May 1, 2000

Embedded Software – An Overview

To implement an embedded application, a wide range of software choices must be made before implementation is started. Some typical decisions might be:

- Should I use an operating system or not?
- Does my application need real-time performance?
- Do I want my application to execute in a 16-bit or 32-bit run-time environment?
- How will my embedded application be launched at power-up?
- How will I load my application onto the embedded computer?
- How will I manage my application's needs for program and data memory?
- What language will I use for application development?
- How will I debug the embedded software?
- Does my application need a file system?
- Does my application need networking?
- Do I want to use any off the shelf libraries or drivers?
- What bus or busses will I need to access hardware functions across?

The combinations and permutations of the above issues (and dozens of others) truly create thousands of different embedded software strategies. The following discussion is necessarily brief, and may raise additional questions. Application engineers at Micro/sys are always available to discuss your unique requirements and offer suggestions.

1. Operating Systems

If an application requires real-time performance, a Real Time Operating System (RTOS) is most often required. These specialized software system products are designed specifically for embedded systems, and guarantee maximum time delays from stimuli to responses.

In other applications, the decision to use an operating system (OS) can be based on other issues. If a solid state or traditional disk file system is needed, an OS is usually called for. Remember, the 'D' in DOS is for 'disk'.

Also, if there is a third party software driver for a particular hardware function in your embedded system, that driver may only work with a particular OS. Either use that OS, or write your own device driver.

Without one of the three "decision pushers" mentioned above, the decision to use an OS or not is an open choice. On the down side, many operating systems incur a per-unit-shipped royalty payment to the OS vendor. Others may have no royalty, or the embedded computer hardware vendor may hide it in the price of the computer.

Embedded systems are often implemented as self-sufficient code, taking control of the embedded computer from power-up, and handling every aspect of operation. No OS calls are made, because there is no OS. This type of development requires additional issues to be addressed. Additional development tools, such as locators may be needed, which may require a much more detailed knowledge of linker outputs in such areas as segment usage and the need for ROM-to-RAM copies at startup time.

The Micro/sys RUN.EXE system is an easy-to-use hybrid of OS and self-sufficient code that is ideal for many 16-bit applications. It requires no special development tools or equipment.

16-bit vs. 32-bit Run-time Environments

A primary embedded application decision is whether to use a 16-bit ("real mode") or a 32-bit ("protected mode") run-time environment. Obviously, you must have a 32-bit processor - 386 or above - if you want to run your application in a 32-bit environment. A 16-bit application can be run on either processor type. In fact, all 32-bit processors actually power up in 16-bit mode. If you do not specifically enter 32-bit mode, they act as if they are a 16-bit processor.

This creates a historical artifact in terms of processor performance. Because desktop systems all went to 32-bit processors many years ago, improvements in processor silicon have only been applied to 32-bit processors. So while there are 450MHz 32-bit processors, that silicon technology will seldom, if ever, be applied to 16-bit processors. There are also hardware bus structures, such as PCI and USB, that were invented for 32-bit processors, and cannot be used by 16-bit systems. For example, most new 100BASE-T Ethernet chips are PCI-based. Therefore, there are some cases where it is reasonable to run a 32-bit processor in 16-bit mode - just for the faster processor or PCI hardware support.

MSDOS and all of its clone operating systems (referred to hereafter as DOS) are 16-bit run-time environments. Real mode supports 1MB of memory - TOTAL. All CPU addressable memory - program and data, EPROM and flash and RAM, BIOS and video RAM and device option ROMs - must fit in 1MB. To make it tougher, the "segmented" nature of real mode only allows 64KB to be accessed at a time without manipulating CPU segment registers.

In the IBM PC architecture, 384KB of memory is reserved for video RAM, option ROMs, and BIOS code, leaving 640KB of memory for OS and application.

Application programs that will run in 16-bit run-time environments must be created with 16-bit tools - compilers and linkers. If a set of development tools can create programs that will run under DOS, they can create 16-bit applications. Examples are Microsoft C/C++ versions 7.0 and earlier, Microsoft Visual C++ Versions 1.52 and earlier, and Borland C++ versions 5.02 and earlier.

With 386 and higher processors, a full 32-bit mode of operation is available. The 32-bit mode of operation allows 4GB of memory to be directly addressed. It is possible to initialize a 32-bit system into a "linear" mode where there is no longer any segmentation. Today, most desktop operating systems and most RTOS systems operate as 32-bit run-time environments.

