

Debugging Systems with FPGA Embedded Processors

F-Sight improves debugging efficiency through easy-to-use, state-of-the-art features.

by Jorge Carrillo
Staff Software Engineer
Xilinx, Inc.
jorge.carrillo@xilinx.com

Koji Imanishi
Department Manager
Computex Co., Ltd.
ima@computex.co.jp

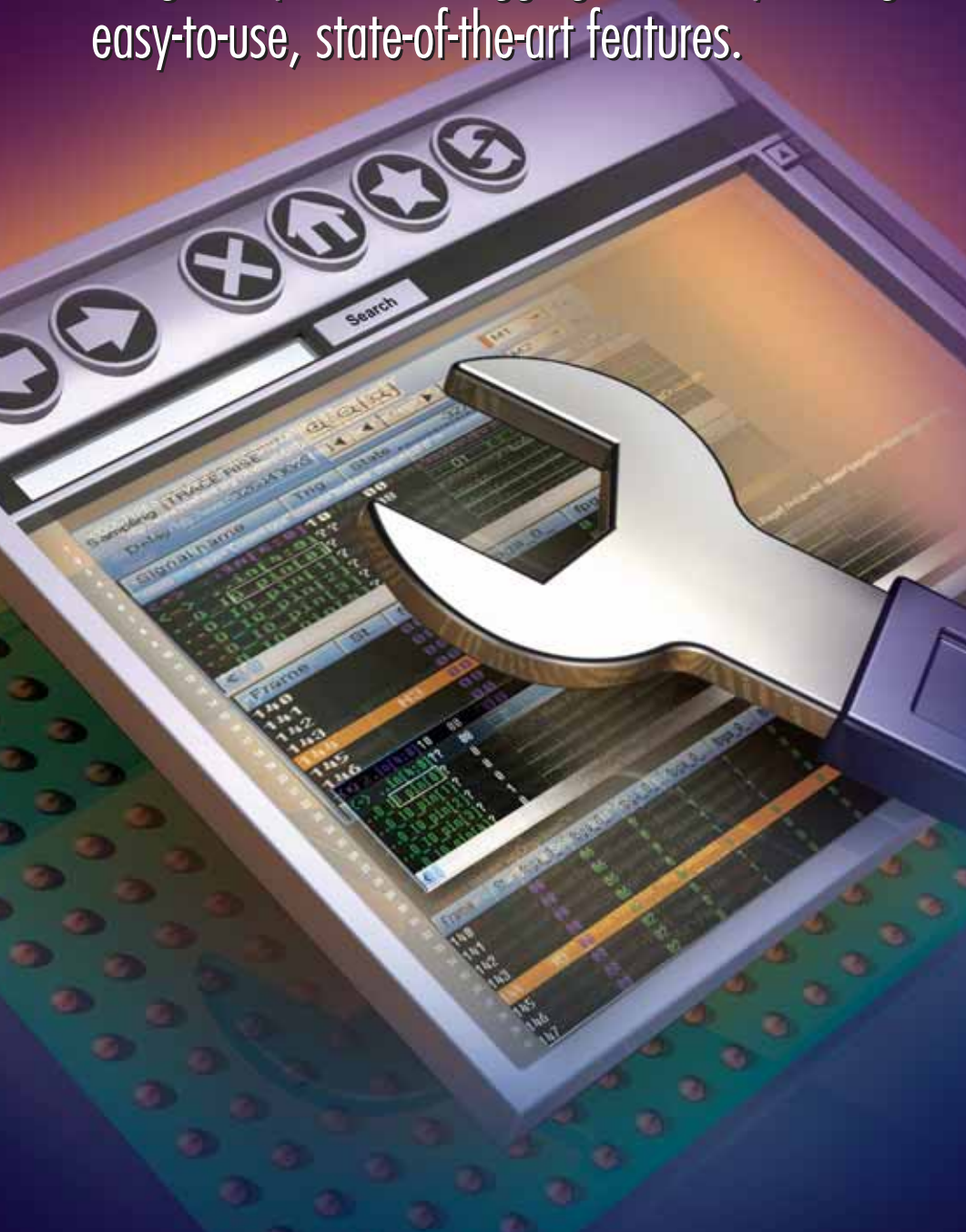
Raj Nagarajan
Senior Software Engineer
Xilinx, Inc.
raj.nagarajan@xilinx.com

Nobuhiro Nishiguchi
Superintending Engineer
Computex Co., Ltd.
nishi@computex.co.jp

Ayako Suzuki
Senior Engineer
Computex Co., Ltd.
suzuki@computex.co.jp

More and more FPGA designs now include embedded processors (like PowerPC and Xilinx® MicroBlaze™ processors) to deal with control tasks that are easier to describe in a software language like C rather than in a hardware description language like VHDL or Verilog.

When designing embedded systems, you probably spend the vast majority of design time in the debugging phase. It is important to reduce the time associated with finding problems and solving them. When you do face design problems, it is important to use the right tools for the job, including a tool that allows you to identify problems quickly.



Computex F-Sight is an integrated debugger that provides both hardware and software debugging capabilities. On one side, it allows full software debugging of processors inside the FPGA. On the other side, it allows monitoring of FPGA hardware signals. In this article, we will describe how F-Sight can help improve your debugging efficiency.

Enabling the Debugger

Xilinx has enabled the features available in the Computex debugger for use on FPGA internal embedded processors. For MicroBlaze processors, you can use the MicroBlaze debugging module (MDM) to control and debug the processor execution. You can also use the Xilinx MicroBlaze trace core (XMTC) to monitor the processor program execution in a non-intrusive manner.



