ARM Cortex Core microcontrollers

8th Debugging

Balázs Scherer

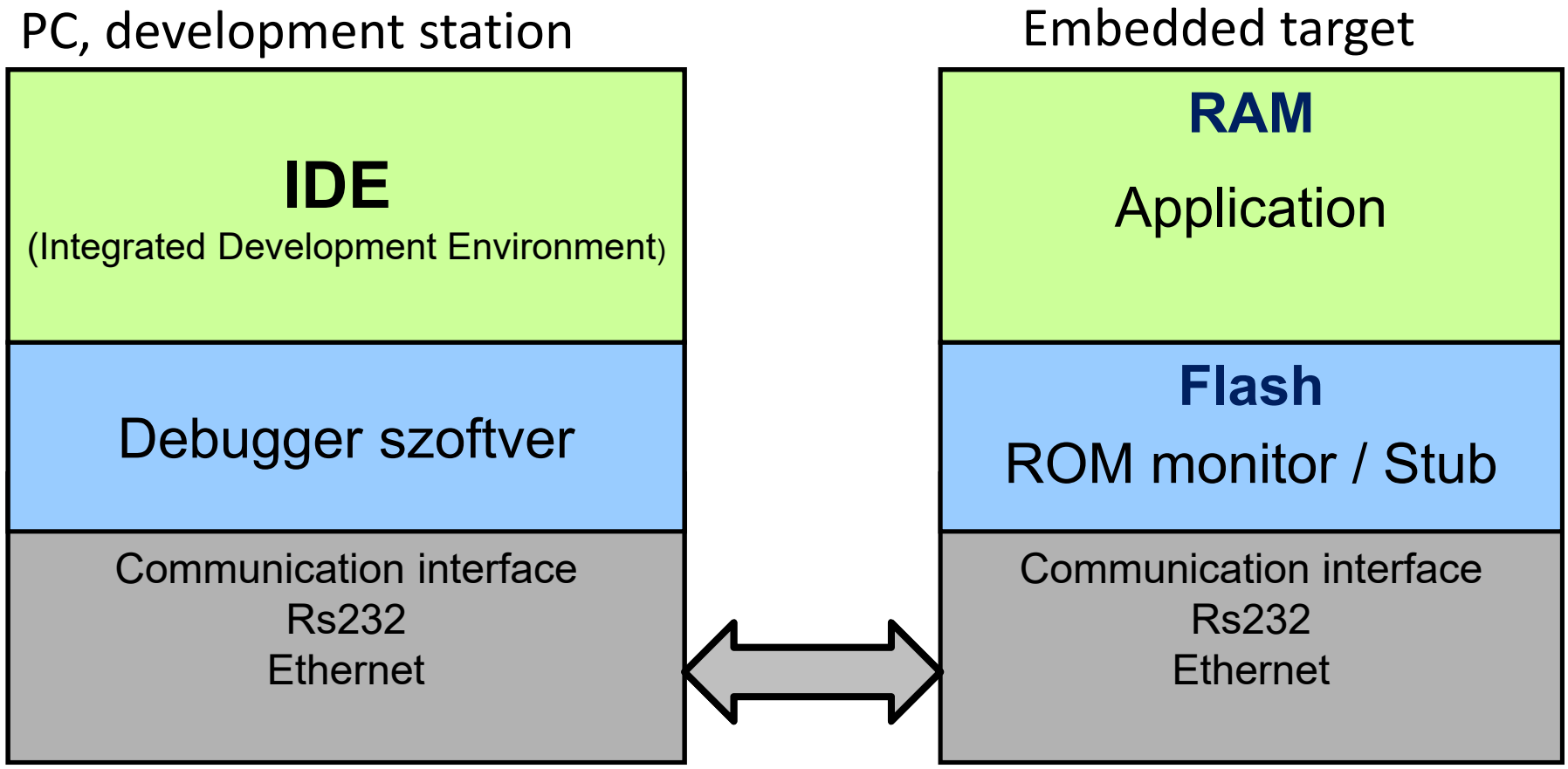
Budapest University of Technology and Economics
Department of Measurement and Information Systems

Traditional debug methods

- Measurement based
  - Oscilloscope
  - Logic analyzer
  - Printf
  - LEDs
- In-Circuit Emulators
  - Debug variant special microcontrollers
  - Dual port RAM
    - Hardware based break points
    - Register reading, writing

Debug methods of early 32-bit microcontrollers

- ROM monitors: GDB stub



ROM monitor

Rom monitor

- Only the RAM area can be used by the application
  - Early 32-bit microprocessors used mainly external RAMs
- Communication between the ROM monitor and the debugger is standardized in GNU environment
  - GDB: Remote Serial Protocol

Debugger software

GDB: GNU Debugger I.

- GDB: command line debugger, many IDE use it
  - Eclipse
  - DDD

Example 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 commands

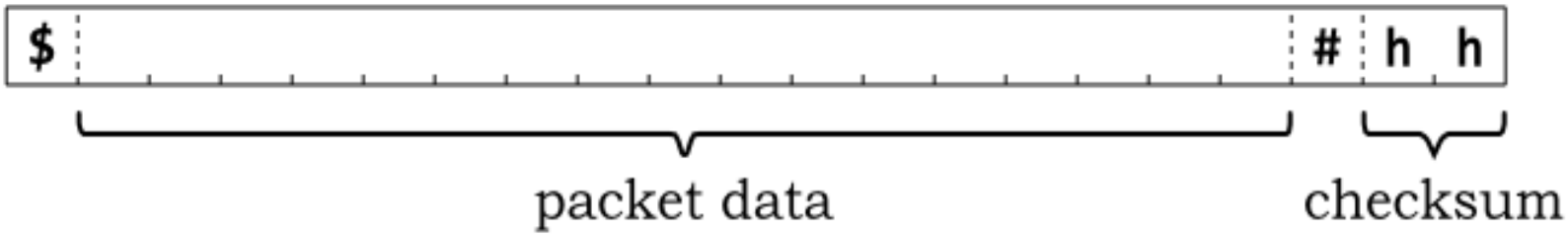
- ***run:*** Program running
- ***continue:*** continuing the program
- ***next:*** next instruction
- ***step:*** step-into
- ***list:*** listing the source code
- ***break:*** setting the break point (can be hw or sw breakpointok)
  - ***disable/ enable***
- ***print:*** displaying a value of a variable
- ***set:*** setting a value of the variable

And many more …

Debugger, ROM monitor connection

GDB: Remote serial protocol

- Connection to the target
  - Rs232
  - TCP protocol
- ASCII characters based commands
  - Start with **$** symbol
  - End with **#** and 8-bit cheksum
- The stub answares
  - Postive answare with **+** negative response with **–**
  - “OK” or Error code is the response for commands



Frequently used RSP commands

- **Read Registers (g):** read the register's of targer
- **Write Register n (P):** write a value to a register
- **Read Memory (m):** read from a memory address
- **Write Memory (M):** write to a memory address
- **step (s):** Step one instruction. Only answare is a +
- **Continue (c):** Continue the program execution
- **Breakpoint-ok (Z0** packet**):**
  - GDP usually tries to set a SW breakpoint (**Z0** packet)
  - **Z1** hardware breakpoint (limited numbers).

