ELEC-H410
REAL-TIME SYSTEMS

Chapter 10

Emulators

Emulators

CONTENTS

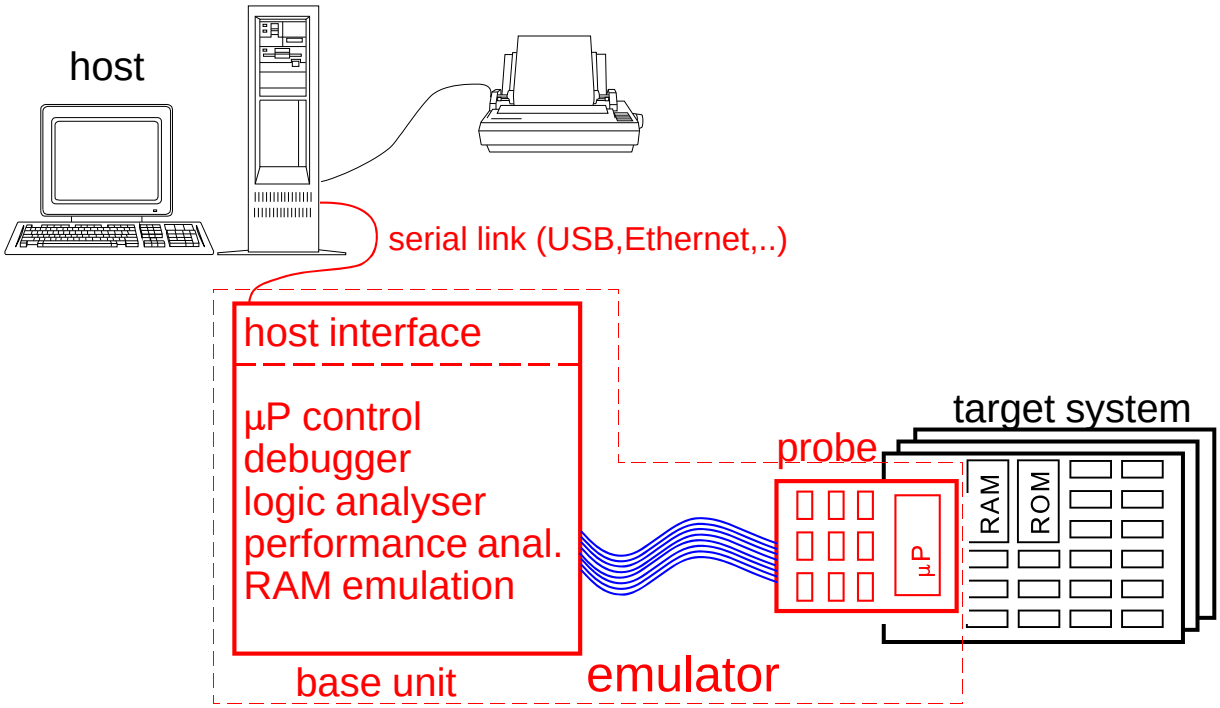
Introduction

Definition and aims

- the emulator is a device which is a substitution of a µP or an add-on
  - it provides functions analog to a debugger like :
    - load executable code
    - control the execution (run, stop, break,....)
    - display code, registers, variables
  - with little or not degradation of the performances
- two principal categories
  - ICE = **I**n **C**ircuit **E**mulation
  - ONCE = **ON** **C**hip **E**mulation

Introduction

ICE : *In Circuit Emulation* replaces the µP



On this figure, we see a complete development system based on an **ICE (In Circuit Emulator)**.

The target system, formed of several printed circuit boards is located at the right bottom. The CPU board is at the foreground.

