

# Lessons on Writing Papers

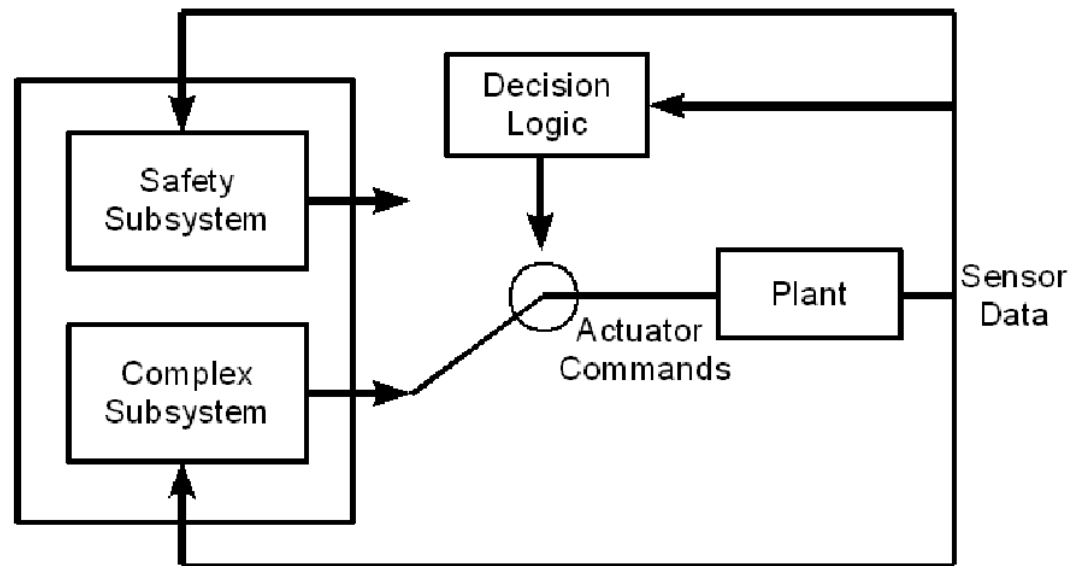
Stanley Bak  
1/23/2009

# History

- Paper submitted in 2008 for RTSS
  - Rejected
- Paper submitted in 2009 to RTAS
  - Accepted
- The difference?

# Main Contribution

- Simplex Architecture on an FPGA
- We move the Safety Subsystem and Decision Logic to hardware for additional safety



# Changes

- Motivation for changes
  - Reviewer suggestions
  - “*The Craft of Scientific Writing*” - Alley
- Combined with recommendations from professors, of course

# Reviewer Suggestions

# Reviewer Comments

- Several categories of comments
  - Misunderstandings (reword and clarify)
  - Small Changes (modify sentence or paragraph)
  - Large Changes (add/remove sections)

# Misunderstandings

- “there is a serious problem when the decision logic does not receive signals from the complex processor... The paper just states that the last value is stored and used. I ask myself, how long is such a stored valid?”
- Unnecessary implementation details may confuse readers

We store the most-recent output for each of the controllers and use that value when a decision must be made

# Small Changes Comments

- “Why FPGA and not software?”
  - Agreed: You can use software, if you accept its associated complexity
- “Your example is bad because...”
  - Remove example
- “You said a formally verified OS is infeasible but what about the work by...”
  - Change to “Although great progress has been made towards producing a verifiable RTOS[3], most current RTOSes have been neither formally verified nor exhaustively tested”



# Small Changes Comments (2)

- “This isn't new or interesting”
  - Emphasize what is new and non-obvious, and say why it isn't obvious

Two Simplex safety-critical components, the safety controller and decision module, are moved to a dedicated processing unit, **not for the typical HW/SW co-design reasons of power and performance**, but instead to provide isolation from software-related complexity.

# Small Changes Comments (3)

- “There's no mention of limitations”
- Not mentioning limitations implies the solution is **always** easier, cheaper and better
  - which makes reviewers nervous

Simplex (both versions) should not be regarded as a one-size-fits-all robustness approach. To guarantee safety, we must [do X, Y, and Z, which are not always possible].

# Small Changes Comments (4)

- Other mentions of limitations

An inverted pendulum, however, does not completely lend itself to our end-to-end design process.

One drawback of the System-Level version of Simplex is that additional resources are required to run the decision module and complex controller, such as the FPGA.

