Low Pin-count Debug Interfaces for Multi-device Systems

Michael Williams*
ARM Limited, 110 Fulbourn Road, Cambridge, England.
*michael.williams@arm.com

***Abstract*-IEEE Std 1149.1-2001 *Standard Test Access Port and Boundary-Scan Architecture* (JTAG) is widely used as a debug interface, providing a path for a debugger to access debug components in complex systems-on-chip (SoCs). By its very nature JTAG accommodates systems containing multiple devices. However, JTAG was primarily intended as a component and board test interface, and is not ideally suited as a debug interface. Its shortcomings have led the industry to search for an alternative. As a result, JTAG interfaces have started to be displaced by dedicated debug interfaces. This paper examines some of these alternatives, and concludes that a dedicated serial wire debug interface can be delivered with lower pin-count and higher performance, whilst maintaining support for multi-device systems and interoperability with test.**

I. Introduction

JTAG [1] was originally designed and intended as a test interface. The four- or five-pin interface comprises two unidirectional data pins (**TDI** and **TDO**), a clock (**TCK**), state machine control pin (**TMS**) and optional reset (**TRST#**). This use of unidirectional data pins at either end of a scan-chain connected to the registers being accessed, coupled with a simple control state-machine allows daisy-chaining of multiple JTAG devices (Fig. 1).

That JTAG is primarily meant for test is reflected in the specification; for example, the specification requires that each device implements a single test access port (TAP), and describes in detail control instructions (INTEST, EXTEST, SAMPLE, PRELOAD) and scan-chain structures for implementing boundary-scan testing.

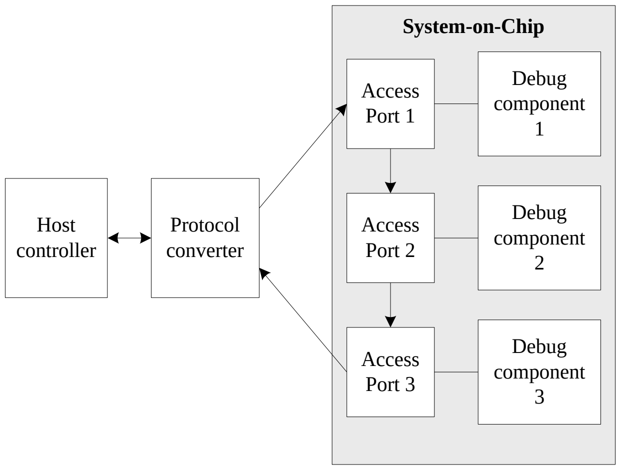


Fig. 1: Daisy-chain debug topology

JTAG is also widely used as an interface for controlling the embedded debug features of processors and SoCs. The simplicity and ubiquity of the interface, and its support for connecting multiple devices through daisy-chaining made it an ideal candidate for the first generations of embedded debug.

However, it has several shortcomings, for example:

- JTAG does not make efficient use of the four pins dedicated to it, for example an efficient direct memory access TAP may only achieve a data rate of 640Kbytes/sec per data pin at 20MHz;
- using JTAG to access multiple debug components on a single SoC (as in Fig. 1) is not strictly allowed by the standard;
- daisy-chaining is intolerant to a debug component and its TAP being removed from the system, for example as a result of power management.

Recent developments in debug interface technology have attempted to address some of these concerns. The primary focus has been on reducing pin-count.

II. Low Pin-count Interfaces

An alternative to JTAG for debug should fulfil the following requirements:

- maximum of two pins: vital for very low connectivity devices or packages;
- support for multiple devices connected simultaneously;
- inter-operability with other debug and test interfaces;
- allow debug through legacy JTAG TAP controllers;
- high performance data rates;
- synthesis-friendly with high maximum clock rate;
- low power;
- small silicon area;
- low tools costs;
- reliable in the face of errors and safe from glitches on pins when tools not connected.

This paper outlines three approaches to a low pin-count interface, describing the third, preferred approach in more detail:

- time-division multiplexed JTAG interface;
- re-visiting the JTAG standard (IEEE 1149.7);
- dedicated serial wire debug interface.

A. *Multiplexed JTAG Interface*

One approach to reducing the pin-count for the debug interface is to multiplex the JTAG interface in the time domain.