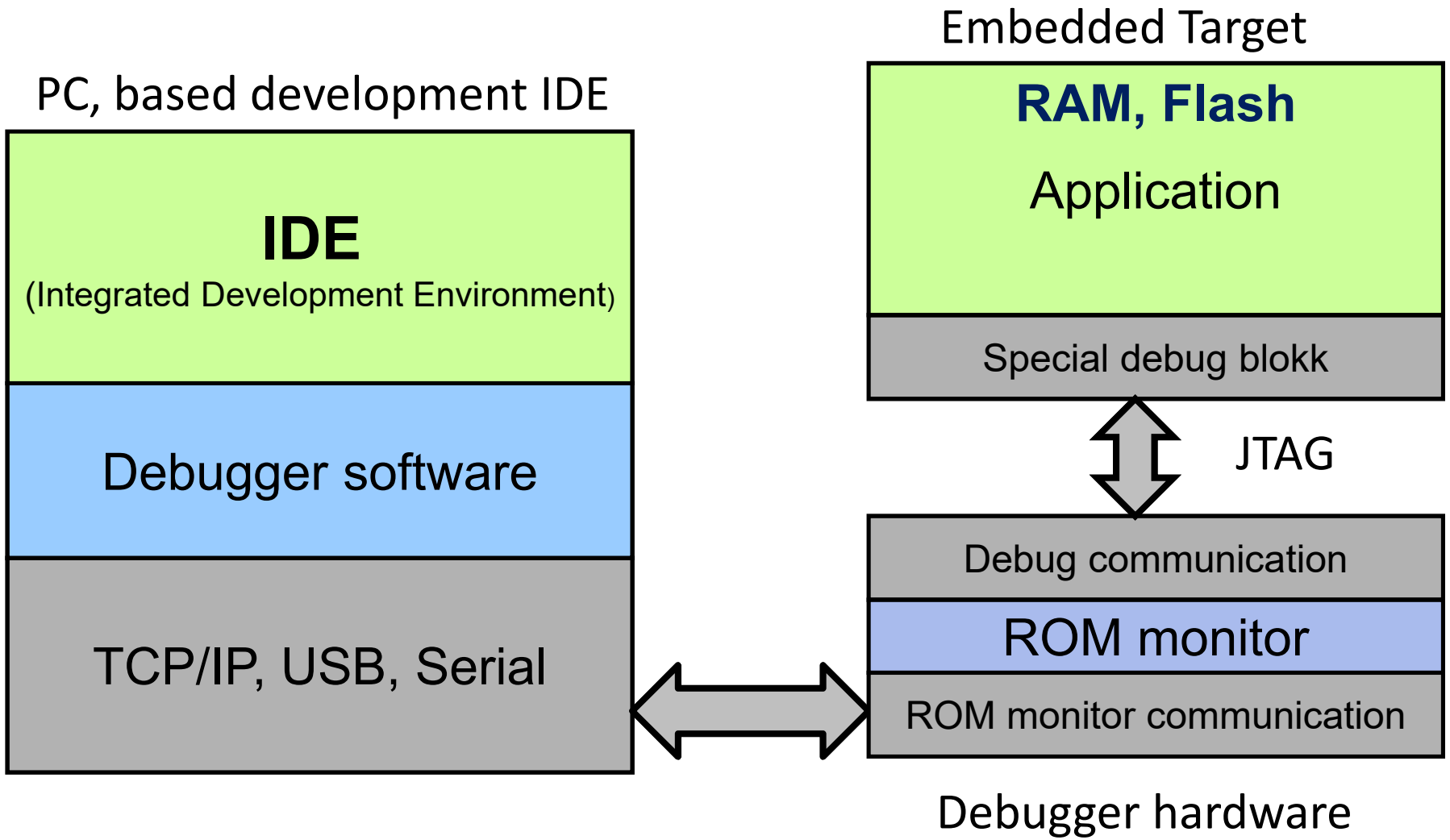
Degubbing support without ROM monitor

- Emulator properties are needed
  - Reading variable values
  - Setting variable values
  - Instruction stepping
  - Breakpoint setting

- JTAG based debugging
  - JTAG is just a low level communication protocol
  - 5 wires
    - TDI, TDO, TMS, TCK, nTRST

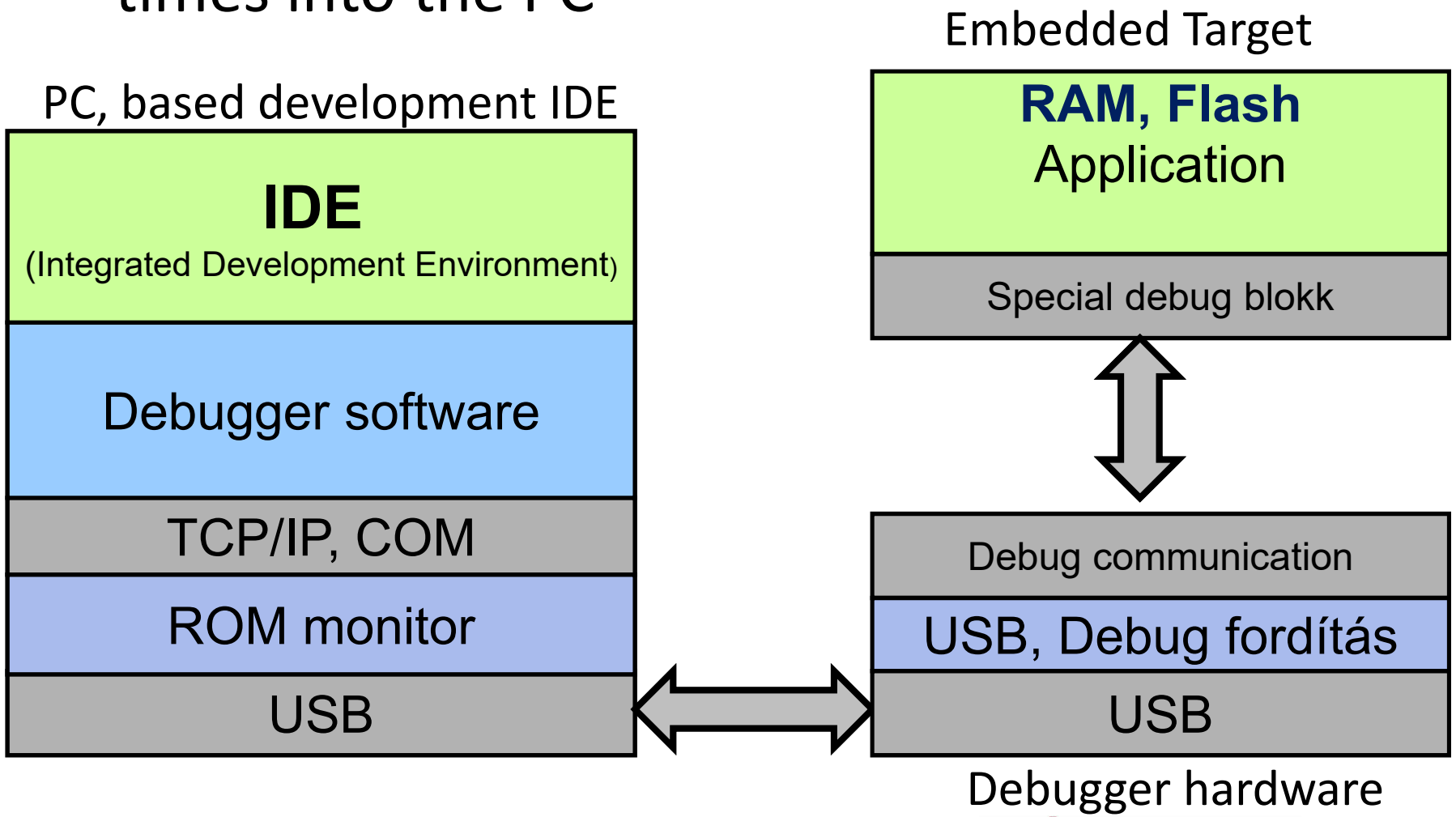
JTAG based debugging

- ROM monitor function is re allocated



JTAG based debugging

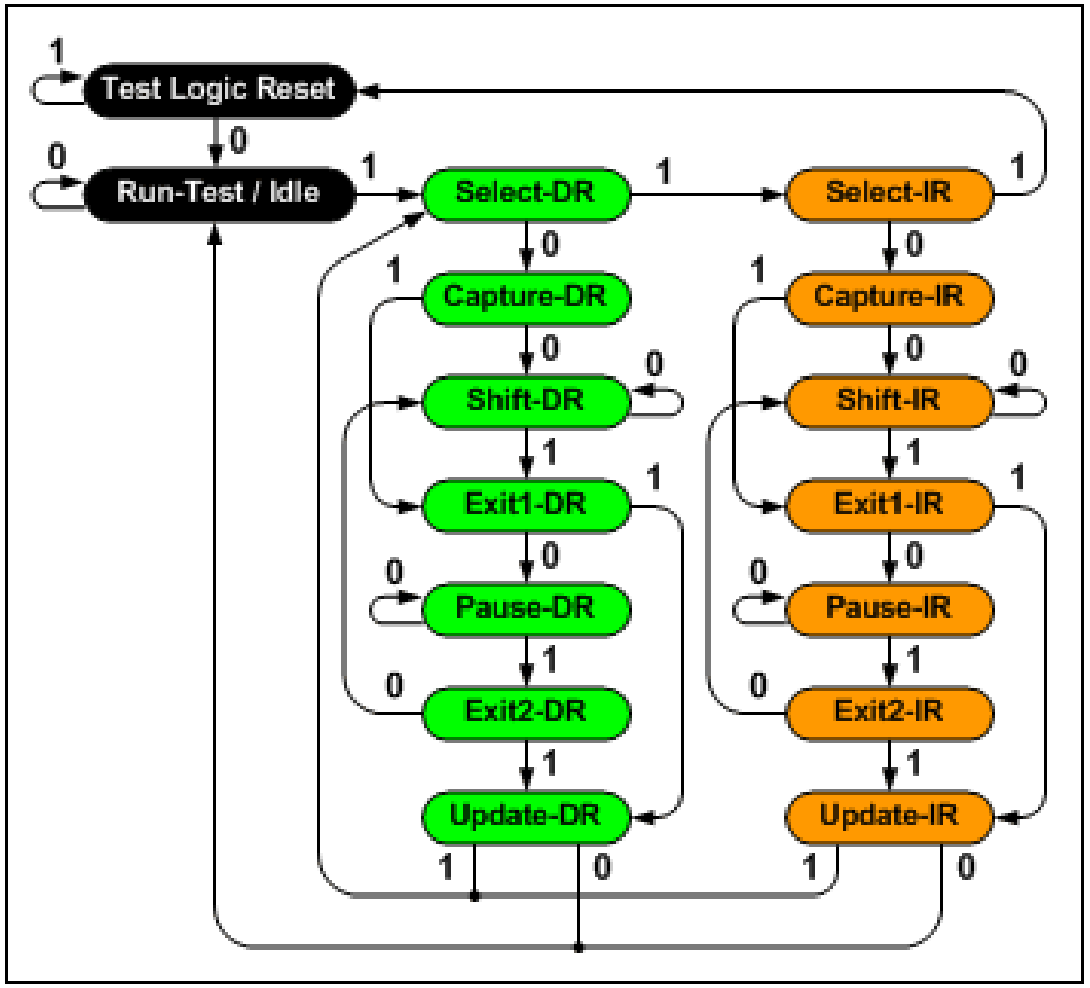
- ROM monitor function is re allocated many times into the PC