# Large Changes

- Originally we had a section which formally model-checked the Simplex Architecture for safety given certain properties of the controllers
- The reviewers were not impressed
  - “Except for informal arguments, there is no proof that these 4 are necessary and sufficient to guarantee the correctness of the architecture.”

# Large Changes (2)

- Instead, the reviewers hinted towards a more interesting direction
  - “The problem is that there don't seem to be any science in the sense of a systematic way of figuring out the correct implementation of the decision logic or the safety subsystem.”
- Solution: add section about modeling checking the decision logic and safety subsystem

# Large Changes (3)

- We also added a section about AADL generation, which generated reviewer comments
  - “Authors should give a brief introduction or related references on the AADL model.”
  - “The AADL material should be improved and better motivated, as it is it seems to add little or nothing to the paper.”

# “The Craft of Scientific Writing”

## Michael Alley

# General Changes from TCOSW

- Avoid pretentious words:
  - Utilize / Utilization -> Use
  - Facilitate -> Case / Bring About
  - Component -> Part
- Change title:
  - Demonstrate clear match to one of the conference topics
  - “*novel hardware or system architectures for real-time or embedded systems*”



# Change Title

- From TCOSW, titles should do two things
  - Identify **field of study**
  - **Separate** from all other publications in that field
- Previous Title
  - *“The Hardware Simplex Architecture for **Real-Time, Safety-Critical Systems**”*
- New title
  - *“The System-Level Simplex Architecture for **Improved Real-Time Embedded System Safety**”*

# List formatting

- Lists should be of 2, 3, or 4 items
- Lists should be concise
- Lists should be precise
- Parallel structure

# Main Contributions

- original:

- 1) The Hardware Simplex Architecture can handle a superset of the failure modes of the SW Simplex Architecture. Additional faults that can be safely

---

handled include RTOS timing faults, middleware bugs, and microprocessor errors.

- 2) The HW Simplex Architecture generates negligible processor overhead when compared to running a single controller with no fail-operational mechanism. This contrasts with the SW Simplex Architecture which requires the processor to run the decision module and safety controller.
  - 3) The design model is formally specified in Maude [11] and two key properties are model checked:
    - a) The system is safe regardless of the output of the complex controller, faults in the RTOS, or failures in the microprocessor
    - b) If the complex controller, RTOS, and microprocessor perform correctly, the system always uses the high-performance output of the complex controller
- 

- fixed:

- The design of the System-Level Simplex Architecture which can handle a superset of the failure modes of previous Simplex versions
- An end-to-end design process that both verifies a valid System-Level Simplex AADL architecture model and can generate hardware VHDL code from a finite-state machine description
- Empirical verification of both the practicality of end-to-end System-Level Simplex design, and the robustness guarantees through fault-injection testing in two case studies

# Safety Rules (original)

Type	✓	✓ <sup>a</sup>
No Output	✓	✓ <sup>a</sup>
Timing	✓	✓ <sup>a</sup>
Middleware	✓	✓ <sup>a</sup>
Operating System	✓	✓ <sup>a</sup>
Microprocessor	✓ <sup>c</sup>	✓ <sup>d</sup>
Power	✓ <sup>c</sup>	✓ <sup>d</sup>
Hardware	✓ <sup>d</sup>	✓ <sup>d</sup>

<sup>a</sup> Addressed in later versions of the SW Simplex; missing in the early specification.

<sup>b</sup> Assuming a RTOS and real-time aware hardware

<sup>c</sup> Assuming an independent power source for the FPGA

<sup>d</sup> When used with triple modular redundancy (TMR)

TABLE 1

The Hardware Simplex Architecture handles failure modes unavailable to the Software Simplex.

a Hardware Simplex system.

1) Safety Controller Produces Safe Output: The system relies on the value of a safety controller to provide a fail-operation mechanism if the complex controller does not work. If the system is currently in a safe state, the safety controller should keep the system in a safe state. Furthermore, it should eventually bring the system into the recoverable region (figure 3), where control can be given back to the complex controller.

2) Decision Module Switches Control Before System Leaves Safe State Region: The decision module needs to switch to the safety controller before the system leaves the safe state region. If this does not happen, the complex controller can drive the plant into a state where the safety controller can not recover. For this reason, the time needed for the system state to from inside the recoverable region to outside the safe region (figure 3) should be larger than one control loop iteration (the system should have a chance to use the safety controller before leaving the safe region). If this is not the case, the recoverable region needs to be made smaller for safety to be guaranteed. This requirement, along with the safety controller requirement above, allow the system to tolerate unsafe actuation values from the complex controller.