Loading and Launching OS and Application

There are many methods to load OS and application programs onto an embedded computer, and to launch them at power-up. Most embedded computers have a BIOS which initializes the computer, then can immediately boot DOS, whether from flash disk, ROM, floppy, or IDE. This is another area that requires consideration when selecting a software strategy.

For extremely simple loading and launching of 16-bit applications, the Micro/sys RUN.EXE system includes a free DOS-compatible OS preinstalled into flash at the factory. You merely XMODEM your application .EXE program onto the board through a serial cable we supply. At power-up, the RUN.EXE firmware launches your application automatically. Other variations can load an MSDOS or a DOS-compatible OS and use AUTOEXEC.BAT to launch your application.

The loading and launching of 32-bit operating systems and applications is much more varied. Windows CE, RTOS packages, Linux, etc. all have unique load and launch requirements. Some emulate a disk boot sector so that a BIOS thinks it is loading DOS, but a special boot sector gets control and loads things differently. Some need ROM-based binary images that must be created on the development PC to be loaded into flash. Then they take control of the embedded computer either with no BIOS in the system, or at some point during or after BIOS startup.

For 32-bit OS and application needs, it is best to find a mix of embedded computer hardware and OS that the computer vendor has experience with, as the loading and launching issues are sometimes hardware specific.

2. Language Selection

C and C++
While a number of computer languages are available, most development of embedded applications has migrated to C, or it's offspring, C++. C offers an efficient, simple language that has capabilities to directly manipulate all forms of hardware interfaces. C and assembly language can be mixed very well. Most modern C compilers even support in-line assembly language sequences.

C++, while based on the C language, adds additional levels of abstraction, and is a much more object oriented language. In some large embedded software systems, the structured framework provided by C++ can offer better management of multi-programmer projects. C++ usually generates larger programs than C.

C and C++ compilers are available from a number of sources, including Microsoft, Borland (Inprise), GNU, Watcom, Metaware, etc. Micro/sys provides complete support for C programming, including compilers, programming books, sample programs, and technical support. We stock CDs for two versions of Borland C++ - version 3.1 is a DOS IDE that generates DOS executables, and version 4.5 is a Windows IDE that generates DOS (and Windows) executables. Version 3.1 generates smaller executables than version 4.5.

Assembly Language
For maximum efficiency, nothing beats assembly language, even though it is quite a bit more complicated than C. For Intel architecture embedded PCs, Borland (Inprise) TASM, and Microsoft MASM are excellent assemblers. Micro/sys provides assembly language support including assemblers, sample programs, and technical support.

BASIC
Some dialects of BASIC can be used in embedded system development, although BASIC is not used nearly as often as C or C++. Because of it's origins as a beginner's language, most BASIC compilers are more tightly integrated with the underlying hardware and operating system than C is. This can require some careful BASIC coding unless both embedded hardware and operating system are 100% desktop PC compatible. For example, many embedded systems do not include a VGA output for user display. Yet the PRINT statement of some compilers blindly copy characters to VGA memory, whether it is there or not. There are workarounds for most of these situations, but a programmer must learn the issues and how to address them.

Microsoft's QuickBASIC and Visual BASIC for DOS are two examples of BASIC compilers for 16-bit DOS (or equivalent) systems. Micro/sys provides very limited support for 16-bit BASIC programming with either of these Microsoft compilers. Some products include examples in BASIC, and general BASIC programming notes and libraries.

Pascal
Borland (Inprise) Turbo Pascal has been used by some as an embedded development language. While applications created with Turbo Pascal may run on Micro/sys products, Micro/sys Tech Support does not support Pascal programming, and there are no Pascal examples supplied with any of our products.

Other Languages
If you have a favorite (or mandated) language, it is important to find what operating systems the language runs on, and which embedded computers can run that operating system. Support from any embedded computer vendor may be quite limited, if available at all. We suggest you check first.

3. Embedded Networking

Ethernet networking has become extremely popular in embedded systems. With the explosion of TCP/IP Ethernet wiring around the world, it makes a lot of sense for new embedded systems to view Ethernet as an important interface to higher level supervisory computers, or to remote users. For 16-bit systems, most OS products do not support TCP/IP networking directly. For this reason, Micro/sys developed the Embedded Netsock system, which is stored in flash with our RUN.EXE firmware. In simplistic terms, Embedded Netsock extends the on-board BIOS to be able to directly manage a very limited subset of TCP/IP networking with no third party products. Therefore, no OS needs to be installed, as a single .EXE file that has included the NETSOCK.H include file can be executed directly by the computer's on-board firmware.

