROMs, other options include EEPROM, ROM, nonvolatile (NV), or battery-backed, RAM, and Flash EPROM. The program memory may be in the microcontroller chip, or a separate component. To save a program in **EPROM**, you must set the EPROM's data and address pins to the appropriate logic levels for each address and apply special programming voltages and control signals to store the data at the selected address. The programming process is sometimes called *burning* the EPROM. You erase the contents by exposing the chip's quartz window, and the circuits beneath it, to ultraviolet energy.
Some microcontrollers contain a one-time-programmable, or field-programmable, EPROM. This type has no window, so you can't erase its contents, but because it's cheaper than a windowed IC, it's a good choice when a program is finished and the device is ready for quantity production. Several techniques are available for programming EPROMs and other memory chips. With a manual programmer, you flip switches to toggle each bit and program the EPROM byte by byte. This is acceptable for short programs, but quickly becomes tedious with a program of any length. Computer control simplifies the job greatly. With an EPROM programmer that connects to a personal computer, you can write a program at your keyboard, save it to disk if you wish, and store the program in EPROM in a few easy steps. Data sheets for EPROMs rarely specify the number of erase and reprogramming cycles a device is guaranteed for, but a typical EPROM should endure 100 erase/program cycles, and usually many more.

EEPROMs are much like EPROMs except that they are electrically erasable—no ultraviolet source is required. Limitations of EEPROMs include slow speed, high cost, and a limited number of times that they can be reprogrammed (typically 10,000 to 100,000).

ROMs are cost-effective when you need thousands of copies of a single program. ROMs must be factory-programmed and once programmed, can't be changed.

NVRAM typically includes a lithium cell, control circuits, and RAM encapsulated in a single IC package. When power is removed from the circuit, the lithium cell takes over and preserves the information in RAM, for 10 years or more. You can reprogram an NVRAM n infinite number of times, with the only limitation being battery life.

Flash EPROM is electrically erasable, like EEPROM, but most Flash devices erase all at once, or in a few large blocks, rather than byte-by-byte like EEPROM. Some Flash EPROMs require special programming voltages. As with EPROMs, the number of erase/program cycles is limited.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

The 8052-BASIC uses two types of program memory. An 8-kilobyte, or 8K, on-chip ROM stores the BASIC-52 interpreter. For storing the BASIC-52 programs that you write, the BASIC-52 language has programming commands that enable you to save programs in external EPROM, EEPROM, or NVRAM.

Other memory. Most systems also require a way to store data for temporary use. Usually, this is RAM, whose contents you can change as often as you wish. Unlike EPROM, ROM, EEPROM, and NVRAM, the contents of the RAM disappear when you remove power the chip (unless it has battery back-up).

Most microcontrollers include some RAM, typically a few hundred bytes. The 8052-BASIC has 256 bytes of internal RAM. A complete 8052-BASIC system requires at least 1024 bytes of external RAM as well.

I/O options. Finally, input/output (I/O) requires design decisions. Most systems require interfaces to things like sensors, keypads, switches, relays, and displays. Most microcontrollers have ports for interfacing to the world outside the chip. The 8052-BASIC uses many of its ports for accessing external memory and performing other special functions, but some port bits are available for user applications, and you can easily increase the available I/O by adding support chips.

Writing the Control Program

When it's time to write the program that controls your project, the options include using machine code, assembly language, or a higher-level language. Which programming language
you use depends on things like desired execution speed, program length, and convenience, as well as what's available in your price range.

Machine code. The most fundamental program form is machine code, the binary instructions that cause the CPU to perform the operations you desire.

Assembly language. One step removed from machine code is assembly language, where abbreviations called mnemonics (memory aids) substitute for the machine codes. The mnemonics are easier to remember than the machine codes they stand for. For example, in the 8052's assembly language, the mnemonic *CLR C* means clear the carry bit, and is easier to remember than its binary code (*11000011*).
Since machine code is ultimately the only language that a CPU understands, you need some way of translating assembly-language programs into machine code. For very short programs, you can *hand assemble,* or translate the mnemonics yourself by looking up the machine codes for each abbreviation. Another option is to use an *assembler*, which is software that runs on a desktop computer and translates the mnemonics into machine code. Most assemblers provide other features, such as formatting the program code and creating a listing
that shows both the machine-code and assembly-language versions of a program side -by-side.

Higher-level languages. A disadvantage to assembly language is that each device family has its own set of mnemonics, so you have to learn a new vocabulary for each family you work with. To get around this problem, higher-level languages like C, Pascal, Fortran, Forth, and BASIC follow a standard syntax so that programs are more portable from one device to another. The idea is that with minor changes, you can use a language like BASIC to write programs for many different devices. In reality, each language tends to develop many different dialects, depending on the chip and the preferences of the language's vendor, so porting a program to a different device isn't always effortless. But there are many similarities among the dialects of a single language, so, as with spoken language, a new dialect is easier to learn than a whole new language. Higher-level languages also simplify programming by allowing you to do in one or a few lines what would require many lines of assembly code to accomplish.

Interpreters and **compilers** are two forms of higher-level languages. An interpreter trans-lates program into machine code each time the program runs, while a compiler translates nly once, creating a new, executable file that the computer runs directly, without re-trans-lating. As a rule, interpreters are very convenient for shorter programs where execution speed isn't critical. With an interpreted language, you can run your program code immediately after you write it, without a separate compile or

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

assembly step. A compiler is a good choice when a program is long or has to execute quickly. A single language like BASIC may be available
in both interpreted and compiled versions. Each device family requires its own interpreter or compiler to translate the higher-level code into the machine code for that device. In other words, you can't use QuickBASIC for IBM PCs to program an 8052 microcontroller—you need a compiler that generates program code for the 8052. Compared to an equivalent program written in assembly language, a compiled program usually is larger and slower, so assembly language is the way to go if a program must be as fast or as small as possible. A higher-level language also may not offer all of the abilities of assembly code, though you can get around this by calling subroutines in assembly language when necessary.

BASIC-52 is an interpreted language, but BASIC compilers for the 8052 are also available. In fact, you can have the best of both worlds by testing your programs with the BASIC-52 interpreter, and compiling the finished product for faster execution and other benefits of the compiled version.

Testing and Debugging

After you've written a program, or a section of one, it's time to test it and as necessary, find and correct mistakes to get it working properly. The process of ferreting out and correcting mistakes is called debugging. Easy debugging and troubleshooting can make a big difference
in how long it takes to get a system up and running. As with programming, you have several options here as well.

Testing in EPROM. One way is to burn your program into EPROM, install the EPROM in your system, run the program, and observe the results. If problems occur (as they usually will) you modify the program, erase and reburn the EPROM, and try again, repeating as many times as necessary until the system is operating properly.

Development systems. Another option is to use a development system. A typical development system consists of a monitor program, which is a program stored in EPROM or other memory in the microcontroller system, and a serial link to a personal computer. Using the abilities of the monitor program, you can load your program from a personal computer into RAM (instead of the more permanent EPROM) on the microcontroller system, then run the program, modify it, and retry as often as necessary until the program is working properly. Most development systems also allow single-stepping, setting breakpoints, and viewing and changing the data in memory. In single-stepping, you run the program one step at time, pausing after each step, so you can more easily monitor what the circuits and program are doing at each step. A breakpoint is a program location where the program stops executing and waits for a command to continue. You can set breakpoints at critical spots in your program. At any breakpoint, you can view or change the contents of memory or perform other tests.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

The 8052-BASIC's development system

The 8052-BASIC system and a personal com-puter form a complete development system for writing, testing, and storing programs. The personal computer's keyboard and screen make it easy to write and run programs and view
the results. BASIC-52 has many built-in debugging features that make it easy to test programs. You can run a program immediately after writing it, without having to assemble, compile, or program an EPROM. You can use a STOP statement and CONT (continue) command to set breakpoints and resume executing your program. You can use PRINT statements to display variables as the program runs. And, if you wish, you can use your personal computer for writing programs off-line and uploading and downloading them to the 8052-BASIC system.

