A Primer: ARM® Trace

Including:
ETM™, ETB and Serial Wire Viewer, JTAG and SWD
Agenda

- Introduction
- How we talk to your CPU using JTAG or SWD.
- Trace. ETM™, ETB and SWV. How are they different?
- Triggers, filters and stuff.....
- Instrumentation. DCC and ITM.
- Breakpoints and Watchpoints
- What is trace good for? Some points.
Debugging Skill Set

- Trace: around for a long, long time.
- Not many use it – maybe < 5%. The best do.
- Trace helps you find the nastiest problems...
- And often fairly easily... just like in...Mexico
- ...is good for testing software and hardware too.
- Developer productivity is very important....
- So – let’s look at the various technologies ARM has....
ARM Public Processor Roadmap

ARM Cortex

ARM7, 9, 11
ARM9
ARM926EJ-S
ARM7TDMI
ARM11

Application
Real-time
Microcontroller

EmbeddedICE
CoreSight™

Up to 2.5 GHz
Up to 2 GHz
~600 to 1 GHz
72 – 150 + MHz
50 MHz
200+ MHz
200+ MHz

V8 (64 Bi
Cortex-A5
Cortex-A5
Cortex-A15
Cortex-A12
Cortex-A9 (Dual)
Cortex-A9 (MPCore™)
Cortex-A8
Cortex-A5
Cortex-A7
Cortex-R7
Cortex-R5
Cortex-M0
Cortex-M0+

Not to scale

All dates are 2007
### Versions, cores and architectures?

- What is the difference between ARM7™ and ARMv7?
- Search for ARM architecture on Wikipedia to get the full list.
- ARM doesn’t make chips….well maybe a few test chips.

<table>
<thead>
<tr>
<th>Family</th>
<th>Architecture</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>ARMv4T</td>
<td>ARM7TDMI(S)</td>
</tr>
<tr>
<td>ARM9</td>
<td>ARMv5TE(J)</td>
<td>ARM926EJ-S, ARM966E-S</td>
</tr>
<tr>
<td>ARM11</td>
<td>ARMv6 (T2)</td>
<td>ARM1136(F), 1156T2(F)-S, 1176JZ(F), ARM11 MPCore™</td>
</tr>
<tr>
<td>Cortex-A</td>
<td>ARMv7-A</td>
<td>Cortex A8, A9, A9MP</td>
</tr>
<tr>
<td>Cortex-R</td>
<td>ARMv7-R</td>
<td>Cortex R4(F)</td>
</tr>
<tr>
<td>Cortex-M</td>
<td>ARMv7-M</td>
<td>Cortex M3, M4</td>
</tr>
<tr>
<td></td>
<td>ARMv6-M</td>
<td>Cortex M1, M0</td>
</tr>
</tbody>
</table>
What is Trace good for? Part 1

- Tells where the program has been and how it got there.
- and for how long.
- Nasty problems can often be found quickly with trace.
- Especially where the bug occurs a long time before the consequences are seen.
- Or where the state of the system disappears with a change in scope(s). What caused the problem is off the stack.
- Race Conditions. A really big benefit here!
- Asynchronous events lining up.... as in an RTOS.
What is Trace good for? Part 2

These are the types of problems that can be found with trace:

- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), corrupted stack.
- Out of bounds data. Uninitialized variables and arrays.
- Slow programs – is something eating up your CPU time?
- No trace availability responsible for ongoing problems.
What is Trace good for? Part 3

These are the types of problems that can be found with trace:

- **Stack overflows.** What causes the stack to grow bigger than it should?

- **Runaway programs:** Your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. But this is a very common problem. Stack not always has the answer.

- Communication protocol and timing issues.

- **Profile Analyzer.** Where is the CPU spending its time?

- **Code Coverage.** Was all the code exercised? Might be a certification requirement (FDA & FAA etc).