Other solutions for 16-bit systems include networking add-ons from DOS-compatible OS vendors, and the PC/TCP linkable library from NetManage, which requires a full DOS compatible OS as opposed to RUN.EXE. These approaches provide support for many more TCP/IP protocols than Micro/sys' Embedded Netsock.

If you are using a 32-bit OS, it is most advisable to use the TCP/IP network stack that is supplied with that OS. This provides the maximum amount of integration, and therefore the least amount of work. Windows CE, Linux, and all RTOS packages offer excellent networking capabilities.

4. Debugging Embedded Applications

Once the edit, compile, link, and optionally locate, cycle is done, you need to have a debug strategy. At the very lowest level, you can always embed debug printout messages into your source code, and have it tell you what it's up to. At the other end of the spectrum are hardware In Circuit Emulators (ICE) which can allow complete control over the CPU chip socket on a computer board - even if the board hardware has not been debugged yet.

Debug printouts are free, ICE systems can cost tens of thousands of dollars. If you plan on using an ICE, contact your embedded computer supplier to ensure that CPU chip packaging, socketing, and orientation will support the specific ICE you plan on using.

One of the most popular approaches is that of a remote software debugger. With this approach, you connect a desktop development PC (where you do your editing, compiling, and linking) with the target embedded computer through a serial cable or a network connection. A program is downloaded into target RAM, and it can be debugged using breakpoints, source code display, variable examination, single-stepping, etc.

For 16-bit remote debugging, Micro/sys provides, free of charge, remote Turbo Debugger interface programs that can be run on Micro/sys embedded computers. This allows remote debugging with the Borland Turbo Debugger product. Because there are many versions of Turbo Debugger, we supply many versions of the interface program. We also stock full release CDs for two Borland C++ versions, each with the proper Turbo Debugger included. You must match compiler, linker, and debugger versions in order to use remote debugging. Serial connection at 115KB is supported.

Each 32-bit operating system has its own debug strategy. Some require a disk-based version of the OS on the target system, so that a "native" debugger can be used. Others, such as Linux with its GDB debugger, include remote debug capabilities. With a 32-bit OS, we advise you to discuss debug strategy with the OS vendor, and with the embedded computer hardware vendor, to ensure that there are no architectural incompatibilities.

5. 16-bit Run-time Environments

Micro/sys embedded PCs support a number of run-time environments. For 16-bit diskless systems, our innovative RUN.EXE™ run-time firmware is often the leading choice. It provides automatic application load on power-up, and includes an industrial BIOS and DOS emulation. It is small and fast, and there are no royalties for incorporating RUN.EXE into OEM products.

RUN.EXE

Written entirely in assembly language, Micro/sys' RUN.EXE firmware is a small, fast 16-bit firmware system that includes an industrial BIOS and a DOS emulator. RUN.EXE is placed in flash on Micro/sys embedded PC products. Upon power-up, RUN.EXE creates a software execution environment that makes an application program think it is running under DOS. All BIOS and DOS function calls, except for disk accesses on some models, are serviced by RUN.EXE firmware.

The built-in "Implied AUTOEXEC.BAT" feature of RUN.EXE takes over each time power is applied to the embedded PC. First, the firmware checks to see if an application program's .EXE file has been loaded into the computer's user flash area. If so, the application program is immediately loaded and run. The RUN.EXE startup routines are fast and tight, so your program gets control quickly after power-up in order to allow early initialization of critical hardware.

To perform configuration and setup of an embedded PC, RUN.EXE includes a Flash Setup™ utility, also preloaded into flash. By attaching a special cable (supplied by Micro/sys at no charge) to the COM2 connector, and resetting the embedded PC, a terminal emulator on a laptop or desktop, such as PROCOMM, can be used to set embedded PC operating modes and parameters, format solid state disks, and download application programs. For example, OS operating mode, disk drive parameters, and even network IP address can be entered into setup screens. Because these parameters are stored in on-board flash instead of battery-backed CMOS RAM, there is no possibility of losing configuration data because of a dead battery.

RUN.EXE has a number of advantages over other methods of 16-bit embedded system run-time execution. Traditional embedded tools require complex locator utilities to separate code and data segments. Run-time startup code, initialized data, and self-modifying code are significant challenges with this approach, requiring intimate knowledge of all most library internals. RUN.EXE includes a relocating loader, much like COMMAND.COM, that executes at startup.

