Lecture 27: Safety

Seyed-Hosein Attarzadeh-Niaki

Based on Slides by Philip Koopman

Outline

- Critical Systems
- Embedded Software Safety Overview
- Safety Plan & Safety Standards
- Safety Requirements
- Single Points of Failure
- Isolation Mechanisms
- Safety Architectural Patterns

CRITICAL SYSTEMS

Critical Systems

- Critical systems require low failure rates
- **SIL** = Safety Integrity Level
  - Higher level of integrity needed for higher risk
- Safety critical: Loss of life, injury, environmental damage
  - Special care must be taken to avoid deaths
- Mission critical: Brand tarnish, financial loss, company failure
  - Consider a safety critical approach

What Is The Worst Case Failure?

- Worst case might not be obvious
  - Aircraft: software can cause a crash
  - Thermostats/HVAC: software can freezing plumbing
    - Can - rarely! - also kill small children due to overheating
- Key thought experiment
  - What's the worst that can happen if ...
    - ... your system intentionally tried to cause harm?
  - ... this identifies system hazards to mitigate
- Failure consequence varies, typically
  - Multiple fatalities (e.g., plane crash)
  - Single fatality (e.g., single-vehicle car crash)
  - Severe injuries
  - Minor injuries
  - Can consider analogies for mission-critical goals

Malfunctioning heater leads to Fort Worth toddler's death

MALFUNCTIONING HEATER LED TO BABY'S DEATH

Embedded Real-Time Systems

5

Safety Integrity Level (SIL)

- SIL represents
  - The risk presented by a system-level hazard
  - The engineering rigor applied to mitigate the risk
  - The permissible residual probability after mitigation

https://en.wikipedia.org/wiki/Bhopal_disaster

1984: Bophal Chemical Plant
Thousands of deaths
(not software related;
pre-dates IEC 61508)
https://goo.gl/GGHWRn

Example:
DO-178 (aviation flight hours)

- DAL A (Catastrophic): 10^9hrs/failure = 114077 years
- DAL B (Hazardous): 10^7hrs/failure= 1141 years
- DAL C (Major): 10^5hrs/failure= 11 years
- DAL D (Minor): 10^3hrs/failure= 42 days
- (DAL: Design Assurance Level)

Example:
IEC 61508 (industrial controls)

- SIL 4: 10^8hrs/dangerous failure = 11408 years
- SIL 3: 10^7hrs/dangerous failure= 1141 years
- SIL 2: 10^6hrs/dangerous failure= 114 years
- SIL 1: 10^5hrs/dangerous failure= 11 years

Embedded Real-Time Systems

6

Higher SIL Invokes More Engineering Rigor

Example: IEC 61508
- HR = Highly Recommended
- R = Recommended
- NR = Not Recommended (don't do this)

SIL 1: lowest integrity level (low risk)

SIL 4: highest integrity level (unacceptable risk)

	Technique/Measure*	Ref	SIL1	SIL2	SIL3	SIL4
1	Fault detection and diagnosis	C.3.1	---	R	HR	HR
2	Error detecting and correcting codes	C.3.2	R	R	R	HR
3a	Failure assertion programming	C.3.3	R	R	R	HR
3b	Safety bag techniques	C.3.4	---	R	R	R
3c	Diverse programming	C.3.5	R	R	R	HR
3d	Recovery block	C.3.6	R	R	R	R
3e	Backward recovery	C.3.7	R	R	R	R
3f	Forward recovery	C.3.8	R	R	R	R
3g	Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h	Memorising executed cases	C.3.10	---	R	R	HR
4	Graceful degradation	C.3.11	R	R	HR	HR
5	Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6	Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a	Structured methods including for example, JSD, MASCOT, SADT and Yourdon.	C.2.1	HR	HR	HR	HR
7b	Semi-formal methods	Table B.7	R	R	HR	HR
7c	Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8	Computer-aided specification tools [IEC 61508]	B.2.4	R	R	HR	HR

