A Case Study of Toyota Unintended Acceleration and Software Safety

Prof. Phil Koopman

Carnegie Mellon University
koopman@cmu.edu
betterembsw.blogspot.com

Electrical & Computer ENGINEERING

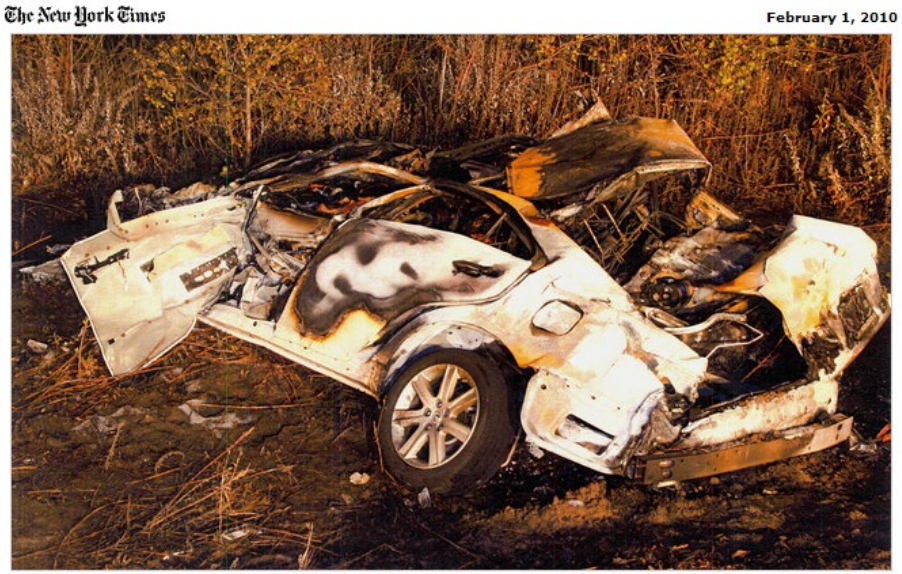
NATIONAL ROBOTICS NREC ENGINEERING CENTER

Overview

- Brief history of Toyota UA events
  - Recalls, investigations, lawsuits
  - Fines & jury awards – **$$Billions**
- Technical discussion of the problems
  - **This is a Case Study** – what can we learn?
- What does this mean for future automobiles?
  - The bar is raised, at least for now
    - E.g, handling of GM ignition switch & Honda hybrid SW UA
- I testified as a Plaintiff expert witness
  - I saw a whole lot of stuff, but not “source code”
  - I can only talk about things that are public

Aug. 28, 2009, San Diego CA, USA

- Toyota Lexus ES 350 sedan
  - UA Reached 100 mph+
- 911 Emergency Phone Call from passenger during event
  - All 4 occupants killed in crash
- Driver:
  Mark Saylor, 45 year old male.
  Off-duty California Highway Patrol Officer; vehicle inspector.
  - Crash was blamed on wrong floor mats causing pedal entrapment
  - Brake rotor damage indicated “endured braking”
- This event triggered escalation of investigations dating back to 2002 MY


The New York Times February 1, 2010

Gomez Law Firm

The wreckage of a Lexus ES 350 in which four people died in August after it accelerated out of control.

http://www.nytimes.com/2010/02/01/business/01toyota.html?pagewanted=all&_r=0
http://www.autoblog.com/2009/10/26/nhtsa-releases-new-info-about-crash-that-prompted-toyota-floorma/

Recalls & Public Discussion

(*Brakes might not mitigate open throttle* – more later)

- Floor mat recalls
  - Sept. 2007 recall to fasten floor mats
  - Wider recall Oct./Nov. 2009 after Saylor mishap
- Sticky gas pedal recall
  - Jan. 2010 and onward
- Congressional investigation
  - Toyota President testifies to US Congress, Feb. 2010
  - April 2010: Economic loss class action venue selected

http://www.cnn.com/2010/POLITICS/02/24/toyota.hearing.updates/
http://money.cnn.com/autos/storysupplement/toyota_timeline/
http://articles.latimes.com/2010/feb/18/business/la-fi-toyota-exponent18-2010feb18

May 25, 2010

Toyota "Unintended Acceleration" Has Killed 89

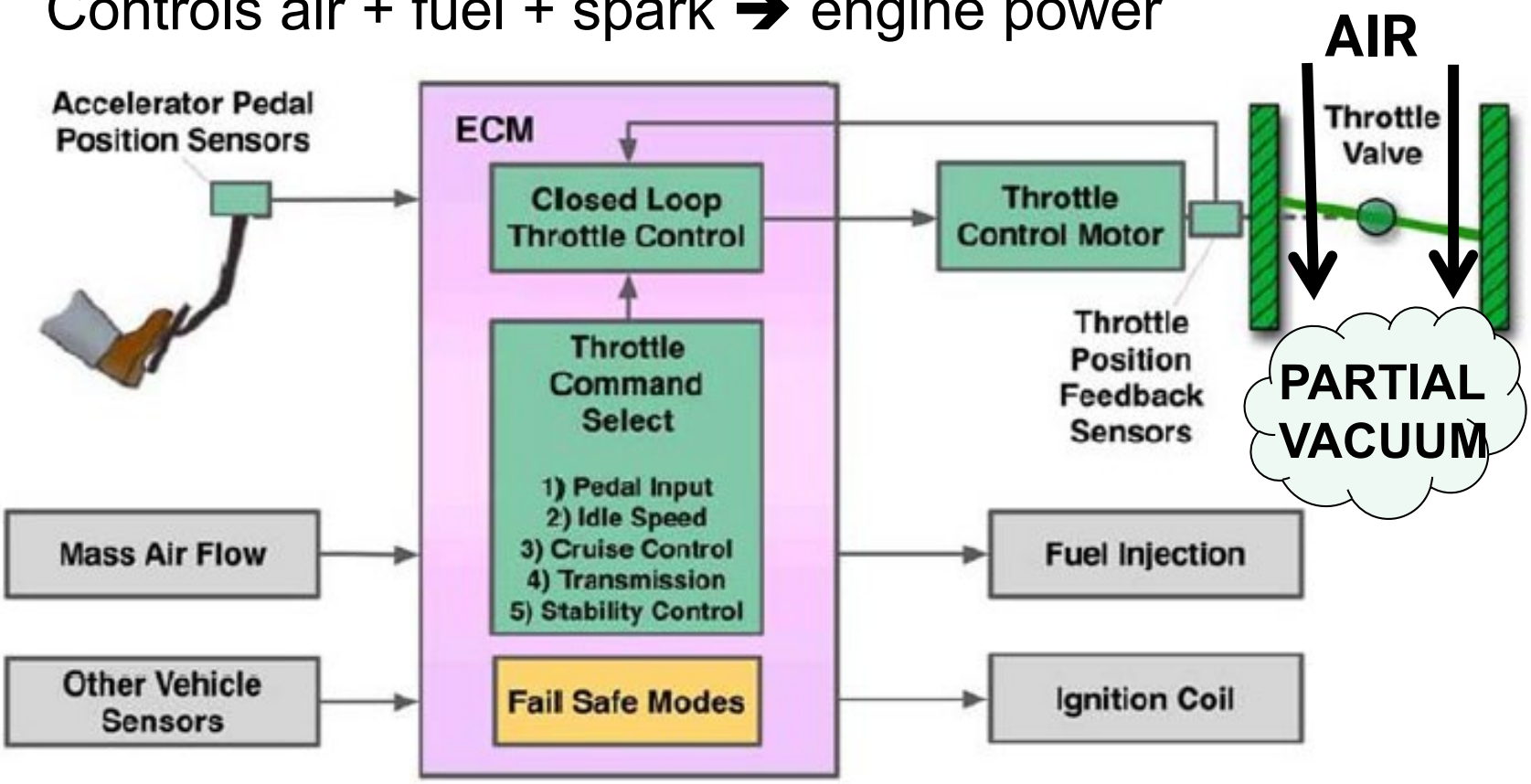
A 2005 Toyota Prius, which was in an accident, is seen at a police station in Harrison, New York, Wednesday, March 10, 2010. The driver of the Toyota Prius told police that the car accelerated on its own, then lurched down a driveway, across a road and into a stone wall. (AP Photo/Seth Wenig) / AP PHOTO/SETH WENIG

Unintended acceleration in Toyota vehicles may have been involved in the deaths of 89 people over the past decade, upgrading the number of deaths possibly linked to the massive recalls, the government said Tuesday.