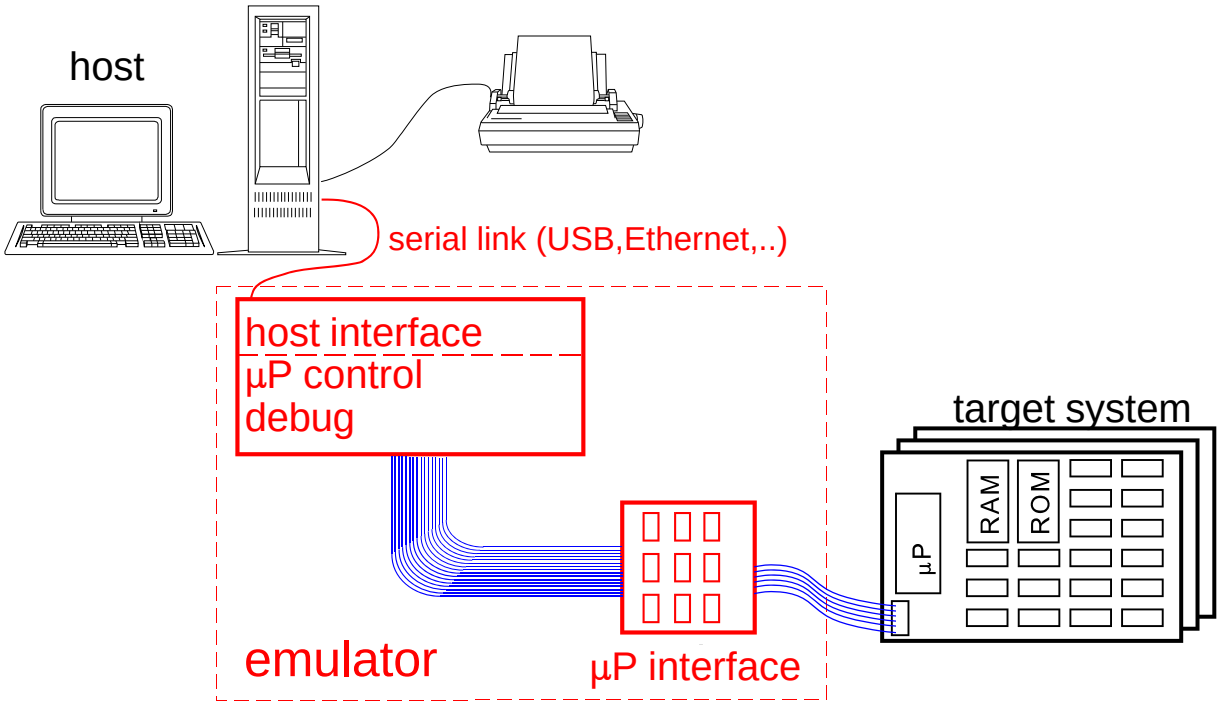
The name In Circuit Emulation comes from the fact that the **microprocessor is withdrawn** from the system and replaced by a **probe** which includes a microprocessor fulfilling the same functions, plus some auxiliary circuits. This solution works only if the processor is placed in a socket (generally expensive).

The probe is connected to the emulator itself, which contains:
- circuits allowing to control of the microprocessor in the probe (start, stop..)
- the debugger
- an optional logic analyser
- an optional performance analyser
- emulation memory that can replace the target ROM (and possibly the target RAM)

All is controller by the host computer via a communication line.

Introduction

ONCE : ON Chip Emulation



In the ONCE emulator, on the other hand, the microprocessor is left on the target. This solution is thus easier to implement, and less expensive, because you do not have to pay for a socket and the microprocessor is directly soldered on the PCB.

ON-Chip-Emulator means that the functions of emulation are carried out within the processor itself; they occupy an additional silicon surface, which slightly increases the cost of the processor.

The emulator is connected to the target system by a small size connector (one of the standards is the JTAG using 4 or 5 signals). The ONCE emulator is definitely simpler than the ICE and includes:
- the buffers to drive the cable to the target
- the interface to the host
- the functions to control the target microprocessor.

The ONCE emulators are thus less powerful (not internal RAM, no logical analysers), but much less expensive.

Introduction

functions of the host

- development of the software
  - edition
  - assembly, compilation
  - linking
  - mass memory
- debug
  - user interface of the emulator
  - load the program to be executed
  - input of the commands
  - debugger
  - logic analyzer
  - **simulation + simulated I/O**

The host PC, in addition to the software development which was treated in the previous chapters, is used as man-machine interface for the emulator system.

Let us also mention two additional functions:

- **simulate** the microprocessor to develop the algorithms (and if a convenient and sufficiently fast simulator exists for this microprocessor)

- provide **simulated input-outputs**: the program on the target system can be developed even if it is not connected to the process; I/Os are redirected to the host PC facilities (windows, files, keyboard,...). Of course the target code has to be modified. For example, instead of reading the register of an analog-to-digital converter to measure a temperature, the target code makes a request to the emulator. The emulator passes the request to the host and a message "enter temperature value" appears on the screen.Of course the manufacturer of the emulator has to provide a library of appropriate function calls). **Simulated I/O are not real-time I/O.**

Emulators

CONTENTS

- Introduction
- **ICE (In Circuit Emulators)**
  - **probe**
  - **memory**
  - **debugger**
  - **logic analyzer**
  - **stages of development**
  - **advantages**
  - **disadvantages and limits**
- ONCE
- conclusions

ICE

the probe is the heart

- functions
  - µP control (start/stop, display & modif registers,....)
  - predict certain events (opcode fetch, jump)
- very complex with current µP
  - several internal units working in //
  - several pipelines
  - internal cache memory
  - branch prediction
  - virtual addresses, ...

