How do we analyse, verify and evidence that we can make better use of modern processing infrastructure for safety-critical real-time?

Context

The high-integrity systems we deploy today have to demonstrate sufficient predictability to ensure the outcomes are safe, yet as we move towards more software intensive electronic system solutions the fabric of the underlying systems continues to frustrate our ability to provide that evidence.

Typically the performance enhancing mechanisms such as deep pipelines and cache mechanisms in microprocessors and microcontrollers, or at their memory interfaces, whilst not truly non-deterministic, are sufficiently complex because of previous state or sequence information which, without significant contrivance, cannot be reasonably established a priori. Without these mechanisms the electronic system is deterministic (i.e. no timing variance or distribution other than that from system timing accuracy) but its performance, notably in Memory to CPU transfers, is significantly degraded and its available throughput performance is not realised.

The set of outcomes that are possible from the wide range of prior state or sequence information is however, able to be accurately defined (deterministic to a CPU clock cycle level), although the temporal spread of those solutions may be significant. The component of this spread due to system timing accuracy (i.e. CPU/Memory/Peripheral clock) is likely to be negligible.

It is the mismatch of the need for software execution performance (i.e. using the enhancement mechanisms of the fabric) and the need for a sufficiently minimal spread of temporal variation to ensure sufficiently predictable system performance that defines the challenge.

In the limit, when computational performance (timing resolution and task execution time) is several orders of magnitude greater than system temporal performance demands, the variance in execution, with regards its temporal distribution, becomes negligible (or can be considered ‘instantaneous’), in that the system is indifferent to that level of timing granularity and resolution, and it can be ignored.

For any given design, the system intolerance to the spread of execution time becomes increasingly apparent as processing load increases, as both task execution latency (the queue before it gets to execute) and execution speed affect the ‘completion’ deadline of a given task.

Our appetite for ever more complex control, data, or faster response leads us to consume more computational performance, both at the outset of a product design, but also through-life enhancements that increase functionality or efficiency.

The electronic system mitigating response is to provide us with more complex System on Chip designs which have dedicated hardware (e.g. Dynamic Memory Access (DMA), Analogue-to-Digital conversion with periodic sampling and various integral filters) or additional processing units (e.g. Intelligent communication peripherals that assemble serial streams and deal with lower levels of protocol, queuing, error detection and correction, re-transmission) to ‘offload’ work from the system application control processing.

Questions

Sufficiently predictable systems can be easily demonstrated if the system application is tolerant to (the entire) spread of Worst Case Execution Times (WCET) (irrespective of their frequency), but what happens if the system application is tolerant to the entire spread of WCET assuming their frequency of distribution?

How do you characterise the frequency of distribution of WCET – as it is application specific, context dependent? (Scenario: ” What do you want me to measure, and under what circumstances?”)

Is predictably of a system sufficient if the probability of WCET to which the system is intolerant, is sufficiently low? (Scenario: “Yeah, I know it’s a problem, but it doesn’t happen very often!”)

Can we improve predictability of the overall system by ensuring the system application is tolerant to “infrequent” excursions of individual WCET to which it would be habitually intolerant? (e.g. Control systems that were tolerant to inputs/data samples that were missing, or late, on a (statistically) infrequent basis).(Scenario:“...and, if on occasion, we get a bad input, the system is filtering with this really long time constant, over thousands of samples and applies a maximum excursion rate limit, so it doesn’t really disturb the outcome... most get lost in the noise!”)

Is the ‘statistical’ basis of this a problem – in that the circumstances under which a WCET being the ‘abnormal’ case are likely to be associated with ‘abnormal’ system behaviour (e.g. failure conditions, overload), which is likely to be ‘persistent’ (at least at the timing granularity of the system application) and so be a degenerative case? (Scenario: “... it became overloaded due to some extra processing to do with a hardware failure, which took all the usual “fast access” data into “slow access”, which caused the execution times to jump, which caused delays, which caused the inputs to the control system to have significant errors, which caused additional limit and filtering to be triggered that kept the “fast access” data out, which kept the execution times up until the system could no longer cope!”)

Given that we can’t influence the distribution of WCET (or can we?) then how do we define systems that are tolerant to a proportion of the (high-end) of those distributions... and enumerate, verify and validate that they are tolerant to that effect ... predictably?

Can we use alter the distribution of WCET by changing the configuration of the infrastructure (number of cache ways and/or their explicit association with particular data areas/instruction areas)?

How do we instrument and verify/validate the above techniques and measure the outcomes without the instrumentation skewing the results?