- **Execution Profiling:** Times to execute source code.
Overall ARM Debugging Modules

- **EmbeddedICE:**
  - The original modules for ARM processors.
  - JTAG port.
  - ETM, ETB, breakpoints, watchpoints.
  - ARM7™, ARM9™, ARM11™

- **CoreSight**: newer debug technology
  - Found mostly on Cortex processors.
  - JTAG & SWD (Serial Wire Debug) ports.
  - ETM, ETB, SWV, breakpoints, watchpoints.
  - PTM (Program Trace Macrocell) Cortex-A9, Cortex-A15

- Some items are optional – *check your datasheet.*
What CoreSight delivers to you:

- Provides program flow and other activity from the processor.
- And in real-time without:
  - intrusion, stolen CPU cycles or instrumentation code.
  - And without stopping the CPU including most configuration!
- This means you can debug on-the-fly!
- Can see:
  - Program counters
  - Data reads and writes
  - Exception (including interrupt) activity...and all are timestamped!
  - CPU counters.
The debugging parts of ARM processors:

- **JTAG debugger**: (ICE & CoreSight)
  - JTAG is really the port to the debugging module in core.
- **SWD – Serial Wire Debug**: (CoreSight)
  - A 2 pin alternative to JTAG – same capabilities. (no boundary scan)
- **ETM**: Embedded Trace Macrocell (ICE & CoreSight)
  - Sends out PC and data reads and writes as fast as the core is. (E.ice)
  - CoreSight only sends out all PC values. Use SWV for other data.
- **PTM**: Like ETM but for Cortex-A9 and later.
- **ETB**: Embedded Trace Buffer (ICE & CoreSight)
  - Get ETM info out the JTAG connector – not ETM connector.
  - Is visible to the CPU.
- **SWV**: Serial Wire Viewer (CoreSight only)
  - 1 wire output: can see PC samples, data r/w and exceptions + more.
- **MEM-AP**: Memory Access Port - read/write memory real time.
A few more bits ‘n pieces:

- These debug modules sit on the chip beside the core.
- Can see internal busses.
- Do not affect speed of core or steal cycles.
- We will call this “real-time”. Or Non-intrusive.
- Data Communication Channel. EmbeddedICE
- ITM® – Instrumentation Trace Macrocell.
- These are used to send messages or data out the JTAG or SWO ports in real-time. More on these two later...
- But ITM is really cool as you will see. Almost no code needed
- Multicore Debug – cross triggering between cores.
JTAG Debugging

- This is how we talk to the core.
- With this setup we can:
  - Start and stop program.
  - Single Step one source or assembler line.
  - Set breakpoints and watchpoints when CPU is stopped.
  - Note: with ARM CoreSight can set breakpoints on the fly
  - Read and write memory only when CPU is stopped. (CoreSight can !)
  - Program FLASH memory. Internal or External. Via a debugger.
  - JTAG used to test circuit board connections from chip to chip. (boundary scan)
- Can stop CPU then look and see state of system.
- Much better than monitor systems.
- Debug instructions are put directly into the core pipeline.
- JTAG adapters are called many things – but they are not in-circuit emulators (ICE).
- JTAG = Joint Test Access Group (IEEE 1149.1)
JTAG Connector – Keil STR9 Board

Diagram showing JTAG connector with labels like JTRST, JTDI, JTMS, JTC, JRTCK, JTD, and others, along with resistors and +3.3V connections.
JTAG Scan Chain

- Serial in and out in a “chain”
- Chip can have > 1 module or “TAP” “test access port”
- Such as CPU, Flash program, FPGA, ETM, etc.
- Can “Bypass” a TAP if not used. Saves time
- Data is clocked serially in and out by TCK
- TDI = + edge, TDO = - edge
Scan Chain – a couple of acronyms

- **ID** – ID Code. 32 bits. Provides info about device
- **IR** – Instruction Register – in bits
  - IR = All ones = bypass. (i.e. 11111 for 5 bit)
  - In and out of device is one flip-flop – BYPASS saves time
CoreSight on Cortex-M3 JTAG vs SWD:

Platform: MCBSTM32E (Debug/Trace using CSHD Probe)
SWD: Serial Wire Debug

- New to CoreSight. A 2 wire debug port.
- Very popular on Cortex processors.
- Bidirectional line (SWD) + clock (SWCLK). Simpler than JTAG.
- Same functionality as JTAG except no boundary scan.
- Most adapters now have this.
- Serial Wire Viewer goes hand-in-hand with SWD.
- SWD a.k.a. SW.
**SWD pins?**

- Only 2 pins to control ARM core:
  - SWD – signal – shares TMS.
  - SWCLK – clock – shares TCK.

