Automotive Safety Concepts : 10^{-9}/h for less than 100 € a piece.

Jean-Louis DUFOUR, safety expert for Powertrain and Chassis electronics

PSA Peugeot Citroën, La Garenne technical centre (Paris)

jeanlouis.dufour1@mpsa.com

Abstract

Safety-critical EE functions are now common in our cars, the canonical examples are ABS and motorized throttle systems. These functions are no more reserved to high-end vehicles, and this should amaze Railway and Avionics engineers, who are used to "pay" for safety.
This small miracle is to credit to the imagination of Automotive engineers, who are invariably faced with a price/performance dilemma. The aim of this presentation is to survey the most common safety architectures (or "safety concept", at the computer level, not at the vehicle level), and to show how we can claim to reach the mythical 10^{-9}/h hazardous failure rate without appealing to 2-out-of-2, 2-out-of-3 or "coded processor" architectures.
The respective strengths and weaknesses will be evaluated, with two goals :

- the strengths should give ideas to Railway and Avionics engineers to improve the quality/price ratio of their architectures,
- the weaknesses should give motivation to Automotive engineers to reach a common and adequate State of the Art.

1 - Introduction

1.1 – "By-wire" systems in the car industry …

"By-wire" substitutes for mechanical systems are a growing trend, but by-wire safety-related systems are not so easy to introduce, particularly in a cost-driven industry like the car industry. To present them, let's start with a classical gasoline engine, where the torque is mainly determined by the air intake. To adjust this input, the accelerator pedal drives a kind of shutter, the "throttle", interposed between the air filter and the inlet valves : (almost) closed, the engine idles, completely opened, the engine runs at full power. A spring brings this throttle back to the idle position. From a safety point of view, the typical failures leading to an engine beyond control are a locking of the throttle or the pedal, a breaking of the spring, a leak in the inlet manifold (downstream from the throttle), and of course problems in the pedal/throttle link.

On "old" engines (say, before 1996 for Peugeot-Citroën cars), this pedal/throttle link was simply a cable. But new functions like speed regulation, stability control (ESP) or a robotized gearbox generate significant torque requests independently of the pedal. This is why the cable had to be replaced by a more clever link : a pedal sensor, an engine management unit (based on a 16-bit or 32-bit micro-controller, connected to a CAN bus for external torque requests) and a throttle motor, the whole forming a "drive-by-wire" system (the patriarch of the automotive "X-by-wire" family). And the seizing risk associated with the cable had also to be replaced by more "clever" risks : electrical risks (shorts/cuts), electronic risks (particularly on VLSI components) and software risks (bugs).

Still in the powertrain domain, the same story occurred at the same time for Diesel engines (replace the throttle angle by the injection duration). More recently, robotized gearboxes (risk : unwanted gear shift) associate the attractiveness of an automatic transmission with the fuel consumption permitted by a manual gearbox.

In the chassis domain, ABS dates from the eighties, but firstly it is not strictly speaking a "by-wire" system (i.e. a substitute for a mechanical system), but a "modulation" system on top of a classical brake, and secondly we had to wait the mid-nineties before entrusting it with the front/rear Brake force Distribution (EBD, critical for stability in case of emergency braking; interest : optimal braking on the rear axle, whether the road be dry or wet). At the beginning of the decade, Power Steering has also become electric (EPS, risk of "self-steer"; interest : fuel consumption), and very recently it was the turn of the Parking Brake (EPB, risk of loss of immobility or of unwanted braking; interest : hill start aid).

We conclude the inspection with the body domain, which contained for a long time EE safety-related systems (airbags, lighting) and even by-wire safety-related systems (electric windows, even if it was recognized as such

only recently). It now experiences the invasion of "comfort" by-wire systems like programmable seats, automatic doors or electrical steering column lock (the associated risks are easily foreseen).

1.2 – … and in mass transportation systems …

Of course, Railways and Avionics have been forerunners of the use of computerized safety-related functions :

- The first computerized railway interlocking : 1978, in Sweden,
- The first computerized driverless subway : January 1986, the "Skytrain" in Vancouver (the driverless "VAL" system in Lille [France] opened in May 1983, but the safety system is analogue and not digital),
- The first digital fly-by-wire airliner : April 1988, the A320 (it was anticipated by military aircrafts and the space shuttle, but we can not speak of "mass transportation").

Moreover, fly-by-wire systems are not comparable with the current X-by-wire automotive systems mentioned in the introduction : they correspond to the Steer-By-Wire system (SBW, not to be confused with EPS), expected for the next decade (so Avionics lead is at least of 20 years !).