The JTAG interface has three functional pins (**TDI**, **TDO** and **TMS**) each of which carries a data value that can be valid on every clock cycle.

Dividing the clock by three enables each of these data values can be presented in turn on a shared data pin, thereby reducing the pin count from four pins to only two.

This scheme has several obvious disadvantages:

— it reduces the bandwidth by a factor of three, reducing debug performance;
— does not allow daisy-chaining of devices, making debug of multi-device systems very difficult;
— it requires turnaround of the data pin from host to target drive in a single data cycle, limiting the maximum operating frequency and further impacting performance.

The scheme maintains compatibility with JTAG test, but the increased tester time is probably prohibitive.

B. *Re-visiting the JTAG Standard, IEEE 1149.7*

The IEEE 1149.7 *Draft Standard for Reduced-pin and Enhanced-functionality Test Access Port and Boundary Scan Architecture* [2] extends the 1149.1 standard with support for:

— multiple power modes;
— system level bypass;
— star topology;
— two-pin operation.

IEEE 1149.7 describes six classes of compliance (Fig. 2). These classes are hierarchical, meaning that a device wishing to benefit from the T4 two-pin operation must also implement all of T0-T3, even though the device is targeted at debug and not test.

This paper only provides a brief overview of IEEE 1149.7; for a more detailed introduction, see [3].

Star topology

A star topology (Fig. 3) supports true power and clock isolation; the access port or ports can be isolated from the power and clocks of the components being debugged, continuing to function even when power is removed from the component.

It also provides higher performance than daisy-chain topology, as there is no need for un-addressed components to be in a "pass through" bypass state.

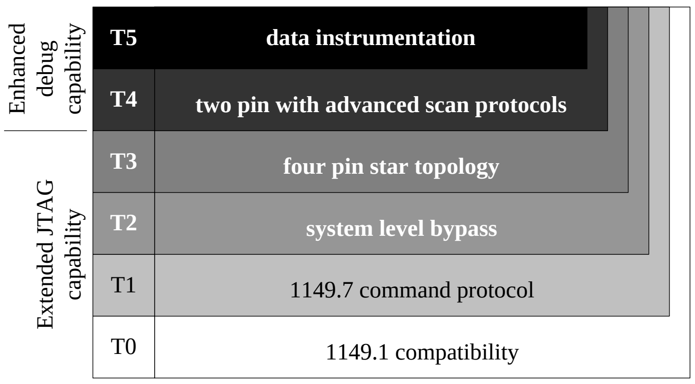

Enhanced debug capability
Extended JTAG capability
T5 data instrumentation
T4 two pin with advanced scan protocols
T3 four pin star topology
T2 system level bypass
T1 1149.7 command protocol
T0 1149.1 compatibility

Fig. 2: IEEE 1149.7 compliance classes

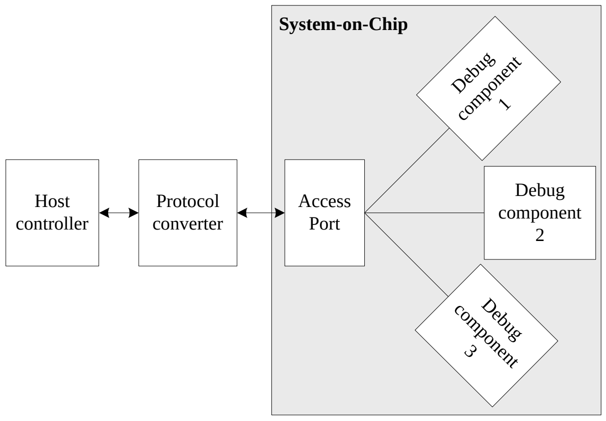


Fig. 3: Debug star topology

IEEE 1149.7 provides commands to select a component to address in star topology.

Command protocol

IEEE 1149.7 operation is controlled by command sequences sent over the JTAG interface. Because of the need for backwards compatibility with IEEE 1149.1, these sequences consist of:

— special paths through the JTAG TAP state machine that are benign if sent to an IEEE 1149.1 implementation that has just been reset;
— out-of-band messaging using clock as data and data as clock: by holding the clock signal **TCK** HIGH and toggling the **TMS** signal, control messages can be sent out of band of the data stream.

