Embedded Debugger

Project Requirements and Specifications

Schweitzer Engineering Laboratories

Team SEL Embedded-Debugger
Subham Behera, Daniel Ochoa, Howard Potter

September 28, 2023

TABLE OF CONTENTS

I. Introduction **3**

II. System Requirements Specification **3**

II.1. Use Cases 3

II.2. Functional Requirements 4

II.2.1. Exception Handler *4*

II.2.2. System Snapshot Storage *4*

II.2.3. Scripts and TCL Commands 4

II.2.4. Test setup 4

II.2.5. Graphic User Interface 5

II.3. Non-Functional Requirements 5

III. System Evolution **5**

IV. Glossary **6**

V. References **6**

I. Introduction

Schweitzer Engineering Laboratories provides world products such as protective relays that enhance the power grid in many ways such as preventing problems like blackouts and implementing cutting-edge solutions which add cybersecurity and automation [1]. This makes the grid very efficient, however it is not impervious. To minimize downtime and quickly resolve issues, preventative measures must be implemented. This is where the embedded debugger comes into picture.

The embedded debugger strives to collect snapshots of a CPU's core so if the current version isn't performing well, we can easily just call the last saved snapshot without hopefully losing much. This is essential because if such a system is never built, potentially lots of future products would be wasted as no compatible software can run them. This in the end means lots of financial loss.

II. System Requirements Specification

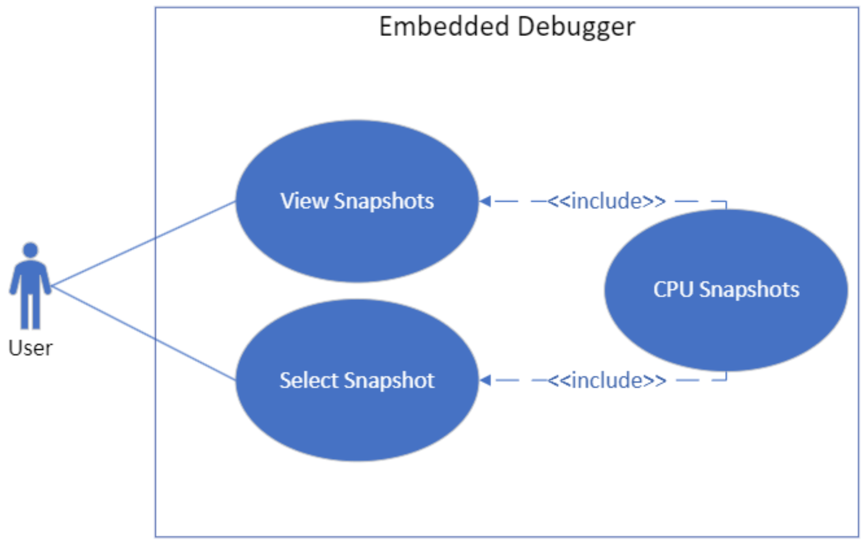
As this project's goal is to create a debugger for a specific SOM (AMD Kira K26) [2], the scope of this project will be contained to this specific SOM (AMD Zynq UltraScale+) [3], and the SOC that runs on this SOM. All stakeholders that are involved in this project will have this specific SOM with SOC.

The requirements of this project will be to collect data that is currently stored within the CPUs and memory of the SOC, and to store it in the system's flash memory as a snapshot. This information will be used for two goals.

Goal one: to store the information for later analysis to learn how an exception occurred to correct any potential error.

Goal two: to use this information to restore the system automatically, preventing extended downtime. A simple GUI will be implemented, to easily identify and display the information stored on fault detection.

