Generating Evidence for Certification of Modern Processors for use in Safety-Critical Systems

Iain Bate, Philippa Conmy, John McDermid
Department of Computer Science
University of York, York, YO10 5DD, UK.

{ijb, philippa, jam}@cs.york.ac.uk

Abstract

This paper investigates the implications of using a modern super-scalar processor in the safety-critical domain. Firstly, a description of current certification practice and devices is given as background. This is followed by an assessment of how the certification argument and its supporting evidence are affected by the use of a super-scalar processor. Two types of modern processor are considered, a Commercial Off The Shelf (COTS) processor and a purpose designed bespoke device. The respective benefits and drawbacks of both are examined. We then identify some key areas where change in current certification practice is necessary to allow for modern processors.

1 Introduction

Safety-critical systems typically use well-established, well-understood scalar processors such as the Motorola 680x0 range. However, this generation of processor is generally no longer in production or is incapable of meeting performance requirements, leaving companies that build safety-critical systems with limited options for new development. One solution to this problem is to make a lifetime buy of the remaining processors and keep them in storage. However, the supply could be exhausted through use in production runs and also through storage degradation. In addition, there are difficulties in preserving the development environment over, say, a thirty-year life span (such longevity is not unusual in the aerospace domain). Although this approach is currently used in practice, it is increasingly being called into question.

An alternative solution is to design and manufacture a purpose built processor, so that in the future the changing commercial market does not affect the company. This solution would necessitate the development of bespoke support tools, e.g. a compiler. A cheaper alternative is to use a COTS super-scalar processor such as the PowerPC.

The latter two options are considered in this paper in order to determine their relative benefits and drawbacks, and their effect on the certification process. Although the two types of processor would have many similar features, such as a pipeline and cache, a COTS processor is likely to be built for optimum average case performance. In contrast, the bespoke processor would be designed to ease the certification process (e.g. to ensure predictability). Typical types of evidence required for certification are worst-case execution time of instructions, hardware reliability and information on systematic design flaws in the processor. We consider all these issues but, for brevity, focus on timing behaviour.

This paper is structured in the following way. Section 2 gives some background on current systems and their certification process, focusing on the aerospace sector. Section 3 describes typical features of modern processors and the implications of designing and using them in the safety-critical domain. In section 4 consideration is given to certification of a system using a modern processor. Section 5 examines some of the differences between bespoke and COTS solutions. Finally, in Section 6 we present our conclusions.

2 Background on Current Systems Including their Certification Process

2.1 General Features of Currently Used Processors

This section describes some basic features of processors currently found in safety critical systems.

2.1.1 Execution of Instructions

The execution of instructions on a scalar processor takes place one at a time and in the same order as the instructions appear in object code. The processor may have multiple units (e.g. load/store units and floating point units) that combine to provide the necessary functionality, but these are never used concurrently. The execution strategy is therefore simple to understand and analyse. This is

particularly useful when performing timing analysis. No matter what type of scheduling mechanism is used, all forms of timing analysis depend on knowing the maximum time a piece of software takes to execute. This is usually known as the Worst-Case Execution Time (WCET). WCET analysis can be performed by splitting software into blocks, where a block is a sequence of instructions that has only one branch at the start or at the end. Every feasible path through the blocks is then examined in order to find the longest [1]. The length is determined by simply adding up the WCET of each individual instruction.

2.1.2 Memory Access

The principal storage area for data and instructions is main memory. This is often significantly slower than the processor. Although faster memory is available, this is expensive and it is only practical to use it where small quantities are sufficient. Thus, some processors utilise cache memory, which is relatively small and fast, as temporary storage. A proportion of the memory accesses will be made from the cache, rather than accessing the slower main memory for each instruction. A memory manager is used which attempts to optimise the contents of the cache memory for maximum throughput. However, using a cache memory increases the complexity of analysis. In particular:

- there are greater variations in the execution times of the software [10],
- it is harder to deduce the actual WCET of the software as the analysis would have to include a model of the cache mechanism. It could be assumed that all memory accesses result in a cache miss but this gives results much greater than the actual WCET, leading to wasted resources [10],
- the more complex the processor is, the harder it is to validate the models used and the higher the likelihood of systematic design errors [12].