Embedded Real-Time Systems

7

Fleet Exposure & Probability

- Bigger fleets have increased exposure
  - 250 Million US vehicles @ 1 hour/day = $2.5 * 10^8$ hrs/day exposure
  - If "unlikely" failures happen every million hours... that's: $2.5 * 10^8$ hrs/10^6 hrs per event -> 250 events every day
  - This is why 10^8 to 10^{10} hrs is a typical goal
- Hardware components fail at ~10^5-10^6 hrs
  - Need two independently failing components to get to 10^9 hours!
    - This motivates redundancy for life-critical applications (SIL 3 & SIL 4)
- For mission-critical systems, consider:
  - Fleet exposure = # units * operational hours/unit
  - Number of acceptable failures
  - Compute failure rate = failures / hours; pick an appropriate SIL
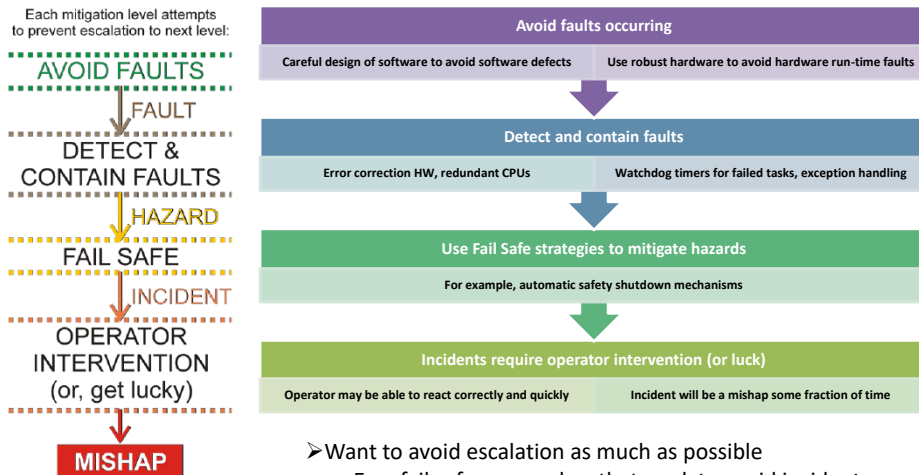
Embedded Real-Time Systems

8

Best Practices For Critical Systems

- **Characterize worst case failure scenarios**
  - Assign SIL based on relevant safety standard
  - Use engineering rigor for software SIL
  - Use redundancy for ultra-low failure rates
  - Consider fleet exposure, not just single unit

- **Pitfalls**
  - Software redundancy is difficult, and diversity is usually impracticable
  - Designer's intuition about "realistic" faults usually optimistic
    - At 10^{-9}/hr, random chance is a close approximation of a malicious adversary
  - Going through the motions not enough for SIL-based process

Embedded Real-Time Systems   9

EMBEDDED SOFTWARE SAFETY OVERVIEW

Embedded Real-Time Systems   10

Defense-In-Depth For Safety

Each mitigation level attempts to prevent escalation to next level:

AVOID FAULTS

↓ FAULT

DETECT & CONTAIN FAULTS

↓ HAZARD

FAIL SAFE

↓ INCIDENT

OPERATOR INTERVENTION (or, get lucky)

↓

MISHAP

Avoid faults occurring	
Careful design of software to avoid software defects	Use robust hardware to avoid hardware run-time faults

Detect and contain faults	
Error correction HW, redundant CPUs	Watchdog timers for failed tasks, exception handling

Use Fail Safe strategies to mitigate hazards	
For example, automatic safety shutdown mechanisms	

Incidents require operator intervention (or luck)	
Operator may be able to react correctly and quickly	Incident will be a mishap some fraction of time

➤ Want to avoid escalation as much as possible
E.g., fail safe approaches that work to avoid incidents

Embedded Real-Time Systems   11

Basic Safety Principles

Safety must be seen to be present
- System presumed **unsafe** unless convincing safety argument made
- Outsider must be able to determine safety purely from documents

The greater the risk, the greater the need for information
- Riskier systems require more engineering rigor

Safety must be built in, not added on
- If code is created without a safety process, throw it away; start over

Systematic, random, and malicious faults all matter
- Consider design errors and transient faults (e.g., soft errors)
- If it's not secure, it's not safe

Safety must be argued in writing and demonstrated
- Failure-free testing isn't enough

Safety is a lifecycle concern
- "Mission critical failures" can be considered "safety" as well

Embedded Real-Time Systems   12

Safety Culture:
Everyone Is Sure It's Safe!

