Embedded Debugger

Project Solution Approach

Schweitzer Engineering Laboratories

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

October 5, 2023

TABLE OF CONTENTS

I. Introduction

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 data collected by Schweitzer Engineering Laboratories (SEL). 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.

The purpose of this document is to show all the major components of this project. By doing this, it can show all the expected major deliverables and even help us developers not forget about working on any part of the project. This means the major intended audience is SEL as they will ultimately use this system. They can reference this document to perhaps see all the features this embedded debugger will have.

II. System Overview

The embedded debugger will be developed using the VITIS software to program a modified mini-computer to act as the controller of the debugger. This controller will have many different functions such as viewing snapshots, taking actions with snapshots, and in general storing all data it collects which can be useful to analyze trends. All of these functions are discussed in detail below.

III. Architecture Design

III.1. Overview

As this project is made to handle exceptions automatically, as well as be accessible for debugging and system restoration purposes, a model in which different systems directly interacted with each other both with and without direct user input was required. With this specification in mind, a Model-View-Controller architecture will be used for this. While this model is generally used for GUI and web application development[1], it will also be suited for this project. See appendix 1 for the model.

III.2. Subsystem Decomposition

The Model in this instance will be the main logic handler. It will interpret data that is being sent from the controller for interpretation, dumping, and restoring. It will be connected to a storage medium of some form which it can read and write from. As this pertains to just reading and writing, it is well suited to being dedicated as a singular component.

The controller is almost self-explanatory. It will control the signals from the main components of the program, as well as the SOM that the debugger will be running on. It will detect any exception thrown, either by the SOM or manually triggered, which will then communicate with the Model to handle the logic of dumping data. A very similar case happens when the exception is being handled by restoring a previous snapshot, again by the SOM's exception or a manual trigger.

The view represents the user interactivity end of this application. From here, a user can send manual signals to the controller. This can range from loading snapshots manually, causing an exception, or viewing logs of the system. All of these requests will be sent to the Controller to handle.

III.2.1. Controller

a) Description

The primary responsibilities include processing logical commands received from the Controller Exception Detector/Handler and orchestrating the collection of system snapshots. This subsystem ensures efficient snapshot capture and coordinates with the Data Storage subsystem for the seamless storage and retrieval of captured data.

b) Concepts and Algorithms Generated

Our approach is to directly access CPU core registers and memory areas using low-level assembly instructions. This approach provides real-time access to the system state. We want to capture exceptions as soon as they occur so we will also be using ISRs to capture snapshots during specific software exceptions. ISRs are routines triggered by specific events, making them suitable for capturing system states during exceptions. Although managing multiple ISRs can be complex, careful design and modularity can simplify the implementation. Each ISR can focus on capturing specific types of exceptions, ensuring a modular and maintainable codebase. We also want to minimize our interrupt handling overhead by using careful optimization and efficient code implementation. This will help ensure that the system's overall performance is not significantly impacted.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
CaptureSnapshot	Data Storage	The Handler will Handler and store it in the Data Storage.
RetrieveSnapshot	Controller Exception Detector/Handler	The Handler will get CPU snapshot from Data Storage and provide it to Controller Exception Detector/Handler.

Services Required:

Service Name	Service Required	Description
SendData	Data Storage	The Handler receives data from the Data Storage.
SendSnapshot	Controller Exception Detector/Handler	The Handler receives data from Exception Detector

III.2.2. View

a) Description

While direct user input into this system is limited, it is an important consideration during the design process of this application. This component of the system will handle just that – the user's ability to interact with this debugger. This includes accepting user inputs to select different menus and send different signals, as well as printing information on screen.

b) Concepts and Algorithms Generated

This component will handle printing onto the screen and accepting input from the user. Entering inputs will be limited to text inputs from the user, as the user interface is planned to be a command line interface. This will also handle loading information required for the user to know important information about snapshots.

With the limited scope of this component, the only algorithms needed will be ones to read data being given to it by the controller through a request, as well as the logic for the menu selection to send requests as intended to the controller.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
SaveSnapshot	Controller	Sends a signal to the controller to save the current information in the CPU as a snapshot.
LoadSnapshot	Controller	Sends a signal to the controller to load a selected snapshot into the CPU.
CauseException	Controller	Send a signal to the controller to manually trigger an exception
ViewLog	Controller	Sends a signal to the controller to send a list of snapshot information (names, dates, cause of exception, etc.)