- SWO – Serial Wire Output – shares TDO.
- SWD on your chip? Check your datasheet.
- Most common on Cortex-M0, M1, M3, M4, R4.
Serial Wire Connections:

- Our JTAG diagram from before:
- 3 wire debug (+ ground).
- **RED** is Serial Wire.
- Talk about SWO later.

- All Debug connectors have both JTAG and SWD.
Cortex-M3 Hi-Density 10 pin

- 10 pin **NEW** connector.
- Note JTAG and SWD pins are present here too!
- No nTRST pin.
- SWD in CoreSight only.

**TIP:** DDI0314F_coresight_component_trm.pdf on www.arm.com
How do we use SWD?

- End user – just plug in connector and select in the software.
- SWD in the core is automatically selected.
- Debugger will do necessary steps.
- This means debugger must be SWD capable. Most are.
- Select speed as in JTAG.
JTAG/SWD Adapters

- **USB-JTAG adapter** (for Flash programming too)
  - ULINK2 - $395  *(ULINK2 and ME - SWV only – no ETM)*
  - ULINKpro - $1,395  – Cortex-M3 SWV & ETM trace
  - Signum JTAGJetTrace – Cortex-M3 SWV & ETM - $1,500,  ARM9+ - $3,500 to $4,500
  - Segger J-Link and J-Trace are about the same prices.
    - **Note**: If you use an ETM adapter – don’t need a JTAG one too.
So what do we have so far?

- JTAG debugging
- Serial Wire Debug
- These are used to connect your debugger with the core.

What is next on our agenda?
- Tracing!
- 1st ETM, then SWV.
Why Trace?

- Shows history of instructions as they happened.
- This is called TRACE – collected in trace memory somewhere.
- Very high speed – as fast as the CPU...but there are limits.
- Can usually configure and view trace without stopping CPU.
- Often associated with Triggering and Filtering.
- These used to narrow your search. (else billions of records)
- Some have internal memory, use ETB or are Streaming Trace.
ETM: Embedded Trace Macrocell:

- For high-end users or those with really nasty problems.
- Cost is $1,500 to $4,000+. For some customers, it is worth it.
- ETM has an easy, quick payback time.
- Displays where the program went and what data was read or written plus when exceptions happened.
- Non-intrusive. Real-time. Uses no CPU cycles. Most customers will know if they need ETM.
- Controlled through JTAG or SWD.
- Data sent out the special port: Trace Port.
What comes out the ETM?

- **Instruction Trace:**
  - Address of executed instruction indicating program flow.
  - Instruction will be disassembled by debugger.

- **Data Trace:** (only EmbeddedICE – CoreSight uses SWV)
  - The data value and its address resulting from a load or store.
  - Instruction causing this will be displayed on another packet.
  - Can send out only address or data to save time.
  - Why?

- Can see/record either everything out the trace or filter out what frames you don’t want.

- A timestamp is provided.
How do we connect to ETM?

- ETM is 1, 2, 4, 8, 16, 24 or 32 bit output.
- Uses a special Mictor connector.
- 1 Mictor up to 16 bit.

- JTAG/SWD signals on Mictor too.
- Speed to 480 mHZ or more <?>
ETM Pins...Mictor connector

- 38 pin Mictor for fast chips – STR9.
- Note JTAG and SW pins are present.
- Trace clock.
- Can use for M3/M4
More on the ETM...

- Cortex-M3, M4 can use newer, smaller connector.
- This is on Keil MCBSTM32E board.
ETM Cortex-M3/M4 Hi-Density 20 pin

- 20 pin **NEW** connector.
- Has ETM pins.
- JTAG, SW
- Note JTAG pins are present here too!
- Trace simplified.
- TraceCLK PCB layout more critical than trace data.

The pinout of the ARM high-density 20-pin Cortex connector. Top view.