II.1. Use Cases



II.2. Functional Requirements

II.2.1. Exception Handler

Description	This application will be an embedded debugger that contains error and fault handling. The embedded debugger must create a system snapshot that retains system information related to the CPU core registers and associated memory.
Source	Internal requirements
Priority	Level 0: Essential and Required Functionality

II.2.2. System Snapshot Storage

Description	This application needs to store the system snapshot and CPU core and memory information in flash memory
Source	Internal requirements
Priority	Level 0: Essential and Required Functionality

II.2.3. Scripts and TCL Commands

Description	This application needs to contain scripts to read the snapshot information. It must also have TCL commands to restore the CPU to the snapshot state
Source	Internal requirements
Priority	Level 0: Essential and Required Functionality

II.2.4. Test setup

Description	This application must contain a test setup that verifies required functionality.
Source	Internal requirements
Priority	Level 0: Essential and Required Functionality

II.2.5. Graphic User Interface

Description	This application can also contain a graphic user interface instead of command lines for ease of function for the user. This would be a desktop application that would display all the necessary information and options to the user.
Source	Internal requirements
Priority	Level 1: Desirable Functionality

II.3. Non-Functional Requirements

Self-Contained

The application should be complete and specific enough to be understood and implemented without additional information. It will include all the necessary details for the user to be able to use all the functionality of the application.

System Compatibility

The application developed must be compatible with the SK-KV260-G development boards provided by Schweitzer Engineering Laboratories (SEL). Compatibility refers to the seamless integration and functioning of the software/system on these boards without any technical constraints or issues.

Programming language

The application must be primarily developed in C/C++ for core functionality. C/C++ are essential programming languages for embedded systems development due to their efficiency and direct hardware interaction capabilities. TCL scripting language should be used for specific tasks, such as automation, board communication, and debugging processes. TCL is a scripting language commonly used for hardware/software interaction, automation, and testing.

Efficient Performance

The application must demonstrate efficient performance under normal operating conditions and expected user loads. The average response time for critical user interactions should be within an acceptable range, defined based on user expectations and industry standards. The application must also be capable of efficiently storing and loading a system snapshot within acceptable timeframes.

III. System Evolution

While considering different systems and the evolution of hardware is critical for most software projects, as this project pertains to a specific set of hardware used for industrial and infrastructure purposes, there is not much to consider. Additionally, as this project will rely on an ARM CPU using ARMv8-A and the ARM Exception Handler [4], it may be portable to other SOCs that are based on an ARMv8-A architecture, however this falls outside of this project’s scope.

IV. Glossary

Error Handling: The process of identifying, reporting, and managing errors or exceptions that may occur during the execution of a program. Error handling is primarily concerned with handling unexpected events, exceptions, or errors that arise as a result of incorrect or unexpected input, software bugs, or other exceptional conditions.

Fault Handling: The process of detecting, isolating, and responding to faults or failures that occur within a system or application. Fault handling deals with managing faults or failures that can occur due to a variety of reasons, including hardware failures, software bugs, environmental factors, or unexpected system behaviors.

GUI: Graphical User Interface. The application in which the user interacts with graphical components.

System Snapshot: A capture of the system's current state at a specific point in time. It represents an image of the system, including the state of memory, CPU core registers, and other relevant system components.

TCL: Tool Command Language. It is a scripting language that is commonly used for various purposes, including automation, scripting, and embedded systems.

System On a Chip (SOC): An integrated circuit that incorporates different components onto one chip, as an integrated system.

System-On-Module (SOM): An integrated circuit board that is modular by design and can accept many different components to suit a need, including SOCs [5].

V. References

[1] “We do our part so they can do theirs, and together we power the future,” SEL, available at selinc.com, https://selinc.com/company/our-part/ (accessed Sep. 28, 2023).

[2] “Kira K26 SOM Data Sheet”, AMD, available at https://docs.xilinx.com/r/en-US/ds987-k26-som/Overview (accessed Sep. 28, 2023).

[3] “Zynq UltraScale+ MPSoC Data Sheet”, AMD, available at https://docs.xilinx.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview (accessed Sep. 28, 2023)

[4] “Arm Architecture Reference Manual for A-Profile Architecture”, Arm Limited, 2023, available at https://developer.arm.com/documentation/ddi0487/latest/ (accessed Sep 28, 2023).

[5] “System-on-Modules (SOMs): How and Why to Use Them”, AMD XILINX, 2023, available at https://www.xilinx.com/products/som/what-is-a-som.html, accessed (accessed Sep 28, 2023)