Solutions for Embedded Systems Development

ARM On-Chip Debug Interfaces – Chances and Limitations

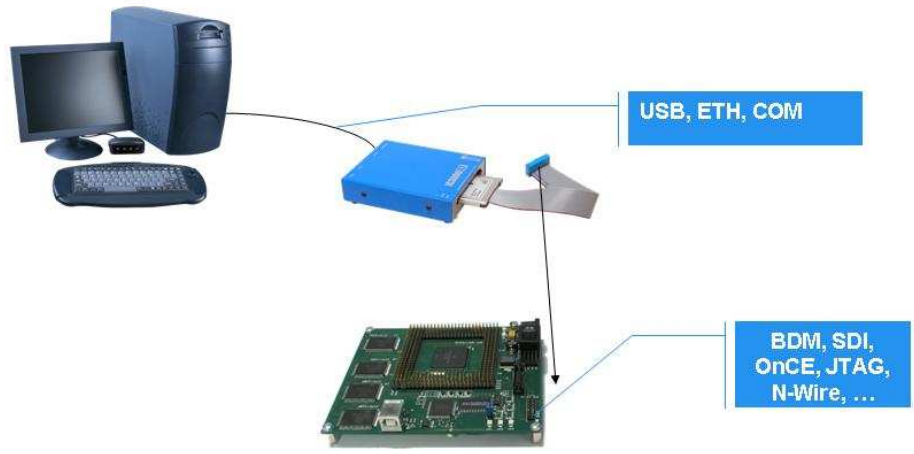
Holger Wild, iSYSTEM AG

When selecting and evaluating debug interfaces and tools for ARM microcontrollers many aspects have to be regarded. This article provides some guidance what should be taken into consideration. More detailed information is available from ARM or the according chip manufacturer. We will discuss:

What are On-Chip debug interfaces (OCD)?
How is the ARM OCD used?
What are important differences of OCD debug tools?
What are the differences of JTAG, CORTEX SWD/SWT and ETM (ETB)?

What are On-chip debug interfaces (OCD)?

On-chip debugging is a method to start, stop and read the core (or a user application) by having additional silicium on every MCU of a distinct manufacturers´ series. To access this specific MCU area (similar to a CPU in a MCU) one or more dedicated lines are used that are then no longer available to the user. These lines are referenced as the OCD interface. Often these lines can be used in multiple ways (e.g. ETM trace PIN-select register, see 4). This must be considered when designing the destination hardware.
ARM uses JTAG as OCD interface. This interface was originally designed for boundary scans and uses at least 5 lines (plus ground). Section 4 describes the details of various debug interfaces.



Picture 1: Attachment of a hardware debugger to target system over JTAG interface

How is the ARM OCD used?

To use the OCD a specific protocol and a tight timing with distinct voltage levels is needed on the lines of the OCD interface. Due to that a PC is not capable to control this interface directly. A sophisticated level converter between PC and MCU is necessary, a so called hardware debugger.
The simplest form of hardware debuggers are passive devices or TTL ICs connected to the PCs´ LPT port – a so called “wiggler”. This hardware is controlled by debugger software (e.g. GNU “GDB”). Today there are many hardware debuggers available. Dependent on price and quality of the debugger software most hardware debuggers use 8bit or 16bit controllers and the slow serial interface or USB 1.1 interface. But there are also universal and expandable high-performance devices with powerful, integrated 32bit controller and high speed USB 2.0 as well as Ethernet interface. Methods for team work is integrated as well as an API interface to remote control the tool with standard PC programming languages. Both are necessary for work flow automation.

What are important differences of OCD debug tools?

There are major differences between OCD Debug Tools caused by the MCU manufacturer, the hardware debugger and the debugger software. Many tool manufacturers use only the simplest features of the MCU. Other tool providers offer in the debugger software the direct flashing of the MCU and enhancements for on-chip ETM trace usage (see also 4, Embedded Trace Macrocell) and different MCUs. The often higher price for these sophisticated tools is compensated by shorter development times and an easy to use development and test environment.