∃ activities invisible from the buses
=>need to access internal signals

ICE

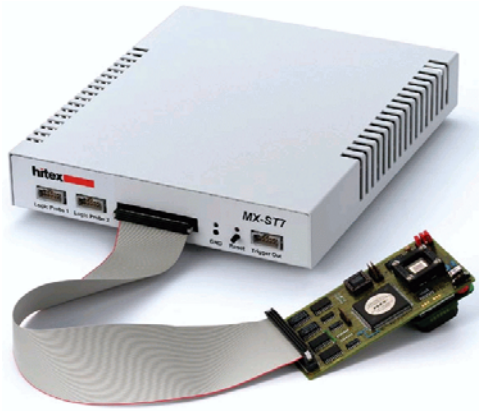
probe : emulation processor

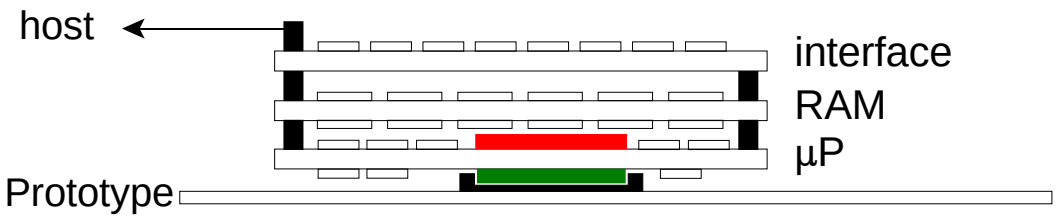
- special µP (*bond-out chip*)
  - access to internal signals
  - sold by the manufacturer of the µP
  - very expensive
- normal µP
  - + hard (MSI, LSI, FPGA, ASIC)
    - reconstructs the inaccessible internal signals
    - designed by the manufacturer of the emulator
- advanced µP
  - special debug registers
  - special combinations of signals on normal pins

ICE

probe

- low frequency
  - µP in the probe (not too far!)
  - flat cable link
- > 50 MHz
  - µP within a few mm of the prototype
  - probe in several layers of SMD





From a hardware point of view, connecting the probe becomes more complex when the frequency of the processors increases.

Connections by simple flat cable are possible for small microcontroller at low clock frequencies.

As soon as the speed exceeds 10MHz, special twisted cables should be used.

Beyond 40MHz, the probe must be very close (a few millimetres) of the target circuit and all the critical components must be located in a very low volume. Multilayer PCBs are populated on the two faces by SMD (Surface Mounted Devices) components.

ICE

emulation memory

- RAM
  - to emulate the target ROM (allways)
  - to emulate the target RAM (not allways)
- mapping mechanism
  - divide memory in
    - RAM/ROM
    - target/emulator
    - forbidden zones
  - granularity from 1 byte to some Kbytes
  - real-time decision => fast memory and decoding => expensive
- protection mechanisms
  - detect write operations in a zone emulating ROM
  - detect accesses to non-existent zones

To develop the program, we wish to download the code easily to the target and to modify it at will; it is thus convenient to have in the emulator a read-write memory (RAM) which will replace the target program memory (ROM). When the code is ready, it will be loaded in the non-volatile ROM.

If the RAM on the target is not yet available, it is sometimes possible to also use the RAM of the emulator to place the data.

The operation of mapping consists in configuring (via commands sent to the emulator) the emulator RAM to fix:
- which are the program memory locations provided by the emulator (which will be considered as read-only once it is loaded because it emulated the target ROM)
- which are the program memory locations provided by the target (for example the portion of the code already debugged)
- which are the addresses of the data memory respectively provided by the emulator and the target
- which are the zones where no memory will ever exist on the target system

During the execution, the emulator must make a real-time decision at each cycle, so that the processor of the probe is connected to the emulator or to the target memory. This makes the system expansive, since the memory must be very fast to compensated for small delays due to these decisions.

ICE

debugger

- same functions as normal debugger +
- carried out by another CPU
  - debugger cannot be hanged
  - occupies no memory in the target
- complete control of target µP in hardware
  - start / stop
    - any time (even if hanged)
    - on breakpoint
    - techniques
      - NMI (Non Maskable Interrupt)
      - cycle stealing
      - special registers of µP
  - handling of the registers at the breakpoint

ICE emulators provide the same services as a classical debugger, but by a different, safer and more powerful mechanism.

Indeed:

- the ICE has got its own processor to manage the debugger. **Debugging takes no resources on the target** (neither memory space, nor CPU time)

- the microprocessor of the probe is always under control of the emulator even if the program is hanged; the execution can always be stopped without losing the state of the processor; better still, the logical analyser will give invaluable indications on what the processor was doing when the operator decided to stop the program.

