ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

Vol. 3, Issue 5, May 2015

# Trace Management, Debug, Analysis and Testing Tool for Embedded Systems

Bhagyalakshmi C, Pushpa S Tembad

Student, Dept. of Computer Science and Engineering, STJIT, Ranebennur (India)

Prof, Dept. of Computer Science and Engineering, STJIT, Ranebennur (India)

**ABSTACT:** The growing complexity of embedded system hardware and software makes their behaviour analysis a challenging task. In this context, tracing appears to be a promising solution as it provides relevant information about the system execution. However, trace management and analysis are hindered by several issues like the diversity of trace formats, the incompatibility of trace analysis methods, the problem of trace size and its storage as well as by the lack of visualization scalability. In this we present solution for trace management, debug, analysis and testing of embedded systems during development and testing phase. It provides generic solutions for trace storage and defines interfaces and plug in mechanisms for integrating diverse analysis tools even in case of multicore targets.

**KEYWORDS***:* Execution traces, debugging, profiling, embedded systems, multicore, trace management, infrastructure, trace formats, visualization, scalability, analysis tools

## I. INTRUDUCTION

Nowadays, embedded systems are made of increasingly complex hardware and software components. Their hardware architectures are possibly multicore, heterogeneous and distributed. Their software stack is composed of numerous layers including, for example, middleware's to abstract the platform. In this context, application debugging and performance optimization become tremendously difficult tasks. In this paper we focus on tracing and trace management, which address the above issues by gathering information about an embedded system execution and then reasoning about it. However, we do not tackle trace collection mechanisms, which should be ideally designed to minimize intrusivity, possibly by using hardware support. In our work we consider the issues that need to be managed after the trace collection. Namely, we focus on four of them: the heterogeneity of trace formats, the storage of large traces, the management of the trace analysis flow and the visualization.

**Heterogeneity of Trace Formats**: There are multiple trace formats. In most cases, a trace format is closely related to a specific type of application or platform and they are designed together to fulfill specific needs. This approach tends to associate a format with a specific debugging framework. This prevents analysts from using external tools and does not help the diffusion of the techniques they implement.

**Storage of Big Traces**: Execution traces of embedded systems need to include low-level events such as CPU activity, interruptions, context switches, memory accesses, etc. A trace collected even for a very short execution may contain a large quantity of information, which translates into a large data volume (from MB to GB). As trace analysis may consider random parts of a trace, an efficient management of trace storage is mandatory.

**Trace Analysis Flow**: Different treatments are often needed to understand traces. Statistics provide general information about application behavior, while pattern recognition or extracting information and synthesizing the

trace representation. Besides, filters or noise elimination processes help to reduce the amount of information. Trace analysis can be performed by applying such treatments on raw data, or within a flow where the result of
one computation is reused as an input of another (for instance, one to filter the trace, another one to process it, and a last one to visualize the result). Usually, because of the variety of analysis techniques and tools, output data is not

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Issue 5, May 2015**

standardized. Thus, the analysis flow requires an adaptation to enable data sharing between tools. This leads to a strong software complexity, whereas output data standardization would have provided a straightforward compatibility.

**Visualization Scalability**: An embedded system execution trace usually contains too much information to be entirely represented on the screen. Furthermore, different information needs to be represented differently. For example, time views are dependent of information granularity and execution duration, while structural representations depend on the number of entities. To solve these issues, analysts need synthetic representation of traces, where information loss is controlled and quantified, but still enables the detection of hot spots.

## II. RELATED WORK

### 1) Debugging Embedded Multimedia ApplicationExecution TracesThrough Periodic Pattern Mining

AUTHORS**:** Mr, Frederic Petrot, Mr, Hiroki Arimura, Mr, Gilles Sassatelli.

Nowadays, multimedia embedded systems populate the market of consumer electronics with products such as set-top boxes, tablets, smart phones and MP4 players. This highly competitive industry pressures manufacturers to design better products (better in terms of performance and power consumption), always providing new features, and before the competitors(time-to-market). Moreover, semiconductor manufacturers need to provide costumers not only with the hardware, but also with software that simplifies the development of applications. Therefore, a significant part of this pressure is passed on to software developers, who need to develop, debug and optimize their software in as little time as possible while dealing with platforms which never cease to increase in complexity.

### 2) Real-time, Unobtrusive, and Efficient Program Execution Tracing with Stream Caches and Last Stream Predictors

**AUTHORS:** Vladimir Uzelac, AleksandarMilenković, Milena Milenković, Martin BurtscherECE Department, The University of Alabama in HuntsvilleIBM, Austin, Texas, ICES, The University of Texas at Austin

This paper introduces a new hardware mechanism for capturing and compressing program execution traces unobtrusively in real-time. The proposed mechanism is based on two structures called stream cache and last stream predictor. We explore the effectiveness of a trace module based on these structures and analyze the design space. We show that our trace module, with less than 600 bytes of state, achieves a trace-port bandwidth of 0.15 bits/instruction/processor, which is over six times better than state-of-the-art commercial designs.

