Certification of High Integrity Software

Authors:

Paul E. Bennett
Transport Control Technology Ltd.
email: peb@transcontech.co.uk

Malcolm Bugler
Malvatronics Ltd.
email: MalcolmB@compuserve.com

Abstract:

As software is employed in more and more applications for which Safety and/or security are major concerns, there is a growing demand by Notified and Regulatory Bodies that such software is certified for compliance with requirements, legislation, and standards. That certification can easily be applied to Forth source in a manner that will stand the scrutiny of close inspection and audit is to the benefit of the language.

This paper describes the method of applying certification to Forth source code and describes some of the lessons learnt from its application to Forth software in a medical "life support" product.

Introduction

Certification of Software is being demanded with increasing frequency by regulatory bodies and those customers who are seeking assurance of the continued correct operation of their mission critical systems. Applying such certification is by no means easy for the vast majority of software as the compilers used are far too complex to be fully verified for correct production of code. Only in Assembler, Forth and similar simply structured compilers can such a feat be achieved easily.

Experience

The authors recently undertook the task of certifying a medical life support product [EF97MB] to meet the CE requirements for programmable medical systems. The software was written in FORTH and therefore lent itself readily to the certification approach outlined in this paper. The learning experience from performing the certification was very valuable and several interesting issues came to light during the process.

The Development Environment

The software development was performed as a sub-contract with one programmer working remote from the customers site, and making weekly visits to the clients premises. This environment was very productive and allowed close liaison between client and sub-contractor. This level of interaction proved to be essential due to the lack of a detailed specification (covered below). FORTH allowed rapid coding and testing of modules, and also provided a very portable development ‘workbench’, which was brought to the clients premises on each weekly visit. The client, who had no previous experience of software development, was put at ease by the relative simplicity of the development environment.

Specifications

The major problem found here was the client did not really produce an adequate specification for the system They had been used to the design of purely electromechanical devices with no programmability, these being very simple to specify. This lack of a specification for the software resulted in difficulty in communicating requirements and in proving the system performance. The major learning point here is that it cost the client at least 50% more in software engineering because the specification was inadequate. It also resulted in a great deal of frustration for the sub-contractor who was designing and coding the software as many sections of code had to be re-worked. One further area that was not sufficiently covered was that of risk management. Almost all regulatory bodies prefer that software is designed and documented from the risk control viewpoint. A risk management worksheet is therefore a necessary adjunct to the specification process.

Coding Standards

Another area which caused problems was that there were no laid down coding standards at the start of the project. This resulted in several non-optimum practices being adopted during software development, some of which were highlighted in the later verification phase, as follows:-

a. A great deal of variables and constants were used which is not the best practice in FORTH based systems

b. Many definitions were greater than 16 lines of code. FORTH would usually be more factored

c. Insufficient documentation of stack parameters and very terse glossary texts.

d. A large number of ‘magic numbers’ used without sufficient descriptive explanation.

e. Cross-reliance’s of modules weakening coherence and increasing coupling.

Specification Reviews

As the original specifications were inadequate, changes in the requirements were done ‘on the fly’ with no real formal reviews. This again resulted in more re-work when the real requirements were finally made clear.

Verification & Validation

This part of the process did run very smoothly with no real problems except that with very loose specifications, it took longer than really necessary to define and execute a final system test. Several design problems did become apparent during this phase which required some remedial work, but for the most part the code was proved to be very resilient. Some of the issues raised are detailed below:-

a. Year 2000 issues and calculation of leap years.

b. Pointer ranges unchecked.

c. Documentation errors in stack comments and glossary texts.

d. Intricate dependencies of modules requiring almost complete compilation of the source.

e. Three serious software problems.

3rd Party Software Issues

One issue was that the FORTH low level and nucleus code supplied by the vendor was un-certified and did require certifying as part of the overall process. This increased the V&V time and resource required, but of course, now certified, it can be treated as a validated programming surface and used in other products with confidence. However, to have code supplied pre-certified would be considered a great advantage, especially when porting code across different processor platforms.

Revision Control

Although the client had a very well controlled and detailed revision control system for drawings and other documentation to ISO9001, there was no dedicated system for the control of software. As a result, the sub-contractor actually performed the revision control. This was fine up to a point, but effectively shielded the client from needing to address this issue until the regulatory requirements of CE forced them to do so. Again, if this had been planned for at the outset, much of the panic would have been avoided.

Documentation

Here it was a story of too much, too late. The entire issue was not thought about until right at the end of the project, which resulted in a 2-3 month extension in the time scale. This was primarily caused by the client not knowing the real requirements and not seeking help on them early enough in the process. This made the last few weeks of the project very hectic and rushed in an attempt to 'just get it through' CE.