The control of the processor is obtained by several techniques
- action on the non-maskable interrupt request
- cycle-stealing: a multiplexer allows to direct the next opcode fectch from a special ROM belonging to the debugger
- a lot of microprocessors include special registers dedicated to the debug

ICE

debugger

- *hardware breakpoints*
  - via special internal registers of µP (si ∃)
  - hardware : comparators on the bus (add,data,ctl)
  - => breakpoint on code, data and I/O, RD&WR
  - several simultaneous breakpoints
- *run-time checks* (hardware)
  - illegal addressing of a memory area
    - writing in ROM
    - reading/writing in a non-existent zone
  - illegal opcode (for µP without TRAP)

The mechanism of the breakpoint for high-end processors has been explained in the chapter on the debuggers: special registers can be loaded with the conditions on address, data and control signals corresponding to the desired breakpoint.

For processors that do not provide such registers, debuggers generally try to replace the opcode by a software interrupt.

ICE emulators use a very different mechanism: hardware binary comparators permanently compare the content of the buses and to the break condition. This is quite powerful since:
- we can put a breakpoint not only on the code, but also on the data; a condition like " ***break IF the CPU writes in variable X a data>2534 OR IF .....***" can be written.
- several comparators are thus working simultaneously

The emulator will also take care of a series of tests during the execution (run-time checks) like the detection of an illegal access to a memory area (write in a zone emulating ROM, access to a non-existent zone,..).

If the emulator comprises a "parallel" control unit, reconstructing normally invisible internal signals, it will be able to detect illegal operations (nonexistent opcode, abnormal value of Stack Pointer, ...)

ICE

debugger

- monitoring/modification of the variables
  - at the breakpoints
  - in real time (! requires a dual-port memory i.e. with double access from debug µP + target µP)
- symbolic debug and HLL support
  - loading of the symbol tables (var, func, line#)
- ASM inline
  - patch the code "on the fly"
  - write small test programs

One of the classical functions of debuggers is the modification of global variables (for example coefficients of a PI regulator) during the execution.

In the simple emulators, this operation requires to stop the target processor, i.e. real-time is lost.

In more sophisticated emulators, we can modify the variables in real-time, because the emulation RAM is "dual-ported": the microprocessor of the probe and the emulator have a concurrent access to the data memory.

The debugger of the emulators is of course symbolic.

Finally some emulators have an integrated assembler allowing to directly produce small pieces of code in the emulation memory for fast correction (!do not to forget to correct the source code on the host!), or to write small test programs.

ICE

integrated logic analyzer

- permanent connections to μP
- + some external lines
- software oriented but timing sometimes included
- **real-time trace**
- coordination with ASM/HLL source code
- performance analyser
  - measure times
  - histograms of calls
  - histograms of execution times
  - code coverage

A logic analyser integrated into the emulator has two main advantages

- the harnessing problem disappears: the logic analyser is always and automatically connected to the processor of the probe. Some extra channels are sometimes available.

- real-time trace is possible. The analyser "films" the buses of the processor during the execution at full speed. When the processor is stopped, the logic analyser is frozen.

In the current devices, the ergonomics are excellent: the "film" of the busses can be synchronized with the source code in high-level language on the host (i.e. cursors move synchronously in both windows).

Hardware debug

printed circuit board and power supply

- before installing the circuits
  - connect the power supply and check
    - the value of the current, should be 0
    - if not check for smoke and hot spots
    - check voltage at all the power supply pins
      - /!\ multiple power supply
- install the circuits
  - progressively if possible (! no open inputs)
  - remake the same checks, plus the temperature of the circuits

think of the price of some circuits and especially of the emulation probe (k€)

An emulator can be very helpful for the hardware development. Here some advices related to the order of the operations.

Once the prototype of the PCB is done, you can perform a few simple tests before populating it with expensive integrated circuits (including the emulator probe which can be worth several thousands of €) which could be destroyed if the circuit would be defective:

- connect a power supply with an ammeter and current limitation, current should be zero
- if not, check for smoke and hot points, indicating short circuits between tracks; remove them
- with a voltmeter, check the power supply pads of all the integrated circuits (conformity compared to the schematic, voltage level of tension); be particularly careful if you have multiple voltages to avoid overvoltages on some circuits

You can then start to install the circuits gradually, in a logical order, avoid leaving inputs unconnected. Check that no circuit reaches an abnormal temperature.

Development of the hardware

clocks and RESET

- check the clocks
  - frequency
  - duty cycle
  - rise time
  - noise
  - ringing
  - overshoot
- check RESET line
  - ? stuck to 0 or 1
  - power-up timing
  - manual push-button
  - inopportune intervention of the watchdog?