1.3 – … pose the same problems.

Hazards when using computerized systems have 2 main sources :

- As for non-computerized systems, hardware faults (also called "random"), quantified with a failure rate (per unit of time, e.g. 10^{-7}/h for a complex micro-controller in an engine environment) and a "useful lifetime" for wear out mechanisms (e.g. 10^5 write cycles for its integrated Flash memory),
    - But for highly integrated components, attention is turned on "0 km" faults (manufacturing defects escaping the screening processes, which can no more reach a 100% coverage), quantified with a defect rate (e.g. 50 ppm for the same micro in the latest 0.15 µ technology).
- But also design faults (also called "systematic", in other words "bugs"), mainly on software, and more and more on system aspects (we have not yet emerged from the "software crisis", but we have already entered the "system crisis").

Design faults are today the main challenge of computerized systems (safety-related or not), but they are not the subject of this paper and **in all that is coming after we will assume the design « correct »**, to focus on the first point : the coverage of hardware failures. In a first part, we will recall the traditional digital safety architectures. In a second part, we will present the specificities of the car industry and the choices made for the automotive safety concepts.

2 - The traditional solutions

There are two traditional ways of making secure the computation of a function :

1. by **Hardware** redundancy : two units "A" and "B" perform in parallel this function, and their outputs are compared. If there is **no common mode**, when a failure affects A, B has no reason to be faulty.
2. by **Software** redundancy : a single computer runs two diversified implementations "A" and "B" of this function, and their outputs are compared. If diversification is strong enough, there are **no common effects**, i.e. for any failure, its effect on "A" has no reason to be the same than its effect on "B".

In both cases, when a discrepancy between A and B is detected, it is not possible to tell which one is true and which one is wrong, so the only reasonable behaviour is to enter a safe state :

- for the rail, emergency braking for an on-board system, power switch off for a wayside system,
- for a car it can be a more or less severe and sharp performance limitation : of the engine ("limp-home"), of power steering, inhibition of the speed regulation, …

This is called a **fail-safe** architecture, in contrast to **fail-operational** architectures used in "availability-critical" (e.g. : driverless systems, interlockings with heavy passenger traffic) railway systems ("2-out-of-3" or "dual-duplex" redundancy) or in airplanes ("system" redundancy for the A320 [TRA 91] : e.g. roll and pitch controls can be driven by 2 kinds of computers ["ELAC" and "SEC"], which are themselves safe computers). Here we will confine ourselves to fail-safe architectures.

2.1 – Hardware redundancy

It is the most obvious solution, with 2 main variants :

- the **cross-check** : A and B exchange their results and each one has the potential to reach a safe state,
- the **external check** : A and B send their results to a third party, which has the responsibility to reach a safe state.

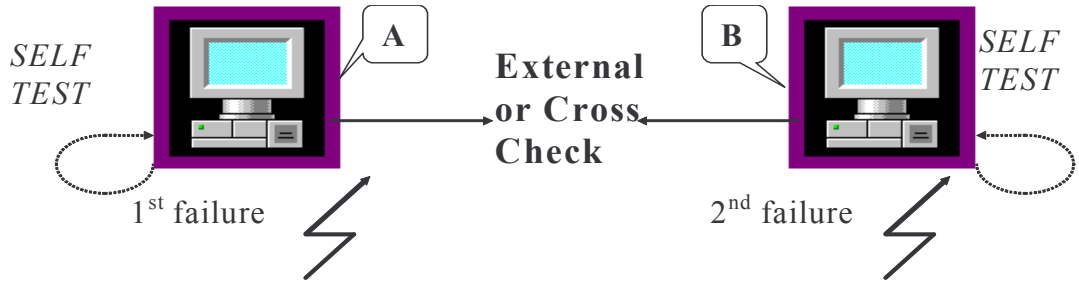

Figure 1 : redundancy

The concept is obvious, but the numerical evaluation of the safety level is a bit subtle, and we will state carefully the hazardous scenario :

- a first error occurs on A or B (say A for the example), so with hourly probability 2λ (if λ is the failure rate of A and of B),

This failure will be detected (by its effects on the outputs of A) at the next A/B check, and in the case of a cyclic software of period T_{cyc} (for typical railway applications, 100ms < T_{cyc} < 400ms), this check will occur within the next T_{cyc} period. Therefore, for the first error to remain undetected,

- a second error occurs on the other unit (B in our example) within the next T_{cyc} period (and masks the first one), so with absolute probability λT_{cyc}.