Debugging and testing of embedded processors is traditionally done through a JTAG port that supports two basic functions: stopping the processor at any instruction or data access, and examining the system state or changing it from outside. This approach is obtrusive and may cause the order of events during debugging to deviate from the order of events during “native” program execution without interference from debug operations. These deviations can cause the original problem (e.g., a data race) to disappear in the debug run. In addition, stepping through the program is time consuming or programmers and is simply not an option for debugging real-time embedded systems, where setting a break point may be impossible or harmful. A number of even more challenging issues arise in multi-core systems. They may have multiple clock and power domains, and we must be able to support debugging of each core, regardless of what other cores are doing. Debugging through a JTAG port is not well suited to meet these challenges.

### 3) SoC-Trace Infrastructure Benchmark

**AUTHORS:** Generoso Pagano, VaniaMarangozova-Martin

This document presents the performance benchmark performed on two different implementations of the SoC-Trace Infrastructure. The first implementation, based on a distributed database, the second implementation has been

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Issue 5, May 2015**

realized later in order to explore the alternative centralized approach. This document describes also the latest evolutions of the SoC-Trace Infrastructure.

One of the main objectives of SoC-Trace project is the development of a trace management infrastructure, storing traces and facilitating the access to trace data for analysis tools. The first implementation of SoC-Trace Infrastructure deals with the storage problem using a distributed database solution, where each trace has its own database for storing events and analysis results, while a central database is used to store trace metadata and organize trace databases. In order to avoid the logical replication of this distributed solution (each trace database has the same schema) and the creation of a new database for each new trace, a centralized design seemed to be an interesting option. We therefore designed a centralized version of the database and implemented the corresponding version of the SoC-Trace Infrastructure software library, in order to compare the two different approaches.

## III. EXISTING SYSTEM

Traditionally, raw trace data are stored in files (event logs): this approach has the advantage of simplicity and can achieve good storage performance with binary formats, but it has not special support for optimized random trace accesses or basic processing, like text filtering, class filtering. Thus, the analysis requires to load the whole file into main memory.

Other approaches propose the use of a structured trace file, more suitable for specific kinds of access, like a frame-based file format for fast time-guided navigation, or even more structured file formats where the access is optimized in both time and space (processes) dimensions. These approaches facilitate the access to trace information only in a fixed and limited number of dimensions and are not flexible for arbitrary selections. The existing tool called Console_IO supports only the single core trace messages. This has very less GUI features.

**Disadvantages:**

- No support for optimized random trace accesses or basic processing, like filtering.
- There is no support for tracing multicore processing traces messages. Only single core trace messages.
- No support for custom logging of trace messages for the GUI for the further future reference.

## IV. PROPOSED SYSTEM

We propose a new approach to address this shortcoming. A trace management infrastructure for embedded systems. It addresses the problems of trace formats heterogeneity, large traces management, trace analysis flow setup and scalable visualization.

To tackle the problem of format heterogeneity we proposed a generic data-model able to represent not only raw data but also meta information about the trace and analysis results. The expressiveness of our model has been validated by experiments with several real trace formats. For the future, we foresee refinements to our

data-model induced by new formats conversions. We are also interested in the introduction and the efficient implementation of new types of analysis results, supporting, for example, multi-trace analysis.

**Advantages:**

- Tracing of multicore processing traces messages with separate window for each core.
- Support for optimized random trace accesses or basic processing, like filtering.

- Custom logging of trace messages for the GUI for the further future reference.
- Separation between the command entry window and display window which gives clear view to the user of the tool.

## V. DESIGN OF THE SYSTEM

The purpose of the design is to plan the solution of the problem specified by the requirements document. This phase is the first step in moving from problem to the solution domain. The design of the system is perhaps the most critical factor affecting the quality of the software and has a major impact on the later phases, particularly testing and maintenance. System design describes all the major data flow, use cases, activities as well as major modules in the system.

**Existing system Architecture:**

The existing system supports only the single core trace messages and can only be connected to only one device/target at time. This has very less GUI features and is incapable to satisfy the inherent requirements from multiple clients in terms of logging schemes, scripting, maintaining a sessions. To address this problem, we consider a Console_IO for multiple cores as well as multiple devices/targets called ConsoleIO_Multicore.

