10.1 Introduction

Embedded systems are a blend of hardware and software. Each must complement the other. Hardware people can make the firmware easier to implement.

Many of the suggestions in this chapter will make a system easier to debug. Remember that a good design works; a great design is also one that’s easy to debug.

10.2 Diagnostics

In the nonembedded world, a favorite debugging trick is to seed print statements into the code. These tell the programmer whether the execution stream ever got to the point of the print. But firmware people rarely have this option. So, add a handful of unassigned parallel I/O bits. The firmware people desperately need these as a cheap way to instrument their code. Seeding I/O instructions into the code that drives these outputs is a simple and fast way to see what the program is doing.

Developers can assert a bit when entering a routine or ISR, then drive it low when exiting. A scope or logic analyzer then immediately shows the code snippet’s execution time.

Another trick is to cycle an output bit high when the system is busy and low when idle. Connect a voltmeter to the pin, one of the old-fashioned units with an analog needle. The meter will integrate the binary pulse stream, so the displayed voltage will be proportional to system loading.

If space and costs permit, include an entire 8-bit register connected to a row of 0.1 inch spaced vias or headers. Software state machines can output their current “state” to this port. A logic analyzer captures the data and shows all the sequencing, with nearly zero impact on the code’s execution time.

At least one LED is needed to signal the developer—and perhaps even customers—that the system is alive and working. It’s a confidence indicator driven by a low-priority task or idle loop, which shows the system is alive and not stuck somewhere in an infinite loop. A lot of embedded systems have no user interface; a blinking LED can be a simple “system OK” indication.
Highly integrated CPUs now offer a lot of on-chip peripherals, sometimes more than we need in a particular system. If there’s an extra UART, connect the pins to an RS-232 level shifting chip (e.g., MAX232A or similar). There’s no need to actually load the chip onto the board except for prototyping. The firmware developers may find themselves in a corner where their tools just aren’t adequate and will then want to add a software monitor (see www.simtel.com) to the code. The RS-232 port makes this possible and easy.

If PCB real estate is so limited that there’s no room for the level shifter, then at least bring Tx, Rx, and ground to accessible vias so it’s possible to suspend a MAX232 on green wires above the circuit board.

**Note:** Attention, developers: if you do use this port, don’t be in such a panic to implement the monitor that you implement the RS-232 drivers with polled I/O. Take the time to create decent interrupt-driven code. In my experience, polled I/O on a monitor leads to missed characters, an unreliable tool, and massive frustration.

Bring the reset line to a switch or jumper, so engineers can assert the signal independently of the normal power-up reset. Power-up problems can sometimes be isolated by connecting reset to a pulse generator, creating a repeatable scenario that’s easy to study with an oscilloscope.

### 10.3 Connecting Tools

Orient the CPU chip so that it’s possible to connect an emulator, if you’re using one. Sometimes the target board is so buried inside of a cabinet that access is limited at best. Most emulator pods have form factors that favor a particular direction of insertion.

Watch out for vertical clearance, too! A pod stacked atop a large SMT adaptor might need 4 to 6 inches of space above the board. Be sure there’s nothing over the top of the board that will interfere with the pod.

Don’t use a “clip-on” adaptor on a SMT package. They are simply not reliable (the one exception is PLCC packages, which have a large lead pitch). A butterfly waving its wings in Brazil creates enough air movement to topple the thing over. Better, remove the CPU and install a soldered-down adaptor. The PCB will be a prototype forever, but at least it will be a reliable prototype.

Leave margin in the system’s timing. If every nanosecond is accounted for, no emulator will work reliably. An extra 5 nsec or so in the read and write cycle—and especially in wait state circuits—does not impact most designs.

