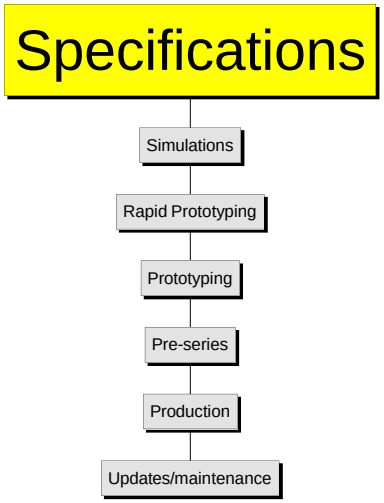
ELEC-H410
REAL-TIME SYSTEMS

Chapter 2

Workflow

Workflow of embedded systems



Workflow

specifications (1) : functional aspects

- paramount importance
  - sometimes seems "lost" time, it is the contrary
  - mistake at the beginning => enormous final cost
  - cost x10 at each stage of the workflow
- frequent difficulties
  - specifications evolve
    - real needs of the customer not well known or not well expressed
    - potential of the new system underestimated by the client
    - part of a system in development by several partners
  - lack of dialogue
    - between you and the client
    - at the client (final user != writer of specifications)

Danger = beeing obliged to change basic options and go back at a high level of the workflow

To avoid expensive back steps in the workflow (i.e. not to be obliged to reconsider basic options), writing good specifications is of primary importance. The time spent at this first stage is a real investment.

The ideal is obviously that the client and the designer manage to stabilize the functional specifications and that they agree perfectly on its contents and its significance.

In practice, the specifications often evolve
- either because the customer badly evaluated his own needs
- or because he does not realize the potential of the future system (and the designer has to guide him in this matter)

Do not undervalue the dialogue problems
- between the customer and the designer
- within the customer's firm, where the end-user was not necessarily consulted when the specifications were written. This leads to blocking situations, lots of frustration, or even sabotage!

Workflow

Specifications (2) : contractual aspects

- foresee all tasks
  - design
  - prototypes, samples
  - production (materials, tools, ...)
  - startup
  - period of the guarantee
- foresee how tasks can be added
  - add clauses to the contract with fixed price
  - cost per hour
  - as significant for the customer as for the designer
- beware of penalties
  - late delivery
  - insufficient performance

When you have to propose a price for the design, it is very important not to omit any item.

Considering the risk of evolution of the specification, the method to add new tasks (and calculate the supplementary costs) must be envisaged from the beginning:
- an addendum to the contract for a fixed amount
- cost per hour of extra work.

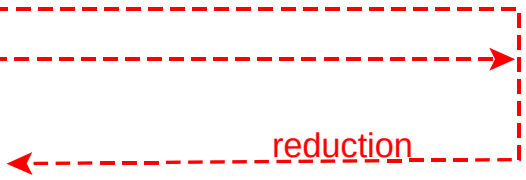
Finally, the respect of the deadlines is often essential, because most of the contracts foresee penalties
- proportional to the delay
- if the performance of the equipment does not meet the specification

REM: seen from the customer, the specifications are quite as significant, because he wants to avoid paying supplements in the event of a fault or of lack.

Workflow

buy or build? (1) : structure of the costs

- fixed costs
  - buy hard/soft tools
  - buy patents/licences
  - buy components for prototypes
  - labour for R&D (several 100€ /h !)
  - maintenance costs
- proportional costs
  - buy components for production
  - royalties
  - labour for production
  - maintenance costs

