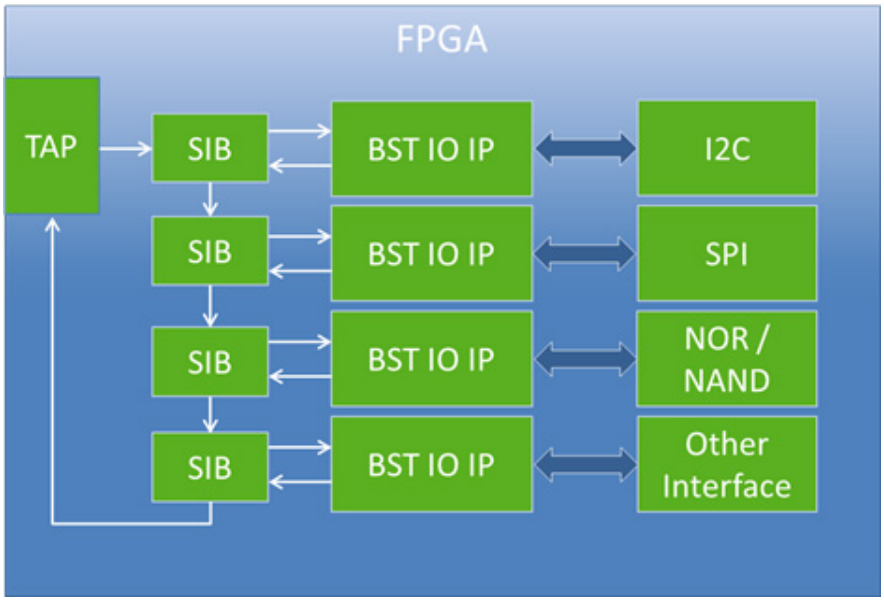
FASTER FLASH PROGRAMMING VIA FPGA AND IJTAG



EBOOK

BY KENT ZETTERBERG

By Kent Zetterberg – Product Manager

Kent Zetterberg started his career in the automation industry, working with systems from ABB, Siemens and others. Following graduation from the University of Gävle with a Bachelor's of Science Degree in Computer Engineering, he worked 15 years in the telecom industry where he held various positions involving hardware test and debug. He joined Ericsson AB in Sweden in 1997 where he developed functional test programs for processor boards, and designed interface boards and test fixtures. At Ericsson he became an expert in boundary scan and eventually led the boundary scan team. With ASSET Kent has held several positions in support, serving as a customer trainer and European support team leader. Currently he is the technical product manager for ScanWorks boundary-scan test products.

Table of Contents

Table of Figures

Executive Summary

When and how to program memories connected to field programmable gate arrays (FPGA) is a challenge for both system development and production engineers. Development engineers need to bring up circuit board prototypes, but often the system software, including the FPGA configuration files, may not be complete when prototypes are produced. That rules out pre-programming the FPGA-connected flash memories. And many board bring-up or production engineers either need to program FPGA flash memories with the most recent version of a software image that may still be changing or they never know which user-specific version of the software to load until a particular board configuration is ordered by a user. In these and other such cases, the most effective answer to the 'when-to-program' question would seem to be after the devices have been assembled onto a circuit board.

Unfortunately, the methods most frequently employed for in-system programming are relatively slow. The problem with slow programming speeds is exacerbated by the fact that FPGAs are growing in size, which means that the files stored in connected flash memory devices are likely also getting bigger every day. In addition, many FPGAs may be configured with embedded processor cores connected by various buses such as I2C or SPI to many different types of memory, such as NOR Flash, NAND Flash and possibly boot EEPROMs. As the amount of data that must be loaded into FPGA-connected memories increases, programming times will also increase unless newer high-speed methods are deployed. Moreover, production engineers are very cognizant of the manufacturing beat rate on the production line. Time is money and adding as little as a minute to the time it takes to produce a circuit board is a significant consideration.