Therefore, if the processor has cache memory it is often switched off for safety critical use. However, if it is used then it is assumed that each memory access results in a cache miss. The longest execution path is then intuitively the worst-case.

2.2 Guidance for Gathering Evidence for Processors from Safety Standards and Reports

This section looks at a number of safety standards and guidance documents, examining the types of analysis and arguments relating to the processor.

2.2.1 UK Defence Standards

The UK defence standards (referred to as DEF-STANs) give requirements and guidance for the development of safety critical systems. DEF-STAN 00-54 [3] discusses Safety Related Electronic Hardware and is similar in nature to the software standard, DEF-STAN 00-55 [4]. DEF-STAN 00-54 requires a safety case for the system that “presents a readable justification that the hardware is safe in its specified context”. The context of a processor includes both the application software and its physical operating environment. Discussion of operating environment limits and conditions is outside the scope of this paper, but key issues include temperature, pressure and humidity ranges.

Assurance of the probability of random hardware failures is required, but can usually be derived from historical data for other devices made in the same technology. Knowledge of systematic design flaws is also required. This may be derived from use of the processor. However, this evidence is only deemed credible if it is generated from usage in a similar system. This is one reason why the same type of processor is often found in many different safety critical systems.

As part of the system safety case, justification must be made for the use of a COTS component such as a processor. An example justification could be that the cost of designing and manufacturing a bespoke processor outweighs the gain in integrity over the use of a COTS processor.

DEF-STAN 00-55 requires both static code analysis and software testing as part of the system safety case. Static software analysis examines those properties of the code which can be determined prior to execution. Results of code analysis that are affected by the processor are:

- the maximum execution time taken and the amount of memory required by the software should be bounded and statically determined,
- the software should be tolerant to and respond to random failures of the hardware,
- the system should have built-in tolerance to avoid overload and still be capable of running in a degraded state, and
- any direct access from software to the hardware needs to be analysed.

Software testing must include running the software on the target hardware. Parts of the testing may be undertaken using an emulator of the hardware, but only if its correctness can be demonstrated. Testing should be used to confirm the results of static analysis. To confirm the accuracy of software results, any calculations which rely on processor units, e.g. floating point, should

be tested. It is noted specifically that the use of floating point co-processors introduces the need for additional validation.

2.2.2 Civil Avionics Guidance

The civil avionics arena uses guidance documents rather than standards although, in practice, it would be difficult to get approval for systems which did not follow the guidance. The relevant guidance documents are DO178B [5] for software and DO254 [6] for hardware.

DO-178B requires the developers to produce an accomplishment summary identifying the evidence that the software meets its requirements (this is analogous to the DEF STANs idea of a safety case). It places much greater emphasis on testing and human review than DEF STAN 00-55. The summary includes evidence that the software is compatible with the hardware – which implicitly includes timing. It also requires the gathering of certification evidence through hardware/software integration testing. The testing should demonstrate that high-level requirements are met for software running on the target hardware, including timing requirements where these are significant.

DO-178B considers the use of multiple dissimilar processors with dissimilar software. It notes that some of the required hardware evidence may be replaced with evidence that equivalent output and performance is achieved by both systems. Evidence of hardware failures can also be reduced. For example, if the processors are dissimilar in design it can be argued that the likelihood of simultaneous failure is lowered, reducing the amount of failure rate evidence needed. For example the Boeing 777 Primary Flight Control System [7] uses different types of processors in each of three computing channels, with cross lane monitoring between each channel.

DO254 has only just been issued, and we are not aware of industrial experience in using these guidelines. In our experience an accomplishment summary similar to that used for software has been produced[1] for bespoke hardware.

2.2.3 Other Standards and Guidance

There are many other standards for software and hardware in safety critical systems, including IEC 61508 [8]. This is an international “meta-standard” from which sector specific standards are meant to be developed.

Analysis of the published standards, for example see [9] suggests that the underlying principles in the majority of the standards are the same, although there is significant variation in the details. However few of the standards deal with the sorts of issues identified here.

One useful source of advice is the Ada HRG working group report [2] summarises the code analysis techniques needed for software in high integrity systems. The report is mainly concerned with how this analysis is affected by the use of Ada. However, the design of the processor affects the manner in which analyses are performed and the results obtained. The analyses are discussed in more detail in Section 4.2 and in Table 1, but include both static analysis techniques and dynamic testing of software.