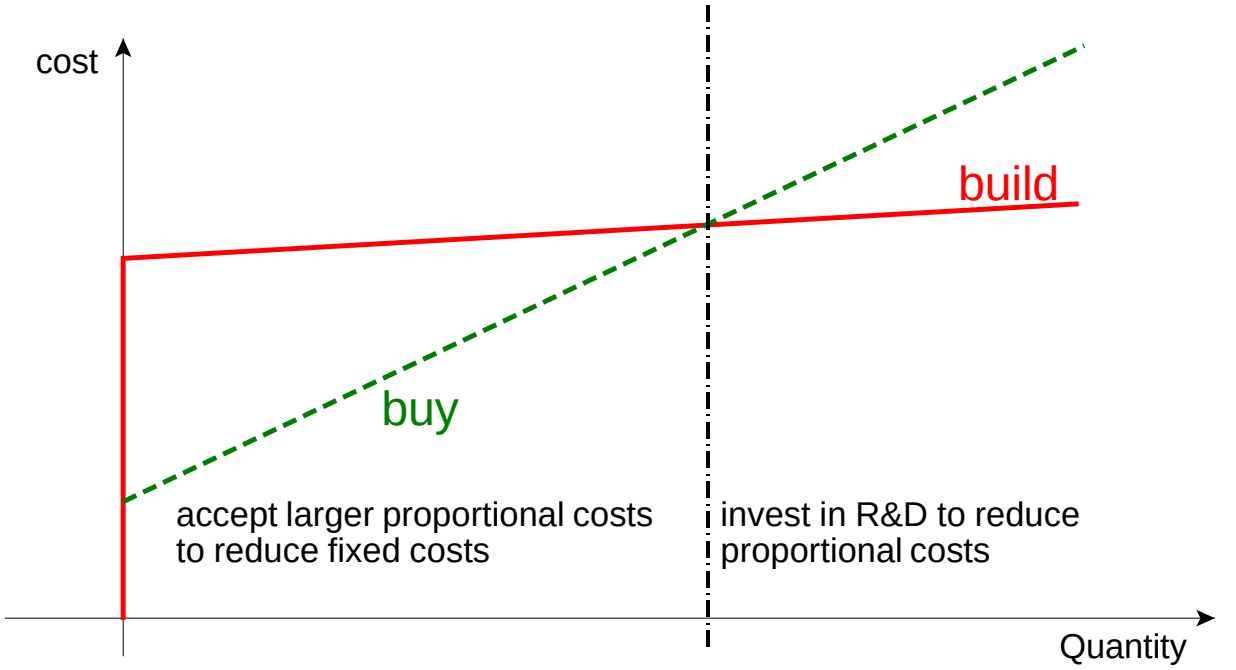


At the time of the design and prototyping, an important question arises:
- what to design?
- what to buy?
  - already made hardware or subsystems
  - IP (intellectual properties): libraries of codes, patents

This cost must compared with the very expensive manpower for research and development. Buying as much as possible is often advantageous, especially for the prototypes or for a feasibility study.

Workflow

buy or build? (2) : production



The decision can be different when looking at the production costs. There is generally a threshold in the quantity produced beyond which higher fixed costs (for designs and tools),will be compensated by the fall of the unit price of manufacture.

An example is the design of a peripheral board for a PC (PCI, PCIexpress, USB, ...). The development of the hardware and of the software stack takes approximately six months-man. It seems evident that only a very large series of production can justify this cost. For prototypes or even rather large series, it is better to buy a microcontroller including the appropriate peripheral, or an off-the-shelf interface ASIC or a piece of VHDL code for an FPGA.

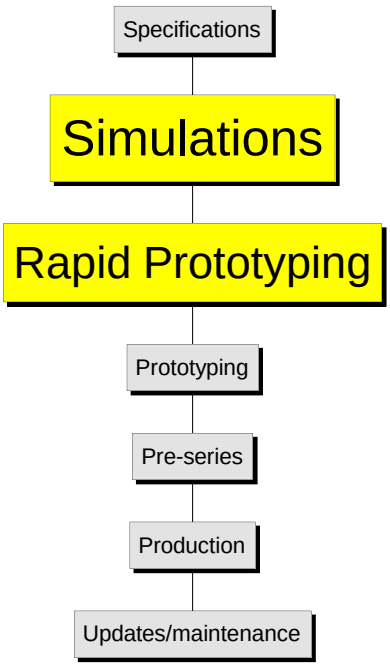
Workflow

buy or build? (3)

- other considerations
  - no existing solution is available or suitable
  - policy: will to completely master the products
  - "in-house" special competences/skills
  - availability of the required manpower
  - more performing (and expensive) tools => diminution of *time-to-market*

The decision to spend more money on the development is also dependant of other factors presented in this bullet list.
We will see in further lessons that **development tools can be very expensive, but can also considerably accelerate the design**.

Workflow of embedded systems



Simulations / Computer-Aided-Design

Why ?

