Analyzing ARM CoreSight ETMv4.x Data Trace Stream with a Real-time Hardware Accelerator

Seyed Mohammad Ali Zeinolabedin, Johannes Partzsch, and Christian Mayr
Chair of Highly-Parallel VLSI Systems and Neuro-Microelectronics, Technische Universität Dresden, Dresden, Germany
Email: {ali.zeinolabedin, johannes.partzsch, christian.mayr}@tu-dresden.de

***Abstract*— Debugging and verification of modern SoCs is a vital step in realizing complex systems consisting of various components. Monitoring memory operations such as data transfer address and value is an essential debugging and verification feature. ARM CoreSight technology generates a specific debug trace stream standard to monitor the memory without affecting the normal execution of the system. This paper proposes a hardware architecture to analyze the debug trace stream in real-time. It is implemented on the Xilinx Virtex xc6vcx75t-2ff784 FPGA device and can operate at 125 MHz and occupies less than 8% of the FPGA resources.**

Keywords—Debugging, System analysis and design, System-on-chips, System verification.

I. INTRODUCTION

Current multi-core SoCs consist of many processing engines, different peripherals, and networks on-chip hierarchies [1]-[4]. The software running on such platforms is complex and requires extensive performance analysis, behavior profiling as well as bug fixing. International Business Strategies (IBS) has reported that software development at 22nm technology accounts for 75% of the total cost (79M$ vs. 26M$ for hardware) [1]. Many state-of-the-art SoCs generate debugging information through an embedded trace mechanism without influencing the program execution such as ARM CoreSight technology. However, current approaches can only process debug data offline and up to a few seconds. For example, if there is 4 GB trace buffer and the recording speed is 10 Gbit/s, it can only record up to 3.2s.

Fig. 1 depicts a general SoC design and verification chain utilizing a debug trace stream. Generally, the trace decoder and analyzer units are realized in software that in turn require large memory footprints and violate real-time operation. It is mainly because of the complex nature of the debug trace encoding mechanism. A novel approach has been recently introduced to overcome these issues by designing the dedicated hardware accelerators to process the debug trace stream in real-time. As shown in Fig. 1, a highly compressed debug trace is first decoded in the trace decoder unit in real-time. It then translates the decoded information into meaningful packets. The trace analyzer unit can subsequently perform various analyses over the decoded packets and provides information such as events, various interrupt requests, memory read/write operations, and so on [5].

ARM CoreSight ETMv4.x technology, supporting a variety of modern ARM processors, allows the designers to equip the SoCs with varieties of debug trace generators. The key feature of ARM CoreSight technology is introducing a separate dedicated bus that is shared between all debug trace sources without halting the program execution as shown in Fig.2. For example, Embedded/Program Trace Macrocell (ETM and PTM) units are introduced to debug the processors with flexible and various options. Whereas, Instrumentation Trace Macrocell (ITM) generates a software application driven trace [10]. In Fig. 2, Cross Trigger Interfaces (CTIs) select which signals should be sampled or triggered and they are connected using a Cross Trigger Matrix (CTM). Debug Access Port (DAP) and Debug Advanced Peripheral Bus (APB) provide real-time access to all debug components. Finally, all generated traces are integrated into a single trace stream by Funnel and Trace Port Interface Unit (TPIU) outputs it [5], [6].

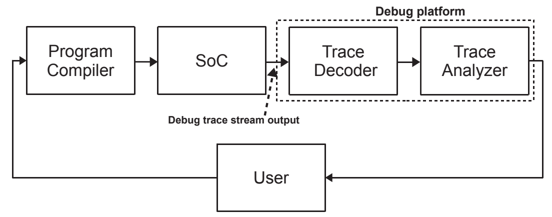


Fig. 1. SoC design and verification chain.

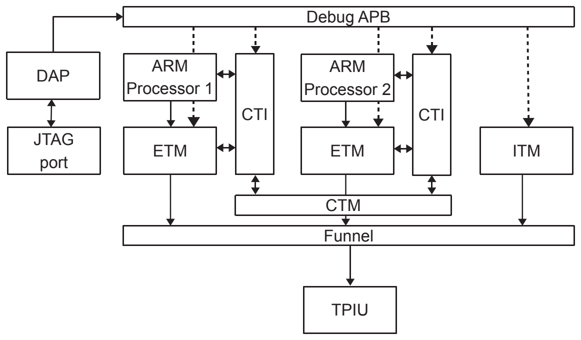


Fig. 2. Block diagram of CoreSight technology based on [5], and [6].