Figure 1 – F-Sight connected to a Spartan-3 board

Because of pin limitations in FPGAs, it is important to reduce the amount of signals that are brought out to pins. The XMTC provides encoded instruction and data traces with pin requirements only 10% of what unencoded signals would require.

To enable the debugger to collect a trace, all you need to do is connect both MDM and XMTC cores to the MicroBlaze processor's debugging and trace interfaces, respectively, and bring out the encoded trace signals to FPGA pins so that F-Sight can collect the data. After FPGA implementation, connect the F-Sight debugger to your board's Mictor connector. If you are using a Xilinx-made

ML400 series, ML500 series, or Spartan™-3E/3A/3AN FPGA board that does not include a Mictor connector, you can still employ the processor trace feature in F-Sight by using the corresponding Computex F-Sight adapter. Figure 1 shows F-Sight connected to a Spartan-3 board using the F-Sight adapter.

Using Processor Trace

A processor trace monitors program execution without stopping the processor. This allows you to analyze program flow over long periods to identify problems in the code, all without changing the processor execution state. Computex F-Sight provides processor trace capabilities that can prove very useful in different situations.

Imagine a program that is constantly generating exceptions. Exceptions may

happen anywhere in the program; the challenge is to find out why and where the exceptions are being generated. For this, you can set a breakpoint either at the start of the exception being thrown or at the exception vector so that the program stops when it reaches the breakpoint. When the program stops, you can look at the execution history collected by F-Sight. It will show which instructions were executed before going into the exception handler.

Stack overflow is also a common problem in embedded systems. All of a sudden a program starts executing in a location that just does not look right. The stack is probably getting corrupted because of the

overflow. If you suspect this problem is occurring, you can set a trigger to start or stop trace data collection. By setting the trigger condition to allow a comparison between the stack pointer and the upper stack limit, the program will break right when the condition occurs. You can then easily identify the stack overflow and the location where it happened.

In certain cases with real-time systems, stopping a processor for debugging may not be an option, as stopping might change the program behavior. Other times the problem might occur very rarely, which might require you to monitor the program execution over a long time. Using F-Sight, you can set complex trigger conditions and collect trace data, which can be post-analyzed to debug the problem.

Probing Internal Signals

When debugging FPGAs, it is common to begin by simulating the design. The simulator is not capable of finding faults with the specifications, although it is able to find errors in the design. Also, it often happens that designs that pass all tests when simulated do not eventually work once implemented on the FPGA. When this happens, you are forced to debug the problem on the actual target system by using a logic analyzer.

The problem arises when you try to bring the signals out of the FPGA so that the logic analyzer can monitor their waveform patterns. As in the majority of cases when designing large-scale embedded systems, it will take a considerable amount of time to go through the FPGA synthesis and implementation tools even for minor changes (such as the changes needed to route signals out to pins). Plus, you risk running into timing problems caused by the different placement and routing. The actual time to run the implementation tools depends on the scale of circuitry and the host computer performance, but it's possible that you will only be able to debug a few times in a day.

Fortunately, Computex F-sight has a very useful feature that allows you to modify the design to bring internal FPGA signals out to pins without going through the synthesis and implementation tools. The



Figure 2 – F-Sight probing

feature is called “probing.” Simply select the internal FPGA signals in the display showing HDL source (Figure 2). F-Sight will automatically take care of the rest, adding the appropriate routing to the test pins on your behalf. It does this by utilizing the FPGA Editor included in Xilinx ISE™ software tools. With this feature, the time required for logic synthesis and place and route – time that you could have used for debugging – is minimized, allowing you to spend more time monitoring waveform patterns.

Cooperative Debugging

When the system is not functioning properly, the only thing you can do is investigate the cause of the problem based on actually occurring events. In some cases, event tracking is easier if you do it from the hardware; in other cases, it is easier if you do it from the software. For example, in the case of hardware, if the signal indicating abnormality is asserted you can set the signal as a trigger. In the case of software, if the exception handler is called out, you can set a breakpoint to the exception handling routine and run the user program. The process of event occurrence will be captured into the tracing buffer in F-Sight.

However, the issue here is that even if the history was captured, indentifying causes will take time unless the correlation between

the hardware and software is known. For this reason, Computex has implemented a cooperative debugging feature that allows you to synchronize the hardware (analyzer) and software (trace) histories in F-Sight. With this feature, you can check the wave pattern and program behavior at the time the event occurred on the same time axis. The program execution history and the source code view will follow as you scroll through the wave pattern display in the analyzer window (Figure 3). The cooperative debugging feature is powerful in that it quickly identifies causes by allowing cooperative debugging from both the hardware and software sides.



Figure 3 – F-Sight cooperative debugging

Debugging Flash Memory

The FPGA’s internal memory is commonly used to store the embedded processor program. However, this memory’s capacity is often not sufficient if the size of the program is too big. One alternative is to use external flash memory and store the user program in it.

Although some debuggers do not have the capability to write to flash memory, F-Sight allows you to fully debug programs that are located in flash memory just as you debug programs located in internal memory. For example, you can download the user program, apply patches to a part of the memory, or even set software breakpoints in the flash memory.

In F-Sight, you can select from among more than 1,000 available types of flash memory options. Even if the flash memory you are using is not available from the options in the list, you can manually and easily add the entry using the graphical user interface.

Conclusion

Debugging systems with FPGA embedded processors should not be a time-consuming task. When you encounter problems, you need to have the right tools that will allow you to efficiently deal with the problem, saving you time to focus on the development part. F-Sight certainly proves to be a tool that will improve your debugging efficiency. 🌈