The National Highway Traffic Safety Administration said that from 2000 to mid-May, it had received more than 6,200 complaints involving sudden acceleration in Toyota vehicles. The reports include 89 deaths and 57 injuries over the same period. Previously, 52 deaths had been suspected of being connected to the problem. http://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89/

NASA Investigation

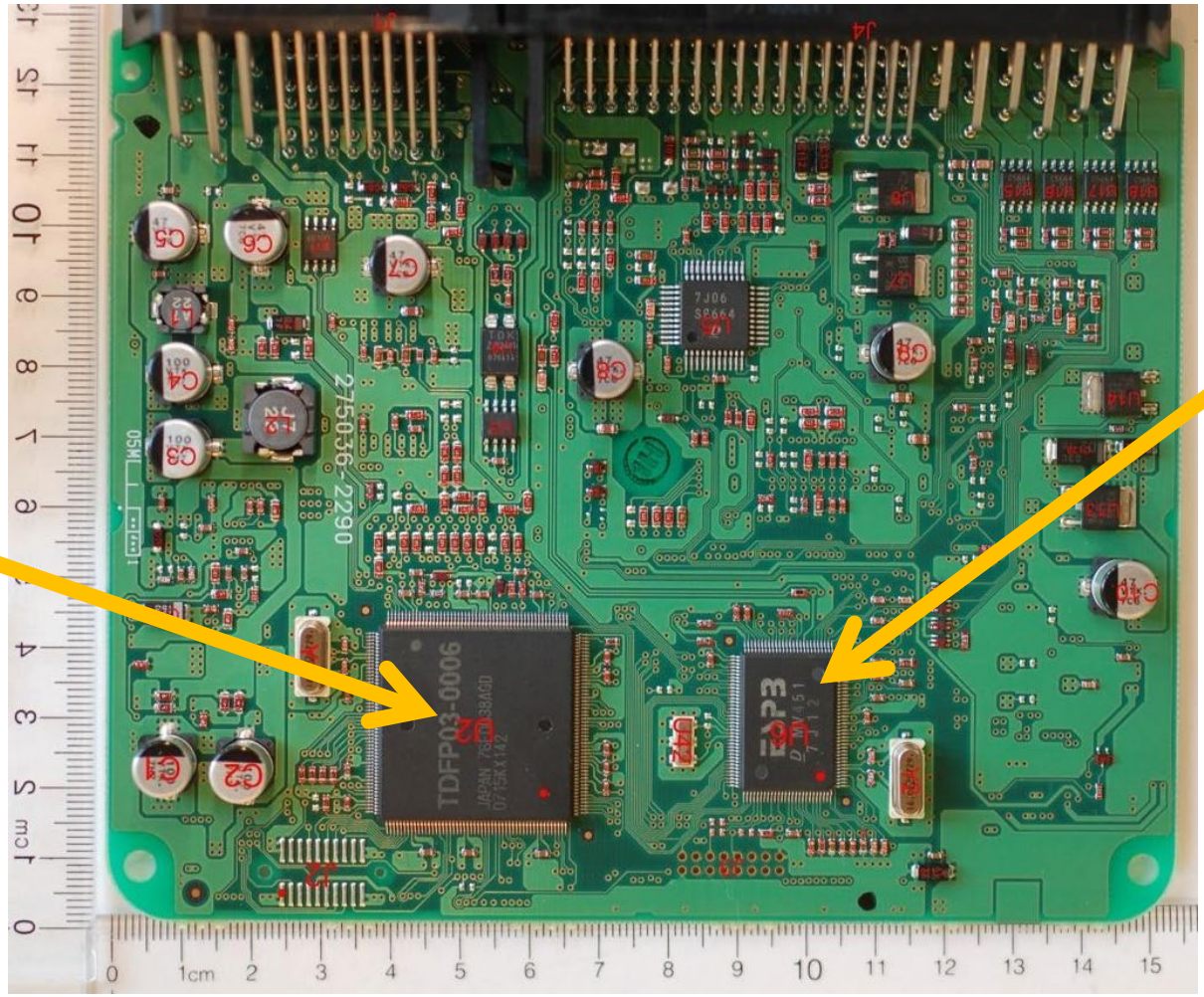
- NASA team investigates UA (2010-2011)
  - Including **Electronic Throttle Control System (ETCS)**
  - Controls air + fuel + spark ➔ engine power



[NASA UA Report Fig 4.0-1]

Toyota 2008 ETCS – Two CPUs

Main CPU (Contains Software)



Monitor Chip (Contains Software)

http://m.eet.com/media/1201063/Toyota_ECM.jpg

Toyota ETCS Is Safety Critical

- If driver pumps brakes, loses vacuum power-assist
  - With depleted vacuum, holding against WOT requires average of
    ➔ ➔ **175 pounds of force on brake pedal**
    across vehicles tested [NHTSA data]
    - With vacuum it's only 15.0 - 43.6 pounds force
      [http://www.nhtsa.gov/staticfiles/nvs/pdf/NHTSA-Toyota_vehicle_characterization.pdf]
- A software defect could command UA, for example via Wide Open Throttle (WOT)
  - **The brakes will not necessarily stop the car**
    [Consumer reports: http://www.youtube.com/watch?v=VZZNR9O3xZM]
  - **Potential** to command WOT matters for safety
    - Not just whether there is an actual bug in that does that
  - Drivers will not necessarily perform countermeasures ([NASA UA Report, p. 66]: **shift to neutral**; key-off while moving)

NASA Conclusions

- NASA didn't find a "smoking gun"
  - Tight timeline & limited information [Bookout 2013-10-14AM 39:18-40:8]
  - Did ***not*** exonerate system

Proof for the hypothesis that the ETCS-i caused the large throttle opening UAs as described in submitted VOQs could not be found with the hardware and software testing performed. …

Because proof that the ETCS-i caused the reported UAs was not found does not mean it could not occur. However, the testing and analysis described in this report did not find that TMC ETCS-i electronics are a likely cause of large throttle openings as described in the VOQs.

[NASA UA Report. Executive Summary]

- But, U.S. Transportation Secretary Ray LaHood said, "We enlisted the best and brightest engineers to study Toyota's electronics systems, and the verdict is in. There is no electronic-based cause for unintended high-speed acceleration in Toyotas."

http://www.nhtsa.gov/PR/DOT-16-11

Did NASA Have Correct & Complete Information?

- The ESP-B2 Monitor Chip has software in it
  - But NASA does not analyze ESP-B2 software in its reports – analysis is limited to Main CPU software.
- NASA credited Error Correcting Codes in RAM for 2005MY:

> The Main and Sub-CPUs use two types of memory: non-volatile ROM for software code and volatile Static Ram (SRAM). The SRAM is protected by a single error detect and correct and a double error detect hardware function performed by error detection and correction (EDAC) logic.

[NASA REPORT P. 54]

  - Apparently because Toyota told NASA it had EDAC (ECC) [Bookout 2013-10-14PM 83:19-84:25]
- Exponent public report claims ECC for Main CPU [p. 201]
  - Only claims SEC, not SECMED
- **But, actually no EDAC on RAM** for 2005MY vehicle [Bookout 2013-10-11 AM 55:22-25; 2013-10-14 AM 72:5-73:11; 2013-10-14AM 78:16-79:17]

$1.6B Economic Loss Class Action

- "Lawsuit pursues claims for breach of warranties, unjust enrichment, and violations of various state consumer protection statutes, among other claims."
  - https://www.toyotaelsettlement.com/
  - 2002 through 2010 models of Toyota vehicles
  - Toyota denies claims; settled for $1.6 Billion in Dec. 2012
  - Brake override firmware update for in some **recent** models

https://www.toyotaelsettlement.com/
24 Nov 2014

Please be advised that the Brake Override System installation is now available for the following make and model vehicles:

Toyota Models	Model Years	Deadline
4Runner	2003-2009	3/16/16
Corolla	2009-2010	8/7/15
Corolla Matrix	2009-2010	8/7/15
Highlander	2008-2010	12/11/15
Land Cruiser	2008-2010	8/7/15
RAV4	2006-2010	12/11/15
Tundra	2007-2010	3/16/16

Lexus Models	Model Years	Deadline
LX	2008-2010	8/7/15
RX	2010	8/7/15

Bookout/Schwarz Trial

[http://money.cnn.com/2013/10/25/news/companies/toyota-crash-verdict/]

- October 2013, Oklahoma
  - Fatal 2007 crash of a 2005 Toyota Camry
  - Neither floor mat nor sticky pedal recalls cover this MY; no "fixes" announced
- Toyota blamed driver error for crash
  - Mr. Arora (Exponent) testified as Toyota software expert
  - "[Toyota's counsel] theorized that Bookout mistakenly pumped the gas pedal instead of the brake, and by the time she realized her mistake and pressed the brake, it was too late to avoid the crash" [http://bigstory.ap.org/article/oklahoma-jury-considers-toyota-acceleration-case]
- Plaintiffs blamed ETCS
  - Dr. Koopman & Mr. Barr testified as software experts
  - Testified about defective safety architecture & software defects
  - 150 feet of skid marks implied open throttle while braking

IN THE DISTRICT COURT OF OKLAHOMA COUNTY
STATE OF OKLAHOMA

JEAN BOOKOUT; CHARLES SCHWARZ,
Plaintiffs,
vs.
TOYOTA MOTOR CORPORATION; and
TOYOTA MOTOR SALES, U.S.A., INC.;
Defendants.

Case No. CJ-2008-7969

[http://www.beasleyallen.com/webfiles/toyota-sua-jury-verdict-form-1.pdf (excerpts)]

VERDICT FORM – Claims of Jean Bookout

Section I – Manufacturers' Products Liability Claim

We, the jury, empaneled and sworn in the above entitled cause, do, upon our oaths, find as follows with respect to the manufacturers' products liability claim of Plaintiff Jean Bookout (check one):

X In favor of Plaintiff Jean Bookout and against Defendants.

We, the jury, empaneled and sworn in the above entitled cause, do, upon our oaths, find the dollar amount of damages sustained by Plaintiff Jean Bookout is the sum of $ 1,500,000 .

Section IV – Punitive Damages

1. We do X do not ____ (check one) find by clear and convincing evidence that Defendants acted in reckless disregard of the rights of Plaintiff, Jean Bookout.

Toyota Case: Single Bit Flip That Killed

Junko Yoshida

10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including bugs that can cause unintended acceleration.

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they "uncovered gaps and defects in the throttle fail safes."

The experts demonstrated that "the defects we found were linked to unintended acceleration through vehicle testing," Barr said. "We also obtained and reviewed the source code for the black box and found that it can record false information about the driver's actions in the final seconds before a crash."

Stack overflow and software bugs led to memory corruption, he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of tasks' deaths studied by the experts in their experiments "were not detected by any fail safe."

