Programming Languages and Tools

Embedded systems are usually programmed in high- level language that is compiled (and/or assembled) into an executable machine code. This machine code is loaded into the ROM and is called firmware, microcode or microkernel. Embedded systems are among the most difficult computer platforms for programmers to work with. Most embedded systems lack a monitor or analogous output device. And those that do have one typically require a special piece of embedded software, called a display driver, to be implemented first.

A common denominator across all embedded-software domains is the use of programming languages that allow direct access to interfaces, memory, and so forth. More than 80 percent of all companies are using C and to some degree C++. More than 40 percent are using assembler for lower-level interfaces. Java is increasingly used for GUI and application programming. Eclipse-based development tools dominate engineering workbenches due to the many different tools that need to be federated, such as modeling and simulation tools (for example, Matlab/Simulink, Rose, and Tau), testing environments(LabView, CANoe, HIL/SIL, and emulators), product life-cycle management environments(TeamcenterandeASEE), configuration management tools (Subversion and CVS), requirements tools (DOORS and Caliber), and of course compilers, debuggers, and the like. Due to the intensive supplier interaction and collaboration, which is much higher than in traditional IT, tools such as DOORS or Matlab/Simulink have respective market shares of more than 50 percent.

Debugging

Even the most perfectly designed and implemented program will probably have some bugs. You notice a bug by conducting a test and detecting a divergence between the expected behavior and the observed one. Naturally, you want to find out what went wrong and fix it. Even though elaborate debugging tools are nowadays available for all popular architectures, you may still occasionally be forced to work on a system that has no such support. However, as long as the system has some output mechanisms, not all is lost. One of the most primitive debugging techniques available is the use of an LED as indicator of success or failure. The basic idea is to slowly walk the LED enable code through the larger program. In other words, you first begin with the LED enable code at the reset address. If the LED turns on, then you can edit the program, moving the LED enable code to just after the next execution milestone, rebuild, and test. This works best for very simple, linearly executed programs like the startup code.

The processor/microcontroller fetches and executes the instructions at a high rate of speed and provides no way for you to view the internal state of the

program. This might be fine once you know that your software works and you're ready to deploy the system, but it's not very helpful during software development. Of course, you can still examine the state of the LEDs and other externally visible hardware but this will never provide as much information and feedback as a debugger.

Remote Debuggers

If available, a remote debugger can be used to download, execute, and debug embedded software over a serial port or network connection between the host and target. The front end of a remote debugger looks just like any other debugger that you might have used. It usually has a text or GUI-based main window and several smaller windows for the source code, register contents, and other relevant information about the executing program. However, in the case of embedded systems, the debugger and the software being debugged are executing on two different computer systems.

A remote debugger actually consists of two pieces of software. The front end runs on the host computer. But there is also a hidden backend that runs on the target processor and communicates with the frontend over a communications link of some sort. The backend provides for low-level control of the target processor and is usually called the debug monitor.The debug monitor resides in ROM -having been placed there in the manner described earlier (either by you or at the factory) – and is automatically started whenever the target processor is reset. It monitors the communications link to the host computer and responds to requests from the remote debugger running there. Of course, these requests and the monitor's responses must conform to some predefined communications protocol and are typically of' a very low- level nature. Examples of requests the remote debugger can make are "read register x," "modify register y," "read n bytes of memory starting at address," and "modify the data at address." The remote debugger combines sequences of these low-level commands to accomplish high- level debugging tasks like downloading a program, single-stepping through it, and setting breakpoints.

Emulators

Remote debuggers are helpful for monitoring and controlling the state of embedded software, but only an in-circuit emulator (ICE) allows you to examine the state of the processor on which that program is running. In fact, an ICE actually takes the place of - or emulates - the processor on your target board. It is itself an embedded system, with its own copy of the target processor, RAM, ROM, and its own embedded software. As a result, in-circuit emulators are usually pretty expensive often more expensive than the target hardware. But they are powerful tools, and in a tight debugging spot nothing else will help you get the job done better.

Simulators and other tools

Many other debugging tools are available to you, including simulators, logic
analyzers, and oscilloscopes. A simulator is a completely host-based program, that simulates the functionality and instruction set of the target processor. Since it is a lot more comfortable to develop on the PC than it is to work on the target hardware, instruction set simulators (ISS) were developed to allow the execution of target software on the host PC. The ISS accurately simulates the target controller down to the number of clock cycles required to execute different instructions.The simulator provides all the debug features typically found in modern debuggers, ranging from single-stepping to memory manipulation. It also allows the user to watch processor-internal registers to better help track problems.

A logic analyzer is a piece of laboratory equipment that is designed specifically for troubleshooting digital hardware. It can have dozens or even hundreds of inputs, each capable of detecting only one thing: whether the electrical signal attached to it is currently at logic level l or 0. Most logic analyzers will also let you begin capturing data, or "trigger," on a particular pattern. For example, you might make this request: "Display the values of input signals 1through 10, but don't start recording what happens until inputs 2 and 5 are both zero at the same time."

An oscilloscope is another piece of laboratory equipment for hardware debugging. But this one is used to examine any electrical signal, analog or digital, on any piece of hardware. Oscilloscopes are sometimes useful for quickly observing the voltage on a particular pin or in the absence of a logic analyzer, for something slightly more complex. However, the number of inputs is much smaller (there are usually about four) and advanced triggering logic is not often available. As a result, it'll be useful to you only rarely as a software debugging tool.