Safety logic on top of complex hardware software systems utilizing dynamic data types.

Nicholas Mc Guire
Distribued and Embedded Systems Lab
SISE, Lanzhou Unviversity
mcguire@lzu.edu.cn

Abstract

Utilizing computers for safety critical systems, notably contemporary super scalar multi-cores, let alone NUMA systems running general purpose operating systems like GNU/Linux, is quite contended in the safety community - their hopes still rest on determinism and KISS. While keeping things simple in the safety related components is undoubtedly preferred, it is questionable if keeping the hardware model simple is realistic - notable with the divergence of reality from model with respect to determinism already being dramatic for widely used general-purpose single-core CPUs. Further actually deterministically covering the impact of all complex software components is not doable with an economically tolerable effort (if it is technically doable is a different issue).

The consequence of this belief in determinism, is an, in our opinion useless, fight against complexity and non-determinism - two inherent properties of modern hardware/software systems. Quite to the contrary, we propose to utilize the properties of complex systems to enhance safety related systems. This seemingly paradox approach can be seen as an attempt to take the bull by the horns as it seems inevitable that the time of simple CPUs and back-box proprietary operating systems, that continue to entertain the illusion of determinism, is coming to and end.

Safety mechanisms, drawing enhancements from underlying complexity, we see as potentially suitable for building safety related systems are:

- computation: Inherent diversity
- data: mapping value domain to complex data representations
- time: loos coupling: inherent randomnes

and we are quite sur that this little list is incomplete at this point.

In this article we will describe an attempt at the second category called dynamic data types, which essentially combine the value domain with the temporal properties of data to map data to a value in the frequency domain rather than to a value in the time-domain. We outline the concept of dynamic data types and a rational for why it seems a promising approach for covering of particular fault classes. Finally we describe how building simple logic utilizing dynamic data types on complex systems can yield a safe system never the less and thus allow to co-locate safety related logic with non-safety related general purpose applications and services on a single contemporary system.

Keywords: Safety logic, complex systems, dynamic data types

1 Introduction

Digital systems have a number of well known advatages, notably with respect to signal rectification and complexity of signal interaction without degeneration - simply speaking the absence of noise. Though this absense of noise has great technical advatages, it has some interesting side-effects with respect to safety properties. During failure mode and effect analysis for digital systems one regularly stumbles across a number of fault classes semingly inherent to digital systems:

- permanent bit faults (a memory cell, a shorted relay, etc)
- transient bit faults (the infamous cosmic ray - or more earthly EMV issues)
- systematic operational faults (i.e. the FOOF bug in pentium I)
- transient operational faults (i.e. critically low voltage)
- temporal faults (i.e. clock drifts or clock updates)

This is a bit course grained but serves well for the discussion here. It also should be noted that failures are only one possible cause of hazards, so this is in principle incomplete [5].

All of these fault classes are then mitigated by different technologies (see [1] IEC 61508 part 7 for a overview of available technologies) , starting from redundancies and different levels, by adding diversity in hardware or software, and by introducing protocols that mitigate against consequences of these faults (i.e. using sequence numbers, CRC, etc.[4]). All of this sourounding safety related mitigations of the respectiv fault classes may well be technically suitable, but the question to ask first is - why do these faults exist in the first place - and if they are inherent to digital systems can they not be mitigated at a generic level rather than growing the complexity at the application level ?

There have been some indirect efforts to mitigate these issues at a generic level, think of the many IEC 61131 [3] runtime systems allowing to focus again on the relatively simple logic and handling the safety related issues below-the-hood. The assumption being that for a simple set of operations a complete list of potential faults can be established and mitigated. The methods though used for this mitigation again are relatively specific and in general limited to lader-logic or boolean-logic constructs - so the question remains - what is the root cause and can it be mitigated at a principle level?