The Lessons Learnt

A number of lessons arose out of this project. They are listed below:-

1. Ensure a complete and signed off specification exists for the project before any design work is commenced. If you are the software sub-contractor, do not even quote for a project without having one unless production of a specification is a part of the project itself. Always review the specifications and obtain a contractually binding agreement between client and supplier.
2. Produce a risk management worksheet which lists all the identified hazards and details how these hazards have been mitigated in the design cross referencing this to the software documentation.
3. Before starting software design and coding, define and agree the coding standards to be used. These could be based on an existing standard, (MPE, TCT, etc) or be 'home grown'.
4. Specification and code reviews should be mandatory to ensure all who have input to a product design process are consulted and to make specifications as complete as possible.
5. Verification and Validation should be applied formally and a written report produced which is acted on and reviewed. Reviews should be performed at the module level as soon as each module completes coding..
6. All 3rd Party Software should be fully verified and validated unless supplied in that state by the vendor. Vendor applied production and certification processes should be audited as part of the ISO9001 based assurance evaluations.
7. An integrated software revision control system is not essential if you are attempting to control the production, issue and maintenance of mission critical software, however it does make the task much easier.
8. Start the documentation (and it's control record) when you start the project. This will mean that you can submit completed documentation to a regulatory body as soon as your testing is complete, leading to an improved time to market and of course profit!

Doing the task right.

The Development Environment - ISO9000, TickIT, Def-Std 00-56

All software development requires some form of systematic process which is well defined and controlled. Examples of such processes are ISO9000, TickIT and Def-Std 00-56. Although these themselves provide good guidance, they are not essential. Many organisations posses excellent processes even though they have not proceeded with full certification to a 'standard'. The basic essentials in a development process are as follows (*see also [EF97PEB] and [SSS98PEB]*):-

- Written procedures on how to perform the process
- Documentation standards on how information should be presented.
- Someone responsible for it being done.
- Visibility and support at senior management level.
- Some form of audit to verify the process continues to produce the desired results.
- Control of the documentation.

Specification of Requirements - Granularity

Every system or product needs a specification which completely describes its form, function, integrity and environment. Major issues with specifications are granularity, technical content and relevance. It is far better to break a complex project down into smaller, modular specifications than to have to deal with a single 'tome'. In this way a 'complete' specification would comprise separate documents for such as System Requirements, Risk Assessment, System Architecture Description, Software Requirements, Interface Design, Electronic Hardware and Test and Validation, although this is not intended as an exhaustive list.

Traceability Issues

It is all very well detailing the requirements in the various specifications, but it has to be proven that the requirements have been implemented correctly. In order to verify this a cross reference between the implementation and the requirement it satisfies is required. This is either in the form of a table (traceability matrix) or in document references and/or virtual links (hyperlinks). These cross references would need to be checked during each of the review stages.

Review of Specification

One other vital requirement for a specification is that it is reviewed and agreed by those who have interest in the product or system. Reviews should be formal and cover the following three aspects:-

- **Completeness** — In that the product or system is fully described by the specification, specifically its form, function, integrity and environment.
- **Correctness** — That the original requirements are accurately represented at each level in the specification to perform the desired functionality.
- **Intention** — The original intention of the system and only that is represented in the specification without other requirements being added or side effects caused.

It is also important to review all levels of the specification as a separate activity. This is especially so where different technologies are dealt with by separate documents.

Review of Code

So we now have code that has been written to meet the specification at all levels. Given a well controlled development environment, the code should be accurate, well documented and laid out. However, it requires further effort to ensure it meets the full system requirements. Consideration here should be given to designing and validating to the component level. An example would be to take the FORTH system itself as a core module and verify it as a programming surface compliant with ISO/IEC 15145 [ISOstd1]. In this way when your application is built on top of a certified FORTH kernel it would behave in exactly the same way regardless of underlying processor or technology. This can be applied to every level in a system from device driver to icon!

In performing the review, the following tasks would be undertaken:-

- **Comparison to Specification** — Ensuring that the code under review deals with the subject of the specification from which it was created.
- **Compliance of Specified Intent - Code Inspection** — Confirmation by inspection that the code under review calls on only those elements that will support the intended functionality described by the specification.
- **Compliance of Execution - Function Test** — Validation of the intentional behaviour of the function under review.
- **Behaviour in face of adversity - Limits Test** — Verification that the code's functional integrity is not compromised by out of bounds values or unintentional stimuli.

Verification and Validation

A certain amount of confusion has often surrounded these two terms. We will endeavour to remove some of this confusion.