Debugger communication

JTAG (Joint Test Access Group)

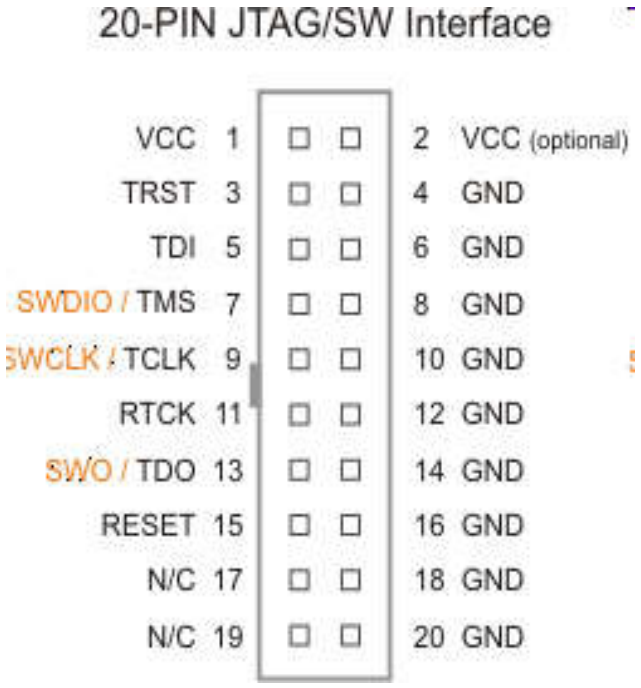
- State machine based communication



Debugger hardware

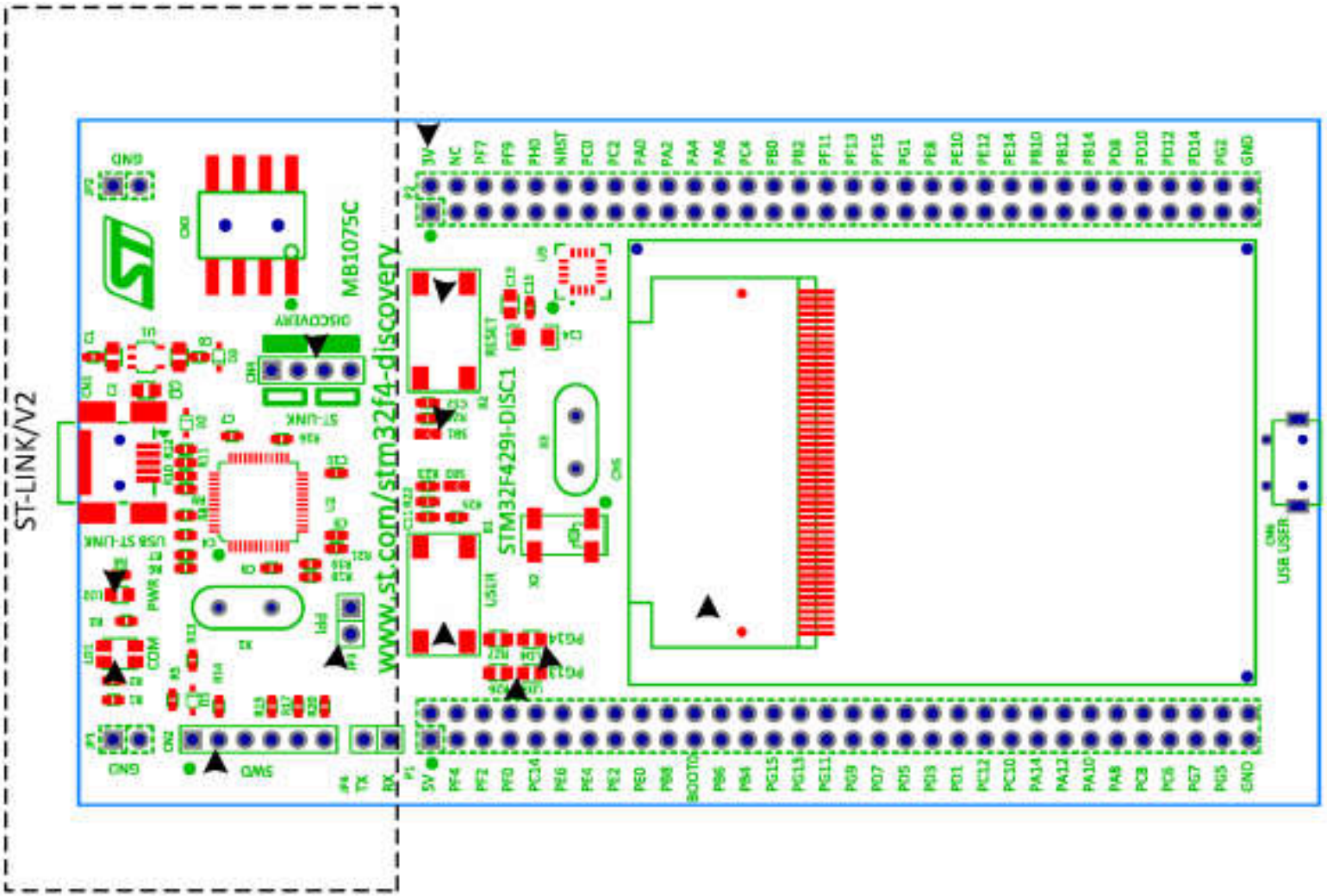
JTAG tools

- **FT2232(H)**
  - Used by many cheao USB based JTAG tools
  - Max. 30 MHz JTAG clock



Board Controllers

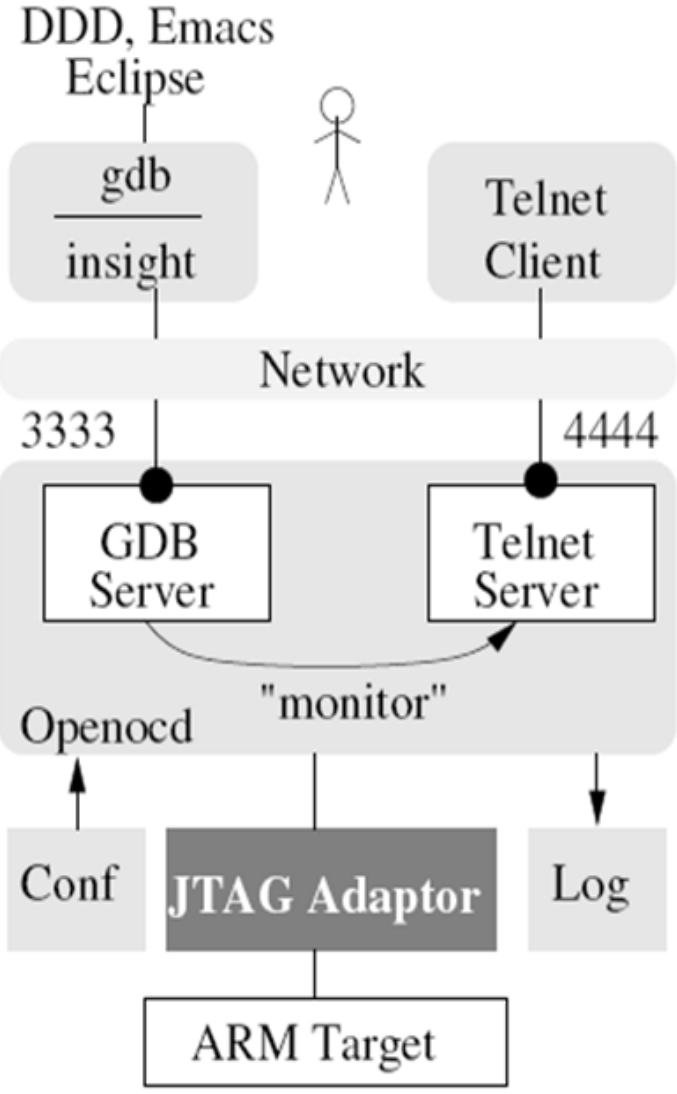
- **Many development tools use this method**