Here are important differences between high-end and lower-end tools (this list is not complete!):

Speed of the interface to the PC:
Many USB 2.0 tools use just “full speed” which is 12Mbit/sec or only USB 1.1. High-end tools support “high speed” USB with 480Mbit/sec or Ethernet. That speeds up not only the flash process but also debugger reaction (e.g. window update) and the ETM trace (see 4).

Programming of the internal and/or external flash supported by debugger:
Before flashing the software high-end tools can set the MCU speed to maximum (e.g. by INI file) and the flash process as well as the verify process are run in the fast RAM monitor of the target MCU.
Here an example that shows the differences in speed when loading some larger code into MCU internal flash (all other conditions stay the same): flash process with a lower-end tool (without verify) takes 74 seconds, flash process with a high-end tool (including verify) runs 10 seconds. This time lag is there at every software change.

Flash-Patch-Breaks:
With ARM cores only two MCU internal break points are available, with CORTEX there are at least 4 (+2) hardware break points (see 4).
For the debug tool often one break point is necessary (for high level language debugging). Beside that complex trigger functionalities also need several break points.
Up to date hardware debuggers offer ways for code breaks to use so called “Flash Breaks” in the flash (in unlimited number) and to use the internal triggers for high level requirements instead for simple program stop.

User Interface (GUI):
“Easy to use” is one of the most important criteria when selecting a tool. Well known is the “look&feel” of Microsoft’s Visual Studio and the ECLIPSE user interface. Although the ECLIPSE user interface seems pretty complex to start with, it is independent from any manufacturer.

Unlimited usage of the Debug Communication Channel (**DCC**) with ARM7:
DCC is a one-byte connection between the application and the debugger software through the hardware debugger. To use the DCC the target MCU has to run a monitor software. High-end tool manufacturers offer code – sometimes also the source code – to use the DCC in terms of the user, for example to implement a kind of “printf” to output the value of a variable or for a terminal function.

Usage of different compilers for debugging:
The quality of the compiler is always a snapshot and depends on the application. To compare different compilers and to detect compiler errors it is important that the debugger software allows the integration of different compilers. This should be no problem if the debugger manufacturer conforms to the “ELF” output data specification and stores all debug information there.

What are the differences between JTAG, CORTEX, SWD/SWT and ETM (ETB)?

Below table compares OCD values before we see the details of ARM OCDs (JTAG, ETM, CoreSight with CORTEX).

General Overview					
	JTAG	**ETM7**	**ETM9**	**Cortex M3 SWD**	**Cortex M3 JTAG**
Debug Pins minimum + GND	5	5 + 10	5 + 10	2 (3 if SWT trace)	5
Debug Pins maximum (MCU dependent) + GND	7	27 (16bit trace-lines)	27 (16bit trace-lines)	CORTEX ETM, SWT 2 lines	CORTEX ETM, +4 lines for synchronous trace, not available for low-pin-count MCUs
Connector Plug (Mictor includes die JTAG function!)	14 / 20 pin connector	standardized MICTOR connector	standardized MICTOR connector	no standard	no standard, like JTAG or more compact
DCC Usage	yes	yes	yes	replaced by SWD	replaced by SWD
RT Watches with Monitor (#1)	yes	yes	yes	included in product	included in product
"real" RT-Watches	no	by trace, can be limited	by trace, can be limited	yes	yes
On-Chip Trace	no	trace port	trace port	yes – limited, but other internal functions	yes – limited, but other internal functions
Number of Hardware-Breakpoints in MCU	2	2	2	FPB + DWT, often 6 Code + 2 Comp.	FPB + DWT, often 6 Code + 2 Comp.
Soft- or Flash-Patch-Breaks (#2)	available	available	available	available	available
ETM + ETB in MCU, read-out via JTAG	available, but could require new tool	not necessary	not necessary	available, but could require new tool	available, but could require new tool

Notes on table 1: (#1) wastes MCU processing power due to monitor usage, (#2) depends on tool: low-cost tools do not use any or only low-speed flash patch breaks