3 Key Features of a Super-Scalar Processor

The principal influence on processor development has been the general-purpose computing market (e.g. personal computers, mobile ‘phones, etc), driven by market forces and economy of scale. The entire production run of a safety-critical system may use a few thousand processors, whereas a successful mobile ‘phone will use millions. In general, non-safety-critical systems are more concerned with good average-case performance, even if this is at the expense of poor but rarely encountered worst-case performance. This has led to dramatic increases in the speed of processing compared to memory speed (over 20 years, a factor of 100,000 for processing times compared to 10 for memory). Consequently, there are increasing numbers of wait states inserted during instruction execution whilst waiting for memory. A wait state is the term used to describe the condition when processing is halted for a clock cycle whilst waiting for an event to occur, e.g. for a memory access to complete.

The new features of modern processors are described in the following subsections. It is assumed that both a COTS and bespoke processor have similar features, but that these may be implemented in different ways.

3.1 Pipeline

Figure 1 illustrates the pipeline structure for the PowerPC 603e, a typical example of a modern processor. The figure shows that the execution of instructions is split up into a number of stages and that some of these stages (principally the execute stage) have multiple units to support concurrent execution of instructions. This approach allows multiple instructions to be handled simultaneously and means the processor can do something productive if the currently executing instruction has been delayed (e.g. from a cache miss).

[1] The airworthiness authorities suggest using this approach where exhaustive analysis is not possible.

The following list contains some features of the pipeline mechanism. It should be noted that these features cannot simply be “turned off” and the programmer has little influence over their operation.

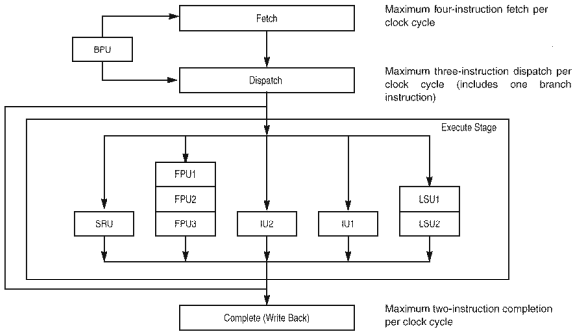

Figure 1 - The Pipeline Structure of a Typical Modern Processor - The PowerPC

1. *Multiple Issue* - The ability to fetch, dispatch and complete a number of instructions per clock cycle.
2. *Parallel Execution* - From Figure 1 it can be seen that there are four types of processing unit in the execute stage; floating point, system register, load/store and integer (possibly multiple).
3. *Out of Order Execution* – This can prevent the pipeline stalling when an instruction takes a long time to execute and/or there has been a cache miss.
4. *Speculative Execution* – The pipeline mechanism does not always wait for the result of a comparison before processing the associated branch instructions (and subsequent instructions after the branch folding or falling through). Instead the pipeline mechanism guesses which branch is the likeliest to be taken using branch prediction.

To simplify the design and verification of a bespoke processor, and to ease the analysis of the software that executes on it, the bespoke design would probably only use a subset of these features, such as parallel execution and multiple issue.

3.2 Cache

Due to the disparity in speed between processing and memory, modern processors have become more reliant on cache memory and some processors have two levels of cache. The first level of cache is a small but very fast “primary” cache on the actual processor (typically with zero wait states). The other is a larger “secondary cache” which has more wait states than the primary cache, but less than main memory. The methods for optimising the contents of, and access to, the cache have also changed significantly. One complication of having cache is determining the impact of an interrupt / pre-emption on the cache contents and hence on the program flow. To simplify the design and verification process, a bespoke design would probably only use a primary cache and its contents may be statically defined.

4 How the Change to a Modern Processor Affects System Verification and Design

The following section examines the changes that a move to a modern generation of processor makes to system verification and design. This section is encapsulated in Table 1.