Integrated Debug monitor

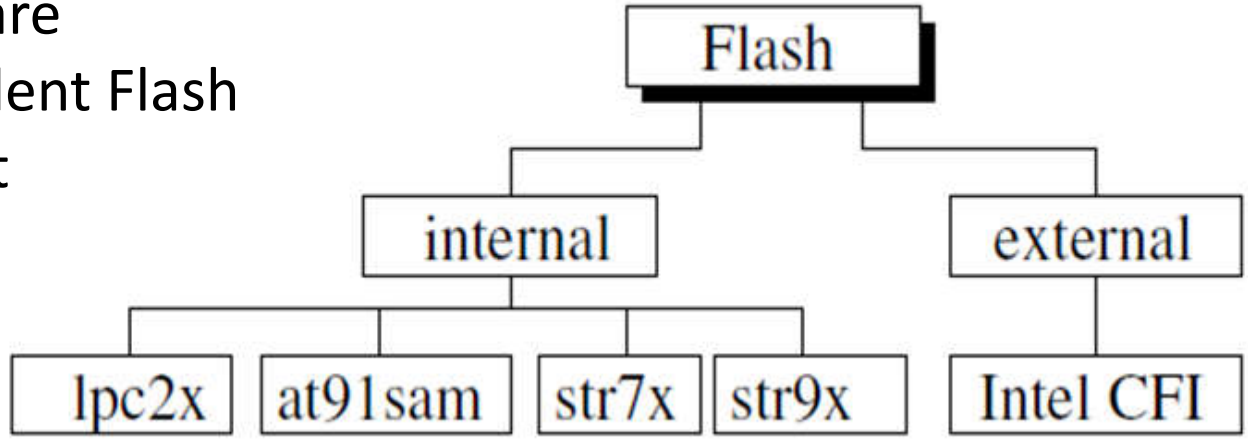
JTAG, GDB adapter

- ***Open OCD***
  - GDB server
  - JTAG, SWD support
  - RSP commands conversion to JTAG or SWD commands
  - Separate port for configuration
    - Server handling
    - Target management
    - JTAG memory management
    - Flash memory management



Target dependent modules of OpenOCD

- **Target modul**
  - Separate core files for targer core architectures
- **Flash modul**
  - Hardware dependent Flash support



Configurating the OpenOCD

- **Configurating the OpenOCD**
  - Server (daemon) configuration
  - JTAG configuration
  - JTAG scan chain configuration
  - Target configuration
  - Flash configuration

Example of an STM32 configuration

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

Example of Server and JTAG configuration

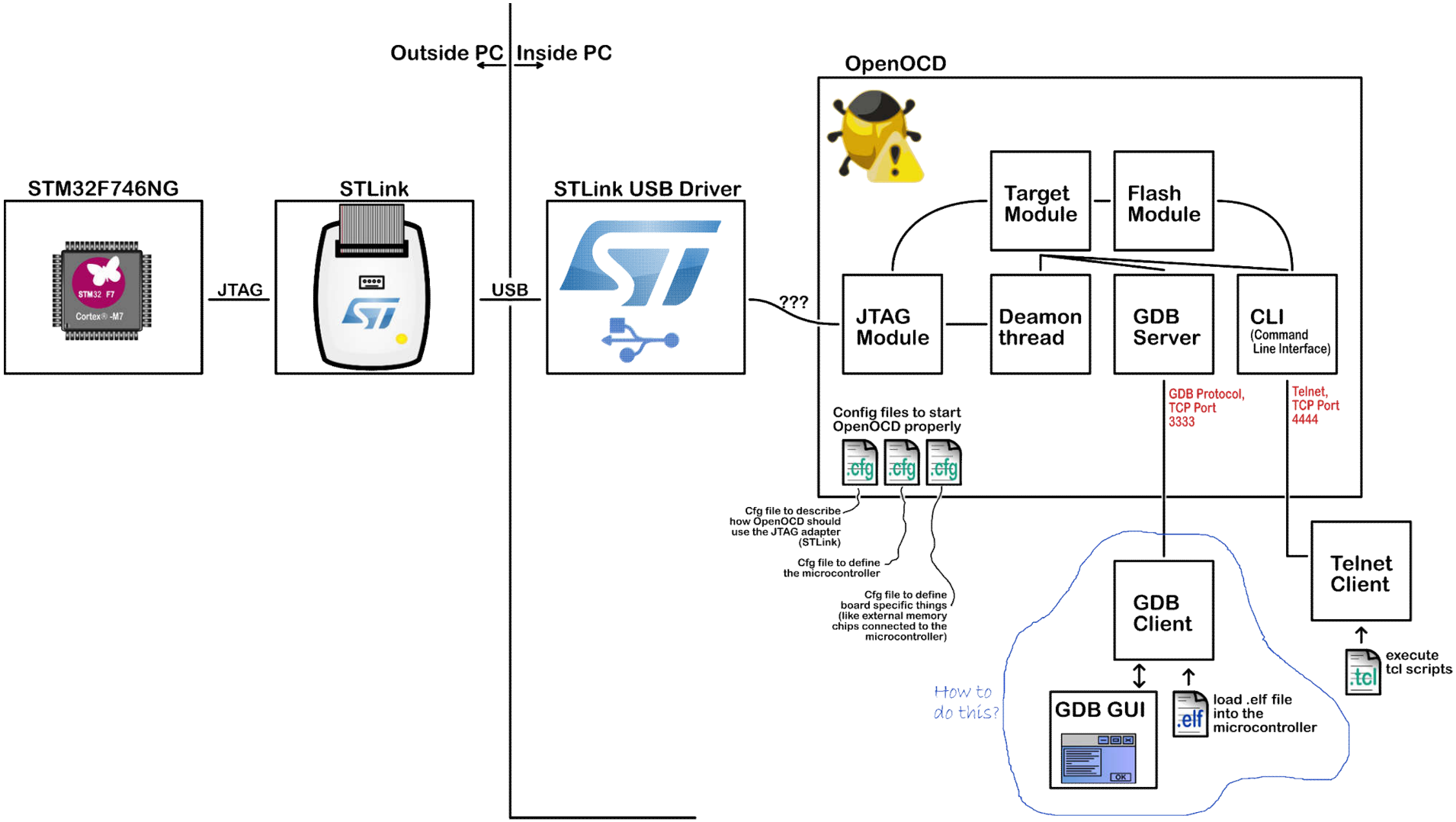
stm32_gdb.cfg

```
# 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
```