Out-of-band messages are used in two-pin operation to change data multiplexing mode. This effectively embeds the clock in the data signal, which is problematic for standard synthesis flows. They are also used to provide *online* and *offline* state selection.

IEEE 1149.7 also supports online and offline state selection using the *unlikely data sequence* approach described herein in *Selecting a device from dormant mode*.

Data multiplexing

In two-pin operation, IEEE 1149.7 uses only the **TCK** and **TMS** pins. In this mode of operation, the pins are named **TCKC** and **TMSC**, respectively. **TDI** and **TDO** data can be multiplexed onto **TMSC** in a manner similar to that described herein in *Multiplexed JTAG Interface*.

Thus **TMSC** is a bidirectional data pin. To support this, IEEE 1149.7 specifies that **TMSC** is not driven for the second phase of each clock cycle, and must either be sampled on the rising edge of **TCKC**, or held by bus keepers to be sampled on the falling edge.

This phase is used for turnaround on the bidirectional pin. This need to establish a stable signal level in a half a clock cycle impacts the maximum operating frequency of the interface. (Fig. 4.)

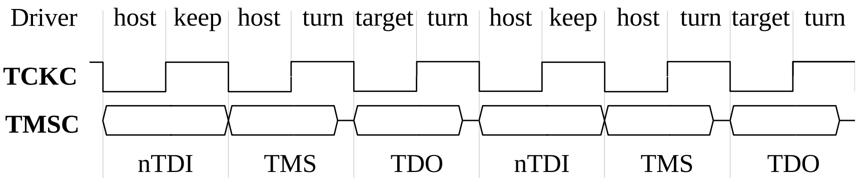


Fig. 4: IEEE 1149.7 T4 two-pin protocol, TAP.7 Advanced Protocol Optimized Scan Format 1 (OScan1)

However, a more detailed analysis of the way JTAG is used, particularly for debug, allows optimizations to be made:

— when not in or entering a *Shift* state, the values on **TDI** and **TDO** are irrelevant and can be omitted;
— when in a *Shift* state, **TMS** is held HIGH until ready to exit the *Shift* state; hence **TMS** may be omitted;
— for many scans, only **TDI** or only **TDO** is relevant for part or most of the scan; hence either may be omitted.

To accommodate these optimizations, IEEE 1149.7 level T4 describes various *optimized scan formats* (OScan) and *segmented scan formats* (SScan), use of which can improve scan performance by a factor of at least two. The standard does not require implementation of all scan formats.

C. *Dedicated Serial Wire Debug Interface*

To address the issues with JTAG, ARM Ltd. (Cambridge, England) took a fundamentally different approach. A packet-based protocol, *Serial Wire Debug* (SWD) was developed [4], [5].

SWD replaces the four-pin JTAG debug port with a clock plus a single bidirectional data pin, providing a dedicated packet-based debug interface that encompasses all the normal JTAG debug functionality.

This interface is now considered in more detail.

III. Serial Wire Debug in Detail

During the development of the *ARM® CoreSight™ debug architecture* [6], [7], the opportunity was taken to analyze the areas in which the requirements for debug and trace differed from those for test, and to design a debug interface protocol appropriately. The result of this detailed analysis is the SWD protocol.

Although it was developed as part of the CoreSight debug architecture, SWD is defined as a standard independent of CoreSight. Two versions of the protocol are defined: SWD protocol version 1 and SWD protocol version 2.

To understand how the SWD protocol was developed, it is necessary to first provide an overview of the CoreSight debug architecture.

A. *CoreSight debug architecture overview*

The most significant change introduced with CoreSight and SWD was a move from using a serial scan interface to using a bus-based approach for on-chip debug control and access.

This modular approach allows for SoCs that consist of multiple IP blocks from multiple vendors often using multiple clock and power domains. Each block provides register-based access to debug configuration and status information: this provides a more consistent programmers' model and eases software development. These same registers can also be accessed by the CPU itself, giving additional flexibility.

The star topology of a bus (Fig. 3) also supports true power and clock isolation, which is important in all application areas, in particular mobile, as the market drives devices to reduce standby power consumption. It is possible to power-down individual blocks of logic or control the clocks independently, including when debugging.