2 Naive case study

This case study might seem quite trivial but from a safety perspective we believe it demonstrates the potential advantage of the concept introduced in this paper - dynamic data types.

```
 switch <---> actuator <---> indicator lights
(up/down)                         (red/green)
```

if this is implemented with common means then we have a signal level (voltage or digital makes no difference here) indicating the switch position of up/down, we have an actuator that operates on such a signal and we have the indicator lights that will provide operator feedback on current actions.

possible single faults (simple model)

- up stuck:
  actuator "sees" up pushed and lifts the dor, it is fed back to the operator via indicator light but the operator might not be able to close it, and the actuator can't actually determin that the input is unintended or invalid. The hazard is probably limited as the indicator is corect and humans thus can respond according to preset procedures to achieve a "safe state" - long term consequences of "up" being persistant are obviously only menaingfully interpreted in a particular context.
- down stuck:
  actuator "sees" down permanent and closes the dor - feedback to the operator indicates closed dor - safe state assumed - thus no hazard, but availability is impacted.
- actuator stuck:
  the actuator does not respond to input, it stays in what ever state it happens to be - this may be visible in the indicator light but the operator would not see that the actuator is damaged in the signal - an additional diagnostic input would be needed (i.e. indicator for the actuator operation status it self). Depending on the actuator stuck position this may or may not be safe - again context would be needed.
- indicator stuck green: - this would not allow the operator to detect a potential hazardous situation and exposur to a hazard could occure based on procedures (gree indicating entering permitted). The problem would not be detected until an operator attempts to change state by requesting "up" and not getting any response (indicator light changing to red) - but it would relie on human observation and even if detected the diagnostics would require additional means.
- indicator stuck red:
  impacts availability primarily.

Note that we are not concidering any temporal issues here - there are of course a whole set of temporal failure modes as well, nor are we concerned with the (and often dominant) non-technical failures like management and safety culture failures.

2.1 Adding diagnostics

As noted above there are a number of faults that could be diagnosed if additional information were availalbe from additional sensors - but the obvious disadvantage of this is that additional sensors mean reduced availability on the one hand and the problem of accumulated faults for any indicators of "rare-events" - notably the later is problematic from a safety perspective. A further issue is simply that the system intended to be keept simple starts becomming more complex than would be needed for the pure logic.

The above example exibits two main problems:

- lack of indication of "invalid" state for all components
- inability to identify the cause without additional sensors

If we look at the potentially hazardous situations then these are associated with the problem of single points of failure aswell as the inability to signal "invalid state".

Before concidering methods to implement redundancy, signal-diversity, safet communication, etc. we would like to look at the root causes and then derive requirements that could potentially eliminate them all together.

3 Root cause

Paradoxically, from a safety perspective, one of the problems of digital systems is the absence of noise - strictly speaking it is not absense, it just is a "rare" phenomena and thus perceived as being absent - we don't notice the rare event of bit-flips on the wire, either it is covered by detection and retransmission (i.e. ECC RAM, or noice cancelation on ethernet wires) or it leads to a failure that we then "fix" by rebooting our PC...

3.1 Value domain

The assumed absense of noise on the signals in digital systems has the irritating consequence that the rare event of bit-level alteration (SEUs) does not lead to a violation of value domain constraints. That is to say if a register contains an integer and a bit is fliped it still conatins an integer and is within range - the bit-flip is not noticable in the data-type. The reason for this is that the value domain is dense (even though it is discrete - dense in the sense that there are no "holes" in the representation) - conversely analog signals had to be constructed with a granularity to ensure detectability of divergence, so the analog signal spectrum, effectively usable, could not be dense.

SEU

Single event upsets are random alterartions of some state of the system that can't be predicted, they only can be covered by some form of fault-tollerance or fault detection/reaction - they are in principle not testable as they are not visible until they actually occured. In case where the affected resource may stay in the altered state, the alteration may be detectable if additional information about the expected state is available.

```
logic high    .---- perceived value
              |
logic low------- - - intended value
              ^
             SEU
```

Ommision

Ommision faults are due to the inavailability of some resource. They are in principle detectable, though the effort for this may be high - note that ommision faults may also have systematic characteristics (i.e. wire cross-talk).

```
logic high    . - - -  intended value
              |
logic low ------------ perceived value
              ^
         bit failure   output is a legal value
         wire breakage
```

So the SEU or ommision occurance is not visible in the output (think of this as a register or memory cell in which 0 was originally stored - due to wire cross-talk it is fliped to 1 and then read as 1), the bit failure on the other hand could also be due to accumulated faults or physical failures of memory/register/wire but stay unnoticed until activated by accessing. Testing for this fault is also not simple as it need not have caused a permanent damage, so writing to the SEU affected location "clears" the fault aside from the problem of some paths not being easily testable until you need them (think of fault handling or emergency shutdown procedures). With other words the problem is that this type of fault is nither detectable in the value domain, nor is it mitigated in the data type.