III.2.3. Model

a) Description

This part of the system will be the backbone of the debugger. This means there will be no user interaction here as it will be in the backend performing tasks based on requests from other parts of the system. Its main job is to process requests and handle data depending on what type of request. Whenever there isn't a request, it passively collects data.

b) Concepts and Algorithms Generated

As this handles storage and retrieving data, we need to think about data structures. Perhaps a stack data structure to store snapshots is the best because it implements last in, first out (LIFO). The most recent snapshot will always be stored at the top and when the most recent snapshot is needed, it can be instantly popped from the top, providing efficiency.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
SendData	Controller	Send the snapshot that was requested.
SendStatus	Controller	Send the status of an action.

Services Required:

Service Name	Service Provided From	Description
SendRequest	Controller	Controller sends what action to take.

IV. Data design

This project will require moving and storing data. The application will read data from the CPU core registers and store it in flash memory. It will also load data into the CPU core registers. The data that is being manipulated will be binary data. We will be using a System Snapshot structure to hold the data. As well as some internal and temporary data structures.

System Snapshot Structure:

- A struct or a collection of variables to store CPU core registers and associated memory states captured during a software issue.

Internal Data Structures:

- Data structures used internally within the software for processing and manipulation of captured data.
- Status Flags: Flags or enumerations can be used to indicate the status of various operations or conditions within the system. For instance, a flag might indicate whether a snapshot capture operation was successful or if an error occurred during the process.
- Arrays, linked lists, or queues for managing multiple snapshots. Data structures for handling stack decoding and exception handling logic.

Temporary Data Structures:

- Buffers: Temporary buffers can be used during the process of capturing system snapshots or during the restoration process. For instance, when reading data from CPU core registers or writing data to flash memory, temporary buffers are employed to hold the data before processing or storage.
- Stacks or Queues: Data structures like stacks or queues can be used temporarily for managing intermediate data during the snapshot capture or restoration process.

V. User Interface Design

As the purpose of this debugger is to catch exceptions to dump and restore snapshots on an embedded system, the User Interface will be kept basic, as doing these actions manually should not be a common occurrence. A Command Line Interface (CLI) will be implemented for user interaction. This will still allow basic information to be printed on screen and allows the user to perform required functions. As this debugger is not meant for consumers or a typical user, it can be assumed that any individual interacting with this machine will be comfortable with a command line interface. This has the added benefit of being very simple, keeping it easy to implement and adapt to any requirements.

This interface will largely consist of a main menu giving the user several options for input. One of these options will be to view the log of saved snapshots. This will provide the user with contextual information on any snapshots available (names, date of origin, cause of exception if applicable, etc.). This will allow the user to view any available snapshot, as well as select one for further usage (loading, examining for further information). An example of this concept can be found in the appendix.

Additionally, from this main menu, the user will also be able to manually create a snapshot of the current CPU's data, for examination and backup reasons. The user will also be able to manually trigger an exception from this menu – this feature may not be included in the finished product, or if it is only when a debug mode is activated, as it is mainly for diagnostic purposes. This can also be decided by an end user, as these options would be configurable.

This menu could be made available whenever the SOM is running, and accessible to any connection made to the SOM (for example, a terminal), as a debugger like this should be otherwise left to run.

VI. 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.

ISR: Interrupt Service Routine. It is a function in embedded systems and operating systems that is invoked when a specific interrupt condition occurs. Interrupts are signals sent by hardware devices or software processes to the CPU, indicating that an event has occurred that needs immediate attention. When an interrupt is triggered, the CPU interrupts its current execution, saves its state, and transfers control to the appropriate ISR to handle the interrupt event.