ADIv5

Previous incarnations of JTAG debug ports on ARM processors were described as the *ARM Debug Interface*. For instance, the JTAG port on ARM11™ processors is known as ADIv4.

Decoupling the external debug interface from the internal bus-based debug and trace infrastructure allows the interface to be described independently.

The resulting *ARM Debug Interface v5* (ADIv5) [4] allows tools compatibility across a range of processors without requiring a full implementation of the CoreSight debug architecture.

Debug Access Port

The external debug interface (for example, SWD) is decoupled from the internal bus-based debug and trace infrastructure by a *Debug Access Port* (DAP).

Fig. 5 shows the basic structure of an ADIv5 DAP. The external debug interface to the SoC can be any communications interface, such as JTAG or SWD.

The external debug interface connects to the DAP through a *Debug Port* (DP), such as an SW-DP for the SWD interface.

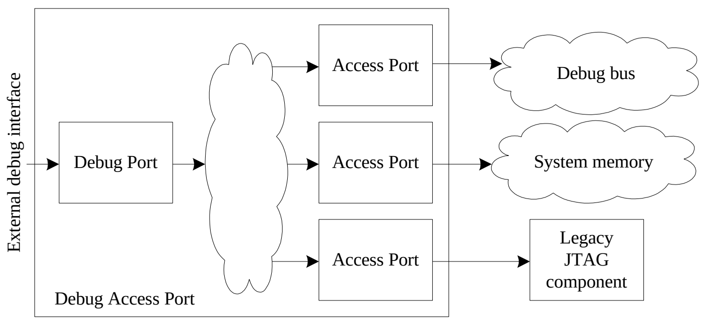


Fig. 5: Debug Access Port

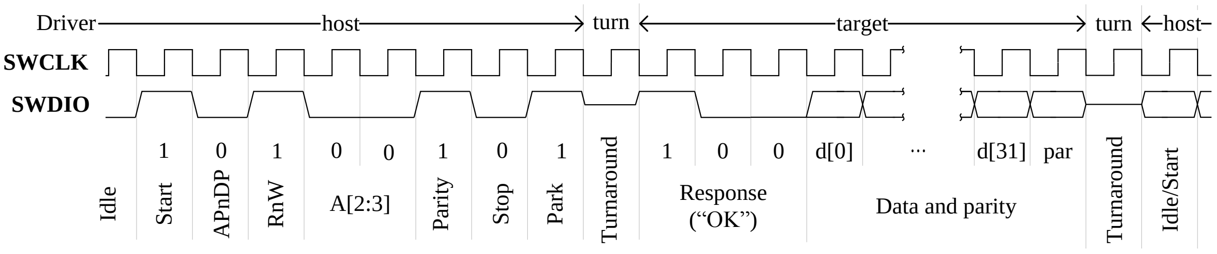


Fig. 6: Serial Wire Debug protocol, read of the ID register

The DAP further comprises *Access Ports* (APs) which can access various slave devices, for example:

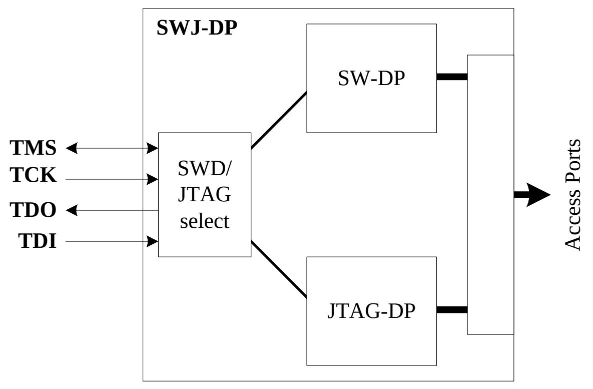
- — legacy JTAG-equipped cores via a *JTAG Access Port* (JTAG-AP);
- — system memory via a *Memory Access Port* (MEM-AP), such as the *AMBA Advanced High-performance Bus Access Port* (AHB-AP) or *AMBA AXI Access Port* (AXI-AP);
- — bus-based debug functionality on a debug bus (including ARM Cortex processors) via a MEM-AP, such as the *AMBA 3 Advanced Peripheral Bus Access Port* (APB-AP);
- — dedicated debug or other control devices, such as a *rights manager* for security-based applications, or a *built-in self-test (BIST) controller* for packet-based test.