ETMv4.x generates two separate debug traces to monitor the instruction and data buses with distinct source IDs [6]. Each trace unit can be configured to explicitly trace some instructions, to enable various trace unit resources, and to enable/disable data tracing. Trace unit resources can be counters, sequencer, external inputs/outputs, single instruction or data address comparators, instruction or data address range comparators, and so on (12 different resource types in ETMv4). The data trace unit is specifically tracing the memory involved operations. It generates varieties of packets to track the recent data transfer addresses and values and the timestamps.

The recent works are mainly focusing on realizing the trace decoder in hardware such as [7]. It introduces the latest general real-time hardware architecture of ETMv4.x decoder. It consists of two-levels decoders named as L1 and L2 decoders. The L1 decoder is shared between all trace units whereas there is a specific L2 decoder for each trace unit i.e. one for instruction debug trace stream and one for data trace stream. L1 decoder is fully configurable and L2 decoder is a packet-length-independent architecture compared to other implementations. [7] supports a data rate of 1 Gbit/s for a specific source ID available in the trace stream. It also provides a sample showcase of the trace analyzer unit that processes the packets and extracts the interrupt and event sources and counts them. [8] and [9] decode the debug trace stream using similar customized architecture, however, it is packet-length-dependant architecture that does not apply to recent debug traces such as ETMv4. [10] introduces a multi-core debug platform that decodes a debug trace in software within a time interval. [11] is utilizing PTM (with only 11 packet types compared to 400 packet types in ETMv4) to do code reuse attack application, however, PTM implementation details are not explained.

To our knowledge, this paper proposes the first dedicated hardware accelerator to “analyze” the ETMv4.x data trace stream in real-time. To achieve this goal, [7] is exploited to decode the data trace and then a new hardware architecture is proposed to process the packets and extract the memory-related information such as an address, value, and time.

II. ARM CoreSight ETMv4.x Trace Stream

An ETMv4.x debug trace stream consists of a byte-based packet protocol. Each packet contains a single header followed by zero or more payload bytes. The size of some packets is not fixed and needs to be determined after analyzing the packet [6]. An instruction trace stream is comprised of two different elements. There are P0 elements and other instruction trace elements. Similarly, a data trace stream is comprised of three different data trace elements. These are P1 elements, P2 elements, and other data trace elements. P1 and P2 elements contain data transfer addresses and data transfer values, respectively. Other data transfer elements comprise trace info, time stamp, and so on [6].

For example, if the processing element runs a load instruction, the load instruction is traced as a P0 element and the address and data are traced as P1 and P2 elements in the data trace stream. Therefore, P1 and P2 elements can be utilized to monitor the memory involved operations such as address and value. P0, P1, and P2 elements are associated with right-hand and left-hand keys. Fig. 3 shows that a P0 element has only a right-hand key, a P2 element has only a left-hand key whereas a P1 element has both right-hand and left-hand keys. These keys behave like a parent-child structure to relate P0, P1, and P2 elements. The maximum number of right-hand keys and left-hand keys of P1 elements (defined as p1_right_key_max and p1_left_key_max) are specific to each implementation and must be retained between packets.

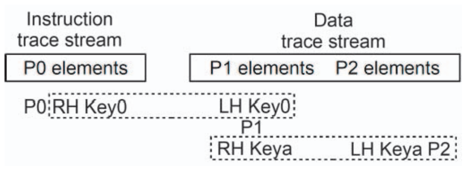


Fig. 3. P0, P1 and P2 elements relationship. P1 element has only one child P2 element. P1 and P2 elements contain data transfer addresses and data transfer values, respectively.