Historically, most in-system programming methods have taken advantage of the JTAG port on a device connected to the memories. The connected device's JTAG port, as defined in the IEEE 1149.1 Boundary-Scan Standard, gives access to the memories, which lack a JTAG port, so they can be loaded with data. With the recent ratification of the IEEE 1687 Internal JTAG (IJTAG) standard, new more advanced IJTAG-based programming methods can be employed which still take advantage of an FPGA's JTAG port, but which also speed up the programming process by as much as a factor of 1,000. Depending on the context, of course, programming times can be reduced from 10 or more minutes to one or two seconds. This eBook explains how IJTAG methods involving FPGA-based intellectual property (IP) can accomplish this.

Background

During design, when prototypes of a new board design have been fabricated and they are being brought up to verify the functionality of the design, engineers often need to program memory devices like EEPROMs and flash memory in-system. The software may not have been completed when the prototypes were produced or the firmware for the EEPROMs or flash memory is still being modified. Removing and re-programming pre-programmed devices that were soldered down during assembly with an incomplete version of the software or one that subsequently changed could significantly lengthen the board bring-up phase, quite possibly delaying the design's transition into volume manufacturing. Moreover, removing soldered-down chips from a circuit board frequently damages the devices or the board.

In addition, during volume production of certain board designs, the on-board flash memories are not loaded with software until a specific version of the software is selected by the end user just prior to shipping the system. In these and other scenarios, the ability to program flash and other types of memories in-system increases the designer's and the manufacturing engineer's flexibility and increases the efficiency of the entire design and manufacturing process. (Another eBook in ASSET's eResources center, titled "At-Speed SPI Flash/EEPROM Programming using FPGA and JTAG", discusses several alternatives for programming flash memory connected to an FPGA via the SPI bus.)

JTAG Register Access

It is not uncommon to program memory devices connected to an FPGA via the FPGA's JTAG port and its boundary-scan registers. Unfortunately, this method for accessing memory is typically very slow because every write or read transaction on data, address and control pins must be scanned in through the FPGA's entire boundary scan chain, which can be anywhere from approximately 500 to 3,000 cells wide. Since FPGAs are completely programmable, they can be configured to include fast memory programming IP or even at-speed programming engines. The former method essentially replaces the long boundary-scan register with a shorter register for programming purposes and the latter removes the need for scanning through any boundary-scan register. At-speed programming engines typically have a FIFO buffer design with direct access to the FPGA pins. When these types of programming IP are embedded in the

FPGA, flash programming times can be reduced quite significantly. This reduction is approximately proportionate to the reduction in the length of the scan register or the number of register cells in the scan chain. For example, if the cell count can be reduced from 1,000 to 100 cells to program a NOR flash device, the programming time can be reduced in the range of five to 10 times. Similarly, if a flash device were connected to an FPGA over the SPI bus, reducing the number of register cells from 1,000 to five would reduce programming time by a factor of 100 to 200. With the at-speed programming engine, the time savings can be even greater. These methods have been deployed by several system suppliers over the last few years.

Multiple Interfaces Accessed through IJTAG

The scenario becomes somewhat more complicated when the FPGA is connected to several different flash interfaces, such as NOR or NAND flash, or other types of memories with interfaces to the SPI or I2C buses. In these cases, an IJTAG-based programming method can be very beneficial and accommodate such diversity.

Typically an FPGA supports several instructions that can be used to implement user-defined instructions for accessing register architectures as well as instruments. These secondary instructions and register constructs might implement short scan chains, but, unfortunately, most FPGAs typically only support relatively few of these user-accessible instructions. This limits the number of secondary instruments/registers that can be defined (Figure 1).