Bookout Trial Reporting

http://www.eetimes.com/document.asp?doc_id=1319903&page_number=1 (excerpts)

"Task X death in combination with other task deaths"

The Bookout/Schwarz Results

- Jury awarded $3 million compensation
  - Key point in trial was whether ETCS design defects caused the fatal crash
    - To this day, Toyota disputes that their ETCS is flawed
  - $1.5M each to Bookout and Schwarz estate
    - Toyota settled before jury could consider awarding ***additional***, punitive damages
- Subsequent Federal trials put on hold
  - Only ETCS software/safety case to actually go to trial
    - Remaining Federal trials deferred
  - Mass settlements proceeding during 2014
    - Hundreds of cases pending being settled as of summer 2014

US Criminal Investigation

- "Toyota Is Fined **$1.2 Billion** for Concealing Safety Defects" – March 19, 2014
  - Four-year investigation by US Attorney General
  - Related to floor mats & sticky throttle pedals only
- "TOYOTA misled U.S. consumers by concealing and making deceptive statements about two safety-related issues affecting its vehicles, each of which caused a type of unintended acceleration." [DoJ Statement of Facts]
  - Deferred prosecution for three years in exchange for fine and continuing independent review of its safety processes.
  - Toyota said in a statement that it had made fundamental changes in its corporate structure and internal safety controls since the government started its investigation four years ago.

http://www.nytimes.com/2014/03/20/business/toyota-reaches-1-2-billion-settlement-in-criminal-inquiry.html

The Technical Point of View

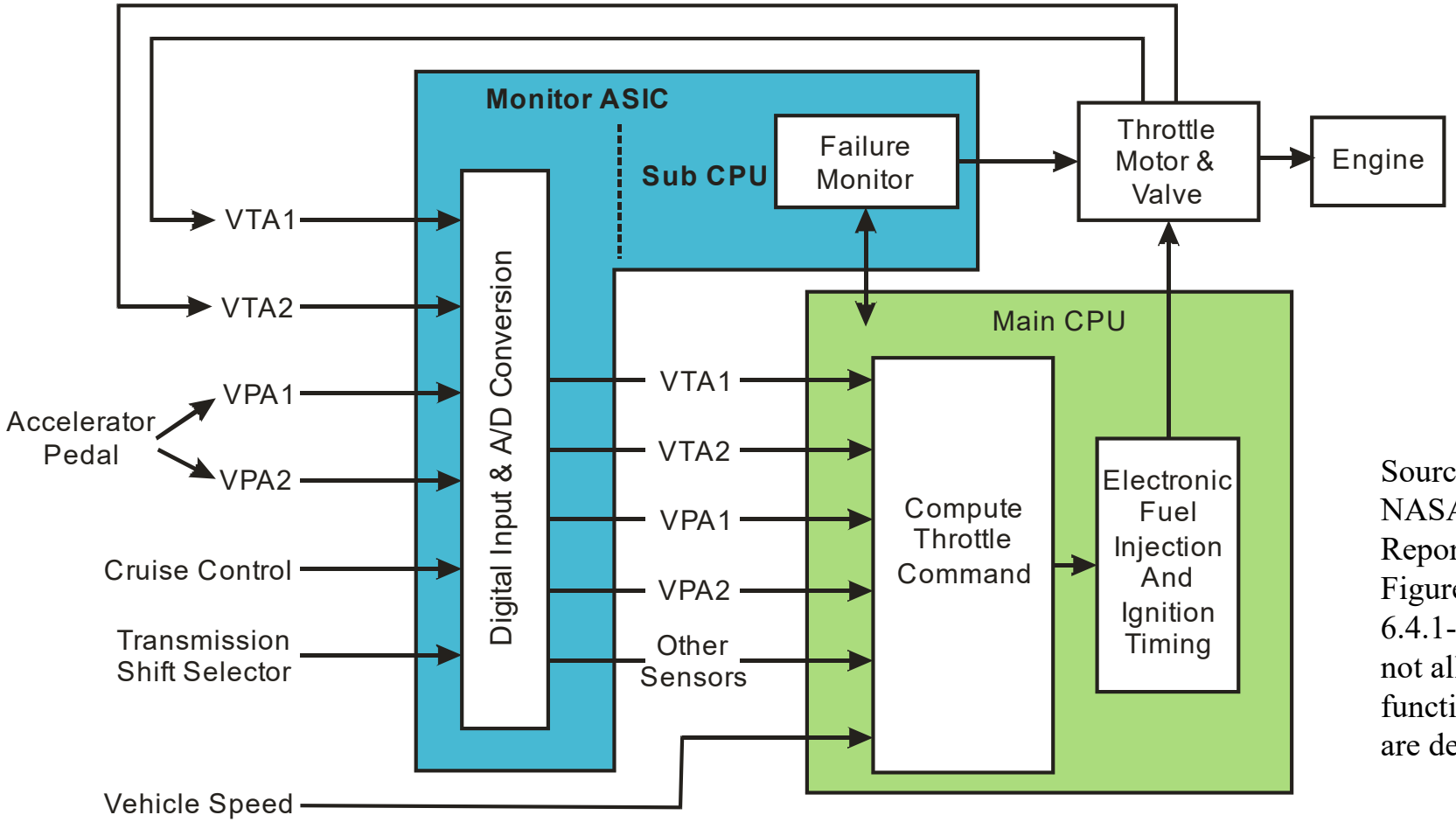
- NASA didn't find a smoking gun, **but**…
  - They found plenty that is technically questionable
  - It was a difficult assignment with limited time & resources
- Jury found that ETCS defects caused a death
  - Experts testified ETCS is unsafe ..

    .. but jury is non-technical
- So……..let's consider public information and

  you can decide if ETCS is safe for yourself
  - Consider accepted practices circa 2002 MY vehicles
  - UA ➔ loss of command authority over the throttle
  - Consider if "reasonable care" was used
  - Standard of evidence is "more likely than not"

ETCS Architecture (simplified)

256.6K Non-Comment Lines C Source + 39.5K NCSL headers (Main CPU) + Proprietary Monitor Chip software [NASA App. A p. 21]

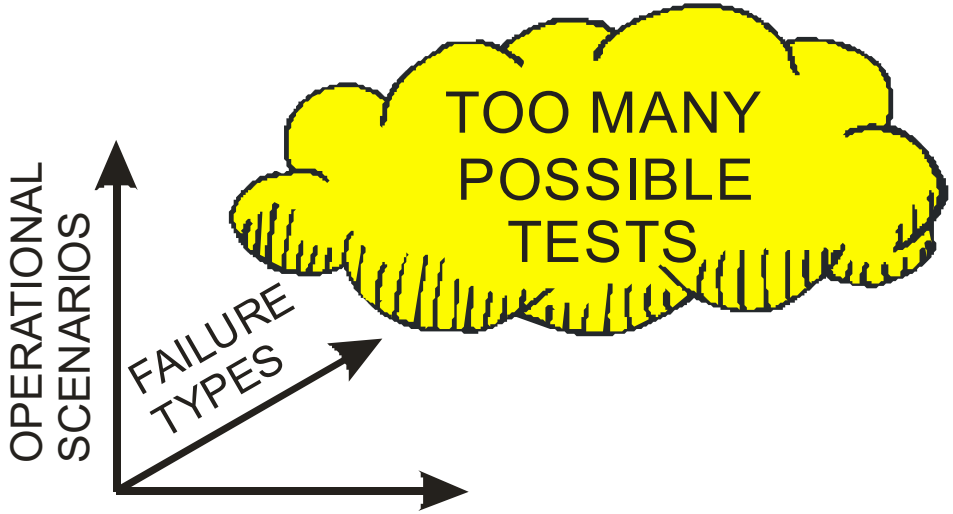


Source: NASA UA Report Figure 6.4.1-1; not all functions are depicted

VTA: Throttle Position
VPA: Accelerator Pedal Position

Didn't Vehicle Testing Make It Safe?

- Vehicle level testing is useful and important
  - Can find unexpected component interactions
- But, it is impracticable to test everything at the vehicle level
  - Too many possible operating conditions, timing sequences
  - Too many possible faults, which might be intermittent
    - Combinations of component failures + memory corruption patterns
    - Multiple software defects activated by a sequence of operations



Testing Is Not Enough To Establish Safety

- Toyota tested about 35 million miles at system level
  - Plus 11 million hours module level software testing
    ([NASA report p. 20], covering 2005-2010 period)
  - In 2010 Toyota sold 2.1 million vehicles [Toyota annual report]
- Total testing is perhaps 1-2 hours per vehicle produced
  - Fleet will see thousands of times more field exposure
  - Vehicle testing simply can't find all uncommon failures

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 19, NO. 1, JANUARY 1993

The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software

Ricky W. Butler and George B. Finelli

life-testing of ultrareliable software is infeasible (i.e., to quantify 10^{-8}/h failure rate requires more than 10^{8} h of testing),

[Butler 1993, p. 10]

Safety Integrity Level (SIL) Approach

- SILs form different "bins" for levels of required safety
  - Based on effects of the fault if not properly mitigated
- Poor Controllability dictates a low Acceptable Failure Rate
  - Acceptable Failure Rate determines which techniques required to achieve a correspondingly rigorous Integrity Level (SIL)

Controllability Category	Acceptable Failure Rate	Integrity Level
Uncontrollable	Extremely improbable	4
Difficult to control	Very remote	3
Debilitating	Remote	2
Distracting	Unlikely	1
Nuisance only	Reasonably possible	0

[MISRA SW p. 18, annotated]

Safety Integrity Level Approaches Are Common

Standard	Domain; Year	Safety Integrity Levels (SILs)
MISRA Software Guidelines **[MISRA Guidelines p. 17]**	Automotive; 1994	**SIL 1 (lowest) .. SIL 4(highest)**
IEC 61508 **[IEC 61508-1, p. 34]**	Process Control; 1998	**SIL 1 (lowest) … SIL 4(highest)**
ISO 26262 **[ISO 26262-3, pg. 10]**	Automotive; 2011	**ASIL-A (lowest) .. ASIL-D (highest)**
CENELEC EN-50128/9 **[EN 50128 p. 12]**	Rail; 1997/1998	**SIL 1 (lowest) … SIL 4(highest)**
FDA **[FDA 1998, pg. 8]**	Medical; 1998	**Minor, Moderate, Major**
NASA NPG 8715.3 **[Herrmann 1999, pg. 151]**	Spacecraft; 1996/1997	**Negligible, Moderate, Critical, Catastrophic**
FAA Do-178b **[Do-178b p. 7]**	Aircraft; 1992	**Level D (minor) .. Level A (catastrophic)**
MIL-STD-882D **[MIL-STD-882D p. 18]**	US Combat Systems; 1977	**IV (negligible) .. I (catastrophic)** (severity; maps to levels 1-20 of risk)

