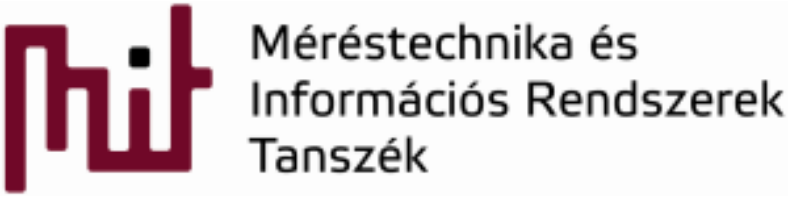
Nagyteljesítményű mikrovezérlők

9. Debug-olás

Scherer Balázs



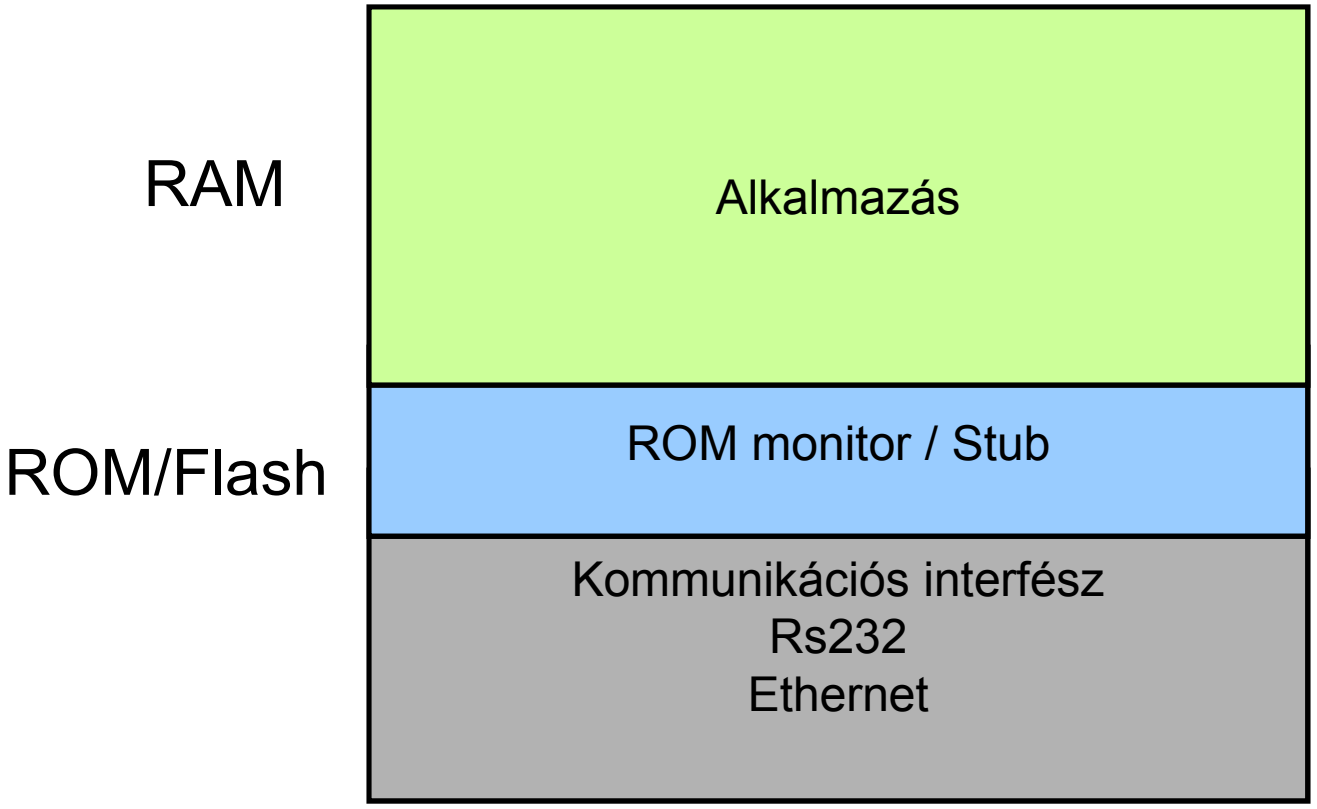
Budapest University of Technology and Economics
Department of Measurement and Information Systems

Tradicionális debug módszerek

- Hagyományos debug
  - Oscilloscope
  - Logikai analizátor
  - Printf
  - LED villogtatás
- In-Circuit Emulatorok
  - Debug variáns
  - Dual port RAM
    - Hardware-es break pointok
    - Regiszter figyelés

Tradicionális debug módszerek

- ROM monitor
  - GDB stub



Rom monitor

- Csak a RAM terület használható az alkalmazás által
  - Régi külső RAM-os 32 bites megoldások
- Kommunikáció a ROM monitor és a debugger között
  - GDB: Remote Serial Protocol

GDB: GNU Debugger I.

- GDB alapvetően egy parancssori debugger, amihez vannak megjelenítők
  - DDD
  - Eclipse

Minta GDB session:

```
localhost$ gdb a.out                    // A GDB elindítása
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
(gdb) set remotebaud 57600
(gdb) target rdi com1      // connect to target machine, process, or file
(gdb) load                   // dynamically link file and add its symbols
Loading section .text, size 0x1280 vma 0x1000
Loading section .data, size 0x760 vma 0x2280
Loading section .stack, size 0x10 vma 0x30000
Start address 0x1000
Transfer rate: 53120 bits in 1 sec.
```

GDB: GNU Debugger II.

```
(gdb) break main              // breakpoint b [file:]line or function
Breakpoint 1 at 0x8048476: file test.c, line 5.
(gdb) continue              // continue running your program
Breakpoint 1, main () at test.c:5
5 for( i = 0; i 10; i++ ) {
(gdb) display j          // show value of expr each time program stops [according to
format f ]
1: j = 1074136126
(gdb) step     // stepping program
6 j = i * 2 + 1;
1: j = 1074136126
(gdb) step     // stepping program
5 for( i = 0; i 10; i++ ) {
1: j = 1
(gdb) quit     // quiting debugg session
```

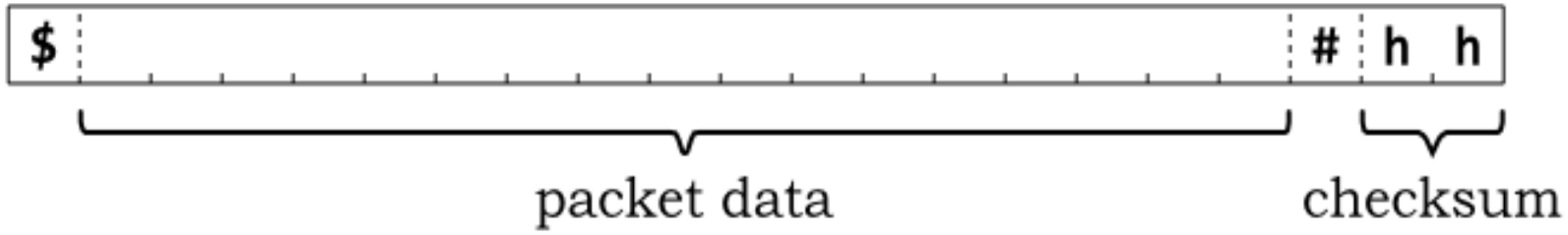
GDB: Parancsok

- ***run:*** Program futtatás
- ***continue:*** program folytatás
- ***next:*** következő utasítás
- ***step:*** következő utasítás step-into
- ***list:*** forrás kód ki listázás
- ***break:*** break point elhelyezés (hw-es breakpointok néha macerásak)
  - ***disable/ enable***
- ***print:*** változó értékének kiírása
- ***set:*** változó érték adása

És még sok egyéb

GDB: Remote serial protocol

- Valahogy tartani kell a kapcsolatot a target-tel
  - Rs232
  - TCP protokoll
- ASCII karakterekből álló parancsok
  - $-jellel kezdődnek
  - #-al majd egy 8bites cheksum-al záródnak
- A stub
  - + al válaszol az elfogadott – al a hibás checksumra.
  - „OK”-val, vagy hibakóddal a parancsra



Gyakran használt RSP parancsok

- **Read Registers (g):** kiolvassa a target összes regiszterét:
- **Write Register n (P):** egy specifikus regiszter érékét írja
- **Read Memory (m):** egy memória területet olvas
- **Write Memory (M):** egy memória területre ír.
- **step (s):** Lépteti egyet a processzort. A + on kívül nincs rá más válasz.
- **Continue (c):** A rendszer folytassa a program végrehajtást normál sebességgel
- **Breakpoint-ok (Z0** packet**):**
  - A GDB alapból szoftver breakpointokat igyekszik rakni (**Z0** packet)
  - TRAP utasítással felülírja az adott memória címet.
  - **Z1** hardware-es breakpoint (limitált).