Inside the 8052-BASIC

This chapter introduces you to the 8052-BASIC chip, including the kinds of projects you can do with it, what equipment, materials, and skills you need in order to design and build an 8052-BASIC project, and a pin-by-pin look at the chip and its abilities.

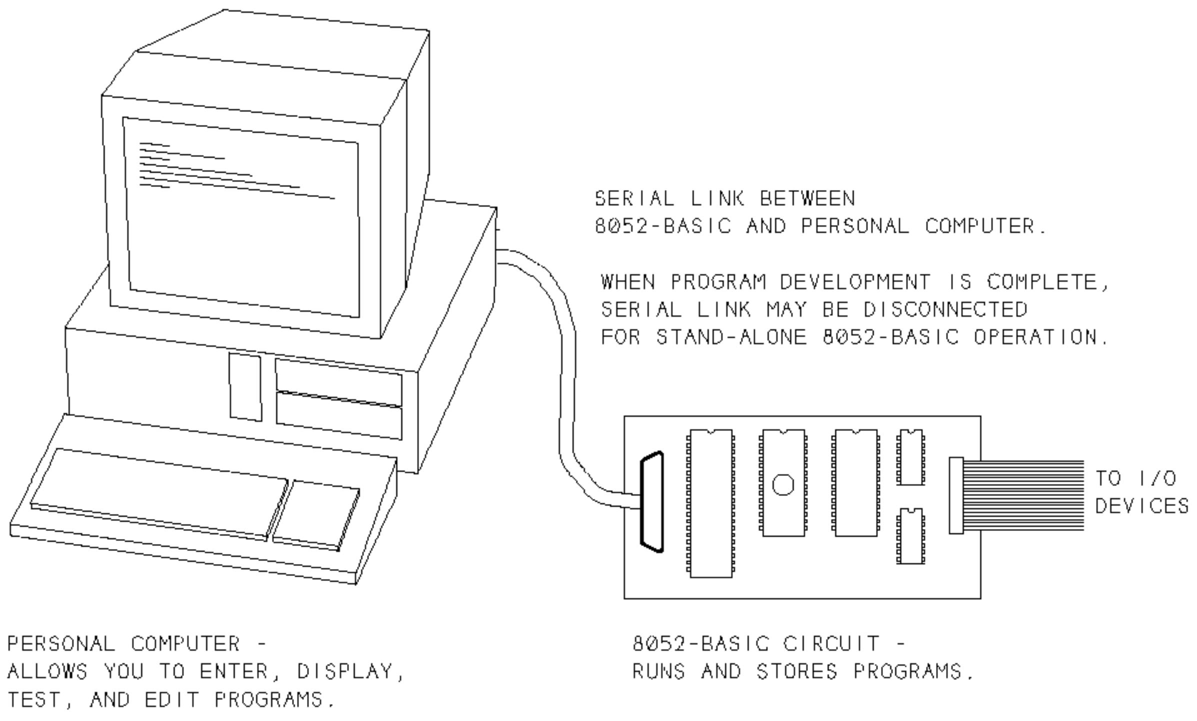
Possibilities

The 8052-BASIC microcontroller is an easy-to-use, low-cost, and versatile computer-on-a-chip. It's ideal for projects that require more than an assortment of logic gates, but less than a complete desktop computer system with a full keyboard, display, and disk drives. If you're interested in doing more with computers than simply running applications programs, the 8052-BASIC gives you a chance to design and build a system from the ground up. With a few support chips and a program stored in memory, you can use the 8052-BASIC to sense, measure, and control processes, events, or conditions.

The 8052-BASIC is actually two products in one: it's an 8052 microcontroller, with the BASIC-52 programming language on-chip. To begin using the 8052-BASIC, you need a minimum circuit consisting of the 8052-BASIC and some support components, plus a personal computer. This book contains specific instructions for use with "IBM-compatible," or MS-DOS, computers, but you can use any computer that has an RS-232 serial port and communications software to go with it.

The Figure below shows the basic setup.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*


SERIAL LINK BETWEEN
8052-BASIC AND PERSONAL COMPUTER.
WHEN PROGRAM DEVELOPMENT IS COMPLETE, SERIAL LINK MAY BE DISCONNECTED FOR STAND-ALONE 8052-BASIC OPERATION.
TO I/O DEVICES
PERSONAL COMPUTER - ALLOWS YOU TO ENTER, DISPLAY, TEST, AND EDIT PROGRAMS.
8052-BASIC CIRCUIT - RUNS AND STORES PROGRAMS.

With an 8052-BASIC circuit connected by a serial link to a personal computer, you have a complete development system with these abilities:

- You can write and run BASIC programs. You use the keyboard, video display, and other resources of the personal computer to type and view the programs and commands that the 8052-BASIC system executes. BASIC-52 is an interpreted language whose programs do not require an additional assembling or compiling step. You can run programs or execute commands immediately after you write them.
- You can use BASIC-52's programming functions to permanently store your programs in EPROM or other nonvolatile memory. You don't need a separate EPROM programmer.
- You can also store programs on your personal computer's disk. You can write or edit programs on your personal computer, and then upload them to the 8052-BASIC system.
- To the basic circuits, you can add displays, switches, keypads, relays, and other components, depending on the needs of your project.
- After program development, you can disconnect the link to the personal computer and let the 8052-BASIC system run its stored program on its own.

Limits

No single product is ideal for every use. These are some of the limitations to the 8052-BASIC:

- Program execution can be slow, compared with programs that run on more powerful computers, or programs written in assembly language. A typical program line in BASIC-52 takes several milliseconds to execute. Because of this, there are some tasks that BASIC-52 just can't handle—for example, detecting and responding to an interrupt within a few microseconds. But for many control, monitoring, and other tasks, BASIC-52 is fine. For example, a weather station that senses conditions once per minute and stores or displays the results doesn't need super-fast response. And, if necessary, you can call an assembly-language routine for a portion of a program where speed is critical. Even if you write your programs in assembly language, C, or another language, you can use the 8052-BASIC system as a development system that enables you to upload your program to memory, run the program, and test and debug your programs and circuits.
- Another limitation of the 8052-BASIC is that a complete project requires additional components. If you're looking for a true single-chip solution, the 8052-BASIC isn't it. Even a minimal system requires an external RAM chip, and most systems also have an external EPROM or other non-volatile memory. The serial link and other optional functions also use some of the on-chip timers and input/output ports, so these may not be available for other uses. Still, the 8052-BASIC lets you to do a lot with a little. When needed, you can easily add chips to expand the input/output ports, timers, and other functions.
- And finally, don't expect BASIC-52 to have the abilities of *QBasic, Visual Basic* or other BASIC programming languages that you may use on your personal computer. BASIC-52 is more capable than many other single-chip BASICs. It includes features like loops, subroutines, string handling, and even floating-point math for handling factional quantities. But there are some primitive aspects to the language. For example, the on-line editing functions are limited. Once you write a program line, you can change it only by retyping from the beginning. The limitations are understandable, because the entire programming language has to fit in the 8052's 8 kilobytes of ROM. Fancy editing and other features just aren't feasible in this small space.

There are solutions here as well. You can get around many of the editing limitations by writing and editing programs off-line, using your personal computer and text editor, and then uploading to the 8052-BASIC system. And, there are software and hardware products that enhance BASIC-52 and make it easier to use, especially for longer, more complex programming jobs.

