A SIMPLE GUIDE TO DEVELOPMENT TOOLS

by K. Bigué

WHAT ARE DEVELOPMENT TOOLS ?

A microcontroller is a highly integrated device which generally carries out an application control function. To achieve this specific task, you must program several (usually several hundred!) instructions in the microcontroller. To develop and load the program, you require a set of devices. There are tools to create efficient and clearly structured programs and tools to transfer and to test these programs in the microcontroller.

The creation tools are software tools. They will allow you to generate your program code. This code is loaded in the microcontroller by the transfer tools which are hardware tools. These devices also allow all the microcontroller functions to be evaluated. You require hardware development tools for final debugging and test.

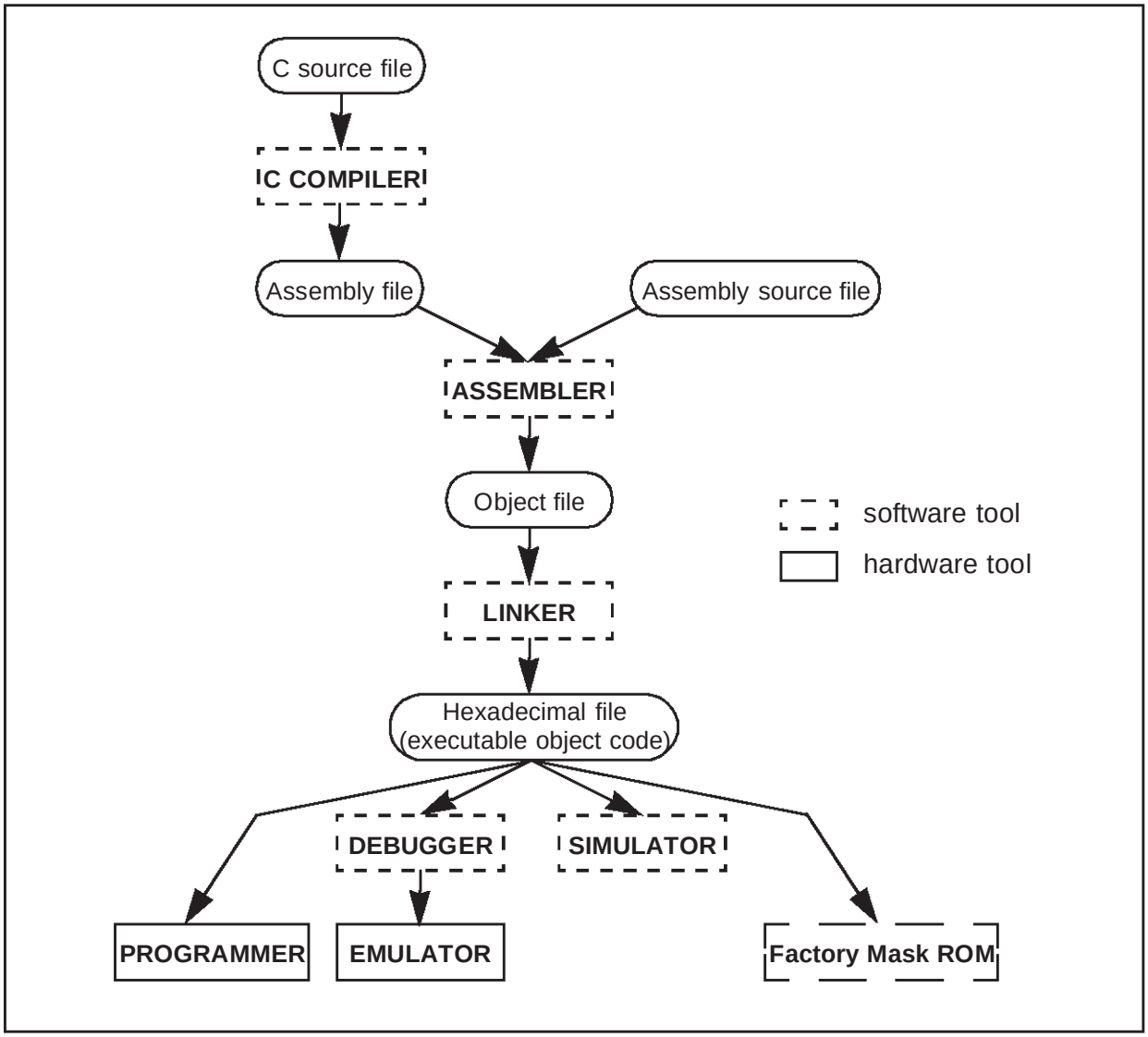
Development tools consist of a package of software tools and hardware tools. They serve to program and to evaluate one or several microcontrollers. Each microcontroller family needs both hardware and software development tools. All software development tools must follow the same principles: reliability, efficiency, ease of use, flexibility. These characteristics are especially important for embedded applications.

