The Java API for Boundary-Scan: Enabling Technology for Internet–Driven PLD Systems

Background

When first introduced, programmable logic devices were used primarily to facilitate rapid prototyping and debug of systems under development. As the price of programmable devices fell, their use increased within higher volume manufactured systems. At the same time, designers soon began demanding access to the programmable capability of the devices in order to facilitate remote debug, test and field upgrade. The access to this reconfigurable capability has been enhanced by near total vendor standardization on IEEE Std 1149.1 (a.k.a. JTAG or boundary-scan) as the protocol for access to PLD programmability. By basing in system programming capabilities on the extant IEEE Std 1149.1 test protocols, better product life cycle support of reconfigurable systems was immediately achievable by allowing total system access through an industry standard protocol for programming, test and debug.

One final piece of the puzzle remained unavailable. This piece is the enabling technology that allows access to the device IEEE Std 1149.1 interface for programming test and debug across the wide variety of platforms required throughout the product lifecycle in a single reusable description language.

IEEE Std 1149.1

The Boundary Scan/JTAG standard, formally known as IEEE/ANSI Std 1149.1, is a set of design rules, which facilitate the testing, device programming and debug at the chip, board and systems level. The standard came about as a result of the efforts of a Joint Test Action Group (JTAG) formed by several North American and European companies. IEEE Std 1149.1 was originally developed as an on-chip test infrastructure capable of extending the lifetime of available automatic test equipment (ATE). This methodology of incorporating design-for-test allows complete control and access to the boundary pins of a device without the need for a bed-of -nails or other expensive test equipment. Each JTAG compliant device includes a boundary-scan cell on each input, output or bi-directional device pin (Figure 1) that under normal conditions is transparent and inactive allowing signals to pass normally. When the device is placed in test mode, input signals can be captured for later analysis and output signals can be set to affect other devices on the board.

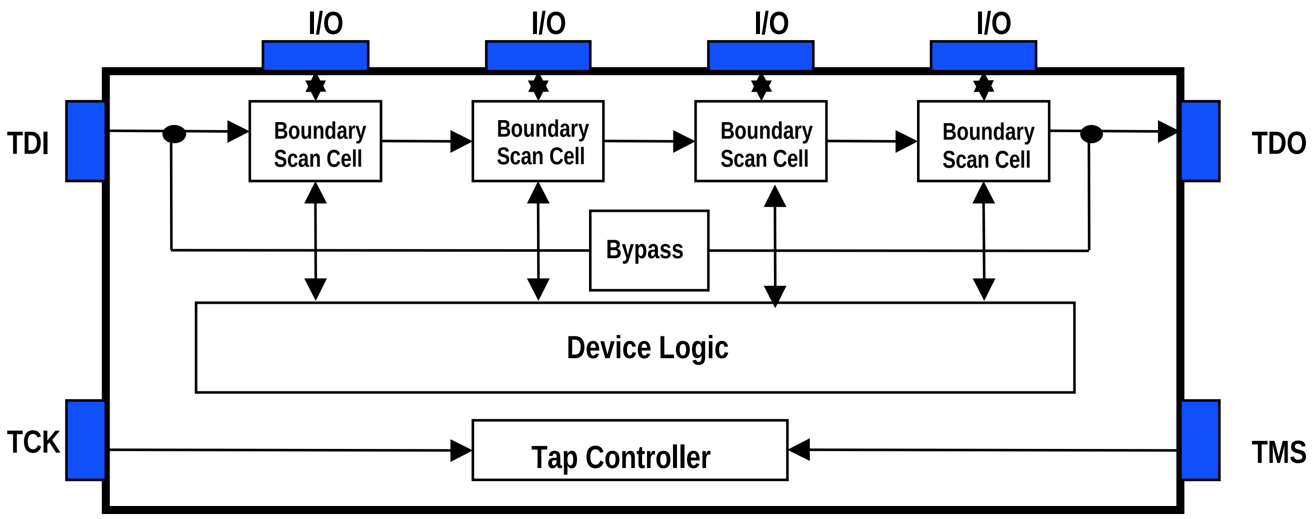

Figure 1

Access is required to only 4 (or optionally 5) pins on a device, regardless of the packaging constraints. These pins define a Test Access Port (TAP) that enables operation of the on-chip test infrastructure that can be used to ensure that:

A. All components on a printed circuit board are mounted properly and in the right place.
B. All interconnections between components are as described in the design.