This is in fact inherently a property of digital systems, and a number of mitigations have been proposed. Some of these mitigations are:

- inverse replicated logic (2-out-of-2)
- diverse replicated logic (2-out-of-2, N-out-of-M for availability reasons)
- periodic testing and statistical SEU occurance models "protecting" the intermediate intervals.
- safe data objects, protected by additional representations (i.e. CRCs)

All of these protections have one thing in common, they tackle the problem at the symtomatic level, and all of them have the potential for common-cause faults, though the probabilities can be argued to be sufficiently low (if reality correlates with these arguments is not the topic here).

A further serious problem, from a safety perspective, of digital systems is that the absense of an active signal, due to failure, may also be a legal value - i.e. a solid 0.

The root cause of the problem though lies in the data type it self not becomming invalid by the alteration, and thus on-the-fly modification, and even permanennt modification (i.e. a stuck relay) are not visible/detectable in the data-type it self.

3.2 Time domain

A similar situation is given for the time domain. Basically the primitive data types don't have any encoding of the time-axis. Even if we pair values with some for of timestamps (also just a value representing some point in time) we don't substantially change the issue, we just have added a level of coplexity for a correlated fault to occure (that is change of the value and synchronous change of the time-stamp to fit the expected value - think of a zero-crossing detection for sinusoidal signal used frequency monitoring on power-lines, the failure now would require a value and time-stamp fault to lead to a fals positive, so the probability is reduced but it is not eliminated).

Clock fault

```
logic high    . - - - - - - .------
                            |
              |             |
logic low ------------------'
             T1            T1'
             ^
        clock fault   output is late but not
                     etectable by the externaly
                        perceived value.
```

4 dynamic data types

In this section we introduce a novel concept (atleast in the context of safety related logic) of encapsulating the general signal in a way that does not exhibit the above mentioned limitations, and would not reuiquire mitigation by additional means but rather enforces inherent mitigation of faults. At this point we are not (yet) claiming that the fault coverage is actually complete - but we do think that the majority of relevant faults (notably single fault hypothesis) are covered without impacting availability by false-negative.

The system model behind dynamic data types roughly is to view the input signals as streaming through the system and accumulating changes based on the operations done on them. There is no reliance on the correctness of operation, there only is a reliance of the effects of operation being uniq and visible in the signal. This is in some ways similar to coded monoprocessors [7] that focus on the operations rather than on the data, and ensure that operations cary such a uniq property. The basic operations that are currently being studied are:

- Signal expansion: i.e. adding of signals - extending the signals content to contain the difference and sum of the inputs.
- Signal reduction: i.e. filtering of signals - reducing the spectrum it represents.
- Delaying the signal: i.e. phase shifting - altering the temporal correlation to other signals in the system.

4.1 Signal Requirements

The requirements listed here are only from a safety perspective and not from a functional perspective. further more they are most likely not yet complete as this project is in an early stage.

- all valid states must be active states - inactivity of any component may not lead to a legal value

- all elements may only react to valid states (implying activ states)
- SEUs shall not impact the value domain (general resilience against SEUs)
- signals must encapsulate temporal properties along with value properties
- Operations on signals may not include a single-point-of-failure (i.e. comparsion to fixed value or fixed offset addition or filter parameters of which modification of only one leads to a false-positive filter)

To satisfy these - admitedly very crude - requirements, we propose mapping logical values to descreet frequencies, with a tollerated range. In this example we mapped:

Logic	Dynamic representation
TRUE	6 Hz
FALSE	12 Hz

Each value needs a specific tollerance - so the value domain which is mapped to the frequency domain is sparse, and thus tollerant against a certain deviation (i.e. transient SEUs). Further signal interaction is by multiplication, thus any signal ommision would lead to multiplication with a 0 value and thus no valid output.