4.1 Hardware Issues

4.1.1 Errors in the Processor Design

For the older generation of processors it was reasonable to assume that the majority of design flaws had been found and documented e.g. for widely used devices such as Motorola's 68020. Even if these errors had not been fixed, design engineers could tailor their system to avoid them. However, a modern processor has a significantly larger and more complex design – 28 million transistors on the Pentium III in 1999 versus 68 thousand transistors on the 68020 in 1979. This increase makes it less likely that all the design errors have been found and documented, although certain problems have been well publicised (e.g. the problems with the Pentium's arithmetic unit [16]). DEF STAN 00-54 notes that processors may have several different variants in a year, each of which may contain subtly different design features and flaws. It can be confidently assumed that a COTS product would not comply with the DEF STAN’s requirements. However, the argument for using a COTS processor can be given weight by using a supplier with a reputation for well-designed products. In this case the given processor would have to adhere to the suppliers usual standards of quality. In any case the certification argument would have to assume the existence of known and unknown errors.

The use of dissimilar processors within a system can reduce the amount of evidence required regarding design errors in each. However, the argument will only be convincing if the processors are sufficiently different in design to avoid flaws being replicated in the same place. When dissimilar processors are used a separate compiler has to be used for each. Whilst it can be argued that this means that systematic failures are unlikely to occur in the same place it could also be argued that the frequency of failure across all processors may increase. This option has the significant disadvantage that fault tolerance is more difficult since processors cannot be run in lockstep.

If a function within the processor is known to

have a design flaw then it may be desirable to ensure it is not used, validating this through object-code inspection. As verification of object code against source code is often currently undertaken by visual inspection, this becomes a more complex process, requiring specific knowledge of the processor hardware. In order to ease inspection it might be desirable to use a programming language sub-set, designed to prevent the use of a particular feature if is deemed to be unsafe, and/or difficult to analyse. Other alternatives would be to adapt the compiler to avoid the feature or modify the object code after compilation.

4.2 Software Evidence

A selection of the software verification techniques described in the Ada HRG working group report is considered in this section. A complete assessment is given in Table 1.

4.2.1 Worst-Case Execution Time Analysis

The timing analysis of the system, and WCET analysis, becomes a much more complex task for a pipelined processor. This is a crucial activity, as most safety critical systems are dependent on real-time deadlines being guaranteed as met. It has already been stated, in section 3.1, that ignoring features of the processor when calculating WCET leads to pessimistic results. An alternative to WCET analysis is to extensively test the system to obtain a measured value for timing. For lower integrity systems this may be an acceptable solution. However, for the highest integrity level it is necessary to perform analysis in order to guarantee timeliness. Simply testing the system cannot guarantee the absence of timing overruns [10].

Developing WCET analysis for modern processors is complicated as it relies on accurate models for the types of features discussed in section 3. However, producing a completely accurate model may not be possible due to the lack of information that can be verified. Verification is difficult because most of the processor's mechanisms are not externally observable. In our experience, processor manuals do contain errors that would easily lead to incorrect analysis. Accurate models also have limited portability. In fact producing a software model for WCET analysis that completely emulates a processor would simply take too great a time to obtain results for any reasonable size of code [10]. However, a simple model may be hard to validate because it is sufficiently different to the actual processor.

To further complicate the issue, not accurately modelling all the processor's features may introduce anomalies [11] and unmanageable pessimism into the results. An example of an anomaly is illustrated in Figure 2 where a cache hit results in a longer overall execution time than a cache miss i.e. the opposite of what is expected. Figure 2 shows two instructions that are dispatched simultaneously where instruction (*A*) that is due to execute first has a shorter execution time than the other instruction (*B*), both instructions use different execute units (e.g. FPU and IU) that can be used concurrently. If *A* suffers a cache miss then it results in *B* being executed before it, i.e. out of order. In this case, both instructions complete earlier than if they instruction *A* had not suffered a cache miss and they had been executed in their original order. Therefore the WCET analysis model needs to take into account potential anomalies. Otherwise, all possible paths would have to be analysed rather than those that could be expected to be the worst-case. Again for realistic software sizes, this would be intractable. Hence a balance must be struck between the pessimism of the results, the computational complexity of obtaining results and the difficulty in validating the model.