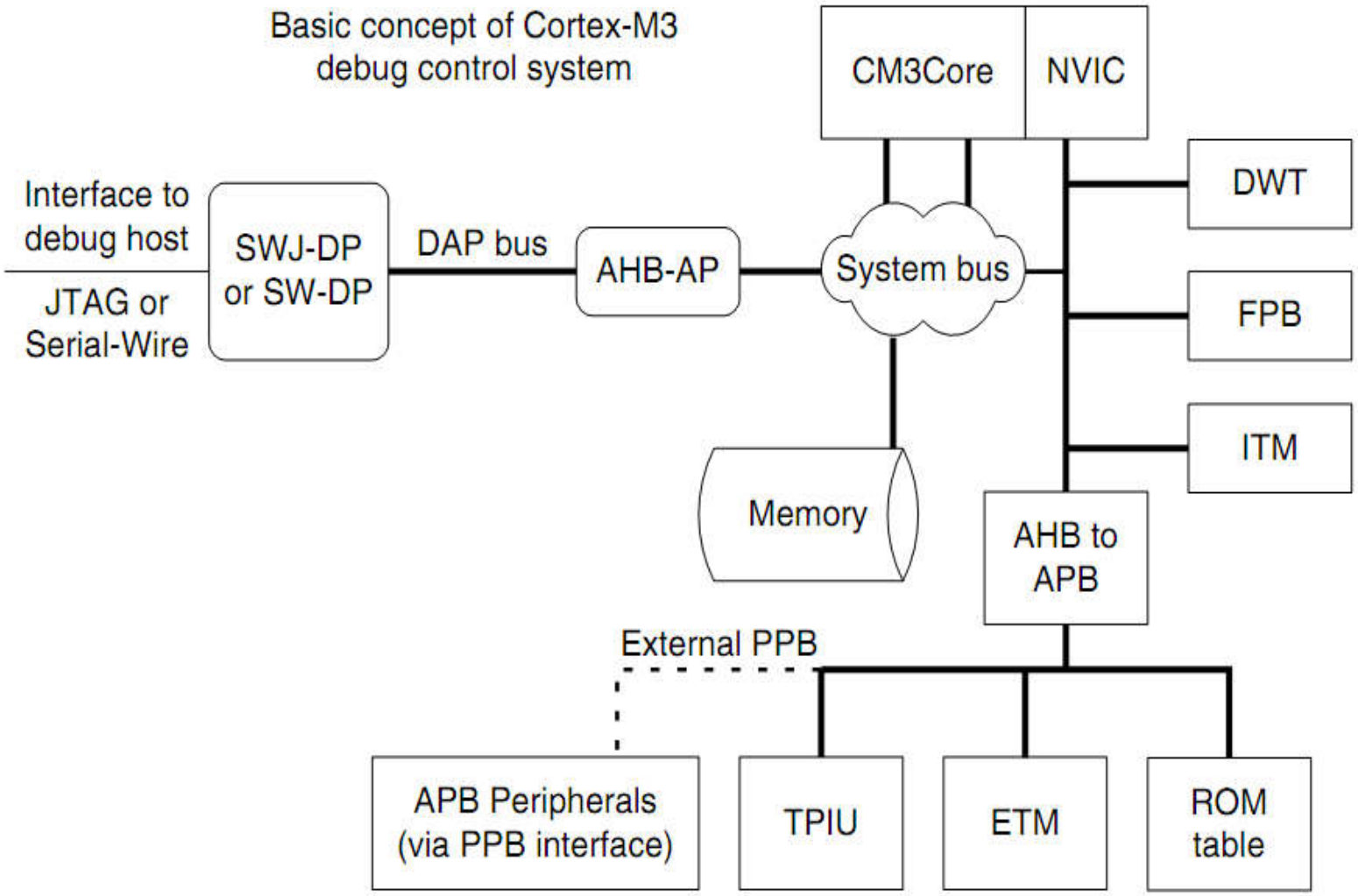
STM32 example configuration



Debug blocks of ARM Cortex M

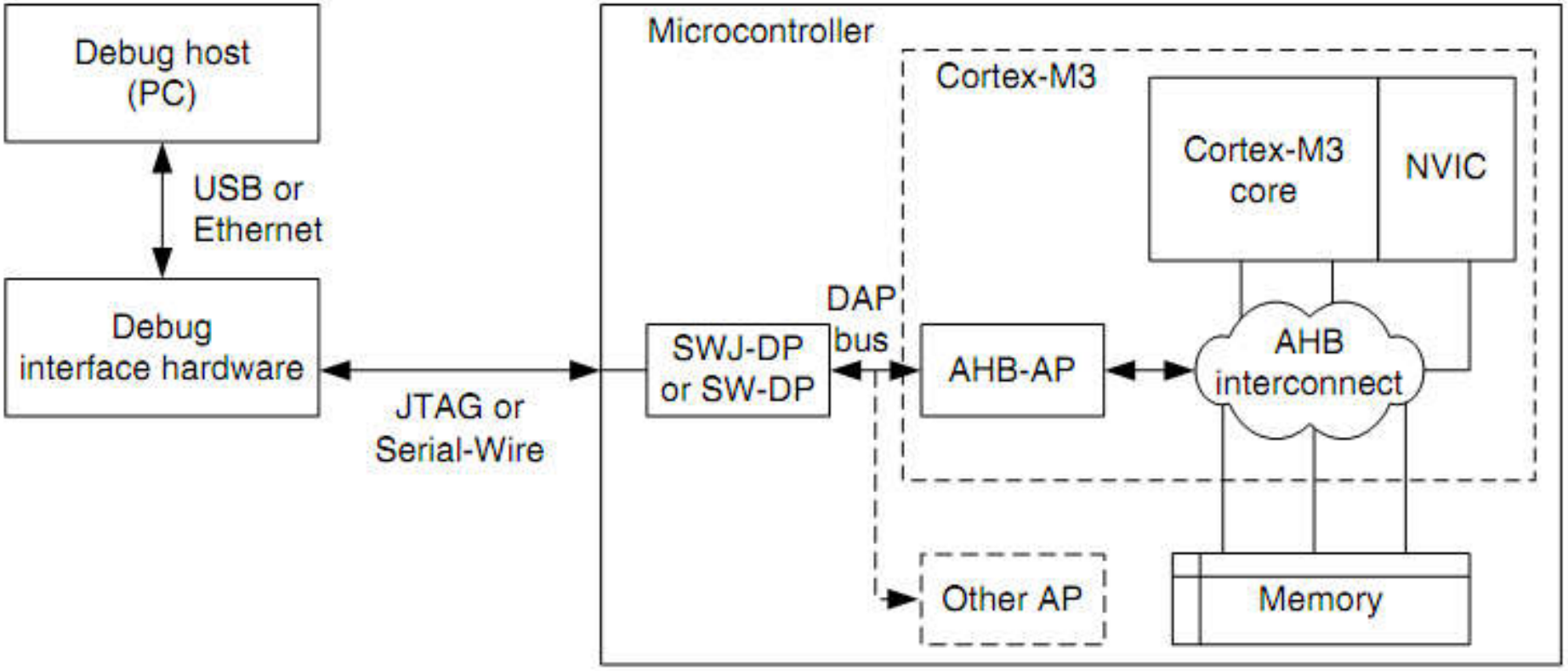
Budapest University of Technology and Economics
Department of Measurement and Information Systems