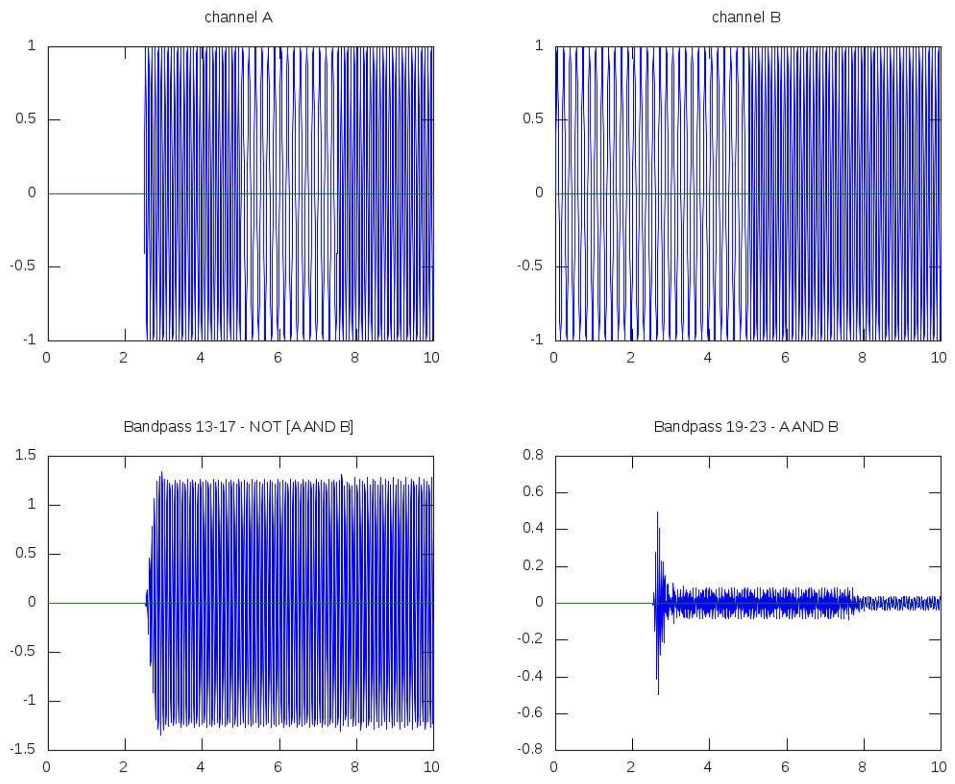

FIGURE 1: *response to input omission failure*

Stuck at failures are a bit more critical, selecting proper frequencies in the genenerated intermeidate specturm in combination with band-pass filtering (in this case butterworth filters) allows covering both omision and stuck-at faults.