So simply stated, the IEEE Std 1449.1 defines a serial protocol requiring at least 4 pins on each compliant device. These pins are as follows:

A. TCK - This is a clock signal that synchronizes the 1149.1 internal state machine operations.
B. TMS - This is the 1149.1 internal state machine mode select signal. This signal is sampled at the rising edge of TCK to determine the next state machine state.
C. TDI - This is the 1149.1 data input pin. When the internal state machine is in the correct state, this signal is sampled at the rising edge of TCK and shifted into the device's test or programming logic.
D. TDO - This is the 1149.1 data output pin. When the internal state machine is in the correct state, this signal represents the data shifted out of the device's test or programming logic. The output data is valid on the falling edge of TCK.
E. TRST (optional) - This is the 1149.1 asynchronous reset pin. When driven low, the internal state machine advances immediately to the reset state. Since the pin is optional and pins are generally high cost additions to devices, it is infrequently used. In addition, the internal state machine (as defined by the standard) has a well-defined synchronous reset mechanism.

The pins of the TAP drive a 16-state controller (state machine). The state machine transitions between states according to the value of the TMS signal value on the rising edge of TCK. The state machine is illustrated in the diagram below (Figure 2).

The '0' and '1' along the transition arcs represents the state of the TMS signal at the rising edge of TCK.

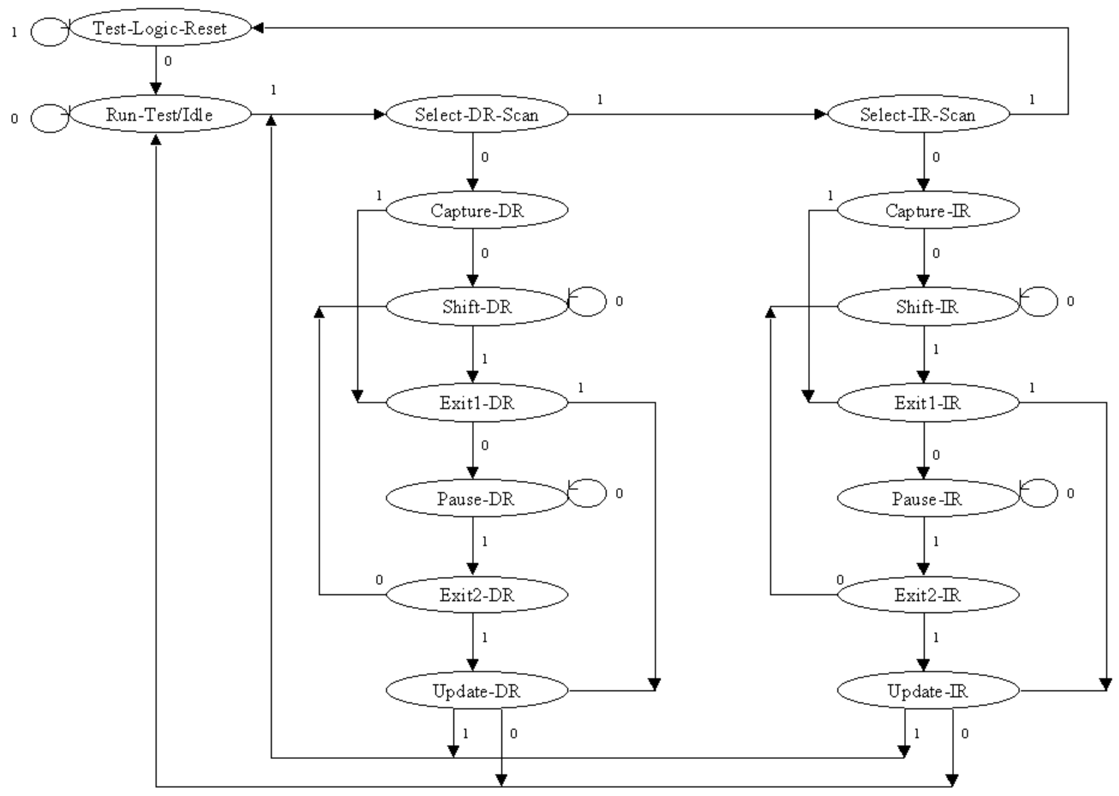

Figure 2

The 1149.1 standard defines that TDI is valid and shifted in (and TDO valid and shifted out) only in the Shift-DR or Shift-IR states. The Shift-IR state selects the device instruction register between TDI and TDO. Depending on the instruction selected different data registers are activated. When in Shift-DR, the data register appropriate for the previously entered instruction is selected between TDI and TDO. The default data register is the mandatory 1-bit bypass register.

An external file known as the BSDL (Boundary-Scan Description Language) file defines the properties and characteristics of any single device's boundary-scan logic. These files are supplied by the IC manufacturer and are used in the generation of any algorithmic description of the operation of the IEEE 1149.1 compliant device.

Multiple boundary-scan devices are linked up serially in a daisy chain. Each device shares the same TCK and TMS. The TDO of one device links to the TDI of the next. Since all devices share the same TCK and TMS, all devices sequence through the TAP controller synchronously and concurrently. All devices are therefore in the same TAP controller state at the same time. When shifting data (in the Shift-IR or Shift-DR state) into the boundary-scan chain, all devices have registers internally linked between their TDI and TDO pins. This results in what appears to be a single shift register of a fixed length from the system TDI pin to the system TDO.