- is the system feasible?
  - design / debug / test algorithms
  - check if functional specifications are met (real-time cannot generally be checked at that time)
- evaluate computing power (count +/x)
  - help to choose the hardware platform
  - floating or fixed-point? which precision?
- check boundary conditions / faults
  - simulators don't catch fire
  - help to design protections
- reduce costs
  - early detection of problems is crucial (x10/stage rule)

Simulations / Computer-Aided-Design

what is required?

- a simulator
  - general purpose: MATLAB/SIMULINK, LabVIEW
  - specialized:
    - thermal problems, fluid mechanics
    - electronics
    - mechanics
    - ....
- models
  - sufficiently elaborated not to mask relevant problems
  - not too sophisticated
    - convergence problems
    - unbearable computing time
- parameter estimation
  - validation in the whole range

Simulations / Computer-Aided-Design

common problems

- model improperly used
  - too simple
  - used outside the validated boundaries
- difficulty to estimate parameters
  - inaccessibility of the process
  - difficulty to instrument the process
  - observability problems
- several models are required (time-constants)
  - power electronics (10ns-1us)
  - current/speed/position (1ms...100ms)
- time for simulations
  - from seconds to days

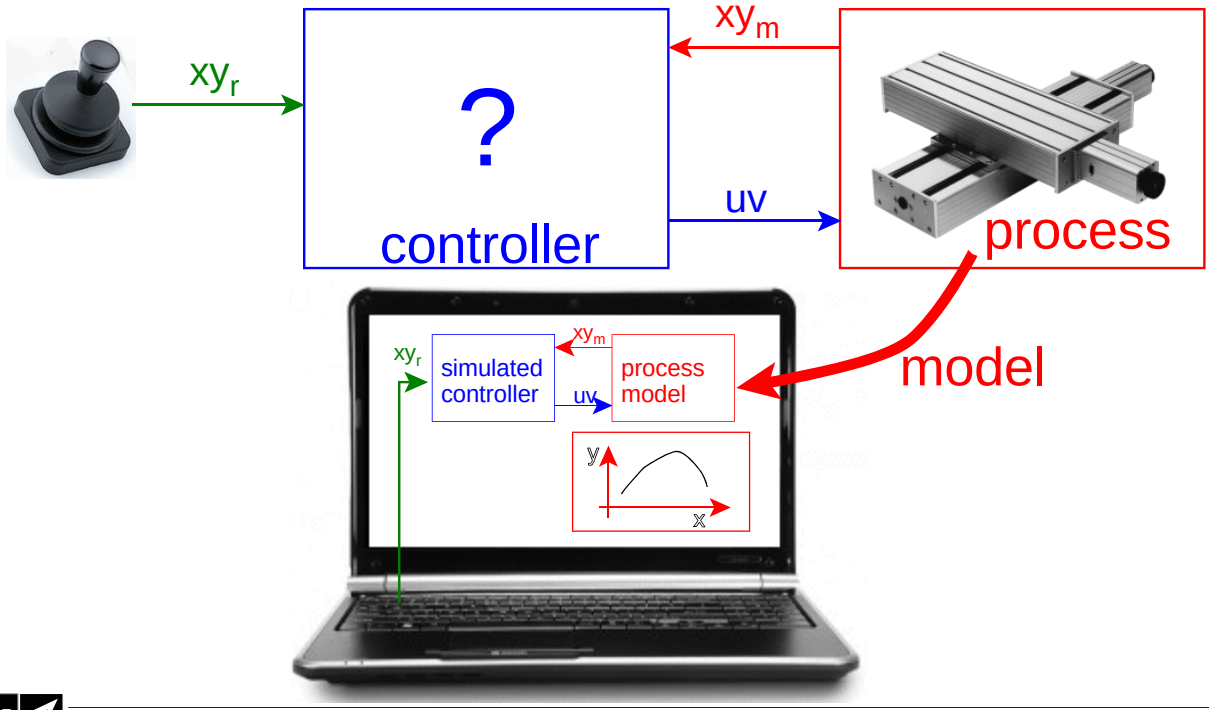
Simulations / Computer-Aided-Design

how to accelerate the simulations ?

