Serial Wire Debug and the CoreSight™ Debug and Trace Architecture

Eddie Ashfield, Ian Field, Peter Harrod*, Sean Houlihane, William Orme and Sheldon Woodhouse

ARM Ltd
110 Fulbourn Road, Cambridge, CB1 9NJ, UK

***peter.harrod@arm.com**

Abstract

This paper describes a reduced pin-count debug interface, known as the Serial Wire Debug Interface, which has been developed as a 2-pin alternative to a traditional IEEE1149.1 compliant interface (JTAG). It provides the interface to debug and trace functionality on processor cores and System on Chip (SoC) devices, especially those that conform to the CoreSight debug and trace architecture[1]. This 2-pin interface has benefits for devices in severely pin-limited packages (e.g. microcontrollers, such as the ARM Cortex-M3) or in designs where few package pins are available for the debug interface. It is also suitable as a debug and trace interface on cost-sensitive packaged products (e.g. mobile phones), where the width of any external connectors must be kept to a minimum, while still providing performance as good as or better than traditional JTAG debug. The additional benefits that Serial Wire Debug provides, for all systems whether pin-limited or not, will be described in this paper: these apply not only for traditional software debug and trace but also for the debug and diagnosis of first silicon and complex hardware/software interactions.

Serial Wire Debug

When developing this alternative to JTAG as the interface for debug and trace, the opportunity was taken to analyse where the requirements for debug and trace differed from those for test and to design the protocol appropriately. In order to exploit the full power of this new interface standard, a packet-based protocol has been developed; this is in contrast to the scan in, scan out protocol of JTAG.

The packet protocol is split into Header, Response and Data, with the data being skipped if the interface is not ready.

Although the Serial Wire Debug protocol is not compatible directly with JTAG, it can be used to connect to legacy JTAG devices, giving additional benefits (e.g. clock and power isolation) over a direct, daisy-chained JTAG architecture. Furthermore, it is possible to connect a debug tool to both JTAG and Serial Wire Debug (SWD) devices over the same connector, by overlaying the SWD pins over existing JTAG pins and using a switching protocol to switch between JTAG and SWD.

SWD Benefits

In addition to the reduction in pin count on the interface, SWD has the following benefits:

- **Performance**: SWD is able to make use of the full clock cycle for data transfer, from rising edge to rising edge of the Serial Wire clock. This is in contrast to JTAG where data is driven on the falling edge and sampled on the next rising edge, giving only one phase for the data to propagate. This has the effect of enabling SWD to be run at up to twice the frequency of JTAG in the same technology. Some efficiency is lost due to the packet-based nature of the SWD protocol but this is minimal. In addition, SWD supports pushed operations – see below. Pushed operations can improve performance, for instance in situations where writes might be faster than reads.

- **Error detection:** SWD provides some protection against errors. It implements simple parity checking and can also check for overrun, enabling blocks of commands to be sent on a high latency, high throughput connection. SWD also gives confirmation that the physical connection is OK, independent of system level operation.

- **Tools.** It is possible to build low cost, lower performance tools very simply.

- **Migration.** Overlaying SWD over JTAG provides a migration path that doesn't force users to upgrade their test hardware.

- **Interface to debug and trace infrastructure:** SWD provides full access to the debug and trace functionality on an SoC. It provides the communication channel, giving full access, via the internal debug bus (the DAP bus), to a CoreSight compliant system.

Pushed Operations

The SWD protocol is designed to make the use of pushed operations easy. For a pushed operation, the value written as an access port transaction is used at the debug port level to compare against a target read. As an example, in order to verify a memory image after transfer to the target, a block of data would normally be read by the debugger and compared with the original. The process of performing a read may involve several layers of handshaking between the read request being issued by the debugger, and valid data being returned through the debug interface. In a pushed operation, the two-way data flow becomes uni-directional. The reference values are transmitted to the target, which takes care of the read and compare. Once a block has been processed, the status (pass or the address of a failure) can be retrieved with a single read operation.

SWD in a Bus-based Debug and Trace Environment

In Figure 1, the combined Serial Wire and JTAG Debug Port (SWJ-DP) is shown as the debug interface to a SOC.