**TIP:** DDI0314F_coresight_component_trm.pdf on www.arm.com
Where is ETM on the chip pins?

- ETM shares pins – often not the best ones....
- But usually GPIO...but not always. See datasheet.
- If need ETM – please leave these GPIO open...If you can....
- If customer might use/need ETM – avoid these pins.
- Can be a big problem.... A show stopper for tracing...

- NAND Flash memory controller.
- Ethernet controller pins.
- Check potential devices to see what is shared.

- ETM is very fast – as fast as the core.
- If the core goes into the weeds – ETM will still function.
ETM: Embedded Trace Macrocell:

- ETM provides instructions: program flow is obvious.
- PC, data R/W’s, Exceptions, timestamp.
- Triggers and Filter Can filter out stuff we don’t want to see.
- Assembly and C source code is displayed.
ETM Operating Modes:

- Need to set this in debugger (some automatic).
- Set processor pins to ETM mode
- Configure ETM
- Can then capture ETM trace records.
- Display in screen.
- Should not have to stop program execution to view trace.
So...is confusing – how do I set all these?

- Check your data sheet.
- Trial and error with best guess....
- Best: call your emulator manufacturer tech support.
- If not set right – probably will not get any traces. Or junk.
- Look for indication of trace clock value. This is important.
- Debugger sets this value. ETM can return this value.
Code Coverage: provided by ETM

- Was every instruction executed and tested?
- You do not want untested code in your product that might get activated under some strange condition!
- Can save to a file.
Performance Analyzer: provided by ETM

- Where did your program spend all its time?
- Can often be quite illuminating and unexpected.
- ETM provides the information.
- Could use PC Samples but must run a longer time.
ETM Triggers and Filters

- **Triggers** – stop either/both program exec or trace collection when some specified event happens
- **Filters** – collect or see only the specified data you want to see
- **Pre- or post- filtering:**
  (don’t want to fill up expensive buffer with junk)
- Triggers and filters allow you to zero in on your problem
- Effective and easy to set T & F
- Takes a bit of practice to be really efficient
- Can do a lot with a little training
ETM Triggers and Filters

- Here is Signum’s tracing configuration window. Easy to use.
- Buffers in debuggers usually 1 – 4 Mbytes.
- Others can stream trace frames into your PC memory for a long, long time.
ARM RVDS Trace Filters

- ARM RVD Profiler + Filters shown.
ETB: what is this?

- When frames sent out ETM port. Very fast - $$\text{debugger}$$.
- What if we could access ETM through JTAG or SWD?
- Could use ordinary (and cheaper) JTAG/SWD adapters.
- This is ETB – Embedded Trace Buffer.
- Is memory – an ETM buffer.
- Size set by chip designer.
- 2, 4 or 8 Kbytes are common.
- Memory accessible by user program via data bus.
- Can access ETM for test or diagnostic purposes.
ETB Advantages

- No expensive ETM hardware required.
- Any JTAG adapter will work. Debugger has to have ETB support.
- If core is too fast – is the only way to get trace data.
- Can be triggered and filtered.
- You can easily access ETB memory with your application programs !!!!! PHONE HOME !!!

Note:
- EmbeddedICE: ETB accessed like any other memory.
- CoreSight: ETB memory accessed through a register.
ETB Disadvantages

- Memory size is very small. (2 to 8K)
- Can’t save large sections of the core run.
- Must use filters and triggers to focus on data.
- Might have to stop the program to read data. (depends)
- ETB memory might be irresistible to “steal”.
- Then can’t use ETB to debug.
ETM & ETB Summary

- ETM is very, very fast. This is why ETM adapters are expensive.
- Displays all PC values, data reads & writes...all timestamped.
- Can filter and trigger.
- ETB a low cost way to get ETM info out JTAG port.
- Cortex-Mx and Cortex-A9 don’t have data tracing out ETM.
- Use Serial Wire Viewer for data R/Ws and exceptions.
- But – check your datasheet to see what is implemented.
Serial Wire Viewer - Introduction

- Only on Cortex microcontrollers:
  - Serial Wire Debug (SW-DB) to connect to core. (2 wires)
  - Serial Wire Viewer (SWV) module in microcontroller.
  - Serial Wire Output (SWO) to output the data. (1 wire)
  - SW-DB and SWO are on the standard JTAG connector.