An Accepted Practice for Safety Critical SW

- SIL approach:
  - Determine SIL based on failure severity
  - Follow SIL-appropriate development process
  - Follow SIL-appropriate technical practices
  - Follow SIL-appropriate validation practices
  - Make sure process is really working (SQA)

BASED ON SIL:
☑ DEVELOPMENT
☑ TECHNICAL
☑ VALIDATION
☑ PROCESS QUALITY

- This includes:
  - "Near-perfect" software
  - Design out single points of failure (per appropriate fault model)
  - Justify real time scheduling with analysis
  - Watchdog timers that have real "bite"
  - Good software architecture
  - Good safety culture

MISRA SW Guidelines Required Practices

The ETCS likely qualifies as MISRA SIL 3 – and this is what it takes:

Development Process	Integrity Level				[MISRA SW, 1995 p. 21]
	0	1	2	3	4
Specification and design	ISO 9001	Structured method.	Structured method supported by CASE tool.	Formal specification for those functions at this level.	Formal specification of complete system. Automated code generation (when available).
Languages and compilers		Standardized structured language.	A restricted subset of a standardized structured language. Validated or tested compilers (if available).	As for 2.	Independently certified compilers with proven formal syntax and semantics (when available).
Configuration management: products		All software products. Source code.	Relationships between all software products. All tools.	As for 2.	As for 2.
Configuration management: processes		Unique identification. Product matches documentation. Access control. Authorized changes.	Control and audit changes. Confirmation process.	Automated change and build control. Automated confirmation process.	As for 3.

SIL 3 requires all SIL 1 + SIL 2 + SIL 3 activities

Electrical & Computer ENGINEERING

More MISRA Required Practices

Development Process	Integrity Level				
	0	1	2	3	4
Testing		Show fitness for purpose. Test all safety requirements. Repeatable test plan.	Black box testing.	White box module testing — defined coverage. Stress testing against deadlock. Syntactic static analysis.	100% white box module testing. 100% requirements testing. 100% integration testing. Semantic static analysis.
Verification and validation		Show tests: are suitable; have been performed; are acceptable; exercise safety features. Traceable correction.	Structured program review. Show no new faults after corrections.	Automated static analysis. Proof (argument) of safety properties. Analysis for lack of deadlock. Justify test coverage. Show tests have been suitable.	All tools to be formally validated (when available). Proof (argument) of code against specification. Proof (argument) for lack of deadlock. Show object code reflects source code.
Access for assessment		Requirements and acceptance criteria. QA and product plans. Training policy. System test results.	Design documents. Software test results. Training structure.	Techniques, processes, tools. Witness testing. Adequate training. Code.	Full access to all stages and processes.

[MISRA SW, 1995 p. 21]

What's The Required Level of Rigor?

- No certification requirement for US cars
  - US standards generally insufficient for software safety
    - FMVSS = Federal Motor Vehicle Safety Standards don't address typical software safety topics
  - US DoT can require recalls and otherwise enforce if safety problems are detected in field

- Legal standards vary
  - Generally can't be **"unreasonably dangerous"** when used for intended purpose
  - Auto makers not required to follow MISRA guidelines
    - But, it was available if they wanted to follow it
    - 2002 was first model year in economic Class lawsuit; ETCS partial redesign for 2005MY [Bookout 2013-10-15AM 32:4-34:10]

What Is The Toyota Level of Rigor?

- Toyota does not claim to have followed MISRA Guidelines
  - (Note that MISRA Guidelines >> MISRA C)
  - NASA did not disclose an auditable software process plan
  - NASA did not disclose a written safety argument from Toyota

- Toyota's expert in Bookout trial offered two basic opinions
  - No "realistic" ETCS fault that explains/caused Bookout mishap
  - Any "realistic" failure will be caught and mitigated by failsafes

[Bookout 2013-10-22PM 47:3-48:1]

- Exponent "public report" basically argues the same things:
  - Same-fault-containment-region fail-safes will mitigate UA
  - Couldn't find a "realistic" fault scenario for unmitigated UA
  - Couldn't find a system-level test that produces unmitigated UA

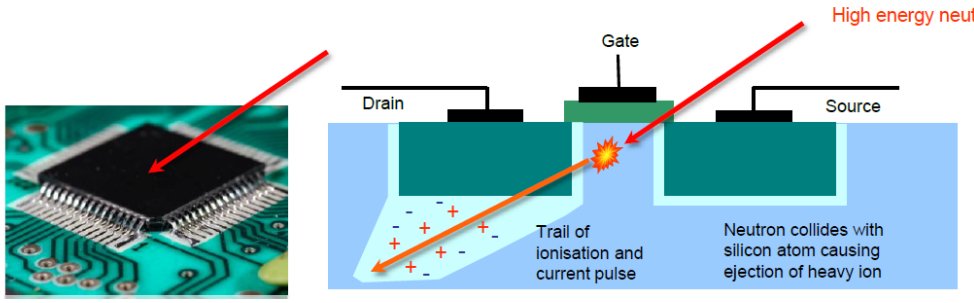
Example Failsafe: Brake Echo Check

- "Brake echo check" is an ETCS failsafe
  - Echo-back brake pedal state from Main to Monitor CPU
- May detect some Task X deaths after a UA
  - BUT, requires a "brake pedal" **transition**
- Thus:
  - If your foot is already on the brake
  - And then a UA event occurs
  - You may have to **completely take foot off the brake** to trigger the failsafe (tail lights must turn off)
    - If you pump brakes without a *complete lift* of your foot off the brake pedal, failsafe is not activated

[Bookout 2013-10-11PM 96:4-11; 2013-10-14PM 16:25-17:22; 74:1-23; 2013-10-14PM 102:16-103:25]

Random HW Faults & Safe Systems

A Single Event Effect (SEE) is when a highly energetic particle (neutron), present in the environment, strikes sensitive regions of an electronic device disrupting its correct operation



Radiation strike causing transistor disruption [Gorini 2012].

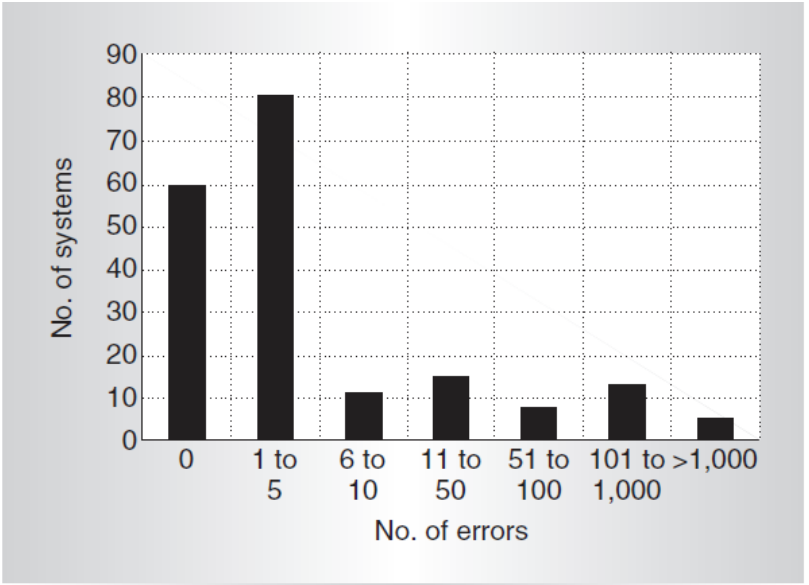


Figure 2. Histogram of the number of memory single-bit errors reported by 193 systems over 16 months.

[Constantinescu 2003, p. 16]

- Hardware (HW) bits flip values due to radiation strikes ("soft errors")
  - Affects memory, control logic, CPU registers – everything on a chip
  - "soft errors must be taken into account" for drive-by-wire automotive components [Mariani 2003, p. 50]
- Software defects can also corrupt memory
  - Result of corruption can be "incorrect output" – not just SW crash

[Sullivan 1991, pp. 6, 8]

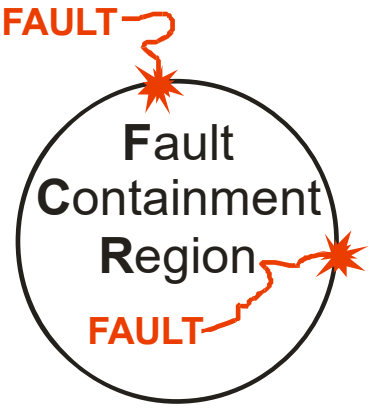
How Often Do Random Faults Happen?

