10 software tips for hardware engineers

Jacob Beningo - January 28, 2013

Embedded system design often requires not only an understanding of the hardware but also of how the software affects and interacts with it. Designing hardware requires a certain engineering paradigm that can be a complete polar opposite of designing software. When transitioning from hardware design to include software design, here are ten software tips that hardware engineers should keep in mind when they start developing software.

Tip #1 – Flowchart First, Implement Second
When an engineer first steps into the realm of developing software, there is an intense temptation to jump right in and start writing code. This mindset is the equivalent of trying to lay out a PCB before the schematics have been completed. It is critical when sitting down to develop software that the urge to start writing code be ignored and instead an architectural diagram of the software be developed using flowcharts. This will give the developer an idea of the different parts and components required for the application much like how a schematic tells an engineer what hardware components are required. By doing this the program overall will stand a better chance of being well organized and thought out which will save time and headaches in the long run by decreasing debugging time.

Tip #2 – Use State Machines to Control Program Flow
One of the great software inventions of the 20th century was the state machine. An application can often be broken up into multiple state machines each of which controls a specific component of the application. Each of these state machines has their own internal states and state transitions that dictate how the software reacts to various stimuli. Designing software using state machines will ease in the development of software that is modular, maintainable and easy to understanding. A wide variety of resources exist that demonstrate state machine theory and algorithms.

Tip #3 – Avoid the Use of Global Variables
In the old days of functional programming, function came before form with the programmers' sole goal being to make the program operate as expected as quickly as possible without regard for program structure or reusability. This programming paradigm held no apprehension about using variables that were global in scope that any function within the program could modify. The result was an increased chance of variable corruption or misuse of variables. In the new recommended object-oriented paradigm, variables should be defined in the smallest possible scope and encapsulated to prevent other functions from misusing or corrupting the variables. It is therefore recommended that you limit the number of variables that use a global scope. These variables can be identified in the C language by the use of the extern keyword.

Tip #4 – Take Advantage of Modularity
If you ask any engineer the part of a project most likely to be delivered late and over budget the answer will be the software. Software is often complex and can be difficult to develop and maintain especially if the entire application resides in a single file or multiple files that are loosely correlated. To ease maintainability, reusability and complexity, it is highly recommended to take advantage of the modularity of modern programming languages and break common functionality into modules. By breaking the code up in this manner will allow the programmer to start building libraries of functions and features that can then be reused from one application to the next thus improving code quality through continuous testing in addition to decreasing time and development costs.

Tip #5 – Keep Interrupt Service Routines Simple
An interrupt service routine is used to interrupt the processor from the branch of code that is currently being executed in order to handle a peripheral whose interrupt has just been triggered. Whenever an interrupt is executed there is a certain amount of overhead required to save the current program state, run the interrupt and then return the processor to the original program state. Modern processors are much faster than they were years ago but this overhead still needs to be taken into account. In general, a programmer wants to minimize the time spent in interrupts in order to not interfere with the primary code branch. This means that interrupts should be short and simple. Functions should not be called from an interrupt. In addition, if an interrupt starts to get too complex or take too much time, the interrupt should be used to do the minimum required at the time such as loading data into a buffer and setting a flag to then allow the main branch to process the incoming data. By doing this it can be ensured that the majority of the processors cycles are being spent running the application and not just processing interrupts.

Tip #6 – Use Processor Example Code to Experiment with Peripherals
When designing hardware it is always helpful to build prototype test circuits to make sure that an engineers' understanding of the circuit is correct before laying out a board. The same can be done when writing software. Silicon manufacturers usually have example code that can be used to test out parts of the microprocessor so that the engineer can get a feel for how the part works. This allows insights to be made into how the software architecture should be organized and any potential issues that could be encountered. Identifying potential road blocks early in the design process rather being found in the last hours before shipping a product. This is a great way to test code snippets out beforehand but be warned that manufacturer code is usually not modular and easily used in the actual application without considerable modification. Over time this has been changing and may one day result in production ready code right from the chip provider.

Tip #7 – Limit Function Complexity
There is an old saying in engineering called KISS; Keep It Simple Silly. When dealing with any complex task the simplest approach is to break it up into smaller and simpler tasks which are more manageable. As tasks or functions become more complex, it becomes harder for humans to keep track of all the details without errors slipping in. When a function is written the complexity may seem appropriate at the time but how an engineer will view the code when it needs to be maintained six months down the road should be considered. There are a number of techniques for measuring function complexity such as cyclomatic complexity. Tools exist that can automatically calculate the cyclomatic complexity of a function. General rule of thumb suggests that functions with a cyclomatic complexity below 10 are desired.

Tip #8 – Use a Source Code Repository and Commit Frequently
Making mistakes is part of being human and when humans write code they don't miraculously change. That's why it is critical that developers use a source code repository. A source code repository allows a developer to check-in a good version of code with a description of what changes were made to the code base. This allows the developer to not only revert or go back to an old version of code but also compare previous versions for changes. In the event a developer makes a bunch of changes that then break the system, going back to a good version of code is just a click away! Just remember that if code is not committed frequently the repository will not work as intended. Waiting two weeks to commit code and then going back will lose a lot of work and time if an irreversible change is made!

Tip #9 – Document the Code Thoroughly
When in the heat of software development battles it is very easy to focus just on getting the code written and debugged and ignoring documenting it. Documentation often becomes an end of project task as it is the last

thing on a developers mind when the pressure is on. However, it is important to document the code when it is fresh in your mind so that a future developer or your future self can read the comments and understand how the code works.

Tip #10 – Use an Agile Development Process
When doing engineering of any kind it is always recommended that some sort of process be defined and followed whose result is consistent quality, costs and on-time delivery. Software developers have been successfully using the Agile development process to develop quality software. The process allows for requirements to be developed with priorities. The highest priority tasks are performed first within a scheduled period of time known as an iteration. The beauty of the process is that it allows the software development process to be fluid, allowing requirements and tasks to adapt and change with each iteration based on the results and needs of the client.

Tip #10a – Stay on Top of Developing Technologies
One of the greatest places to learn about the latest tools and techniques being used to develop embedded software is to attend one of the Embedded Systems Conferences that are held three times a year at various locations throughout the United These conferences draw engineers from around the world, providing the opportunity to interact, attend seminars and hands on exercises that will improve an engineer’s understanding of software development. In addition to attending conferences, EDN offers a wide variety of blogs ranging in hardware and software topics to keep an engineer always engaged and learning so that they are ready to apply cutting edge technologies on their next development project.