1 SOFTWARE DEVELOPMENT TOOLS

The software tools allow the development of your application code. This code will be loaded in the microcontroller to execute the corresponding functions.

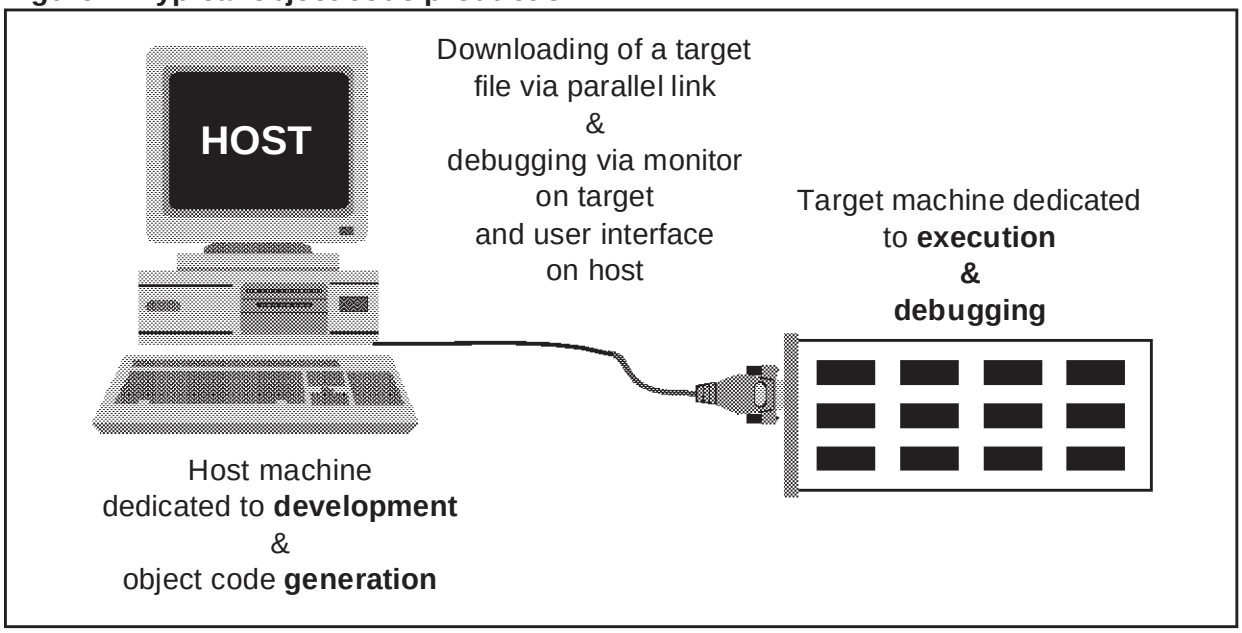
To produce a code ready for execution, you must run the **C compiler** (if program is written in C language), the **assembler** and the **linker**. Then, you must evaluate this code with the **debugger** or the **simulator**.

Figure 1. Code development flow chart



The **C compiler** used with the **assembler** and **linker**, allows the generation of executable object code. The generated object code may be used to debug, to generate EPROM devices test or to produce ROM mask data.