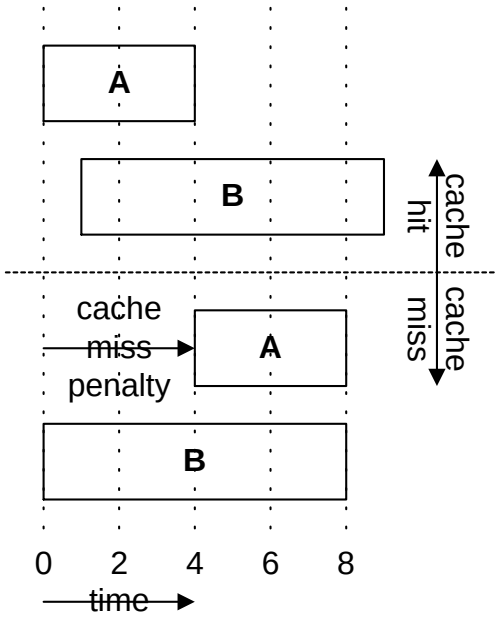

Figure 2 - Example of a Timing Anomaly

There is already a great deal of analysis available including [7, 9], and Engblom provides a useful survey of these analyses in [12]. These approaches can be tailored for our use by taking advantage of the fact that software is written in a particularly disciplined way for the safety critical domain. For example, the software is written to make control flow analysis easier to perform, and/or conforms to a restricted subset (e g. SPARK Ada [14]). Such assumptions allow some simplification in the way that the analysis is performed.

Some of fundamental issues for WCET analysis of a super-scalar processor are summarised here:

- *Re-targetable Analysis* - Producing analysis in a generic fashion means that it can be instantiated for a particular platform with a minimal amount of rework. To achieve this aim a WCET analysis language has been defined at York which allows the WCET analysis software to parse platform-specific information and hence to customise the analysis. For example, the number of cache lines can be defined, as can the number of sets the cache is organised into, and so on.
- *Validity of the Analysis* – Producing a valid WCET is reliant on correct information from the manufacturer. It can be assumed that a COTS processor has not been produced or documented to the standard required for the certification of the system. Assuming that the processor does not need to be re-engineered, a validation strategy is needed. Some work has been performed on this subject [13].
- *Anomalies within the Analysis* - The analysis has to allow for the effect of cache misses and branch predictions without leading to anomalies which understate the worst case.
- *Data Cache Analysis* - Data and instruction cache analysis mechanisms are similar. The key difference being that instructions always have a static location in memory whereas data does not. Examples of data accesses that do not have a static location include; pointers and stack variables. Data cache analysis is the subject of future work.

A suitable strategy for WCET analysis is to produce a simplified model of a modern processor that can easily be tailored to a specific processor. This accepts some pessimism whilst guaranteeing that the results are safe, e.g. the predicted WCET is greater than the actual WCET.

4.2.2 Other Issues

The accuracy of calculation units, such as floating point, needs to be demonstrated, e.g. when rounding floating point variables. Hardware validation could be used to demonstrate the processor correctness.

Control flow analysis is used to ensure that software is executed in the correct order, whilst data flow analysis should ensure variables are not used prior to being set a value. The analysis is normally performed at the source code level based on the assumption that the software semantics are preserved when executing on the hardware platform. A limited amount of flow analysis is performed at the object code level to show the results are consistent with the software executing on the actual processor. When moving to a modern processor further object level verification may be needed to deal with the advanced features of the processor. This all adds to cost. Also hardware validation needs to be included and has to show that the results are consistent within the processor itself and at its outputs.

4.3 Changes in System Design

4.3.1 The Use of Memory Partitioning

In order to take full advantage of fast execution times and expanded memory it is desirable to share processing resources between more than one application. Sharing memory requires a partitioning mechanism that can guarantee that a number of applications (possibly with multiple criticality levels) can share the same memory resource without risk of data corruption [18]. This involves the detection of attempted memory violations by an application. The Memory Management Unit (MMU) in many processors provides functionality to create interrupts when an illegal memory access occurs. To use this feature it would need to be demonstrated that the MMU would reliably provide interrupts when detecting an attempted violation and that the system software could deal with these interrupts.

4.3.2 Designing Systems to be More Portable

To ease the problems of porting software to different platforms, an intermediate language such as ANDF (Architecture Neutral Definition Format) can be used [14]. This approach compiles application source code into ANDF rather than to object code. The ANDF code is then translated for the target hardware using an installer. The advantages are that analysis of the software can be performed at the ANDF level rather than at the object code level, thus porting software to another platform only needs a new installer to be produced. However, the compilation of application code to ANDF would need to be trusted if the analysis were to be conducted at the ANDF level [15]. Using a specified sub-set of ANDF can also ensure features of the processor are avoided which are not trusted.