- STMicroelectronics, Luminary, NXP, Toshiba ++ many more Cortex cores have SWV. Standard for Cortex-M3, M4.
SWO Block Diagram

- Can still have JTAG but – must use SWD for SWO.
- ETM + SWO becoming popular: ST, NXP, Toshiba, Fujitsu...
- SWO is only 1 serial pin output. Limiting.
Serial Wire Viewer: SWV

- Plugs into the regular JTAG connector.
- Non-intrusive. (except for ITM)
- Can view variables and peripherals in Real Time!
- Plus exceptions! All in real-time!
- All other ARM debug methods are still available.
- Can have statistical Profile analysis.
- Code Coverage a problem – would have to run a long time.
- Instruction sampling and data tracing.
- Real-time viewing of RTOS data!
- SWV can come out the SWO pin or Trace Port.
Serial Wire Viewer has these features:

- Data Read and Data Write tracing...*in real time!*
- Can view data in graph, watch, memory or Serial Wire Viewer, Trace Records windows.
- Hardware Breakpoints and Watchpoints.
- ETM trigger (if the processor is so equipped).
- Program Counter (PC) or Data address sampler registers (Four 32 bit registers).
- ITM Viewer (Instruction Trace Macrocell): `printf` debugging
More on SWV:

- In addition, the SWV trace module has counters for:
  - # of CPU clock cycles (32 bits).
  - Total cycles spent in interrupt (exception) processing (8 bit).
  - # of cycles processor is sleeping (8 bit).
  - Total cycles spent in load/store operations.
  - # of instruction cycles (8 bit).
  - # of folded instructions count (8 bits).

- Is useful to see how many extra cycles are used for instructions.
Serial Wire Viewer SWV

- View variables many different ways:

Open this in View/Serial Window/ITM Viewer.
Customer software writes to ITM: comes out SWO.
Serial Wire Viewer SWV

- Logic Analyzer Window:
- Pot on MCBSTM32™ is being rotated.
Trace Windows

- Three Trace Windows:
  1) Trace Records
  2) Exceptions
  3) Counters

- Displays types of information from the trace.
- Cortex-M Target Driver Setup window.
- Will be captured.
- If we select too many variables, this might cause a data overflow.
Trace Records

- Timestamp, PC sample, Read/Write PC, Exceptions, ITM.
- Filter window to remove certain lines. Post.
- Can change filters if CPU is running.
- Time delay and lost cycles noted.
Exception Trace

- Provides information about exceptions. Real-time
- Count and times as shown below.

<table>
<thead>
<tr>
<th>Num</th>
<th>Name</th>
<th>Count</th>
<th>Total Time</th>
<th>Min Time In</th>
<th>Max Time In</th>
<th>Min Time Out</th>
<th>Max Time Out</th>
<th>First Time [s]</th>
<th>Last Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NMI</td>
<td>18324</td>
<td>84.375 ms</td>
<td>0 s</td>
<td>39.980 us</td>
<td>3.293 ms</td>
<td>6.665 ms</td>
<td>0.03109988</td>
<td>61.11085556</td>
</tr>
<tr>
<td>3</td>
<td>HardFault</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MemManage</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>BusFault</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>UsageFault</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SVCall</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>DbgMon</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PendSV</td>
<td>0</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td>0 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SysTick</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 1

- Here only Data writes displayed in Trace Records.
- Write to address 0x20000004 – see the memory window.
- Data increments 1.
- Timestamp shown.
- Both windows are updated in real time!
Example 2

- Cortex-M3 evaluation board. 125 mHz.
- Program is caught in loop 1922 to 1926.
- Trace shows you quickly where you have been.
- PC Sample good for Profile Analysis.
- Note we missed many instructions in this fast tight loop.
- If we want all PCs – must use ETM.
Serial Wire Viewer SWV can see:

- Global and Static variables, Structures.
- Peripheral registers – just read or write to them.
- Can’t see local variables. (just make them global or static)
- Can’t see DMA transfers – DMA bypasses CPU.
- SWV is very easy to setup and use.
- SWV doesn’t have ETM’s throughput – but is still powerful.
- And no expensive ETM adapters are needed – ULINK2 or ME, ULINKPro, ST-Link or Segger J-Link !
Output Data of SWV and ETM - differences

- Serial Wire Viewer
  - Program Counter (PC) samples (from every 64\textsuperscript{th} to 5,000)
  - Exceptions – both system and external (to core).
  - Data read and writes.
  - ITM – Instruction Trace Macrocell. \texttt{Printf} outputs.
  - Some counters on CPU operations like folded instructions etc.
  - Timestamps.