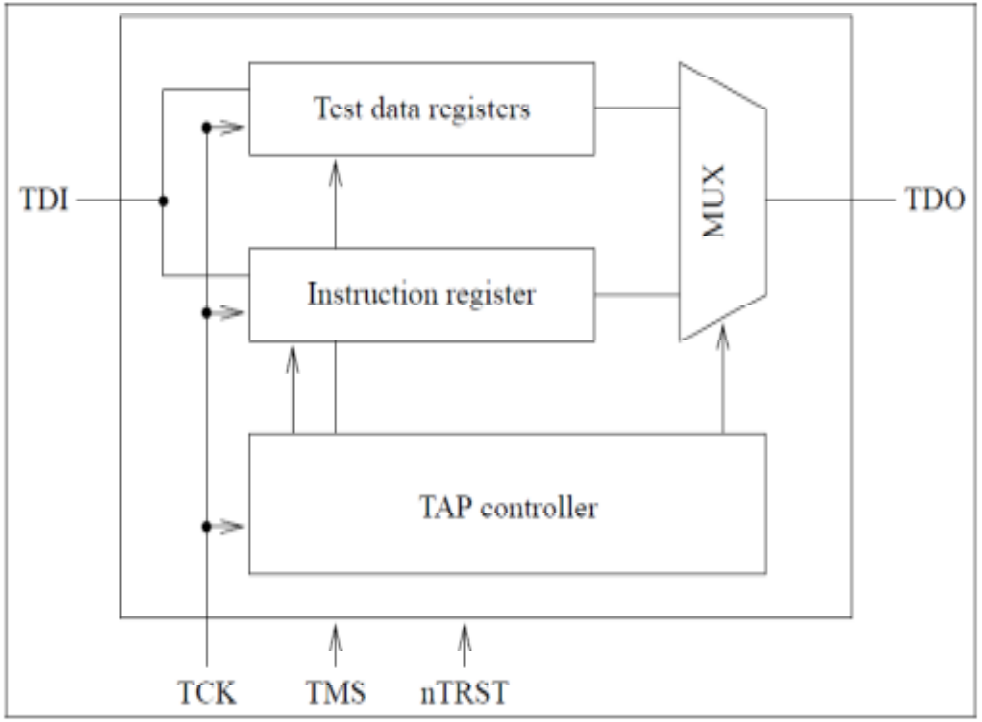
Letöltött monitor nélküli debuggolás

- Emulátor tulajdonság szükséges
  - Változók értékék megfigyelése
  - Változók értékék megváltoztatás
  - Step-elés
  - Breakpoint állítás

- JTAG-en keresztüli debuggolás
  - JTAG alapvetően csak egy kommunikációs séma
  - 5 vezetékes
    - TDI, TDO, TMS, TCK, nTRST

JTAG (Joint Test Access Group)

- 1985-ben alapították
- NYÁK és IC tesztelő szabványt



JTAG-en keresztül adható parancsok hagyományos ARM magok

- **INTEST:** Kiválasztja a boundary-scan register, és engedi abba az adatok ki be shiftelését.
- **IDCODE:** Az eszköz identifikációs regiszterét választja ki adat regiszterként.
- **BYPASS**: Gyakorlatilag az adott eszközt kikapcsolaja az adatáramlásból (1 bites bypass regiszter választódik ki, mint jelenlegi adatregiszter)
- A **SCAN_N** ARM specifikus utasítás ezzel lehet kiválasztani a következő boundary-scan láncok valamelyikét:
  - **0 (Boundary Scan)**
  - **1 (Core Debug)**
  - **2 (EmbeddedICE),**
  - **15 (CP15).**

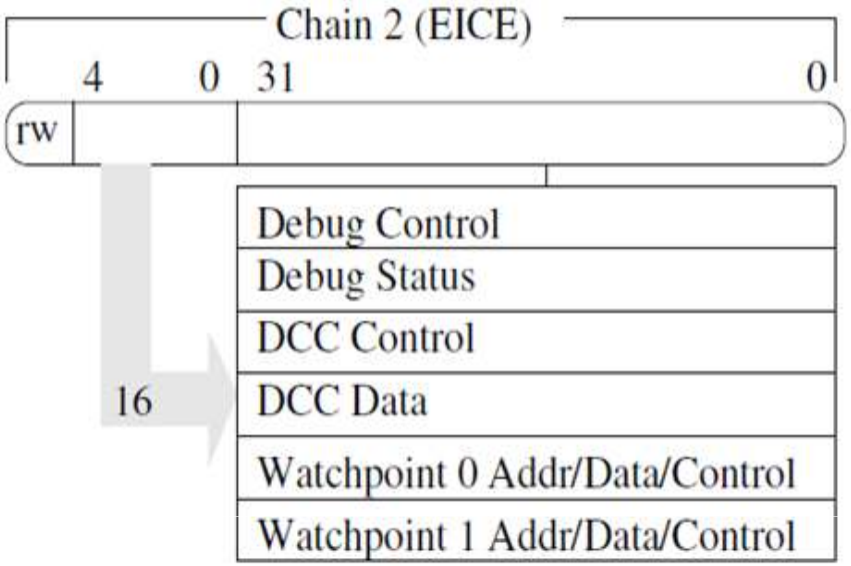
  A kiválasztás után az INTEST utasítással lehet hozzáférni ezekhez a chain-ekkel.

JTAG-en keresztül adható parancsok hagyományos ARM magok

- **1 Core Debug:**
  - rendszer buszra van kötve
  - utasítást és vagy adat lehet a rendszerbe be szkennelni, kiolvasni

  Debugg állapotban a debugger például le tudja kérdezni a rendszer belső állapotát a core regiszteteken keresztül úgy hogy egy “store multiple” utasítást (STM) kishiftel a scan chain 1-re (két NOP-al kiegészítve utána, hogy végigmenjen a pipelineon). Az eredmény kiolvasható a scan chain 1 ből.
- **2 EmbeddedICE:** tipikus debugger funkcionalítást
  - breakpoint-ok
  - watchpoint-ok
  - single stepping
  - 16 registere van, amelyek a scan chain 2-ön keresztül konfigurálhatóak.