Cortex M: Coresight debug system



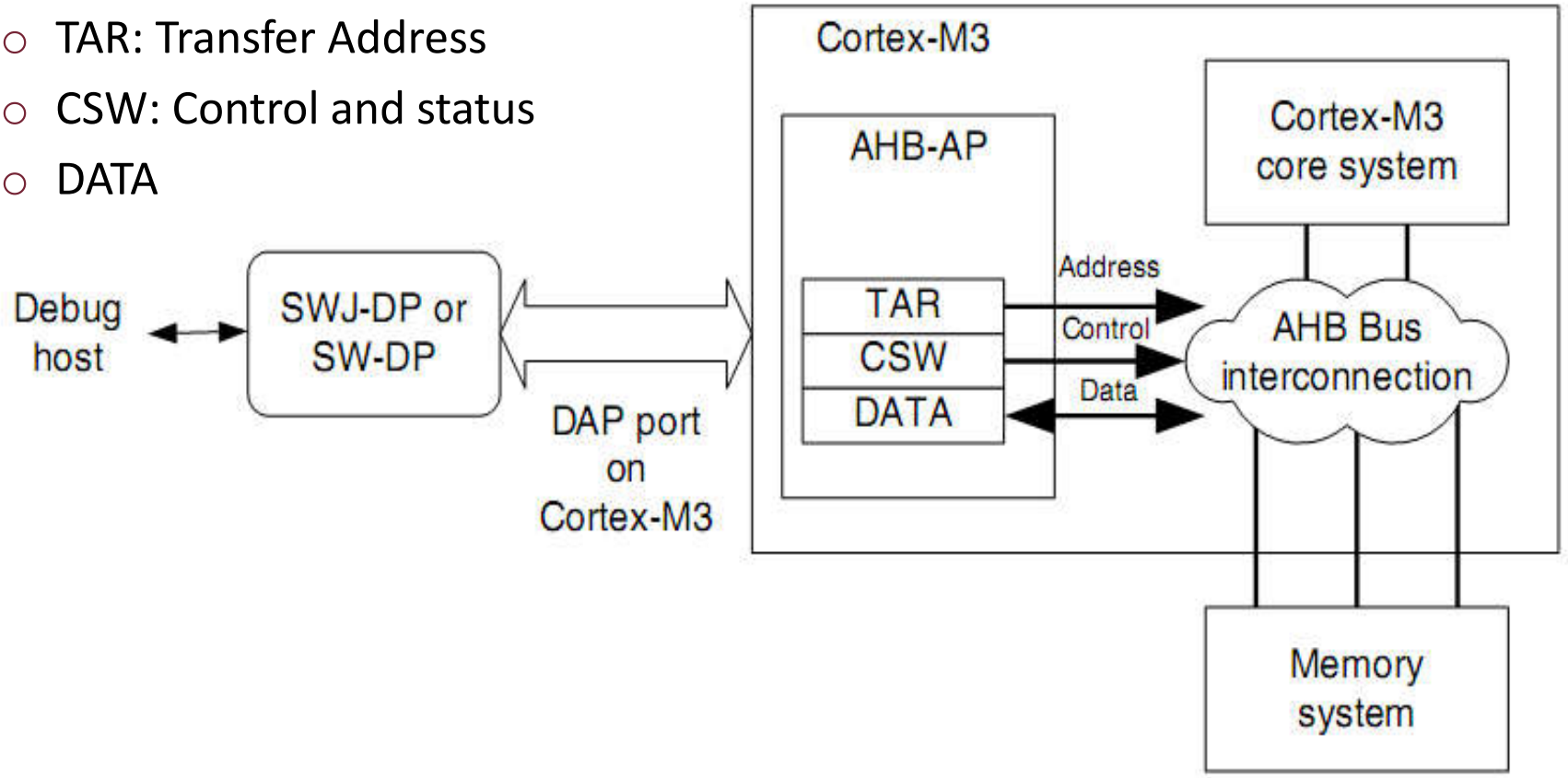
Debug port interface

- SWD or JTAG connection



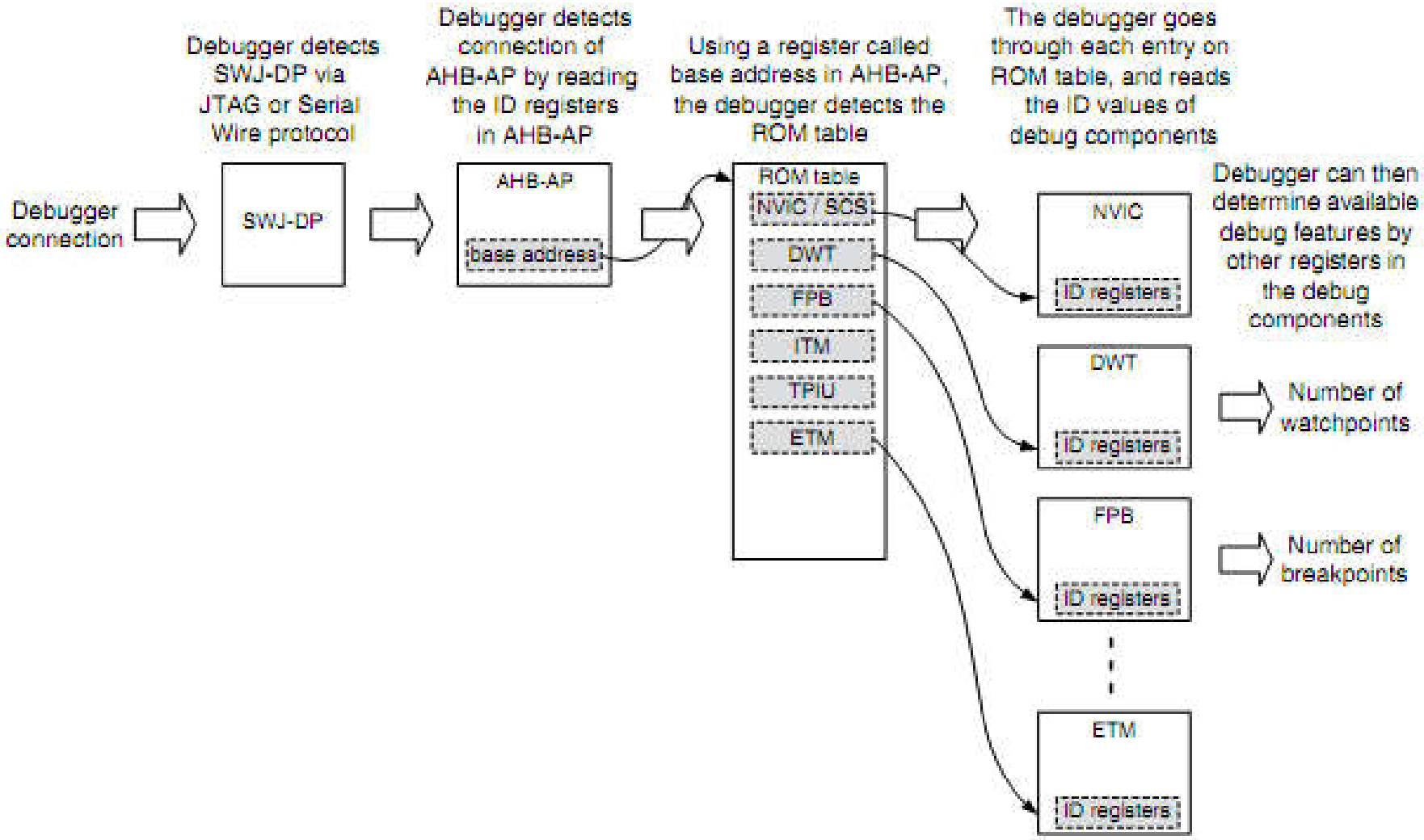
AHB-AP: Advanced High-Performance Bus Acces Port

- Bridge between the debug ports and the Cortex M3 system
  - TAR: Transfer Address
  - CSW: Control and status
  - DATA



Identification of debug blocks

- There is an ROM table containing the address of debug blocks



Debug modes

- C_DEBUGEN bit in the Debug Halting Control and Status register sets the MCU into debug mode
  - Only setable throug the DAP by external device
  - Setting the processor int Halt mode is also in this register. It is setable through SW
- 1. Halt mode
  - Instruction execution is stopped
  - The System Tick Timer (SYSTICK) counter stops
  - Step instruction can be used
  - Interupts are suspended

Debug modes

- 2. Debug monitor mode
  - The MCU executes the 12th interupts (debug monitor)
  - The SYSTICK counter continues to count
  - The interupts can preemtp or not preept the debugger based on their priority levels
  - Support single stepping
  - Memory content can be modified through the debug monitor handler

Debug modes

- 2. Debug monitor mode
  - The MCU executes the 12th interupts (debug monitor)
  - The SYSTICK counter continues to count
  - The interupts can preemtp or not preept the debugger based on their priority levels
  - Support single stepping
  - Memory content can be modified through the debug monitor handler
- For systems where the full stop is non allowed
  - High priority interrupts can serve critical hardware handling

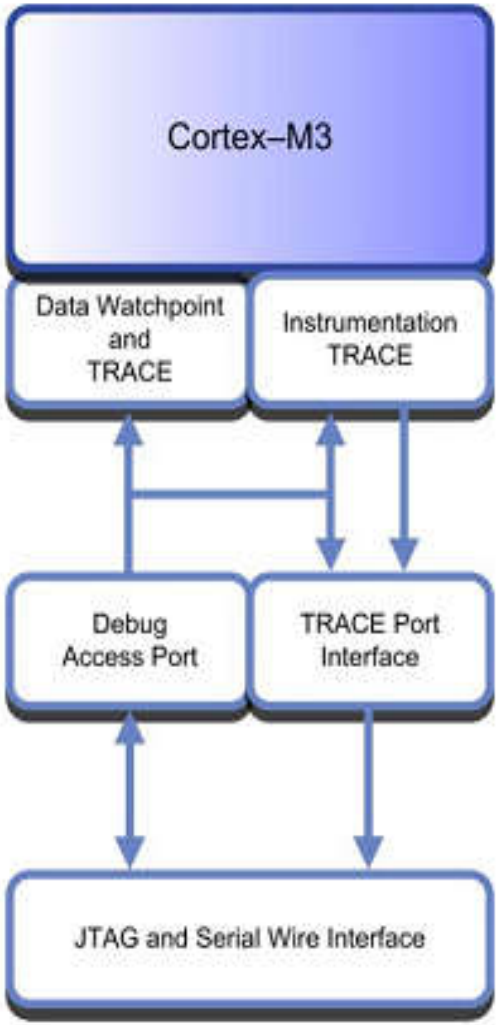
FPB: Flash Patch and Breakpoint Unit

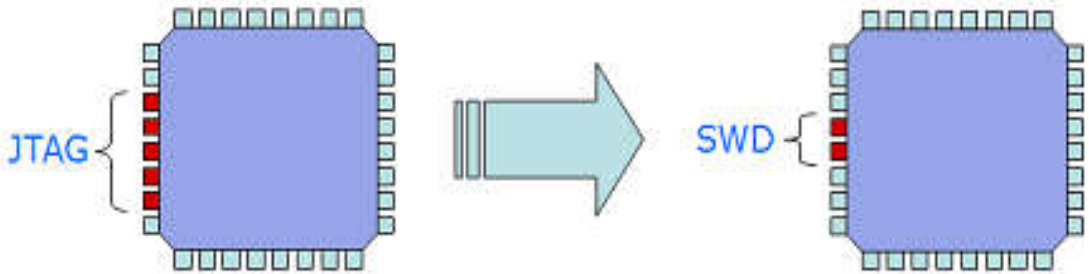
- Generating Hardware breakpoints
  - 8 comparators
    - 6 program address
    - 2 complex literal
- Flash Patch feature
  - Modification to the non modifiable ROM
  - 2 literal comparator
  - Flash area remappel to SRAM
  - Not important for us