- HW faults every 10,000 to 100,000 hours per chip
  - Perhaps 2% are dangerous ➔ 0.2 times per million hrs
    [Data from Obermaisser pp. 8,10, using the favorable number here]
- HW Example: ~430,000 Camry vehicles built/year
  - US cars driven about 1 hour per day x 365 days/year
    - That's 365 * 430,000 hrs = 156.95 million hours / year
    - At 0.2 dangerous faults per million hours ➔ 31 / year for fleet
  - One dangerous fault every 11.6 days at the low end – per chip – for just one model year
    - Approximate numbers – the point is they will happen
- Software faults will only make it worse
- Every large deployed fleet suffers HW & SW faults
  - ETCS fail-safes catch some – but not all – of these faults

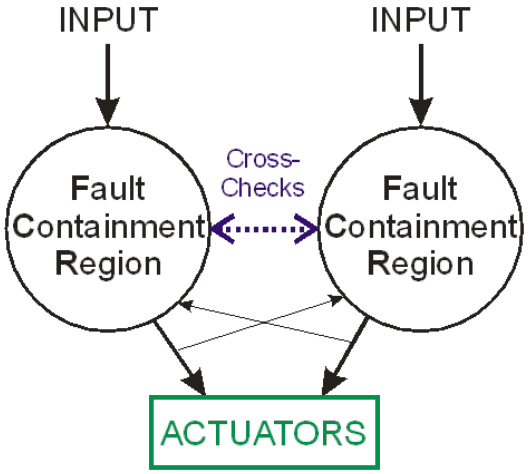
(Note: yearly production is approximate)

Redundancy Required For Critical Systems

- Fault Containment Region (FCR)
  - FCR provides a fault "firewall"

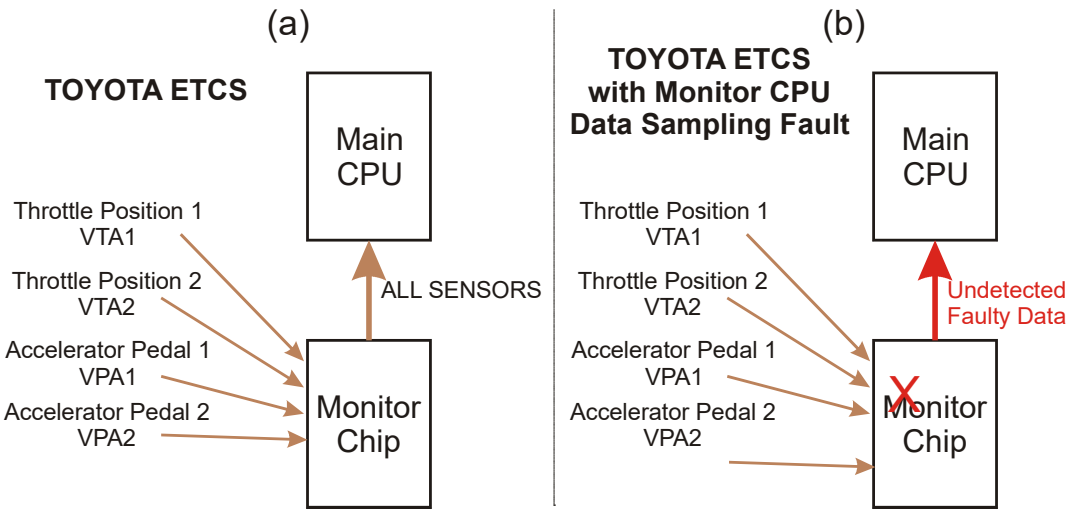


- Need **independent** FCRs to detect faults
  - Single FCR can't self-police its own failure modes
    - Self-test isn't enough for safety
  - Shared resources form dangerous Single Points of Failure



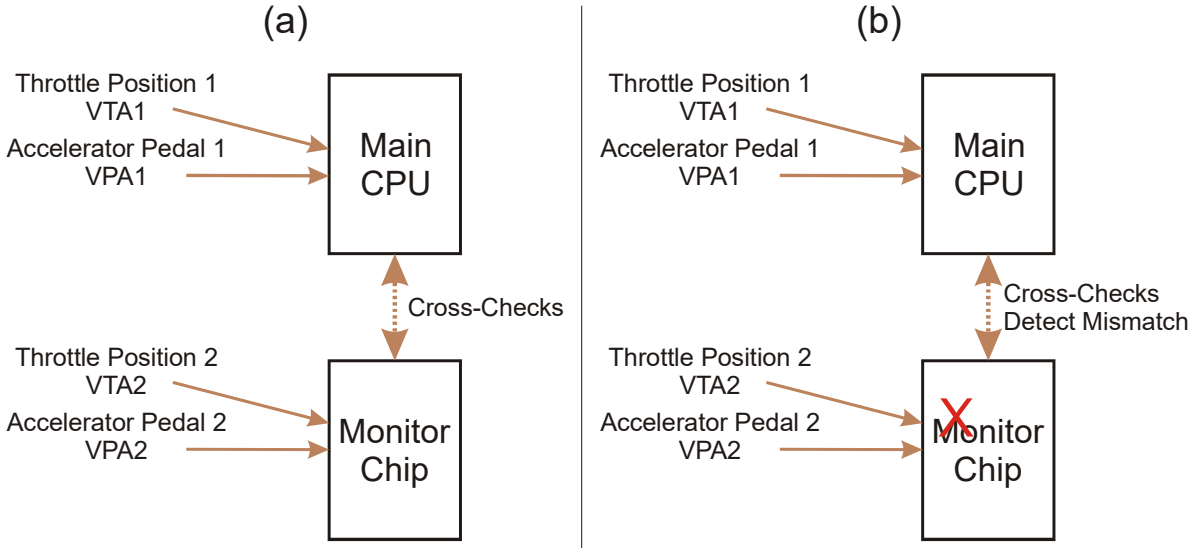
- Complete protection requires redundancy
  - Independent observations of input values
  - You can't trust another FCR – what if it gives you bad data?

Safe Dual Processors Split Input Sources



Sharing of inputs permits undetected faulty data to reach main CPU

Alternate design uses separate A/D inputs to cross-check data

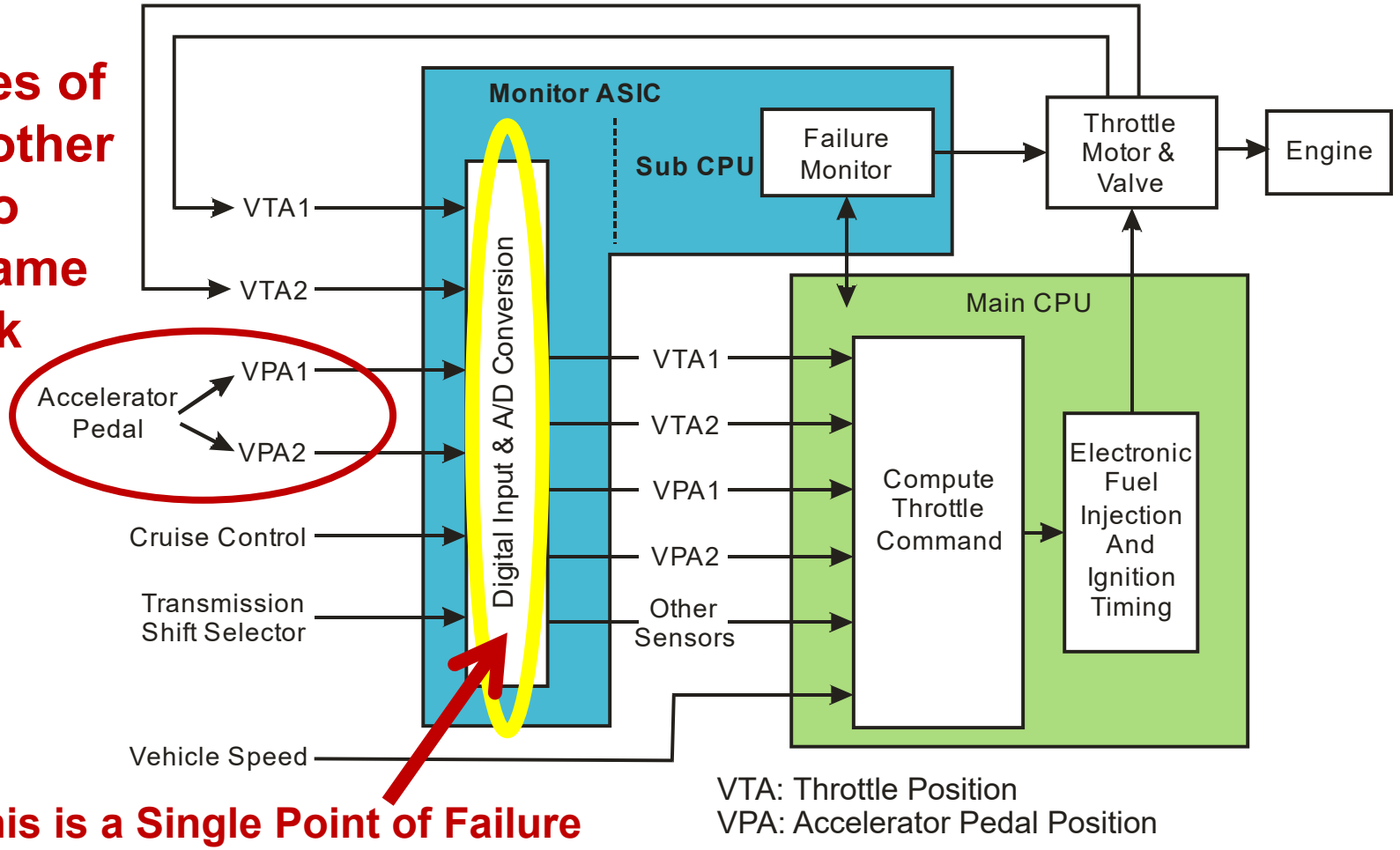


Electrical & Computer ENGINEERING

Redundant Accelerator Position Signals (VPA1/VPA2)