Table I: ETMv4 data trace packets

Packet Type	Format	Number of Headers
P1	1, 2, 3, 4, 5, 6, 7	111
P2	1, 2, 3, 4, 5, 6	76
Reserved	-	64
Others	-	5

Table II: Global parameters shared among all packets

name	Description
Timestamp	Latest timestamp value
address_regs [0:2]	data addresses
p1_left_key	The left-hand key of a P1 element
p1_right_key	The right-hand key of a P1 element
p2_left_key	The left-hand key of a P2 element
p1_index	The index for a P1 element

ETMv4 data trace stream consists of 7 and 6 different formats of P1 and P2 elements, respectively [6]. Some have fixed payload sizes and the rest have a variable payload size. However, all of them are not necessarily available in every data trace stream. ETMv4 has an Alignment Synchronization (A-Sync) packet to identify the boundary of a new trace and a trace info data trace packet indicating the beginning of the process. Furthermore, if global timestamping is enabled, it generates the Timestamp packets periodically. As listed in Table I, 187 packet headers out of 256 are devoted to various P1 and P2 elements, 64 packet headers are reserved, and the remaining 5 headers are for other packets such as Timestamp and so on.

During the analysis of a data trace stream, there are some parameters, listed in Table II, that are globally shared between all receiving packets to calculate the various keys and data transfer address/value. In other words, for some of P1 and P2 packets, the required information is inferred from these global parameters. Therefore, these parameters are required to be updated dynamically while processing the current packet. Some of these packets are very simple to be implemented like P2 Format 4, whereas some like P1 Format 1 has 16 different modes or P2 Format 5/6 can implicitly generate up to four P1 elements in addition to its P2 elements. The latter ones are highly complex and inherently inclined to software implementation to extract and bind the information. The p1 index is a parameter required to calculate the data transfer address spanned over several P1 elements which are sharing the same left-hand keys.

III. Proposed Real-Time Hardware Accelerator for ETMv4 data trace Analyzer

To analyze the ETMv4 data trace stream, it should be first decoded by a Trace Decoder (as shown in Fig. 1). The trace decoder in [7] is utilized to extract the packet in real-time. [7] introduces an architecture which in average extracts two packets per cycle. The decoded packets are then analyzed using the proposed architecture shown in Fig. 4. The Decoded packets are first analyzed in "P1 Packet Analyzer", "P2 Packet Analyzer" and "Global Parameter Updater" units. "P1 Packet Analyzer" unit monitors the decoded packet in real-time and generates the {address, left_key, right_key, index, format} for each P1 element. Likewise, the "P2 Packet Analyzer" generates the {left_key, data, format} in real-time for each P2 element. There are some P1 and P2 formats generating more than one left/right-hand keys and data. Besides, because there can be two packets at the output of the Trace Decoder, there are varieties of P1 and P2 packets combinations. The "Global Parameter Updater" unit monitors the decoded packets and defines different cases as p1_p2_status.

The most common case is receiving a P1 and P2 packets with a variable payload length. In this case, there is only a single valid output generated by the Trace Decoder. So, the global parameters shared among all packets (given in Table II) are accordingly updated and transmitted to the "P1_P2 Combiner" unit. The first special case occurs when either of P2 Format 5 and P2 Format 6, which are variable payload packets, is received. The former one has up to four inferred P1 elements and one P2 element. The latter one has up to four inferred P1 elements and two P2 elements. In both cases, the "Global Parameter Updater" unit should track global parameters and update them. Hence, these cases require having up to two {data, p2_left_key} and up to four {p1_left_key, p1_right_key, p1_index} as shown in Fig. 4. The second special case occurs when there are zero payload P1 and P2 packets coming in sequence. In this case, they are decoded by the Trace Decoder at a single clock cycle. This leads to four different combinations of (P1, P1), (P1, P2), (P2, P1), and (P2, P2). Another special case happens when P1 Format 5 is received. It has up to four P1 right/left-hand keys that should be inferred by the "Global Parameter Updater" unit. All of these main cases are recognized by the "Global Parameter Updater" unit and passed to the "P1_P2 Combiner" unit as p1_p2_status.