**Proposed System Architecture:**

The proposed architecture of system is a layered one. The individual layers consist of components, which have to communicate with each other, and also to client applications through CSM (Client-Server Management) component. Here sending commands to the target device (physical or logical) and receiving responses from the same target device will be handled by the Data Flow Control module. Data Interpreter realizes the basic functionality of the Data Interpretation. Depending on the interpretation files, the clear text input from the client is converted to target recognizable hex data stream. Similarly the target data received from DFC component is interpreted and converted to clear text representation by the Data Interpreter. Client-Server Management acts as the only single-point interface between the clients and other server components, like Data Flow Control components (DFCs), Data Interpreter components (DIs), etc.

ConsoleIO_Multicore is an application with the complete functionality of Monitor program. Apart from Monitor’s functionality, ConsoleIO_Multicore provides easy to interact menus and tool bars to Connect / Disconnect / Switch between the devices.

## VI. MODULE DESCRIPTION

**Number of Modules:**
After careful analysis the system has been identified to have the following modules:

1. Client-Server Management Module.
2. Data Flow Control Module.
3. Data Interpreter Module.
4. Application Module.

**MODULES DESCRIPTION:**

**1. <u>Client-Server Management Module:</u>**

Client Server Management (CSM) is a C++ based COM component out-of-process server. CSM acts as the only single-point interface between the clients and other server components, like Data Flow Control components (DFCs), Data Interpreter components (DIs), etc.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Issue 5, May 2015**

All communication between clients & CSM as well as other servers like DFCs & DIs take place through published COM interface of CSM. Synchronous events from CSM to Clients are supported through connection point. Asynchronous inter-process communication between CSM and clients is implemented using message queue. CSM maintains a queue for each connected client and places response messages in it. It is responsibility of the client to clear the messages from its message queue. And, if the client does not flush the messages from its queue, queue may overflow if virtual memory gets exhausted.

### 2. Data Flow Control Module:

Data Flow Control (DFC) component, which is a device dependent component, is a C++ based in-proc COM server. The DFC COM server also has a complimentary presentation module, which is responsible for displaying windows UI related to configuration information of the device, e.g. interface settings etc. This presentation module is used by the client application in its process space.

DFC COM server communicates with the target device i.e. sends commands to the target device (physical or logical) and receives responses from the same target device. For this, DFC COM server implements the communication protocol required for communicating with the target device. DFC COM server also provides services to other TTFis COM components i.e. Client Server Management (CSM) and Data Interpreter (DI.). For this DFC implements COM interfaces as well as supports interfaces, which are the out-going interfaces of DFC.

### 3. Data Interpreter Module:

The Data Interpreter is a C++ based COM component, in-proc server. Basically it replaces the old Monitor.Exe-Win32 Console Application's Data Interpretation Part. The following are the important components of Data Interpreter.

- The Basic Data Interpreter**.**
- The Command-Response Data Table.
- Sink Interface for DFC Component**.**
- Connection Point Interface for CSM.

It realizes the basic functionality of the Data Interpretation. Depending on the .trt files, the clear text input from the client is converted to target recognizable hex data stream. Similarly the target data received from DFC component is interpreted and converted to clear text representation by the Data Interpreter. Data Interpreter also generates command tokens (clear text tokens) which form a part of a command from its internal database for filling up the command prompt of the Application on TAB Key Press or F3 Key Press on the ConsoleIO_Multicore.

### 4. Application Module:

ConsoleIO_Multicore is an application with the complete functionality of existing Monitor program. Apart from Monitor's functionality, ConsoleIO_Multicore provides easy to interact menus and tool bars to Connect / Disconnect / Switch between the devices. Once the connection to device is established, the main window of the application behaves like a editor through which user can interact with the device by sending commands and receiving responses from the device.

## VII. SIMULATION RESULT

**Importing traces of various sizes into the system**: We imported traces of different sizes, ranging from 5.5MB (100 thousands of events) to 2.75GB (50 millions of events), measuring the time needed to perform the import operation, with and without indexing. Import time (Figure 1.a) grows linearly with trace size in both cases, as proved with a linear regression showing a co-efficient of determination R2 of $1 - 10^{-4}$. Import times keep reasonable values even for huge traces (without indexes about 7.5 minutes for a 2.75GB trace). Using indexes, the import times grow by about 75%.