MSDOS and DOS-equivalent operating systems are larger than RUN.EXE, which increases memory costs. They often require royalty payments for OEM use. RUN.EXE has no royalty charges - a free copy is shipped preinstalled on embedded PCs.

Note that RUN.EXE will operate on advanced 32-bit processors - even Pentiums. However, the environment presented to the programmer is a 16-bit, 1MB memory environment that requires 16-bit development tools. However, this is a very cost effective strategy if your program is small and you just want the speed of the 32-bit hardware, not the memory addressing capacity it offers.

MSDOS 5.0

MSDOS 5.0 is widely used in disk-based embedded systems. It can be loaded from traditional disk drives, or from solid-state disks. It provides compatibility with other systems through removable media, such as 3 1/2" floppy disks, which have an ideal form factor for embedded systems.

MSDOS 5.0 retains compatibility with previous versions of DOS. It has memory management that allows most of MSDOS to be loaded into extended memory on computers with 386 or higher processors. Also, device drivers and TSRs can be loaded above 640K on computers with 386 or higher processors. It is smaller in size and less expensive than MSDOS 6.22, yet has all of the features needed for embedded systems.

Micro/sys can supply MSDOS 5.0 preinstalled in flash on various embedded PC models.

DOS-compatible Operating Systems

There are a number of MSDOS compatible operating systems, some targeted directly to embedded PCs. Micro/sys Technical Sales may have some information on these, although we do not preinstall them on our embedded PCs. Please feel free to discuss any aspects of your 16-bit OS needs with our engineers.

6. 32-bit Run-time Environments

For systems with more than 1MB of memory, a 32-bit operating system provides the path to higher performance systems. Although 386 and above processors are all 32-bit processors, all Intel processors start out in 16-bit "real mode", with the 1MB memory limit. Each 32-bit operating system includes its own startup system, whereby the processor is switched into 32-bit "protected mode", and a 32-bit application is launched.

Versions of Windows®:

Microsoft offers a number of operating systems based on 32-bit protected mode operation. Windows 95 and 98 are traditional desktop operating systems, and are most often embedded with traditional IDE disk drives, VGA displays, and AT keyboards. Use of these operating systems is truly embedding a desktop computer into a system, and shrink wrap software can be used.

32-bit Real Time Operating Systems (RTOS):

A number of companies offer RTOS products that offer excellent reliability and performance on 32-bit embedded computers. Most offer true real time performance, making them the best choice in systems requiring that. Each has its own strengths and features, and each has its own startup sequence.

To port an RTOS to an embedded computer, you need to customize the load system, adding any board startup code needed for CPU, chipsets, and core peripherals. The vendor's startup routines, both real mode and protected mode, must often be customized also. In addition, drivers may need to be written for any custom hardware devices. Porting an RTOS to a particular embedded computer may involve a moderate amount of effort.

Each RTOS will have a preferred set of application development tools. Some include their own set or subset, others use standard tools like Visual C++. There are different debug strategies, also.

The VxWorks RTOS from Wind River Systems is one of the most popular RTOS products. The PharLap ETS system offers an affordable product that uses standard development tools. A number of RTOS products are offered by other companies.

Each RTOS has its own pricing structure, and many require payment of a per unit shipped royalty. Contact each RTOS vendor for details of their policies.

Micro/sys has assisted a number of customers in porting these RTOS systems to Micro/sys embedded computers. Please inquire about experience with a particular set of products.

7. Software Development Tools for Embedded PCs

Micro/sys supports a number of popular language tools for use during the development of systems incorporating Micro/sys embedded PCs

These languages have been developed with PC programming in mind. The 16-bit compilers have run-time calls for DOS and BIOS resources. With a few minor exceptions, these languages will execute on Micro/sys embedded PCs under the RUN.EXE™ firmware. This low cost, royalty-free 16-bit run-time environment allows the .EXE files generated by these languages to be loaded directly onto a Micro/sys computer board for automatic execution upon power-up.

Tools for Generating 16-bit Applications

Borland C++ Versions 2.1 through 5.0

Borland C++ offers a complete development system for developing 16-bit applications. Compiler, linker, and libraries are included, in addition to the Turbo Debugger system that allows source level remote debugging across a fast serial line. The availability of this debug capability makes the Borland tools the preferred 16-bit development tools.

Versions 2.1 and 3.1 generate small executables, and their Integrated Development Environment (IDE) is DOS character-based. Versions 4.5 and 5.0 generate slightly larger executables, but offer a multi-window graphical IDE for editing, visual makefiles, and support for in-line assembler statements. The 16-bit output of all versions can be used on Micro/sys embedded PCs under RUN.EXE or MSDOS with excellent results.