- Safe architectures do not have single points of failure

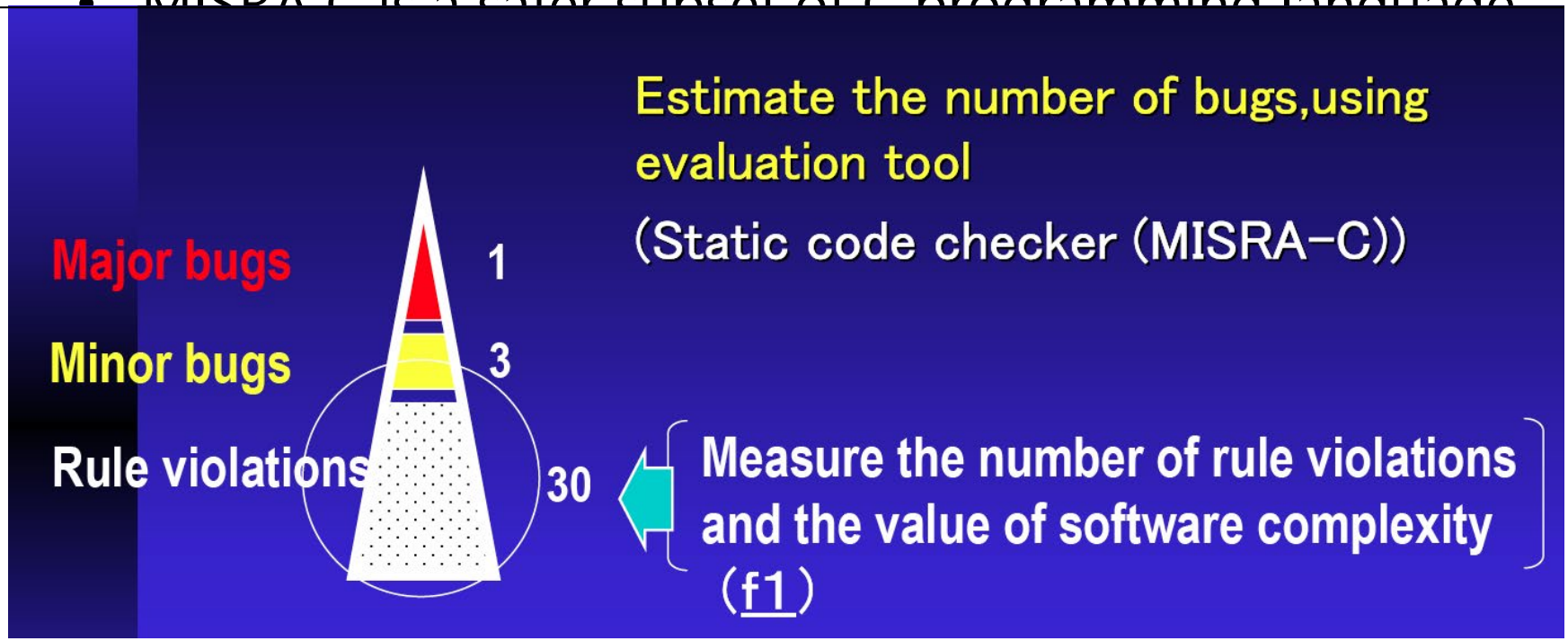
Both copies of VPA (and other signals) go through same input block on the same chip



[Bookout 2013-10-11AM 63:21-64:14]

What About Software Bugs?

- For example what if MISRA C rules are violated?
  - MISRA C is a safer subset of C programming language



Toyota data on infotainment software shows an expected **one "major bug" for every 30 coding rule violations** [Kawana 2004].

What Is Toyota Source Code Quality?

- Coding style: rules for code formatting & language use
  - Toyota claimed 50% MISRA C overlap for coding rules
    - But only 11 MISRA C rules out of MISRA C in the Toyota coding rules
  - Toyota did not always follow its own coding rules
    - For example, 105 out of 343 "switch" keywords without "default" [e.g., NASA App. A pp. 21-23]
  - Reason given for not using MISRA C rules for 2002 MY: Its coding **rules** pre-date 1998 MISRA C [NASA App A p. 28]

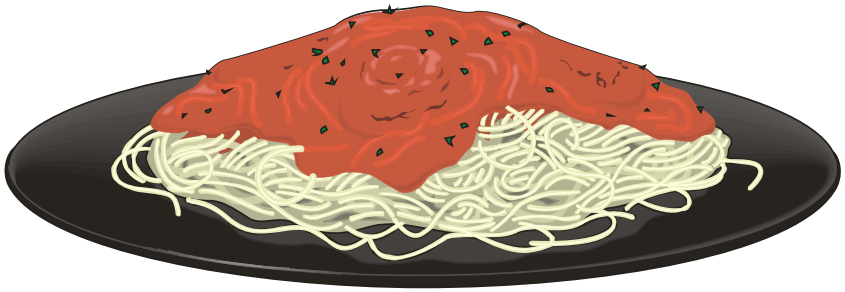
- 35 MISRA C rules that NASA tools could readily check:
  - 14 of 35 rules violated; 7,134 violations
    - Mostly macro use and use of #undef [NASA App. A. p. 29]
- Mike Barr's team found 80,000 violations of MISRA C [Bookout 2013-10-14PM 44:19-45:2]

NASA ETCS Static Analysis Results

- NASA used several static analysis tools [NASA App. A, pp. 25-31]
  - Toyota did not claim warning-free on these, but results informative
- Coverity:
  - 97 - declared but not referenced
  - 5 - include recursion
- Codesonar:
  - **2272 - global variable declared with different types**
  - 333 - cast alters value
  - 99 - condition contains side-effect
  - 64 - multiple declaration of a global
  - **22 - uninitialized variable**
- Uno:
  - **89 - possibly uninitialized variable**
  - 2 - Array of 16 bytes initialized with 17 bytes

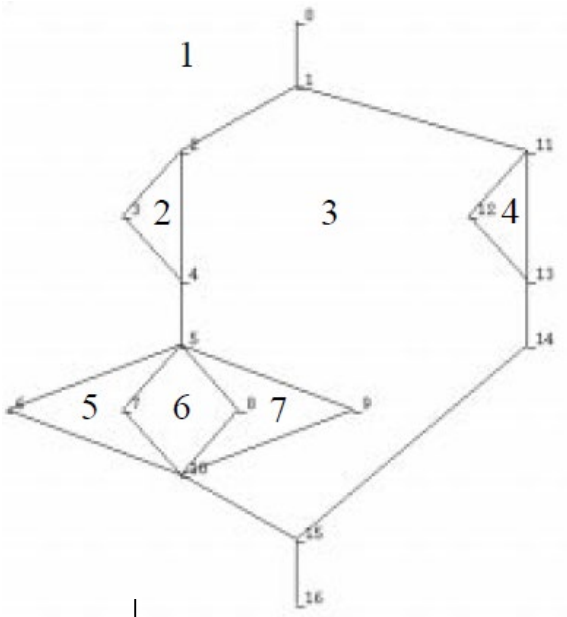
Code Complexity

"Spaghetti code":
Incomprehensible code due to unnecessary coupling, jumps, gotos, or high complexity



- McCabe Cyclomatic Complexity metric
  - Number of "eyes" in flow control graph
  - Unit tests harder with complex graph
    - Over 50 is considered "untestable"
- Toyota ETCS code:
  - 67 functions with complexity over 50
  - **Throttle angle function complexity = 146;** 1300 lines long, no unit test plan

[Bookout 2013-10-14 31:10-32:23; 32:15-23]



Complexity=7

[NIST 500-235, 1996, pp. 28-29]

As the number of branches in the module or program rises, the cyclomatic complexity metric rises too. Empirically, numbers less than ten imply reasonable structure, numbers higher than 30 are of questionable structure. Very high cyclomatic numbers of more than 50 imply the application cannot be tested, while even higher numbers of more than 75 imply that every change may trigger a "bad fix". This metric is widely used for Quality Assurance and test planning purposes.

[RAC 1996, p.124]

Global Variables Are Evil

- Global variables can be read/written from any system module
  - In contrast, local variables only seen from a particular software module
- Excessive use of globals tends to compromise modularity
  - Changes to code in one place affect other parts of code via the globals
  - Think of it as **data flow spaghetti**

1973 February

GLOBAL VARIABLE CONSIDERED HARMFUL

W. Wulf, Mary Shaw
Carnegie-Mellon University

• •

The problems of indiscriminant access and vulnerability are complementary: the former reflects the fact that the declaror has no control over who uses his variables; the latter reflects the fact that the program itself has no control over which variables it operates on. Both problems force upon the programmer the need for a detailed global knowledge of the program which is not consistent with his human limitations.

[Wulf 1973, pp. 28,32]

Toyota Global Variable Use

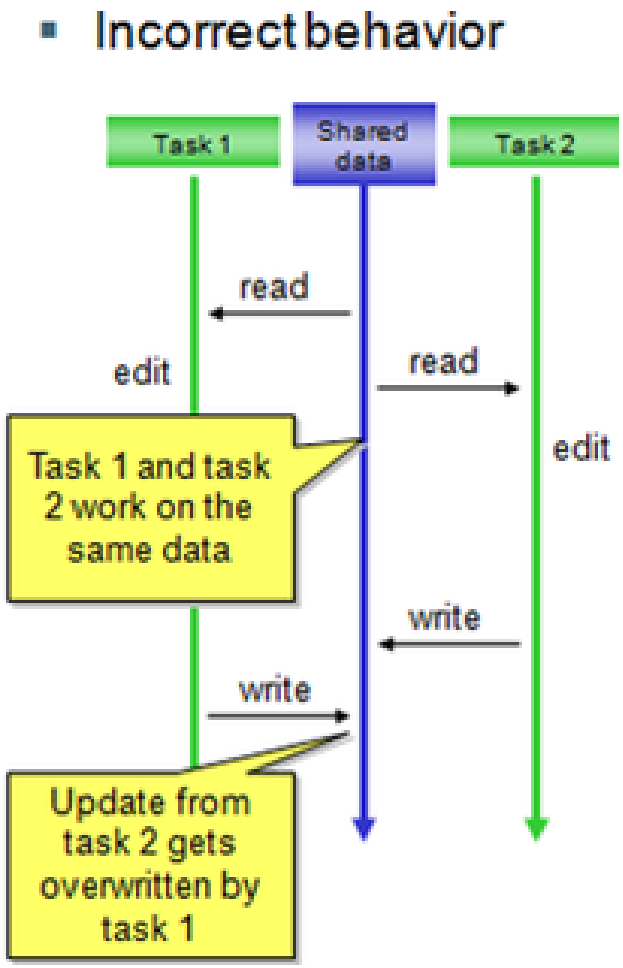
- Ideal number of writeable globals is ZERO
  - OK to have moderate "const" values and configuration data:
  - Toyota code has: [NASA App. A p. 33]:
    - 4,720 read-only & text variables
    - 11,253 read/write variables
- ETCS globals command throttle angle, report engine speed
  [Bookout 2013-10-14 PM 29:4-15]
- **Toyota: 9,273 – 11,528 global variables**
  [NASA App. A pp. 34, 37]
  - "In the Camry software a majority of all data objects (82%) is declared with unlimited scope and accessible to all executing tasks."
    [NASA App. A, pg. 33]
  - NASA analysis revealed: [NASA App. A, pg. 30]
    - 6,971 instances in which scope *could be* "local static"
    - 1,086 instances in which scope *could be* "file static"