B. Serial Wire Debug protocol version 1

Decoupling the external debug interface from the internal bus-based debug and trace infrastructure also allowed the CoreSight developers to optimize the physical protocol to match the external interface. As a result, a packet-based protocol was developed.

Like IEEE 1149.7 SWD protocol uses a bidirectional data pin; however, unlike IEEE 1149.7, the host-to-target and target-to-host phases are kept quite separate in the protocol, allowing for fewer turnarounds on this pin.

Thus the wire protocol passes data between the target system and the debugger in a highly efficient way. Fig. 6 shows a read. Writes from the debugger to the target system are similarly efficient: the turnaround cycle follows the target response but no further turnaround is needed between the data and the next command header.



Fig, 7: SWJ-DP conceptual model

Serial Wire/JTAG Debug Port

SWD allows for an easy and risk-free migration from JTAG, as the data signal **SWDIO** and clock **SWCLK** can be overlaid on the JTAG **TMS** and **TCK** pins.

The *Serial Wire/JTAG Debug Port* (SWJ-DP) provides a mechanism to select between JTAG and SWD interfaces. This enables bi-modal devices that provide both SWD and JTAG interfaces without additional pins.

In logical terms, the SWJ-DP consists of a wrapper around the JTAG-DP and SW-DP. Its function is to select JTAG or SWD as the interface. Fig, 7 shows such a logical arrangement.

The use of a JTAG debug interface must be maintained in where it is vital to:

- — enable inclusion in an existing scan chain;
- — enable the device to be cascaded with legacy devices which use JTAG for debug, although this can also be supported using a JTAG access port (Fig. 5);
- — enable use of legacy tools, for example TAPs accessed by *Automatic Test Equipment* (ATE).

An SoC fitted with SWJ-DP support can be connected to legacy JTAG equipment without modification. If an SWD tool is available, then only two pins are required. The two additional JTAG pins are therefore released for alternative functions.

The switching scheme is arranged so that an SWD debugger is able to connect by sending a specific sequence on **TMS**. The sequence has no effect on JTAG devices, as it is arranged so that the JTAG TAP never leaves the "top-4" states (*Test-Logic-Reset*, *Run-Test/Idle*, *Select-IR-Scan* and *Select-DR-Scan*). SWD defines a similar SWD-to-JTAG sequence.

C. Multi-drop Serial Wire Debug

SWD protocol version 1 is a simple point-to-point architecture, supporting connection between a single host and a single device.

The only way you can access multiple devices is by using multiple, independent connections from the host. In more complex systems, this has a number of disadvantages:

- — it complicates the physical connection standard, by having variants with different numbers of connections;
- — it increases the number of pins required on a package with multiple dies inside;

- it increases the number of pins required on the connector on the device PCB; this may be unacceptable where size is a limiting factor;
- it makes it difficult to integrate multiple independent platforms accessed by SWD into the same chip.

To solve requires a connection that can be shared between multiple SWD devices. SWD protocol version 2 [8] adds such a *multi-drop* capability, which:

- enables a two wire host connection to communicate simultaneously with multiple devices;
- is synthesis friendly, implemented using single-edge clock and separate bidirectional data;
- permits a device to power down completely while that device is not selected;
- is backwards-compatible: provision of multi-drop support in a device does not break point-to-point compatibility with existing host equipment that does not support multi-drop extensions;
- prevents multiple devices from driving the wire simultaneously, and continues to support the wire being actively driven both HIGH and LOW, maintaining a high maximum clock speed;
- enables multi-drop connections sharing a connection with devices that do not implement SWD;
- enables an effectively unlimited number of devices to be connected simultaneously, subject to electrical constraints (Fig. 8).

Dormant operation and interoperability with other protocols

SWD protocol version 2 defines a *dormant mode* operating state. Using dormant mode allows the target to be placed into a quiescent state. A debugger selects the required device and protocol, and when it has finished with the device, places it back into dormant mode.

There is no requirement for all the devices on the shared connection to implement SWD protocol. They must implement a protocol that provides a quiescent state with a mechanism for entering and leaving that state that is compatible with, but not necessarily identical to, the SWD dormant mode selection protocol. In IEEE 1149.7 a compatible dormant mode is known as *offline state*.