The 8051 Family

At the core of the 8052-BASIC is an 8052 microcontroller, a member of the 8051 microcontroller family. Intel Corporation introduced the 8051 in 1980. Since that time, 8051-family chips have been used as the base of thousands of products. Many other
companies, including Philips, Siemens, Dallas Semiconductor, OKI, Fujitsu, and Harris-Matra now also make 8051-family chips. Some companies have expanded the 8051 family by offering compatible chips with additional features. Table below summarizes the differences among popular 8051-family chips. The 8052 is an

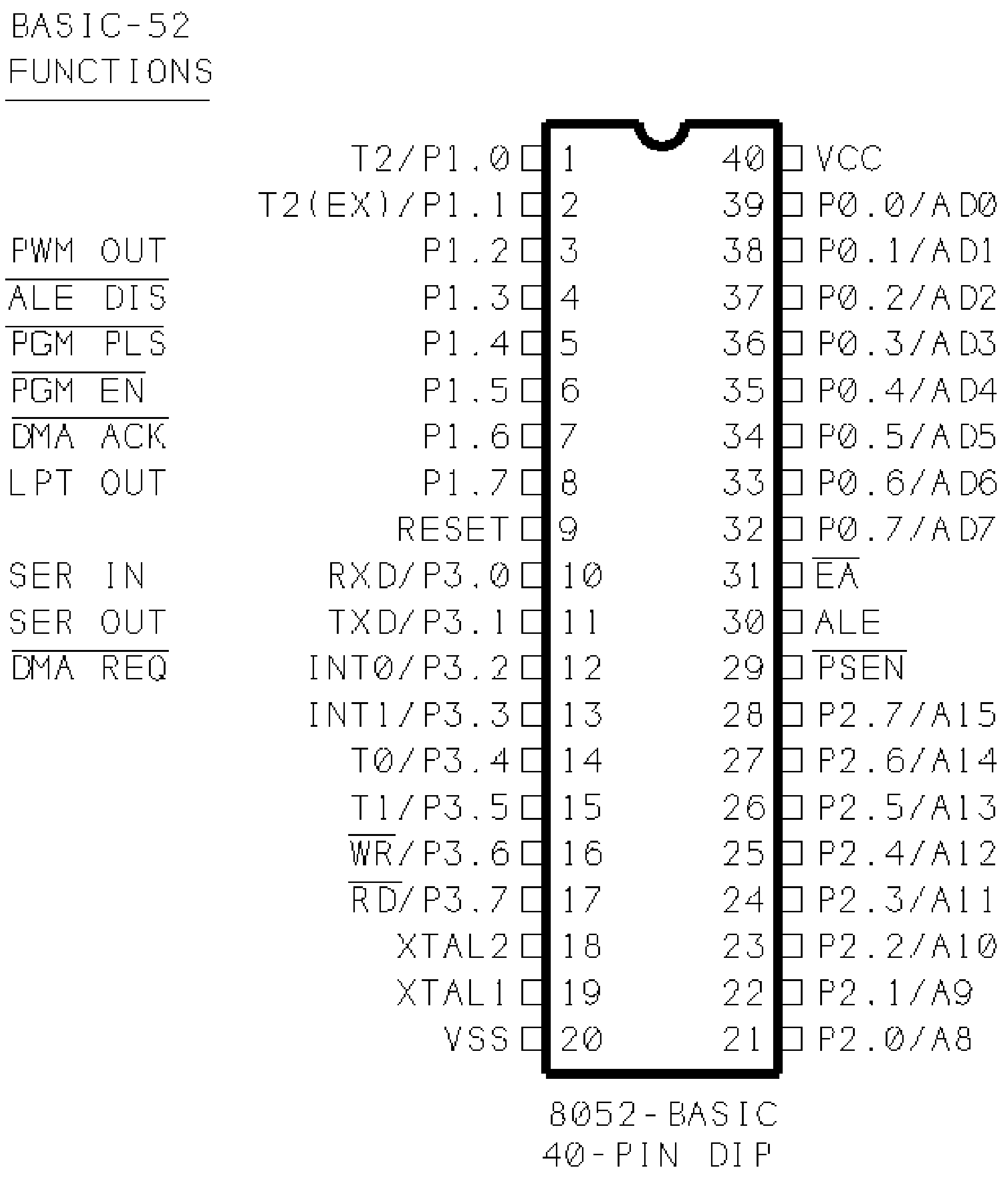
enhanced 8051, with an extra timer and more RAM and ROM. The 8031 and 8032 are identical to the 8051 and 8052, except that the ROM area is unused, and program code must be stored in an external EPROM or other memory chip.

Differences among 8051-family chips.

Chip	**ProgramMem.**	**Kilobyte**	**RAM (bytes)**	**Timers**
8051	ROM	4	128	2
8052	ROM	8	256	3
8031	none	-	128	2
8032	none	-	256	3
8751	EPROM	4	128	2
8752	EPROM	8	256	3

· 80C51, 80C52, 80C31, and so on are CMOS versions of above.
· 80C51FA/B/C add more versatile timers and an enhanced serial channel.
· 8052-BASIC has the BASIC-52 programming language in ROM.
· Packages include 40-pin DIP, 40-lead PLCC, and 44-pin QFP.

The 8052, like other 8051-family chips, is available in NMOS and CMOS versions. Figure 2-2 shows the pinout of the 8052 and 8052-BASIC, and Table below describes the pin functions.



Elements of the 8052 and 8052-BASIC

These are the major elements of the 8052, plus the enhancements included in the 8052-BASIC:

CPU The CPU, or central processing unit, executes program instructions. Types of instructions include arithmetic (addition, subtraction), logic (AND, OR, NOT), data transfer (move), and program branching (jump) operations. An external crystal provides a timing reference for clocking the CPU.

ROM ROM (read-only memory) is the read-only memory that is programmed into the chip in the manufacturing process. In the 8052-BASIC, the ROM contains the BASIC-52 interpreter program that the 8052 executes on boot-up. As far as the hardware is concerned, this is the only difference between the ordinary 8052 and the 8052-BASIC.

RAM RAM (random-access memory) is where programs store information for temporary use. Unlike ROM, the CPU can write to RAM as well as read it. Any information stored in RAM is lost when power is removed from the chip. The 8052 has 256 bytes of RAM. BASIC-52 uses much of this for its own operations, with a few bytes available to users.

I/O Ports I/O (Input/Output) Ports enable the 8052 to read and write to external memory and other components. The 8052 has four 8-bit I/O ports (Ports 0-3). As the name suggests, the ports can act as inputs (to be read) or outputs (to be written to). Many of the port bits have optional, alternate functions relating to accessing external memory, using the on-chip timer/counters, detecting external interrupts, and handling serial communications. BASIC-52 assigns alternate functions to the remaining port bits. Some of these functions are required by BASIC-52, while others are optional. If you don't use an alternate function, you can use the bit for any control, monitoring, or other purpose in your application.

Accessing external memory. The largest alternate use of the ports has to do with accessing external memory. Although the 8052 is a single-chip computer, a complete 8052-BASIC system requires additional components. It must have external RAM in addition to the 8052's internal RAM, and most systems also have EPROM, EEPROM, or battery-backed RAM for permanent storage of BASIC-52 programs. Accessing this external memory uses all of Ports 0 and 2, plus bits 6 and 7 of Port 3, to hold data, addresses, and control signals for reading and writing to external memory. Data here refers to a byte to be read or written, and may be any type of information, including program code. The address defines the location in memory to be read or written. During a memory access, Port 0's eight pins (AD0-AD7) first hold the lower byte of the address, followed by the data to be read or written. This method of carrying both addresses and data on the same signal lines is called a *multiplexed address/data bus*. It's a popular arrangement that many devices use, since it requires fewer pins on the chip, compared to giving each data and address line its own pin. Port 2's eight lines hold the higher byte of the address to be read or written to. These lines make up the high address bus (A8-A15). Together, the 16 address lines can access 64 kilobytes (65,536 bytes) of memory, from 00000000 00000000 to 11111111 11111111 in binary, or 0000h to FFFFh in hexadecimal. Besides pins to hold the data and addresses, the 8052 must also provide control signals to initiate the read and write operations. Control signals include WR (write), RD (read), PSEN (program store enable), and ALE (address latch enable). Some of the address lines may also function as control signals that help to select a chip during a memory access.