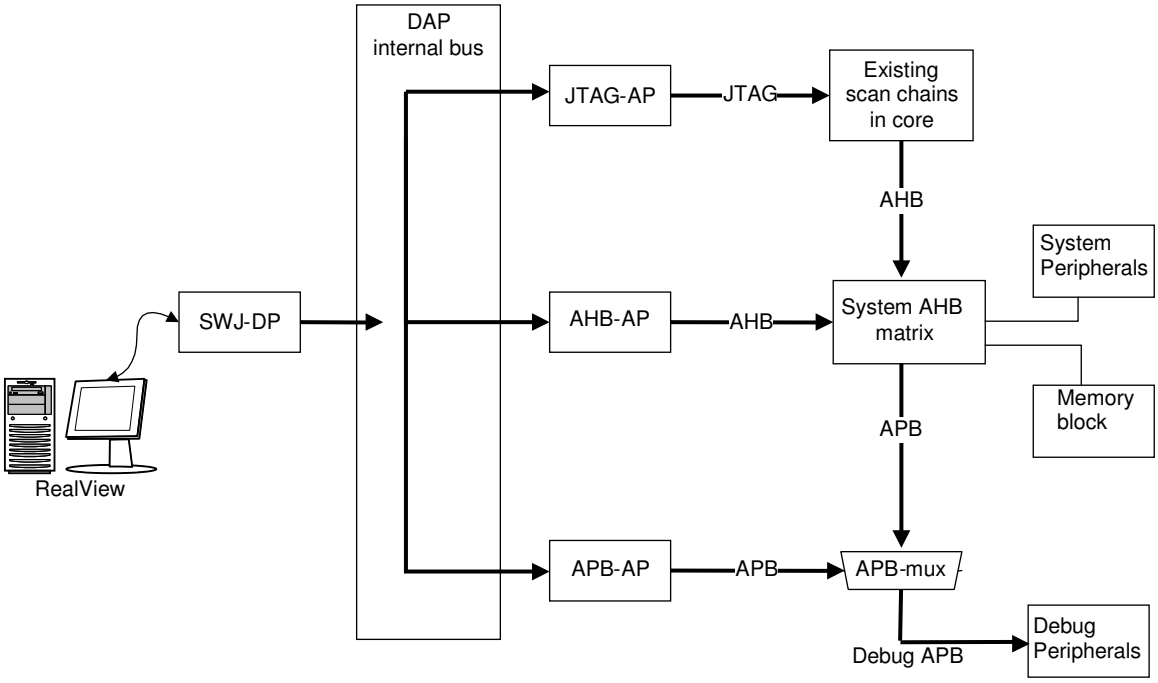

Figure 1 - Serial Wire Debug as interface to a CoreSight Debug and Trace System

Connecting through the DAP internal bus, SWJ-DP various slave devices:

- legacy JTAG-equipped cores via the JTAG Access Port (JTAG-AP)
- system memory via the AMBA High Performance Bus Access Port (AHB-AP)
- bus-based debug functionality on the debug APB bus via the AMBA Peripheral Bus Access Port (APB-AP).
- dedicated debug control devices (not shown in figure 1)

An APB mux is provided so that the CPU in the system can also access debug components.

SWD in a Bus-based Debug and Trace Environment

The move from using a JTAG scan interface to using a bus-based approach for debug control and access is the most significant change introduced with SWD. This approach acknowledges that modern silicon designs frequently contain multiple IP blocks, sometimes pre-hardened, often using multiple clock and power domains. The debug infrastructure is more modular and has removed the requirement for a traditional JTAG scan chain on chip. Consequently the Serial Wire Debug interface no longer needs to support this style of debug architecture directly; instead it provides a narrow channel to a fully bus-based debug and trace infrastructure.
The bus based approach enables register-based access of debug configuration and status information: this provides a more consistent programmers' model and eases

software development, as reflected in the ARM Debug Interface v5 for ARM Cortex cores. These same registers can also be accessed by the CPU itself, giving additional flexibility. With the debug channel being less tightly coupled to the internal debug infrastructure, it becomes easier to optimise the physical protocol to match the external interface as has been done for SWD.

A bus-based debug architecture can also provide efficient access to design for debug features, meaning that the same paradigm can be used for both applications debug and silicon debug and diagnosis – and possibly also for repair in the future. As the debug infrastructure relies less on re-using the core functionality, difficult problems such as system lockup can be probed using the standard debug tools.

This style of debug architecture has additional benefits:

- It permits debug access while the CPU is running, possibly non-intrusively
- Debug access is more abstract, less CPU-specific and more scalable

The DAP supports true power and clock isolation which is becoming essential in most battery-powered applications, where it must be possible to power down individual blocks of logic or control the clocks independently, including when debugging.

Devices that are intended for security applications might need special logic to enable unlocking of debug functionality, via the use of a Debug Authentication Module, which may be added to the DAP bus. Since the debug infrastructure is separated from the primary system function, it becomes simpler and less risky to mask debug access to individual parts of a system. Equally, it is now possible to access the debug infrastructure while all of the functional logic is powered down or held in reset. Making it easy to add vendor specific components such as a Debug Authentication Module helps vendors to protect their secure implementations.

DAP Accesses

Before explaining the Serial Wire protocol, it is necessary to first cover the format of an access to the debug port. As can be seen in Figure 1, the DAP bus has a single master (the DP – or Debug Port - which is the external facing part) and one or more slaves (the APs – or Access Ports) which are typically used to interface to the various on-chip bus standards, or to provide dedicated debug features. Each transaction sent from the external debugger is addressed to a single one of these components (the DP or an AP).