Table 1: Comparison of different ARM OCD interfaces

JTAG Debug Interface

Most ARM processors are controlled by external hardware debuggers over the JTAG interface (Cortex SWD = at least 2-3 pins + GND) except very few derivates with bound-out chip (e.g. TI TMS 470 family with additional emulator features). For JTAG at least 5 (up to 7) pins of the MCU are reserved. This is a problem for low-pin-count controllers because other MCUs need only one line for debugging. There are even more MCUs that allow to switch off the JTAG interface and to use these pins differently.
Here is a list of more features:

- Flashing of the MCU embedded flash and/or the external flash (method and performance depend on tool), downloads into the internal and external RAM
- No real time watches without monitor program (regular update of variable content without influence on run time behaviour); real time watches will be

available with CORTEX core

- System reset and debug reset are connected in many ARM MCUs. Because debug takes longer in ARM MCU to stop the application program, the MCU will start up and execute code at any rate. If this code portion is faulty or sets distinct, one-time writeable SFRs, there is a failure and/or the controller is locked and no longer accessible by JTAG debugger. To workaround this problem include a loop that runs approximately one second. This behaviour is only exhibited by ARM-MCU, other MCUs won't show this.

- ARM cores released before CORTEX without ETM cannot provide trace information without real-time interruption and/or additional, instrumented code which is sometimes pretty bulky (code is provided over interface or DCC) Standard reports like profiler, code execution coverage, watches or tracing (without code instrumentation or interface output) on the target system cannot be made.

Pin Name	Type	Description
TMS	Input	**Test Mode Select**. The TMS pin selects the next state in the TAP state machine
TCK	Input	**Test Clock**. This allows shifting of the data in, on the TMS and TDI pins. It is a positive edge-triggered clock with the TMS and TCK signals that define the internal state of the device.
TDI	Input	**Test Data In**. This is the serial data input for the shift register.
TDO	Output	**Test Data Output**. This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal
nTRST	Input	**Test Reset**. The nRTST pin can be used to reset the test logic within the EmbeddedICE logic.
RTCK	Output	**Returned Test Clock**. Extra signal added to the JTAG port. Required for designs based on ARM7TDMI-S processor core. Multi-ICE (Development system from ARM) uses this signal to maintain synchronization with targets having slow or widely varying clock frequency. For details refer to "Multi-ICE System Design considerations Application Note 72 (ARM DAI 0072A)".

Table 2: Standard JTAG signals at ARM7 (without GND, Reset or potentially necessary VRef)

Résumé on JTAG:

For challenging real-time applications (e.g. in avionic, automotive and medical systems) the basic JTAG debug functions are not sufficient. There is no way to evaluate the real-time behaviour without elaborate and time-consuming code instrumentation, DCC usage, code analysis tools and/or expensive simulators that change the run-time behaviour of the application. Due to that development times for applications increase while the test opportunities decrease.

Good simulators cost the same as a good JTAG debugger, but often they fit together only with distinct compilers and have to emulate all external devices (e.g. CAN, FlexRay, UART, interrupt…). The user must learn all debugger functions plus the configuration of the numerous simulated MCU parts to achieve a feasible result. Real-time behaviour cannot be evaluated – and the complexity of a simulator often leads to faulty configuration and operation by the user. In case of ETM sometimes another simulator is necessary which requires training on the new user interface.

ARM offers a workaround that can be licensed and included in the MCU by the MCU manufacturer: ETM = Embedded Trace Macrocell.

CORTEX-M CoreSight

CORTEX-M CoreSight includes many new features (see table 1). ETM can be integrated although some functions are available without ETM also.

Here a list of the most significant differences:

- Less lines are necessary because the hardware debugger can be connected by JTAG or by 2 lines only – "Serial Wire Debug" = SWD (+GND + RESET + pot. VREF). This is a major advantage on low-pin count MCUs.

- SWD provides real-time watches that allow regular view and update of variable content without influence on run time behaviour in the debugger.

- To access the internal asynchronous trace functions SWDIO (=TMS) and SWDCLK (=TCLK) and an additional line SWO (=TDO) are used. This asynchronous trace is called Serial Wire Trace (SWT).
  With that every 64 CPU cycles the program counter can be dumped for a snapshot of the program flow. Besides that other counters (e.g. exception counter, sleep counter etc.) can be selected for dump. Often these counters are 8bit, but set on overflow an event that can trigger the debugger.