Code and data memory. To understand the operation of the control signals, you need to know a little about how the 8052 distinguishes between two types of

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

memory: data and code, or program, memory. By using different control signals for each type of memory, the 8052 can access two separate 64K areas of memory, with each addressed from 0000h to FFFFh, and each using the same data and address lines. The 8052 accesses code memory when it executes an assembly-language program or subroutine. Code memory is read-only; you can't write to it. The only instructions that access code memory are read operations. Code memory is intended for programs or subroutines that have been previously programmed into ROM or EPROM. The 8052 strobes, or pulses, PSEN when it accesses external code memory. Accesses to internal code memory (the BASIC-52 interpreter in ROM) do not use PSEN or any external control signals. Data memory is read/write memory, usually RAM. Instructions that read data memory strobe RD, and instructions that write to data memory strobe WR. The term*data memory* may be misleading, because it can hold any information that is accessed with instructions that strobe RD or WR. In fact, BASIC-52 programs are stored in data memory, not code memory as you might think. This is because the 8052 does not execute the BASIC programs directly. Instead, the BASIC-52 interpreter program reads the BASIC programs as data and then translates them to machine code for execution by the 8052. If you don't need all of the available memory space, you can combine code and data memory in a single area. With combined memory, WR controls write operations, and PSEN and RD are logically ANDed to create a read signal that is active when either PSEN or RD is low. Combined data/code memory is handy if you want the flexibility to store either BASIC or assembly-language programs in the same chip, or if you want to be able to upload assembly-language routines into RAM for testing. ALE is the final control signal for accessing external memory. It controls an external latch that stores the lower address byte during memory accesses. When the 8052 reads or writes to external memory, it places the lower address byte on AD0-AD7 and strobes ALE, which causes the external latch to save the lower address byte for the rest of the read or write cycle. After a short delay, the 8052 replaces the address on AD0-AD7 with the data to be written or read.

Timers and Counters. The 8052 has three 16-bit timer/counters, which make it easy to generate periodic signals or count signal transitions. BASIC-52 assigns optional functions for each of the timer/counters. Timer 0 controls a real-time clock that increments every 5 milliseconds. You can use this clock to time events that occur at regular intervals, or as the base for clock or calendar functions. Timer 1 has several uses in BASIC-52, including controlling a pulse-width-modu-lated output (PWM) (a series of pulses of programmable width and number); writing to a line Inside the 8052-BASIC printer or other serial peripheral (LPT); and generating pulses for EPROM programming (PGM PULSE). Timer 2 generates a baud rate for serial communications at SER IN and SER OUT. These are all typical applications for timer/counters in microcontroller circuits. If you don't use the optional timer functions, you can program the timers for other applications. In addition to timing functions, where the timer increments at a defined rate, you can use the timers for event counting, where the timer increments on an external trigger and measures the time between triggers. If you use the timers for event counting, T2, T2(EX), T0, and T1 detect transitions to be counted.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

The serial port. The 8052's serial port automatically takes care of many of the details of serial communications. On the transmit side, the serial port translates bytes to be sent into serial data, including adding start and stop bits and writing the data in a timed sequence to SER OUT. On the receive side, the serial port accepts serial data at SER IN and sets a flag to indicate that a byte has been received. BASIC-52 uses the serial port for communicating with a host computer.

External interrupts. INT0 and INT1 are external interrupt inputs, which detect logic levels or transitions that interrupt the CPU and cause it to branch to a predefined program location. BASIC-52 uses INT0 for its optional direct-memory-access (DMA) function.

Programming functions. BASIC-52's programming commands use three additional port bits (ALEDIS, PGM PULSE, and PGM EN) to control programming voltages and timing for storing BASIC-52 programs in EPROM or other nonvolatile memory. Additional Control Inputs Two additional control inputs need to be mentioned. A logic high on RESET resets the chip and causes it to begin executing the program that begins at 0 in code memory. In the 8052-BASIC chip, this program is the BASIC-52 interpreter. EA (external memory access) determines whether the chip will access internal or external code memory in the area from 0 to 1FFFh. In BASIC-52 systems, EA is tied high so that the chip runs the BASIC interpreter in internal ROM on boot-up.

Power Supply Connections And, finally, the chip has two pins for connecting to a +5-volt DC power supply (VCC) and ground (VSS). That finishes our tour of the 8052-BASIC chip. We're now ready to put together a working system.

Powering Up

This chapter presents a circuit that enables you to start using the 8052-BASIC chip. You can write and run programs and experiment with the BASIC-52 programming language.
The circuit has six major components: the 8052-BASIC chip, an address latch, an address decoder, static RAM, a EEPROM and an RS-232 interface. The circuit configuration is a more-or-less standard design, similar to many other microcontroller circuits. When you understand this circuit, you're well on your way to understanding many others. The following paragraphs explain the circuit operation, component by component.

The Microcontroller: EA, the External Access Enable input (pin 35, connects to +5V. This causes the 8052 to run the BASIC-52 interpreter in ROM on boot-up. If EA is low, the 8052 ignores its internal ROM and instead accesses external program memory on bootup. You can wire EA directly to +5V, or use a jumper as shown in the schematic, to allow you to bypass BASIC-52 and boot to an assembly-language program in external memory.

The crystal: XTAL1 is an 11.0592-Mhz crystal that connects to pins 20 and 21 of the 8052 chip. This crystal frequency has two advantages. It gives accurate baud rates for serial communications, due to the way that the 8052's timer divides the system clock to generate the baud rates. Plus, BASIC-52 assumes this frequency when it times the real-time clock, EPROM programming pulses, and serial printer port. However, you should be able to use any crystal value from 3.5 to 12 Megahertz. If you use a different value, you can use BASIC-52's XTAL operator to adjust the timing to match the frequency of the crystal you are using. The serial communications are reliable if the baud rate is accurate to within a few percent. The higher the crystal frequency, the faster your programs will execute, so most designs use either 11.0592 Mhz or 12 Mhz, which is the maximum clock frequency that the standard 8052 chip can use.
Two capacitors beside the crystal are 10 picofarads each, as specified in the 8052's data sheet. Their precise value isn't critical. Smaller values decrease the oscillator's startup time, while larger values increase stability.

Reset circuit: A logic high on pin 10 of the 852 chip resets the chip. On power up, pin 10 get first a high signal, falling slowly from +5V to 0 V as the capacitor charges through resistor. Connecting a small switch over the capacitor give you a reset switch.

External Memory: The remaining connections to the 8052 chip have to do with reading and writing to external memory.

Read and write signals: Writing to data memory is controlled by WRITE (*WR), reading is controlled by READ (*RD) signal. AD0-AD7 connect to a address latch, a 74ACT273 octal transparent latch that stores the lower address byte during memory accesses. The chip contains a set of D-type latches that store logic states.
A latch-enable input (C1) controls whether the outputs are latched (stored), or not latched (immediately follow the inputs). When pin 11 is high, 1Q-8Q follow 1D-8D. When pin 11 goes low, outputs 1Q-8Q will not change until pin 11 goes high again. During each external memory access, 1Q-8Q store the low address byte, so the eight lines that connect to these outputs carry the label LOW ADDRESS BUS.
Because AD0-AD7 hold the data to be read or written during a memory access, the signals as a group carry the label DATA BUS. Each line of AD0-AD7 has a 10K pullup resistor. You can use eight individual resistors, or a resistor network that contains eight resistors in a package. In a bussed resistor network, one pin connects to one side of all of the resistors, so you have fewer connections to wire.
The remaining bus is the HIGH ADDRESS BUS (A8-A15), which consists of the upper eight address lines, and is not multiplexed.