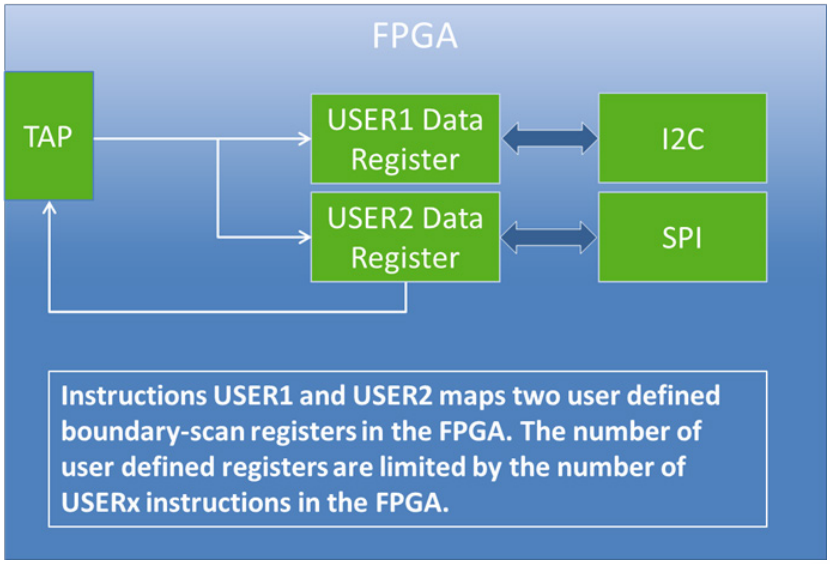


Figure 1: FPGA with a user instruction implemented programming interface

The traditional IEEE 1149.1 Boundary-Scan Standard (JTAG) pairing of instruction register/data register is also limiting, insofar as only one register can be communicated with at a time.

On the other hand, the new IJTAG standard is able to implement a practically unlimited number of instruments/registers within the FPGA supported by only one of the device's virtual user instructions. This capability is enabled by IJTAG's Segment Insertion Bit (SIB), which acts as a register selection mux. When loaded with the right value, a SIB opens a path into an instrument or multiple instruments. Since an FPGA is programmable, one SIB can be inserted for each programming instrument or engine that has been tailored to a specific programming interface (Figure 2). This kind of architecture affords access to each instrument individually and the overhead associated with each SIB is only one bit, considerably less than the hundreds of bits if the full boundary-scan chain must be traversed to program the connected memory device.

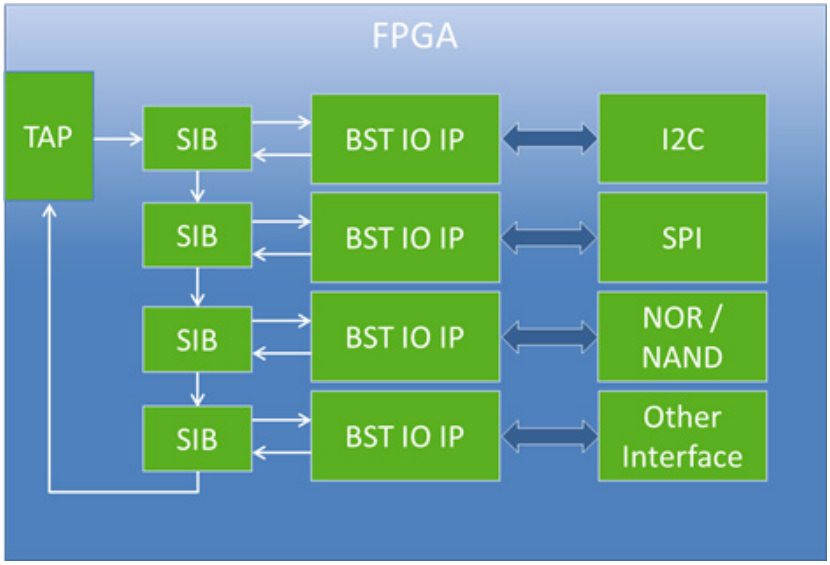


Figure 2: IJTAG Network with multiple programming interfaces

It should be noted that the IJTAG standard is not limited to the deployment of programming engines or programing embedded instruments. A FPGA might also be populated with functional IJTAG instruments such as frequency counters, UART, SPI and I2C masters, as well as functional embedded instrument IP for memory test and logic BIST. An IJTAG network of embedded instruments could include programming engines as well as all sorts of functional test IP, all of which could be accessible via IJTAG (Figure 3). Of course, deploying a large number of individual instruments each with its own SIB can result in a large number of SIBs in the scan

chain. Care should be taken during the design of the IJTAG network to ensure that the IJTAG network structure will be efficient in terms of its speed and responsiveness.