Programmable Logic Systems

The development of any system entails transition through several phases (Figure 3). The first phase is the prototyping and debug phase. This phase is typified by rapid iterations of the intended system design as the designer develops and debugs his system. The PLD's are downloaded many times as the system is being debugged. The download is accomplished by connecting a special purpose cable to a workstation or a PC. The designer might use a logic analyzer or an oscilloscope to debug his system.

The next phase is the pre-production phase. This is typified by building up a small number of prototypes of the final system. These prototypes are then run through programming and test systems to determine and improve the testability and manufacturability of the system. Some special purpose hardware or software might be used in this phase that is hosted on a PC or workstation. It is also possible that Manufacturing Defects Analyzers (MDA's) are used at this point.

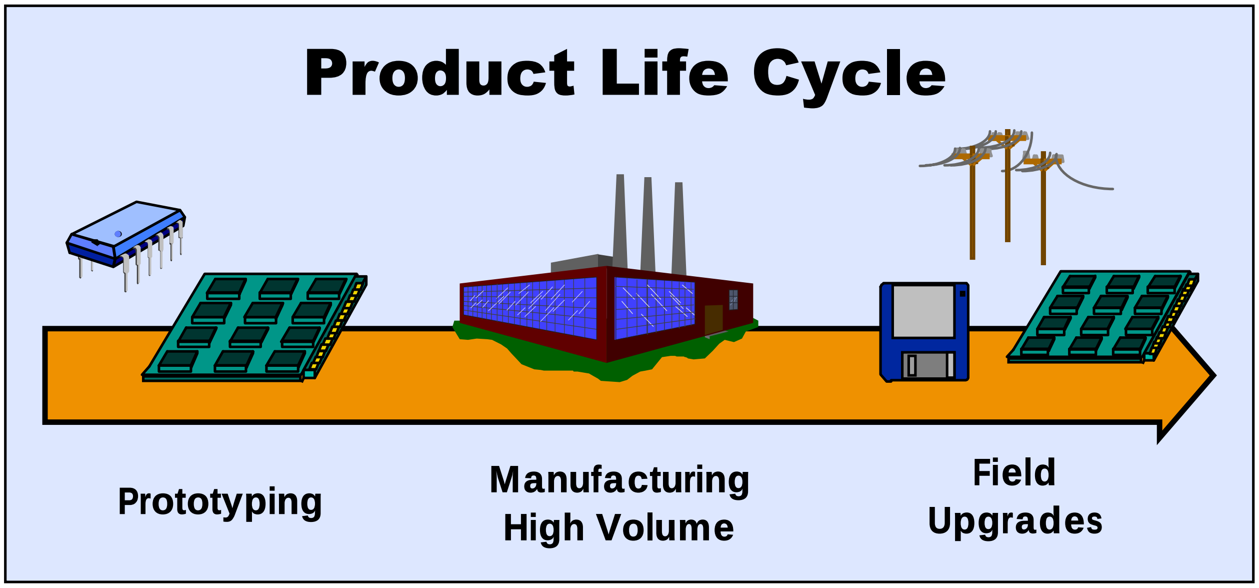

Figure 3

Depending on the expected shipment volumes, higher speed test and programming equipment may be used in full-production. This would include deployment of automatic test equipment (ATE), programming hardware or customer-developed programming and test systems.

Once the systems are out in the field, the designer is interested in having easy access to his system for remote test debug and potentially system upgrade capability. This would typically be accomplished through use of an embedded microprocessor within the target system. Access to the processor would typically be through a remote system link or by using debug routines in local storage.

Ideally, to reduce the time and effort required for multiple deployments, the algorithmic description of the programming and test of the target system should be platform independent. The same algorithm description file used on a PC or workstation should also work on an ATE or embedded processor.

The Java Programming Language

The Java programming language was developed by Sun Microsystems as an object oriented, easy-to-use application development language. It was designed to allow large amounts of code reuse as well as total portability across a myriad of platforms. This high level of portability was achieved by having the language compile down into a processor non-specific generic assembler code. This processor non-specific generic assembler code was then interpreted by a software processor (called a Java Virtual Machine (JVM)) on the target platform. Portability is therefore achieved by creating JVM's for all target platforms (Figure 4)

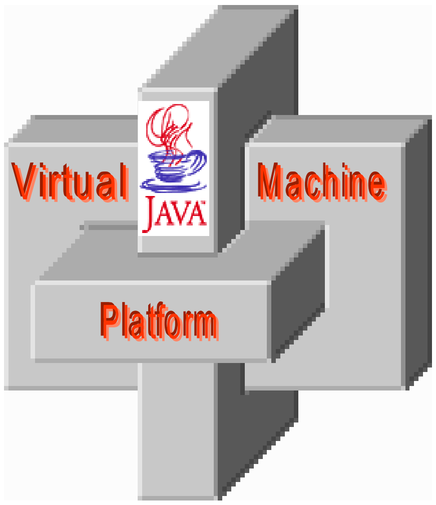

Figure 4

The Java Programming language found its immediate application in internet-based solutions in which programs would need to be run on a wide variety of platforms, the nature of which would not be known until deployment through a browser or other launching tool. This means that a single Java program would be able to be written for deployment on any possible platform for which a JVM exists and that it would behave identically on all platforms.