- SWT allows to trigger on data access and to dump these values and/or program counter. In case of asynchronous trace this output is restricted by the bottle neck of the serial line that leads often to "trace hardware overflows". Therefore trace data volume should be limited and tests should be limited to small parts of the application.

- Another nice feature is the Instrumentation Trace Modul (ITM). Like a "printf" values can be written to specific virtual trace ports with a few commands in the application. These ports can be read by SWT. For that code instrumentation is necessary.

- At CORTEX-M a synchronous trace uses four additional lines and has a very high throughput.

ETM = Embedded Trace Macrocell

ETM is an add-on on ARM CORE with different characteristics that allows to dump directly the program flow and with some restrictions data accesses. ETM is a superset of JTAG debugging and needs more dedicated lines as pure JTAG (see

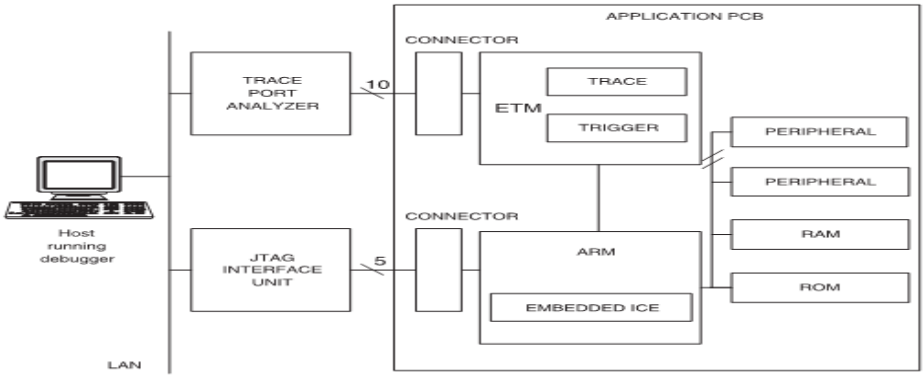
table 1). There are different levels and variants of ETM. With every new generation ARM provided significant improvements.

With CORTEX the ETM will also trace peripheral events (up to now no CORTEX-M with ETM is available; with other ARM the ETM is connected directly to the core).

NXP ARM7 ETM (and other ARM derivates) toggle the additional lines by the PIN-SELECT register (SFR). These lines cannot be used differently if ETM is applied. This can lead to a situation where a low-pin count NXP ARM7 MCU either has a single free port available for target I/O or provides ETM trace, but not both. However within this family a larger MCU can be found that can be used piggy back with small software changes. The advantages of ETM counterbalance the costs of this change.

ETM requires e.g. at ARM nearly the same chip space as the core itself. At larger ARMs the ratio is much better why ETM is often included with ARM9. Despite higher costs NXP integrates a small ETM solution in ARM 7 LPC series, where other manufacturers like e.g. ATMEL have no alternative.

To prevent problems with improper connectors for this high-speed interface ETM uses so called “Mictor” connectors. These connectors are expensive, but very well suited, durable and a common standard. JTAG lines are integrated which means that no separate JTAG interface has to be implemented. In mass production a needle bed adapter can save costs.



Pic. 2: ETM implementation with NXP ARM7

An up-to-date hardware debugger contains both debugger units for ETM (in contrast to above picture) and is joined to the target system with one Mictor connector.

ETM/NEXUS: For automotive applications Freescale provides MCUs with ARM core that contain NEXUS trace interface instead ETM. Both interfaces exhibit similar functions and performance levels. ETM is standard invented and maintained by ARM, NEXUS is a conciliation between different MCU and tool manufacturers. More information is available on http://www.arm.com/miscPDFs/11655.pdf and www.nexus5001.org .

To get the advantage of all features of ETM the user has to buy additional hardware from the tool manufacturer (see pic. 3). Low cost solutions are not feasible, often not even available. Many users do not know about the advantages of these trace interfaces because tool manufacturers often do not support them at all or only limited.