"P1_P2 Combiner" unit should store {address, left-hand key, right-hand key, index} of each receiving P1 elements for all possible P1s (defined by p1_left_key_max). This information is utilized together with coming P2 elements to bind the data transfer address and value. To realize this efficiently, two look-up tables are exploited as shown in Fig. 4. The size of LUT0 and LUT1 are defined by p1_left_key_max and p1_right_key_max, respectively. The given address of the current P1 packet is stored at the LUT0 cell pointed by the current p1_left_key. Whereas, its {p1 index, p1_left_key} is stored at LUT1 cell pointed by the current p1_right_key as shown in Fig. 5. Once a P2 packet is analyzed and passed to "P1_P2 Combiner" unit, the corresponding {p1_index, p1_left_key} from LUT1 is first retrieved using p2_left_key (refer to Fig. 3, p2_left_key equal to p1_right_key). Subsequently, the corresponding address is retrieved from LUT0 using p1_left_key that is read from LUT1. Using this approach, P1 and P2 can be combined and as a result, there would be at least one data transfer address/value at the output.

It is worth mentioning that at the beginning the "Global Parameter Updater" unit looks for the trace info data trace packet to reset all global parameters and then from that point onward all the receiving packet are analyzed. Each time a new trace info

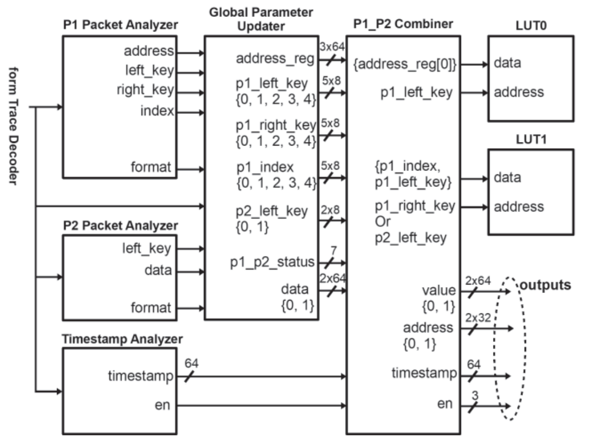


Fig. 4. Proposed hardware accelerator for real-time analysis of ETMv4 data trace stream. The output is the combination of data transfer address and value.

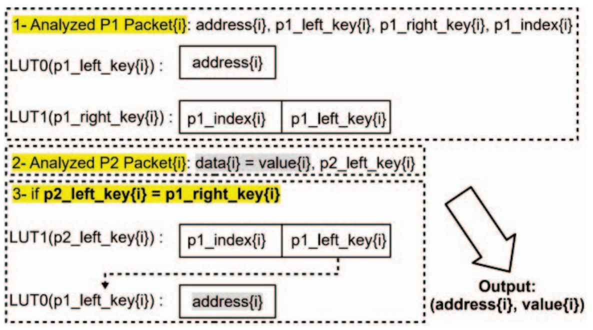

1- Analyzed P1 Packet{i}: address{i}, p1_left_key{i}, p1_right_key{i}, p1_index{i}
LUT0(p1_left_key{i}) :
address{i}
LUT1(p1_right_key{i}) :
p1_index{i}
p1_left_key{i}
2- Analyzed P2 Packet{i}: data{i} = value{i}, p2_left_key{i}
3- if p2_left_key{i} = p1_right_key{i}
LUT1(p2_left_key{i}) :
p1_index{i}
p1_left_key{i}
LUT0(p1_left_key{i}) :
address{i}
Output: (address{i}, value{i})

Fig. 5. Example of how LUT0 and LUT1 are utilized to combine P1 and P2 packets to obtain the output as a combination of (address, value). Having P1 packet analyzed, LUT0 and LUT1 are updated. After analyzing the P2 packet, the extracted data can be combined with the proper address retrieved from LUT1 and LUT0.

data trace packet is received, all global parameters should be set to zero. Another unit in Fig. 4 is "Timestamp Analyzer" which calculates the timestamp if it is enabled in the data trace stream. The final output is at most two pairs of (address, value) with the latest timestamp. Using the proposed hardware architecture, the memory involved operations can be real-time monitored while running applications on the supported SoCs.