- **Verification** An Inspection of the Code Structure, Layout, Commenting, Stack Item Consistency, Data Typing adherence (ie. not using @ on byte sized data items), Naming Conventions followed, Adherence to Coding Standards, Initialisation of Data Structures and Conformance to Specification.
- **Validation** A confirmation by test that operation is as described and expected,; that out of limit conditions are either handled or ignored as appropriate; does not cause adverse operation in the face of adverse stimuli (ie: unauthorised writing over data areas for which the component has no reference).

Certification and Documentation.

In the process so far, the specification documents should fully define the "requirements", architecture, functionality, detailed design, interfaces etc. to the level that well laid out and formatted source code would be the final component of the full system description. Validation and verification of the resultant source code would then almost guarantee adherence to many recognised certification standards [ENMedi1], [IECstd1], [Defstd].

More and more regulatory authorities are insisting on risk assessments being performed as an integral part of the specification process. Specifications, themselves, are more and more structured in terms of risk management than system functionality. A risk management work-sheet may almost be the second point after a features list for a product or system. It is probably better to consider the product a hazard until proved otherwise.

Creating the Certification

Certificate Creation is simplicity itself. Using the natural structure of a FORTH definition, and the aspects included in the coding standard, a paper certificate can be created in a standard form. Space is allowed on the form for the inspection and testing personnel (preferably not the author) to sign acceptance of the code or enter comments about problems or potential problems (a form template is included in the appendix to this paper). For the software author, certification is just a form filling exercise.

The Form's Anatomy

The anatomy of the form is listed below by the title in the box.

- **Original Design** The author of this particular word or module
- **Organisation** Identification of the organisation or company to which the author belongs.
- **Issue** The issue identification of the word or module. This should be incremented with each change made to the source code. A history should be maintained in addition to this information.
- **Date** Date of source code submission for certification.
- **Sht of** In multiple word modules this helps keep all document pages in order and easier to find. A word, however should only occupy one sheet.
- **Word/Module Requirements Description (concise)** Essentially the glossary entry for the word/module.
- **Input Stack(s)** A list of the input parameters for the word/module passed on the parameter, return or any other stacks.

- **Output Stack(s)** — A list of output parameters for the word/module passed on the parameter, return or any other stacks.
- **Word/Module Definition** — The source code of the word or module.
- **Code Check** — The signature of the person conducting the static code inspection who signs when he is satisfied that the code performs the action described and declared in the Requirements Description box.
- **Function Test** — The signature of the person conducting the functional performance testing who signs when he is satisfied that the code performs the action described and declared in the Requirements Description box.
- **Limits Test** — The signature of the person conducting the adverse stimuli testing who signs when he is satisfied that the code continues performs the action described and declared in the Requirements Description box and sensibly handles stimuli/values outside the range declared in the Requirements Description box.
- **Test Comments** — Any comments or observations by the inspection or testing team relating to the code described above.

Whilst the form is useful for certification effort it does pose some management problems. It takes time to fill out ready for the testing effort (not a great deal of time for the author of a single word but over many words it adds up). It is therefore evident that an automation tool is required which can produce the form in a suitable format for direct printing from the electronically held source code. For this reason certain constructs were included in the coding standards [Fcoding] which aim to assist in this production.

Programming Surfaces

Now we have a method of certifying the software, it is important to be able apply this logically to any project or system. The principal outlined here is that of programming surfaces. A programming surface is the interface to a software kernel or operating system that has a documented set of functions that provide services for an application. An example of a programming surface is the MSDOS operating system.

In order to be able to fully certify an application, then it is first required to certify the programming surface. For example, any application built on an MSDOS platform that requires certification, will first require that the MSDOS system be certified. Obviously, this would only be required to be done once, with future applications only requiring their own code to be certified.

With a FORTH system, this means that a cross-compiler and it's kernel would only need certifying once, and then could be used as the platform for many products, only the new application code requiring certification after that. Obviously, if any of the core cross compiler or kernel code is modified in the future, the changed sections would need re-verification.

In this way, an application could be moved readily from one hardware platform to another, as long as the two programming surfaces provided the same functions and were fully certified. The diagram in Figure 1 details this approach.