Increasing the portability of software doesn't ease the difficulty of validating a modern processor, but does help solve the problem of hardware obsolescence.

5 How the Change to a Bespoke Processor Affects the System Design and Certification

This section explores the ways in which a bespoke processor design could differ from a COTS processor. The observations are again summarised in Table 1.

5.1 Hardware Issues

5.1.1 Errors in the Processor Design

One of the key areas of concern with a bespoke processor is that of design errors. It is often assumed that any bespoke processor proposed for safety-critical systems has been designed to more rigorous standards and that the design is simpler. Whilst the latter assumption is almost certainly true, the same cannot be said of the rigorous standards assumption. A hardware manufacturer, such as Intel, has a great deal invested in the correctness of their hardware including the value of their reputation not to mention the cost of mistakes, particularly when they lead to product recalls. Since the Pentium floating point arithmetic error was uncovered, the importance of hardware correctness has received a higher profile. Significant effort has been applied to formally proving parts of the design [17].

It is arguable that the use of a bespoke processor would increase the need for hardware testing. Whilst a COTS processor may not have had all possible flaws detected, a bespoke processor would have had significantly lower operational usage which means there is an even greater likelihood that design faults have not been found.

A further complication to the process is that the designer of the bespoke processor may not have a track record in the design of complex micro-electronics. This, along with the lack of operational usage may necessitate a greater level of certification evidence to be generated.

Upon first considerations, it is not apparent that dissimilar processors would be employed when the system uses bespoke processors. However, by initially using the bespoke processor in combination with COTS processors, confidence can be gained in the bespoke processor's integrity and also integrity of the overall system design.

5.2 Software Evidence

5.2.1 Worst-Case Execution Time Analysis

The use of a bespoke processor would mean that the processor model used during WCET analysis could be made significantly simpler than for a COTS processor. The processor could be designed without features such as speculative and out-of-order execution. This could also eliminate anomalies. Assessing the WCET would then be within the capability of existing tools.

5.3 Changes in System Design

5.3.1 Memory Partitioning

The use of a bespoke compiler and processor could make the provision of memory protection significantly easier as:

- the use of scoping of rules of the high-level language, combined with confidence that these are enforced within the compiler, would mean that the impact of systematic design errors could be contained, and
- various tactics for protection against random hardware failures could be employed, such as storing each variable in two sufficiently separated memory locations.

5.3.2 Bespoke Compiler

A key consideration to this approach is that a bespoke processor also necessitates the production of bespoke compilers and tools to support the software production process. Many of the issues discussed in this section, such as lack of track record and ensuring the integrity of a new product, apply equally to these. A major concern is that the first-time effort needed to produce the processor and all the necessary support could be greater than for the actual safety-critical system itself. However, the compiler could be designed to simplify analysis, e.g. not using features such as optimisation and multi-platform support. This would simplify many of the analysis problems encountered with a COTS processor.

Further simplification in analysis could be achieved if the compiler provided suitable outputs, e.g. information concerning mutually exclusive paths and loop bounds that are helpful during WCET analysis. Other information that could be preserved for use during this analysis is data flow and control flow information as well as information from the symbol table. The compiler could also enforce static memory locations for data items (i.e. local variables would have a fixed location and a stack would not be used) which would make data cache analysis significantly simpler.

When designing the compiler, particularly the back-end, design decisions could be made that would not concentrate on optimum average-case performance but instead on good worst-case performance as well as easing object-code verification.

6 Conclusions

To summarise, the following list indicates some of the key areas of change a super-scalar processor would make to current certification practice:

1. *Calculation of WCET* – this analysis is the most significantly affected. COTS processors are designed with average case performance in mind, at the expense of worst-case performance and predictability. The validity of

complex pipeline models for the analysis is questionable but simplification leads to overly pessimistic results.

2. *Flow analyses (Data/Control)* – pipeline features such as out of order execution affects the results of this analysis. Validation is needed to ensure this cannot cause unpredictable results.

3. *Object code analysis* – this becomes more complex and requires specialist knowledge of the hardware to avoid, for example, the use of a hardware feature with a known design flaw. The use of programming language sub-sets may ease this analysis.