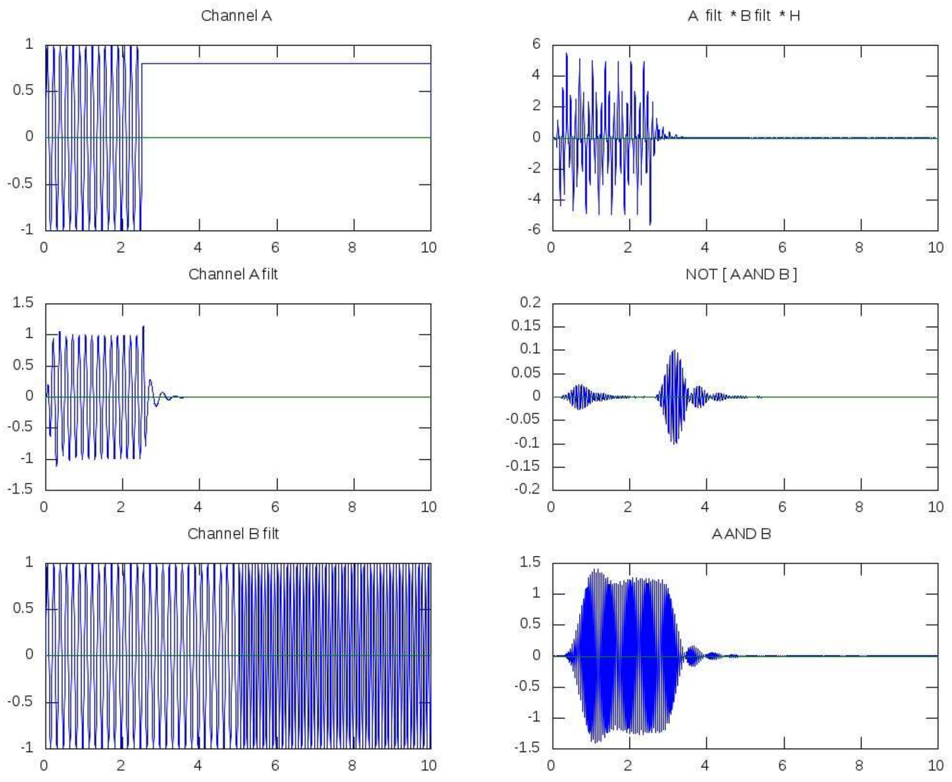

FIGURE 2: *response to stuck-at failure of input*

4.2 Core concepts of dynamic data types

From a safety perspective dynamic data types concept can be summarized as

- variable values are bound to a frequency spectrum and so conserve temporal information of the signal
- only active states are legal (recognizable) states - that is signal omission or stuck-at does not yiedl a valid signal
- combinations of signals lead to valid signals or no signal
- Complex logic can be described by signal composition
- compromising dynamic data types would require arbitrarily complex correlated alterations of data/code to yield a valid signal
- increased system complexity reduces the probability of a false-positive signal generation

Essentially the goal of dynamic data types is to "scale" with system complexity groth which we concider inevitable.

4.3 Hardware elements

The concept of dynamic data-types does not only allow implementing computer based logic ontop of complex hardware/software systems safely but also would lend it self to low-complexity system that can be fully integrated with computer based systems without reduction of generality.

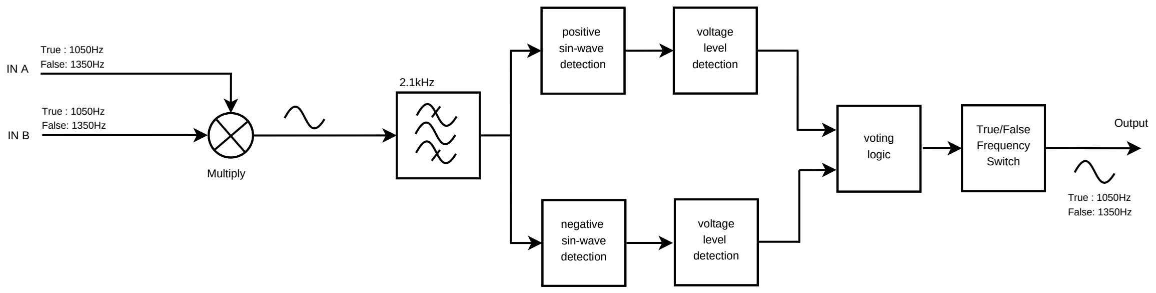

FIGURE 3: *Realisation of logic element as low-coplexity hardware*

Also as the whole point of dynamic data types is to encapsulate not only state information but also status and temporal information in the data representation this only makes sense if the end-elements, the actuators in the system, also can utilize these inherent safety properties.

A basic model of an end-element for the use in dynamic data types, is outlined here. The basic concept is to process the bandpass signal as seperate half-waves that provide a diverse representation of the signal - thus any stuck at at best could trigger one side but not both - finally utilizing both half-waves to generate the ouptup action. Naturally such an end-element is quite problem specific thus we can't generalize this, but this shema does show how the transition from the digital logic representation/processing to final elements can be done retaining the technical safety properties of the hardware/software system.

