Developing an embedded application requires hardware design, software coding and programming a system that is subject to real-world interactions. Hardware and software component must work together in an effective design.

A debug tool can

• help bring a prototype system up,
• it can help identify hardware and software problems both in the prototype and final application and
• can assist in fine-tuning the system.

Welcome to this seminar.

This session is going to discuss the debug tools, examining the reasons why you might choose one over another.
"Why Debug at All?" Why do we need this kind of tool?

• One answer is that designing an embedded system is a fairly complex activity, involving hardware and software interfacing with the environment -- and few engineers get it exactly right the first time.

• Secondly, even well-designed systems will have unknown interactions when deployed.

• Third, there may be performance issues with the code. Even correctly running code may not be effective with rapidly recurring interrupts.

  Alternately, the real time execution of code may not be as expected because of unpredictable behavior when external inputs are applied.

• Unit testing may or may not be a requirement of the design. However, to verify that each element is performing according to design, the engineer may need debug functions to test the various modules across a range of controlled conditions.

• Finally, when the entire system is finished, tuning its overall performance may require a debugger to focus on time-critical sections of code.
The factors that go into choosing a debug tool are:

- cost,
- ease of use and
- the features each tool brings to help analyze your design.

For many, cost could be the main factor. Others may find that the ease of use will outweigh costs.

Specific features of these various debug tools might be required in particular situations:

For instance, an ICE owner might use the simulator on occasion to debug code fragments: just because a quick test on the PC can be done without setting up the hardware.
Embedded system engineers have used different debug tools to get designs up and running. Here are the most common:

1. **Burn and Learn** programs a device and then puts it into the application to see if it works.
2. **Software simulation** runs on your PC to simulate the instruction execution, the CPU and the peripherals.
3. An **oscilloscope** is a standard hardware tool to monitor a few signals at a time.
4. A **logic analyzer** can monitor busses and high speed digital lines.
5. Hardware emulation or “ICE” for “in-circuit emulator” replaces the target CPU with a specialized chip designed for complex control from the emulator.
6. **In circuit debug** uses special logic on the target CPU to provide hardware debugging functions similar to an ICE.

Now we'll look a little more closely at each of these methods.
Burn and Learn is the simplest method for developing code for an embedded system:

1. Write the code,
2. program it into a chip,
3. test it to see if it works.
4. If it doesn't work, go to step one.
5. Continue going around this loop until the design works.

When something is wrong, you must figure out where the application malfunctioned and then pore over source code to spot the error. If you are on a tight schedule, trying to figure out a problem with few clues is no fun.
A software simulator is a software program running on a PC to simulate the CPU, instruction set and I/O.

Simulation provides breakpoints, single-stepping, memory access and other tools to help you examine algorithms, change values in registers and measure code performance. Incorrect logic or faulty computations can be analyzed by stepping through the code. Simulation is a quick and easy way to validate the logic of code flow.

Peripherals are also simulated so that devices such as analog to digital converters, I/O pins, serial communications peripherals and timers can be active.

The peripherals can use inputs from

- simulated waveforms,
- simulated digital signals and
- can accept manual inputs to simulate interrupts and sensors

The simulator can log register changes to files for performance analysis.
Simulator inputs and outputs can be complex. Another in this series of web seminars covers the simulator. It’s called “Introduction to the MPLAB SIM Software Simulator.” Please review that seminar for more detailed simulator information.
The advantages and drawbacks of using the Microchip simulator for debugging are listed here.

• On the plus side, it’s free and
• supports all Microchip microcontrollers and digital signal controllers.
• It needs only a PC to operate. Breakpoints can be set and code can be single-stepped, registers and memory can be monitored and changed.
• Simulated peripherals allow complex scenarios to be modeled.
• And timing tools such as a stopwatch and trace time stamp can be used for performance measurement and analysis.

The downside of a simulator is that it probably
• won’t run as fast as modern high speed embedded controllers. Depending upon the complexity of the simulation, the speed of the PC will determine the speed of simulation.

Simulation is designed to be very fast, but still hundreds --if not thousands-- of operations must be computed for each instruction and peripheral simulated.