Development of the hardware

connect to the external world

- monitor-debugger (see appendix)
  - install the code in ROM via a programmer
  - establish the connection (RS-232, network, ...)
  - if does not function
    - connect an emulator if available
    - if not, replace the monitor by ultra simple test programs (see next slide)
- emulator
  - connect the probe
  - check status via the emulator
    - power supply
    - clock

One of the most popular methods for the development of the microprocessor-based systems is the **monitor-debugger**. It's a small piece of code that you install in the program memory of the system via a special programmer

- old programmers required that memories were placed in sockets so that they could be removed from the target system to be placed in the programmer
- currently most ROM are FLASH memories which are permanently soldered on the PCB and can be loaded "in situ" by injecting serial bitstreams (which is called **In Circuit Serial Programming**). You have to foresee special connectors on the PCB
  - JTAG standard for separate memory chip
  - if the ROM is internal to a microcontroller, follow the indications of the manufacturer

Once the monitor-debugger is installed, you can connect the target to the host via a **serial link** (RS-232 or network) which is managed by the monitor-debugger.

If you don't use a monitor, or if the monitor does not function properly, some simple test programs can loaded in ROM to debug the board, or you will have to resort to an emulator.

After having taken all the precautions, you can connect the emulator probe; this one will already provide us indications on the presence of voltages (that should have been checked before!) and the correct operation of the clock oscillator.

Development of the hardware

buffers / address decoder

- test busses and ChipSelect signals
  - methods
    - emulator
    - monitor/debugger
    - R/W commands in memory and I/O
    - mini test programs in ASM (R/W loops)
  - instruments: logic probe, oscilloscope, logic analyzer
  - check
    - logic levels, transition times
    - chronograms / access times, setup & hold times
    - operation of the address decoder

Hardware development

- Specific I/O (peripherals)
  - digital I/O ports
  - communication ports: serial, network
  - ADC, DAC
  - use R/W to fill configuration registers and exchange data
- code memory
  - install (or fill) a test memory with known content
  - dump, disassemble
  - comparison with reference file
  - compute a *checksum* (should also be in the power-on self tests)

Hardware development

RAM tests

- ! do not rely on too simple tests
  - write immediately followed by read works without memory !
- fast tests with block R/W commands
  - unique address test
    - fill memory
    - modify a block
    - check that only this block has been changed
  - test of bits stuck to 0 or 1 (00,FF)
- small test program
    - write AA in even bytes and 55 in odd bytes
    - reread each byte
    - rewrite its 1's complement
    - reread to verify
- sophisticated patterns

Debug software without hardware

without emulator

- only possible on the host
- development and debug of the HLL algorithms
- **simulator** of the target processor
  - very efficient (sometimes faster!) for small µP
    - ASM+HLL
    - simulated I/O (display, files, keyboard, mouse)
  - very slow for special processors
    - video
    - DSP
  - special tools for ASIC, FPGA (see ELEC-H409)

Debug software without hardware

with ICE emulator

- develop software on the host
- load it in emulated program memory
- execute on the µP of the probe (internal clock)
- debug with the emulator
  - **powerful**
  - total control of the execution
- simulated I/Os:
  - change code to make calls to the OS of the emulator
  - user interface on the host (display, keyboard, files)

ICE

hard/soft integration: from emulateur to target

TARGET	EMULATOR
nothing	CLK, RAM, ROM, I/O(simul)
CLK	RAM, ROM, I/O(simul)
CLK, real I/Os	RAM, ROM
CLK, I/O, RAM	ROM
CLK, I/O, RAM, ROM(partial)	ROM (partial)
CLK, I/O, RAM, ROM(total)	nothing BUT - debugger - logic analyzer - performance analyzer

- difficulty = cross-influence
  - *glitch hard* => incorrect execution of the software
  - *bug soft* => incorrect behavior of the hardware

After the separate tests of the software and hardware, the **integration phase** starts, which is the most delicate one. This table shows how you shall gradually transfers the various elements from the emulator towards the target.

You should work **as soon as possible** with the **real input-outputs** of the process (or of a hardware simulator of the process, using rapid prototyping), because it is the part which was not tested yet and which is likely to highlight problems in real-time and inmeasurements (noise,...).

At the end of the integration phase, the emulator does not provide any resources to the target, but it is interesting to leave it connected, to continue to benefit from its excellent debugging functions.

The principal difficulty of integration lies in the cross-influences between software and hardware which can lead to a bad identification of the bug.
Indeed, a hardware problem, like a parasitic pulse due to cross-talk, can give the same effect as a software bug.
Conversely, an error in the program can cause parasitic pulses on some signals, that you could attribute to race conditions in the hardware.