If there are no common modes, both events are independent, and the hourly probability of the scenario is

$$2\lambda^2 T_{cyc}$$

In the case of λ = 1e-4/h[1] and T_{cyc} = 360 ms = 1e-4 h, the astronomical result of 2e-12/h is very satisfactory compared to the usual (railway and avionics) requirement of 1e-9/h.

Note : the careful reader has noticed that we have not valued the hazardous failure rate at "λ^2 = 1e-8/h", because firstly we should rather write "λ^2 = 1e-8/h^2", and something "per square hour" has absolutely no physical meaning, and secondly even if we come down to the first hour, the absolute probability 1e-8 is the probability that both units have failed at the end of this first hour, which has not much to do with the feared event.

Unfortunately, this formula is only a vague indication of the real safety level, because

- if there are no common modes, the second error is unlikely to mask the first one by having similar effects on the outputs (particularly if the outputs are integers and not Booleans) : the real safety level can be much better (but almost impossible to evaluate),
- if there are common modes, similar effects are more likely to occur, but the product is no more valid : the real safety level can be not so good (and can even reach the reliability of a single unit, λ).

Worse, this scenario is not the most likely to happen : when the first error occurs, it has not necessarily immediate effects on the outputs, therefore it may be undetectable from outside. In this case, the fault is said to be **latent**, the **latency** being the period while the fault has no visible effects. During this period, the second fault can quietly settle on the other unit and mask the first fault (if both reveal at the same cycle, but the reader certainly knows that safety engineers are a bit paranoiac). Latent faults are not difficult to imagine :

- in the RAM of a railway on-board controller, the stuck-at-0 of the Boolean variable requesting an emergency stop (reveals only in an emergency situation),
- in the FLASH of a wayside controller of a Paris-Berlin rail route, the change of the site or of the curve of the last bend before Berlin (reveals only as it gets near Berlin),
- in FLASH of an airliner controller, a change in the code which lowers the undercarriage (reveals only at landing),
- in the ALU[2] of a railway, avionics or automobile ABS, a stuck-at-1 of the high-order bit of the result of an unsigned multiply operator, used only for the computation of the releasing period of the brakes (reveals only during an ABS regulation).

So if T_{lat} is the latency of the first fault, the masking fault has the absolute probability λT_{lat} to occur in the right time-slot, and the whole scenario occurs now with an hourly probability of $2\lambda^2 T_{lat}$, with of course T_{lat} potentially much greater than T_{cyc}. If we take T_{lat} =1h, we obtain $2\lambda^2 T_{lat}$ = 2e-8/h for the whole scenario : 20 times the target.

---

[1] Just for illustration, real figures are better.

[2] « Arithmetic and Logic Unit », i.e. the part of the core of a micro which contains calculation and test operators (the other parts are the register bank, the addressing block, the instruction decoder and the instruction sequencer).

The problem is that the maximum latency is very difficult to determine, and the mean latency (which is the pertinent data to inject in the above formula) is almost impossible to determine.

To limit the latencies, we have to "go latent fault hunting" : in parallel with the application, we will run self-tests on the hardware parts which may shelter latent faults, in particular memories (RAM, ROM/FLASH and EEPROM) and core (the instruction set). These self-tests will force latent faults to reveal in a bounded time : for example if they take 3 mn (1/20 h), it implies that the mean latency T_{lat} is shorter than 3 mn, so the hourly probability for the whole scenario $2\lambda^2 T_{lat}$ is less than 1e-9/h.
Two sensitive points concerning the self-test approach are worth mentioning :

1. their coverage, which can never be 100%. It means that a part of the failures will never be detected (by the self-tests), and in this part a subpart may have longer latencies (depends on the application) than the self-test period. Therefore the mean latency can no more be bounded by the self-test period. Moreover, this coverage itself is very difficult to determine (except for the very special case of a ROM diagnosed by a CRC, where the coverage results directly from the polynomial used). There is no satisfactory answer to this point (only "state of the art" and safety standards).
2. their intrinsic helplessness in the face of transient failures : let's consider the reversal of a RAM cell during an electromagnetic perturbation. If the reversal occurs in the middle of the write/read-back test of this very RAM cell, of course it will be detected, but if it occurs anywhere else, the test will not detect anything. From a theoretical point of view, this second point can be reduced to the first one, by considering that transient failures are included in the failure rates and in the test (non-)coverage, but from a practical point of view this is a problem of ever-increasing importance, because of ever-thinner (therefore ever-sensitive) VLSI technologies.

These two points are often the subject of theoretical Markov-type studies ([Ame96], [Fre92]), but the problem is the evaluation of the parameters of these models.