- cluster of computers (ex: Computed Fluid Dynamics)
  - extensively used for research
  - not for the real-time embedded market
- hardware help
  - use GPU as computing accelerator
  - faster CPU (n-core)
  - subcontract some tasks to another platform with // computing
    - multi-CPU with RTOS
    - multiple ALU in FPGA
    - code for the platform can be written automatically and compiled from the simulation code (see rapid prototyping)

Simulation

simulated design of the controller



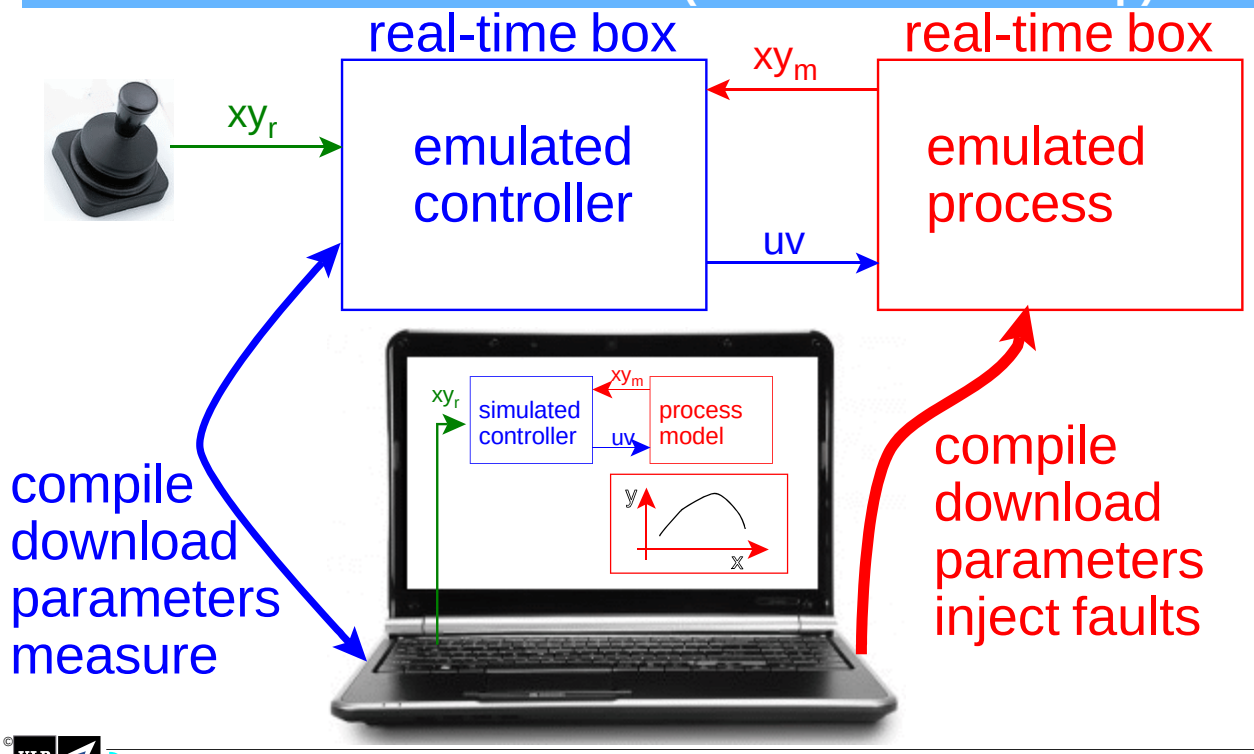
Let us imagine a simple process consisting in an X-Y table controlled by a joystick. The exact nature of the control is not important, we can imagine:

- speed controlled by the amplitude of the movement of the joystick and visual feedback (i.e. the position control loop is in the brains of the operator)
- step-by-step mode for very accurate positioning
- position controlled by the joystick

The design of the controller can be done on the simulator, with the help of a model of the process. The references x_r and y_r are given via the keyboard or via input files.

Simulation

Real-time simulation : HIL (Hardware In the Loop)



Rapid prototyping consists of **emulating** (1) the process and/or the controller by using "real-time general purpose boxes" containing:

- a microcomputer platform based around a powerful embedded processor and a real-time OS
- one or more FPGA(s) to handle very short time-constants (like fast sampling of the inputs/ouputs)
- analog inputs/outputs
- digital inputs/outputs
- a fast communication link with the computer (Ethernet and USB are the most common ones)

For slower processes, a PC running an RTOS can do the job.