* Various counts differ due to use of different analysis tools with slightly different counting rules

Concurrency Bugs / Race Conditions

- One CPU can have many tasks (e.g., with Real Time Operating System)
  - Tasks take turns, sharing the CPU and memory ("multi-tasking")

- Concurrency defects often come from incorrect data sharing
  - One way to fix this is to **disable interrupts** before reading to ensure one task reads/writes at a time
  - Defects may be due to subtle timing differences, and are often **difficult to reproduce**



[Wind River]

ETCS Concurrency Issues

- NASA identified a specific concurrency defect

> This rule is based on the fact that equal priority tasks cannot interrupt each other. Both can still be interrupted by a higher priority task. If because of this interruption the second task does not complete, and the first task restarts in the next time interval, it could still overwrite the result of the interrupted second task.

[NASA App. A pp. 33-34]

- Nested scheduler unlocks [Bookout 2013-10-14PM 21:10-22:1]
- Shared global variables not all "volatile" [NASA App. A pp. 33-34]
- Shared globals not always access with interrupts masked

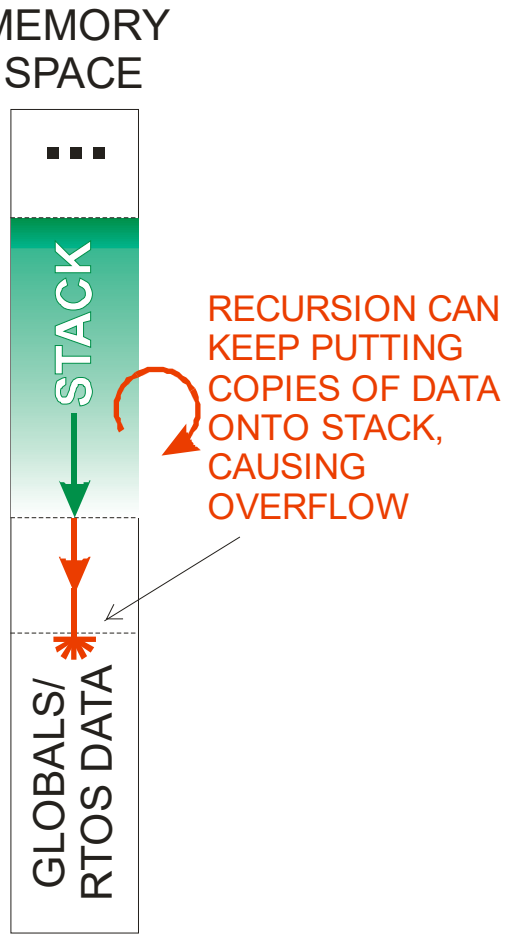
[NASA App. A pp. 33-34]

> If two tasks running at different priority levels access the same data, then the lower priority task must use interrupt masking to protect against interference from the higher-priority task.
>
> This rule is not always followed in the code. In a few cases, the lower priority task merely sets a flag before entering its critical section and the higher priority task checks this flag before accessing the same data.
>
> There are cases in the code where tasks of different priority levels (e.g., levels 14, 12, and 4) access the same global variables without using interrupt masks (pattern: read, store local copy, update, write new value). An example is the use of variable *s2s_eafsfb_gaind*, which is declared as a *non*-volatile static variable. This use would appear to be in violation of coding rule 651.
>
> . . .
>
> Despite the rigor in the use of the *volatile* qualifier on *constant* data, other shared global variables are not always declared *volatile*. The team counted **11,528** non-constant, shared global variables in the code.
>
> There are only **865** uses of interrupt masking in the code, in **194** different source files. This indicates that access to global variables is not always done under protection of interrupt masks.

Electrical & Computer ENGINEERING

Toyota ETCS and Recursion

- Safety rules typically **forbid recursion**
  - Risk of stack overflow
  - Recursion makes it impossible to do the V&V necessary for a system like the ETCS [NASA APP A. p. 129]
- Toyota ETCS uses recursion
  - Stack is 94% full PLUS any recursion [Bookout 2013-10-14PM 35:7-24]
- No mitigation for stack overflow
  - Incorrect assumption that overflow *always* results in a system reset [NASA APP A pg. 130]
  - Memory just past stack is **OSEK RTOS area** [Bookout 2013-10-14 PM p. 39:1-5]



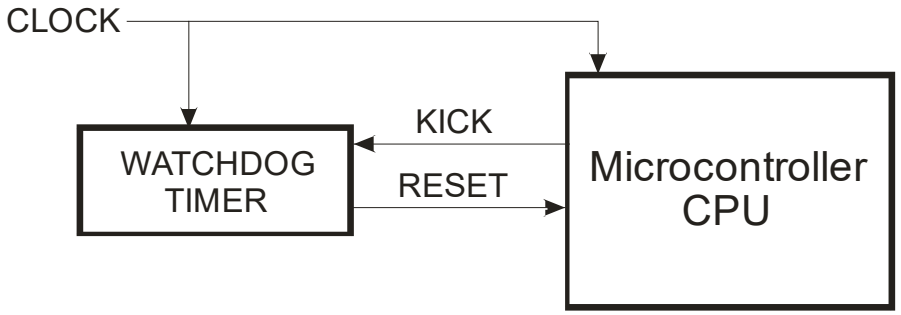
Rule 70 (required): **Functions shall not call themselves, either directly or indirectly.**

This means that recursive function calls cannot be used in safety-related systems. Recursion carries with it the danger of exceeding available stack space, which can be a serious error. Unless recursion is very tightly controlled, it is not possible to determine before execution what the worst case stack usage could be.

[MISRA C, page 43]

Watchdog Timers Must Detect Task Deaths

- Watchdog Operation:
  - If count-down timer reaches zero, it resets the CPU
  - CPU periodically "kicks" watchdog to indicate it is awake, replenishing the counter
- Proper watchdog use includes:
  - Kicking from timer tick interrupt service routine usually a bad sign
  - **Any single task death must inhibit watchdog kick**
    - That means checking all tasks for liveness before kicking [e.g., Lantrip 1997]



[Koopman 2010, p. 335]

I've seen some multitasking systems that use an interrupt to tickle the watchdog. This approach defeats the whole purpose for having one in the first place. If all the tasks were blocked and unable to run, the interrupt method would continue to service the watchdog and the reset would never occur.

[Brown 1998, p. 46]

ETCS Watchdog Operation

- Toyota ETCS detects if CPU load too high
  [NASA App. A p. 18]
  - (Note that if a task dies, that ***decreases*** CPU load)

- But…
  - System ignores RTOS error codes (e.g., task death)
  - Watchdog kicked by a hardware timer service routine
  - Watchdog does not detect death of major tasks, including "Task X"
    - Task X includes throttle angle calculation & most failsafes
    - Task X sets most Diagnostic Trouble Codes (DTCs)
    - Watchdog only detects death of the 1 msec task, not others

[Bookout 2013-10-14PM 70:10-72:23; 81:5-11; 105:14-21]

Safety Culture

- "No knowledge at Toyota" for some parts of the "V" process (e.g., module inspections)
  - No independent certification for parts they couldn't/didn't check

    [Bookout 2013-10-11 PM 32:22-33:19]

- Example of Toyota's UA investigation philosophy:
  - "In the Toyota system, we have the failsafe, so a software abnormality would not be involved with any kind of UA claim."
    (Employee tasked with examining vehicles with reported UA problems)

    [Bookout 2013-10-11 PM 35:6-21]

- 2007 e-mail internal Toyota e-mail says:
  - "In truth technology such as failsafe is not part of the Toyota's engineering division's DNA." …
    … "Continuing on as is would not be a good thing."