ICE

advantages (1)

- no modification of the prototype
  - emulates the ROM by RAM inside the emulator
  - consumes no resources in target memory
- debugger
  - very powerful
  - never hanged
  - target µP always under control
  - *breakpoints* always available (even in ROM)

ICE

advantages (2)

- logic analyser
  - real time trace
  - correlated with the µP
    - symbolic debug
    - synchronized with source code (ASM and HLL)
  - performance analysis
- facility of development
  - of hardware without software
  - of software without hardware
  - hardware/software integration

ICE

disadvantages

- cost
  - 2k€ => 50k€
- interaction with target
  - volume/thickness of the probe
  - slight differences of timing or of fan-out
  - use of the NMI
  - differences in the behaviour of the RESET
- lacks (cheap systems)
  - multitasking / dual-ported memory
    - stop the execution to
      - enter a command
      - put a breakpoint
      - define a trigger
    - not convenient for slow processes
  - emulation of multiple processors

The ICE emulator is a very useful and powerful instrument, it has got some disadvantages

- the cost, which can prevent you from using an emulator

- its incidence on the target is not negligible, in particular because it is necessary to foresee the clearance to install the probe.

Then, it can happen that, when you remove the probe and put the microprocessor back in its socket, the system does not function correctly. This is due to the slight differences which remain between the microprocessor and its emulator:
- difference of some ns in the chronograms of the signals
- the fact that the emulator uses the NMI (Non Maskable Interrupt) which has not been tested yet
- the fact that the RESET of the emulator is different, in particular to be able to control it from the host

Finally the emulators are not always complete, in particular in the cheapest ones. When the emulation RAM cannot be accessed simultaneously from the target and from the emulator, real-time debug is more difficult because we have to stop the target to give some commands to the emulator.

ICE

limits of technology

- µP/µC more and more sophisticated
  - complex architecture
  - nGHz
  - many internal operations
    - caches / pipelines / registers
  - many alternatives of same CPU core
- *embedded core*: (ex ARM)
  - ASIC with µP core : unique circuit
  - no µP signal visible from outside
- =>abandon of the probe
  - too expensive
  - too much time to develop (more than µP ?)
  - impossible to manufacture

ICE emulators reach their limits in sophisticated processors in which
- more and more of operations take place inside the case
- pipelines are feeding several units working in parallel
- the clock frequencies reach several gigahertz
- the same microprocessor core is sold under licence to several manufacturers who make different versions

The extreme case is reached when the microprocessor core is used to produce a micro-controller; it becomes a unique circuit with all pins dedicated to the input-outputs and all the events related to the program occur within the case.

One must then give-up the idea to manufacture an ICE which will be too long (or even impossible) to develop and hence too expensive.