AP Accesses

Each AP provides up to 64 registers of 32 bits, arranged in 16 banks of 4, including one register which identifies the particular AP type. In the case of an AP which accesses a memory mapped part of the target system, a pair or registers are used as address and data, each access to the Data Read/Write register resulting in an access to the target system being performed. The AP can be designed to optimise common functions (for example an address incrementor allows a write to *n* addresses using only *n+1* writes to the AP), and can also provide detailed status and control of the system segment to which it is attached.

DP Accesses

Registers in the DP are used to provide status and control of the external link (SWD), interface to power and reset controls and provide modifiers for accesses which are passed to the APs. The DP also has registers to select the current AP, and the register bank within that AP. The Control/Status register for SWJ-DP is shown in Figure 2 below.

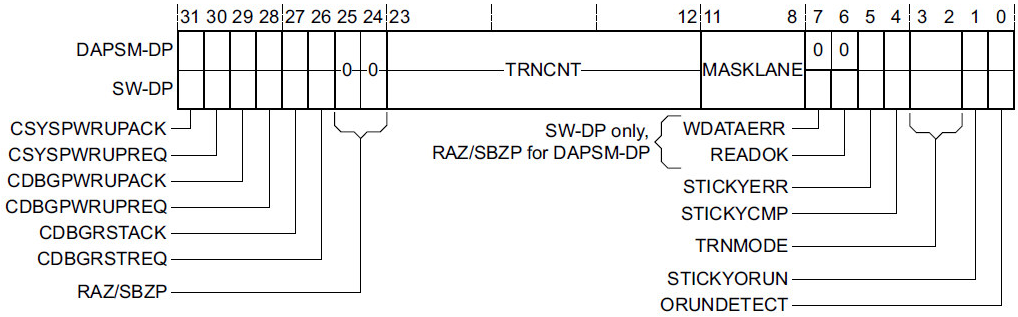

Figure 2 – SW-DP CTL/STAT register

The DP also provides functions which are designed to improve the performance of common debug operations. When the transfer count is set, the next AP access will be converted into a stream of accesses within the ASIC, independent of the clock frequency provided to the external interface. This can be used in conjunction with the Transfer Mode and Bit Mask, which allow the DP to perform a 'read and compare' operation. These functions together can be used for efficient block fill, memory verify, search for match, search for non-match – and at close to system speed when the data value is constant.

Sticky Bits

Having described the structure of the DAP, it is now possible to introduce the sticky bits, which play an important part in the Serial Wire Debug protocol. Transfers from the debugger can fail – either due to data corruption, or a system being unable to perform a requested access – or a pushed comparison (using the comparator in the DP) may be triggered. When these conditions occur, it is necessary for the debugger to recognise the fact, and to be able to recover to the point where the event occurred. Rather than requiring the external debugger to check for success after every single access, the DP has a number of sticky bits which can be set, and once these bits are set, most new transfers will be accepted, but ignored. Although the debugger is not required to respond immediately to the state of the sticky bits, the SWD protocol provides an indication of any of the 'ERR' or 'STICKY' bits from the status register in its handshake on every transfer. It is important to recognise that the status of the physical interface and the status of the debug port are isolated, if the link is broken then re-attached, the port will be found to have retained its state – even to the point of allowing a read which was in progress when the link was lost to be recovered without re-reading it from the system (since repeating that read is not always possible).

SWD Protocol

Each successful SWD transfer consists of 3 parts:

- A header (always from the external debugger)
- An acknowledgement from the target (provided it recognises the header)
- A data payload, the direction of which is determined by the header.

A write transfer is shown in Figure 3 below.

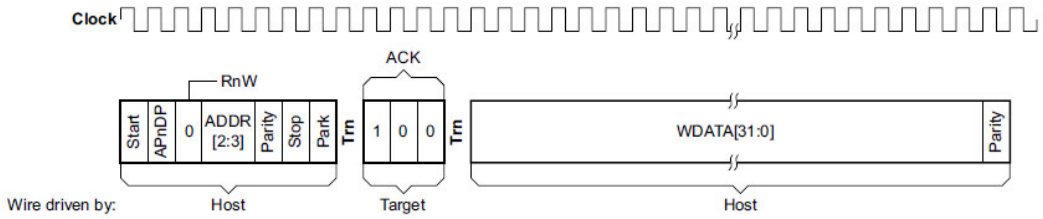

Figure 3 - Successful SWD Write Transfer

A start bit is used to enable the line to idle, a period when the clock can be stopped or free running (note that the clock also does not need to have a set frequency).
One bit defines the transfer as an AP access or a DAP access, the direction of data on the SWD interface is provided, and 2 address bits are given. These address bits allow a sequence of AP accesses to use the 4 registers in a bank of a specific AP without having to change the AP select register in the DP. A parity bit and a stop bit are added to provide some tolerance to data corruption and hot plugging. The header ends by driving the line high, where it should be held by a pull-up. After the header, the target will respond (after a single cycle) giving an indication of the status of the interface, and if the acknowledgement matches the OK pattern, write data is sent with a parity bit. A successful read transfer is similar, as shown in Figure 4.

