報告番号	※甲　　　　第　　　　号

主　論　文　の　要　旨

論文題目　Development Toolchain and Platform for Writing MCU-Based Embedded System Software
（MCU ベースの組込みシステムを対象とした開発ツールチェーンとプラットフォーム）

氏　　名　　河田　智明

論 文 内 容 の 要 旨

Microcontrollers (MCUs) are ubiquitous in today’s embedded systems. Their use cases run the gamut from primary processing units in small appliances to internal controllers of application-specific ICs (such as storage controllers). Their increasing processing power allows for larger, more highly-integrated applications that were once only tractable by general-purpose systems. However, with that come all the problems general-purpose systems used to have (and are still having, to some extent), such as development costs and quality issues.

Embedded software uses a technique called partitioning to isolate software faults. Safety-critical system developers (e.g., automotive system developers) find it attractive because they need to integrate software provided by multiple sources, and it allows them to prevent software faults from affecting safety-critical software components. Unfortunately, partitioning has been shunned outside the safety-critical circles because it adds extra costs, increases a runtime overhead, and breaks code when misused. Memory protection is the essence of partitioning and has conventionally been implemented using a processor’s ring protection mechanism. However, owing to the mechanism’s highly generic nature, this approach has led to a significant software overhead.

A perpetual problem in engineering is the development cost. Component-based development (CBD) attempts to address this issue by dividing the system into separate components and enabling code reuse on a by-component basis. This allows for large and complex software to be constructed efficiently from reusable components, significantly reducing development costs and time. This benefit has led to an increasing interest in CBD by embedded system developers. A *component system* is a software framework that standardizes the component interface and the method of combining components to build a functioning system. The TOPPERS Embedded Component System (TECS) is an embedded-oriented component system designed to be integrated into ITRON 4.0-style RTOSes’ configuration systems. However, the support for the TOPPERS third-generation kernels, including TOPPERS/ASP3 and TOPPERS/HRP3, was impeded by their updated API design to accommodate partitioning.

The embedded system landscape is ever-changing. The latest change is the emergence of connected devices or the so-called “Internet of Things”, aiming to optimize embedded systems’ activity by the seamless connection between embedded systems offered by the broad availability of an Ethernet connection. With this seamless connectivity, embedded system security becomes a concern since a vulnerability in embedded systems can be devastating. As such, it would be prudent to leverage the abundant security knowledge acquired through general-purpose systems in embedded systems.

This dissertation takes on three research topics. The first topic explores how to add memory protection support to an existing operating system by utilizing TrustZone for Armv8-M, a hardware-assisted security feature for microcontroller-based embedded systems. During the process, we identify and perform a qualitative comparison of three possible system configurations for TrustZone for Armv8-M. Based on one of such configurations, the SBI (single binary image) scheme, we develop ASP3+TZ, a memory-protection-enabled operating system, by modifying an existing operating system named TOPPERS/ASP3, which does not have memory protection. Finally, we show that the proposed method achieves memory protection at much lower overhead while offering the almost same level of memory isolation as existing operating systems.

The second topic proposes *TZmCFI*, a lightweight control-flow integrity (CFI) scheme for RTOS-based applications. CFI is a class of defensive techniques against control-flow attacks such as Return-Oriented Programming. Although it has been widely deployed in general-purpose systems, such existing implementations are inapplicable to small embedded systems because of architectural differences. TZmCFI embodies variants of several existing CFI techniques to provide a self-contained toolset for building an instrumented application. The toolset is comprised of a modified LLVM-based compiler and a runtime library called *Monitor*, which is isolated from untrusted code using TrustZone for Armv8-M. The modified LLVM code generator implements the traditional shadow stack technique by inserting calls to Monitor. Monitor wraps the application’s interrupt handlers to protect them by *shadow exception stacks*, a variant of the traditional shadow stack technique. The performance evaluation indicates that our shadow exception stack implementation’s runtime overhead is moderate if not significant, whereas our shadow stack implementation incurs an overhead lower than previous works.

The third topic presents a method to componentize the time event notifications from the TOPPERS third-generation kernel for the TOPPERS Embedded Component System (TECS). TECS supports the generation of static API statements, but it is limited insofar as it cannot generate certain complex statements, including those of the time event notifications. Therefore, we propose a TECS generator plugin that generates the complex static API statements required by the time event notifications. We evaluate the proposed time event notification component to demonstrate its runtime efficiency and establish the proposed plugin’s utility.