Further, the Java programming language was enriched through the development and deployment of a wide variety of system class libraries that afford reusable, easily integrated capabilities for such high utility operations as data manipulation, security protocols, multi-threading, exception handling and internet-based file access.

There are over 180 licensees of Java technology worldwide. These licensees provide a wide variety of software development tools including compilers, debuggers and other CASE tools as well as JVM implementations, reference materials and workshops.

The Applications Programming Interface (API)

In order to facilitate rapid code re-use, it is possible within the Java object oriented paradigm to declare Applications Programming Interfaces (API). API's are use to define often re-used methods to facilitate rapid development of applications. Typically, API's are used in order to provide commonly re-used building blocks to applications developers.

One such natural application is the communications protocol defined by IEEE Std 1149.1. It turns out that this protocol naturally defines its own API since the operations it allows are well defined. By defining these protocols within object oriented methods, either an automated tool or an applications developer can rapidly generate a programming or test application for an entire system of PLD's or general boundary-scan devices.

Java API for Boundary-Scan

What is apparent is that the write once, run anywhere properties of Java are immediately applicable to the PLD design solution space. Since all PLD's already support the IEEE Std 1149.1 protocol, a publicly available API immediately makes available the Java solution and its related tools and technologies for PLD systems designers across all phases of the product life cycle (Figure 5).

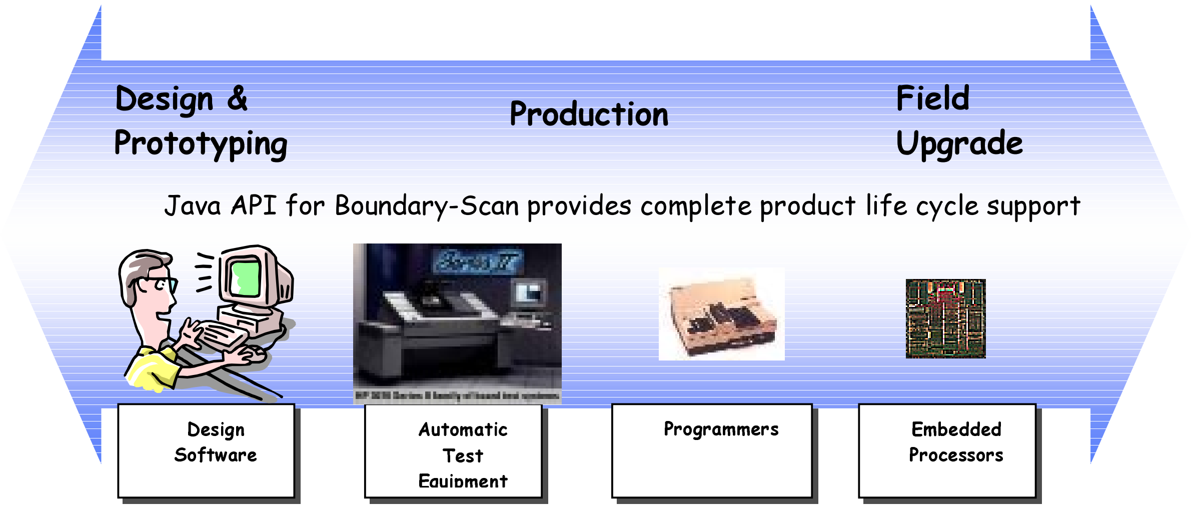

Figure 5

The Java API for Boundary-Scan is a publicly available API that describes IEEE Std 1149.1 protocols. It is used to describe all functionality required to develop programming, test and debug algorithms for electrical systems.

Applications developed using the Java API for Boundary-Scan can also make use of the other general purpose Java class libraries to enable Internet based update of PLD-based systems, enhance system security, integrate multi-threading of configuration and test applications and interface to remote configuration databases.

Conclusions

It is natural and logical to use an available, well-supported and proven software technology like Java to provide a solution to facilitate Internet enabled reconfiguration and debug of PLD-based systems. By leveraging on a wide and deep installed base as well as a wellspring of development knowledge and experience, we are able to provide a turnkey solution to exploitation of the reconfigurable aspects of PLD technology.