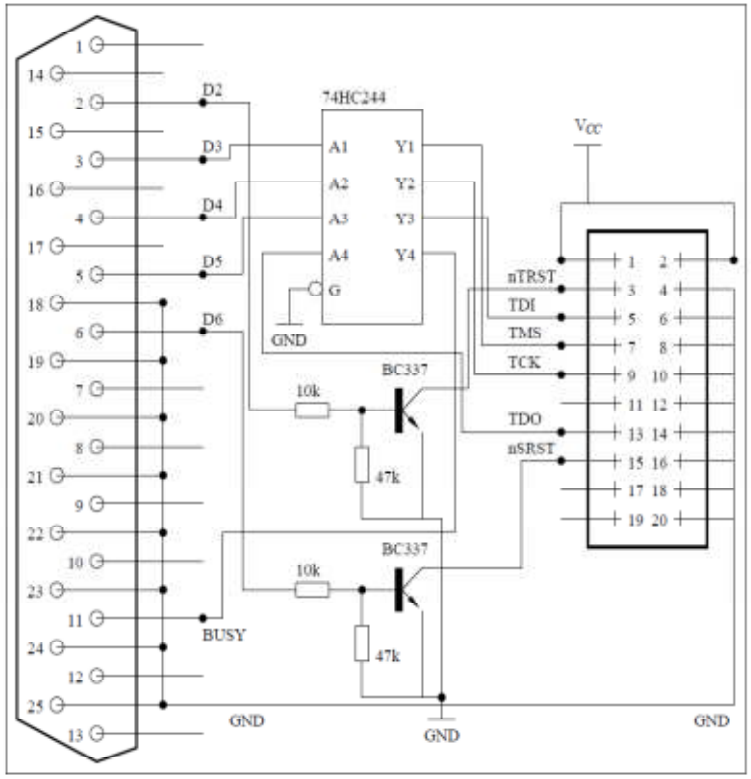
Embedded ICE



- **Debug control regiszter, debug status regiszter:** Debug módba lehet rakni a target-et és a debug státuszt kiolvasni. Mindkét watchpoint egység lehet annak az eseménynek a forrása, ami kiváltja a debug állapotba menetelt.
- **Watchpoint 0, 1:** Minkét watchpoint 6-6 regisztert tartalmaz (address value és mask, data value és mask, control value és mask)., ezek használatával elérhető, hogy a watchpointok: breakpoint-ot (stop at instruction), watchpoint-ot (stop at data access) tartalmazhat.

JTAG eszközök

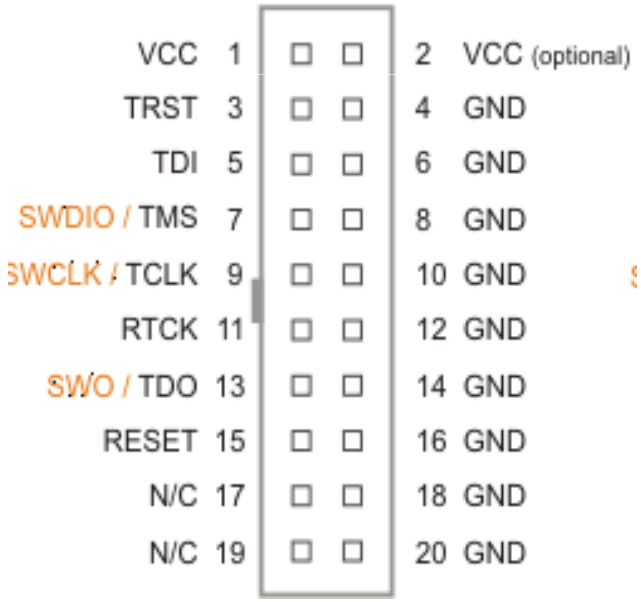
- **Macraigor Wiggler, Parallel Port Wigglers**
  - http://www.macraigor.com/wiggler.htm
  - ~500KHz



JTAG eszközök

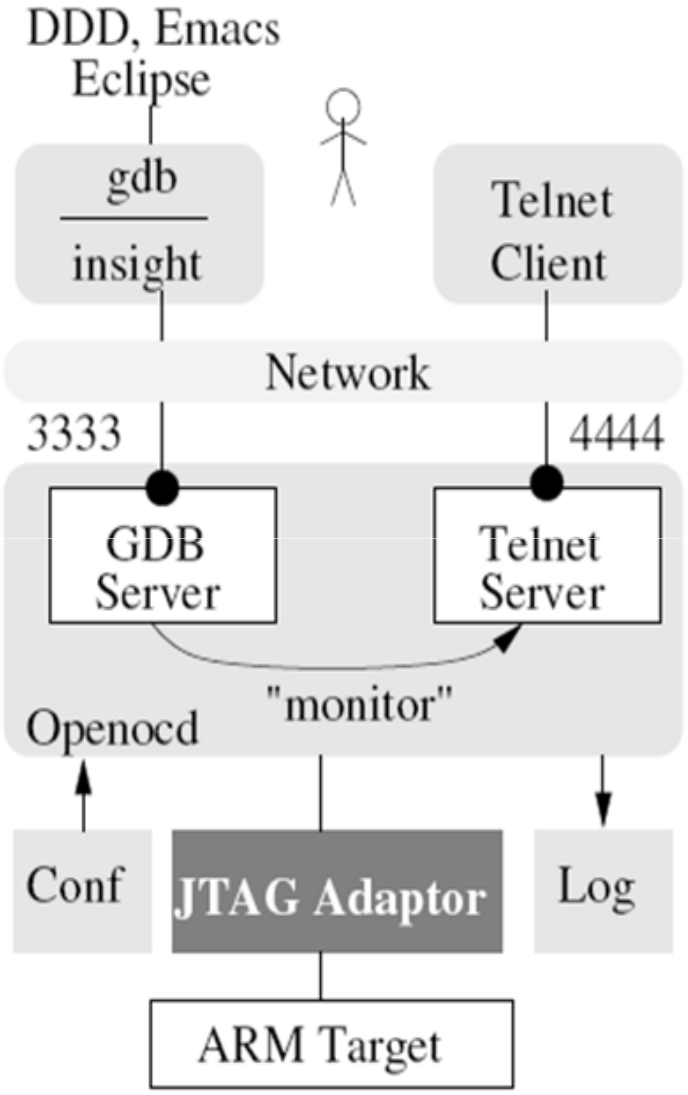
- **FT2232(H)**
  - USB alapú JTAG megoldás
  - Az összes újabb „olcsó” JTAG ezt használja
  - 30 MHz-es JTAG clock

20-PIN JTAG/SW Interface



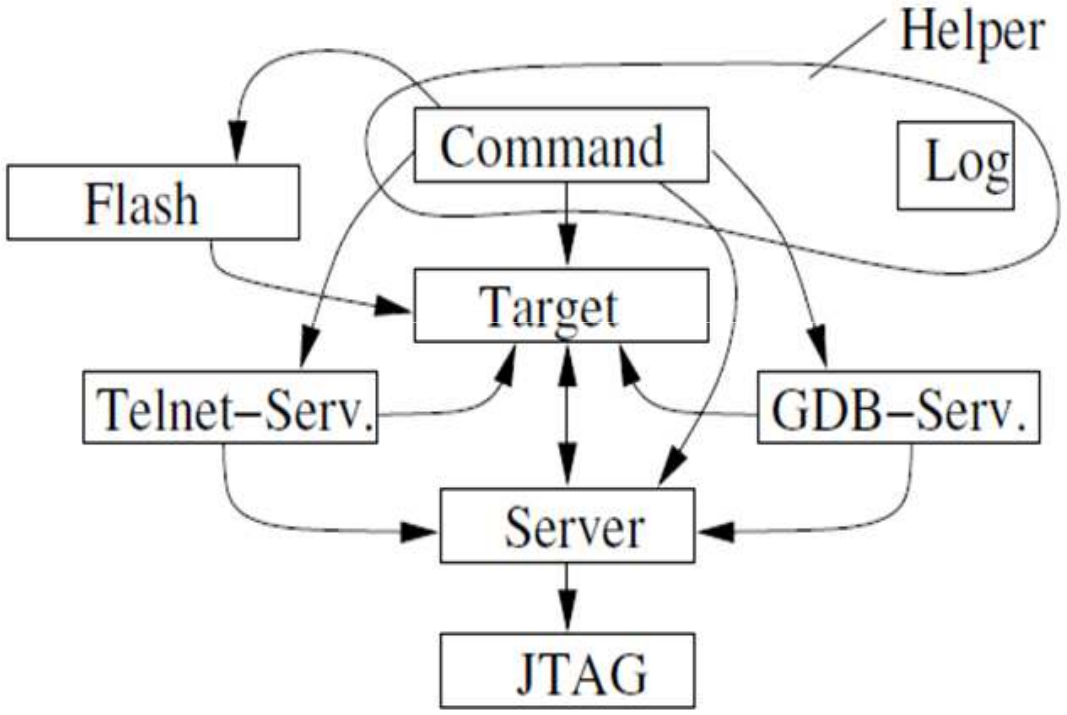
JTAG, GDB csatolás

- ***Open OCD***
  - GDB server
  - JTAG támogatással
  - RSP parancsok konverziója JTAG parancsokká
  - Külön konfigurációs port
    - Server handling
    - Target management
    - JTAG memory management
    - Flash memory management