Address decoding: A PALce22V10 is used for address decoding. It functions as an address decoder for the 64K external memory space. Address decoding allows multiple chips to connect to the address and data buses, with each chip enabled only when it is selected.
If you use a 32K RAM: For all of the 32K RAM's addresses (0 to 7FFFh), A15 is low, and for all other addresses (7FFFh to FFFFh), A15 is high.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

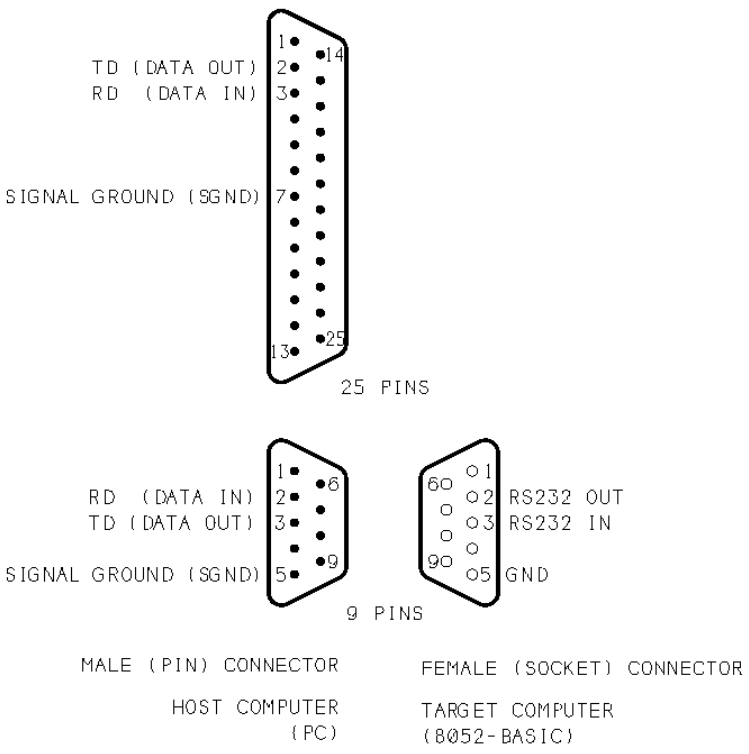
RAM choices: The minimal circuit includes just one RAM memory chip, which can be an 8K or 32K static RAM, or SRAM. BASIC-52 requires at least 1K of RAM, but I've used the larger capacities, since the extra room is useful and doesn't cost much more. The 8K chip has 13 address inputs (A0-A12), while the 32K chip has 15 (A0-A14). Eight data I/O pins (I/O1-I/O8) connect to the data bus and hold the bytes to be read or written. The RAM has three control inputs whose functions match those of the 8052's control outputs. Pin 20 (CS, or ChipSelect) enables then RAM whenever the 8052 reads or writes to the chip, with the address decoding determining the address range of the chip. Pin 27 (WE, or Write Enable) is driven by WRITE, and is strobed low during each write to external data memory. Pin 22 (OE, or Output Enable) is driven by READ, and strobes low when either external data or code memory is read.
Each write cycle follows this sequence: The 8052 brings ALE high and places the address to be written to on AD0-AD7 and A8-A15. For addresses from 0 to 7FFFH, A13-A15 are low, so the RAM is selected at its pin 20. After a short delay, the 8052 brings ALE low, which causes U7 to store the lower address byte. After another short delay, the 8052 replaces the address on AD0-AD7 with the data to be written. A low pulse at pin 27 (WE) causes the RAM to write the data into the address specified by A0-A12. Read cycles are similar, except that a pulse at pin 22 (OE) causes the requested data to appear on AD0-AD7, where the 8052 reads it.
Static RAM chips are rated by their read-access time, which is the maximum time the chip will require to place a byte on the data bus after a read is requested. With a crystal frequency of 12 Mhz or lower, an access time of 250 nanoseconds or less is fine for accessing external data or code memory. Access times and other timing characteristics are described in the timing diagrams in the data sheets for the 8052 and RAM. When you use the 8052-BASIC, you don't have to worry about any of these specifics about the read and write cycles. If the circuit is wired correctly, and if all of the components are functioning as they should, reading and writing occur automatically in the course of executing BASIC-52 statements and commands. A single program line in BASIC-52 can cause dozens or more read and write operations to occur.

Power Supply: A final essential component is the power supply. For the basic system, all you need is a regulated +5-volt supply. An output capability of at least 500 milliamperes is recommended. Some onboard capacitors provide power-supply decoupling. Digital devices draw current as they switch. These capacitors store energy that the components can draw quickly, without causing spikes in the supply or ground lines. The exact values aren't critical, but they should be a type with good high-frequency response, such as ceramic, mica, or polystyrene.

Serial Interface: For the serial line the TTL level signals from the 8052 chip must be converted to the RS-232 standard levels. For this two transistors are used, switching between a +/- supply. The - level for the output line is generated from the - signal at the input line. So the circuit can run with only a +5 Volt power supply.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

Connections to RS-232 OUT and RS-232 IN depend on the type of serial connector you have on your personal computer or its serial cable.
Connectors vary, but two common ones are a male 25-pin or 9-pin D-connector. (The outer
shell of a D-connector is roughly in the shape of a *D*.) For the 8052-BASIC system, you'll need a mating female 9-pin D-Connector. The connection has just three wires.
The Figure shows the wiring for 9- and 25-pin connectors. A few computers require additional handshaking signals. BASIC-52 doesn't support these, but you can simulate them by connecting together pins 5, 6, 8, and 20 at the personal-computer end of the link. (Pin numbers are for a 25-pin connector.)



Powering Up

The first time you power up an untested circuit, it pays to be cautious. I recommend the following steps:

First Steps

Visually inspect the circuit. You don't have to spend a lot of time on this, but sometimes a missing or miswired wire or component or another problem will become obvious. Set Jumper B1 to VCC (means internal ROM-BASIC). With an ohmmeter, measure the resistance from +5V to ground, to be sure these aren't shorted together by mistake. The exact value you measure isn't critical, but if you read less than 100 ohms, something is miswired and you need to find and fix the problem before you continue. If you suspect a problem, check the wiring of the power and ground connections and be sure all components are oriented correctly.
When all checks out, you're ready to boot up BASIC-52.

Booting BASIC-52

For the initial check, begin with everything powered down. I'll use the term *host computer,* or *host system,* to refer to the personal computer, and *target computer,* or *target system*, to refer to the 8052-BASIC circuits. Included are some specific tips for users of Datastorm's *Procomm Plus* for DOS and Microsoft Windows 3.1's Terminal.

Accessory, but other communications software should have similar features and abilities. Turn on the host computer and run your communications software. Configure the software for 8 data bits, no parity, and 1 stop bit. The baud rate you select isn't critical, since BASIC-52 automatically adjusts to what you are using. To start, use a rate of 9600 or less. Don't enable any handshaking or flow-control options such as XON/XOFF or RTS/CTS. Select the appropriate serial, or COM, port, if necessary. If you're using an MS-DOS (IBM-compatible) computer, you must find a COM port and interrupt-request (IRQ) level that aren't being used by your modem, mouse, or another device. Because COM1 and COM3 often share an IRQ level, as do COM2 and COM4, you generally can't use COM1 and COM3 at the same time, or COM2 and COM4. If you have an external modem, you can unplug it and use its serial port. In *Procomm Plus,* use the line/port setup menu (ALT+P) to configure. In the Windows Terminal, use the Settings menu. Cable together the serial ports of the host and target systems.