4. *Hardware failure rates and design flaws* – the increased complexity of the processor and the number of different designs and can lead to greater incidence of failure. It is also more difficult to provide evidence of integrity from usage in similar systems; for example, the integrity of cache, pipeline and registers has to be shown in the event of interrupts.

5. *Shared resource usage* – to take advantage of the increase in processing power it is desirable to share resources between multi-criticality components. For this, evidence is required for partitioning mechanisms e.g. using the MMU.

When considering the choice of bespoke and COTS processor, there are clear benefits in both. The bespoke processor approach means that obsolescence is now under the system designer's control rather than external suppliers, and the processor can be designed to ease certification. However the cost of development may be high and, the lack of operational usage would make arguing integrity more difficult. The advantage of the COTS approach is cost – ignoring the high cost of first-time use. However, systems are still susceptible to obsolescence. It is also more difficult to demonstrate the integrity of the resultant system due to the lack of processor design information and the difficulty in predicting performance.

7 References

[1] *Static Timing Analysis and Program Proof*, R. Chapman, Department of Computer Science, University of York, PhD Thesis, YCST-95-05, 1995.

[2] *Guide for the Use of the Ada Programming Language in High Integrity Systems*, ISO/IEC PDTR 15952, 1998.

[3] *Requirements for Safety Related Electronic Hardware in Defence Equipment*, Ministry of Defence, UK, 00-54, March 1999.

[4] *Requirements for Safety Related Software in Defence Equipment*, Ministry of Defence, UK, 00-55, August 1997.

[5] *Software Considerations in Airborne Systems and Equipment Certification*, RTCA/EUROCAE, DO-178B/ED-12B, December 1992.

[6] *Design Assurance Guidelines for Airborne Electronic Hardware*, RTCA/EUROCAE, DO-254B, 2000.

[7] *Dependability of the 777 Primary Flight Control System*, Y.C. (Bob) Yeh, Boeing Commercial Airplane Group, USA, 5th IFIP Conference on Dependable Computing for Critical Applications. IFIP, 1995.

[8] *Functional safety of electrical/electronic/ programmable electronic safety related systems*, International Electrotechnical Commission, IEC 61508, Draft, 1998.

[9] *The Potential for a generic Approach to the Certification of Safety-Critical Systems in the Transportation Sector*, Y. Papadopoulos, J McDermid, Reliability Engineering and System Safety, 63(1), pp. 47-66, 1999.

[10] *Static Cache Simulations and its Applications*, F. Mueller, Department of Computer Science, Florida State, PhD Thesis, 1994.

[11] *Timing Anomalies in Dynamically Scheduled Microprocessors*, T. Lundqvist, P. Stenström, Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99), pp. 12-21, December 1999.

[12] *Worst-Case Execution Time Analysis for Optimized Code*, J. Engblom, MSc Thesis Uppsala University, DoCS 97/94, September 1997. 104 pages.

[13] *Effectiveness Evaluation of the Buffer-Oriented Microarchitecture Validation*, N. Utamaphethai, R.D. Blanton, J. P. Shen, Proceedings of the Second International Workshop on Microprocessor Test and Verification, Atlantic City, September 1999

[14] *TDF Specification*, I. E. Currie, Defence Evaluation Research Agency, United Kingdom, Version 4, DRA/CIS(SE2)/CR/94/36/4.0, June 1995.

[15] *Portable Code for Complex Systems*, N. Audsley, I. Bate, A. Grigg, Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, pp. 111-119, December 1999.

[16] *Statistical Analysis of Floating Point Flaw*, Intel White Paper, November 1994

[17] *Formally Verifying IEEE Compliance of Floating Point Hardware*, Intel Technology Journal, 1st Quarter, 1999

[18] *Assessing the Safety of Integrity Level Partitioning in Software*, John A McDermid and David J Pumfrey, Proceedings of the Eighth Safety-critical Systems Symposium, Southampton, UK, 2000