- Space Shuttle Challenger Mishap
  - January 1986 launch explosion; 7 fatalities
  - Dual O-rings keep hot gases inside solid booster
    - History of sometimes failing if too cold
    - At launch, joint temperature was below freezing
  - Booster team told: "prove launch is unsafe"
    - Should have been: "no launch *unless* proven safe"
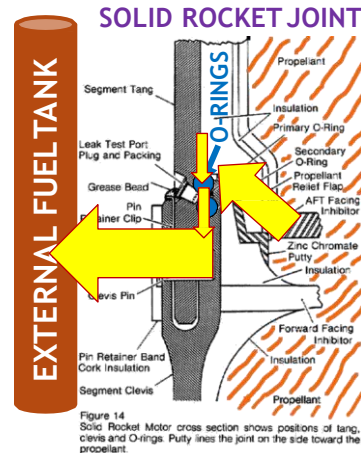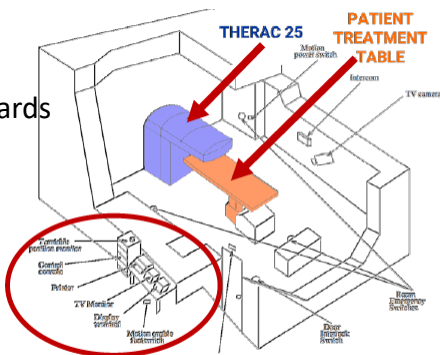   - Getting lucky is not the same thing as being safe

SOLID ROCKET JOINT

Embedded Real-Time Systems

13

Overview of Embedded System Safety

- Safety Topics
  - Safety Plan & Safety Standards
  - Safety Requirements
  - Critical System Design
  - Dependability
  - Single Points of Failure
  - Redundancy Management
  - Isolation Mechanisms
  - Safety Architectural Patterns
- Pitfall
  - Safety isn't just about whether you think it's safe ......
    it's about whether you can **prove it is appropriately safe**

(1985 – 1987) THERAC 25
Software-Controlled Radiation Therapy Mishaps

Embedded Real-Time Systems

14

7

SAFETY PLAN

Safety Plan: The Big Picture for Safety

Safety Plan
- Safety Standard: pick a suitable standard
- Hazards & Risks: hazard log, criticality analysis
- Goals: safety strategy, safety requirements
- Mitigation & Analysis: HAZOP, FMEA, FTA, ETA, reliability, …
- Safety Case: safety argument

Safety Standards

- Usually "functional safety" (safety functions)
  - IEC 61508 is a generic starting point
  - Many domains have specific standards
    - ISO 26262, EN-50126/8/9, MIL-STD-882, IEC 60730, DO-178, …
- Key elements of a safety standard
  - Method for determining risk
    - Usually Safety Integrity Level (SIL)
  - SIL determines engineering rigor
    - Analysis techniques
    - Mitigation techniques
  - Life-cycle approach to safety

[IEC 61508]
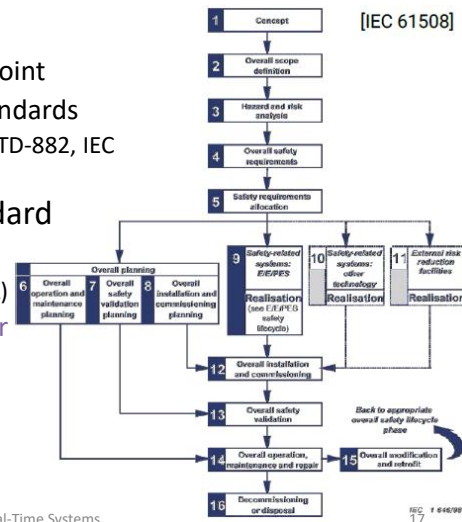
Embedded Real-Time Systems
17

Safety Goals & Safety Requirements

- **Safety Goal**: top level definition of "safe"
  - *Example*: vehicle speed control
    - Hazard: unintended vehicle acceleration
    - Goal: engine power proportional to accel. pedal position
  - Safety strategy: how you plan to achieve goal
    - Example: correct computation AND engine shutdown if unintended acceleration
- **Safety Requirements**
  - Goals at system level; requirements provide supporting detail
  - Supporting requirements generally allocated to subsystems
    - Might include functionality and fail-safe mitigation requirements
  - *Examples*
    - Engine torque shall match accelerator position torque curve
    - Pedal/torque mismatch shall result in engine shutdown