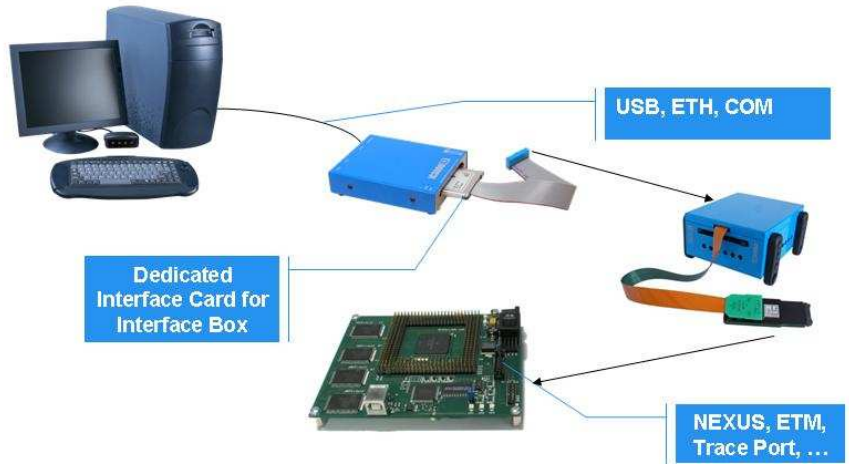
Embedded Trace Buffer = **ETB**: Instead of additional external lines ARM provides a very small, dedicated RAM buffer internally in the MCU. ETB can be read by a hardware debugger using JTAG/SWD. By that MCU pins and a dedicated high speed trace port are saved. ETB can help to detect problems with variable access, but an evaluation about program run time is impossible – same as with JTAG or CORTEX SWD/SWT.

Several MCU manufacturers solved the problem at NEXUS (and in some extent also at ETM9) in the following way: they provide two different or an additional chassis of the chip. With higher pin count the ETM/NEXUS port is led out, with smaller chip chassis it's not. The user can test his application with the larger version – with piggy back solutions even in the target hardware.

Most suitable seems to have ETB and the lines led out. With that the customer can use both depending on the request. Beside the run time trace the features of ETM depend on the ARM controller version and the implementation of the MCU manufacturer and as well from the attached hardware debugger.

How does ETM work?

Before program start and also during run time trigger, filter and output criteria are set directly in the ETM by the hardware debugger or debugger software using JTAG commands. With 4 to 16 output lines and a low trace clock is it not possible to dump a complete program run of a high performance CPU. But with a little trick it is possible.



Pic 3: Debugger with ETM trace box attached to an ETM interface over Mictor connector

The trick is to pass over ETM only direct branch messages. These consist of very few compressed bytes. The MCU communicates e.g. “10 commands executed, 50 commands executed, 3 commands executed”… Besides that there are sometimes sync or other messages to know exactly the program counter status and also data outputs (e.g. value of a global variable). The ETM trace tool (see pic. 3) stores the static image of the MCU program and is now able to reconstruct the program run including interrupts or other characteristics. The tool time stamps the messages at the ETM port and has all correlations in real time. This works fine for static applications and for applications linked to fixed addresses. For dynamic code or dynamically changing RTOS projects no ETM code trace is possible.

To buffer fast changing program runs ETM owns an internal FIFO buffer (size depends on MCU type).

Per default ETM has a control flag that slows down the CPU to prevent a buffer overflow. If this flag is not set or not implemented, messages are lost when the buffer is full. ETM notifies the trace tool of this situation and at the next sync everything is fine again. When code is executed without data output this is not a problem even if the maximum clock frequency is used, provided that ports and tools are high performance.