Micro/sys stocks book/CD packages for Borland C++ 3.1 and 4.5. For a very low cost, we can supply an entire 16-bit development toolset.

Microsoft C/C++ Versions 5.1, 6.0, and 7.0

Microsoft C/C++ compilers, from version 5.1 through 7.0 can be used to generate 16-bit .EXE files for execution under 16-bit environments such as RUN.EXE or DOS. Early versions are DOS command line driven, while later versions execute in DOS boxes under Windows. Note that these compilers are no longer supported by Microsoft, but they are installed in thousands of locations. Microsoft does not support remote debugging, and this is a disincentive for their use in embedded applications.

Microsoft Visual C++ Version 1.52

Visual C++ Version 1.52 presents a full Windows graphical IDE, offering visual makefiles, project trees, and multi-window editing. This was the last version of Visual C++ that generated 16-bit executables, and it is no longer supported by Microsoft. It can be used for creating 16-bit applications, but suffers from the same lack of a remote debugger as other Microsoft C/C++ compilers. Once again, it is installed in thousands of locations.

Microsoft QuickBASIC & Visual BASIC for DOS

Micro/sys offers limited support for the Microsoft QuickBASIC and Visual BASIC for DOS compilers. These development tools let you edit and compile 16-bit BASIC programs. They have some idiosyncrasies that must be addressed on low end embedded PCs. On high end embedded PCs with video, keyboard, and disks, they can be run natively on the embedded PC, offering reasonable debug capabilities. There are no remote debug capabilities.

Borland TASM Assembler

Coding an application in assembly language is the way to create the tightest, fastest code. The Borland TASM assembler is an excellent package. It can be used stand alone, or in conjunction with Borland C++. If you are including TASM-generated OBJ files in a Borland C++ application, you must use the linker and Turbo Debugger supplied with the C++ system. Since Borland C++ supports in-line assembly statements, use of TASM may only be needed in special cases.

Microsoft MASM Macro Assembler

MASM supports many high-level language constructs and language extensions. There are no remote debug capabilities. It is still supported by Microsoft.

Turbo Pascal

Micro/sys RUN.EXE firmware does not support Turbo Pascal. Turbo Pascal programs may execute on Micro/sys embedded PCs with other operating systems. We do not provide technical support for Turbo Pascal.

Tools for Generating 32-bit Applications

In most cases, the 32-bit operating system to be used will have a preferred set of development tools for use in developing applications.

For instance, the Linux OS includes the GNU set of compilers and linkers, and the OS includes dynamically linked C run-time libraries. Other operating systems may suggest Microsoft Visual C++, Borland C++, or other specific compilers for application development.

Check with your 32-bit OS vendor for development tool suggestions.

8. On-board UDP Subset of TCP/IP: Embedded Netsock™

The innovative Embedded Netsock system from Micro/sys provides easy to use network capabilities for applications written as 16-bit C programs. Embedded Netsock firmware is factory-programmed into flash, and dynamically linked into an application program at run-time. Merely including the header file NETSOCK.H into a source program provides full network access.

Embedded Netsock presents a subset of the Winsock standard API, which is itself based on Berkeley sockets. The underlying transport layer is based on the UDP protocol that sends unacknowledged datagrams. This presents SOCK_DGRAM socket capabilities to the programmer. SOCK_RAW sockets are also supported to allow implementation of other protocols. The IP network layer provides standard internet-type IP addressing (i.e. 192.168.1.39). A driver for the on-board Ethernet adapter completes the stack.

IP address and subnet mask can be manually entered, or dynamically set through the DHCP protocol. In addition, Embedded Netsock sends and receives ICMP ping packets, providing low level network troubleshooting.

Embedded Netsock solves the problem of sourcing a small, simple network stack for 16-bit development. Few viable alternatives exist in the 16-bit world. Embedded Netsock programs run under Micro/sys' RUN.EXE operating system, and under MSDOS and compatible operating systems.

Embedded Netsock cannot be linked into 32-bit programs. If a network stack is needed for 32-bit applications, it is advisable to use the networking capabilities of the 32-bit operating system being used.

Embedded Netsock is not sold as a standalone product. It is available only preinstalled on the Netsock series of Micro/sys embedded computers.

Additional Embedded Netsock features and protocols may be available. Please call Micro/sys Technical Sales Department with any questions.