ARM Cortex M trace blocks

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

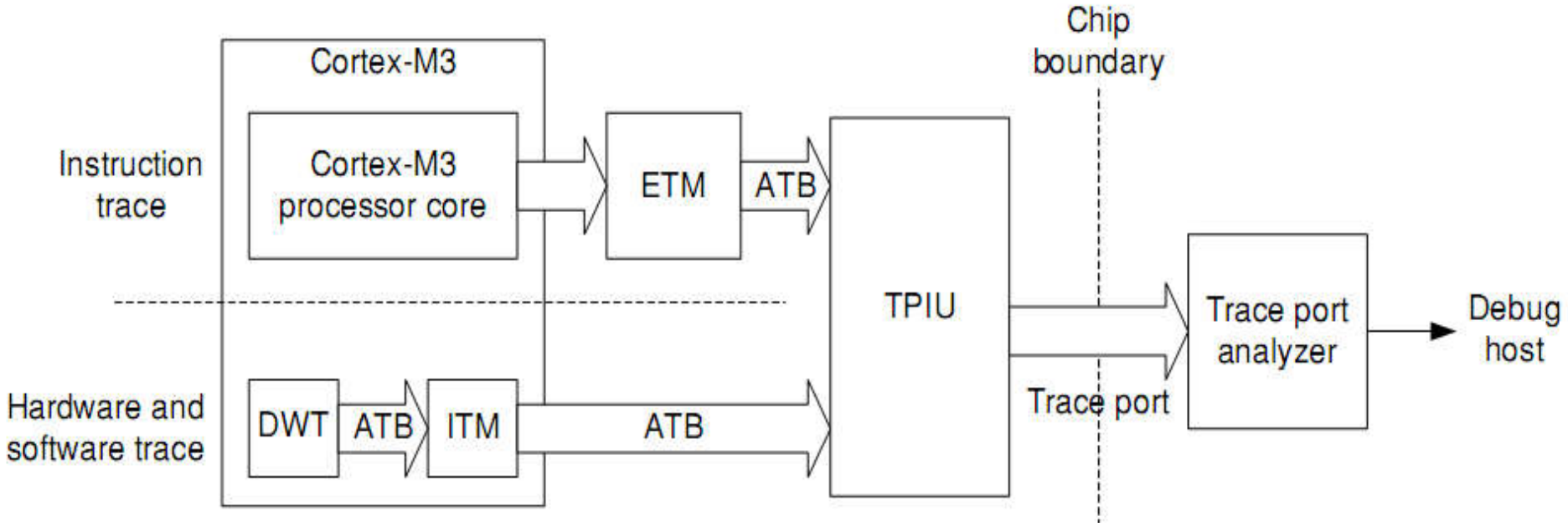
Trace ports





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 system

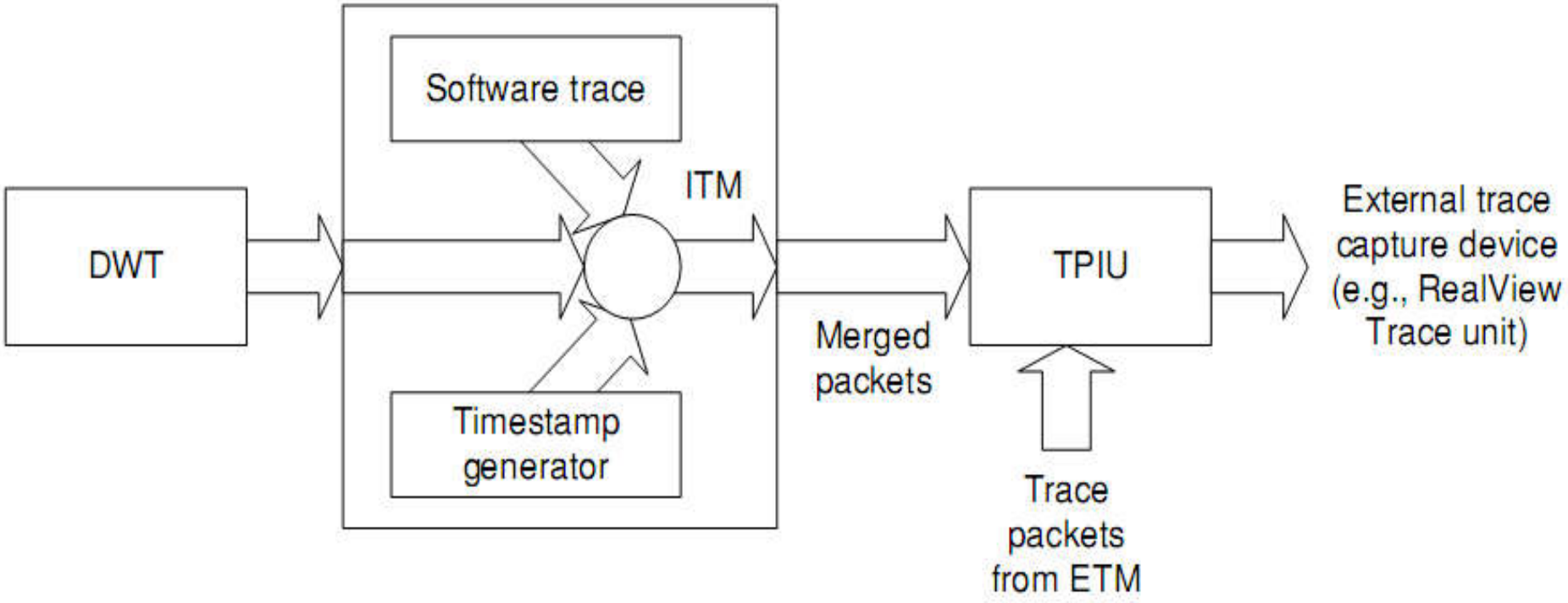


DWT: Data Watchpoint Trace

- 4 comparators: data address / program counter
  - Hardware watchpoint: took the processor into debug mode
  - ETM trigger: trace packet sending indicator
  - PC sampling trigger
  - Data address sample trigger
- Counters
  - CPU core clock counter
  - Sleep cycle counter
  - Interrupt overhead counter
- PC sampling
- Interrupt trace

ITM: Instrumentation Trace Macrocell

- Consol messages (printf)
- Can be used by the DWT
- Timestamp generator

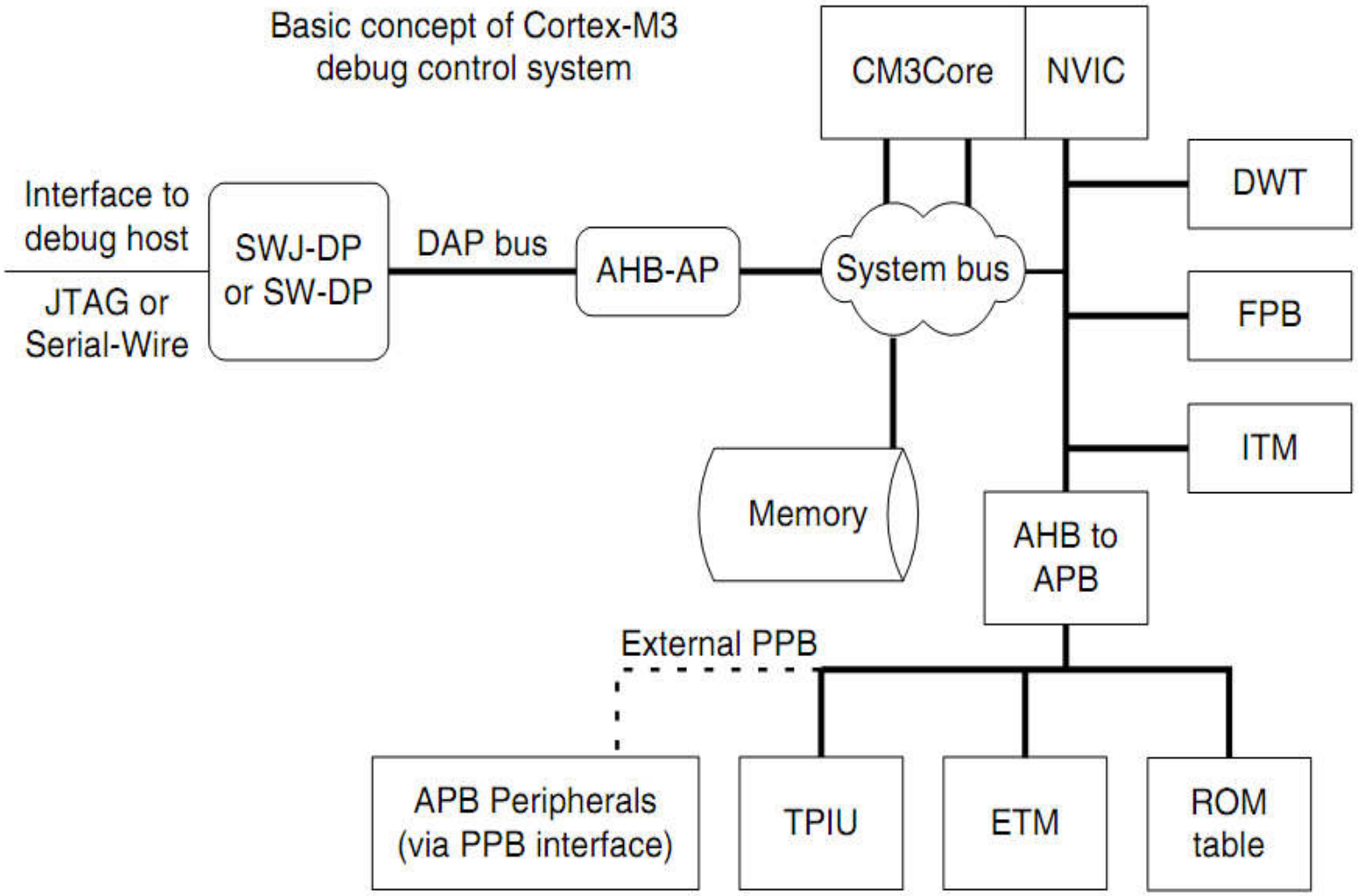


ETM: Embedded Trace Macrocell

- Introduced for the ARM7 cores
- Instruction execution trace
- DWT can be used as comparators