2.2 – Software redundancy

The (source) code diversity

From the very beginning of the use of computers for safety functions, designers have tried to avoid hardware duplication. It resulted at the beginning of the eighties in a few railway systems based on software redundancy : a single hardware running two diverse versions of the application ([JON81],[VOG88]). Fig. 2 illustrate an elementary way of doing this (on a Boolean function, typical of railway interlockings) :

- Reversal of the polarity of the inputs,
- Use of De Morgan laws[3] to systematically transform all the computations in the application (in particular, all intermediate variables are reversed),
- Reversal of the polarity of the output.

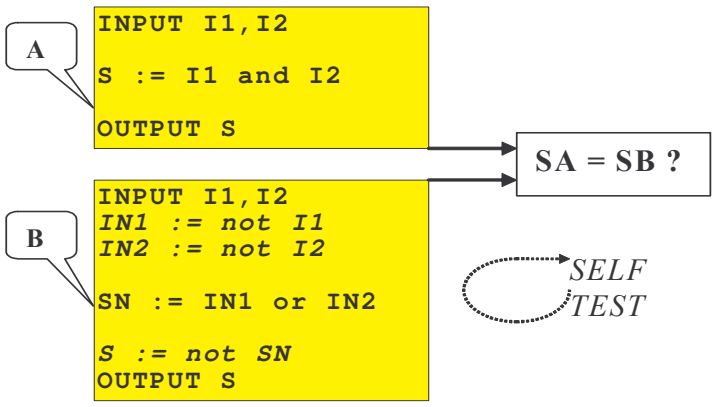

Figure 2 : software redundancy by diversification of coding

Both versions are completely in common mode in relation to hardware failures, and we hope that when a failure occurs, it will have different effects on the two versions. The hazardous scenario is :

- A failure occurs, this with hourly probability λ ,

[3] $\overline{A \wedge B} = \overline{A} \vee \overline{B}, \overline{A \vee B} = \overline{A} \wedge \overline{B}$.

- it has the same effects on A and B, this with absolute probability p,

therefore the hourly probability of the scenario is λp, and the challenge is to find an upper bound for p.

The method to estimate p is the same than the one used for the coverage rate of a self-test : we choose a fault-model (e.g. stuck-at) and we inject these faults in the system (hence the name "fault injection"; it can be done physically or by simulation). But contrary to the hardware redundancy case, in which the coverage rate played a minor part, here it impacts directly the error detection capability.
This experimental determination of p is the Achilles heel of software diversity, because fault injection requires

- to show the representativeness of the fault model,
- to reach an ambitious goal : $p < 10^{-5}$ (if $\lambda p < 10^{-9}$/h and $\lambda = 10^{-4}$/h) with an « acceptable » (to be defined) confidence interval,
- important means (at least for physical fault injection) and time,
- to start it again as soon as hardware or software changes.

In this connection, continuous self-tests are more than ever required, not against latent faults, but simply to reach an acceptable p. As before, their coverage is thus very important, but now the detection time must be much shorter : against the accumulation of latent faults, 3mn was an acceptable value, but with common effect faults, the first fault may be hazardous : how long would you tolerate an engine out of control or a complete loss of the brakes ? 300 ms is a typical target value, and it means that the CPU load of these self-tests becomes very important.

Note : these two versions, A and B, aim only to detect hardware failures : B can be automatically derived from A by a tool. It mustn't be confused with N-version programming, which aims to detect also coding (and design ?) errors and relies on development team redundancy.

The "coded processors"

However, a few manufacturers (2 to our knowledge) have managed to get round the problem of the experimental determination of p, using a special diversification method which gives directly p : information encoding ([Rut84] for General Railway Signal, [For89] for Matra Transport). The idea can be seen in two ways :

- an extension to software instructions of the CRC on messages or of the checksum on files/ROM,
- an application to software of the « casting out the nines » of schoolboys, to check their multiplications.

We will choose the second way, and we will remind the reader how to cast out the nines on the addition 17+25 = 42 :

- we compute the remainder of 17 divided by 9 : it makes 8 (because 17 = 1*9+8, and for a pupil 1+7 = 8 [exercise : why ?]),
- the « modulo 9 remainder » of 25 equals 7 (for 25 = 2*9+7, or also 2+5 = 7),
- the remainder of 42 equals 6 (for 42 = 4*9+6, or also 4+2 = 6),
- and we check that 8+7 = 15 is definitely equal to 6 modulo 9 (see fig. 3).

$$\begin{array}{rr} & 17 \mid 8\ (=1+7) \\ + & 25 \mid 7\ (=2+5) \\ \hline & 42 \mid 6\ (=4+2) \end{array}$$

Figure 3 : to cast out the nines (on an addition)

A pupil would draw the conclusion that the addition is correct. We will appear more cautious and conclude that if an error occurred, it would have a one in nine chance to remain undetected.

