BILL WONG | Embedded/Systems/Software Editor
bill.wong@penton.com

The Many Faces of Embedded Security

While there is no such thing as absolute security, it is possible to make things much more difficult for a would-be attacker.

That embedded system you are designing better be secure, or it might be hacked. Of course, preventing this depends on what you're going to protect. There are those who simply want to crash a system, others who want to take it over for nefarious purposes, and still others who wish to get inside to "steal" those valuable algorithms.

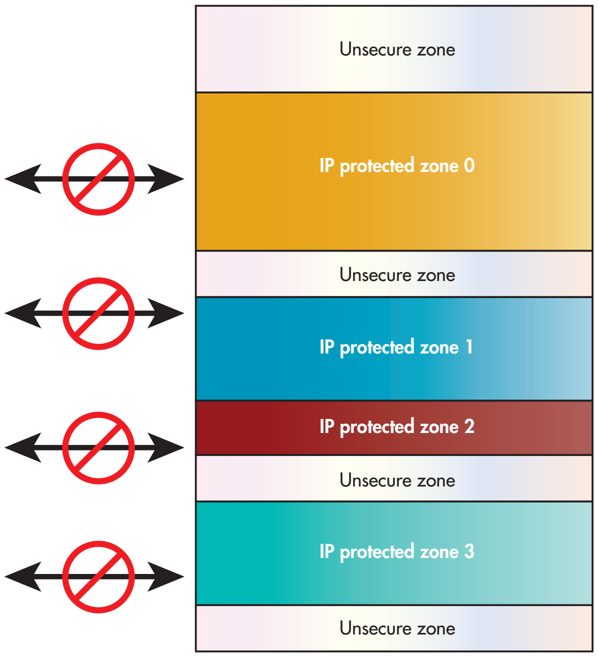
In the past, processors simply started running and security was something that was implemented in the program or operating system. It came down to who you trust and how well they did their job. Burroughs mainframes I worked on many years ago were protected by the compilers, as well as a file system that prevented hacking by limiting the instructions that were emitted or preventing files from being marked as executable.

Initially microcontrollers and microprocessors took a similar approach, with features such as memory protection being added that allowed applications to be placed into a sandbox. Virtualization extended this further to the point where high-end systems virtualize the entire system. Securing the sandbox works, assuming the base software/hardware can't be compromised.

That assumption is not always warranted. For example, most hardware has some form of software involved like the controllers on hard drives. One may think that the data on the hard drive would be maintained and just the data needed protection, possibly using an encrypted hard drive. Unfortunately, we now know that this is not the case, since modifying this firmware is one way a group has infiltrated some storage devices *(see "Hacking Hard Drives and Other Nasties" on electronicdesign.com)*.

Secure boot and encrypted code are ways to prevent the initial attack from succeeding. One-time-programmable (OTP) or ROM-based solutions others, but these prevent essential field updates. Some platforms also allow the debug or JTAG support to be disabled, often via OTP flags, so application code is no longer directly accessible—making reverse engineering more difficult. Some systems go to tamper-resistant extremes that erase the code and data if an attack is detected.

Texas Instruments' (TI) MSP432 has an interesting approach to protecting application code that has some useful implications. The MSP432 is based on an ARM Cortex-M4F core *(see the figure)*. The protection scheme allows multiple blocks of flash to be execute-only, but it takes this a step further by allowing the code to access data within the same block. JTAG will not reveal the contents of the code or data.



Texas Instruments' MSP432 microcontroller is based on an ARM Cortex-M4F core, but adds a software IP protection scheme that allows multiple blocks of flash to be execute-only. Even JTAG will not reveal the contents of the code.

There can be multiple blocks defined and encryption keys are used to verify and allow updates so a block cannot simply be removed and replaced by a malicious actor. The MSP432 does not have a security key store or OTP support, but these can be implemented using the software protection scheme.

The approach allows a vendor to include runtime support on a chip and then provide it to a developer. The developer can call the support routines, but they won't have direct access to the code that could be disassembled. The scheme actually works for multiple vendors to provide services on the same chip. ed