This allows multiple JTAG TAP, SWD, and SWJ devices to share a physical connection to a host, as shown in Fig. 9. These different devices may be in different packages, on different dies in a single package, or on a single die: for instance, implementing CoreSight debug and boundary-scan test with a single connection.

Selecting a device from dormant mode

When in dormant mode, a device is listening for a *selection message*. However, some other device will be actively using the interface, and hence it is important that data traffic on the interface is not mistaken for the selection message.

In choosing a selection message, the designers of multi-drop SWD were aware of the requirement to support multiple protocols, in particular IEEE 1149.7.

IEEE 1149.7 out-of-band messages were rejected as they require complicate the implementation by using clock as data.

Hence the designers of multi-drop SWD chose an *unlikely data sequence* approach. The selection message consists of a 128-bit *selection alert*, followed by a *protocol selection command*. This selection alert method has been adopted by IEEE 1149.7, and multi-drop SWD has adopted the IEEE 1149.7 protocol selection command, ensuring compatibility between the two protocols.

The selection alert sequence was set to 128-bits long to

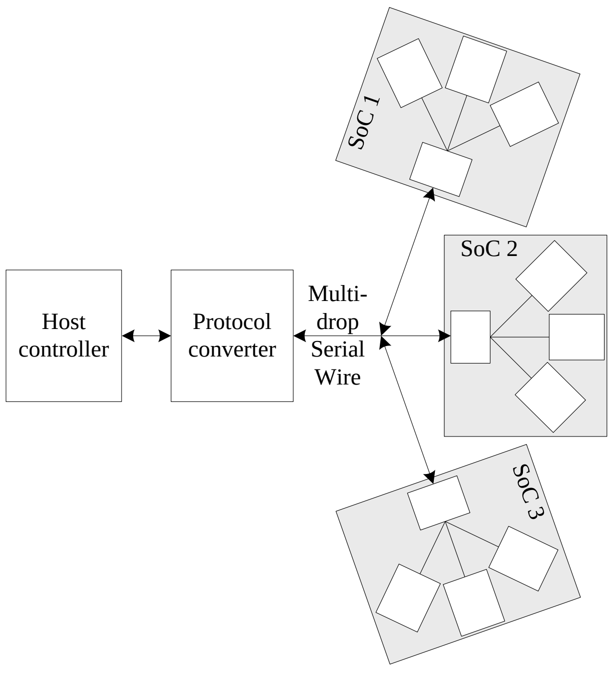


Fig. 8: Multi-drop Serial Wire Debug

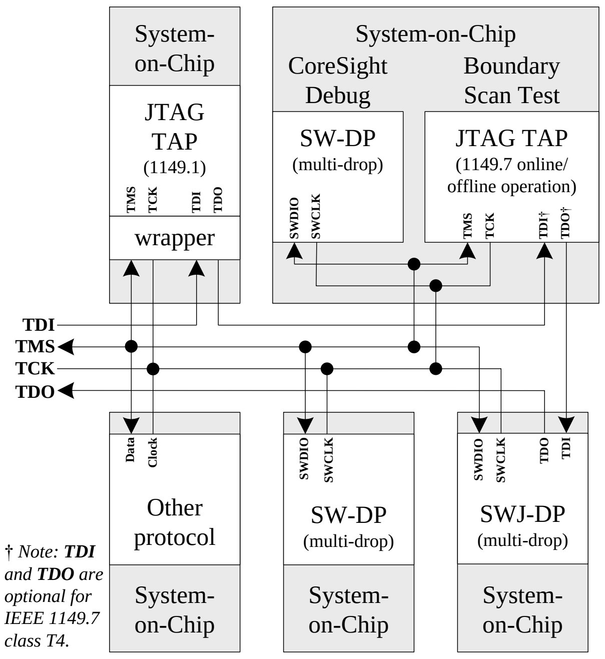


Fig. 9: Multiple JTAG, SW, SWJ (multi-drop) and other protocol devices on a shared connection

