Debugging embedded software is my least favorite activity, but unfortunately it's a necessary evil. Thankfully, advances in technology and tool chain innovation have resulted in a plethora of techniques to drastically speed up the debugging process. Let’s examine a few of them, starting from traditional debugging by break-point through the more advanced instrument trace techniques.

Technique #1 – Traditional break-point debugging

Every developer is familiar with the traditional debugging technique of setting breakpoints, executing code, and then stepping through the code while monitoring registers and variable values. Break-point debugging is a technique that I see used more than any other technique. This is disheartening, however, because break-point debugging is inefficient and generally yields sub-optimal results.

So, then, why are breakpoints used so often? The main reason seems to be because breakpoints are easy to use, are readily accessible, and developers are optimistic that breakpoints are the right tool for the job. That optimism needs tempering. Breakpoints have the potential to disrupt the systems real-time performance and can suck a developer into a black hole of endlessly single stepping through code blindly looking for a solution to their problem.

Technique #2 – IDE Value Graphing

Nearly all modern debuggers and IDEs now allow a developer to monitor the value stored inside a memory location such as a variable. A developer selects the memory location and the value refresh rate, and then starts the debug session. Some IDEs have the ability to monitor the value built into the IDE while others rely on using an external software.

Value monitoring can be very useful, but it is far more valuable if the data being monitored is tied to a graphical representation. Graphing the data values in real-time can be extremely useful for discovering unexpected changes or to verify that a particular waveform is generated. Take, for example, a three phase BLDC motor. A developer may want to monitor the current and voltage on each motor leg, which need very specific waveforms to be generated in order to drive the motor. Plotting the voltage and current on each motor leg can allow a developer to visualize what is happening in real-time.

Technique #3 – Reroute printf to SWO

On higher-end ARM Cortex-M parts, such as the M3/M4, developers are provided with additional debugging capability known as the Serial Wire Viewer (SWV). The SWV comprises the standard serial wire debugger in addition to a Serial Wire Output (SWO). The SWO can be used to do really cool things such as retrieving the program counter, event counter, and trace data, to name a few.
Developers can customize the information that they would like to have transmitted over the SWO. Many developers set up `printf` in order to get debugging information from their embedded system. Rather than using serial pins on the microcontroller, a developer can use SWO to reroute `printf` information through the debugger. Using the debugger in this way can save the need for a dedicated serial interface, eliminate the development time for the UART or USB device, and is far more efficient. Overhead that would have originally been used in the application is now off-loaded through the SWO and debugging hardware, which saves precious clock cycles that would have otherwise been used by the application code.

Technique #4 – RTOS Tracing

Trying to peer through the veil at what an RTOS is doing can be quite challenging. Developers don’t want to disturb the real-time system performance but still need some method to understand the system's behavior. Blinky LEDs have often been the go-to trick, but more recently trace tools have added an amazing technique to the developer's toolbox. Free and commercially available RTOS trace tools exist such as TraceX, SystemView and Tracealyzer, to name a few.

Trace tools allow a developer to analyze when the RTOS is idle and when each task is entered and exited. Developers can monitor for system exceptions, response time, execution time, and many other critical details needed to properly develop an embedded system. The coolest feature available in RTOS trace tools is their ability to graph what is happening in the system. Reviewing and monitoring timing diagrams in real-time or in the recorded log can help give developers a confidence level that the system is working as expected or help them to discover small issues that would have otherwise taken a lot of time to discover.

Technique #5 – Use instruction trace technology (ETM/ETB/ETM)

Sometimes developers face debugging problems that are just at the lowest levels imaginable within the processor. Trace technologies exist that can monitor individual instructions the processor executes. Such low level tracing can be useful for monitoring branch coverage when testing and validating software. The debugger tools used for instruction trace are different than those a developer would use for Serial Wire Viewing and generally cost a little bit more.

Closing Thoughts

Debugging tools and technologies have been rapidly evolving over the past few years, especially for higher end microcontrollers. Engineers in general are visual creatures and tool vendors are finding ways to reveal in visually stimulating ways what exactly is happening within a real-time system. Setting up debugging tools can require some upfront time but the potential to spend less time debugging and more time designing is well worth the time investment. At a minimum, developers should become familiar with the different debugging tools and capabilities available to them so that when problems arise and the system needs to be debugged, they can select the right tools exist to get the job done. What other techniques do you use to help engineers debug their systems faster and more efficiently?

Jacob Beningo is principal consultant at Beningo Engineering, an embedded software consulting company. Jacob has experience developing, reviewing and critiquing drivers, frameworks and application code for companies requiring robust and scalable firmware. Jacob is actively involved in improving the general understanding of embedded software development through workshops, webinars and blogging. Feel free to contact him at jacob@beningo.com, at his website www.beningo.com, and sign-up for his monthly Embedded Bytes Newsletter here.
Also see:

- 7 Tips for debugging embedded software
- The hidden world of MCU start-up code
- How to debug embedded systems
- Speed debugging, gain system behaviour insights with trace and visualizations
- ARM Cortex-M4-based ultra-low power MCUs target connected devices
- ARM Cortex-M embedded development tools for the Mac