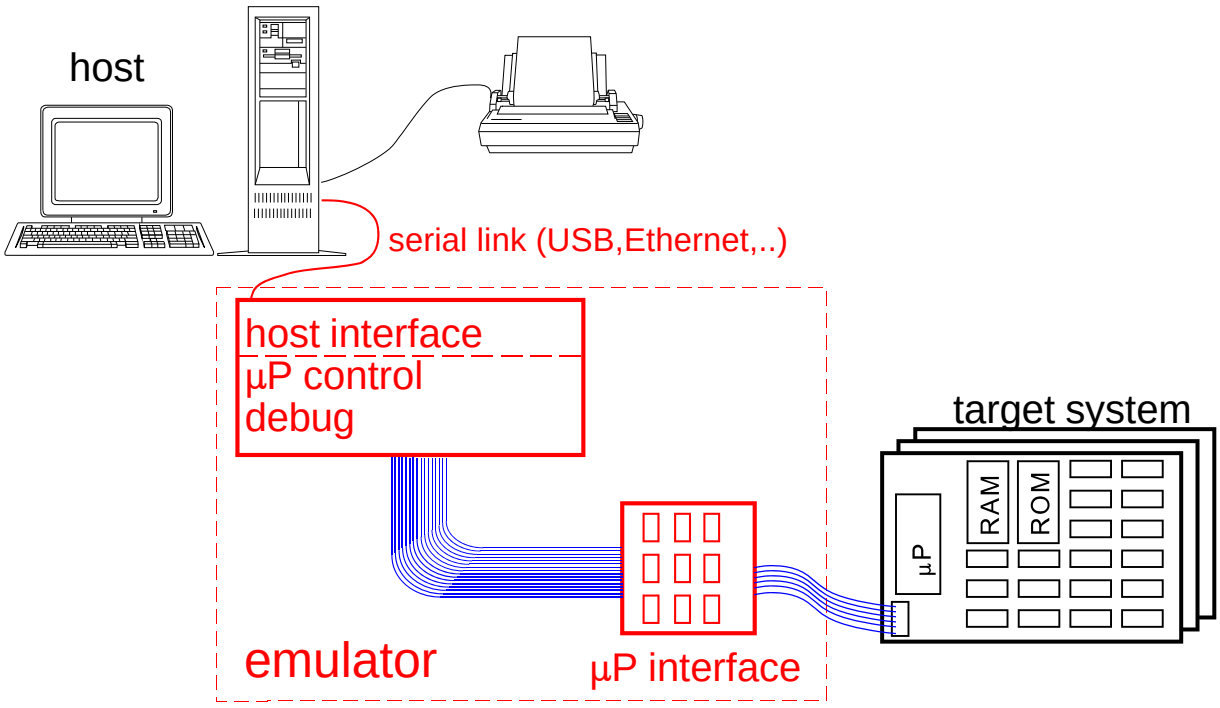
Emulators

CONTENTS

- Introduction
- ICE (In Circuit Emulators)
- **ONCE** (On Chip Emulators)
  - **principles**
  - **advantages**
  - **limitations**
  - **combination with logic analyzer**
  - **instrumentation of the code**
  - **evolution**
- Conclusions

ONCE

ON Chip Emulation



ONCE

principles

- debug functions integrated to µP
  - loading of the code
  - run/stop
  - *breakpoint / watchpoint*
  - read / modify registers
- emulation interface: only a few pins
  - JTAG port (or proprietary)
  - the external box is chiefly a host/target interface dedicated to the target

ONCE

advantages

- simple and inexpensive emulator
  - difficulty deferred into µP (emulation cost shared by all the purchasers of the µP)
  - rapid to be developed
  - same hardware for a family based on same core
  - no expensive emulation RAM
- execution in real environment
  - all the timings are those of the real processor
  - full clock speed
- interface not very intrusive
  - few pins
  - also used for production tests

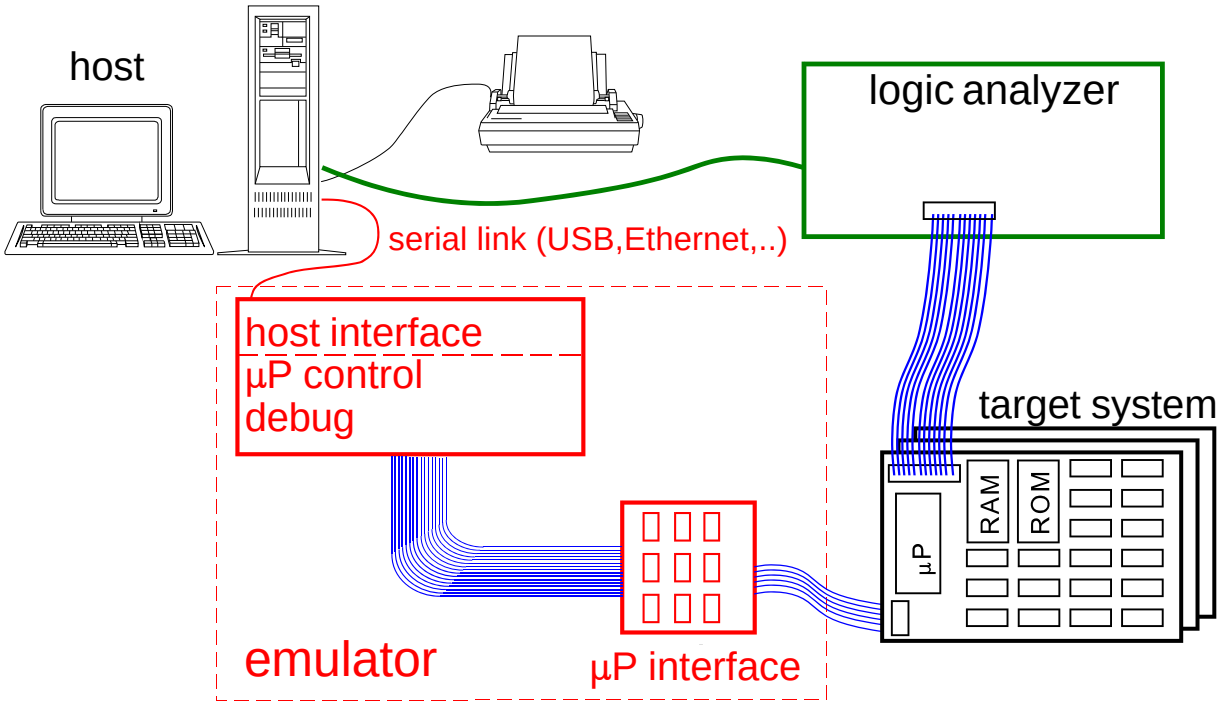
ONCE

disadvantages

- real time difficult to achieve
  - only between *breakpoints*
  - need to stop the processor to exchange data with the host
  - real-time trace:
    - limited by throughput of the serial link
    - on-chip logic analyser (rare)
- partial remedy:
  - ONCE + logic analyzer
  - ONCE + real time kernel to exchange data with host without stopping the program

ONCE

ON Chip Emulation



This figure shows the joint use of a ONCE and a logic analyser. Both are connected to the target system by their own connector.