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 [2]

VII. References

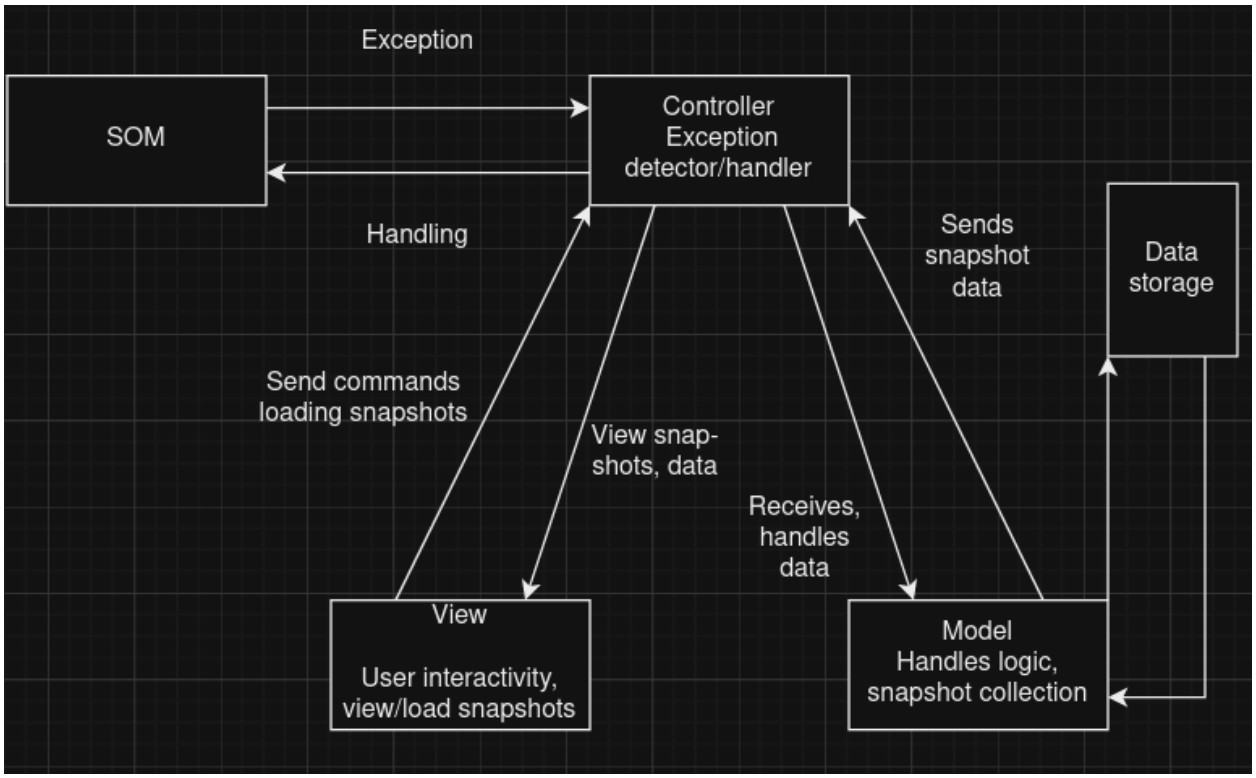
(Dutoit, 2010), 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice Hall, 2010.
[1] “MVC Framework Introduction”, geeksforgeeks, available at https://www.geeksforgeeks.org/mvc-framework-introduction/# (accessed October 5, 2023)

[2] “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)

.

VIII. Appendices

VIII.1. Appendix 1



VIII.2. Appendix 2, UI Mockup Main Menu

```
Debugger
MainMenu

1: View Snapshots
2: Take snapshot
3: Create exception
0: Exit
Option:
```

VIII.3. Appendix 3, UI Snapshot Main Menu

snapshot	date	error code
snapshot-00	10-05-2023 14:10	0
snapshot-01	10-05-2023 14:12	4
snapshot-02	10-05-2023 14:14	9
snapshot-03	10-05-2023 14:16	14
snapshot-04	10-05-2023 14:17	16
snapshot-05	10-05-2023 14:19	19
snapshot-06	10-05-2023 14:20	24
snapshot-07	10-05-2023 14:21	29
snapshot-08	10-05-2023 14:23	33
snapshot-09	10-05-2023 14:24	37
snapshot-10	10-05-2023 14:25	39
snapshot-11	10-05-2023 14:27	41
snapshot-12	10-05-2023 14:29	44
snapshot-13	10-05-2023 14:30	48
snapshot-14	10-05-2023 14:31	53
snapshot-15	10-05-2023 14:33	58
snapshot-16	10-05-2023 14:35	62
snapshot-17	10-05-2023 14:36	66
snapshot-18	10-05-2023 14:38	71