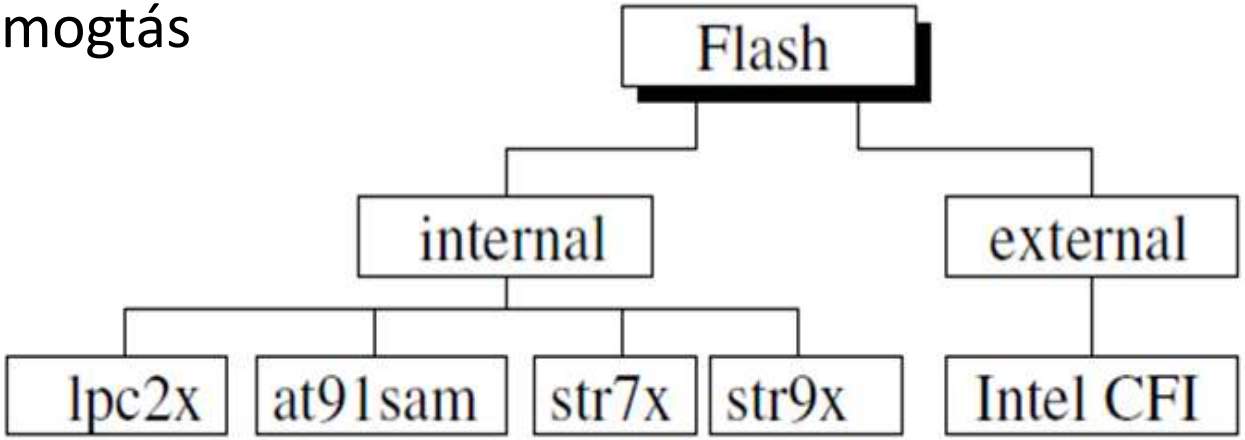
Az OpenOCD belső felépítése

- **Server modul**
  - Telnet parancsok
  - GDB-Server
- **Command modul**
  - Parancsok regisztrációja, kezelése
- **Target modul**
  - Minden target core variánshoz külön c file
- **Jtag modul**
  - Jtag kezelés
- **Flash modul**
  - Hardware függő Flash támogtás



Az OpenOCD fordítás függő részei

- **Target modul**
  - Minden target core variánshoz külön c file
- **Flash modul**
  - Hardware függő Flash támogtás



Az OpenOCD konfiguráció

- **Konfigurációs rész minden target-hez**
  - Server (daemon) konfiguráció
  - JTAG konfiguráció
  - JTAG scan chain konfiguráció
  - Target konfiguráció
  - Flash konfiguráció

Példa egy STM32 konfiguráció-ra

Parancs:

openocd -f olimex-jtag-tiny.cfg -f stm32.cfg -f stm32_gdb.cfg

Server és JTAG konfiguráció

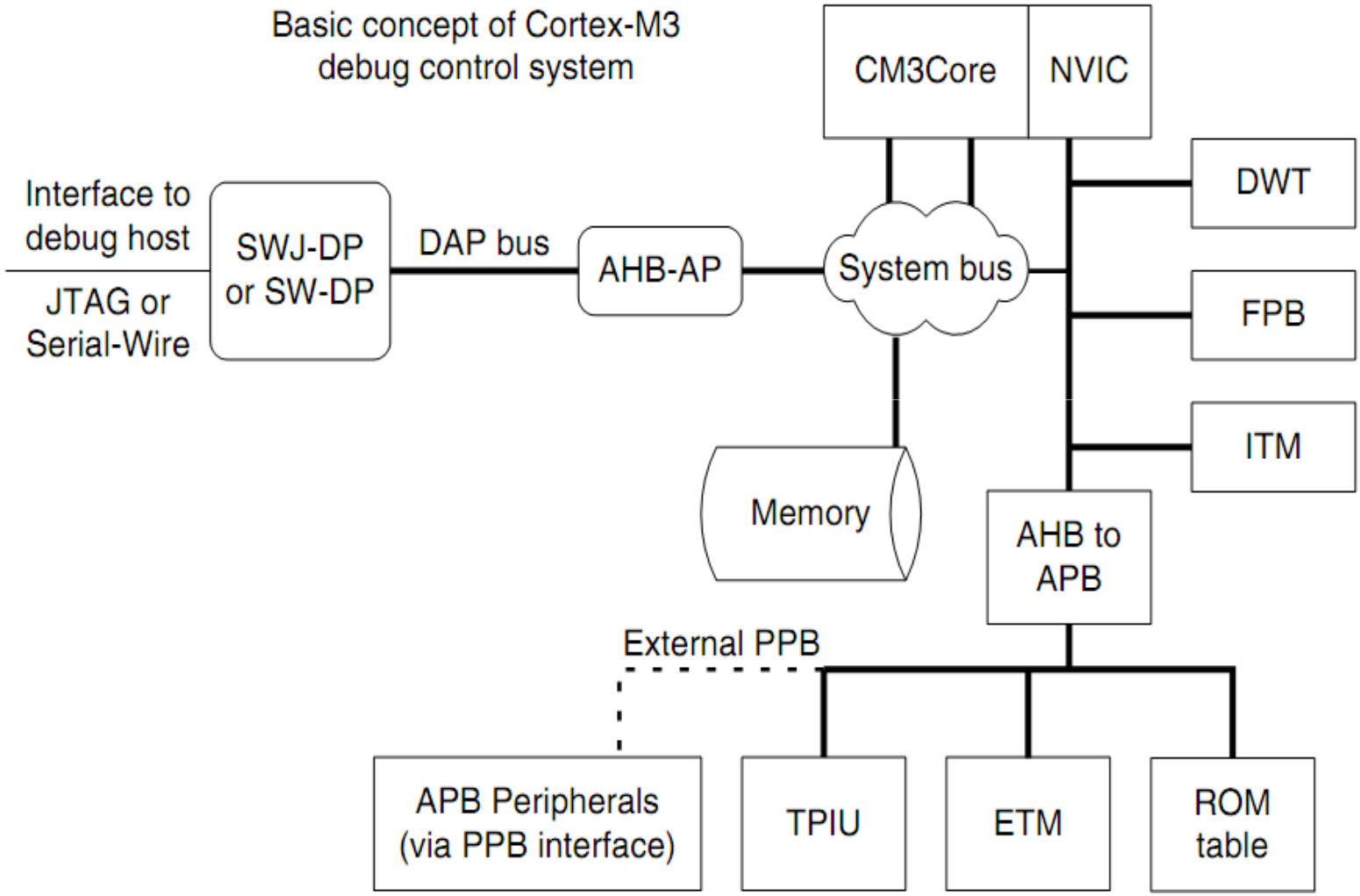
stm32_gdb.cfg

```
# default ports
telnet_port 4444
gdb_port 3333
tcl_port 6666
init
jtag_khz 565
reset init
verify_ircapture disable
```

