Methods for establishing Freedom From Interference and detecting Data Access Violations, Control Flow Violations within Automotive Safety Software

Akhila Shamsunder
Automotive Microcontroller Software division
Infineon Technologies
Bangalore, India
Akhila.Shamsunder@infineon.com

Karan Mundhra
Automotive Microcontroller Software division
Infineon Technologies
Bangalore, India
Karan.Mundhra@infineon.com

Sandeep Chandrashekar
Automotive Microcontroller Software division
Infineon Technologies
Bangalore, India
Sandeep.Chandrashekar@infineon.com

Swasati Baishya
Automotive Microcontroller Software division
Infineon Technologies
Bangalore, India
Swasati.Baishya@infineon.com

***Abstract*— The complexity of automotive software has only increased several folds over the years. Since the automotive systems are life critical it becomes only imperative that software quality is not compromised. Complex software can lead to an increase in systematic software faults which could result in undefined and dangerous behavior in the system. Hence, it becomes imperative to have more stringent mechanisms to avoid and detect faults early in development without compromising the use of the software. The paper describes an approach to detect data access & control flow violations using static analysis. It also describes an additional test method to establish freedom from interference between software elements in a mixed-criticality system**

Keywords— Static Analysis, Control flow Analysis (CFA), Data Access Analysis(DAA), Concurrency issues, Freedom From Interference(FFI), Data Access violation(DAV), Control flow Violation(CFV), Functional Safety, Mixed criticality software

I. INTRODUCTION

Recent years have seen immense technological advances which have revolutionized the automotive industry with a greater focus on software. This has led to the increased complexity of the software. Systematic software faults that persist in such a system then could be one of the major causes of data access and control flow violations. These violations may lead to undefined, and in many cases, catastrophic behavior of the system. Also, in a complex system like automotive, there will be several safety critical and non-safety critical components that coexist with each other. The primary reason for such mixed-criticality systems is to lower the development cost. Applications that are ascertained to be safety critical are developed as safety software (ASIL rated) hence higher development cost, while thers stay at QM (no safety requirement). Since they all are on the same ECU, the interaction between the software components has to be avoided and it becomes even more important for the software components in mixed-criticality systems to exhibit Freedom from Interference (FFI).

In the industry currently, DAA and CFA are done with the help of static code analysis tools which cannot accurately predict the occurrence of data flow/control flow violations at the design level, discrepancies between design and code, and concurrency issues like race conditions, deadlocks, etc., very effectively. Also, there is no systematic approach to achieving FFI between the software components and a method to perform CFA and DAA which could detect the violations at both the design and implementation levels. This paper aims to address this lacuna and provide a systematic and streamlined method to detect these violations.

II. FREEDOM FROM INTERFERENCE

A. Background

Freedom from interference is the absence of any cascading failures which may lead to undefined behavior in the system. One of the primary causes leading to such failures is unintended access to memory regions, primarily global variables and hardware registers. These are classified as the hardware software interface (HSI) elements as per the safety standards (ISO26262). Hence, the safety objective of a system is to avoid such cascading failures, where unintended access to a memory location by a software element leads to the failure of another software element.

B. Proposed Method

In general, tests are always performed to check that the intended memory region achieves the desired value after the completion of the test. This test also ensures that no other memory region is disturbed or corrupted due to a software issue. Test cases specified for unit-level verification are largely reused for FFI verification. These test cases are enhanced to also monitor and evaluate the memory accesses. Any violation of the memory access is reported as fail to the test framework tool.

For example, as depicted in Figure 1, the intended access to hardware resources is C, and D for Func2 and not B. Hence accessing B is a violation.

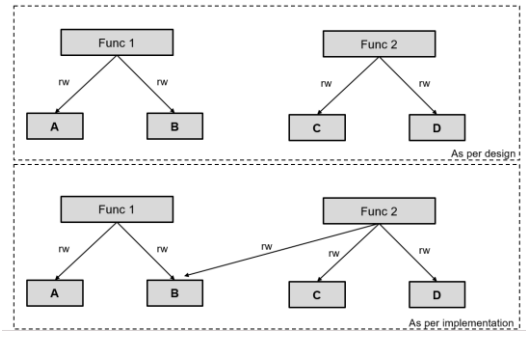

Figure 1: Example of unintended access

The method involves integrating the design methods for hardware resources and global variables and finally verification of the design methods and implementation to formulate a consolidated framework. As depicted in the figure (Refer to Figure 2), the verification involves the below steps

- As part of the detailed design, the intended memory regions that would be accessed by the software are specified with the exact access types read/write/read and write.
- The unit test framework builds a log of memory regions accessed by every test in the software component.
- The FFI Framework tool then takes this input log of memory region access and compares it against the intended memory region specified in the design.
- Any unintended access is then reported as a failure and is feedback to the design and code to make the required correction.