- ETM
  - Every PC is captured and timestamped 😊

- Note: you can see SWV and ETM go hand in hand!
DCC and ITM – what are these?

- A method to send or receive messages from user to debugger.
- Except for your read or write to a 32 bit register – is real-time.
- Data Communications Channel: DCC
  - send and receive – 1 transmit and 1 receive register.
  - ARM EmbeddedICE only.
  - ARM has “Real Time Agent” to send info out DCC. Intrusive.
- Instruction Trace Macrocell: ITM
  - Transmit only. CoreSight only. Replaces DCC.
  - 32 32-bit Stimulus Registers.
  - This is the printf debugging.
- Can R/W memory real-time with Memory Access Port.
How to use the ITM:

- Define it:
  
  ```c
  #define ITM_Port8(n) (*((volatile unsigned char*)(0xE0000000+4*n)))
  ```

- Send some data out the SWO pin:
  
  ```c
  ITM_Port8(0) = ch;
  ```

- The write to ITM intrusive – all other operations to get this out the SWO pin are non-intrusive.

- This is much easier to use than Real Time Agent with DCC.
Summary:

- Three ways to connect to the debugger in ARM microcontrollers:
  - Note: To Start, Stop, set breakpoints & watchpoints, R/W memory.
  1) Regular JTAG connector.
  2) ETM connector.
  3) Serial Wire Debug (SW-DB) (this shares the JTAG connector).

- Five ways to get debug information out of an ARM processor:
  1) Real Time Agent: Debug Communications Channel (DCC) - JTAG (not CorSight)
  2) ETM: Embedded Trace Macrocell. (not all processors have this option)
  3) Serial Wire Viewer (SWV): out the Serial Wire Output pin (SWO).
  4) ITM Viewer – Instruction Trace Macrocell (comes out SWO pin).
  5) Plus, well OK sure – stop the processor. But you will stop it a lot.
PCB Hints for Cortex-M3 ETM™

1. Only one trace connector (preferably new 20-pin connector).
2. Physical placement (as close to the chip as possible) - it is easy to do so with new, compact 20-pin connector.
3. JTAG connector (new 10-pin) should be there too, but do not need to be so close to the CPU. Please note, that 10 pins on that connector are same as exactly the half of new 20-connector, so all JTAG pins may "pass through" the trace 20-pin connector.
4. How to run TRACECLK and TRACEDATA signals on PCB: short and parallel to each other, best would be a ground in between them. Note that all trace pads are in the middle of one side of the CPU package. There are GND and VCC pads in the middle of trace pads.
5. Orientation for proper clearance and orientation with emulators.
6. Pads for new 20-pin ETM connector should be on every board –the new connector does not take too much space.

Remember Cortex-M3 parts gone = 144 mHz so layout more important.
What else is there?

- **Watchpoints**
  - Conditional breakpoints – stop program when xyz...

- **PTM: Program Trace Macrocell**
  - For Cortex-A8 and A15 and beyond.
  - No data tracing. ITM tracing exists.

- **HSST: High Speed Serial Trace**
  - Exists today.
What is in the future?

- **STM: System Trace Macrocell**
  - Up to application level debugging.
  - On Cortex-R and A – will eventually replace ITM.

- **TMC: Trace Memory Controller**
  - Adds ability to send data out Ethernet or HSSTP.

- **Linux application aware debuggers using trace.**
Conclusions:

- Trace is an excellent feature.
- Now you know how this all fits together.
- ETM and SWV are easier to use than you might think.
- Thank you!

Comments:
Robert Boys
Bob.boys@arm.com
San Jose, California