Figure 2. Typical object code production



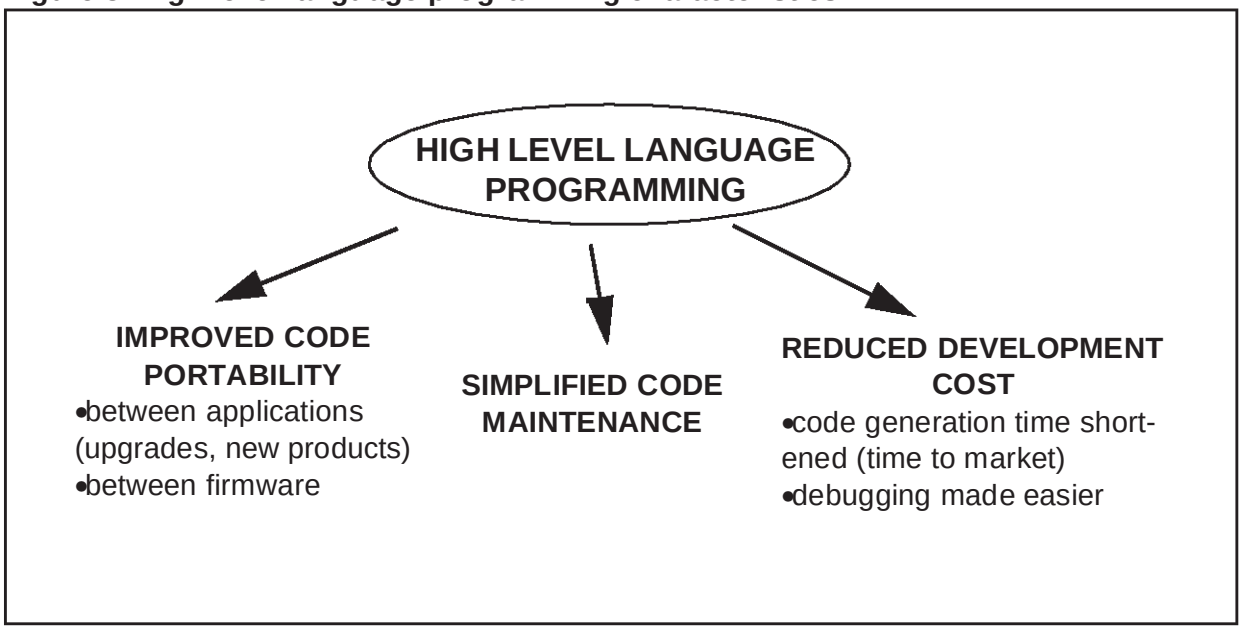
1.1 C COMPILER

The C compiler translates C-code into assembly code. It allows you to write in C and to benefit from all its advantages. It is easier to program using a high level language. But for small applications, the generated code can be less compact than by direct assembly programming.

1.1.1 High level language characteristics

It is highly advisable to write your software using a high level language. It allows you to structure your source code and improve program portability and reusability. Routines written in high level language can be also used in other firmware or applications. Furthermore, development costs are usually lower because code generation time is shortened and debugging is made easier.

Figure 3. High level language programming characteristics



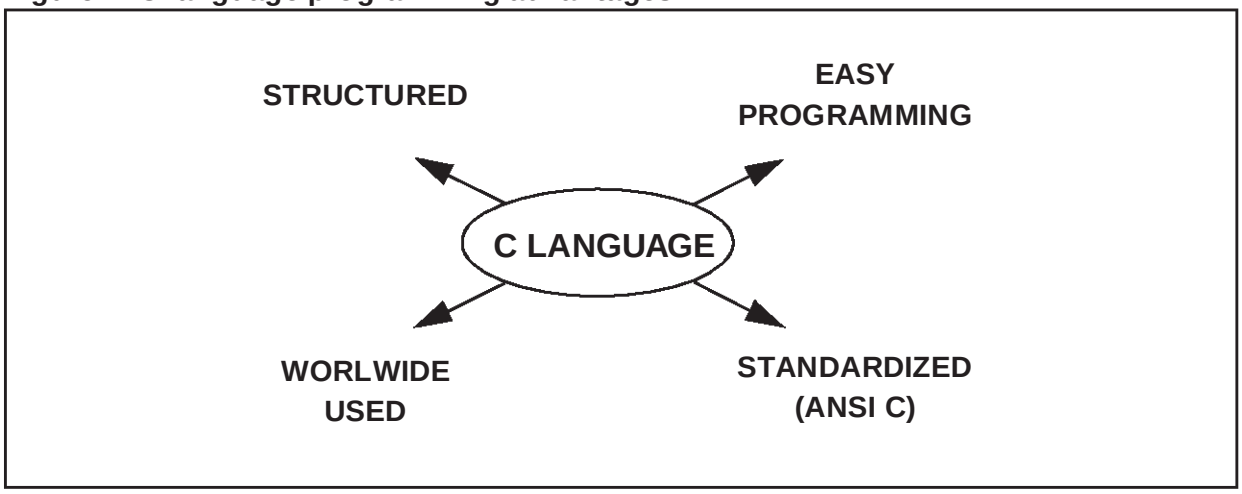
1.1.2 Advantages and drawbacks of C language

The C language is an example of a high level language i.e. the instructions are expressed in a language similar to natural language. It was developed in Bell laboratories by Brian Kernigan and David Ritchie in the early seventies. The language is rich: it offers a library of ready-programmed functions. The C language is very flexible and permits functions to be written with very few lines of code. This makes program development much easier.

Furthermore the program is more reliable because C language is a standardized language following the ANSI norm. It ensures that your code will meet some standard criteria.

Sometimes C language is not practical. For example, when a part of the program requires very fast processing, this section should be written in assembly language. Also to get precise control of hardware resources, it is sometimes useful to write a part of an application in assembly language. It is quite easy to interface an assembly function with a C program or a C function in an assembly program.