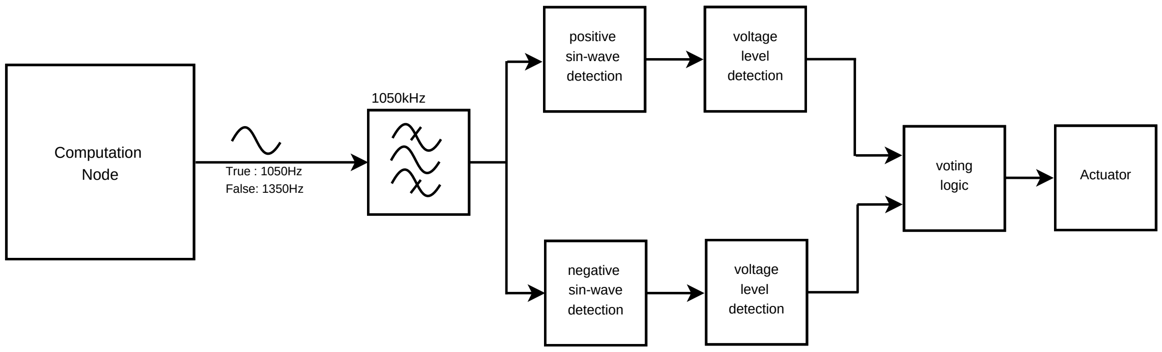

FIGURE 4: *Realisation of safe end-element*

While this is still active research, this preliminary hardware implementation does show the generality of the concept allowing to establish a uniform safe signaling model within a safety related system.

4.4 Composable logic

A further issue for any control system is composability. While we are restricting the model here to boolean logic at present extensions do seem possible. Never the less the first step is to demonstrate composability of basic logic elements.

Even though the presented example is not a wildly impressive level of complexity - many times one can find relatively simple safty logic involved in monitoring opperations of complex systems. Basically the majority of the control is not safety related - think of this as a CNC machine where the only real safety related issue is if the protection dors are closed during opperations or in case of opperation with open dors (for posssitioning and maintanance) the movement of components must be must be limited to very low speeds - the essencce of this example is that the actual safety conditions are relatively simple and it is typically resolved by monitoring the system and intervening in case of constraint violationrather than relying on the entire control software to be safetyrelated and "bug free". This relatively simple logic components (i.e. dorALock & dorBLock) can now be monitored by a dedicated safety related system

- hat is some MCU or could be run on the available computing capacity provided adequat isolation could be enforced as stated in IEC 61508-3 Clause 7.4.2.8/7.4.2.9 [?]. To provide such simple logic one can use dynamic data types that allow creating reliable logic from signal summations followed by digital filtering to extract the intended logic operation. This is achieved by mutiplying the inputs and the resulting frequency summation/difference again by a constant helper frequency.

```
Inputs
A  B   A*B   H1  Spectrum of A*B*H1
6  6   0,12  27    ,  ,15,  ,27,  ,39
6  12  6,18  27    ,9 ,  ,21,  ,33,  ,45
12 6   6,18  27    ,9 ,  ,21,  ,33,  ,45
12 12  0,24  27  3 ,  ,  ,  ,27,  ,  ,  ,51
                         | XOR |
```

The output frequency spectrum now allows to use a bandpass to extract A XOR B from the above spectrum. By changing the input H1 (a helper frequency) the frequency spectrum generated by multiplying the two inputs withthe helper frequency allows extraction of A AND B. By following the input multiplication stage by a digital filter (in our case we use a 4th or 5th order FIR filter) one then can extract the desired compositional logic term from the signal.

```
Inputs
A  B   A*B   H1  Spectrum of A*B*H1
6  6   0,12  9   3 , 9,  ,21,  ,
6  12  6,18  9   3 , 9,15,  ,27,
12 6   6,18  9   3 , 9,15,  ,27,
12 12  0,24  9     , 9,15,  ,  ,33
                      | AND |
```

While this may seem quite a complex way of doing this, it is precisely the complexity introduced that is the protective mechanism against systematic and stochastic faults in the system as the singals logic value is encoded in the frequencey of the signal and thus to transform a signal into a diffferent logical signal (i..e. a signal that shuld be N Hz representing on to a frequency of N Hz representing off) would require a highly complex sequence of synchronous faults to appear - all faults that appear randomly will ither be in the tollerance of the singalling system (filter bandpass tollerances) or will cause the system to enter an identifiable invalid state and lead to a safety reaction. SFor systematic faults it may be a bit more complicated to argue such an approach but equally the impact of any systematic fault at the lower level (i.e. operating system, system libraries) would have to generate a very complex and synchronous responce and not merely a local fault like the infamouse FOOF bug [6].