In practice, the "coded processor"[4] of Matra Transport (now Siemens Transport) looks like fig. 4, except that instead of 9, a more respectable key of the code is used.

- on MAGGALY (driverless subway of Lyons [France]), the key's size makes 31 bits, thus $p = ½\ 10^{-9}$,
- on the Meteor line (Paris) or the Canarsie line (N.Y.), a double key of 2*24 bits is used (C5E975h and BAC239h), thus $p = 6\ 10^{-15}$.

[4] This name is misleading, because it reminds of hardware, while it can be completely software-implemented.

With such values of p, the accurate value of λ is no more really important, that's why it is sometimes said that the safety level of a "coded processor" is independent of the underlying hardware (in fact, there are theoretical reasons for such margins).

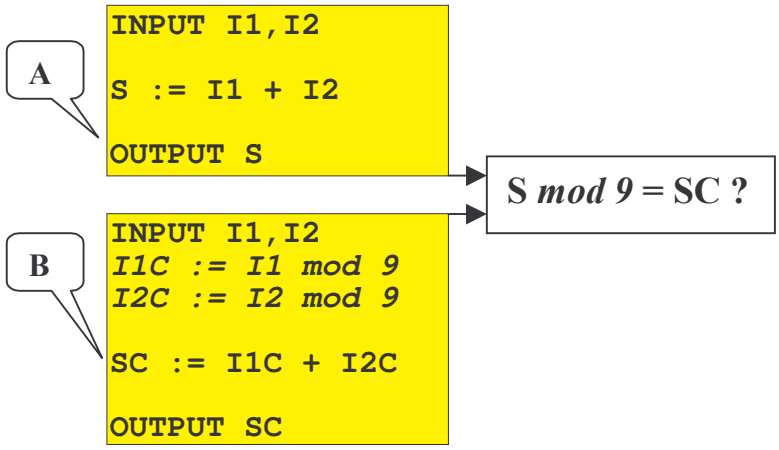

Figure 4 : software redundancy by information encoding (with an arithmetic code[5]).

"Coded processors" have also their Achilles heel : the CPU load is at least 10 times greater than for "normal" computations. It must be moderated by the fact that continuous self-tests are no more required (because p is fully guaranteed by the encoding principle).

3 - The automotive solutions

3.1 – The automotive context

An engine control unit is a small box (typ. 15*15*5 cm) which must cost less than a hundred Euros, to compare with a railway or avionic rack, which can be more than hundred times more expensive. Their only common point is a difficult environment (temperature, vibrations, EM noise), which requires industrial range components.

It is useful to keep in mind the following differences between the automotive industry and the traditional actors of safety systems :

1. Automotive electronics are not subjected to safety standards (EC regulations exist, but for electronics they don't ask for a safety demonstration), and there is no quantitative safety target,
    a. Automotive electronics are not subjected to independent safety assessors (the activities of approval bodies [UTAC in France] are not comparable with those of the JAA for aviation or of the "Notified bodies" [Certifer in France] for railway),
2. Automotive electronics are mass-produced, the car market is highly competitive, and the lower-end and middle-end groups are very price-sensitive,
    a. Automotive electronics must use the latest technologies available on the electronics market,
    b. Automotive electronics is highly integrated : no processor board, I/O board, network board or power supply board in a rack, but a single board based on a micro-controller including its own RAM, FLASH, CAN module and even now EEPROM.
3. Automotive electronics is more and more responsible for innovative functions, and this at a more and more rapid pace.

The 3 architectures shown in the first part all pose a problem in this context :

- Classical redundancy is not compatible with the production cost objectives,
- Fault injection needed for software diversity is not compatible with the continuous technological advancement,

[5] This is an *arithmetic* code (i.e. based on addition) because the SACEM system (Paris subway) needed kinetic energy computations (so products and sums). Concerning the other "coded processor", GRS (now Alstom Transport) supplied only interlockings (so Boolean operators), thus it is build on a *polynomial* code (i.e. based on bitwise exclusive-or, just like a CRC).

- Fast processors needed for coding are not compatible with the production cost objectives.

3.2 – Redundancy-based computers

Contrary to what has just been said (first item of the last paragraph), truly redundant computers exist in Automotive systems :

- mainly for very simple functions (active sensors [pedal or throttle angle], intelligent sensors [steering angle]), usually based on 2 components (ASICs or 8-bits micro-controllers)
- but there are at least two cases of complex functions : two Chassis systems ([Fru00] and another ABS/ESP supplier), both based on single « dual-core » components (two identical 32-bits RISC cores in the same chip, running synchronously the same application).