Figure 4. C language programming advantages

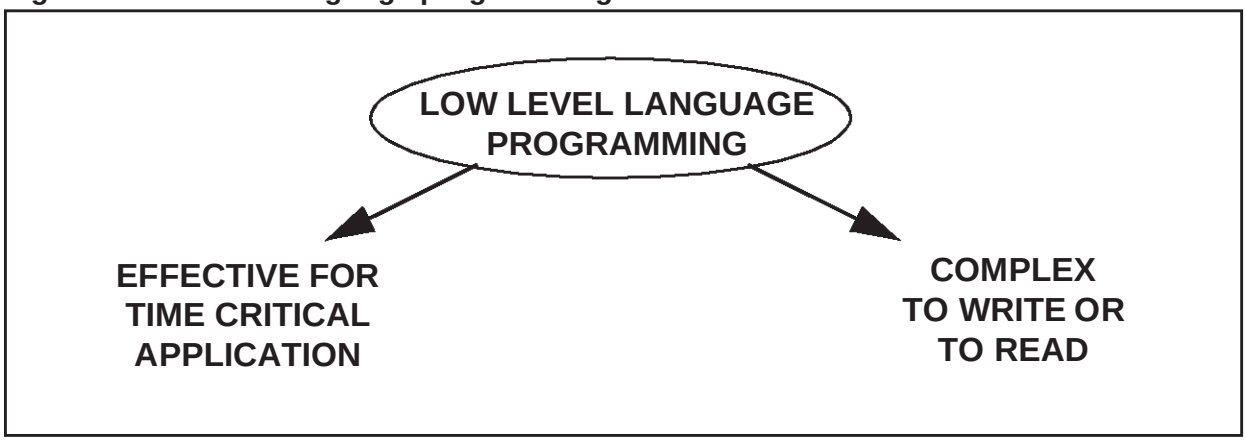


It is possible to mix other languages with the C compiler. You can include assembly instructions with access to C program symbols. Assembler must be used for critical instructions. Thus execution time is optimized.

1.1.3 Low level language characteristics

A low level language is a language close to machine code i.e. it generates fast and compact code which means faster execution and lower memory requirements. It is more difficult to write and harder to read. But you can use all the capabilities of the machine.

Figure 5. Low level language programming characteristics



1.1.4 Advantages and drawbacks of assembly language

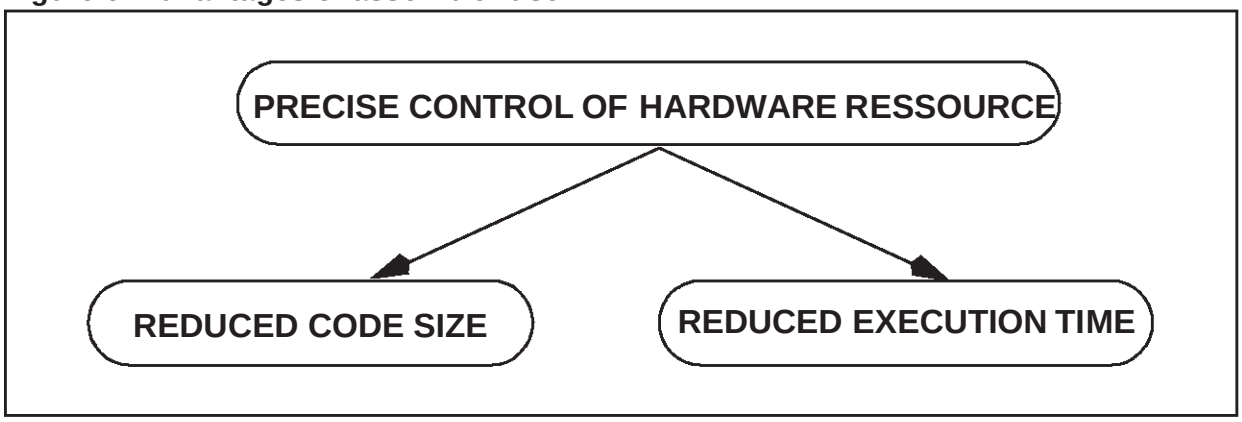
Assembly language is a low level language. It provides very fast execution times and is very useful for programs which continuously calculate large amounts of data. However, it often requires a large number of code lines whereas a high level language uses only one statement. Its complexity results in high level languages being used more and more often and assembly language only for time critical applications.

1.2 ASSEMBLER

The assembler transforms an assembly file into an object file. It assembles the instructions before translating it automatically into a language understandable by the machine. This machine language is hexadecimal code. It would be hard for you to write or to read machine language directly so you must use assembly language to make programming possible.

The assembler works as follows: it accepts one or more source files written in assembly language and changes them into relocatable object files. During the process, the assembler checks for many types of errors and it reports when you use a wrong instruction or operator.

