Real-time Trace Reconstruction (RTR)

Solution to circumvent the limitations of the Nexus and ARM/ETM on-chip trace

Autor

Matthias Stumpf
Key Account Sales Manager
iSYSTEM AG, Carl-Zeiss-Str. 1
85247 Schwabhausen (Munich), Germany
Tel. +49 (8138) 697151, Fax: +49 (8138) 697146
Email: matthias.stumpf@isystem.com
Web: http://www.isystem.com

Abstract

Developers are using real-time analysis tools in the development stage, to debug real-time events either not observable during “run-stop” debug sessions, or limited by the requirements of the application; such as inherently not freezable systems like engine control or pace makers. Furthermore, engineers are using real-time analysis tools in their final development stage to do performance tuning and coverage or in the test stage for HIL (hardware-in-the-loop) testing and white box testing. A typical requirement is that the real-time analysis tool must not disturb the real-time behaviour of the system.

Today, many current CPU architectures contain an integrated trace or on-chip debug interface like Nexus (www.nexus5001.org) or the "Embedded Trace Macrocell" (ETM) of ARM. New upcoming CPUs will own almost excluding on-chip debug and/or trace interfaces. In the long term they will replace the so-called very powerful bond out chips. However, this on-chip trace interface allows the user to get a good overview about the real-time program execution within the microcontroller. Indeed, restrictions apply according to the interface technology and pose a challenge for the developer – the search for the needle in a haystack.

The presentation covers challenges that come with current on-chip trace interfaces of 32-bit CPU architectures. The developer will learn how to overcome those challenges by using different solutions. Furthermore, engineering managers will see that it is possible to reduce the engineering and verification time for development projects. Also, the test and verification manager can run endurance tests and get an overview of the CPU internals.

Comparison between a traditional In-Circuit Emulator and Nexus On-Chip trace
The in-circuit emulator (ICE) is a tool that offers complex real-time analysis capabilities to enable:

- bus trace
- execution profiling
- data profiling
- execution coverage
- data coverage
- etc.

On the ICE, the CPU bus is used directly in the trigger and filter logic because the emulator is attached to CPU's external bus. For the traditional trace the address, data and control buses (see Fig. 1) – can be recorded in their natural form or get filtered in the complex trigger and filter logic of the ICE.

The on-chip trace was introduced to address issues that emerged with on-going miniaturization and integration of ever faster microcontrollers. When all the memory is located on the chip, there is no need for an external bus any more. Although this reduces the pin count as well as raising the maximum operation frequency, a traditional trace is no longer possible.

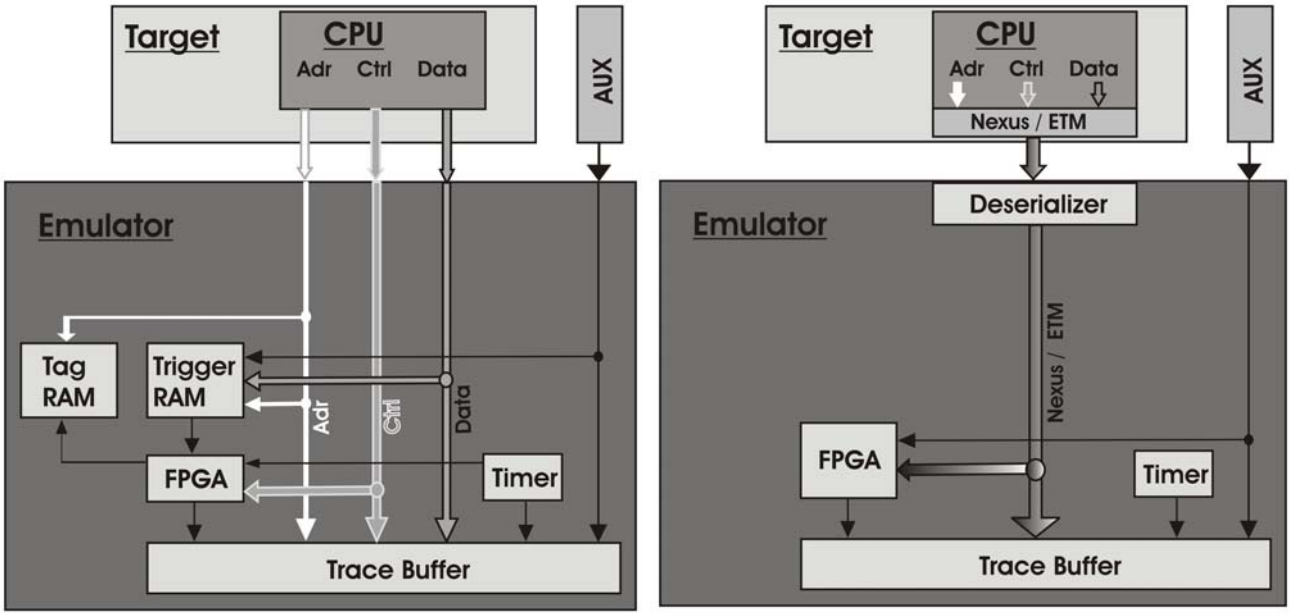

Figure 1: Complex trigger and filter logic on an in-circuit emulator (left) compared to the "fairly simple" Nexus on-chip logic (right)