A dual-core is interesting compared to a dual-micro, because a 32-bits RISC core takes only on the order of 5% to 10% of the chip area (area = price) : the chip area is in fact driven by the FLASH size, and the FLASH needs not to be duplicated (it can be secured by parity / SEC / SEC-DED codes[6]).
If no dual-core is available, if duplication of the micro is to be avoided, it is still possible to use redundancy : by associating the "main" micro with a smaller one (let's call it the "secondary" micro), which runs a simplified version of the application. The comparison between both channels needs then to tolerate small differences. This is called "**asymmetric redundancy**", and it is used at least in an EPS system.
From a conceptual point of view, the comparison with tolerance weakens error detection, moreover for availability developers will naturally define important tolerances, so the safety engineer has to check this point carefully. But the concept is sound and gives a safety level not far from a true "symmetric" redundancy. In addition the induced software diversity contributes to bugs eradication (but it is off the subject).
From a practical point of view, the drawback is that the secondary micro is application-specific :

- the simplified application will also evolve during the development (so we have to manage two parallel software versions, with compatibility problems),
- small 8-bit micro-controllers don't always have FLASH and are not always downloadable,
- if the computer must implement a new critical function, both micros (and their exchanges) must be updated/upgraded.

We will not develop further redundancy, because there is nothing new compared with traditional concepts. But the other Automotive safety concepts can not be described as (absolutely) non-redundant : they are most of the time a mixture, mainly because of analogue inputs and (analogue or digital) outputs.

As regards critical analogue inputs, let's take the example of the air throttle angle : the sensor, based on 2 potentiometers or 2 Hall probes, when powered (voltage "Vdd" [5V] provided by the ECU [Engine Controller Unit]) makes 2 voltages available (for the ECU), say "thr1" and "thr2", such that :

- thr1 is (roughly) proportional to the angle, and varies from 10% to 90% of the supply voltage Vdd (unused ranges are for shorts/cuts diagnostics),
- thr1 + thr2 = Vdd.

Both signals are acquired by the ADC (Analogue-Digital Converter) of the micro-controller of the ECU. Their complementarity permits the detection of the "offset" failures of this ADC, of the "stuck-at" failures, except the "stuck-at-50%" failure, and of a part of the "gain" failures. Thus, to reach a "100%" coverage of the ADC failures, something more has to be done on the ECU side : one of the two signals is acquired by a second ADC on a second micro-controller, and the value is transmitted to the first micro via typically a SPI link.

[6] SEC-DED = Single Error Correcting (=Hamming) - Double Error Detecting (= Hamming+Parity).

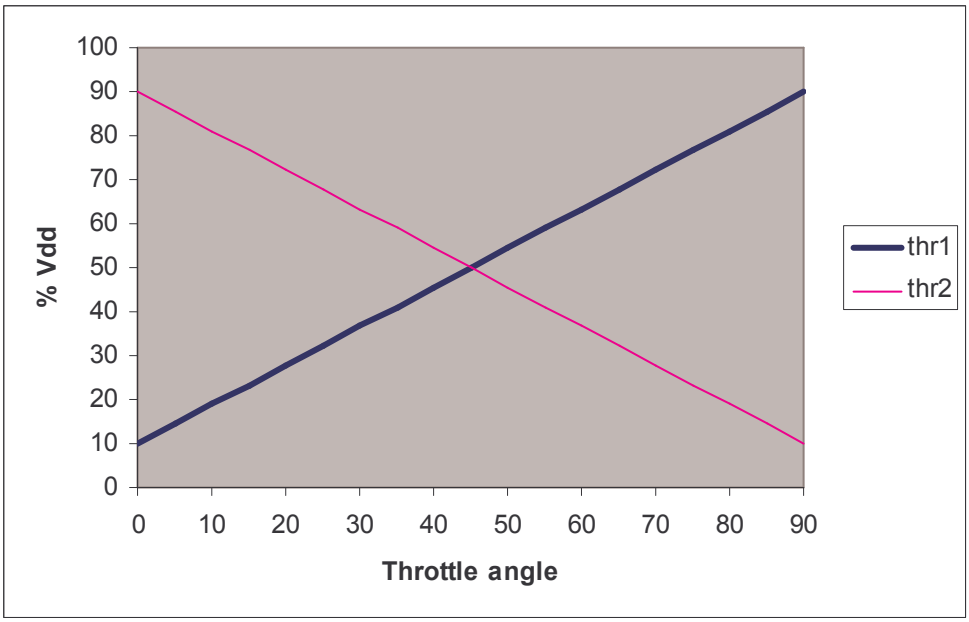

Figure 5 : the angle/voltage characteristic of a throttle sensor.