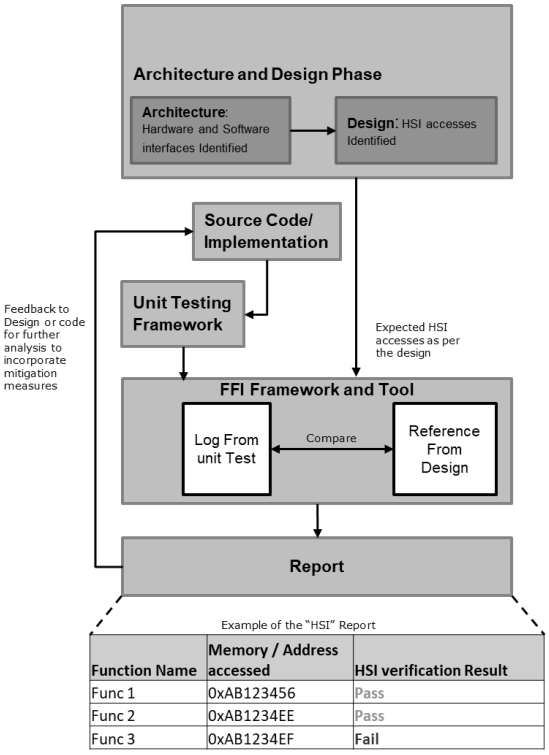

Figure 2: FFI verification Method

III. CONTROL FLOW ANALYSIS

A. *Background*

CFA is performed as a safety measure to overcome control flow violations. By definition, it is a static code analysis method to determine the control flow of a program.

Control flow violations (CFV) can occur due to unintended execution of instructions or function calls. Such violations can lead to unintended behavior in the system. In most cases, these violations are due to incorrect interrupt handling, incorrect data inputs, and incorrect data computation. One of the measures to identify such violations is Control Flow Analysis.

B. *Proposed Method*

CFV can be identified by measures listed below

- Range check of input and output parameters. The boundary and plausible values of a function's input and output parameters are checked and appropriate error reporting is done to the application invoking these functions.
- Measures like requirement-based testing, unit verification testing to cover equivalence class and boundary value analysis, inspection reviews, and safety analysis also aid in identifying CFV.
- In this analysis, each design decision, entry-exit points of functions, lock unlock of spinlocks, etc. of every logical design block are mapped to design requirement and identified by a unique identifier. Each logical block from the design can hence be correctly mapped to a definite code block. Any subsequent block of code or logical block which cannot be mapped to the design or code block respectively can be detected. This verification can be automated or can be part of the manual inspection review. In the proposed approach, a CFA tool along with an inspection review then identifies any incorrect mapping in design and source code indicating a violation in control flow which is fed back as a mitigation measure to design and code (Refer to Figure 3).

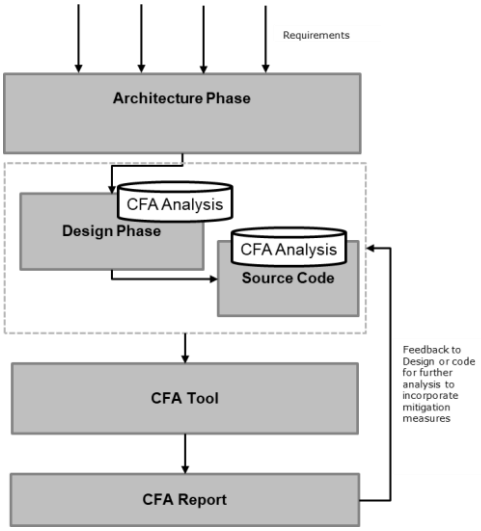

Figure 3: Enhanced Control Flow Analysis depicted in conventional development workflow

IV. DATA ACCESS ANALYSIS

A. *Background*

DAA is performed as a safety measure to overcome data flow violations. By definition, it is a process of analyzing memory regions accessed by software programs in a multi-thread, multi-core, and pre-emptive execution environment.

DFV can occur due to incorrect handling of shared data. and could lead to undefined behavior of the system. In most cases, these violations are due to incorrect/unintended modification of the global data, hardware resources, incorrect handling of critical sections, or insufficient mechanism to avoid any concurrency issues. One of the measures to identify such violations is Data Access Analysis (DAA).