olimex-jtag-tiny.cfg

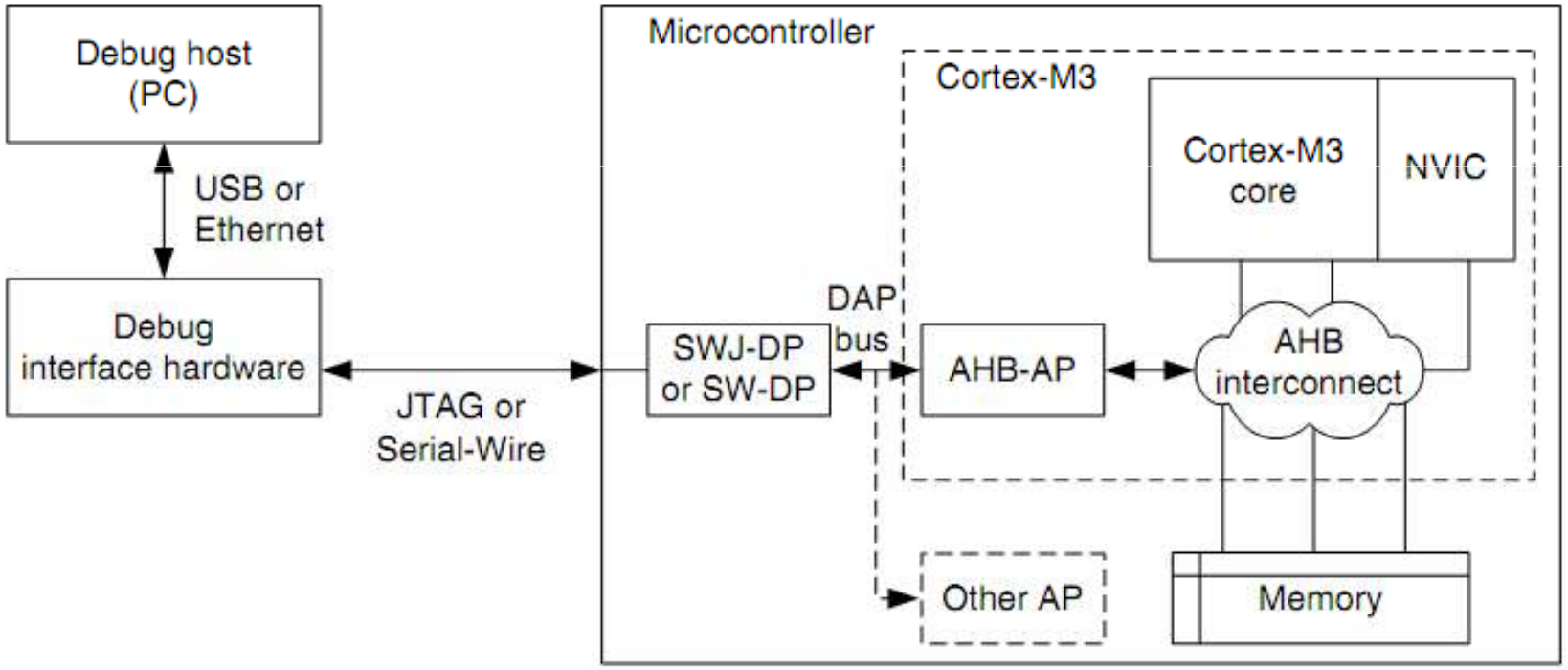
```
interface ft2232
ft2232_device_desc "Olimex OpenOCD JTAG
TINY"
ft2232_layout olimex-jtag
ft2232_vid_pid 0x15ba 0x0004
```

Cortex M3: Coresight debug rendszer



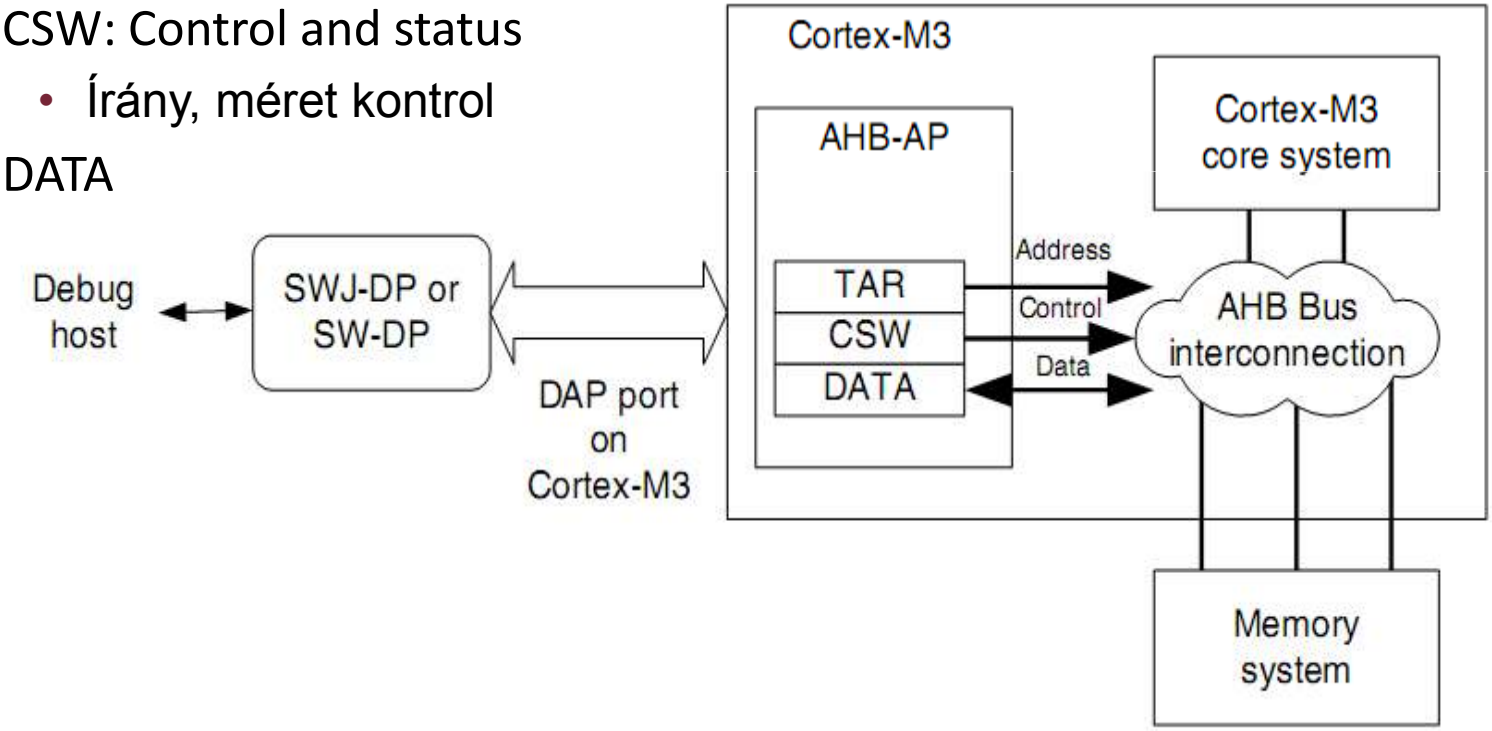
Debug Port csatlakozás

- SWD, vagy JTAG csatlakozás



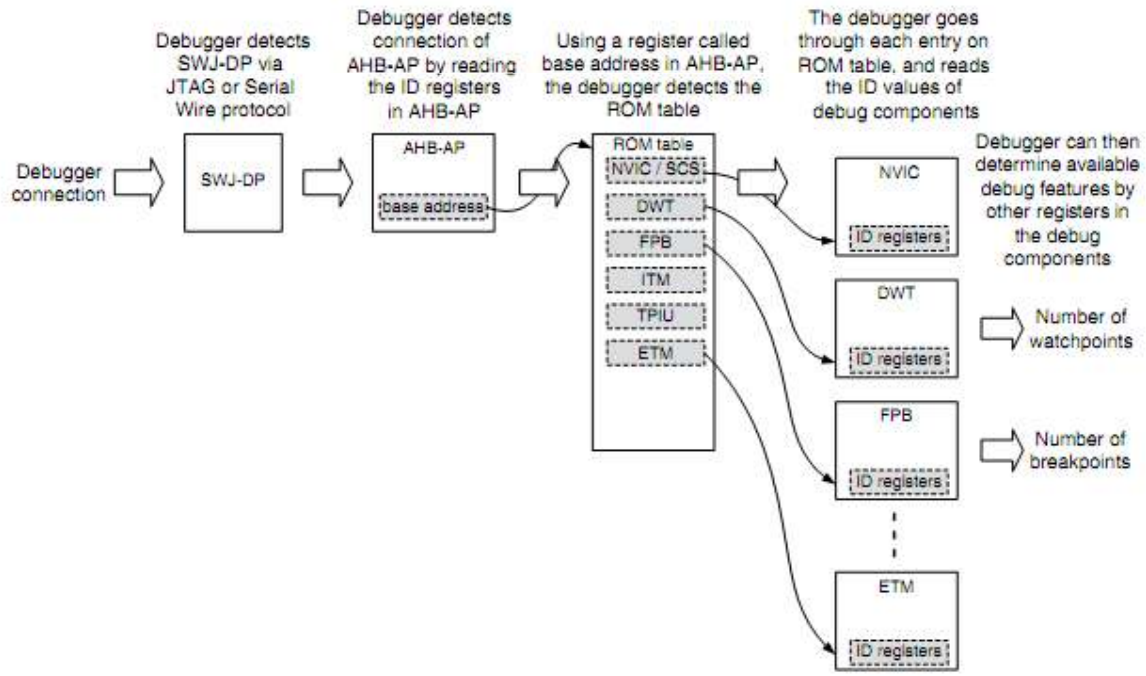
AHB-AP: Advanced High-Performance Bus Acces Port