As regards critical (digital) outputs, let's take the example of the ABS valves actuation : to tolerate a stuck-at-1 of the output of the micro-controller, the ABS computer must contain an independent inhibition path, thus a second micro-controller (or ASIC).
Network critical outputs (CAN bus) are usually protected by applicative checksums and timestamps (as a supplement to the CAN 15-bit CRC), but there is one case of use of hardware redundancy : to avoid the "babbling idiot" problem (a node which speaks continuously), the second micro/ASIC can be used to switch on/off the power supply of the CAN line driver.

Now, to really stop speaking about redundancy, we will omit I/Os and focus on the safety of the computations (i.e. core + memories).

3.3 – Monitoring-based computers

The idea is the following :

Failures detection by looking at their effect on the input/output behaviour of a function (with hardware or software redundancy) is superfluous if we can detect them at the source, that is to say on the underlying hardware.

It means that to secure an application, we simply run in parallel self-tests with the good detection coverage and speed. The second micro, called the "watchdog micro" or the "intelligent watchdog", has just to keep up to date with the results of the self-tests of the main micro (his "state of health"). This **(hardware) monitoring**-based concept, used in some ABS/ESP systems, is very attractive, because it has a great practical advantage (for conceptual drawbacks, next paragraph) : the watchdog micro (and the self-tests on the main) is not application-specific, which means that

- it doesn't change during the development of the application,
- cheap ROM/OTP[7]-based 8-bits micro-controllers will do,
- if the computer must implement a new critical function, only the main micro is affected.

It can be seen as an "evolution" of the software redundancy concept (with source coding diversity), in which self-tests are so efficient that we can do without the redundant application ! The quantitative evaluation is the same, the hazardous scenario is :

- A failure occurs (hourly probability λ),
- and it is not detected by the self-tests (absolute probability p),

therefore the hourly probability of the scenario is λp. With a realistic λ equal to 10^{-7}/h (for present-day 32-bit micro-controllers), the target for p is now 10^{-2}. It means for the self-tests a 99% coverage of the failures. For permanent failures, we will see that it is not an unreachable target (detection time is still a problem). But

[7] OTP = One Time Programmable (fuse-like technology).

transient failures are absolutely not dealt with, because in redundancy solutions (HW or SW), they are only covered by the redundancy ! And as was explained earlier, transient failures gain in importance.

Coming back to permanent failures, let's try to evaluate a test coverage on a standard 32-bits micro-controller (see fig. 6 below) :

- first hypothesis : the failure rate of a part of the core is in proportion with its area (it is clearly wrong, because a RAM technology has nothing to do with a FLASH or gate technology, because we have 5V parts and 3.3V parts, …, but without additional information from the supplier, this is the only reasonable assumption to do);
- let's focus on the "numerical" core resources (memory and operations) : roughly speaking, we have :
  - ¼ of the core for the Flash,
  - ¼ of the core for the RAM,
  - 1/8 of the core for the processing unit plus the multiplier bloc.