Windows Terminal accessory for communications.
You're now ready to power up the target system. Turn on its power supply, and press the SPACE bar at the host's keyboard. You should see this BASIC-52 sign-on message and prompt:

```
*MCS-51(tm) BASIC V1.1*
READY
```

Troubleshooting

If you don’t see the prompt, it’s time to troubleshoot. Getting the system to boot up the first time can be the most challenging part of a project, especially when serial communications are involved. Here are some things that may help you isolate the cause of the problem:

- Try again by reseting the 8052-chip and pressing the space bar. If you are using a 32K RAM, BASIC-52 requires about 1 second to perform its memory check after a reset, before it will respond to the space bar. (proportionately longer with slower crystals).
- Double-check the easy things. Are the communications parameters correct? Did you select the correct serial port? Are all ICs inserted?
- Verify that pin 10 of the 8052-chip goes low, then high when you reseting the CPU.
- Check the power and ground pins of all ICs for proper voltages.
- Connect a oscilloscope probe to pin 11 (RXD) of the 8052-chip. When you press the space bar, you should see the logic level toggle as receiving the ASCII code for a space (20h). If not, you probably have a problem in the serial line interface, in your communications software or in the serial cabling.
- Verify that pin 33 of the 8052-chip is toggling (at 1/6 the crystal frequency, if you have an oscilloscope to measure). This indicates that the oscillator circuit is functioning.

Basic tests

When your system boots, you’re ready for some basic tests. The BASIC-52 programming manual is a useful reference at this point.
In some ways, BASIC-52 is similar to BASIC compilers like Microsoft’s *QuickBASIC*. Many of the keywords and syntax rules are similar. But BASIC-52 is closer to older interpreted BASICs like *GW-BASIC* or *BASICA.* You can type a statement or command and execute it immediately when you press ENTER, or you can type a series of statements and run them later as a program. When a line begins with a line number, BASIC-52 treats it as a program line rather than as a command to execute immediately.

Here are some quick tests and experiments you can do:

Memory Check

```
        PRINT MTOP
```

to learn the amount of external data memory that BASIC-52 detected on boot-up. With an 8K RAM, MTOP should be 8191, and with 32K, it should be 32,767. If you prefer hexadecimal notation, type: `PH0. MTOP`

Crystal Frequency

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

The special operator XTAL represents the value of the timing crystal that clocks the 8052-BASIC. The default value is 11059200, or 11.0592 Mhz. You can verify this by typing:

```
PRINT XTAL
```

Most BASIC-52 statements don't use the XTAL operator, so it doesn't matter if the value sn't accurate. Exceptions are the real-time clock, programming commands, PWM output, nd LPT output. For these, XTAL should match your crystal's frequency. To set XTAL for a 12 Mhz crystal, type:

```
XTAL=12000000
```

Line Editing

After typing a few commands, you may discover some of BASIC-52's line-editing abilities. While typing a line, you can correct mistakes by deleting back to the mistake and retyping. In *Procomm Plus,* if you select VT100 terminal emulation (under Setup menu, Terminal Options), you can use either the DELETE or BACKSPACE key to delete. With the *Windows* terminal, you must use the DELETE key (not BACKSPACE). Many communications programs allow you remap the keyboard, so you can select whatever delete key you wish. Once you press ENTER, you can't edit a line you've typed, unless you retype it from the beginning.
BASIC-52 treats upper and lower-case characters the same. In most cases, spaces are ignored, so you can include them or not as you wish.

Running a Program

Here is a very simple program to try:

```
10 FOR I=1 to 4
20 PRINT „Hallo=",I
30 NEXT I
20 END
```

Enter each of the lines, including the line numbers. BASIC-52 automatically stores the program in RAM. To run the program, type RUN. You should see this:

```
Hallo=1
Hallo=2
Hallo=3
Hallo=4
```

To view the program lines, type	LIST
To erase the current program, type	NEW

You can change individual program lines by typing the line number, followed by a new statement:

```
10 FOR I=1 to 20
```

To erase a line, type the line number and press ENTER:

```
20
```

Heinz Rongen Forschungszentrum Jülich, ZEL 52425 Jülich, Germany
Tel: +49-(0)2461-614512 Fax: +49-(0)2461-613990 Email: H.Rongen@fz-juelich.de

Getting Out of Trouble

Occasionally, a programming error may cause a program to go into an endless loop or crash the system. If it's an endless loop, you can exit it and return to the READY prompt by pressing CONTROL+C. If that doesn't work, your only choice is to reset the 8052-BASIC system. Resetting will erase the program in RAM, so you'll have to reenter it.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

Simple things to try

The following sections offer some short programs to try, to help you explore your system and become familiar with BASIC-52. Don't worry if you don't understand every line of the programs.

Reading, Writing Port 1

You can use BASIC-52 to read and write to Port 1 (pins 2-9) on the 8052-BASIC.
The command

```
PRINT PORT1             or       PH0.PORT1
```

will display the value of the entire port in adecimal or hex number. If a port pin is open, or unconnected, its internal pull-up resistor will cause it to read as 1. If you connect a jumper wire from a port pin to ground, or bring the pin low by driving it with a logic low output, it should read 0.

You can control the bits of Port 1 by writing to them.

```
PORT1 = 123
```

Accessing Memory

You can read and write to the external memory with the XBY command.

PRINT XBY (8000h)	read the contents of memory 8000h and prints the value to the screen
XBY (8000h) = 123	writes the value 123 into memory cell 8000h

If you write to an address outside the range specified as free memory, you will overwrite the RAM currently in use to store your program and run BASIC-52. If you do this accidentally, your system may crash and you'll have to reset the system and reenter the program.

Real-time Clock

The Listing demonstrates BASIC-52's real-time clock by displaying an on-screen 60-second timer. For the timer to be accurate, you must set XTAL to match the timing crystal your system uses.

```
10 CLOCK 1:TIME=0:SEC=0
20 DO
30 ONTIME 1,60
40 WHILE SEC<60
50 END
60 TIME=TIME-1
70 SEC=SEC+1
80 PRINT SEC
90 RETI
```

Wait for CONTROL+C

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

You can always end a program by pressing CONTROL+C at the host’s keyboard. The only exceptions are runaway programs that have crashed the system and force you to reboot.

Quick Reference to BASIC-52

This quick reference to the BASIC-52 programming language lists the most important basic keywords alphabetically, along with brief descriptions of function and use.

C = command mode
R = run mode

variable **=** *expression* C,R
Assigns a value to a variable

expression **+** *expression* C,R
expression **-** *expression* C,R
expression ***** *expression* C,R
expression **/** *expression* C,R
Add, Subtract, Multiply, Divide

expression ****** *expression* C,R
Raises first expression to value of second expression (exponent)

expression **<>** *expression* C,R
Inequality test (relational operator)

expression **<** *expression* C,R
Less than test (relational operator)

expression **>** *expression* C,R
Greater than test (relational operator)

expression <= expression C,R
Less than or equal test (relational operator)

expression >**=** *expression* C,R
Greater than or equal test (relational operator)

ABS (*expression*) C,R
Returns the absolute value of *expression*

expression **.AND.** *expression* C,R
Logical AND

ASC(*character*) C,R
Returns the value of ASCII character

ATN(*expression*) C,R
Returns the arctangent of *expression*

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

BAUD *expression* C,R
Sets the baud rate for LPT (pin 8). For proper operation, XTAL must match the system's crystal frequency.

CHR(*expression*) C,R
Converts *expression* to its ASCII character.

CLOCK0, CLOCK1 C,R
Disables, Enables the real-time clock.

CONT C
Continues executing program after STOP or CONTROL+C.

SIN, COS, TAN (*expression*) C,R
Returns the Sine, Cosine, or Tangent of *expression*

DATA *expression* [*,...,expression*]R
Specifies expressions to be retrieved by a READ statement.

