Bootloader design for microcontrollers in embedded systems

Jacob Beningo - September 03, 2013

Microcontrollers have proliferated into every nook and cranny of our daily lives from simple 8-bit devices that control our toaster ovens to powerful 32-bit DSPs that provide us with the rich media and entertainment that we have all become accustomed to. Without microcontrollers, our lives would not only be less exciting but we would lose a level of control over our world that we can no longer live without.

Billions of microcontrollers are sold each year with the number continually climbing. What happens to these microcontroller based products when millions of units have shipped and a software “enhancement” needs to be made? Does every unit need to be returned to the manufacturer every time the software is updated? Are televisions, blue-ray players and other devices returned to the manufacturer periodically so that the customer can continue to have the latest and greatest software operating on their device? The obvious answer to these questions is absolutely not and the primary reason why is that most systems ship with a boot-loader on-board.

A boot-loader is an application whose primary purpose is to allow a systems software to be updated without the use of specialized hardware such as a JTAG programmer. In certain cases, it can also be the earliest point at which the integrity of an embedded system can be checked. The boot-loader manages the systems images. There are many different sizes and flavors to embedded boot-loaders. They can communicate over a variety of protocols such as USART, CAN, I2C, Ethernet, USB and the list goes on for as many protocols that exist. Systems with boot-loaders have at least two program images coexisting on the same micro-controller and must include branch code that performs a check to see if an attempt to update software is in progress.

During the initial stages of product development it is very common for the boot-loader to be overlooked by the development team. The primary reason for this is that the boot-loader is not the primary end product that is going to be sold to the customer but the boot-loader is potentially the most important part of that product. The boot-loader allows a company to launch their product with software that only fulfills a portion of their final feature set and then add features to their product once it has been launched into the market. It also allows them to fix bugs that are discovered after the system has been released into the wild.

For an embedded software engineer, a boot-loader requires a full understanding of how a processor works, how to utilize its memory and how to work on the processor at the lowest levels. Boot-loader development can be an extremely challenging endeavor to undertake but absolutely a rewarding one. Once a developer has gone through the process, each additional boot-loader becomes easier and easier to implement by following a common and consistent approach to boot-loader design.

The purpose of this paper is to break down the components necessary to develop a boot-loader and
present it in an easy to understand methodology that the reader can then use to implement their own boot-loader. While boot-loaders are small programs, they often bring out the worst and nastiest bugs that can be very intimidating to a developer. Not only does the programmer have to dig into flash programming but there is also a need to do a deep dive into memory maps, re-locatable vector tables, copy down functions, flash partitioning, code branching and a number of other tasks which can at first seem monumental and nearly impossible to overcome on a short development schedule.

Download and read the complete paper, “Bootloader Design for Microcontrollers in Embedded Systems.”

Jacob Beningo is a lecturer and consultant on embedded system design. He works with companies to develop quality and robust products and overcome their embedded design challenges. Feel free to contact him at jacob@beningo.com, at his website www.beningo.com or on twitter @Jacob_Beningo.

If you liked this and would like to see a weekly collection of related products and features delivered directly to your inbox, click here to sign up for the EDN on Systems Design newsletter.