Figure 6. Advantages of assembler use



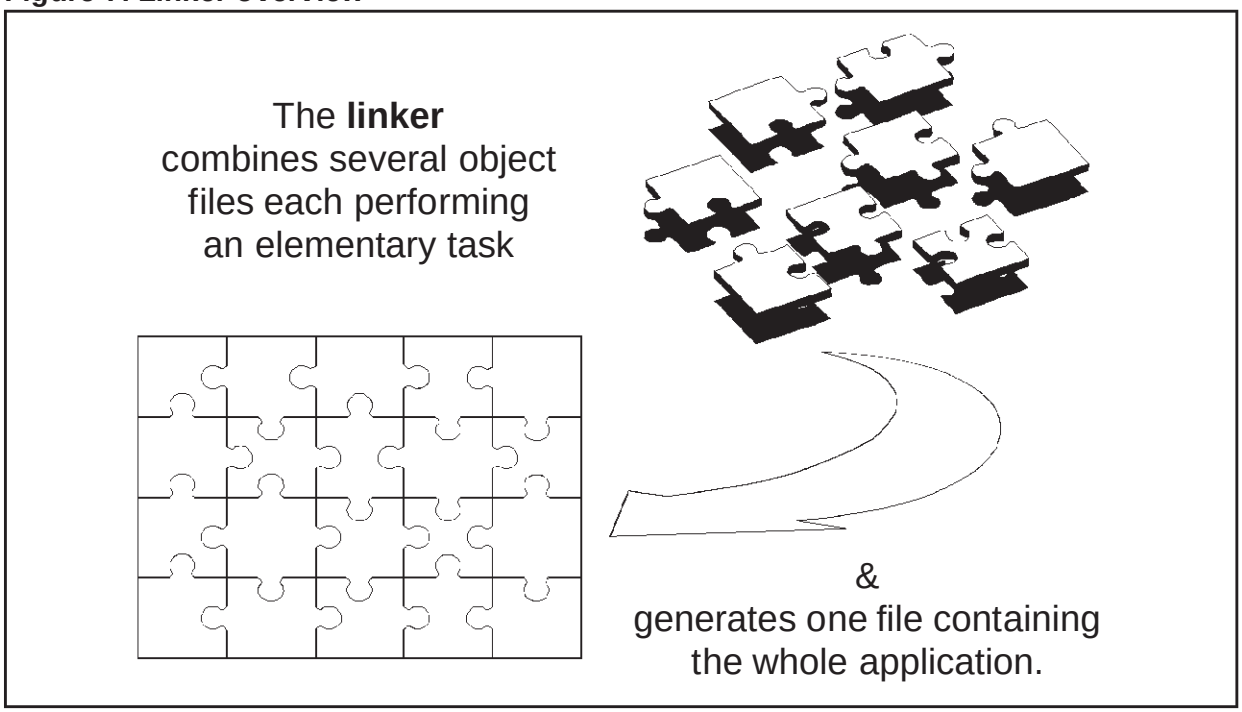
After having separately assembled all the component modules in your program, the next step is to link them together into a file which can then be sent onto its final destination.

1.3 LINKER

Nearly all applications are too large to be developed in a single file: this file would become too long to be easily managed by editors, and editing it would become tedious. Applications broken down to several pieces of code are much easier to work with. Separating each application into functional units and managing one file simplifies development and maintenance.

As a final step, it is necessary to group all these parts of the application together, and to produce one file containing the whole application. This is what the linker does.

Figure 7. Linker overview



The linker combines the different object code files issued by the *assembler* and places the code at predefined addresses in memory. It produces an output file in a hexadecimal format, which can be downloaded to the *emulator* by the **debugger**.

It is the last step in building a compiled program and making an executable application. In some cases, additional steps may follow, but these are only to format the output of the linker in another structure.

After your program has been assembled and linked to form a executable file, it needs to be sent to the place where it will be executed.

1.4 SIMULATOR

Using a simulator, you can debug programs written for a microcontroller and execute them on a computer, with no need for additional hardware.

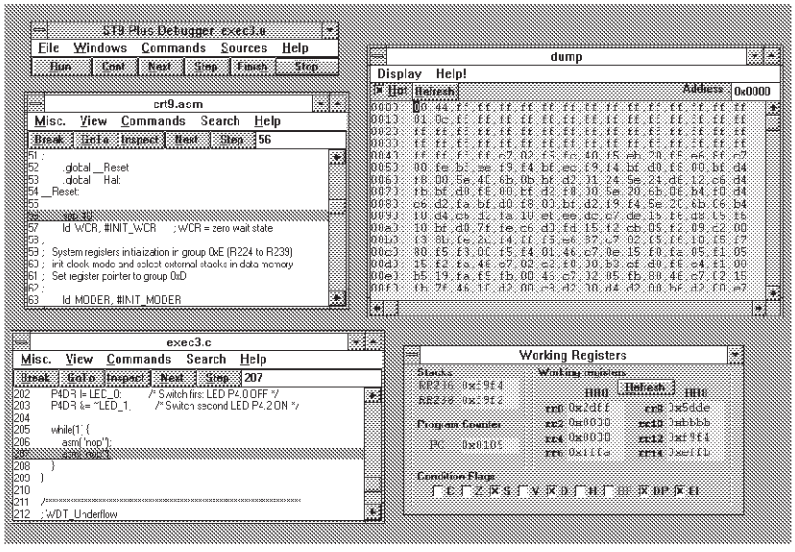
The simulator functionally duplicates the operation of the microcontroller and completely supports the instruction set. It uses its clock frequency with the number of clock cycles needed by each instruction to keep track of the real time execution speed.