DIM *array name* [(*size*)] [*,...array name*(*size)*] C,R
Reserves storage for an array. Default size is 11 (0-10). Size limits are 0-254.
Example: DIM B(100) Reserves storage for 100-element array B

DO: [*program statements*]**: UNTIL** *relational expression* R
Executes all statements between DO and UNTIL until *relational expression* is true.

DO: [*program statements*]**: WHILE** *relational expression* R
Executes all statements between DO and WHILE until *relational expression* is false.

END R
Terminates program execution.

EXP (*expression*) C,R
Raises *e* (2.7182818) to the power of *expression*

FOR *counter variable* **=** *start-count expression* C,R
TO *end-count expression* [
STEP *count-increment expression*]**:** [*program statements*]**:**
NEXT [*counter variable*]
Executes all statements between FOR and NEXT the number of times specified by the counter and step expressions.

FREE C,R
Returns the number of bytes of unused external data RAM.

GET R

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

Contains the ASCII code of a character received from the host computer’s keyboard. After a program reads the value of GET (For example, G=GET), GET returns to 0 until a new character arrives.

GOSUB *line number* R
Causes BASIC-52 to transfer program control to a subroutine beginning at *line number.* A RETURN statement returns control to the line number following the GOSUB statement.

GOTO *line number* C,R
Causes BASIC-52 to jump to *line number* in the current program.

IDLE R
Forces BASIC-52 to wait for ONTIME or ONEX1 interrupt.

IF *relational expression* R
THEN *program statements*
[**ELSE**] [*program statements*]
If *relational expression* is true, executes program statements following THEN.
If *relational expression* is false, executes program statements following ELSE.

INPUT [**“***Prompt message***”**][,] *variable* [*,variable*] [*,...variable*] R
Displays a question mark and optional prompt message on the host computer and waits for keyboard input. Stores input in *variable*(s). A comma before the first variable suppresses the question mark.

INT(*expression*) C,R
Returns integer portion of *expression*.

LEN C,R
Returns the number of bytes in the current program

[**LET**] *variable* **=** *expression* C,R
Assigns a variable to the value of *expression*. Use of LET is optional.

LIST[*line number*][*-line number*] C,R
Displays the current program on the host computer.

LIST# [*line number*][*-line number*] C,R
Writes the current program to LPT (pin 8).

LOG(*expression*) C,R
Returns natural logarithm of *expression*.

MTOP [**=***highest address in RAM program space*] C,R

Heinz Rongen Forschungszentrum Jülich, ZEL 52425 Jülich, Germany
Tel: +49-(0)2461-614512 Fax: +49-(0)2461-613990 Email: H.Rongen@fz-juelich.de

Assigns or reads the highest address BASIC-52 will use to store variables, strings, and RAM programs. Usually 7FFFh or lower, since EPROM space begins at 8000h.

NEW C
Erases current program in RAM; clears all variables.

NOT (*expression*) C,R
Returns 1's complement (inverse) of *expression*.

NULL [*integer*] C
Sets the number (0-255) of NULL characters (ASCII 00) that BASIC-52 sends automatically after a carriage return. Only very slow printers or terminals need these extra nulls.

ON *expression* **GOTO** *line number* [*,line number*] [*,...,line number*] R
ON *expression* **GOSUB** *line number* [*,line number*] [*,...,line number*] R
Transfers program control to a subroutine beginning at one of the line numbers in the list. The value of *expression* matches the position of the line number selected, with the first line number at position 0.
Examples:

```
X=1
ON X GOTO 100,200,400
```

Transfers program control to a subroutine at line 200 (position 1 in the list)

ONERR *line number* R
Passes control to *line number* following an arithmetic error.
Arithmetic errors in-clude ARITH. OVERFLOW, ARITH. UNDERFLOW, DIVIDE BY ZERO, and BAD ARGUMENT.

ONEX1 *line number* R
On interrupt 1 (pin 13), BASIC-52 finishes executing the current statement, and then passes control to an interrupt routine beginning at *line number*. The interrupt routine must end with RETI.

ONTIME *number of seconds, line number* R
When TIME = *number of seconds,* BASIC-52 passes control to an interrupt routine beginning at *line number.* The interrupt routine must end with RETI. CLOCK1 starts the timer.

expression **.OR.** *expression* C,R
Logical OR

PH0. C,R
Same as PRINT, but displays values in hexadecimal format. Uses two digits to display values less than 0FFh.

PH0.# C,R

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

Same as PRINT#, but displays values in PH0. hexadecimal format

PH0.@ C,R
Same as PRINT@, but outputs values in PH0. hexadecimal format.

PH1. C,R
Same as PRINT, but displays values in hexadecimal format. Always displays four digits.

PH1.# C,R
Same as PRINT#, but displays values in PH1. hexadecimal format.

PH1.@ C,R
Same as PRINT@, but outputs values in PH1. hexadecimal format.

PI C,R
Constant equal to 3.1415926.

PORT1 C,R
Retrieves or assigns a value to PORT1 (pins 1-8).

PRINT [*expression*] [*,...expression*] [,] C,R
Displays the value of *expression*(s) on the host computer. A comma at the end of the statement suppresses the CARRIAGE RETURN/LINEFEED. Values are separated by two spaces. Additional PRINT options are CR, SPC, TAB, USING.

PRINT# C,R
Same as PRINT, but outputs to LPT (pin 8). BAUD and XTAL values affect the PRINT# rate.

PROG C
Stores the current RAM program in the EPROM space.

PROG1 C
Saves the serial-port baud rate. On power-up or reset, BASIC-52 boots without having to receive a space character. The terminal's baud rate must match the stored value.

PROG2 C
Like PROG1, but on power-up or reset, BASIC-52 also begins executing the first program in the EPROM space.

PROG3 C
Like PROG1, but also saves MTOP. On power-up or reset, BASIC-52 clears memory only to MTOP.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

PWM *expression1, expression2, expression3* C,R
Outputs a pulse-width modulated (PWM) sequence of pulses on pin 3. *Expression1* is the width of each high pulse, expressed in clock cycles. *Expression2* is the width of each low pulse, expressed in clock cycles. *Expression3* is the number of PWM cycles output. One clock cycle = 12/XTAL. One PWM cycle = one high pulse plus one low pulse. *Expression1* and *Expression2* must each be at least 25. Maximum for each *Expression* is 65535.

RAM C
Selects the current program in the RAM space.

READ *variable* [*,...,variable*] R
Retrieves the expressions in a DATA statement and assigns each expression to a variable.

REM C,R
Introduces a comment, or remark. BASIC-52 ignores all text after REM in a program line.

RESTORE R
Resets READ pointer to the first expression in the DATA statement.

RETI R
Returns program control to the line number following the most recently executed ONEX1 or ONTIME statement.

RETURN R
Returns program control to the line number following the most recently executed GOSUB statement.

RND C,R
Returns a pseudo-random number between 0 and 1 inclusive.

ROM [*program number*] C
Selects a program in the EPROM space (beginning at 8000h). Default program number is 1.

RROM [*program number*] C,R
Changes to ROM mode and runs the specified program.
Default program number is 1.

RUN R
Executes the current program. Clears all variables.

SGN (*expression*) C,R
Returns +1 if *expression* >=0, zero if *expression* = 0, and -1 if *expression* <0.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

SPC(*expression*)
PRINT option. Causes the display to place *expression* additional spaces (besides the minimum two) between values in a PRINT statement.
Example:

```
PRINT “hello”,SPC(3),"good-by"
hello     good-by
```

SQR(*expression*) C,R
Returns square root of expression.

STOP R
Halts program execution.

STRING *expressions, expression2* C,R
Allocates memory for strings (variables each consisting of a series of text characters). *Expression1* is the total number or characters the user which to allocate. *Expression2* denotes the maximum characters in each string.

Expression1= (*Expression2* * number of strings) + 1.
Expression2 = maximum number of bytes (characters) per string + 1.