B. Proposed Method

DFV can be identified by measures listed below

- Measures like requirement-based testing, unit verification testing to cover equivalence class and boundary value analysis, static analysis tools, inspection reviews, and safety analysis also aid in identifying DFV.
- Data access analysis is performed during the design and code phase of the development cycle and can be used to identify various issues to avoid catastrophic failures in the system (refer to Table 1). These issues are very difficult to identify and debug in the later stages. The design for software components uses a modeling approach indicating: the hardware software interface specifying which hardware resources are being used by which of the component. Apart from this, the kind of access (read, write, read-write, read-write only by hardware) is specified, and reentrancy level, multicore/partition capability, and synchronous attributes are listed as part of the function description. All of these parameters specified in the design along with the source code are taken into the DAA tool and the output file is then analyzed to identify any violations which are fed back as mitigation measures to design and code (Refer to Figure 4).

TABLE 1. AREAS THAT ARE FOCUSSED IN DAFA

AREA	***Description***
Concurrency issues and Reentrancy	Data race, deadlocks, missing locks, etc., will be identified. Data access violation can also happen if proper protection mechanisms are not applied for functions characterized as reentrant.
Global Data	Global data used and shared across functions in a driver must be justified and proper protection mechanisms must be in place to protect any sequence of instructions that are identified as a critical section. The analysis will focus on access to this global data and identifies the shortcomings.
Hardware Resource	Hardware resources must not be corrupted while being accessed by the driver. The analysis focuses on identifying all functions which access given shared hardware resources and checking whether these accesses are protected using suitable mechanisms. If any violations are identified, then suitable mitigation measures are identified.

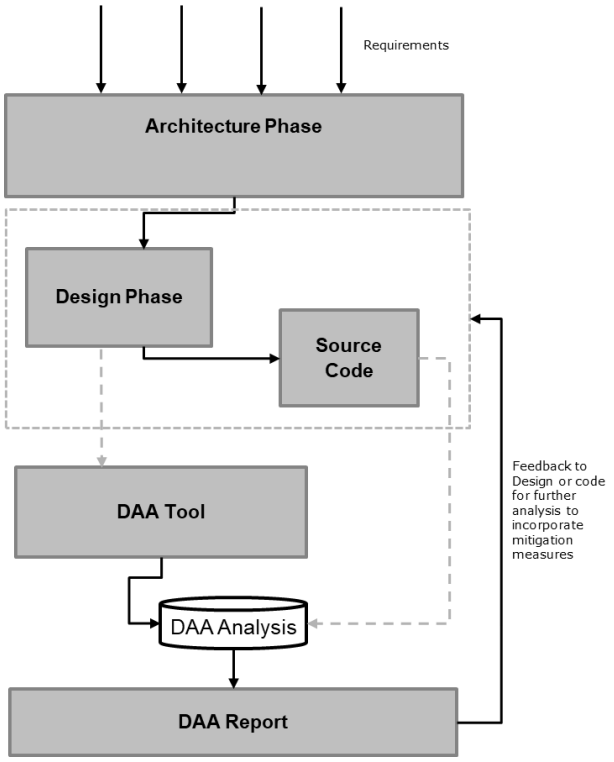

Figure 4: Enhanced Data Flow Analysis depicted in conventional development workflow

V. CONCLUSION

DAA, CFA, and FFI can also be achieved in different methods than what is proposed in the paper. For example, logging a hardware trace at run time into memory and then analyzing this data to identify any violations or fafaultsHowever the approach/method must be analyzed as per the project needs.

The standards governing safety-critical projects generally advise the CFA, DAA, and FFI to be performed as one of the measures. In most scenarios, a tool is used to do this which does not consider all the possible scenarios. Thus it becomes imperative to have the methods proposed in the paper in place to ensure no systematic software faults leak into the final product. These methods have been deployed in safety projects and have helped to establish the safety claim of the product. The framework has been successful in identifying the unintended access of hardware resources that are due to erroneous design or code. CFA, DAA, and FFI verification proposed in this paper have at an early stage helped in identifying violations and thus improving the quality of the product.

VI. REFERENCES

[1] ISO 26262, Road vehicles - Functional safety, ISO copyright office, 2011-11-15

[2] Freedom from Interference for AUTOSAR-based ECUs: a partitioned AUTOSAR

[3] Infineon TriCore Architecture Manual

[4] ISO/IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related Systems, IEC central office, 2010

[5] Sandeep Chandrashekar, Akhila Shamsunder, Swasati Baishya, Sumit Khandelwal; Infineon Technologies; Shared resource analysis for embedded multi-core systems, United States Patent, US11061745B2, 2021-07-1