If your processor has a BDM or JTAG debug port, be sure to add the appropriate connector on the PCB. Even if you’re planning to use a full-blown emulator or some other development tool, at least add PCB pads and wiring for the BDM connector. The connector’s cost approaches zero and may save a project suffering from tool woes.
A logic analyzer is a fantastic debugging tool yet is always a source of tremendous frustration. By the time you’ve finished connecting 100 clip leads, the first 50 have popped off. There’s a better solution: Surround your CPU with AMP’s Mictor connectors. These are high-density, controlled impedance parts that can propagate the system’s address, data, and control buses off-board. Both Tektronix and Agilent support the Mictor. Both companies sell cables that lead directly from the logic analyzer to a Mictor. No clip leads, no need to make custom cables, and a guaranteed reliable connection in just seconds. Remove the connectors from production versions of the board or just leave the PCB pads without loading the parts.

Some signals are especially prone to distortion when we connect tools. Address latch enable (ALE), also known as address strobe (AS) on Motorola parts, distinguishes address from data on multiplexed buses. The tiniest bit of noise induced from an emulator or even a probe on this signal will cause the system to crash. Ditto for any edge-triggered interrupt input (like NMI on many CPUs). Terminate these signals with a twin-resistor network. Though your design may be perfect without the terminations, connecting tools and probing signals may corrupt the signals.

Add test points! Unless its ground connection is very short, a scope cannot accurately display the high-speed signals endemic to our modern designs. In the good old days it was easy to solder a bit of wire to a logic device’s pins to create an instant ground connection. With SMT this is either difficult or impossible, so distribute plenty of accessible ground points around the board.

Other signals we’ll probe a lot and that must be accessible include clock, read, write, and all interrupt inputs. Make sure these each have either test points or a via of sufficient size that a developer can solder a wire (usually a resistor lead) to the signal.

Do add a Vcc test point. Logic probes are old but still very useful tools. Most need a power connection.

10.4 Other Thoughts

Make all output ports readable. This is especially true for control registers in ASICs because there’s no way to probe these.

Be careful with bit ordering. If reading from an A/D, for instance, a bad design that flips bit 7 to input bit 0, 6 to 1, etc. is a nightmare. Sure, the firmware folks can write code to fix the mixup, but most processors aren’t good at this. The code will be slow and ugly.

Use many narrow I/O ports rather than a few wide ones. When a single port controls three LEDs, two interrupt masks, and a stepper motor, changing any output means managing every output. The code becomes a convoluted mess of ANDs/ORs. Any small hardware change requires a lot of software tuning. Wide ports do minimize part counts when implemented using discrete logic, but inside a PLD or FPGA there’s no cost advantage.
Avoid tying unused digital inputs directly to Vcc. In the olden days this practice was verboten, since 74LS inputs were more susceptible to transients than the Vcc pin. All unused inputs went to Vcc via resistor pull-ups. That’s no longer needed with logic devices, but it is still a good practice. It’s much easier to probe and change a node that’s not hardwired to power.

However, if you must connect power directly to these unused inputs, be very careful with the PCB layout. Don’t run power through a pin; that is, don’t use the pin as a convenient way to get the supply to the other pins or to the other side of the board. It’s much better to carefully run all power and ground connections to input signals as tracks on the PCB’s outside layers, so they are visible when the IC is soldered in place. Then developers can easily cut the tracks with an X-Acto knife and make changes.

Pull-up resistors bring their own challenges. Many debugging tools have their own pull-ups, which can bias nodes oddly. It’s best to use lower values rather than the high ones permitted by CMOS (say 10 k instead of 100 k).

PCB silkscreens are oft-neglected debugging aids. Label switches and jumpers. Always denote pin 1 because there’s no standard pin 1 position in the SMT world. And add tick-marks every 5 or 10 pins around big SMT packages, and indicate whether pin numbers increase in a CW or CCW direction. Otherwise, finding pin 139 is a nightmare, especially for bifocal-wearing developers suffering from caffeine-induced tremors.

Key connectors so that there’s no guessing about which way the cable is supposed to go.

Please add comments to your schematic diagrams! For all off-page routes, indicate the page the route goes to. Don’t hide the pin numbers associated with power and ground—explicitly label these.