Executing STRING clears all variables. Maximum number of strings is 255.
Examples:

STRING 91, 9	reserves space for ten 8-character strings
STRING 9, 4	reserves space for two 3-character strings

TAB(*expression*),
PRINT option. Specifies the position (number of spaces) to begin displaying the next value in the PRINT statement. Example:

```
PRINT TAB(5) “hello”
     hello
PRINT TAB(2) “hello”
  hello
```

TIME C,R
Retrieves or assigns a value, in seconds, to BASIC-52’s real-time clock.

TIMER0 C,R
Retrieves or assigns a value to the 8052’S special function registers TH0 and TL0.

TIMER1 C,R
Retrieves or assigns a value to the 8052’s special function registers TH1 and TL1.

TIMER2 C,R
Retrieves or assigns a value to the 8052’s special function registers TH2 and TL2.

TMOD C,R
Retrieves or assigns a value to the 8052’s special function register TMOD.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

USING (**F***N*)
PRINT option. Causes BASIC-52 to output numbers in exponential format with *N* significant digits. BASIC-52 always outputs at least 3 significant digits. Maximum *expression* is 8.
Example:

```
PRINT USING(F3),3,4.1,100
3.00 E 0
4.10 E 0
1.00 E 2
```

USING (0)
PRINT option. Causes BASIC-52 to output numbers from ±.99999999 to ±0.1 as decimal fractions. Numbers outside this range display in USING(FN) format. USING(0) is the default format.

USING (#[...#][.]#[...#*]*)
PRINT option. Causes BASIC-52 to output numbers using decimal fractions, with # representing the number of significant digits before and after the decimal point. Up to eight # characters are allowed.
Example:

```
PRINT USING(###.##),3,4.1,100
3.00
4.10
100.00
```

XBY(*expression*) C,R
Retrieves or assigns a value in external data memory.

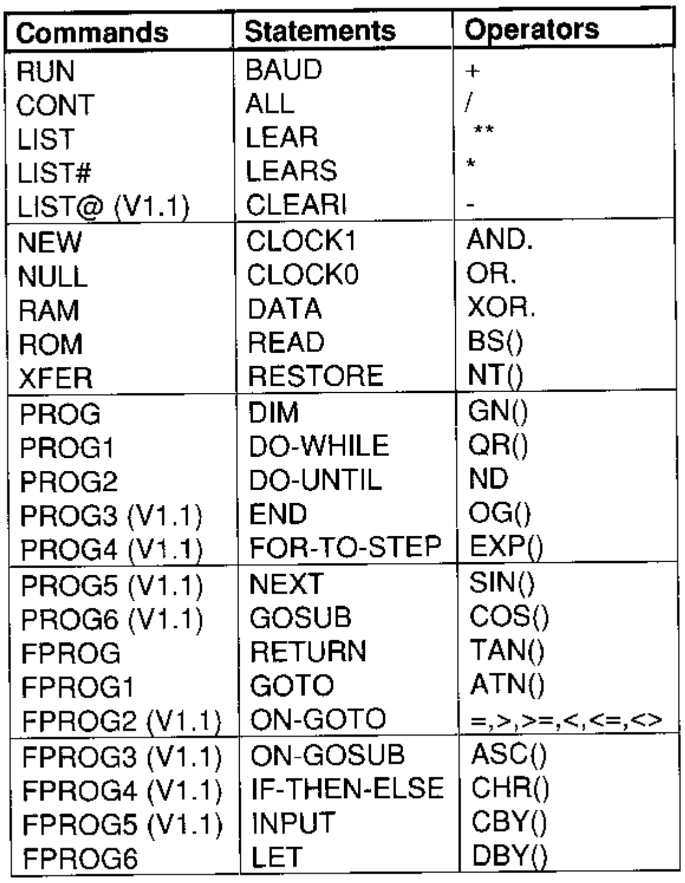
XFER C
Copies the current program from the EPROM space (beginning at 8010h for program 1) to RAM (beginning at 200h), and selects RAM mode.

expression **.XOR.** *expression* C,R
Logical exclusive OR

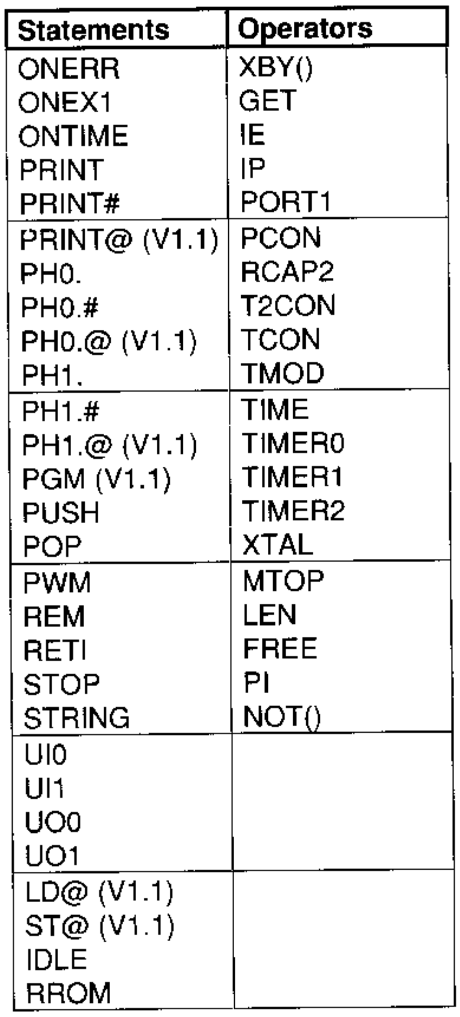
XTAL C,R
Assigns a value equal to the system’s crystal frequency, for use by BASIC-52 in timing calculations.

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*

The 8052-Basic command table:



Commands	Statements	Operators
RUN	BAUD	+
CONT	ALL	/
LIST	LEAR	**
LIST#	LEARS	*
LIST@ (V1.1)	CLEARI	-
NEW	CLOCK1	AND.
NULL	CLOCK0	OR.
RAM	DATA	XOR.
ROM	READ	BS()
XFER	RESTORE	NT()
PROG	DIM	GN()
PROG1	DO-WHILE	QR()
PROG2	DO-UNTIL	ND
PROG3 (V1.1)	END	OG()
PROG4 (V1.1)	FOR-TO-STEP	EXP()
PROG5 (V1.1)	NEXT	SIN()
PROG6 (V1.1)	GOSUB	COS()
FPROG	RETURN	TAN()
FPROG1	GOTO	ATN()
FPROG2 (V1.1)	ON-GOTO	=,>,>=,<,<=,<>
FPROG3 (V1.1)	ON-GOSUB	ASC()
FPROG4 (V1.1)	IF-THEN-ELSE	CHR()
FPROG5 (V1.1)	INPUT	CBY()
FPROG6	LET	DBY()



Statements	Operators
ONERR	XBY()
ONEX1	GET
ONTIME	IE
PRINT	IP
PRINT#	PORT1
PRINT@ (V1.1)	PCON
PH0.	RCAP2
PH0.#	T2CON
PH0.@ (V1.1)	TCON
PH1.	TMOD
PH1.#	TIME
PH1.@ (V1.1)	TIMER0
PGM (V1.1)	TIMER1
PUSH	TIMER2
POP	XTAL
PWM	MTOP
REM	LEN
RETI	FREE
STOP	PI
STRING	NOT()
UI0	
UI1	
UO0	
UO1	
LD@ (V1.1)	
ST@ (V1.1)	
IDLE	
RROM	

Heinz Rongen *Forschungszentrum Jülich, ZEL* *52425 Jülich, Germany*
Tel: +49-(0)2461-614512 *Fax: +49-(0)2461-613990* *Email: H.Rongen@fz-juelich.de*