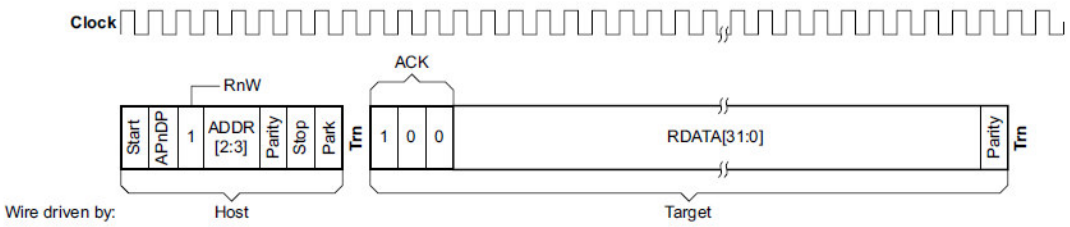

Figure 4 - Successful SWD Read Transfer

The turn-round cycle (TRN in the diagrams) is placed after the data phase for a read, as there is no change of direction between ACK and RDATA. For both reads and writes, the packet is 46 clock cycles, with a payload of 32 bits. In situations where the debugger hardware does not permit analysis and reaction to the ACK bits (for example an ASIC vector replay tester, or a simple device on a high latency interface), the packet timing can be fixed with these 46 cycle frames. Improved bandwidth efficiency can be obtained in the normal mode of operation, where the data phase is only present after an ‘OK’ acknowledge phase. This mode uses a shorter packet, as shown in Figure 5 below if the debug port is not yet ready to receive a new transaction.

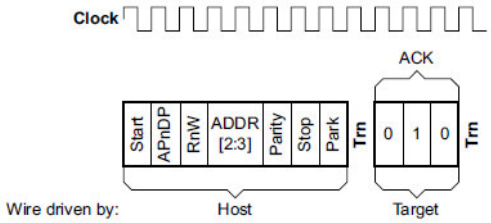

Figure 5 - SWD Wait response.

This response indicates to the debugger that the debug port is still active, and that the communication link is operating, but there is an outstanding transfer which has not completed. This packet is 13 cycles long, and reduces the bandwidth penalty of performing debug accesses which are faster than the target is able to accept them.

It is important to note that the protocol is optimised for performing blocks of transfers, and both read and write data are buffered. When a read transfer is issued on the SWD interface, the response will be the result from the previous read. Thus to read an ASIC memory location, typically 3 transfers are necessary:

- Write to the AP's Transfer Address Register with the target address
- Read the AP's Data Transfer Register to initiate the transaction.
- Read a benign register (DP status for example) to return the required target data.

Similarly, if it is necessary to determine that a write access to the system has completed, that write has to be followed by a DP access, which can return a WAIT response if the write is still in progress.

SWD has a similar packet format which is used when a sticky bit is set. This uses the FAULT response as shown in Figure 6 below.

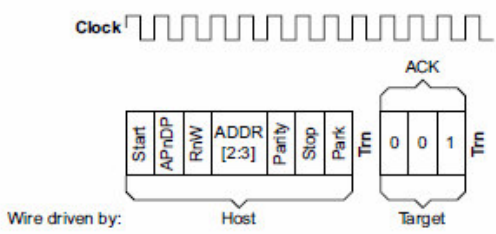

Figure 6 - SWD FAULT response

The FAULT response indicates that the link is still active, but the debug port will only respond to a read of its ID or Status registers, or a write to its ABORT register which is used to clear the state of any sticky bits once they have been read.
Using these responses, the status of the debug interface and the status of the debug infrastructure are separated, making it possible for a debug session to remain connected when system clocks are stopped. If there is a fault and access via a system port becomes deadlocked, the active AP can be instructed to terminate its transaction on the DAP bus. In addition to allowing the AP to be interrogated and its state determined, this frees up any remaining debug infrastructure in the target device giving the possibility of probing the deadlock scenario, possibly giving valuable insight into the cause of the lockup.

Conclusion

The development of the Serial Wire Debug interface standard and protocol has provided a reduced pin count alternative to JTAG, which has the additional benefits of higher performance and error detection that a packet-based communication protocol can bring. Serial Wire Debug is an effective mechanism for accessing a modern bus-based debug and trace design, the packet nature of the communication being well-matched with this bus-based architecture.

References

1. http://www.arm.com/products/solutions/CoreSight.html