When the design is complete, check every input to every device and make absolutely sure that each is connected to something—even if it’s not used. I have seen hundreds of systems fail in the field because an unused input drifted to an asserted state. You may expect the software folks to mask these off in the code, but that’s not always possible, and even when it is, it’s often forgotten.

Try to avoid hardware state machines. They’re hard to debug and are often quite closely coupled to the firmware, making that, too, debug-unfriendly. It’s easier to implement these completely in the code. Tools (e.g., VisualState from IAR) can automatically generate the state machine code.

**10.5 Construction Methods**

Embedded controllers can be constructed using any one of several techniques, but the most common method is a printed circuit board (PCB). The PCB is constructed of insulating material, such as epoxy impregnated glass cloth, laminated with a thin sheet of copper. Multiple
layers of copper and insulating material can be laminated into a multilayer PCB. By drilling and plating holes in the material, it is possible to interconnect the layers and provide mounting locations for through-hole components.

In designing the layout, or interconnecting pattern of the PCB, there are many conflicting requirements that must be addressed to make a reliable, cost-effective, and producible device. For low-speed circuits, the parasitic effects can be ignored and are often assumed to be ideal connections. Unfortunately, real circuits are not ideal, and the wires and insulating material have an effect on the circuit, especially for signals with fast signal rise/fall times. The traces, or wires, on the PCB have stray resistance, capacitance, and inductance. At high speeds, these stray effects delay and distort the signals. Special care must be taken when designing a PC board to avoid problems with transmission line effects, noise, and unwanted electromagnetic emissions.

10.5.1 Power and Ground Planes

When possible, it is a good idea to use two layers of a four-or-more-layer PCB dedicated to the Vcc and ground signals. These are referred to as power and ground planes. One advantage is that there is a beneficial high-frequency parasitic power supply decoupling capacitance, which reduces the power supply noise to the ICs. Power planes also reduce the undesirable emission of electromagnetic radiation that can cause interference and reduce the circuit’s susceptibility to externally induced noise. The power planes tend to act as a shield to reduce the susceptibility to external noise and radiation of noise from the system.

10.5.2 Ground Problems

Although the concept of an ideal circuit ground may seem relatively simple, a great many system problems can be directly traced to ground problems in actual applications. At the least, this can cause undesirable noise or erroneous operation; at the worst, it can result in safety problems, including possibly even death by electrocution. Lest you dismiss the importance of this possibility too quickly, the author has narrowly missed electrocution while testing a device in which the grounding was improperly implemented!

These problems are most often caused by one of the following problems:

- Excessive inductance or resistance in the ground circuit, resulting in “ground loops”
- Lack of or insufficient isolation between the different grounds in a system: earth, safety, digital, and analog grounds
- Nonideal grounding paths, resulting in the currents flowing in one circuit inducing a voltage in another circuit

The solutions to these problems vary, depending on the type of problem and the frequency range in which they occur. Usually they can be simplified to reducing the currents flowing
in common impedances of circuits that need to remain isolated using a single point ground
and the prudent application of shields and insulation to prevent unwanted parasitic signal
coupling.

10.6 Electromagnetic Compatibility

Electromagnetic compatibility (EMC) issues have become much more significant now that
there are a large number of electronic devices which unintentionally radiate electromagnetic
energy in the same frequency ranges used for communication, navigation, and instrumenta-
tion. Regulatory agencies—such as the Federal Communications Commission (FCC) in the
United States, the Department of Communications (DOC) in Canada, and similar organiza-
tions in Europe—have defined limits to the amount of energy such electronic devices are
allowed to emit at various frequencies. Even more stringent requirements are placed on
life-critical equipment, such as aircraft navigation and life support equipment, because of
the sensitive nature of the applications. Among other things, these devices are required to
provide a minimum level of immunity to externally induced noise (radiated and conducted
susceptibility).