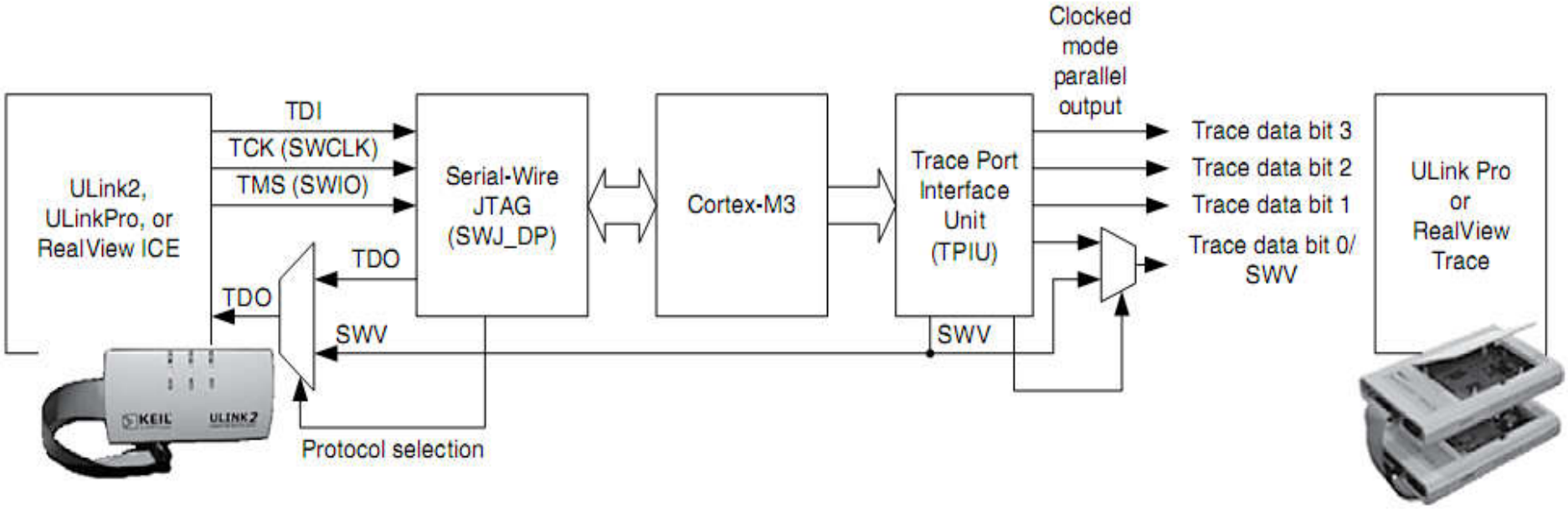
- Traceing interrupts
- Tracing instructions
  - Every executed instruction is traceable
    - Debugger must have the binary code, and debug data to process the trace messages

Cortex M3: Coresight debug system



TPIU: Trace Port Interface Unit

- 4-bit syncronous mode
- 1-bit UART like asyncronous mode



TPIU: Trace Port Interface Unit

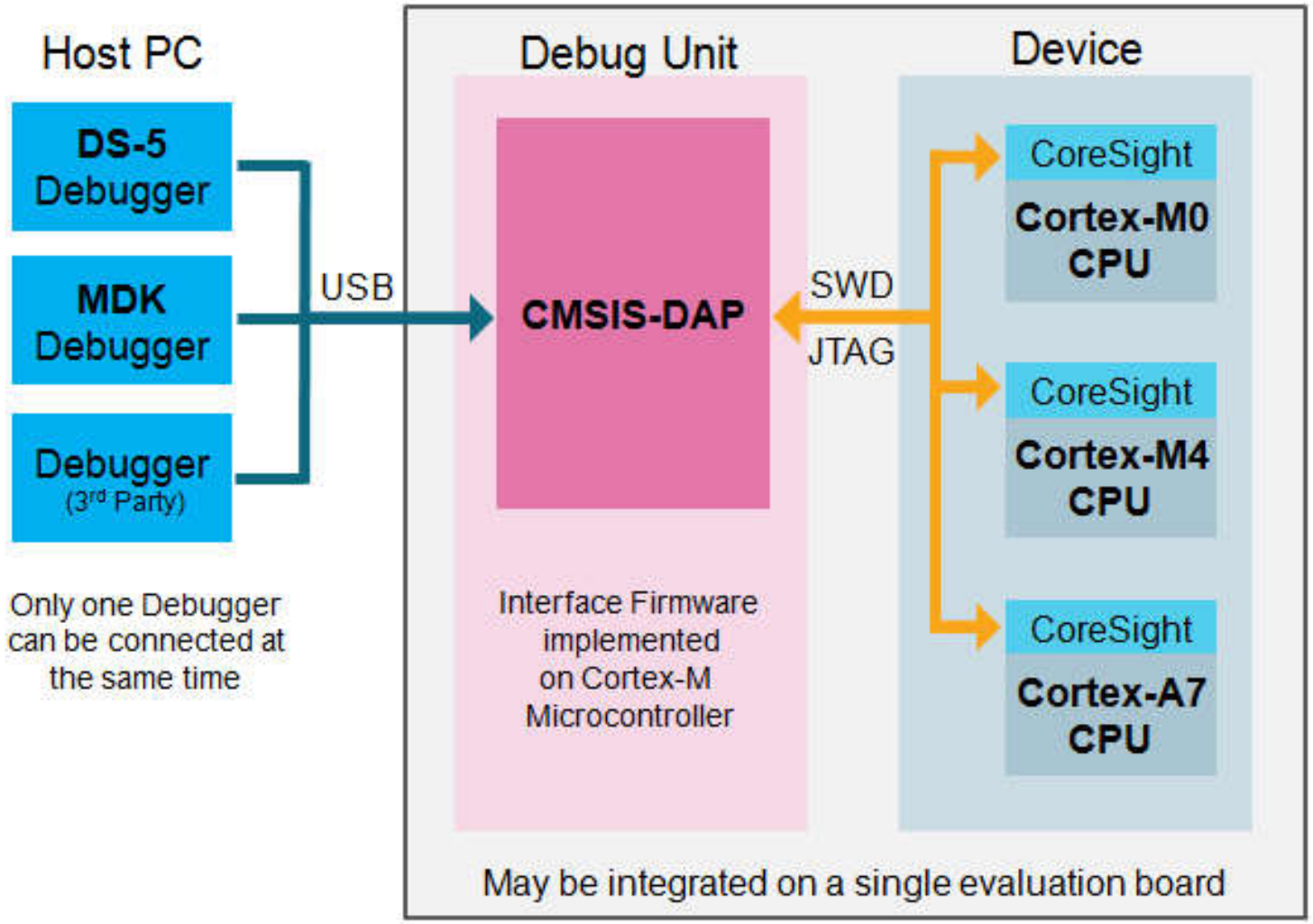
- Trace output

	31 … 24	23 … 17	16	15 … 8	7 … 1	0
Bytes 3-0	Data 0xA7	Data 0x53	0	Data 0xAA	ID 0x03	1
Bytes 7-4	Data 0x52	Data 0x2A	0	Data 0xA8	ID 0x15	1
Bytes 11-8	Data 0xCA	ID 0x03	1	Data 0x54	Data 0x29	0
Bytes 15-12	0 0 0 1 1 1 0 0	Data 0x64	0	Data 0xC7	Data 0x63	0

New generation of debuggers:

CMSIS-DAP

CMSIS-DAP


Host PC
DS-5 Debugger
MDK Debugger
Debugger (3rd Party)
Only one Debugger can be connected at the same time
USB
Debug Unit
CMSIS-DAP
Interface Firmware implemented on Cortex-M Microcontroller
SWD
JTAG
Device
CoreSight
Cortex-M0 CPU
CoreSight
Cortex-M4 CPU
CoreSight
Cortex-A7 CPU
May be integrated on a single evaluation board