The claim here thus is that such relatively low-complexity sfaety logic elements can be run on a unsafe OS while retaining the safty properties of the logic. It should be stressed though that we are not claiming any mitigation of systematic faults in the implementation of the safety logic it self - rather only mitigation of systematic faults in the underlying generic omponents is claimed. Withthe overall complexity of the safety logic and the signal processing benethe it being relatively low we think that it is an absolutely realistic target to implement such a logic according to suitable standard procedures and under control of adequate safety management to ensure with adequate probability that the residual failure rate of the safety logic is sufficiently low. As an example the DDT logic presented here takes two inputs A and B and a control input H and provides an output of A o B and NOT A o B, with the operation being AND or XOR depending on the setting of H.

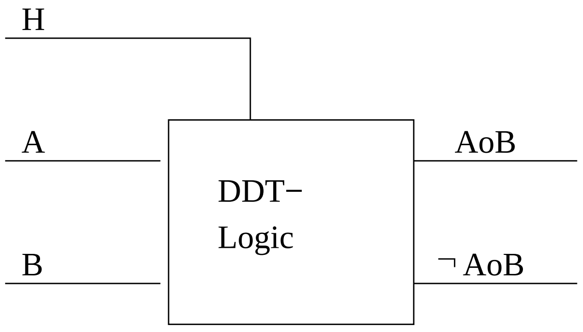

FIGURE 5: *composable logic element structure*

Note that the frequencies selected here during simulation, technically are not sensible, but the principles don't change when transformed to other ranges of the spectrum - any real-life system would be opperating in the kHz range.

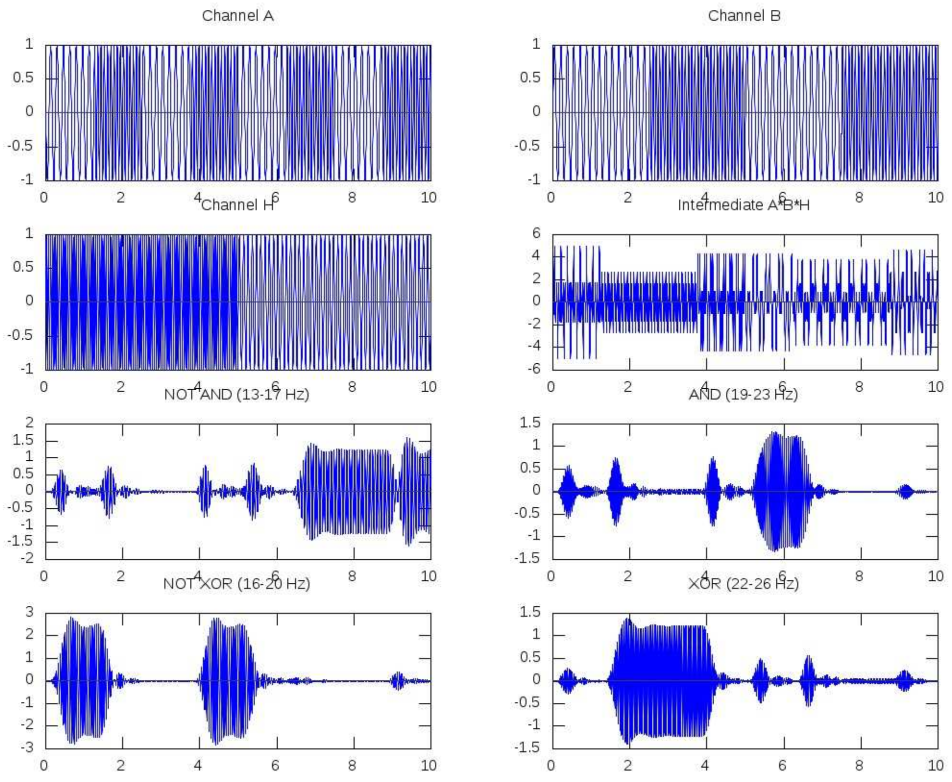

FIGURE 6: *composable logic element based on DDT*

Inspecting the generated frequency spectrum one can extract further logic operations on the inputs from the output spectrum through adjusted filters aswell as additional logic inputs. At this point we nither know the scalability of this composition nor do we know its limitats - but the current state of expreiments is sufficient to allready produce some basic boolean safety logics that could be practicable for safety related applications running on complex (non-safety) hardware software platforms.