Notice the connection between the logic analyser and the host, so that all the post-processing and interactions with the tables produced by the IDE can be carried out.

ONCE + LSA

utility

- contributions of the logic analyzer
  - analyze buses => debug timings
  - correlation hardware/software
    - display HLL source code
    - trigger on HLL source codes
  - real time trace
  - filtering and sophisticated postprocessing of the "film" of the busses
- limits of the method
  - the logic analyzer cannot see in the µP
    - execution in cache
    - pipelining
    - program in internal FLASH / data in internal RAM

ONCE + LSA

"instrumentation": a few extra code can help debugging

- instrument the code by markers
  - "dummy" write to external addresses or I/O port for
    - entry / exit a function
    - discontinuities of execution (branches, IRQ, trap)
  - writing content of internal variables in external RAM
  - RTOS internals
    - tasks name, ID, status
    - trace scheduling, preemption
    - track deadlocks, schedulability
  - overhead of instrumentation
    - negligible with a logic analyser
    - data sent by serial link
      - more CPU time
      - throughput problems

To make unobservable events visible, the programmer will have to make an additional effort called **"instrumenting"** his code, i.e. by adding a few instructions to cause external demonstrations of internal events:

- writing a coded value in a static ***dummy variable*** at a given external address or on an I/O port; the external logic analyser will be able to capture it. A byte can represent 256 different events! In this way, we can trace the passage by a function, know wether a branch has been taken or not, be aware of an exception, ...

- we can create *shadow variables,* i.e. an external copy of an internal variable by the same mechanism

- sometimes it is also important to be able **debug the operation of the RTOS**, and in particular to know the state of the various tasks. Some RTOS provide and document this facility, but it is often necessary to recompile the kernel with some additional instructions to instrument it (see labs).

The overhead corresponding to the instrumentation of the code is generally negligible when using a logic analyser. Sending data by a serial link can also be considered, but takes more CPU time and the throughput is more limited.

ONCE + LSA

improvement of µP

- efforts of the manufacturers
  - addition of status signals:
    - sequential execution
    - branch taken or not
    - pipe-line flush
  - special information port
    - output in real time of compressed internal info
      - branch addresses
      - interrupts
      - exceptions (*trap*)
- example MPC555: flexible debug modes configured by internal registers

Today, it is possible to manufacture relatively inexpensive cases with a large number of pins. Hence the manufacturers help this effort of instrumentation by providing status ports offering a partial visibility of the internal events, without the programmer having to add lines of code.

Some processors have got special function registers that can be programmed when booting to configure how much information the processor will give to the external world. An example is given in the next slides.

ONCE + LSA

MPC555

- µController
  - PowerPC core
  - interface ONCE
  - many internal peripherals (timer/PWM/CAN)
  - mode µC : 64 bits I/O
  - mode µP : 24 bits addr, 32 bits data, 8 bits I/O
- several programmable debug modes
  - few modifications of the code
    - no instrumentation
    - just to configure the internal registers
    - automatic execution of debug cycles

ONCE + LSA

MPC555 : maximum mode

- optional tracing via external buses
  - instructions
  - internal data (Read and/or Write)
  - cache
- advantages
  - total visibility of all the execution
  - easy correlation between the film of the analyzer and source code
- disadvantages
  - loss of 56 I/O if µC mode
  - the more you wants to see, the more you slow down the processor

ONCE + LSA

MPC555 : middle mode

- tracing via external buses
  - instructions (addresses only)
- advantages
  - loss of I/O limited to 24 bits
  - slow-down by a factor 2 "only"
- disadvantages : you don't see data any longer
  - loss of the values of the variables
  - the debugger must reconstitute the execution on the basis of the "film" of the logic analyzer, which contains only the addresses and not the opcodes

ONCE + LSA

MPC555 : minimum mode

- tracing via external buses
  - instructions (branch addresses only)
- advantages
  - minimum loss of I/O
  - weak slow-down (3 cycles per branch)
  - less channels to connect to the logic analyzer
- disadvantages
  - only branches are visible
  - the debugger must reconstitute the execution by "navigation" in the code

Emulators

CONTENTS

- introduction
  - définition et buts
  - structures
  - rôle de l'hôte
- ICE (In Circuit Emulators)
- ONCE (ON Chip Emulators)
- **Conclusions**

Conclusions

- do the maximum in simulation
  - software and hardware
  - accelerates the development
    - detection of the faults
    - reduction of the number of protoypes
- make one(?) prototype
  - simulation cannot show everything
    - real I/Os, interferences, ...
  - assembly of simulated blocks
  - debug by cooperation of tools
    - emulators
    - logic analyzers
    - oscilloscopes