• Ultimately, the simulation can be quite complex, with multiple inputs, interrupts and waveforms interacting with the embedded controller while the contents of registers are being logged to files. Configuring an accurate model for a complex real world application may be challenging.
Traditionally, the oscilloscope was the most common hardware tool for debugging embedded systems.

While a very good tool for viewing and measuring analog signals, it’s not particularly useful for embedded systems debugging. An LED attached to an I/O pin can serve as an almost equally good tool to determine if the application has reached a certain point in the code. Viewing complex signals on pins may show that they appear to be generating the correct signals, but interpreting the actions of the application by monitoring a few pins will likely provide little information, offering slightly more than the burn and learn method.

- Most embedded systems developers have an oscilloscope on their lab bench,
- most engineers are familiar with their use, and
- they can be relatively inexpensive

The drawback of using an oscilloscope is that it can only monitor a few signals on the pins of the embedded application -- all other internal CPU actions have to be deduced from a very few clues.
A logic analyzer can be a very good tool for debugging an embedded system with an external memory bus. A logic analyzer can log address, instruction and some data information as it executes.

The advantage of a logic analyzer is that it
- can run at full speed tracking the application.
- With an external memory bus, the analyzer can trace program flow in its trace buffer.
- Some analyzers have software disassembly tools that decipher the bus activity and create a log of instructions, register reads and writes, and time stamps.

The drawbacks for the Logic Analyzer are that
- it is expensive,
- cannot set breakpoints and
- requires quite a bit of set up.
- More importantly, since most single chip microcontrollers do not have an external bus, they do not lend themselves to robust debugging with a logic analyzer simply because most of their pins are for I/O. In these controllers instruction execution can not be traced, so using a logic analyzer is not much better than an oscilloscope.
Hardware emulation using in-circuit emulators has been the traditional method for debugging embedded microcontrollers.

• Typically ICEs use specialized hardware provided by the embedded controller manufacturer.
• The ICE system replaces the target microcontroller with a Probe.
• Emulation silicon in the Probe along with logic in the emulator Pod allows full access to internal memory, and,
• using breakpoints, registers, the CPU state and application variables can be monitored while single stepping the code.

Due to the physical characteristics of the Probe, some systems may have difficulty due to space constraints.
Here are the components of Microchip’s MPLAB ICE 4000.

In this ICE, the probe consists of

- the Processor Module,
- Device Adapter, and
- optionally, a Transition Socket. The physical package of the target microcontroller will determine whether a Transition Socket is needed or not.

A common Emulator Pod is used for all Microchip devices, and the appropriate Processor Module is plugged into the Pod to support a specific processor or processor family.

The Emulator Pod communicates to the PC via a USB connection.

Logic probe inputs and outputs on the Pod can monitor other signal lines with the trace buffer or allow external signals to initiate breakpoints. Triggers can be synchronized to other instruments, allowing a logic analyzer for instance, to use MPLAB ICE’s complex trigger settings to trace external signals.
The advantage of an ICE is that it

- Is relatively easy to set up — it can even execute code if disconnected from the target application.
- ICE allows the maximum amount of control for debugging. It controls the emulation silicon, maps program memory to internal RAM, and has a trace analyzer connected to the internal embedded microcontroller bus.
- Analyzer features in the ICE like complex breakpoints help find elusive problems that simple breakpoints can miss.

One disadvantage is that it is

- moderately expensive — from about $1000 and up.
- Also, the ICE Probes require special silicon from the manufacturer.
- And the probe may encounter physical problems when attaching to the target application. Finding adequate clearance for the Probe which is typically a larger assembly than the microcontroller chip it replaces can be difficult in compact designs, and the interconnection back to the Pod can encounter physical obstacles.
- The final limitation of in-circuit emulators, and the reason that they are becoming less common is, as chips get faster, emulation memory in the Pod has a more difficult time meeting memory set up times, and the lines from the Pod to the Probe can have physical limits to the speeds they can support.
In-circuit debugging was developed to solve two problems:

- The high cost of ICEs and
- The challenge of debugging very high-speed microcontrollers.