The equations of the process and of the controller are **compiled by tools matching the real-time platforms** to obtain the code of the tasks running on the embedded processor and the bitstreams to program the FPGA. Those files are uploaded to the real-time boxes.

The fast links carry

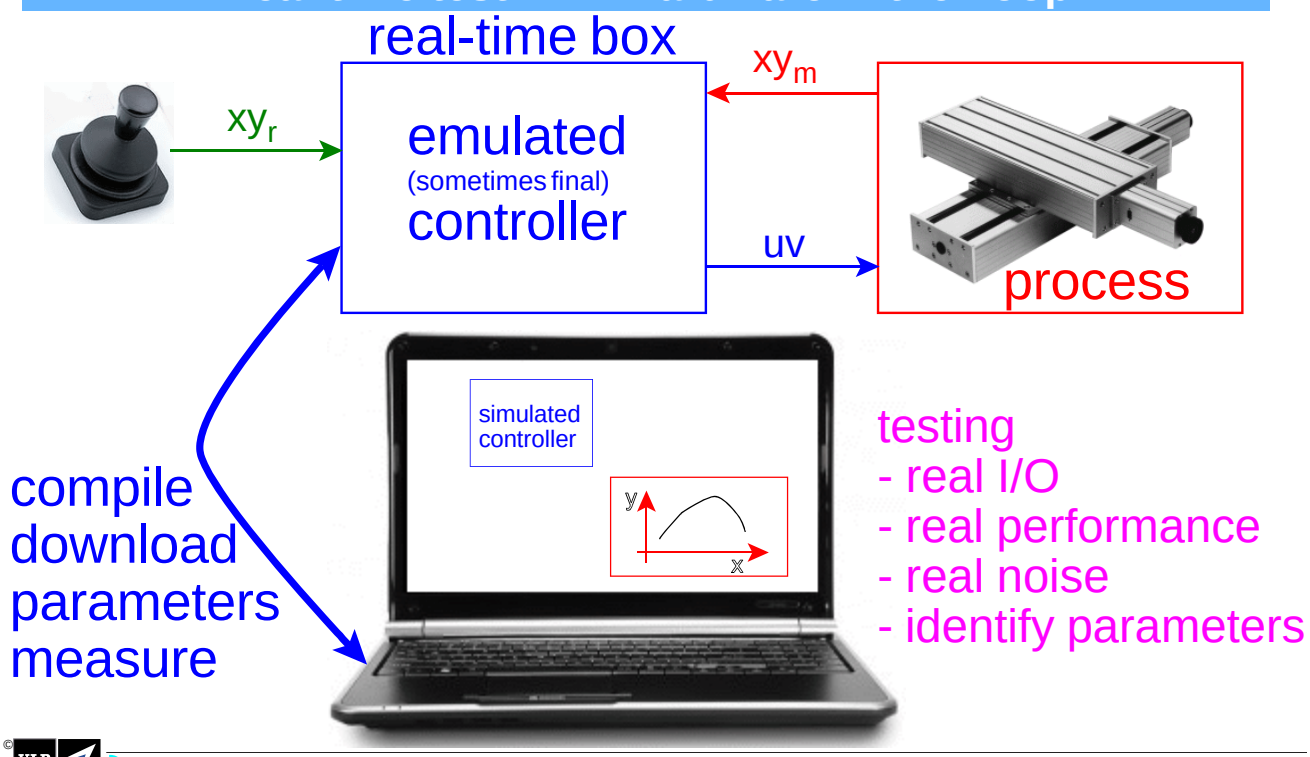
- the controls from the operator
  - run/stop
  - change parameters (i.e. K_P and K_I for a PI controller)
  - inject faults in the process
- the measurements which can be displayed in several windows.

Several manufacturers (DSpace, National Instruments, OpalRT, ...) provide such general purpose (or sometimes more specialized) real-time platforms with the drivers to interface with MATLAB/SIMULINK and LabVIEW and easily pass from the simulation on the PC to the real-time simulator(s).