Nexus message – how does it work?
Nexus – similar to ETM – is a message based protocol that transmits compressed bus information quasi-serially over a few dedicated Nexus pins. Each message is a variable length sequence of bits, including the TCODE header, source processor ID and the number of sequential instructions executed since the last branch taken (see Tab. 1).

The TCODE header identifies the meaning of the remaining bits. The source processor ID specifies the CPU and can be omitted on a single Nexus module configuration. For example, it is needed for the MPC5554 where eTPUs can generate trace too. In general, the CPU has a minimum of 2 Nexus data pins. This can be extended to 4, 8 or 12 data pins if the developer uses more multiplexed ports for the Nexus data pins.

Due to inherent correlations in program flow the program trace is highly compressible, since it is very likely that most of the time the flow is sequential and most branches are short direct branches. However, data access activity is only remotely sequential, so virtually no compression is possible here. Depending on the CPU some modest on-line filtering can be implemented in the Nexus module.

Message Name	Min packet size (bits)	Max packet size (bits)	Packet type	Packet description
Program Trace, Direct Branch Message (DBM)	6	6	fixed	TCODE number = 3
	4	4	fixed	Source processor identifier
	1	8	variable	# sequential instructions executed since last branch taken

Table 1: Message format for a direct branch message (DBM)

Furthermore, the fairly simple trigger and filter logic of the CPU-internal trace interface is surely restricted. The data is captured in a trace buffer of the on-chip emulator. This buffer can take meanwhile a size of some gigabytes to save the compressed data. Afterwards the data will be transmitted to the host. Any kind of program flow reconstruction is performed on the host.

DBM 5		
TCODE = 3	**CPU = 0**	**ICNT = 5**
000011	**0000**	**101**
000011000000101		
MCK #1	**0000**	
MCK #2	**1100**	
MCK #3	**0000**	
MCK #4	**0101**	

In this example the sequence is 13 bits long and the CPU has 4 Nexus data pins, so 4 cycles (MCK) are required to stream it out. The variable length field is padded with zeros to round it up to 16 bits.

Table 2: Serialization of a direct branch message (DBM). The variable length field is padded to the Nexus port width

Exactly this causes the problems. Even the biggest trace buffer will be filled with data after some seconds – partially even minutes – of data recording. The following actions occur to the user:

1.) he must search through these data to find the occasion for the possible bug
2.) even more important, has the possible bug been recorded by the trace tool?

The complex trigger and filter logic of an ICE is desirable, especially if the trace buffer is very large and therefore vast amounts of (not relevant) data are recorded.

Nexus Program Flow Reconstruction – Effects of Nexus on Real-time Tools

All Nexus messages are put in a FIFO and streamed out to the emulator. Note that due to FIFO and streaming latency, the time at which these messages are captured by the emulator is offset and moderately distorted. Every entry in the FIFO requires approximately the same amount of time to be streamed out even if its generation was not linear.

Apart from the deserializer that captures and ‘unfolds’ the Nexus stream, the rest of the trace module is fairly simple compared to an ICE, mainly because there is no trigger or filter logic. Whatever makes it through the Nexus module is recorded along with the timestamp. Any kind of program flow reconstruction is performed on the host. Using the captured Nexus messages and being aware of the downloaded program code, the host performs a reconstruction of the original program flow.

Within a group, all time stamps on instructions are equal, because there is no reliable way to reconstruct the time. The timestamp indicates the time at which the message, that reported the batch of sequential instructions as executed, was captured.

Other limitations are the bus trace where the on-chip trigger and filter provide just simple configurations, despite sometimes a very complicated configuration being needed. Typical for these configurations are a handful of address and data comparators only. Furthermore, much of the trace buffer can be wasted on large loops within a function when using the profiler tool on Nexus.

Software quality is free – innovative Real-time Trace Reconstruction

The Real-time Trace Reconstruction (RTR) was developed to circumvent the limitations of the Nexus and ETM on-chip trace. The logic and data required for reconstructing the Nexus or ETM flow back into address and control bus signals, is packed into a large FPGA (Field Programmable Gate Array). Output of this FPGA is then fed to the familiar powerful trigger and filter logic found on an ICE (see Fig. 2).

Full data reconstruction is not attempted, because an unfiltered data message stream would quickly overflow the Nexus FIFO in the CPU. For that purpose, the Nexus on-chip trigger and filter continues to be used.