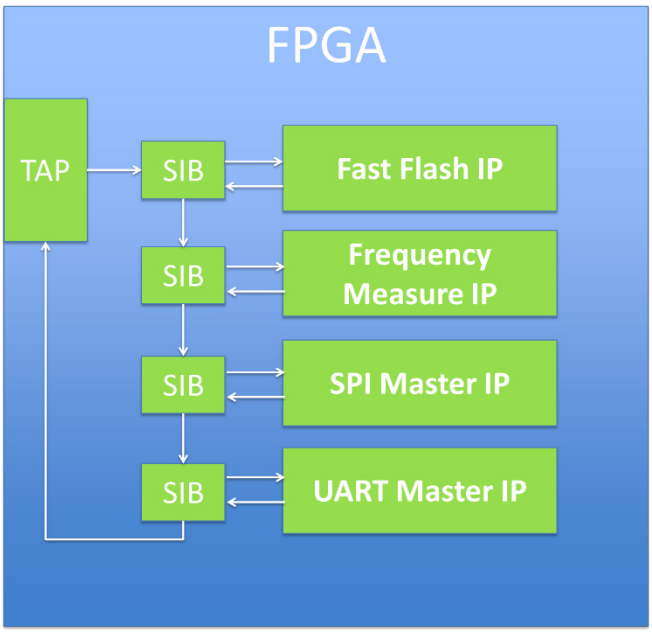


Figure 3: IJTAG network with functional test and programming IP

Today's medium to large FPGAs can have anywhere from several hundred to several thousand boundary-scan cells. For example, a Xilinx Virtex 7 has more than 3,000 cells while an Altera Stratix 5 includes just under 3,000 cells. IJTAG-based access methods can effectively shorten these scan chains down to three or four cells. Considering the use case of programming SPI or I2C memory, the potential time savings are enormous and the theoretic maximum programming rate of the connected flash device can be reached even when the device is already mounted on a circuit board.

Expected Throughput Benefits

The following are several examples of the possible reductions in programming times that can be expected when FPGA-based IJTAG programming engines are deployed. Of course, actual programming times will depend on the specific FPGA device as well as the write times for the memory devices being programmed. The examples below demonstrate the potential throughput acceleration that can be gained by shortening the scan chain through IJTAG access.

NOR Flash

- Between 2 to 50 times faster throughput
- 10 minutes down to 12 seconds

SPI Flash

- 50 - 1,000 times faster throughput
- 10 minutes down to one second

I2C Flash

- 50 - 1,000 times faster throughput
- 10 minutes down to one second

NAND Flash

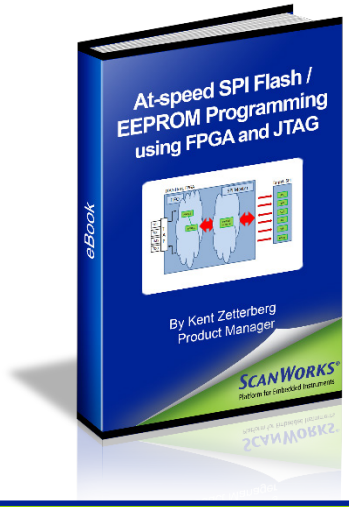
- 4 - 80 times faster throughput
- 10 minutess down to eight seconds

Conclusions

The IEEE 1149.1 Boundary-Scan/JTAG Standard has been around for more than 20 years and it still provides huge value in test, programming and other areas. And now, the newly approved IEEE 1687 IJTAG Standard not only builds on the value inherent in JTAG, but adds to it significantly, bringing a new generation of benefits and value to many of the established JTAG-based applications and also opening up new applications to the greater value of embedded instrumentation. In this instance, IJTAG methods extend the value that JTAG has delivered for years to in-system programming applications by significantly accelerating and streamlining the in-system programming process to the point where development engineers are able to speed their new designs to market faster and production engineers can now incorporate in-system programming into the assembly line without jeopardizing the manufacturing beat rate. Tools like the ScanWorks® platform for embedded instruments are able to add to the established benefits and value of JTAG with the new and more powerful capabilities of IJTAG.

Learn More

For more information on how a functional FPGA can form the basis for high-speed in-system programming, click here.



Register Today!