Safe start to software test

Testing embedded software is as important as hardware test if an embedded system’s operation is to be trusted. By Tim Fryer.

Embedded software testing is a discipline that is both easier and more efficient if it can be conducted concurrently with software development, and preferably with the hardware. The extent to which this can be done varies.

For example, ByteSnap, a design consultancy specialising in embedded system development, can be presented with projects at the concept level and so the complete design flow – hardware and software – is in its hands.

Resource Group, meanwhile, supplies engineering and support services to safety critical sectors. Typically, companies in these sectors want to retain the development work in house, but outsource the review, analysis and testing to Resource.

These two different starting points result in different processes, including how the software tests are created and executed. ByteSnap may start with a list of customer requirements, from which it will select a reference design, typically from one of the processor companies, that at least meets and most closely matches these requirements. Graeme Wintle, director of ByteSnap, comments: “This board usually has additional features that we don’t need, so we will develop a board based on this reference and will exclude the parts that we do not need. We do the same thing from the software point of view, taking the reference software and take out the drivers for the parts we have removed. This development then needs to be tested, because we have effectively made changes to the operating system, or at least the drivers to the OS. In this case, the easiest way of testing and debugging is a serial port to the device and that will provide pointers as to where the code is going wrong. That is the process. Obviously you try and start with something that is as close as possible to what you want the end product to be.”

In cases where someone is developing software with safety implications, they must adhere to industry standards and demonstrate compliance, which is where companies like Resource come in. “Debugging is a process undertaken by the developer to demonstrate compliance to the requirements; it tends to be a demonstration of confidence in the developed system,” said John Minihan, director of Resource Group’s REP Systems and Solutions division. “Testing, or more importantly, verification (review, test and analysis), tends to be a more rigorous approach, often by an independent person, to demonstrate that the developed system conforms to standards, is robust and is functionally correct.

“Testing is a very specialist skill and can take years to learn and perfect. There are many testing techniques and the tester needs to understand when and how to use them. But the tester’s job is basically to ensure there are no dormant software bugs in the system, so the test cases they develop must be able to exercise the software in such a way as to uncover all the anomalies in the system. A lot of techniques for

Fig 1: The software development life cycle

Source: Resource Group
developing software cross over to firmware, where robustness and verification is equally important."

Here, the start point is the architectural design and allocation of the flowed down requirements into high level blocks of common functionality (object orientation). The system can then be partitioned into smaller and smaller blocks of manageable functionality, with clear inputs and outputs, functional activity, data encapsulation and memory/timing constraints.

For high integrity software, testing is performed in stages. Low level test comes first, where the software/firmware modules are tested for functional correctness and robustness at their lowest level of decomposition. Often, low level test is run on a representative target to find errors in the compiler or features of the target environment not anticipated at design time.

The next stage is integration test, where blocks are combined into larger functions and tested end to end to verify that the higher level requirements are met and that the integrated modules work together. Integration testing will tend to find faults in scaling, timing, sequencing, scheduling and interconnects, as well as functional problems. A secondary part of integration test can be hardware/software integration, where large chunks of the software (if not the entire software) are loaded onto the hardware (or representative blocks of hardware) and tested end to end against the high level requirements.

The final stage is the system level test, where the entire system is tested against the customer’s requirements to demonstrate compliance to the specification.

As Minihan explained, while test development can start early in the process, it is not always advisable to put too much resource in straight away. “In an ideal world, once customer requirements are available, the tester can start developing their plans and strategies and even develop the test cases. As the design lifecycle continues, the tester can develop lower and lower level test cases until the actual code is produced. These test cases can then start to be executed and as more modules become available, they can be integrated and integration tests can start. However, in reality, the design is likely to develop in an iterative process, so it is possible that tests developed too early will be superseded by design changes made at a later stage. For this reason, the early stages of a project deal with establishing the test vehicles, the test plan and methodologies; testing only starts in earnest once the development team has reasonable confidence in their developed components.”

The low level tests for ByteSnap can be carried out through USB or Ethernet ports, if available, but this is not always the case, according to Wintle, who said there needs to be an element of ‘design for debug’, citing a case where a jack had to be included on prototypes, that wouldn’t be on the production board, just to allow debug.

From there, the nature of the test depends on the operating system. Wintle said: “In a Linux environment, you have a command line shell over which you can issue commands. In a Windows CE environment, you don’t have that level of connecting. What we would normally do is have a little terminal application that is running, talking over the serial link and asking for commands so we can do things like run a process, kill a process, list things that are running on a device – very simple driver level tests to see if the drivers are working.”

Companies like ByteSnap, developing the system software and applications, can look to reduce development time with variations on reference designs and applications like its own interface framework, SnapUI.

However, at the safety critical end of the market, Minihan issues a word of caution for those looking to use some of the newer debug tools or other ‘shortcuts’. “Some of these newer tools use the actual code to develop the interfaces and can also run tester supplied inputs through the real code to suggest output values. If the tester is lazy and doesn’t check correctly, you can easily end up with code generated tests testing the code. Ideally, testers should develop their own test cases and expected results directly from the requirements, then put them into the test tool and verify that the real code produces the expected results for the input stimulus.”

When asked what is the best starting point for a successful project, Wintle replied: “Something that allows you to have a good way of debugging the application is a good start.”

“Something that allows you to have a good way of debugging is a good starting point”
Graeme Wintle