Embedded Real-Time Systems
18

FMEA: Failure Mode Effects Analysis

- Idea: Start with component failure; analyze results; identify hazards

Component	Potential Failure Mode	Failure Effects	Recommended Action	Status
Resistor R2	Open	Triggers Shutdown	Use Industrial spec. component	Done
	Short	Over-current/ potential Fire	Circuit Redesign	Open
Capacitor C7	Explodes	Potential Fire	Select different component	Open

- Significant limitations for generating hazards
  - "Complex component" failures are not well behaved
    - Software fails however it wants to fail
    - Integrated circuits are usually highly coupled internally
  - Poor at representing correlated and accumulated faults
    - E.g., exploding capacitor damaging several nearby components

HAZard and Operability Analysis (HAZOP)

- Hazard structured brainstorming
  - For each system requirement
    - Modify with a guide word
    - Does the result suggest a hazard?
  - Effective starting point, but not guaranteed to find all hazards
- Examples
  - When pressure exceeds 6000 psig, relief valve shall **NOT** actuate.
  - System shall come to a complete stop ~~within~~ **AFTER** 5 seconds when emergency stop is activated.
    - Alternately: System shall come to a complete stop ~~within 5 seconds~~ **LATE** when emergency stop is activated.

Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN / INSTEAD	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

Hazards & Risks

- Hazard: a potential source of injury or damage
  - A potential cause of a mishap or loss event (people, property, financial)
- Hazard log
  - Captures hazards for a system
  - HAZOP generates some hazards
  - Others are legacy & experience
- Risk evaluation
  - Risk = Probability * Consequence
    - Typically determined via a risk table
  - Risk must be reduced to acceptable levels
    - Risk determines required SIL (e.g. "Very High" -> SIL 4)
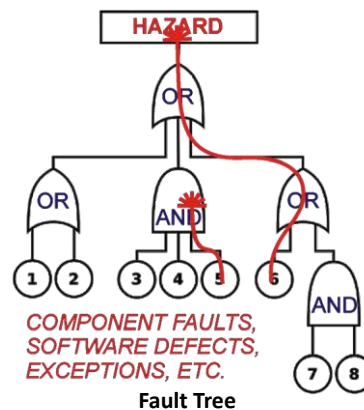

Embedded Real-Time Systems

21

Safety Analysis & Mitigation

- **Failure Mode Effects Analysis (FMEA)**
  - Work forward from fault to mishap
- **Fault Tree Analysis (FTA)**
  - Work backward from hazard to causes
  - Strategy: HAZOP identifies fault tree roots
- Avoid single points of failure
  - If component breaks, is system unsafe?
  - Computational elements fail in worst way
- Life-critical systems require redundancy
  - Also avoid correlated faults
  - High-SIL software techniques to avoid SW defect

COMPONENT FAULTS, SOFTWARE DEFECTS, EXCEPTIONS, ETC.

Fault Tree

Embedded Real-Time Systems

22

11

Best Practices For Safety Plans

- A written Safety Plan including
  - Hazards + risks
  - Safety goals + requirements
  - Safety analysis + Mitigation
  - Following a safety standard
  - Resulting in a written safety case
  - Independent audit of safety case

- Pitfalls
  - Software safety usually stems from rigorous SIL engineering
  - FMEA can miss correlated & multipoint faults – must use FTA
  - Need to include safety caused by security attacks

Embedded Real-Time Systems   23

SAFETY REQUIREMENTS

Embedded Real-Time Systems   24

Specifying safety

- Safety goals: **"working" is not the same as "safe"**
  - How hazards are avoided at system level
  - Can involve correctness, backup systems, failsafes, …
  - Often what the system <u>does not do</u> is as important as what it does
- Safety requirements
  - More detailed safety-specific requirements allocated to subsystems

---

Identifying Safety-Related Requirements

**Requirement Annotation Approach**

- Overly-simplistic approach
  - Start with system requirements
  - Annotate critical system requirements
  - Then, annotate supporting requirements
  - Problem:
    Most requirements can become critical