The decision module is not a blocking component that waits from inputs from both controllers. Instead, it runs in parallel with the rest of the system producing a value periodically. An output from the safety controller must arrive to the decision module, but no guarantee is imposed on the complex controller. If a value is not received from the complex subsystem (perhaps because of a non-functional controller or a timing bug in the RTOS), the decision module uses the output from the safety controller. Running this component in parallel does not impose any processor overhead since it is an

3) Type Checker Inspects Complex Controller's Output: The complex controller's output should be fed into a type checker which is part of the decision subsystem. This type checker is responsible for making sure the data is correctly typed and the decision module can correctly process it. For example, a floating point voltage output from the complex controller may be a floating point value NAN (not a number), which may confuse the decision module. If an incorrect type is detected, the type checker can simply pass on the last valid value<sup>b</sup>. This protects the decision module from processing incorrectly typed values.

4) Complex System Interacts in Safe Place: The complex system should only have one interaction point with the other subsystems. In figure 2, this interaction point is the hardware bus module. The design should enforce that the hardware bus module only access sensors inputs from the safety controller subsystem, and logical outputs goes to the type checker of the decision subsystem. The hardware bus module is permitted to have arbitrary bi-directional communication with the rest of the complex subsystem.

This strict interaction condition must be enforced in order to provide protection from logical interaction faults. The combination of the type checker and decision module with correct input from the safety controller will handle the logical interaction faults listed in table I. If other connections were permitted, they could compromise safety by affecting the output of the safety controller or bypassing the checks of the decision system.

5) Hardware Bus Module Provides Electrical Safety: An electrical short circuit can damage the safety-critical hardware components and should be prevented. When the hardware is wiring to the bus, a multifunctional component may attempt to write a value to the bus at the same time and cause a short. This can be prevented by using a short-circuit protected bus interface module [20] or implementing the entire system on a modernilinx FPGA which prevents internal shorts.

6) Power is Managed or Decoupled: Power is a critical resource in embedded systems which may be shared and can create a physical resource sharing fault. A simple way to prevent this is to decouple power between the complex subsystem and the safety/decision subsystems (provide them with independent power sources). Another alternative is to manage the power usage of the complex subsystem and have the ability to shut it down. If the entire system is implemented on an FPGA, separate clock

<sup>b</sup> The type checker can use a default value if no valid value has ever arrived.

regions [21] can be used for the different subsystems. If the complex subsystem uses too much power, its clock can be disabled which stops effective power consumption.

7) Functional Hardware: The hardware components including the sensors and actuators must be robust to failure. An industry standard method for doing this is triple modular redundancy (TMR) [17]. The Hardware

usefulness of the Hardware Simplex Architecture in this real-time safety-critical application.

We investigate practical considerations for using the Hardware Simplex Architecture:

1) Can the system be divided up into a safe controller and a complex controller such that the most likely causes of failure are contained in the complex controller?

- Long list of seven design rules

– Spanning two pages

- Some with multiple paragraphs

# Safety Rules (fixed)

- We changed the safety rules to three categories of necessary properties

We have identified several necessary conditions required by a System-Level Simplex design that are checked by our OSATE architecture checker. The properties can be classified into resource isolation properties, data consistency properties, and data flow properties. We briefly describe each of these, along with associated failures that may occur if they are not present.

# Added Other Lists

- All items are questions
- All questions start with different word
  - Can the system be divided up into a safe controller and a complex controller, such that the most likely causes of failure are contained in the complex controller?
  - Is the System-Level Simplex end-to-end design process effective in the cardiac pacemaker context?
  - How do the resultant safety guarantees compare to those of existing pacemakers?

# Paper Structure

- Original:

- Introduction
- **Related Work**
- HW Simplex Design
- Case Studies
  - Pacemaker
  - Inverted Pendulum
- **HW Simplex Model**
- Conclusions

- Revised:

- Introduction
- SL Simplex Design
- **Design Process**
- Case Studies
  - Pacemaker
  - Inverted Pendulum
- **Related Work**
- Conclusions

# Questions & Comments?

- I'll send out both versions of the paper and this presentation...