In solving an EMC problem, the first step is to identify the source of the noise, the path to
the problem area, and the destination at which the problem manifests itself. Once these three
characteristics of an EMC problem are identified, the engineer can evaluate the relative merits
of eliminating the noise at its source, breaking the path using shielding and similar techniques,
and reducing the sensitivity of the affected circuit. There are several useful resources, includ-
ing publications, seminars, test labs, and consultants who specialize in solving EMC prob-
lems. The best solution is usually to begin testing a new design at the earliest possible point
in the prototype phase to determine the potential problem areas so that they can be addressed
with the least cost and schedule impact.

10.7 Electrostatic Discharge Effects

Electrostatic discharge (ESD) is an important design consideration in embedded applications
because of the potential for failure and erroneous operation in the presence of external electric
fields. ESD voltages are commonly impressed on embedded interfaces—on the order of tens
of thousands of volts—when someone walks across a floor in a low-humidity environment
before touching an electronic device. One of the most common places where this becomes an
issue is in the keyboard or user input device, which comes in direct contact with the outside
world. This effect can cause immediate damage or upset or may cause latent failures that show
up months after the ESD event. Designers most often use shielding and grounding techniques
similar to those used for safety and emission-reduction techniques to minimize the effects of
ESD. The same resources that are available for EMC problems are also generally of use for ESD problems.

10.7.1 Fault Tolerance

Increasingly, fault tolerance has become a requirement in embedded systems as they find their way into applications where failure is simply unacceptable. Many hardware and software solutions have been developed to address this need.

To understand how to deal with these faults, we must first identify and understand the types and nature of each type of fault. Every fault can be categorized as a “hard” or a “soft” fault. Hard faults cause an error that does not go away—for example, pushing reset or powering down does not result in recovery from the fault condition. Soft faults are due to transient events or, in some cases, program errors.

Self-test and diagnostic programs may be able to identify and diagnose the failure if it is not too severe. Depending on the type of fault that occurs and which device(s) are affected, it may be possible to design a system to detect the fault, possibly even isolating the location of the fault to some degree. In the event of a soft failure, it may be possible for the designer to make the system recover from the fault automatically.

A built-in self-test program can be written for an embedded processor that will be able to detect faults in the following types of devices:

- Processor (if the fault is not too severe)
- Memory
- ROM
- RAM
- E/EEPROM
- Peripheral devices

Note that it is difficult, if not impossible, to detect faults in the control circuits or “glue logic” in a system. Other devices, such as memories, lend themselves to diagnostic methods.

The data contents of ROM devices can be tested for errors using one or more of the following techniques:

- Parity
- Checksum
- Cyclic redundancy check (CRC)
RAM memories and the integrity of information stored in RAM by the processor can be tested for proper operation using one of the following techniques:

- Hardware error detection and correction
- Data/address pattern tests
- Data structure integrity by checking stack limits and address range validity

Additionally, the integrity of the program and proper execution sequence by the CPU can be checked using one or more of the following techniques:

- Hardware parity error detection
- Duplicate, redundant hardware and cross checking or voting
- “Watch dog” timer that operates the CPU chip’s reset line
- Diagnostics that run constantly, when the CPU has nothing else to do

10.8 Hardware Development Tools

There are two general classes of hardware development tools available to the embedded developer: **passive** analysis tools which allow looking at the operation of the system, and **active** tools which allow the designer to intrude on the operation of the system while it’s running (even making changes to the system’s configuration and software while it is under test). The system under test is usually referred to as the “target” system, and the computer that is used to develop, edit, compile, assemble, and download the code to the target system is called the “host” system.

Passive tools include:

- Logic probes to look at static logic levels and detect pulses
- Oscilloscopes to look at signal waveforms
- Logic analyzers, with processor specific probes
- Software to assist hardware development, scope loops

Active tools include:

- In-circuit emulators (ICE) for HW/SW integration are plugged into the application circuit (the “target” system) in place of the CPU, allowing the designer to “see inside” the microcontroller, download, and execute programs selectively.
- ROM emulators (ROM ICE) allow the designer to reduce the time it takes to edit-compile-load-debug programs by replacing the program EPROM with a RAM that can be loaded quickly and easily from the host computer.
10.8.1 Instrumentation Issues

One of the most significant, but often ignored, problems designers must address is the proper selection and use of test instrumentation. Improper selection and application of these tools are frequently the source of much wasted time and confusion for the designer. Two common usage problems relate to the use of oscilloscope and logic analyzer probes.