- Bridge a debug portok és a Cortex M3 rendszer között
  - TAR: Transfer Address
    - Átviteli cím kontrol
  - CSW: Control and status
    - Írány, méret kontrol
  - DATA



Debug feltérképezési folyamat

- Van egy ún. ROM table ami tartalmazza a komponensek címeit.
  - Alapból egységes komponensek, de az új sorozatokban bővíthették őket



Debug Módok

- C_DEBUGEN bit a Debug Halting Control and Status register-ben, hogy debug módba kerüljön a processzor
  - Csak a DAP-on keresztül állítható (külső eszköz)
  - A Halt-ba rakás is ez a regiszter, de az már SW-ből is állítható
- 1. Halt mód
  - Utasítás végrehajtás leállítódik
  - A System Tick Timer (SYSTICK) counter leáll
  - Stepp utasítások
  - Interuptok felfüggesztődnek, és a léptetés alatt vagy végrehajtónak, vagy lemaszkolhatóak

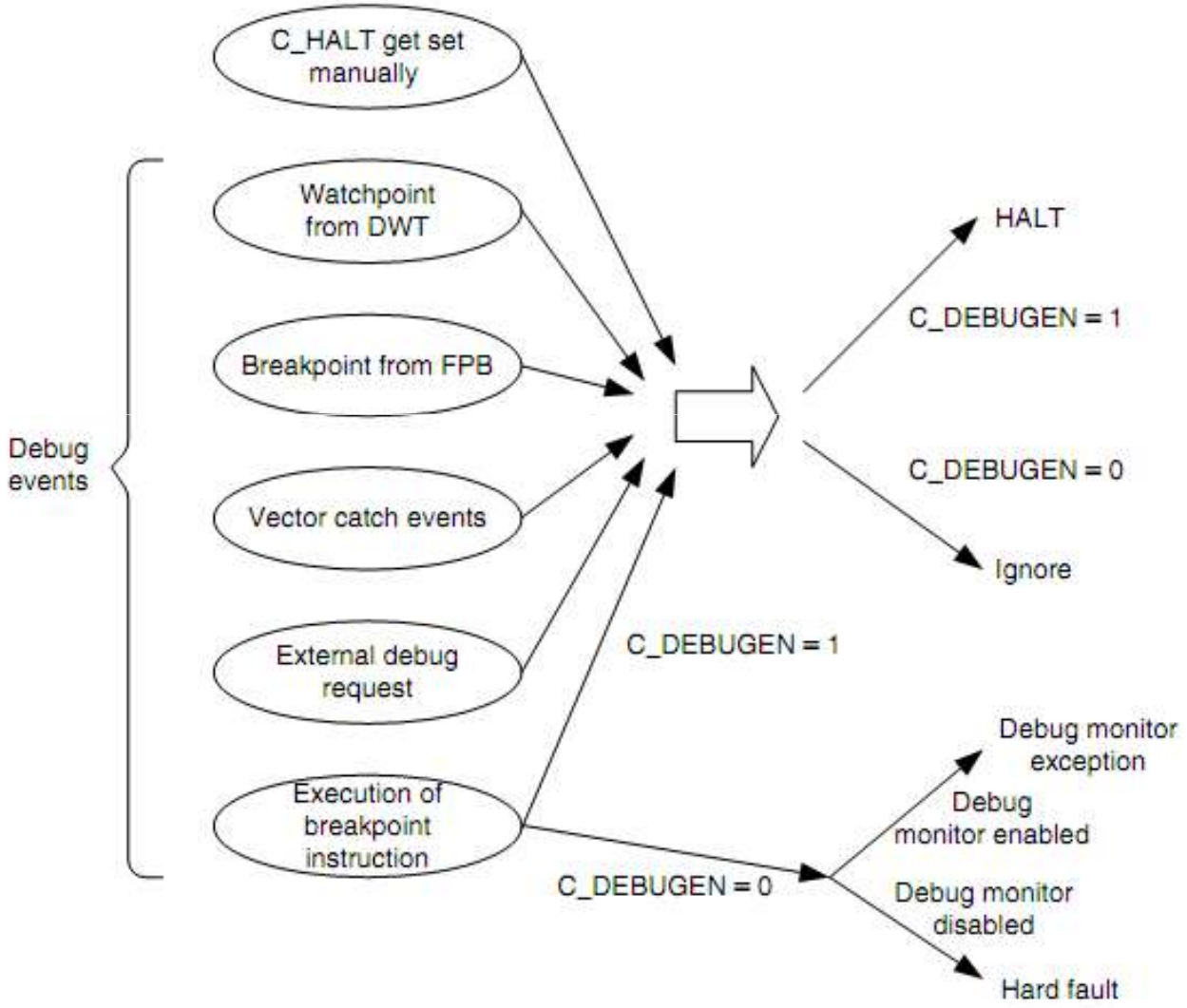
Debug Módok

- 2. Debug monitor mode
  - A processzor végrehajtja a 12. kivételt (debug monitor)
  - A SYSTICK counter tovább fut
  - Az új bejövő interruptok vagy megszakítják, vagy nem a debug monitort annak priorításának függvényében
  - Debug esemény elveszhet, ha magasabb priorítású IT hajtódik végre
  - Támogatja a single-step működést
  - Memória tartalom megváltoztatható a debug monitor handler-en keresztül