Current Nexus and ETM implementations cannot fully replace an ICE, but using smart technologies like RTR, a power user can continue to use most of his favourite real-time analysis tools with Nexus or ETM on-chip trace.

The RTR FPGA reconstructs the CPU bus in real-time. And the complex trigger and filter engine can be used again.

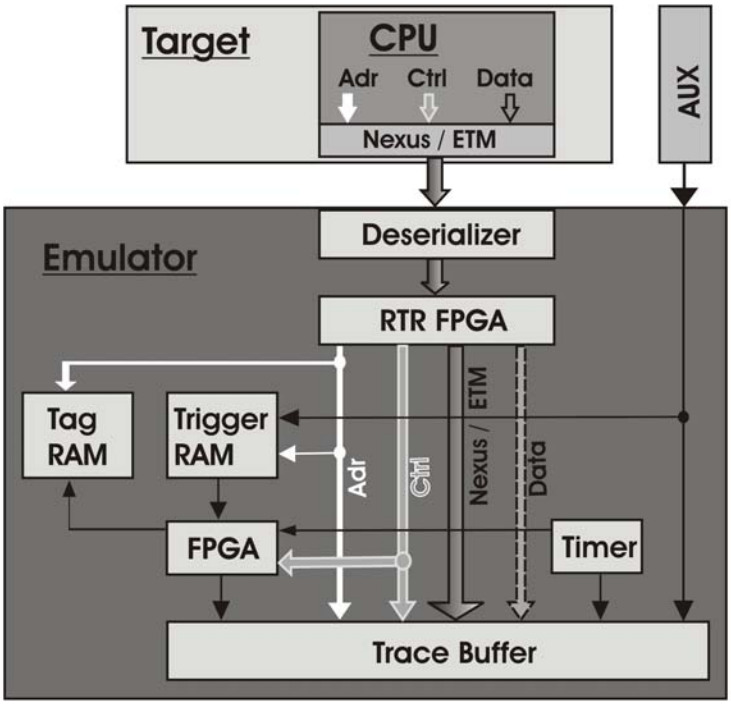

Figure 2: The RTR FPGA reconstructs the CPU bus in real-time

This will help to reduce the engineering and verification time for multi-user development projects, when using a newer generation of CPU. Moreover, it helps to use well-known analysis tools and the same powerful functionality when changing to more flexible hardware tools like on-chip solutions.

The table below (see Tab. 3) gives an overview of what kind of complex real-time analysis is available when using RTR with the on-chip Nexus or ETM port on modern CPUs; compared with a traditional ICE.

In another step the trace buffer is extended to a trace ring buffer. A special logic supervises the trace buffer, select enrolled data and transmit this immediately to the host. The trace buffer is used, so to speak, as a kind of "ring-FIFO". The user is enabled to run a very long recording of trace data according to the application. And by using the right mix of parameters together with the complex trigger and filter logic and, however, only the relevant data are transmitted for the analysis.

Some problems are solved:

1.) the user can use the complex trigger and filter logic and therefore only relevant data for the analysis
2.) the period of the recording can be extended by the right choice of the specific parameters
3.) the smart recording can be expanded up to several days by concurrent transmission to the host ("upload while sampling")

	ICE	**Nexus / ETM**	**RTR**
Trace Triggering	Complex	Basic On-Chip	Complex
Trace Filtering	Complex	Basic On-Chip	Complex
Trace Time Stamp	FULL	Distorted	Distorted
Execution Profiling	YES	Time limited	YES
Data Profiling	YES	Limited number of objects	Limited number of objects
Execution Coverage	YES	Time limited	YES
Data Coverage	YES	Time limited (*)	YES (*)

Table 3: Comparison of real-time analysis tools available on an ICE vs. Nexus vs. Real-time Trace Reconstruction. (*)means it's available on selected regions, if Nexus or ETM port bandwidth permits

By using this technology it is possible, for example, to record and transmit to the host approx. one million function calls per second. The recording length is restricted by fast connection to the host, the hard disk drive and CPU load and speed of the host.

With this solution the user can continue to use most of his favorite real-time analysis tools with on-chip trace. This will help to reduce the engineering and verification time for development projects. Also, the test and verification manager can run endurance tests and get an overview of the CPU internals. Moreover, it helps to use well-known analysis tools and the same powerful functionality when changing to more flexible hardware tools like on-chip solutions.

The RTR technology is available for the MPC55xx and all ARM/ETM families. Other processors with a Nexus or ETM interface will follow soon.

Literature

- NEXUS Forum: http://www.nexus5001.org
- NIST News Release: http://www.nist.gov/public_affairs/releases/n02-10.htm
- In-Circuit-Emulator/On-Chip-Debugger-Technology: http://www.isystem.com
- Trace – Getting Started, V8.02, iSYSTEM AG
- Profiler – User's Guide, V4.1, iSYSTEM AG