A typical scope or logic analyzer is supplied with probes that might not be expected to have an effect on the observed signal or distort the data gathered. With input impedances in the meg-ohm range and parasitic capacitances of tens of picofarads, it might seem that the test equipment would have little or no effect on the measurement, but this is definitely not the case.

There are two common causes for measurement problems: excessive ground lead inductance and excessive capacitive loading. These things cause at least a potential for erroneous measurements or, at worst, they can cause the circuit under test to behave differently. Two things can be done to mitigate these problems:

1. Use the shortest possible test leads, especially for the ground connection on fast logic.
2. Use high-impedance probes, especially designed for high-speed applications, such as high-speed FET input scope probes.

Other instrumentation problems can be caused by misinterpretation of the sampling effects in digital scopes, the lack of glitch detection in logic analyzers, and other obscure but potentially painful "learning experiences." These can only be avoided with a good understanding of the operation of the equipment in use and some practical experience.

10.9 Software Development Tools

Most of the software development tools available to the embedded system designer fall into one of three categories: language translators, debuggers, and utility programs that generally run on the host computer. Most of the available tools have been designed to run on the x86 architecture PC, and many are available as freeware, shareware, or low-cost commercial products for the more common target processor architecture. Translators include:

- Assembler
- Compiler
- Linker
- Interpreter

Debuggers include:

- Software/firmware monitors
• Processor In-Circuit Emulator (ICE)
• ROM ICE

Utilities include:
• PROM Programming
• Performance measurement
• Execution frequency histograms

10.10 Other Specialized Design Considerations

There are several other characteristics that the embedded system designer should become at least somewhat familiar with. These include the thermal characteristics of a system and the concept of thermal resistance, power dissipation, and the effects on device temperature and reliability. Another issue of importance in portable, handheld, and remotely located systems is the application of battery power storage.

10.10.1 Thermal Analysis and Design

The temperature of a semiconductor device, such as a voltage regulator or even a CPU chip, is a critical system operating parameter. The reliability of these devices is also closely related to temperature, so much so because the device’s reliability drops exponentially with increasing temperature. Fortunately, calculating the operating temperature of a device is not too difficult, since there is a simple electrical circuit analogy that is most often used to compute temperature of a device. The temperature is analogous to voltage, the power dissipated is equivalent to current, and the thermal resistance is equivalent to electrical resistance. In other words:

\[
\text{Temperature rise (°C)} = \text{power (watts)} \times \text{thermal resistance (°C/watt)}
\]

The thermal resistance of multiple mechanical components stacked one upon the other add, just as series resistors are equivalent to a single resistor equal to the sum of the individual values.

For example: Given a 5 V linear voltage regulator with a 9 V input providing 1 ampere of load current, the regulator will dissipate:

\[
P = V \times I = (9 - 5 \text{ volts}) \times 1 \text{ amp}, \text{ or } 4 \text{ watts}, \text{ of power}
\]

If the regulator is specified with a thermal resistance between the semiconductor junction and case of 1°C/watt (signified as Θjc), and the heat sink the regulator is mounted to has a thermal
resistance from the regulator mounting surface to still ambient air of 10°C/watt (signified as \(\Theta_{ca}\)), then the total thermal resistance between the semiconductor junction and ambient air is:

\[
\Theta_{ja} = \Theta_{jc} + \Theta_{ca} = 1 + 10 = 11°C/watt
\]

The temperature rise of the junction above that of the air surrounding the regulator will then be given by:

\[
T = P \times \Theta_{ja} = 4 \text{ watts} \times 11°C/watt = 44°C \text{ above ambient}
\]

If the regulator was specified to operate at a maximum junction temperature of 85°C, the device should not be operated in ambient air of temperature higher than \(85 - 44 = 41°C\) or the regulator will fail prematurely. If this is not acceptable, the designer must reduce the input voltage to reduce the power dissipated, reduce the thermal resistance by forced air flow, or change the design to another type (e.g., a switch mode regulator) so as to keep the regulator junction within operating constraints.