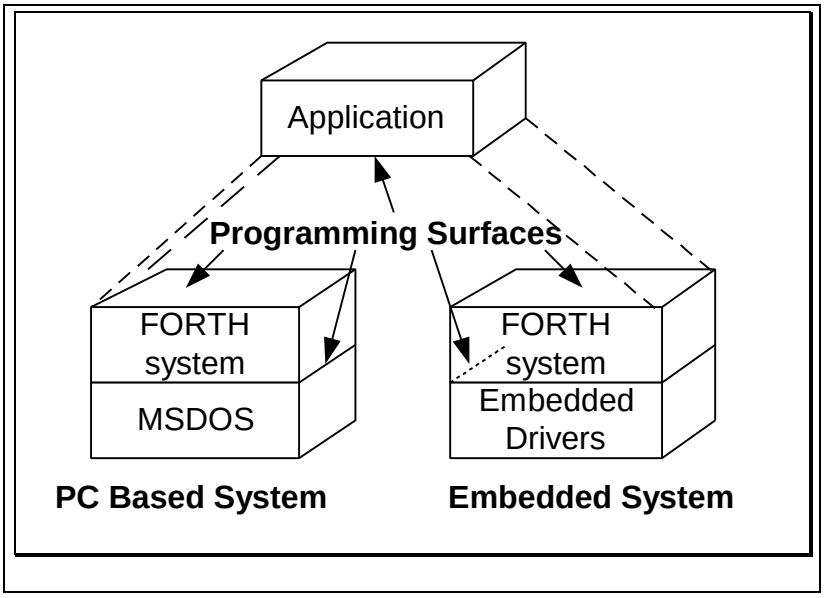

Figure 1 - Programming Surfaces

In the example in figure 1 there are 7 programming surfaces, the low level surfaces the FORTH systems are built on may well be different, although the higher level surface is identical for both systems as the application can sit on either of them.

Pre-Certification of Programming Surfaces

Due to the increased requirements for certification for regulatory approvals, the authors can foresee an increasing need for ensuring that software components - such as cross-compiler cores and kernels, are supplied as fully certified 'objects'. This would allow application builders to concentrate on certifying their own code without the additional effort and cost of certifying the compiler and kernel.

For cross-compilers and kernels it may be possible to provide a simpler certification process based on testing the functions, presented at the programming surfaces, of the kernel words and not doing a full code inspection of the compiler code. Such a technique could cite that the high level FORTH code fully complies with the specified standard behaviour identical for every kernel, and the underlying hidden layers remain uncertified. Such a technique would be quite straightforward.

Summary

This paper has, through the example of a particular project in which the certification techniques were applied (albeit late in the project), shown that certification for a Forth based system, although not trivial, is relatively easy. It has also described the considerations needed at all levels of an organisation involved in producing mission critical systems (especially safety critical ones). The authors have especially highlighted the need for good specifications and early generation of documentation for systems of this genre. They have also indicated the need for a risk-assessment based approach to specification and system development.

That Forth is a system development environment for which certification is a releatively easy task to accomplish (at almost at any stage) does not mean that the systematic issues can be forgotten until the last minute of a project. Indeed, the authors highly recommend the earliest possible efforts in this respect for any project. With an adequate means of software module identification, description and certification to hand, the component library can become a more imminent reality.

Bibliographic References

[EF97MB] "Forth in Critical Care Environments" by Malcolm Bugler, Malvatronics Ltd., Proceedings of EuroForth 97 (Oxford).

[EF97PEB] "Forth in Safety Critical Systems Configuration and Certification" by Paul E Bennett Transport Control Technology Ltd., Proceedings of EuroForth 97 (Oxford).

SSS98PEB "Small Modules as Configuration Items in Certified Safety Critical Systems" by Paul E. Bennett, Transport Control technology Ltd. in "Industrial Perspectives of Safety-critical Systems; Proceedings of the Sixth Safety-critical Systems Symposium, Birmingham 1998" published by Springer-Verlag. ISBN 3-540-76189-6.

[ISOstd1] ISO/IEC 15145:1997 "Information technology - Programming languages - Forth" published by International Standards Organisation. (reference number ISO/IEC 15145:1997(E))

[ENMedi1] EN550601-1-4 "Programmable Medical Devices"

[IECstd11] IEC61508 (formerly IEC1508) "Programmable Electronic Systems; Functional Safety"

[DefStd] Def-Std 00-56 "Hazard Analysis and Safety Classification of the Computer and Programmable Electronic Elements of Defence Equipment.

[Fcoding] Forth Coding Standards by Paul E. Bennett. These coding standards have been given to the public domain and are freely available at http://www.forth.org/.

Note: The following page contains a blank form that was used for the Forth based software certification described in this paper. Two document pages were added as frontispeices for a group of such forms, with each group of forms fully covering a clearly delineated software module (usually a source file).

4th Code Description & Verification

Original Design	
Organisation	
Date	Sht of
	Code Issue

Word/Module Requirements Description (Concise Glossary Entry)		
Input Stack(s)	Output Stack(s)	
Word/Module Definition		
Code Inspection:	Function Test:	Limits Test:
Comments:		