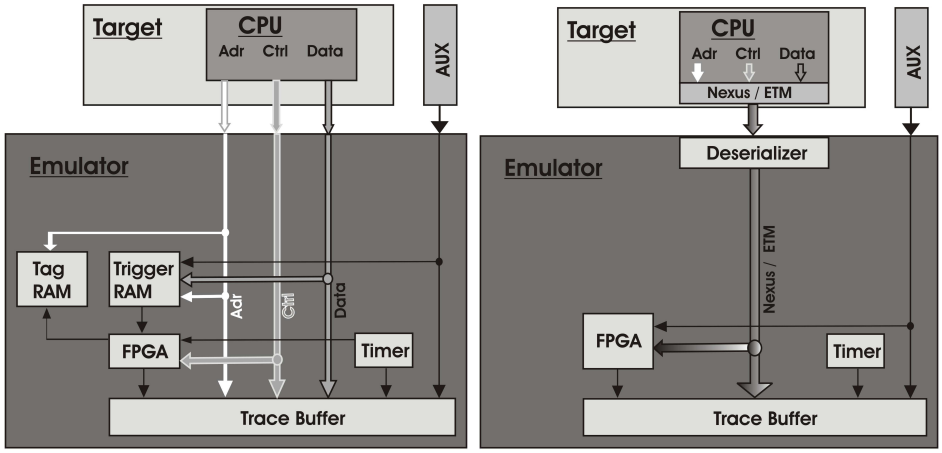
It is totally different when data output is required because many 32bit addresses and 32bit data must be transferred and can be barely compressed. The strategy to trace all read/writes plus executed code is not an option because this will cause a buffer overflow and data loss. Instead use data comparators (see table 1) integrated in ETM to filter data on chip. This is also possible for code.

Comparision between In-Circuit emulator and On-Chip trace

The ICE uses the trigger and filter logic of the CPU bus directly because the emulator is connected to the external CPU bus. Extensive trace functions can record the unfiltered address, data and control bus (see pic. 1, left) or use the complex trigger and filter logic of the emulator.

On-chip-trace (NEXUS, ETM, trace buffer or special trace ports) was introduced because of the challenges of the constantly increasing miniaturization and integration on outsped micro controllers. Due to the integration of the whole memory on the chip an external bus is no longer needed. This reduces the number of chip pins while increasing the maximum internal clock frequency. Unfortunately tracing functions of an ICE are almost lost.

Sophisticated technologies (hardware and software) integrated in modern tools allow the usage of almost all ICE runtime analysis functions with on-chip trace. iSYSTEM calls this technology RTR (Real-Time-Trace Reconstruction: www.isystem.com/iTraceGT)



Picture 4: Extensive trigger and filter logic on an ICE (left) and the “simple” On-Chip logic of NEXUS or ETM (right)

The data rate at the ETM port is very high. Real time analysis is only with a high performance tool possible. All direct branch messages and data output have to be buffered and stored – even if the MCU filters the data in advance. Nowadays storage media have gigabytes volumes. 128kbytes ETM buffer tools (or smaller see ETB) are not applicable.
High performance FPGA tools control and manage those gigabyte storages and offer new features. Upload times for 1MB in a 1GB buffer are not perceptible at all and so the trace is available to the user immediately.

ETM also offer event counters, sequencer and other useful stuff. The handling of the ARM features especially when triggering is pretty complex.
Sophisticated tools offer an easy to use wizard to spare the user the study of ARM manuals and to make the handling nice and easy.

What are the features of the ETM interface?

ETM tools provide following features without code instrumentation and without run time changes:

- Trace including filter and trigger for code and data
- Code execution coverage analysis to detect “dead” and untested code
- Decision coverage to evaluate which code branches are executed
- Code profiling for performance tuning

Conclusion on ETM:

Because many customers do not know about the advantages of ETM, it’s not considered when selecting an ARM MCU. ETM helps in the development process and also during test and documentation. The features offered by ETM supersede on-chip JTAG or CORTEX SWD/SWT. Good tools have a modular design and are extensible, but the user interface stays the same. This is also true between different chip manufacturers.

Additional information:
Embedded Trace Macrocell Beschreibung:
http://www.arm.com/products/solutions/ETM.html
http://www.arm.com/miscPDFs/11655.pdf
Informationen rund um On-Chip Tracing: http://www.isystem.com/ItraceGT

Autor: Holger Wild, holger.wild@isystem.com, Tel. 08138 6971 50