☒ R01. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
R02. Nam suscipit odio aliquam massa finibus, id imperdiet.
☒ R03. Quisque vehicula quam ut dui venenatis varius.
☒ R04. Nulla posuere diam ac augue bibendum, vitae laoreet.
R05. Pellentesque aliquam sem sit amet justo porttitor.
☒ R06. Vestibulum scelerisque lacus ac neque volutpat, dictum.
☒ R07. Ut venenatis ante in ligula efficitur, congue posuere.
☒ R08. Nam a nulla ultrices, tempor quam et, fringilla nisl.
☒ R09. Vestibulum a arcu interdum, placerat eros non, ultrices.
☒ R10. Ut commodo odio eu elit porttitor facilisis.
☒ R11. Etiam et sem eu eros congue sollicitudin.
☒ R12. Proin tincidunt arcu quis dui tristique volutpat.
R13. Fusce quis magna aliquet, venenatis sem ac, rhoncus.
☒ R14. Cras vel nulla eget orci semper varius scelerisque tellus.
☒ R15. Cras mollis lorem vitae libero sollicitudin lobortis.
R16. Vestibulum luctus nisi ac nibh varius congue.
☒ R17. Maecenas consequat augue eu venenatis euismod.
☒ R18. Quisque viverra felis in est ornare consectetur.
☒ R19. Cras pellentesque turpis sit amet justo scelerisque.

- Too many system components promoted to highest criticality level
  - Allocating even one critical requirement to a component makes whole thing critical

Safety Envelope Requirements Approach

- Safety Envelope
  - Specify unsafe regions for safety
  - Specify safe regions for functionality
    - Deal with *complex boundary* via
      - Under-approximate safe region (reduces permissiveness)
      - Over-approximate unsafe region
    - Trigger system safety response upon transition to unsafe region
- Partition the requirements
  - Operation: functional requirements
  - Failsafe: safety requirements (safety functions)
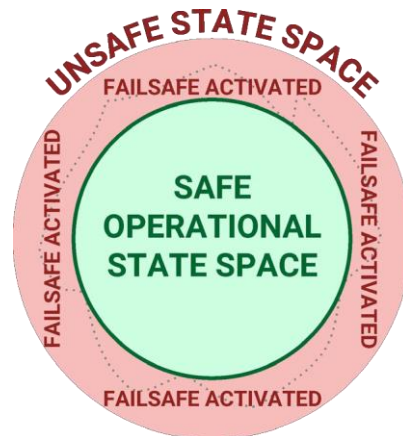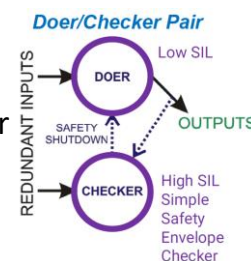

Embedded Real-Time Systems   27

Architecting A Safety Envelope System

- "Doer" subsystem
  - Implements normal functionality
  - Allocate functional requirements to Doer
- "Checker" subsystem
  - Implements failsafes (safety functions)
  - Allocate safety requirements to Checker
- *Checker is entirely responsible for safety*
  - Doer can be at low SIL (failure is lack of availability)
  - Checker must be at high SIL (failure is unsafe)
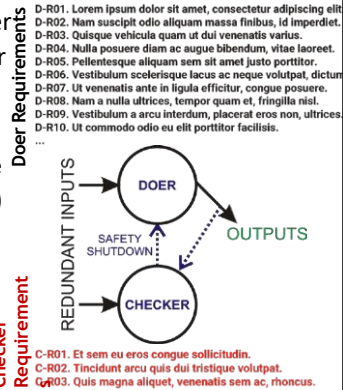   - Often, Checker can be much simpler than Doer

Embedded Real-Time Systems   28

Safety Requirements Best Practices

- Doer/Checker pattern
  - Functional requirements allocated to low-SIL Doer
  - Safety requirements allocated to high-SIL Checker
- Good safety requirements
  - Trace to system-level safety goals
    - Orthogonal to normal functional operation if possible
  - Make safety simple to validate (test, peer review)
    - Safety testing mostly exercises the Checker box
- Pitfalls
  - Tradeoff between simplicity and permissiveness
    - Doer optimality costs Checker validation effort
  - Fail-operational functions may require multiple Doer/Checker pairs

Doer Requirements

D-R01. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
D-R02. Nam suscipit odio aliquam massa finibus, id imperdiet.
D-R03. Quisque vehicula quam ut dui venenatis varius.
D-R04. Nulla posuere diam ac augue bibendum, vitae laoreet.
D-R05. Pellentesque aliquam sem sit amet justo porttitor.
D-R06. Vestibulum scelerisque lacus ac neque volutpat, dictum
D-R07. Ut venenatis ante in ligula efficitur, congue posuere.
D-R08. Nam a nulla ultrices, tempor quam et, fringilla nisl.
D-R09. Vestibulum a arcu interdum, placerat eros non, ultrices.
D-R10. Ut commodo odio eu elit porttitor facilisis.
...