As processors get faster, developing an ICE with its emulator silicon and with relatively long connections from the target back to the Pod become more difficult. The ICD does not connect directly to the internal microcontroller’s high speed logic—it works by running the target microcontroller at full speed, communicating back to the ICD on a slower interface. Increasing processor speeds force the ICE designer to use faster memories and higher speed internal logic—and ultimately to be limited by the distances the high-speed signals have to travel. Increasing processor speeds do not affect the operation of the ICD.

There are new benefits from this approach.

- First, it allows the entire application to be debugged as it executes with the actual embedded controller plugged in or soldered on the target board.
- Second, after the application is debugged, and even after it’s been deployed into the field, it can be reprogrammed without removing any components.
In Circuit Debug (ICD)

Requires specialized circuitry to be on target microcontroller:
- Logic to communicate to ICD
- Logic to halt/interrogate CPU
- Logic to single step

Requires user resources:
- Memory
- Pins

The in-circuit debugger utilizes special logic placed on the chip during manufacture. This extra logic:
- allows the ICD to communicate with the microcontroller.
- It can halt the executing program, and allowing memory and CPU status interrogation.
- It allows single-stepping through the code, resulting in many ICE-like functions --only much cheaper.

For the ICD to operate, though,
- it uses some memory in the application, and
- reserves a few pins to communicate and control the target.
MPLAB ICD 2 requires that a connector be added to the application, connecting to power and ground, the MCLR reset line, and the two pins for MPLAB ICD 2 communication.

If you do not want to put a connector on the target, then you could put pins on the target to connect with a “flying lead” assembly from MPLAB ICD 2.
For the in-circuit debugger, the advantages are

- low cost (typically from fifty to a couple of hundred dollars),
- a simple interconnect that does not require the replacement of the microcontroller like the ICE.
- Yet the ICD can do most of the ICE functions.

The disadvantages of the ICD are that,

- in benefiting from a much cheaper hardware debugger than an ICE, the engineer must ensure that the pins used for communication are available for debugging;
- and some program and data memory (and possibly stack space) will be used.
- Additionally, the target must be electrically compatible with the ICD for the ICD to communicate and control the target.
- And the ICD will work only with target silicon that has ICD logic manufactured on the chip.
Here is a chart outlining the essential differences between the debug tools.

- The first column is the tool.
- The second column is cost of the tool. The Burn & Learn method and the simulator are free, so they show zero cost. The Oscilloscope is shown as medium cost, though it could run very high depending upon features—but in fact, its cost may not be a factor since it is a common piece of test equipment found in most labs. The Logic Analyzer and ICE are shown as high cost, typically $1000 and up.
- The third column is for breakpoints. Only simulation, ICE and ICD allow breakpoints. Breakpoints are needed when interrogating registers, checking CPU status and stepping through code.
  - Some applications, such as motor control, may not allow for halting at a breakpoint. In these cases capturing the program flow and register reads and writes in the trace buffer may be a good alternative. The Logic Analyzer or ICE trace buffer can be used to follow register reads and writes, allowing the engineer to monitor the values in registers and variables at select positions in the trace.
- The last column is a little arbitrary. Burn & Learn and the Oscilloscope are the easiest to use, because the engineer is familiar with programming a device and using an oscilloscope.
  - Simulation can be easy or more difficult depending upon the peripherals and signals being simulated, so it shows as medium complexity—your usage will dictate the complexity of the set up.
  - The Logic Analyzer and ICE are listed as medium ease of use, although tracing all the address and data busses might require a great many clips and wires—still it is a common task done often by firmware engineers.
  - The ICE is simpler to set up since all the trace lines are connected internally in the Probe and Pod.
  - ICD requires observance of some design considerations for it to function, and it shares some resources on the application. Minor hardware and software trade-offs are necessary to use the ICD, but the ICD provides many of the functions of the more expensive ICE.
Thank you for going through this presentation. Please take the time to provide us with feedback on the usefulness of this seminar, and check out our other technical webinars on the Microchip web site.

Also available on the Microchip web pages are a series of write-ups called "Getting Started with Microchip Development Tools." All tool manuals are available for download from the Microchip product pages, and forums are available to discuss questions about embedded systems development with other experienced designers.