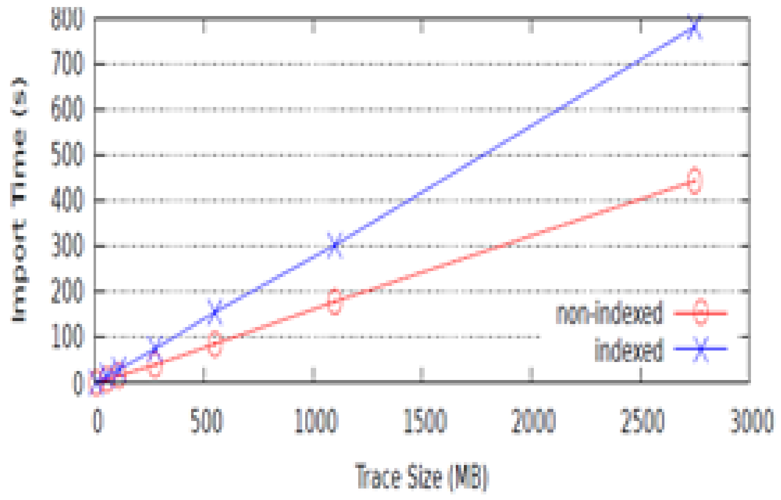

Fig 1.a. Import time for traces of various sizes, with and without indexing

**Querying a given trace over different dimensions:** A great advantage brought by the use of a database for storing traces is the flexibility it offers when performing requests in various dimensions. Using a synthetic trace of 2 million events, we performed requests to retrieve events respectively in a given time interval (a), from a given producer (b), of a given type (c), or having a given value as a parameter (d). For each request, the result set has the same size (20000 events). No indexing has been used in databases. The time needed to _lter trace events using each of the four different dimensions (Figure 1.b) remains in the same order of magnitude. This confirms that the joint use of a well designed data-model and database technology lets trace analysts explore a given trace from different perspectives at a comparable cost. On the contrary, a structured-_le trace format as OTF [6] optimizes only producers and time dimensions.

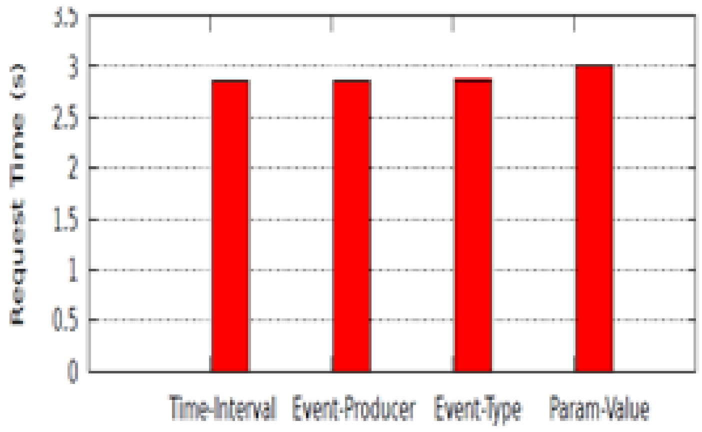

Fig 1.b.Time to retrieve 20000 events from a trace of 2 millions events using various dimensions for filtering.

## VIII. CONCLUSION AND FUTURE WORK

A trace management infrastructure for embedded systems, it addresses the problems of trace formats heterogeneity, large traces management, trace analysis flow setup and scalable visualization.

To tackle the problem of format heterogeneity we proposed a generic data-model able to represent not only raw data but also meta information about the trace and analysis results. The expressiveness of our model has been validated by experiments with several real trace formats. For the future, we foresee refinements to our data-model

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Issue 5, May 2015**

induced by new formats conversions. We are also interested in the introduction and the efficient implementation of new types of analysis results, supporting, for example, multi-trace analysis.

## ACKNOWLEDGEMENT

I consider it is a privilege to express my gratitude and respect to all those who guiding me in the progress of my paper.

I wish my grateful thanks to **Prof, Pushpa S Tembad** $_{\textbf{M.Tech}}$ project guide, for invaluable support and guidance.

## REFERENCES

[1] Debugging Embedded Multimedia Application Execution Traces Through Periodic Pattern Mining. By:Mr, Frederic Petrot, Mr, Hiroki Arimura, Mr, Gilles Sassatelli, Mr, Takashi Washio, Mr, Jean-Franc¸oisBoulicaut,
[2] Real-time, Unobtrusive, and Efficient Program Execution Tracing with Stream Caches and Last Stream Predictors. By: Vladimir Uzelac, AleksandarMilenković, Milena Milenković, Martin Burtscher*ECE Department,*
[3] SoC-Trace Infrastructure Benchmark. By: Generoso Pagano, VaniaMarangozova-Martin.
[4] Summarizing Embedded Execution Traces Through A Compact View. By:*Carlos Prada-Rojas, Miguel Santana, Serge De-Paoli, Xavier Raynaud.*
[5] Large Event Traces in Parallel Performance Analysis. By: Felix Wolf, Felix Freitag, Bernd Mohr, Shirley Moore, Brian Wylie.
[6] Debug and Trace for Multicore SoCs. By: William Orme. ARM Limited