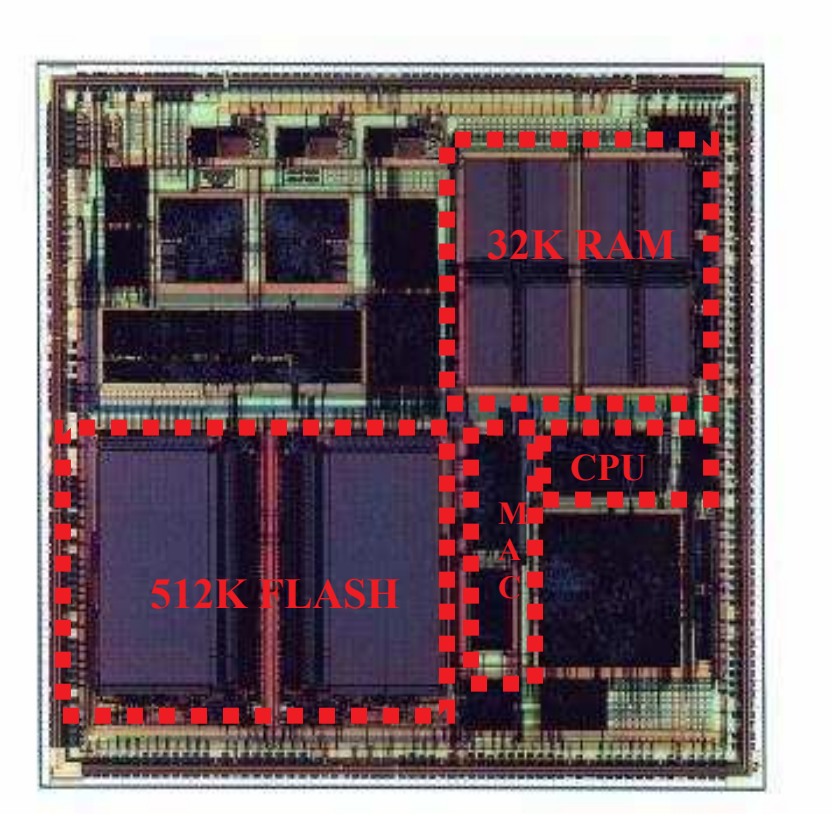

Figure 6 : a 32-bits micro-controller.

- For these 3 areas, let's imagine the tests are :
  - 16 bit CRC for the Flash : 99.998% coverage ($1-1/2^{16}$),
  - "Abraham" or "galpat" RAM test : 99% coverage (see [IEC] Table A.6),
  - instruction test developed by the core designer : 95% coverage (figure achieved on the ARM7 core with a stuck-at gate-level fault model).
- The average coverage is 98.6%, this can be considered as acceptable.

Now, the remaining problem is the detection time :

- RAM tests with a good coverage can be very time-consuming and not very easy to implement in a real-time system (because they rely on testing as a whole large blocs of memory, which means disabling interrupts for a "long" time), so pragmatic solutions can be considered, like :
  - A "serious" test ([IEC]-like) at initialization,
  - A "light" test (consecutive writing of 00/55/AA/FF in each RAM byte) periodically during operation.
- CRC computation are also time consuming (and ROM/FLASH areas grow faster than the CPU frequencies), checksums are faster but their coverage is not so important, thus we can imagine
  - A CRC verification at initialization,
  - A checksum verification periodically during operation.

3.4 – The VDA architecture.

The VDA architecture is the state of the art in the Engine Control domain : it associates the advantages of asymmetric redundancy and of monitoring, while avoiding theirs drawbacks ! The best is to have a look at the official specification ([VDA]), but to sum up the key points are :

- The main micro is associated with a watchdog micro, which is application-independent,
- Inside the main micro, the critical functions are checked by "simplified versions",
- To detect the (still possible) common effects between the critical functions and their simplified versions, these simplified versions are run periodically on fixed input sets (and their outputs are compared to the corresponding pre-computed output sets).

This architecture is a specificity of the Automotive industry, and has probably the capacity to ensure the highest safety requirements (railway and avionics). It would be very interesting to try a certification based on the safety standards of these two domains. Preliminary investigations in PSA Peugeot-Citroën with the IEC 61508 have shown that the subtleties of the VDA concept make this work very challenging.

References

[Rut84] D. Rutherford, Jr. (General Railway Signal Co.)
Fail-safe microprocessor interlocking – An application of numerically integrated safety assurance logic
Proc. IRSE 84

[JON 81] A. Jonassen, N. Siggard
Microcomputers take over the interlocking function
Railway Gazette International, dec. 1981

[Tra91] P. Traverse
Dependability of digital computers on board airplanes
Dans « Dependable computing for critical applications » (ed : A. Avizienis, J.-C. Laprie), Springer, 1991

[Ame96] A. Amendola et coll.
Architecture and safety requirements of the ACC railway interlocking system
Proc. IPDS 96

[Fre92] P. Freudenreich, G. Le Goff
Validation et homologation du système de transmission voie-machine pour le TGV Nord (TVM 430)
8ième Colloque λμ, 1992

[Vog88] U. Voges
Software diversity in computerized control systems
Springer (Dependable Computing and Fault-Tolerant Systems, vol 2), 1988

[For89] P. Forin (MATRA Transport International)
Vital coded microprocessor : principles and application for various transit systems
Proc. IFAC-CCCT, 1989

[Ham03] R. C. Hammet, P. S. Babcock
Achieving 10-9 dependability with drive-by-wire systems
SAE World Congress 2003.

[IEC] IEC 61508-2
Functional safety of E/E/PE systems / Part 2
2001

[VDA] Audi/BMW/DC/Porsche/VW
Standardisiertes E-Gas-Überwachungskonzept für Motorsteuerungen von Otto- und Dieselmotoren
16/01/2002

[Fru00] T.L. Fruehling (Delphi)
Delphi secured microcontroller architecture
SAE tech. paper 2000-01-1052