IV. FPGA Implementation Results

We have described the trace decoder and the proposed hardware accelerator in Verilog. It is then implemented on the Xilinx Virtex xc6vcx75t-2ff784 FPGA device. Table III lists the device resource usage. It shows that the ETMv4 trace decoder and data trace analyzer occupies 1.96%, 7.82%, and 9.49% of slice registers, LUTs, and internal block RAM, respectively. It achieves the highest operating frequency of 125 MHz and therefore it can support a data rate of 1 Gbit/s [7] and the trace analyzer processes the P1 and P2 elements in real-time.

To verify the functionality of the data trace decoder and analyzer, the various real and synthetic data traces were recorded and generated. Different memory read and write operations were utilized to cover the more varieties of P1 and P2 packets. To validate the design operation, the recorded data traces are first tested with the software-based implementation of the design as shown in Fig. 6. Then the results are used to verify the proposed hardware implementation. Because all P1 and P2 packets are not available in the real data traces, a synthetic data trace stream

Table III: Summary of used hardware resources on Virtex xc6vcx75t-2ff784 (% is calculated w.r.t total available resources on the FPGA device).

Logic Utilization	Trace decoder Used (%)	Trace Analyzer Used (%)
Number of Slice Registers	987 (1.06%)	84 (0.09%)
Number of Slice LUTs	3153 (6.77%)	491 (1.05%)
Number of Block Ram	7 (4.49%)	0 (5%)

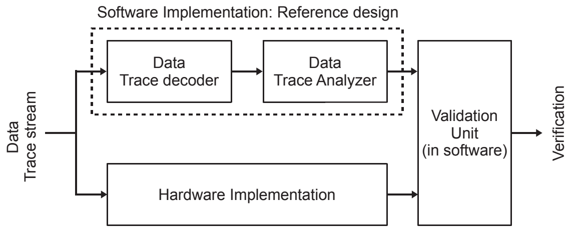
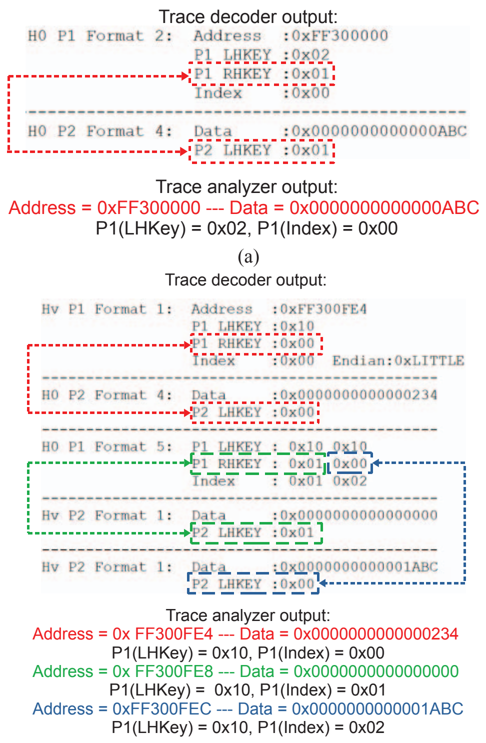


Fig. 6. Proposed hardware architecture verification flow.


Trace decoder output:

```
H0 P1 Format 2:  Address   :0xFF300000
                 P1 LHKEY  :0x02
                 P1 RHKEY  :0x01
                 Index     :0x00
---------------------------------------------
H0 P2 Format 4:  Data      :0x0000000000000ABC
                 P2 LHKEY  :0x01
```

Trace analyzer output:
Address = 0xFF300000 --- Data = 0x0000000000000ABC
P1(LHKey) = 0x02, P1(Index) = 0x00

(a)

Trace decoder output:

```
Hv P1 Format 1:  Address   :0xFF300FE4
                 P1 LHKEY  :0x10
                 P1 RHKEY  :0x00
                 Index     :0x00  Endian:0xLITTLE
-----------------------------------------
H0 P2 Format 4:  Data      :0x0000000000000234
                 P2 LHKEY  :0x00
-----------------------------------------
H0 P1 Format 5:  P1 LHKEY  : 0x10 0x10
                 P1 RHKEY  : 0x01 0x00
                 Index     : 0x01 0x02
-----------------------------------------
Hv P2 Format 1:  Data      :0x0000000000000000
                 P2 LHKEY  :0x01
-----------------------------------------
Hv P2 Format 1:  Data      :0x0000000000001ABC
                 P2 LHKEY  :0x00
```