Type of Evidence	*Level of analysis*	*Information being gathered/ Purpose*	*Method for collection*	*Affect of transition to a COTS processor*	*Affect of transition to a bespoke processor*
Static Analysis					
Control Flow	Source Code	Source code is examined for sequence of execution, any unreachable code, and termination of loops/recursion etc.	Static Analysis tools e.g SPARK Ada Examiner. Unaffected by scalar processor.	Some object code verification may be required to examine pipelined execution of code	Some object code verification may be required to examine pipelined execution of code
Data Flow	Source Code	Source code is examined to ensure no variables are used before they have been set a value	Static Analysis tools e.g SPARK Ada Examiner. Unaffected by scalar processor.	Some object code verification may be required to examine pipelined execution of code	Some object code verification may be required to examine pipelined execution of code
Range Checking	Source code	Code is examined for overflow / underflow, rounding errors, array bounds, and data ranges	Static analysis tools e.g SPARK Ada Examiner. Unlikely to depend on hardware as FPU can be switched off	The code may be dependent on specific hardware e.g. for floating point calculation, the accuracy of this needs to be checked for the application	The code may be dependent on specific hardware e.g. for floating point calculation, the accuracy of this needs to be checked for the application
Stack Usage Analysis	Object Code	Examines the size of stack required, to ensure no stack/heap collision	Static analysis tools e.g SPARK Ada Examiner	Stack usage is affected by the MMU and data caching mechanisms which should be verified	The bespoke compiler and processor could be designed without dynamic memory allocation so that the stack is only used where necessary.
Timing Analysis	Object Code	Calculation of Worse Case Execution Time	Simple addition of object code instruction execution times	A complex model should be built for the processor with branch prediction, cache and pipelining, this can be used to calculate WCET	The processor could be designed to simplify the model needed for WCET analysis and the compiler could provide support for WCET analysis. The problem of model validation would be significantly reduced.
Shared Resource Analysis (OMU)	Object Code	To demonstrate lack of interference between object code components when accessing a shared resource	Static analysis tools e.g SPARK Ada Examiner	May be partially dependent on hardware enforcement to prevent interference e.g. MMU, therefore the HW software interactions must be modelled	Hardware/compiler techniques could be employed to simplify the verification of correct use of shared resources.
Object Code Analysis	Object Code	Compare source code to object code to ensure correct translation	Visual inspection of source code and object code	This may become more complex if object code makes specific use of hardware feature	Having a bespoke compiler would significantly reduce the analysis that needs to be performed on object code.

Software Testing					
Requirements Based Testing	Object Code / Hardware	To ensure execution is compatible with high-level requirements	Executing object code, including execution on target platform or on verified emulator with test cases generated from requirements	More of the testing may need to be performed on the target platform due to increased hardware dependencies and difficulty in producing a verified emulator	More of the testing may need to be performed on the target platform due to lack of operation usage for the processor.
Hardware/ Software Interactions for Interrupts etc.	Object Code / Hardware	To ensure system is robust in presence of errors etc. and still meets high-level requirements	Executing object code, including execution on target platform or on verified emulator with test cases designed to enduce faults	More of the testing may need to be performed on the target platform due to increased hardware dependencies and difficulty in producing a verified emulator	More of the testing may need to be performed on the target platform due to lack of operation usage for the processor.
Hardware testing					
Operating Environment	Hardware	To examine the performance of processor in context e.g. for tolerance to temperature ranges	By testing processor in given environment or by using a specially designed processor for specific operating environment	COTS processor may not be built for appropriate operating environment and must be tested	Bespoke processor can be fabricated for the specific operating environment.
Operating Design Limits	Hardware/ Object Code	To examine the performance of processor in context e.g. for overload to ensure meets high-level requirements	Using information from manufacturers and dynamic testing	Unchanged, although the amount of evidence may increase due to increase in complexity of the processor	Unchanged, although the amount of evidence may increase due lack of operation usage for the processor.
Hardware Failure Rates	Hardware	To gather probabilities per fixed period of a hardware failure	Evidence gathered from thorough usage of the processor in similar situations	Unchanged, but maybe less convincing due to larger number of chip components	Unchanged, but maybe less convincing due to larger number of chip components and lack of operational usage.
Systematic Design Flaws	Hardware	To gather information on systematic design flaws to compensate for them in complete system	Evidence could be gathered from thorough usage of the processor, and if specially designed by formal proof of the design	Evidence is viewed as less convincing (and incomplete) due to slight differences in families of design and increasing complexity of processor	Evidence is viewed as less convincing (and incomplete) due to lack of operation usage for the processor.

Table 1 – Examination of Effect on Evidence Gathering of Move to a Modern Processor