10.10.2 Battery-Powered System Design Considerations

The rapid increase in the use of portable, battery-operated electronic devices has spurred the development of new battery technologies for these applications. The older single-use and rechargeable battery chemistries have been supplanted by newer ones, providing improved power densities, operating life, and other enhancements. Unfortunately, these new energy storage devices come with new and different characteristics and limitations.

Batteries are generally divided into two common groups: primary (one-time discharge and discard) and secondary (rechargeable) batteries. Primary memories include the nonrechargeable alkaline and lithium cells sold commercially; secondary cells include the older lead-acid and nickel-cadmium (NiCd) chemistries as well as the newer nickel metal hydride (NiMH) and rechargeable alkaline and lithium ion chemistry products. There is also a wide range of special-purpose batteries that are optimized for some specific characteristic, such as the zinc-air primary cell, which uses atmospheric air as an “electrode” to provide very high energy density at low operating current.

Primary batteries, such as alkalines and lithium coin cells, are relatively simple to use but are often limited to one to three years of operation. This is primarily due to the shelf-life limit imposed by internal leakage current that discharges the battery slowly over time, especially at high temperatures.

The secondary, rechargeable battery types each have slightly differing charge/discharge requirements and limitations that must be considered for effective application in a battery-powered system. There are special algorithms to optimize the performance and service life of the batteries, and there are even chips designed specifically to manage the charge and discharge of common secondary battery types.
Many embedded devices must be designed to operate for long periods of time with very little power obtained from solar cells, batteries, and other limited power sources. As a result, there are CMOS processors and memories which have been designed with very low power consumption operating modes, frequently referred to as “sleep,” “power down,” or “idle” modes, that consume current in the \( \mu \)A range.

### 10.11 Processor Performance Metrics

In an effort to compare different types of computers, manufacturers have come up with a host of metrics to quantify processor performance.

The successful application of these devices in an embedded system usually hinges on the following characteristics:

- IPS (instructions per second)
- OPS (operations per second)
- FLOPS (floating-point OPS)
- Benchmarks (standardized and proprietary “sample programs”) that are short samples indicative of processor performance in small application programs

#### 10.11.1 IPS

The term IPS, or the more common forms, MIPS (millions of IPS) and BIPS (billions of IPS), is commonly thrown about but are essentially worthless marketing hype because they only describe the rate at which the fastest instruction executes on a machine. Often that instruction is the NOP instruction, so 500 MIPS may mean that the processor can do nothing 500 million times per second!

#### 10.11.2 OPS

In response to the weakness in the IPS measurement, OPS (as well as MOPS and BOPS, which sound fun at least) are instruction execution times based on a mix of different instructions. The intent is to use a standard execution frequency weighted instruction mix that more accurately represents the “nominal” instruction execution time. FLOPS (megaFLOPS, giga-FLOPS, etc.) are similar except that they weight floating-point instructions heavily to represent heavy computational applications, such as continuous simulations and finite element analysis. The problem with the OPS metric is that the resulting number is heavily dependent on the instruction mix that is used to compute it, which may not accurately represent the intended application instruction execution frequency.
10.11.3 Benchmarks

Benchmarks are short, self-contained programs that perform a critical part of an application—such as a sorting algorithm—and are used to compare functionally equivalent code on different machines. The programs are run for some number of iterations, and the time is measured and compared with that of other CPUs. The weakness here is that the benchmark is not only a measure of the processor but also of the programmer and the tools used to implement the program. As a result, the best benchmark is the one you write yourself, since it allows you to discover how efficiently the code you write will execute on a given processor with the tools available. That’s as close to the real application performance as you’re likely to get, short of fully implementing the application on each processor under evaluation.