Debug Módok

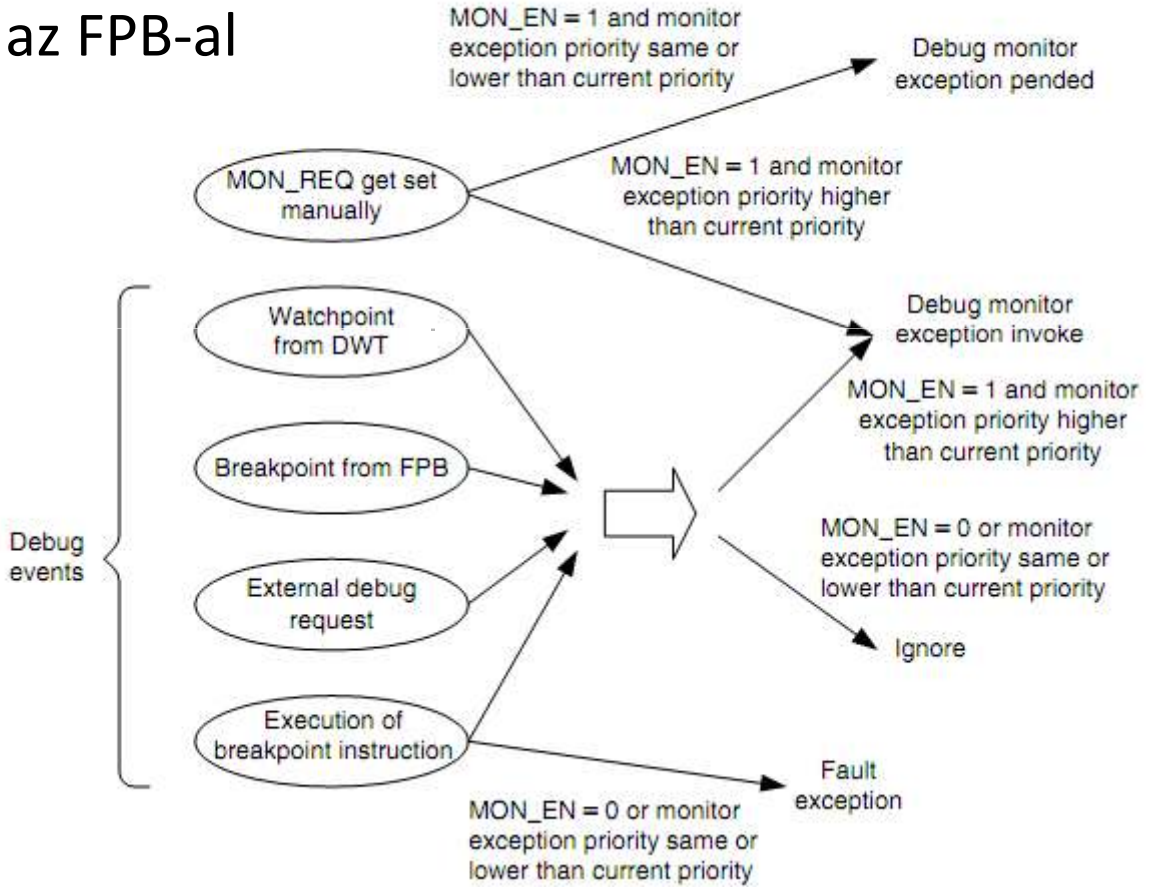
- 2. Debug monitor mode
  - A processzor végrehajtja a 12. kivételt (debug monitor)
  - A SYSTICK counter tovább fut
  - Az új bejövő interruptok vagy megszakítják, vagy nem a debug monitort annak priorításának függvényében
  - Debug esemény elveszhet, ha magasabb priorítású IT hajtódik végre
  - Támogatja a single-step működést
  - Memória tartalom megváltoztatható a debug monitor handler-en keresztül
- Azokhoz a rendszerekhez, ahol nem engedhető meg a teljes rendszer leállítása (motor control, disk vezérlés)
  - Debugger tudja a thread szintet és az alacsony priorítású IT-ket figyelni, a magas priorítású IT meg működik zavartalanul.

Debug módba lépés



Break Point-ok

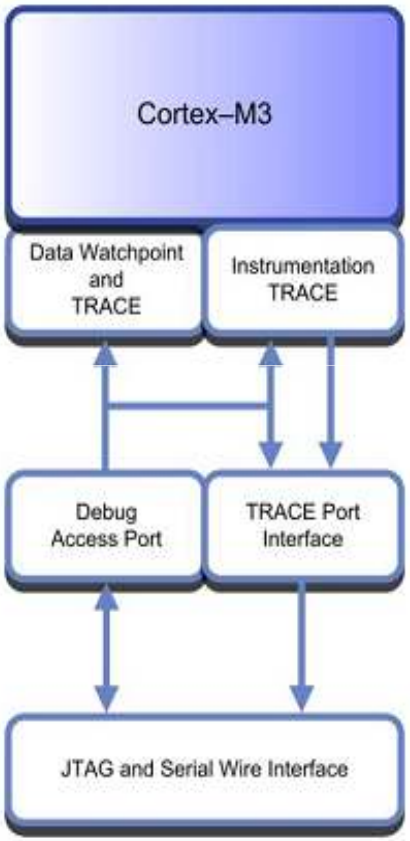
- Breakpoint utasítás (tipikusan RAM-ból)
- Címkomparálás az FPB-al

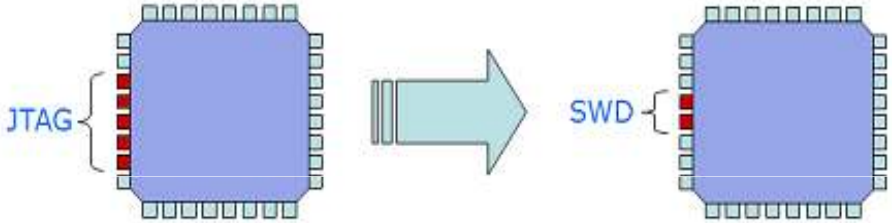


FPB: Flash Patch and Breakpoint Unit

- Hardware-es breakpointok generálása
  - 8 komparátor
    - 6 program cím
    - 2 literál
- Flash Patch feature
  - Módosítások a nem módosítható Flash kódon
  - 2 literál komparátor
  - Flash területeket remappel az SRAM-ba
  - Számunkra nem lényeges, csak ROM alapú eszközöknél

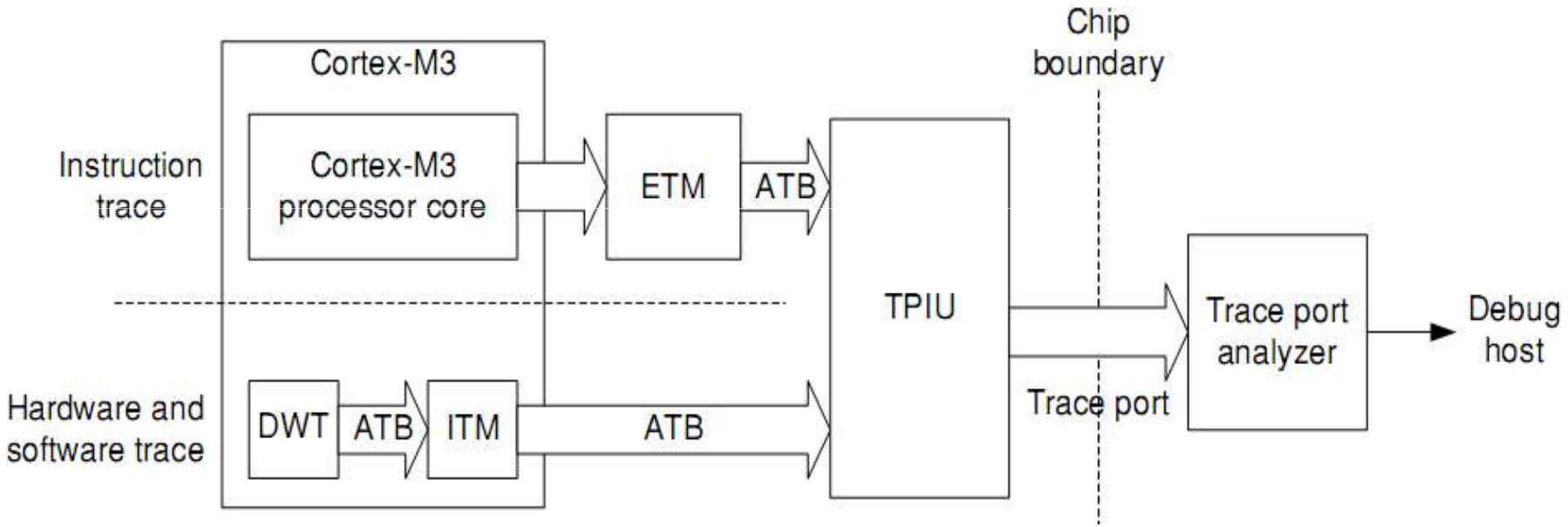
Debug portok, Trace portok





The Cortex CoreSight debug system uses a JTAG or serial wire interface. CoreSight provides run control and trace functions. It has the additional advantage that it can be kept running while the STM32 is in a low power mode. This is a big step on from standard JTAG debugging.