REDUNDANT INPUTS → DOER → OUTPUTS

SAFETY SHUTDOWN

CHECKER

Checker Requirement

C-R01. Et sem eu eros congue sollicitudin.
C-R02. Tincidunt arcu quis dui tristique volutpat.
C-R03. Quis magna aliquet, venenatis sem ac, rhoncus.

SINGLE POINTS OF FAILURE
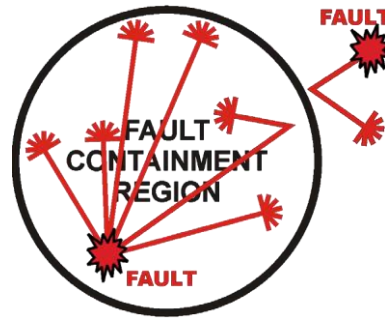
Avoid Single Points of Failure

- Fault Containment Region (FCR)
  - Faults from outside FCR are kept out
    - Faults inside FCR are kept in
  - But, within FCR a single fault has arbitrarily bad effects
    - It's like a blast inside the FCR
   - Applies to both SW faults and HW faults (e.g., single event upsets)

Embedded Real-Time Systems   31

Eliminating Single Points of Failure

- ➤ Multiple FCRs required for life-critical and highly mission-critical systems
  - This isolates faults in redundant components – no single point of failure
  - Avoid an Achilles' Heel in your system
    - All software on CPU can be a "single point"
- Multi-channel (e.g., 2 of 2)
  - Compare identical component outputs
- Doer/Checker (monitor/actuator pair)
  - "Checker" makes sure "Doer" is safe
- Safety gate
  - Only permits safe outputs to issue
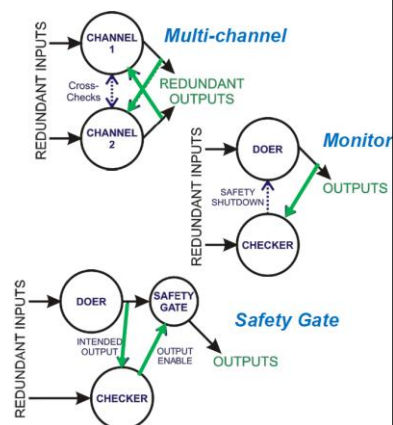
Embedded Real-Time Systems   32

Correlated & Accumulated Faults

- <u>Correlated</u> faults if multiple FCRs are likely to fail together
  - Common design faults (including software)
  - Common manufacturing faults
  - Shared infrastructure (e.g., power, clock)
  - Physical coupling
    - Shared wiring harness, connectors
    - Shared location (e.g., hot spot)
- Accumulated faults
  - Fault not detected
  - Fault not repaired before next mission

Embedded Real-Time Systems

33

Best Practices
To Avoid Single Points of Failure

- Safety is improved by using multiple FCRs
  - Hardware redundancy / HW isolation
    - Typically each FCR should be an *independent chip*
  - Software must be practically "perfect"
  - Common patterns: multi-channel, checker, safety gate

- Pitfalls are numerous and sometimes subtle
  - Two copies of same SW fail the same way
  - Ensure multi-channel doesn't fail as "always trust one channel"
  - Ensure the checker doesn't fail as "always checks OK"
  - Look for hidden correlation (HW design defects, shared libraries, shared requirement defects, physical connection, shared clock, shared power, …)

Embedded Real-Time Systems

34

CRITICAL SYSTEM ISOLATION

Embedded Real-Time Systems 35

Critical System Isolation

Need isolation between different SILs

- Lower SIL assumed to compromise High SIL
  - Higher SIL -> "trusted" (critical tasks)
  - Lower SIL -> "untrusted" (non-critical tasks)
    - Corrupts high-SIL data values, timing, configuration
- Hardware isolation is best option
  - Different SILs separated on different chips
  - Different networks for safety vs. non-safety data
    - Network data exchange is safety critical
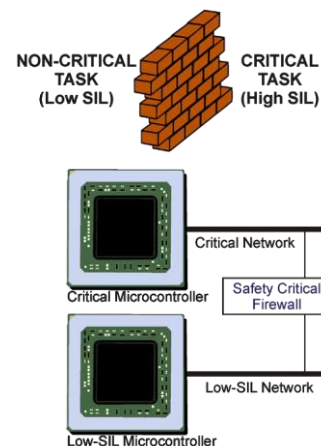
Embedded Real-Time Systems 36