Trace analyzer output:
Address = 0x FF300FE4 --- Data = 0x0000000000000234
P1(LHKey) = 0x10, P1(Index) = 0x00
Address = 0x FF300FE8 --- Data = 0x0000000000000000
P1(LHKey) = 0x10, P1(Index) = 0x01
Address = 0xFF300FEC --- Data = 0x0000000000001ABC
P1(LHKey) = 0x10, P1(Index) = 0x02

(b)

Fig. 7. Two sample outputs of data trace analyzer. (a) a memory transaction which is monitored in a single P1 and P2 packet. (b) a memory transaction which is encoded in several P1 and P2 packets. (H0, Hv indicating a zero-payload and variable payload packets, respectively).

generator was developed to cover all cases. It receives the data trace stream size as an input and then randomly divides it into m sections beginning with an “A-Sync” packet. Each section is then filled by various packets. There is also an option to force the synthetic data trace generator to add a group of user-desired packets that are not often available in a real data trace stream.

Fig. 7(a) shows the sample output of data trace decoder and analyzer for a single memory operation. The data trace analyzer combines the data transfer address and value using P1 right-hand key and P2 left-hand key. Fig. 7(b) shows a memory operation spanned over several packets. Once a P1 format 5 packet is decoded, two P1 right-hand keys and two P1 indexes are generated. The following two P2 format 1 packets are then utilized to group the new (address, value). It is noticeable that the last two addresses are inferred using their corresponding P1 indexes and the latest address of a P1 packet sharing the same P1 left-hand key with them.

V. Conclusion

In this work, we have proposed the first ETMv4 data trace analyzer implemented completely in hardware to monitor memory operations and to combine data transfer addresses and data transfer values. The proposed architecture is implemented on the Xilinx Virtex xc6vcx75t-2ff784 FPGA device and operates up to 125 MHz. Using this approach, the debugging and verification of complex SoCs are feasible in real-time. The future work would be extending this idea to instruction trace stream and combining with P0 elements.

References

[1] “ITRS Reports,” International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs2.net/itrs-reports.html.

[2] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L. Wang, “Challenges and Trends in Modern SoC Design Verification,” IEEE Design & Test, vol. 34, no. 5, pp. 7–22, Oct. 2017.

[3] G. Cabodi, P. Camurati, S. F. Finocchiaro, F. Savarese, and D. Vendraminetto, “Embedded Systems Secure Path Verification at the Hardware/Software Interface,” IEEE Design & Test, vol. 34, no. 5, pp. 38–46, Oct. 2017.

[4] M. Peña-Fernandez et al., “Online Error Detection Through Trace Infrastructure in ARM Microprocessors,” IEEE Transactions on Nuclear Science, vol. 66, no. 7, pp. 1457–1464, Jul. 2019.

[5] ARM Ltd., “CoreSight Components Technical Reference Manual,” ARM DDI 0314H, 2009.

[6] ARM Ltd., “ARM CoreSight Architecture Specification v2.0,” ARM IHI 0029D, 2013.

[7] S. M. A. Zeinolabedin, J. Partzsch, C. Mayr, “Real-time Hardware Implementation of ARM CoreSight Trace Decoder, ” IEEE Design & Test, (available online in Early Access).

[8] N. Decker et al., “Online analysis of debug trace data for embedded systems,” in 2018 Design, Automation Test in Europe Conference Exhibition (DATE), 2018, pp. 851–856.

[9] A. Weiss, “Trace-data processing and profiling device,” US 9286186 B2, 15-Mar-2016.

[10] A. P. Su et al., “Multi-core software/hardware co-debug platform with ARM CoreSightTM, on-chip test architecture and AXI/AHB bus monitor,” in Proceedings of 2011 International Symposium on VLSI Design, Automation and Test, 2011, pp. 1–6.

[11] Y. Lee et al., “Integration of ROP/JOP monitoring IPs in an ARM-based SoC,” in 2016 Design, Automation Test in Europe Conference Exhibition (DATE), Mar. 2016, pp. 331–336.