(1) **emulate** means "provide the same functions/performances" with a different hardware

Simulation

Real-time test : HIL Hardware In the Loop



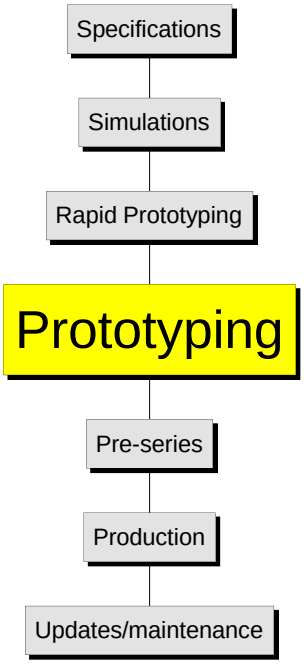
Control algorithms can be further tested by connecting the emulated controller to the real process. In this configuration we can

- test the actual inputs-outputs of the process
- be confronted to noise problems due to the power electronics driver of the motor
- refine model/parameter estimation of the process by making step responses, slope responses, harmonic analysis
- assess the actual performances

REM: it is possible to stop the development of the controller and to uses the real-time box as a final hardware for applications where:

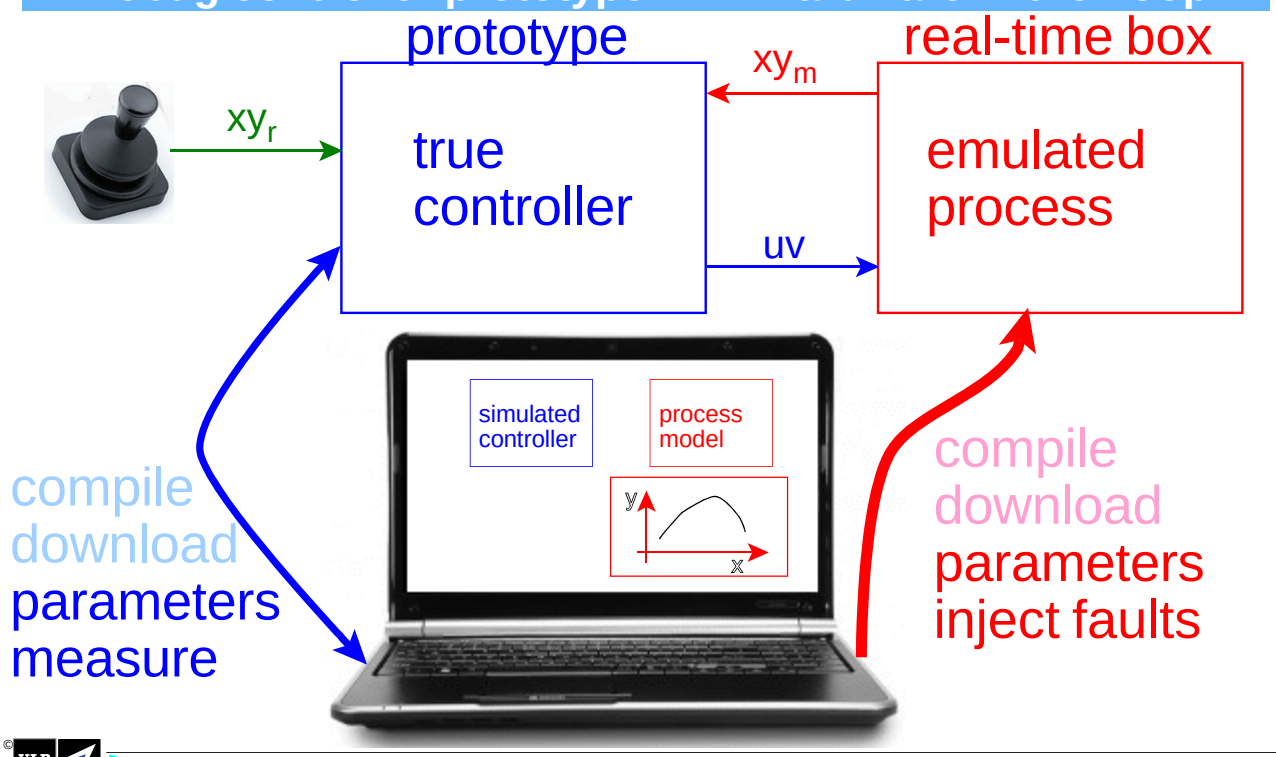
- cost are less important (ex: medical devices)
- time-to market must be very short
- compliance to standards (CE, FDA, ...) with small efforts; using already qualified sub-systems can help a lot
- the series are small