Mixed-SIL Interference Examples

Memory value interference
- Non-critical task modifies critical variables
- Non-critical ISR causes critical task stack overflow
- Non-critical task memory leak; heap exhaustion

CPU time interference
- Non-critical task runs at high priority; starves critical tasks
- Non-critical task disables interrupts; delaying critical tasks

Watchdog timer
- Non-critical task kicks watchdog regularly
- Non-critical task disables watchdog

System configuration
- Non-critical task changes digital output to input

Network
- Non-critical node sends unsafe critical message

CRITICAL TASK (High SIL)

NON-CRITICAL TASK (Low SIL)

MEMORY
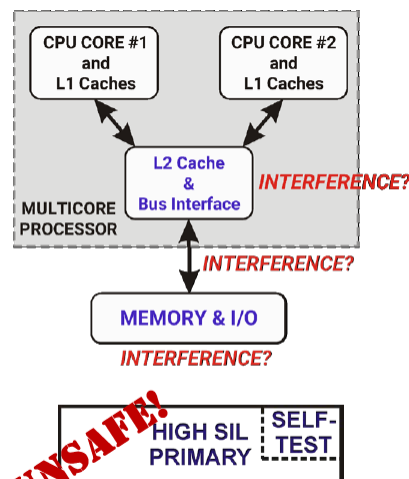CPU
Watchdog Timer
Config. Data
Network
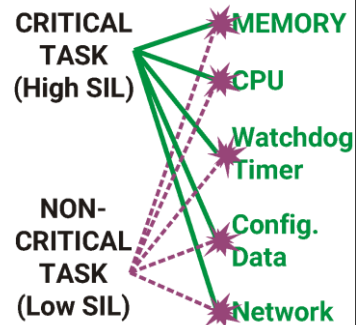
Embedded Real-Time Systems

37

Mitigating Cross-SIL Interference

- **Develop all software at highest SIL**
  - Avoids isolation, but increases expense
- **Hardware solution – separate CPU chips**
  - Multi-core provides only partial isolation
- **High-SIL RTOS approaches**
  - Hardware memory protection (MMU)
  - Hardware CPU time isolation (e.g., multi-core)
  - Virtualization of I/O and configuration
- Other techniques can help for Low-SIL
  - Variable mirroring (two one's complement copies)
  - Critical tasks run at high priorities or in ISRs
  - Non-modifiable watchdog timer configuration
- Self-test is insufficient for High-SIL integrity
  - Fault in high SIL hardware can subvert self-test

CPU CORE #1 and L1 Caches

CPU CORE #2 and L1 Caches

L2 Cache & Bus Interface

INTERFERENCE?

MULTICORE PROCESSOR

INTERFERENCE?

MEMORY & I/O

INTERFERENCE?

UNSAFE!

HIGH SIL PRIMARY

SELF-TEST

Single CPU at SIL 3 or SIL 4

Embedded Real-Time Systems

38

19

Isolation and Security

- Lower-SIL task is ~ a malicious attacker
  - How can it disrupt higher-SIL software?
  - Consider:
    memory corruption, timing, configuration, network
- Implications for safety
  - A weaker fault model means making assumptions
  - Lower-SIL update means revisiting assumptions
- Implications for security
  - Higher-SIL functions more resistant to attack if isolated
  - Bad pattern: everything on one CPU with desktop OS
  - Better pattern: isolated CPUs with high-SIL critical RTOS

Embedded Real-Time Systems   39

Best Practices For Critical System Isolation

- Use as much hardware isolation as you can
  - Consider
    - Data value isolation
    - CPU time isolation
    - Configuration corruption
    - Shared resource isolation

  - Applies to any different SILs
    - Crucial for non-SIL <-> SIL 3/4
- Pitfalls
  - Multi-core CPU isn't enough on its own (other shared resources!)
  - IEC 60730: Arguing that low-SIL software won't interfere...... requires re-arguing after every low-SIL change

Embedded Real-Time Systems   40