1.5 DEBUGGER

The debugger is a powerful tool, helpful when writing, executing and debugging programs for microcontrollers.

The debugger controls the downloading of the output file of the linker in a target machine. Associated with an *emulator*, it allows to monitor your program in the context of the application. For debugging purposes, the debugger provides several commands for displaying and setting memory and registers, for executing programs, setting breakpoints, tracing instructions.

Figure 8. Example of a Windows debugger

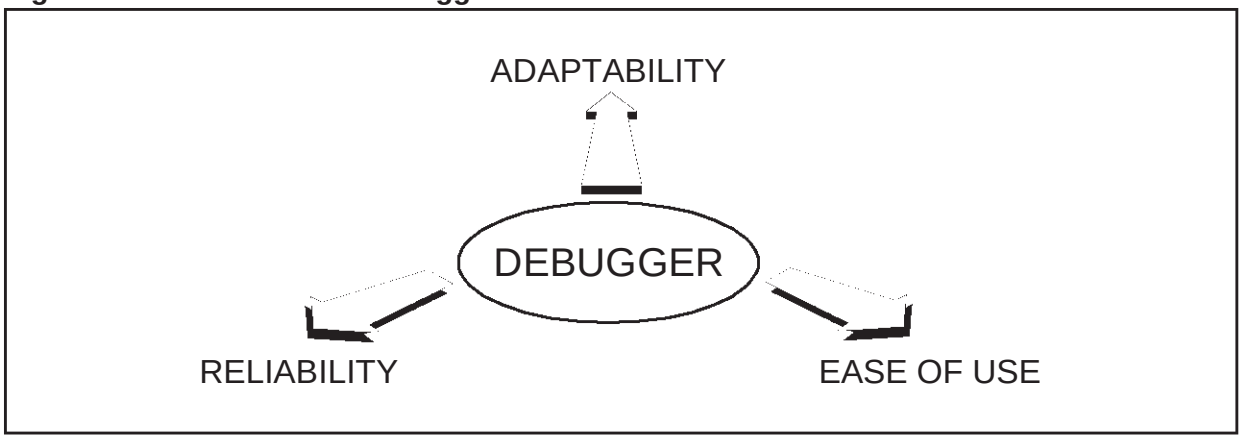


The debugger is typically a multiwindow environment. This type of debugger provides some very useful features. First, it is easy to use using the familiar Windows environment. The program or the microcontroller resources are directly accessible: you can examine resources, access variables or functions, control memory and access registers.

The debugger should be very flexible. It must be adapted to your needs by displaying only information you require for your debugging. You can choose the basic characteristics of each window by selecting filters. In this way, you customize your environment and get only the information you need for a particular problem.

Using source level debugging, it is easy to identify which part of the program is displayed. The source and generated codes are on the same screen. The environment is easy to learn since it makes full use of window controls such as toolbars, tooltips, drag-and-drop, and context-sensitive pop up menus. From these windows, you fully control the execution of your program.

Figure 9. State-of-the-art Debugger Characteristics



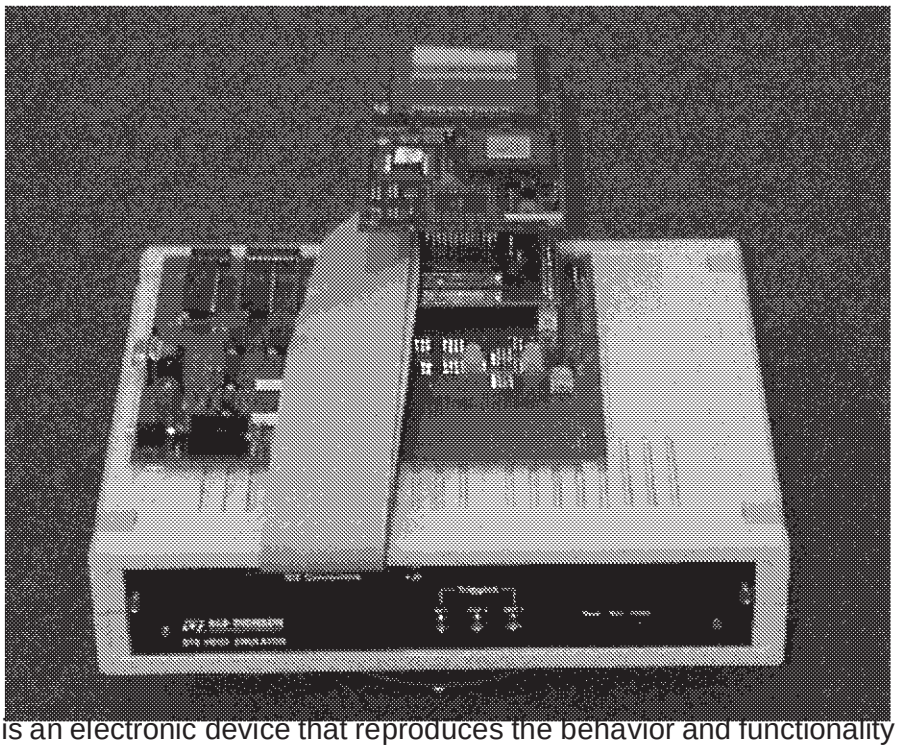
The debugger is an essential part of the development chain. By improving your productivity in the debug phases, a windowed symbolic debugger reduces your development costs and speeds up your time to market.