    [Bookout 2013-10-14 PM 96:20-97:3]

Other Issues

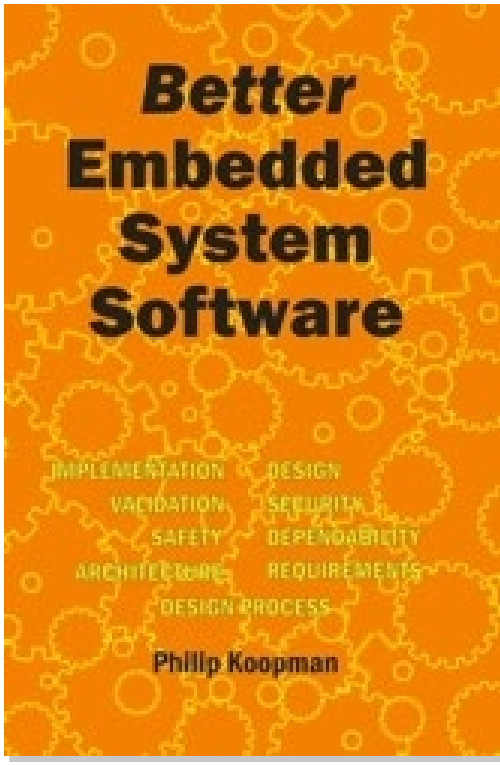
- Poor isolation of task functions
  - "Kitchen Sink" "Task X" both computes throttle angle AND is responsible for many of the failsafes (same CPU, same task). Brake Override function in 2010 MY Camry is in this same task. [Bookout 2013-10-14 AM 80:5-82:16]
- OSEK RTOS not certified; 80% CPU load (> 70% RMA limit)
  [Bookout 2013-10-14PM 42:6-25] [NASA App. A p. 119]
- Many large functions
  - 200 functions exceeded 75 lines of non-comment code [NASA App. A p. 23]
- Reviews informal and only on some modules
  [Bookout 2013-10-11 PM 29:24-30:5; 2013-10-14 49:17-21]
- No formal specifications [Bookout 2013-10-11 PM 29:24-30:5]
- No bug tracking system [Bookout 2013-10-14 PM 49:3-50:23]
- No configuration management [Bookout 2013-10-11 PM 30:7-10]

Some Legal Concepts

- The trials have been civil, not criminal
  - "Beyond reasonable doubt" is a criminal standard
    – not applicable in the death & injury lawsuits
- US typical decision threshold for civil lawsuit liability is:
  - **"More likely than not"**
- For product defects, common ideas are:
  - Was reasonable care exercised in the design?
    - For example, were accepted practices followed?
  - Unreasonably dangerous/defective for intended use?
    - Would it have been economically feasible to cure the defects?
  - The defects were a plausible cause of the loss event
    - Not necessarily the *only* possible cause

(Laws vary by US State; these are just common ideas)

Discussions



- My blog with relevant topical postings
  - Postings starting February 17, 2014 are edited excerpts from my Toyota expert report
    http://betterembsw.blogspot.com/
  - Contains full citation information
- Technical Reporting
  - EE Times articles by Junko Yoshida
- Mike Barr talk on code details
  - http://www.barrgroup.com/files/killer_apps_barr_keynote_eelive_2014.pdf
- Video of pumping brakes with open throttle
  - Consumer reports goes 80 mph with full braking:
    http://www.youtube.com/watch?v=VZZNR9O3xZM

Source Documents

- NASA & NHTSA Reports (significant redactions)
  - http://www.nhtsa.gov/UA
- Some Bookout Trial materials (small redactions)
  - http://www.safetyresearch.net/2013/11/07/toyota-unintended-acceleration-and-the-big-bowl-of-spaghetti-code/
- Exponent public report: (from Toyota experts)
  - http://pressroom.toyota.com/article_download.cfm?article_id=3597
- Timeline: (from plaintiff lawyers)
  - http://www.toyota-lawsuit.com/toyota-recall-timeline/
- Criminal case statement of facts (US Dept. of Justice)
  - http://www.justice.gov/iso/opa/resources/972014319414965522 4.pdf

Acknowledgements

- The NASA Toyota UA review team

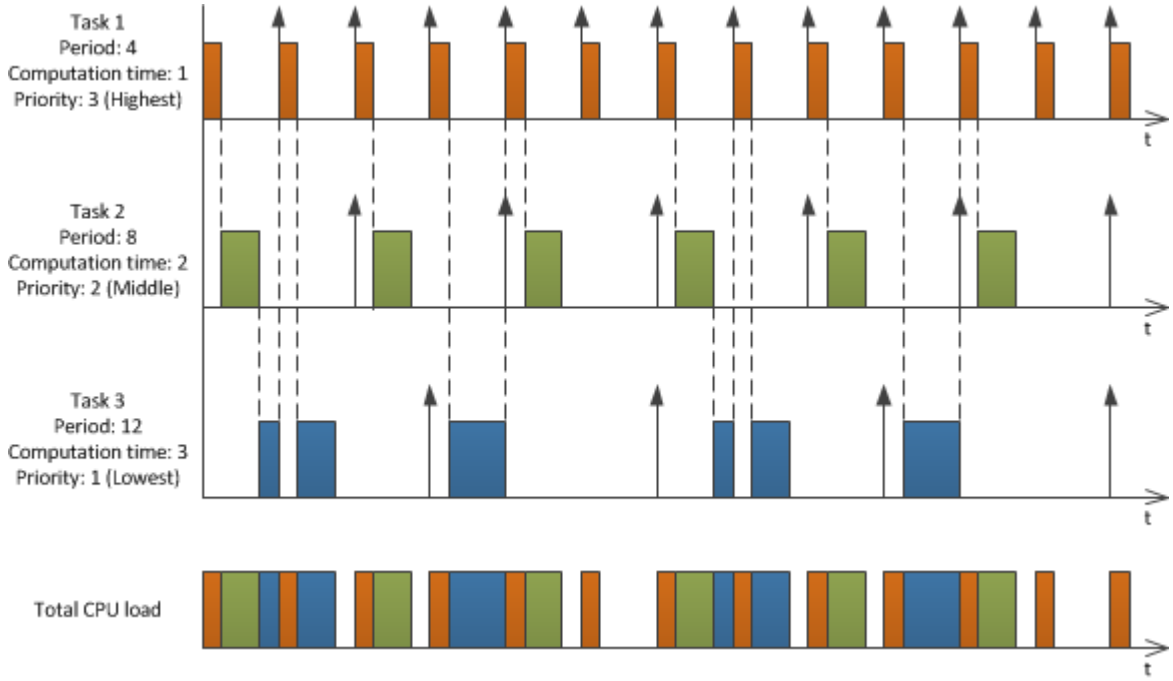
- The Plaintiffs' source code review team
  - Michael Barr
  - Nathan Tennies, Dan Smith, Nigel Jones
  - Carl Muckenhirn, Steve Loudon, Doug Denney

- Many people who have worked on this topic
  - Lawyers who decided to take on these cases
  - Other, numerous efforts

Additional Slides

Real Time Scheduling

- Hard real time systems have a deadline for each periodic task
  - With an RTOS, the highest priority active task runs while others wait
  - System fault occurs every time a task misses a deadline
  - Mathematical analysis is accepted practice for ensuring deadlines are met (…next slide…)



[Alexeev 2011, p. 5]

Schedulability

Meeting hard deadlines is one of the most fundamental requirements of a real-time operating system and is especially important in safety-critical systems. Depending on the system and the thread, missing a deadline can be a critical fault.

Rate monotonic analysis (RMA) is frequently used by system designers to analyze and predict the timing behavior of systems.

[Kleidermacher 2001 pg. 30]

Ensuring Deadlines Are Met

- Mathematical analysis is required to assure deadlines are met
- Rate Monotonic Analysis/Scheduling (RMA/RMS)
  - Ensures no missed deadlines under a set of assumptions
  - Must know worst case execution time of each periodic task
  - Some assumptions (e.g., no inter-task blocking)
- To accomplish RMS:
  - Prioritize tasks based on period (shortest period = highest priority)
  - Leave enough slack to account for worst case task arrivals
  - **Maximum 69.3% CPU usage maximum for an infinite number of tasks**

$$\mu = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(\sqrt[n]{2} - 1\right); \qquad \lim_{n \to \infty} \left(\mu\right) \leq \ln 2 \approx 69.3\%$$

- E.g., for 4 tasks: 4 * (4throot(2) – 1) = 4 * ($2^{0.25}$-1) = 75.68% loaded [Liu p. 53]

(There is special case that has a better bound, but that does not apply here)

Does ETCS Meet Deadlines?

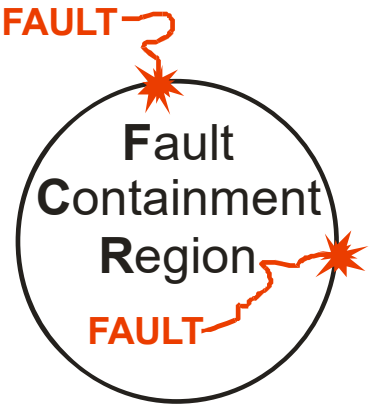
- Toyota didn't follow RMA and…
  - Has task over-run handling code [Bookout 2013-10-14PM 82:9-17]
- Operating System not certified OSEK
  [Bookout 2013-10-14PM 42:6-25]
  - Multiple priority levels; each with round-robin tasking
  - Toyota analysis based on 10 second engine run
    [NASA App. A p. 120]
- Timing analysis too difficult for NASA
  - Worst Case Execution Time difficult due to busy-wait loops, indirect recursion, etc. [NASA App. A pp. 120-133]
  - But, no deadline misses seen [NASA App. A. pp. 120-125]
- Main CPU less than 20% idle time at 5000 RPM
  - In other words, more than 80% loaded [NASA App. A p. 119]
  - Would **miss deadlines even if RMA had been used**

Partial Task List
[NASA App. A p. 37]

Task	Priority
T1_1msec	20
T5	18
T6	16
T7	16
T9	16
T11	16
T13	16
T15	14
T17	12
T19_4msec	10
T20	7
T23	4
T22_8msec	4
T24_idle	1

Fault Containment Regions

- HW/SW faults can have far-reaching effects
  - One hardware bit flip can kill an entire task
  - A wild pointer can corrupt a seemingly unrelated function
- A Fault Containment Region (FCR) provides a fault "firewall"
  - Faults inside stay inside
  - External faults stay outside



- Faults can have an arbitrarily bad effect within FCR
  - A single FCR can't self-police all of its own failure modes
  - Consistency checks – assume at least some data is accurate
  - Within-FCR failsafe might be corrupted by the fault it looks for

Redundancy Required For Critical Systems

- Need multiple, **independent** FCRs to detect faults
  - Each FCR can police other FCRs, but not itself in all cases
  - Shared resources are a dangerous single point of failure
  - Self-test isn't enough for safety
- Two common safety patterns:
  - Multi-channel system
    - Multiple CPUs cross-check or vote
  - Monitor/Actuator
    - Main CPU computes; secondary CPU checks
- Complete protection requires redundancy
  - Independent observations of input values
  - You can't trust another FCR – what if it gives you bad data?