4.5 Masking through complexity

As noted this is in the context of efforts to utilize complexity of udnerlaying hardware/software systems to enhance safety rather than fighting them. With the the mapping presented in the last section we obtain a process to construct outputs that is not suseptible to faults in the underlying software/hardware systems because any false-positive requires an arbitrarily long correlated sequence of faults to lead to the false positive. Any fault in the unerlying system can impact the processing of dynamic data types at arbitrary points, but as there is no single point of failure in the transformation of the imput stream to the output stream, no such single fault could impact the logical corectness - it can of course diminish availability - which is not of concern here. Regarding multiple faults, there is of course the possibility of accumulated dormant faults emerging and these then striking at some mode change - i.e. in the above example when swithihing the mode-input from AND logic to XOR logic - but any such fault would have to again generate a complex sequence of values to lead to a false positive output which is not impossible but unlikely - how unlikely it is is still under active investigation.

As the data stream though always depends on multiple manipulations (one could view dynamic data types as a N-value diverse data representation) SEUs are fully mask (both in signals as well as parameters for i.e. the filters) and at the same time would be detectable by independant singal monitoring. In many ways what is being proposed here is the re-invention of quite simplistic analogue technology in a digital world - but with the advantage of retaining the signal transmission properties and the ability to manipulate the signals in software.

5 Conclusion

Essentially we believe that safety needs methods to build on complexity - ideally appreciate complexity of underlying systems, without compromising safety principles. We see little point in a continuous fight against ever increasing complexity of system software and hardware - the question of "should it be done in software" is legitimate in many cases, but there is also little point in denying the trend towards complexer software/hardware systems and aggregation of functions of different safety levels in systems. If the safety community does not find answers to the complexity groth that fit current technologies and goes on recomminding simple systems with simple or no operating system, this will eventually backfire as we will see the expertise and expreience as well as field data of these systems faid into negligence - resulting in degraded safety in the long run.

We are not claiming that dynamic data types are THE answer to the problem, but we do believe that they are an example of the potential that lies in the paradigm change when switching to "capitalize on complexity" rather than "figth complexity".

As Leaveson notes that software can affect system safety in two principle ways:

- it can exhibit behavior in terms of output value and timing that contributes to the system reaching a hazardous state
- it can fail to recognize and handle hardaware failures that it is required to control or respond to in some way.

As safety is a system property dynamic data types can't in principle guarantee software safety - but we believe dynamic data-types can give reliable mitigations to both of the above mentioned potential of software to impact system safety. Thus while the implementation of dynamic data types (the software components aswell as the frequency/spectrum-gap selection) are directly safety critical, such a system can be allowed to run on a un-safe operating system/hardware due to the inherent detectability of induced faults. Thus the required independance from the co-located software components can potentially be provided.

While we see dynamic data types as a potential solution to some of the typical safety logic needs, we are well aware of this not yet being a mature concept - rather we hope that we will find opportunities to study this approach in more detail. Independant of the suitability of the dynamic data type concept though, we believe that it is time to start thinking about how to utilize complexity and inherent non-determinism for safety related systems rather than continuing to relie on an abstract model of computing that is crumbling - if not vanishing.

References

[1] *IEC 61508-7 Ed 1 Functional safety of electrical/electronical/programmable electronic safety-related systems Part 7: Overview of techniques and measures*, International Electrotechnical Commission, 1998

[2] *IEC 61508-3:1998 Ed 1 Functional safety of electrical/electronical/programmable electronic safety-related systems Part 3: Software requirements*, International Electrotechnical Commission,1998

[3] *IEC 61131*,

[4] *EN 50159-1 Railway applications - Communication, signalling and processing systems Part 1: Safety-related communication in closed transmission systems*, CENELEC, 2003

[5] *Safeware: System Safety and Computers*, Nancy Leveson, 1995 ADDISON-WESLEY

[6] *Pentium F00F bug* ,Wikipedia, May 2010 HTTP://EN.WIKIPEDIA.ORG/WIKI/0XF00FC7C8A

[7] *The Coded Microprocessor Ccertification*, Ozello Patrick, 1992,SAFECOMP