2 HARDWARE DEVELOPMENT TOOLS

There are three different types of development system that are in common use. Most of these systems require a host computer (PC, station) to run the device. Usually they come with software to interface with the host computer.

2.1 EMULATOR

Figure 10. Emulator with its probe plugged in dedicated board



An emulator is an electronic device that reproduces the behavior and functionality of a specific microcontroller in real time (that is it runs at the full speed of the microcontroller even in debug mode). The emulator is used to replace the component being emulated in your system. It allows interactive debugging of the software and can control your program execution while operating in the system.

The emulator is connected to your application by a probe which is plugged directly into the application and controlled by a powerful software **debugger** which sends and receives information to and from the emulator.

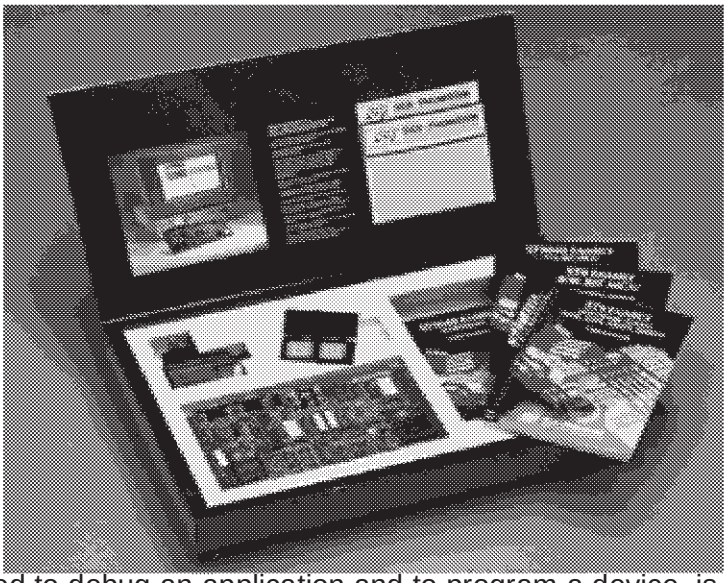
By means of a graphic interface which displays all available information on the PC screen, you can access the microcontroller internal resources (such as registers) and its internal memories. This enables the development of application programs and the system emulation in real time or in single step mode. You can set breakpoints on specific instructions or addresses. The emulator stops the program execution when the CPU reaches that particular instruction. Then you can consult all microcontroller registers and trace the preceding executed instruc-

tions. There is a wide range of debug commands which give you full control of the emulator hardware and several commands for controlling program execution. Memory and registers can be read and written in different formats, while macro commands and conditional block constructs are available for use in automated debugging sessions.

2.2 STARTER KIT OR DEVELOPMENT BOARD

A starter kit is a basic development system for the evaluation and design of microcontroller applications. This evaluation kit can include emulation capabilities and provide a quick introduction to the microcontroller world. The starter kit may include all hardware, software and documentation required to evaluate the microcontroller and to develop simple applications.

Figure 11. Starter kit content



The hardware used to debug an application and to program a device, is an evaluation or development board. It is a board which includes the microcontroller in the version which offers all the most important features of the family to be tested. The microcontroller operates in a special mode that allows internal bus access. Code delivered from the host is put into memory that the microcontroller can access, and the microcontroller can operate as if the code was contained within its internal memory.

For maximum flexibility, the board can run in different modes of operation. Without a host computer, you can prototype an application and build an easy-to-use demonstration board by connecting your own application board to the starter kit. The main mode of operation allows you to debug your target system. Your code can be downloaded into emulation memory and then executed under control of a Windows **debugger**. You have the possibility to program the debugged program into an EPROM or OTP member.

The software includes a *assembler* which supports modular programming, a *linker*, an archiver that manages relocatable objects modules, a functional *simulator*, a Windows **debugger** which drives the evaluation board and the EPROM programming software.

