...Flexible Systems

For Systems Engineering (not just Control Systems) audiences

Introduction

Mechanical systems design become encumbered very early with constraints regards system coherence and timing – e.g. when physically signalled via a shaft, or coupled through transmitted power, say hydraulically.

The introduction of programmable control systems, with electronic/electrical actuation of those power systems, allows an essentially arbitrary relationship to be defined. This unlocks the door for more flexible trades to the system design.

The reality of course is that although arbitrary relationships are possible, there are constraints that bias the choice of control system solution – e.g. speed of reaction, and ultra-low-power operation may still be better conceived in electronic hardware than software, whose processing time and consumption may disadvantage it.

Additionally, the move to software solutions is not without its own set of costs, for example security. Using programmable systems potentially exposes an opportunity for a security threat. Such exploits may change over time, so that the design may need constant re-assessment (and possibly investment) to deal with ever changing threats.

Flexibility Benefits of a programmable system

Most of the benefit derived from programmable control is derived from its (relative) ease of change. That ease of change derives from the fact that the electronic and software components have few 'physical' attributes and therefore the 'trade space' poses less of a constraint when compared with making choices of say a mechanical component that has to consider e.g. temperature, stress and load bearing, hardness, manufacturability type criteria.

At some point, the programmable system has to interact with the real world through physical systems (both sensors and actuators) which have these constraints. The physical systems generally have (albeit, in some cases, complex) physical relationships between their properties that define their behaviour. Programmable systems can have largely arbitrary relationships.

[N.B. This 'arbitrary' capability can drive system designers to an over-simplistic (naive?) view that Control Systems can be used to mop-up any relationships that they cannot secure through traditional (e.g. mechanical/physical) design and miss the opportunity for the control system to play a much wider role in system optimisation.]

Flexibility Pitfalls of a programmable system

We all too readily blame Software Systems for their complexity. However, we turn to programmable controls systems to solve complexity issues that are untenable in other domains (e.g. a mechanical relationship) because of constraining physical properties (e.g. size, weight, energy). [c.f. Babbage's Difference engine]. In general the complexity is not born of the software, but of the magnitude of problem we are trying to solve.

Given an ability to define arbitrary relationships, it becomes all too readily susceptible to generating multiple, dependent, discontinuous relationships that are incredibly difficult to define with regards resultant behaviour. These relationships may be discontinuous in terms of linearity of response, or temporally. The validation of the resultant 'state-space' (the combinatorial explosion of states possible because of stored variables, input and

output conditions and sequential opportunities) of the control system may quickly become untenable leaving to loss of determinism and possibly loss of predictability of behaviour.

Whereas it is perfectly possible to generate a near error-free implementation of any simple system in software, it is frequently much harder to specify the operational bounds and context of use of that system, meaning it is more likely to be incorrectly specified.

Implementation of software requires a rigorous transform of the specification to implementation, yet frequently too little time is spent defining the system and ensuring an appropriate rigour of partitioning that system to its component technologies. Consider the effect on a modern sky-scraper, if an architect failed to rigorously specify the component structures and materials.

Portfolios of related Variant Product

With flexibility of implementation comes the potential to enable systematic or even automatic reconfiguration of systems. By recognising the breadth of the market as variations of a particular product it is feasible to systems engineer solution from components that can be assembled in a different way, or re-configured to give variant answers, thus satisfying a larger portion of the market... potentially with reduced engineering costs compared to engineering unique solutions for every product instance.

With no 'physical engineering' of the system, software systems lend themselves to these changes of configuration readily.

The major benefits of System composition, once defined and engineered as a product to achieve those goals, are fast construction with defined variation. The benefits of such an approach being:

i) the speed of implementation,

ii) the ability to re-use provenance and reliability information from similarly configured products (or of individual components of the product)

iii) the re-use of validation/verification evidence and safety arguments

The pitfalls of the approach being:

i) an increased initial design cost (having to invest in making it flexible)

ii) a need to accurately predict the market/product scope and its potential evolution

iii) a need to recognise and manage the consequence of common errors (e.g. all systems deploying a faulty component have the potential for common error... but happily also the potential for a common fix (1 investigation!)).

Most live experiences show that 'multiple solution design' rather than for 'single solution design' improves the design rigour sufficiently that the common error modes are significantly reduced.

Responding to Flexible Changes (e.g. Security)

Security, like Safety, requires a high degree of rigour to ensure an error-free implementation. More importantly from a System design perspective is that safety tends to be a response to failure of the 'as deigned' system – which is essentially static for any given implementation.

Security on the other hand, is a response to a continuously evolving threat, both of the physical implementation (e.g. the electronic hardware or structure of software, its data, or its performance) but also in its underlying platform, the connected system and the logical interfaces between components of the system.

The evolutionary nature of such threats means that software is a potentially malleable solution to the need for change to mitigate the threat. Such system design needs to ensure that appropriate flexibility is designed in, yet secured from malicious attempts to alter its purpose.

In general, security systems are about 'buying time' to dissuade a potential threat from investing the effort, or to enable discovery of the attack before any ensuing damage is incurred. Unfortunately a highly defensive attitude is also an advert that there is something worthwhile being protected!

Safety in reconfigurable systems

Safety systems exist to mitigate Loss of life, injury or harm (through designed in product safety, or through restricting the interaction in operational safety).

Most product safety attempts to detect, and accommodate, failures in the system. That accommodation may require the system to be reconfigured to disable, ignore or disconnect the errant component from the system, usually replacing its capability with some alternative, redundant, solution. A characteristic of such systems is that they have redundancy (for hardware failures), diversity (for systematic failures of an implementation type), various failure detection systems and some logic to enable exclusion of components. These solutions are a product of design knowledge of the underlying fabric and its modes of failure. Even today, the strategies for detecting failure and managing components in a failed state often outweigh the nominal functionality by a factor of 4 (80% accommodating faults, 20% delivering functionality).

It is worth a short diversion into the causes of failure. For physical components (e.g. mechanical hardware, or passive electronic components) the defects are likely from manufacture, or life/reliability/stress mechanisms. For programmable systems (whether FPGA or Software) it is more likely from implementation errors, from noise, or transient errors in the underlying fabric (e.g. atomic level changes due to radiation). Residual errors in these solutions are a fact... although the rigour of the process can determine the number and their likelihood of being encountered as failures. For high-integrity solutions therefore we rely on several, layered, defences so that such errors, when encountered, are highly unlikely to escape. Dependable systems are designed so that the user can rely on their ability to continue to supply critical functions even in the event of some loss of facility.

In future, that dependability will need to cross the barrier from safety systems to "safe and secure" systems, as we move to an age where safety is dependent on securing data from the system.