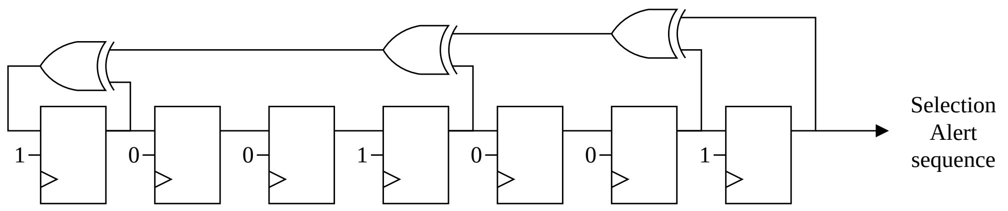


Fig. 10: LFSR for generating Selection Alert sequence

make it vanishingly unlikely for the same sequence to appear in the data traffic of another protocol.

The selection alert sequence can be generated by implementing a *linear feedback shift register* (LFSR). The sequence starts with a zero start bit and continues with the output of the LFSR. This provides a very efficient implementation (Fig. 10).

IV. Conclusions

The development of the Serial Wire Debug interface standard and protocol has provided an alternative to JTAG for debug, which has the additional benefits of reduced pin count and higher performance. It meets all the requirements for a debug interface, offering the following features and advantages:

— only uses two pins: this is vital for very low connectivity devices or packages;

— supports multiple devices connected simultaneously using the multi-drop extensions;

— is inter-operable with other debug and test interfaces;

— provides debug communication to JTAG TAP controllers;

— enables the debugger to become another AMBA bus master for access to system memory and peripheral or debug registers;

— high performance data rates: approx. 1.5Mbytes/sec at 20MHz with a single data pin;

— low power: no extra power or ground pins required;

— small silicon area: approx. 2,500 additional gates;

— synthesis-friendly with high maximum clock rate: single-edge clock and separate bidirectional data with full cycle turnaround;

— low tools costs: < $100 build costs;

— reliable and safe: built-in error detection.

SWD offers a risk-free migration path for JTAG-based debug through the SWD/JTAG debug port and the multi-drop extensions.

A debug architecture is ultimately only as good as the tools ecosystem that supports it.

As SWD is a standard interface backed by the industry's leading IP provider, the software developer can count on a wide choice of interoperable tools from many tool vendors.

Since its introduction in 2003, the CoreSight architecture has rapidly gained support in the marketplace. SWD is supported by major tools vendors and is widely implemented in devices ranging from low-cost mass-market microcontrollers to complex SoCs, making SWD the *de facto* standard for a low pin-count debug port.

Acknowledgement

The author gratefully acknowledges the contributions and assistance of colleagues at ARM Limited.

References

[1] IEEE, "Standard Test Access Port and Boundary-Scan Architecture," IEEE Std 1149.1-2001, 2001.

[2] IEEE, "Draft Standard for Reduced-pin and Enhanced-functionality Test Access Port and Boundary Scan Architecture," IEEE P1149.7, unpublished.

[3] S. Lau, "Reinventing JTAG for SoC debugging," *Embedded.com*, August 2008. [Online] Available: Embedded.com http://www.embedded.com/design/testissue/210200584?_requestid=309116 [Accessed Aug. 28, 2009].

[4] ARM Ltd., "ARM Debug Interface v5 Architecture Specification," ARM IHI 0031, 2006. [Online] Available: http://www.arm.com/products/solutions/ADISpecification.html

[5] E. Ashfield, et al, "Serial Wire Debug and the CoreSight Debug and Trace Architecture." [Online] Available: http://www.arm.com/miscPDFs/15531.pdf [Accessed Aug. 28, 2009]

[6] ARM Ltd., "CoreSight Architecture Specification Rev 1.0," ARM IHI 0029B, 2005. [Online] Available: http://www.arm.com/products/solutions/coresight_spec.html

[7] W. Orme, (2008) "Debug and Trace for Multicore SoCs." [Online] Available: http://www.arm.com/pdfs/CoresightWhitepaper.pdf [Accessed Aug. 28, 2009]

[8] ARM Ltd., "ARM Debug Interface v5 Architecture Specification ADIv5.1 Supplement," ARM DSA09-PRDC-008772, 2009. [Online] Available: http://www.arm.com/products/solutions/ADISpecification.html

Proprietary Notice