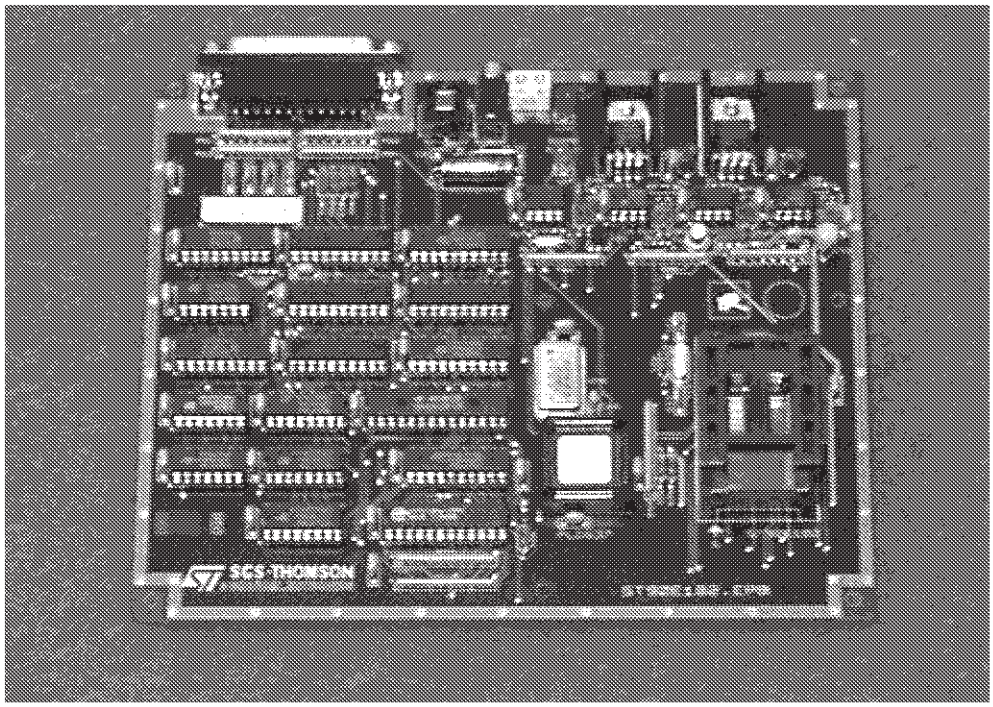
The starter kit also includes full documentation on the microcontroller family, on how to connect and program it and the software manuals which describe how to use the development tools, as well as a diskette containing several application programs for microcontroller devices.

This full evaluation kit provides a low cost platform for developing and evaluating embedded applications.

2.3 EPROM AND GANG PROGRAMMER

The EPROM programmer (or EPROM programming board) is a programming tool for EPROM and OTP members of the microcontroller families.

Figure 12. EPROM programming board



This board is designed to program the EPROM versions of microcontroller, including both the ceramic windowed and plastic OTP packages. Several sockets are provided to receive the different existing packages types.

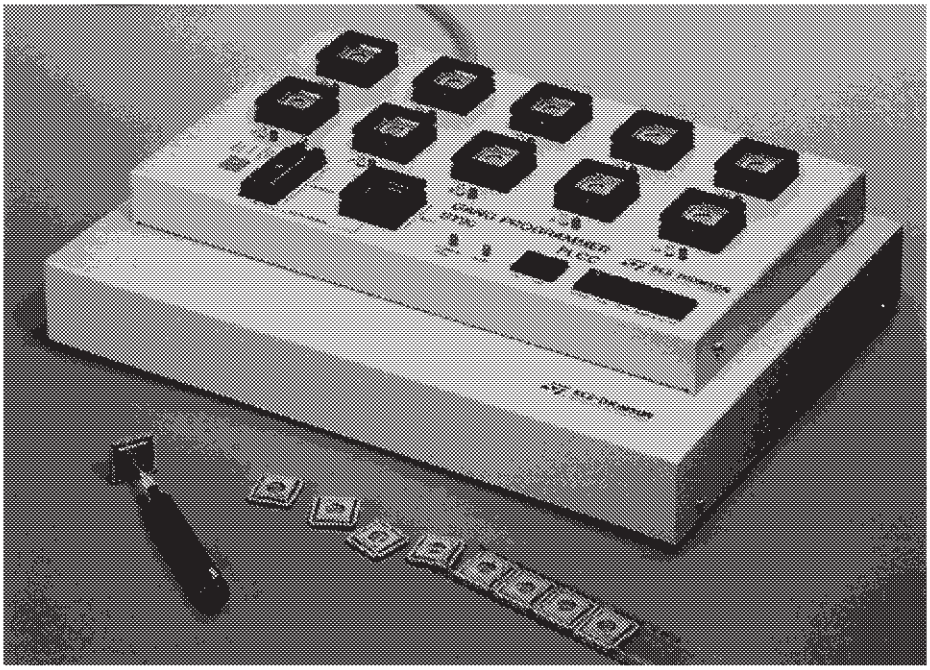
The EPROM programming board uses a RAM in which your code is downloaded. The EPROM device will be programmed from the contents of this RAM.

The board can perform three operations:

- verify the blank state of the microcontroller EPROM;
- program microcontroller with the content of hexadecimal file;
- verify the microcontroller.

The gang programmer is a programmer which has several EPROM programmers put in parallel.

Figure 13. Gang programmer



THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS.