CPS Applications

Heechul Yun

Note: Some slides are adopted from Prof. Pellizzoni
Outline

• Avionics
• Automotive Systems
Avionics

• Electronic systems on an aircraft
  – Avionics = Aviation + electronics
  – Multiple subsystems: communications, navigation, display, flight control, management, etc.

• Modern avionics
  – Increasingly computerized

• Safety culture
  – Safety critical; conservative; regulated (FAA, EASA)
Fly-by-wire

- Modern aircrafts rely on computers to fly
- Pilots do not directly move flight control surfaces (ailerons, elevator, rudder)
- Instead, Electronic Flight Control System does.

![Diagram of Fly-by-wire system]

**FCS**

Yoke → FCS → Control surfaces
Autopilot

• Specify desired track: heading, course, waypoints, altitude, airspeed, etc.
Increasing Complexity

Total Onboard Computer Capacity (OFP)

k Source Lines of Code (kSLOC)

Example: F-22

• In 2007, 12 F-22s were going from Hawaii to Japan.

• After crossing the IDL, all 12 experienced multiple crashes.
  – No navigation
  – No fuel subsystems
  – Limited communications
  – Rebooting didn’t help

• F-22 has 1.7 million lines of code
Verification and Validation (V&V)

• Validation
  – “Are we building the right system?”
  – Check if the system meet the requirements

• Verification
  – “Are we building the system right?”
  – Check if the system meets the specification

• It is possible the a system is verified correct but not useful (not valid)
V & V Cost

Phase in which error was detected and corrected

~80-90% of faults introduced here

~96% of faults found here

Image credit: Dr. Guillaume Brat NASA Ames Research Center
Certification

• Convincing the certification authority that the validation process is correct
• Largely process driven
  – Shows that you followed a good process
  – Document everything
  – Review everything (with independence)
• Evidence driven
  – Use formal methods and automated tools (model checkers, theorem provers, ...) in place of independent reviewers
DO-178 B/C

- Software Considerations in Airborne Systems and Equipment Certification
- A document used by certification authorities (FAA, EASA) to certify avionics software
- Basic idea
  - Access the safety implications of failure modes
  - Map failure modes to 5 safety levels (A to E)
    - A: catastrophic – failure may cause a crash
    - B: hazardous – failure has a negative impact on safety/perf.
  - Must satisfy a set “objectives” (with independence)
    - E.g., algorithms are accurate, software partitioning is confirmed, source code complies low-level requirements, ...
DO-178C and Formal Methods

Demonstrate the use of formal methods to satisfy DO-178C verification objectives.

- Theorem Proving is demonstrated on the Pilot Flying Protocol (Table A.3)
- Model Checking is demonstrated on the FGS Mode Logic design (Table A.4)
- Abstract Interpretation is demonstrated on the Heading Control Law source code (Table A.5)

Image credit: Dr. Lucas Wagner, Honeywell
Avionics Architecture

• Federated architecture
  – A function = a computer (box)
  – More functions ➔ more boxes (computers)
    • flight management, fuel management, flight
      envelope protection, collision avoidance...
  – Each box is uniquely designed for each
    specific aircraft
    • custom hardware/software
    • 100s km cabling

Image credit: ARTIST2 - Integrated Modular Avionics A380
Avionics in Airbus

Image credit: ARTIST2 - Integrated Modular Avionics A380
Integrated Modula Avionic (IMA)

- Use a set of standard computers
- Use a standard OS (ARINC 653)
- Use standard data communication network (AFDX)
- Multiple applications can be executed on the same computer
- Each computer can be configured to partition its resources to serve multiple functions

Image credit: ARTIST2 - Integrated Modular Avionics A380
ARINC 653

• Avionics Application Standard Software Interface (think POSIX for avionics)
• The software base of IMA
• Main idea: integrate software partitions with different criticality levels on the same/communicating computational node.
• A set of OS/Hypervisor provisions for safe partitioning and associated API.
ARINC 653

- Time partitioning

---

**Figure 4**

Timing partitions are scheduled using a time slice mechanism according to ARINC 653. This mechanism defines a period and duration for each partition known as the minor frame time. As shown here, each partition can be scheduled one or more times with the total sum of all minor frames being the major frame duration.

Image credit: http://www.cotsjournalonline.com/articles/view/100736
What about multicore?

• ARINC653 and time partitioning was designed single-core systems in mind.

• Problems of executing multiple partitions in parallel on multiple cores
  – Cache, memory, bus are shared.
  – Isolation is not guaranteed
  – A critical partition may be delay by low critical partitions on different cores
Denial-of-Service Attack

- **Delay execution time** of time sensitive code
  - E.g., real-time control software of a car
  - Observed >21X execution time increase on Odroid XU4 (*)
    - Even after cache partitioning is applied
  - Observed >10X increase on RPi 3 (**)
    - Of a realistic DNN-based real-time control program

Denial-of-Service Attack

Multicore and Certification

• CAST32
  – A position paper by FAA, EASA, and other certification agencies on multicore certification
  – Not a definite rule or guideline, but
  – Discuss “interference channels” of multicore
  – State objectives to meet for certification
MCP_Planning_1

- Identify the specific MCP processor
- Identify the number of active cores,
- Identify the MCP software architecture
- Identify dynamic features in software
- Identify whether the MCP is for IMA
- Identify whether the MCP support "Robust Resource/Time Partitioning"
- Identify the methods and tools used for software development/verification
MCP_Planning_2

• Describe how MCP shared resources will be used, allocated, verified to avoid contention
• Identify hardware dynamic features
• The applicant has identified the interference channels that could permit interference to affect the software applications hosted on the MCP cores, and has verified the applicant’s chosen means of mitigation of the interference.

  – Two cases: MCP w/ or w/o Robust Partitioning
Robust *Resource* Partitioning

- Software partitions cannot contaminate storage space for the code, I/O, data of other partitions (MMU, VM)
- Software partitions cannot consume more than allocated resources
- Failures of hardware unique to a software partition cannot cause adverse effect on the other software partitions.
Robust *Time* Partitioning

• No software partition consumes more than its allocated execution time on the core(s) on which it executes, *irrespective of other partitions on different cores.*
MCP_Resource_Usage_3

• Case 1: MCP Platforms With Robust Partitioning
  – “...may verify applications separately on the MCP and determine their WCETs separately. “

• Case 2: All Other MCP Platforms
  – “... should be tested on the target MCP with all software components executing in the intended final configuration, ...”
  – “... WCET should be determined by analysis and confirmed by test on the target MCP with all software components executing in the intended final configuration.”
The applicant has identified the available resources of the MCP and of its interconnect in the intended final configuration, has allocated the resources of the MCP to the software applications hosted on the MCP and has verified that the demands for the resources of the MCP and of the interconnect do not exceed the available resources when all the hosted software is executing on the target processor.

NOTE: The need to use Worst Case scenarios is implicit in this objective.
The applicant has verified that all the software components hosted by the MCP comply with the Applicable Software Guidance. In particular, the applicant has verified that all the hosted software components function correctly and have sufficient time to complete their execution when all the hosted software is executing in the intended final configuration.

– TL;DR: Need logical and temporal correctness
The applicant has verified that the **data and control coupling** between all the individual software components hosted on the same core or on different cores of the MCP has been exercised during software requirement-based testing, including exercising any interfaces between the applications via shared memory and any mechanisms to control the access to shared memory, and that the data and control coupling is correct.

– TL;DR: need system-level testing
MCP_Error_Handling_1

• The applicant has identified the effects of failures that may occur within the MCP and has planned, designed, implemented and verified means (which may include a ‘safety net’ external to the MCP) commensurate with the safety objectives, by which to detect and handle those failures in a fail-safe manner that contains the effects of any failures within the equipment in which the MCP is installed.
Safety-Net

• Assumption: MCP can fail
• Goal: “fail-safe” operation
  – can safely fly and land, but not at 100% performance
• How?
  – passive monitoring functions
  – active fault avoidance functions
  – control functions for recovery
Multicore and Certification

• Major research topic
• Research goal
  – A) Achieve time predictability and isolation
  – B) Maximize throughput (as long as A. is met)
  – For multicore based real-time embedded systems
  – So that such systems can be certifiable
Outline

• Avionics
• Automotive Systems
Recap: Multicore and Certification

• CAST32
  – A position paper by FAA, EASA, and other certification agencies on multicore certification
  – Not a definite rule or guideline, but
  – Discuss “interference channels” of multicore
  – State objectives to meet for certification
    • Robust Resource/Time Partitioning
Automotive Systems

• 100s of processors (ECU)

Image credit: Simon Fürst, BMW, EMCC2015 Munich, adopted from OSPERT2015 keynote
Automotive Systems

- ECUs, sensors, actuators
- Many subsystems
  - Anti-lock breaking systems, Electronic stability control, Adaptive cruise control, Adaptive light control, Lane departure warning, infotainment, ...
  
- For comports and safety

Image credit: Prof. Brandenburg
Increasing Complexity

• Today’s cars depend on software/computers
  – Safe, dependable software is hard

Image source: https://hbr.org/resources/images/article_assets/hbr/1006/F1006A_B_lg.gif
Safety Challenge

• Unintended acceleration
  – Caused several fatal accidents
    • e.g., Aug. 2009 accident in California
  – Toyota settled to pay $1.2B

• Potential causes
  – Memory corruption
  – Unsafe software design
Safety Challenge

EXAMPLE OF UNINTENDED ACCELERATION

- Representative of task death in real-world
- Dead task also monitors accelerator pedal, so loss of throttle control
  ✓ Confirmed in tests
- When this task’s death begins with brake press (any amount), driver must fully remove foot from brake to end UA
  ✓ Confirmed in tests

Source: Loudon Vehicle Testing

time (seconds)
Automotive System Development

• Each subsystem (function)’s software and hardware is built by a vendor (contractor)
• Car manufacturers (e.g., GM) integrate and validate
• Problems
  – Size, weight, and power issue
  – Lack of standards
  – Difficult to interoperate, share code, refine
Size, Weight, and Power (SWaP)

- Maximum performance with minimal resources
  - Cannot afford too many or too power hungry ECUs

Figure source: OSPERT 2015 Keynote by Leibinger
AUTOSAR

- **AUTOSAR** – **AUT**omotive **O**pen **S**ystems **AR**chitecture
- Same motivation: cope with complex with standardization
- Define standard interfaces for software independent of hardware ECU
AUTOSAR

- Improve interoperability

Image credit: AUTOSAR tutorial at autosar.org
AUTOSAR RTE

- POSIX for AUTOSAR.
  - Define services and APIs for applications

Image credit: AUTOSAR tutorial at autosar.org
Use case ‘Front-Light Management’ in AUTOSAR

SwitchEvent
- check_switch()
- switch_event(event)

LightRequest
- switch_event(event)
- request_light(type, mode)

Front-Light Manager
- request_light(type, mode)
- get_keyposition()
- set_light(type, mode)

Xenonlight
- set_light(type, mode)
- set_current(...)
Controller Area Network (CAN)

- Multi-master serial bus for connecting ECUs
  - Up to 1Mbps
- De-factor comm. standard in automotive
- Safety critical controls
  - E.g.) steering, breaking, throttle, ...

Image credit: https://en.wikipedia.org/wiki/CAN_bus
## CAN Message

<table>
<thead>
<tr>
<th>SOF</th>
<th>11-bit Identifier</th>
<th>RTR</th>
<th>IDE</th>
<th>r0</th>
<th>DLC</th>
<th>0…8 Bytes Data</th>
<th>CRC</th>
<th>ACK</th>
<th>EOF</th>
<th>IFS</th>
</tr>
</thead>
</table>

- **SOF**: Start of Frame
- **RTR**: Remote Transmission Request
- **IDE**: Identifier Extension
- **r0**: Identifier Bit 0
- **DLC**: Data Length Code
- **CRC**: Cyclic Redundancy Check
- **ACK**: Acknowledgment
- **EOF**: End of Frame
- **IFS**: Inter-Frame Space
CAN Networks and Vulnerability

• Set of ECU connected by CAN buses.
• CAN buses are designed for real-time (fixed priority messages), but not security...
• Broadcast with no authentication field: any ECU connected to a CAN bus broadcasts to all other ECU on the same bus. No way to determine the sender.
• Weak Access Control: there is a challenge-response sequence but the codes must be known by all service centers to perform diagnostic = they are out in the open.
• ECU Firmware Update: the firmware of any ECU can be updated over the CAN bus.
• Bridge nodes: there are different CAN buses (critical / non-critical), but they are bridged by dedicated ECU nodes.
• Result: if you can hack any ECU, you can re-flash any other ECU...
External I/O Channels

Comprehensive Experimental Analyses of Automotive Attack Surfaces, USENIX Security, 2011