Coresight trace rendszer

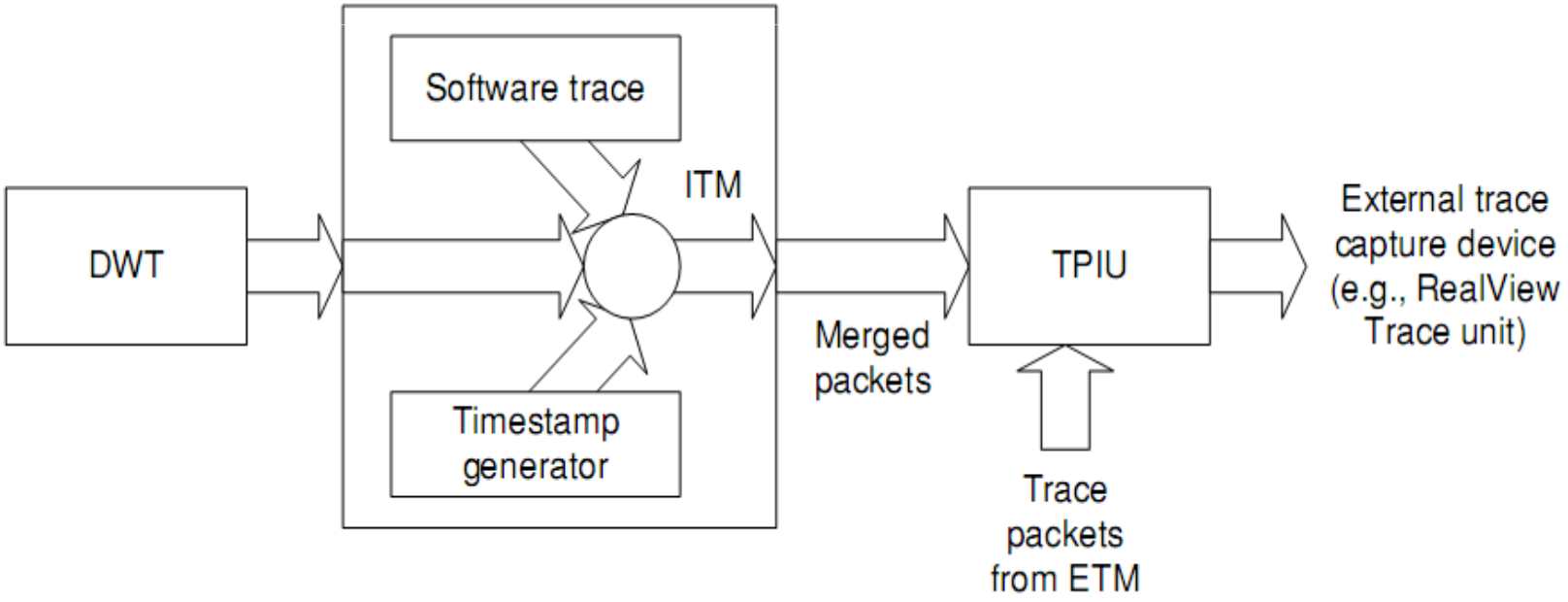


DWT: Data Watchpoint Trace

- 4 komparátor: data address / program counter
  (az első konfigurálható clock cycles counterre is)
  Van hozzá MASK is
  - Hardware watchpoint: processzor debug módba
  - ETM trigger: trace csomag küldés indítás
  - PC mintavételező trigger
  - Data address sample trigger
- Számlálók
  - Órajel számláló
  - Sleep ciklusokat számláló
  - Interrupt overhead számláló
- PC mintavételezés időközönként
- Interrupt trace

ITM: Instrumentation Trace Macrocell

- Direkt konzol üzenetek (printf)
- DWT tud üzeneteket generálni
- Időbélyeg generálás a debuger számára

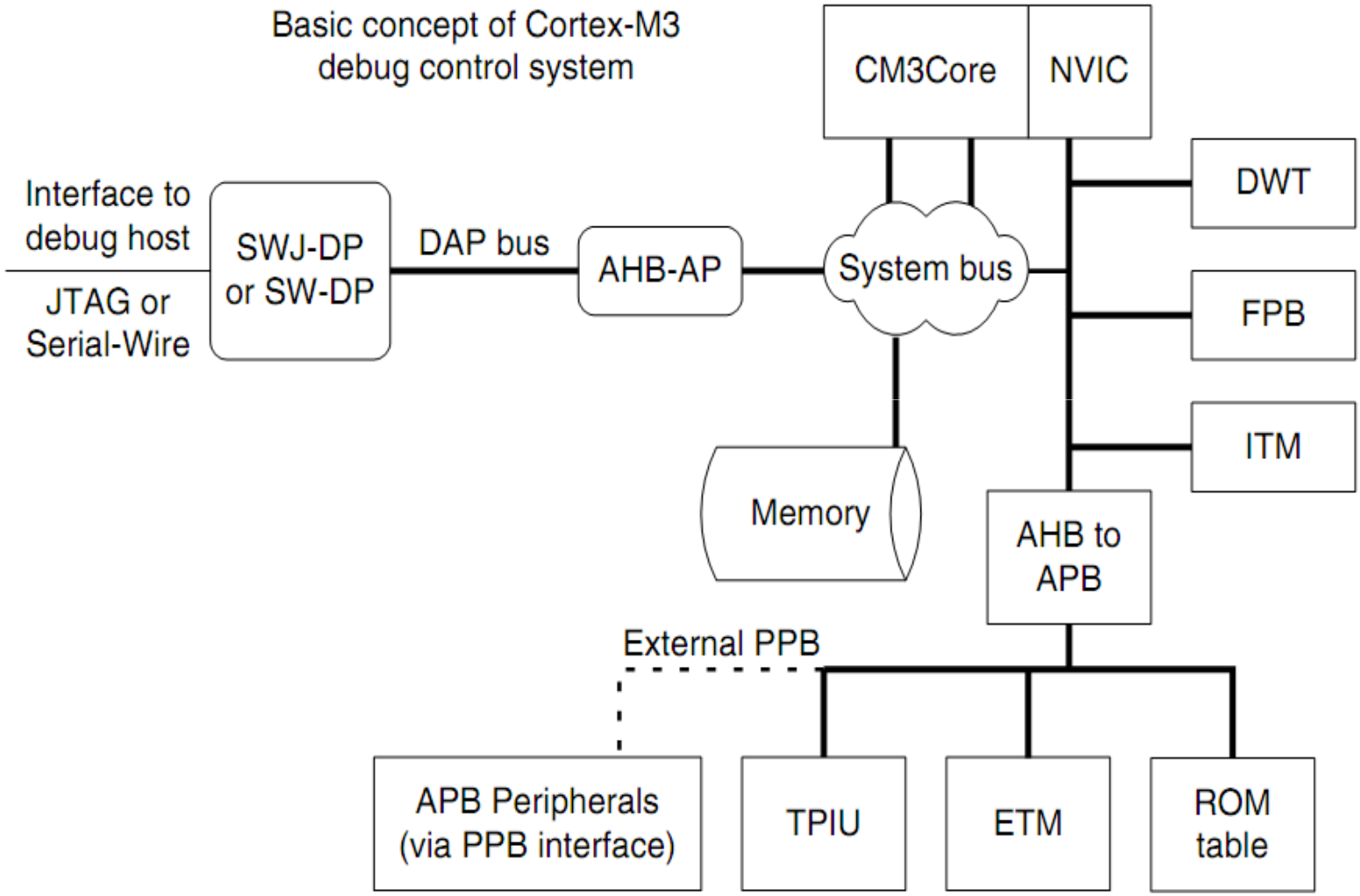


ETM: Embedded Trace Macrocell

- Már az ARM7-eseknél volt
- Utasítás végrehajtás követés
- DWT-et használja komparátorként

- Kivételek nyomon követése
- Utasítás nyomon követés
  - Gyakorlatilag az összes végrehajtott utasítás nyomon követhető
    - Debuggernek meg kell hogy legyen a bináris kód

Cortex M3: Coresight debug rendszer



TPIU: Trace Port Interface Unit

- 4 bites szinkron mód
  - Elég csúnya formátum
- 1 bites UART szerű aszinkron mód