Workflow of embedded systems



Rapid prototyping

Debug controller prototype : HIL Hardware In the Loop



The next stage is to test the prototype of the controller. We shall examine the workflow of the controller design more in details in the next slides.

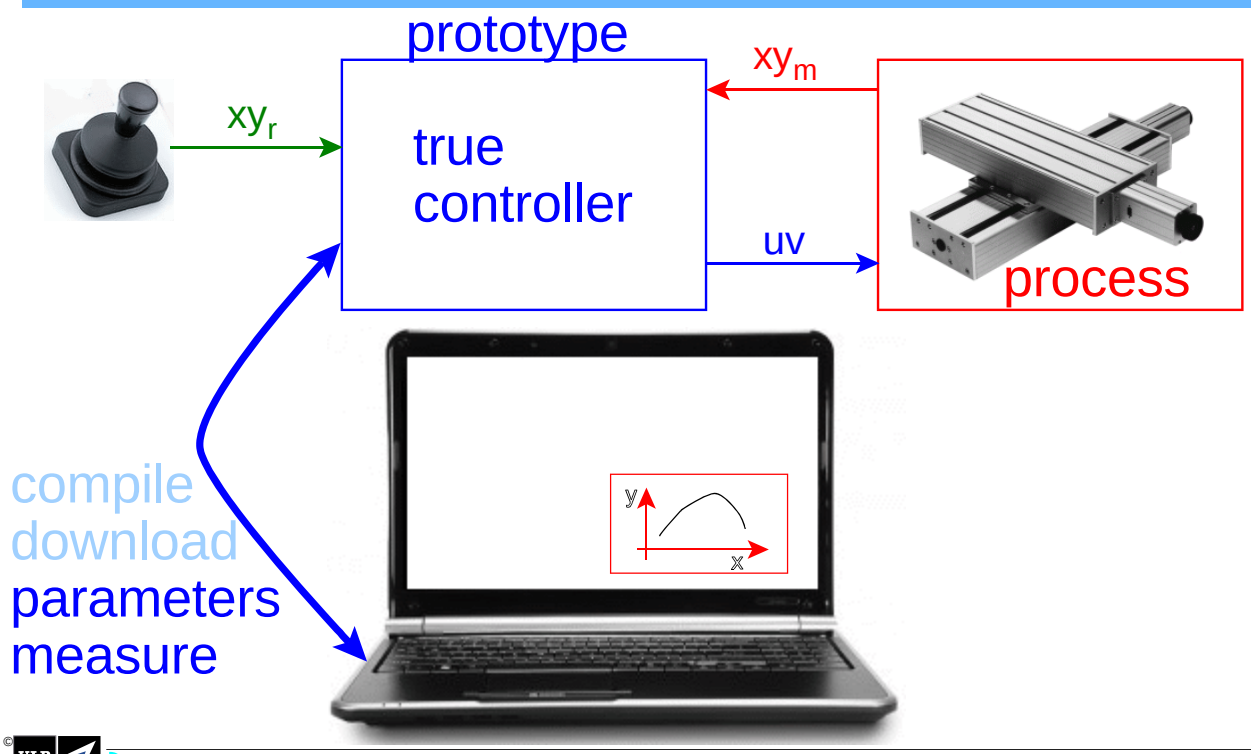
For the moment, suppose that the prototype of the actual controller has just been assembled and that the software is (seems?!) ready.

Its is important to keep the controller under control of the operator's PC and we shall see how it's possible in the chapters devoted to the monitor-debugger and the emulators.

For the first tests, it is interesting to **reuse the emulated process**, because it is much easier (and less dangerous!) to inject **faults** virtually to be able to test the behaviour of the controller for boundary (or out-of-boundary) conditions.

Rapid prototyping

Final tests : actual hardware



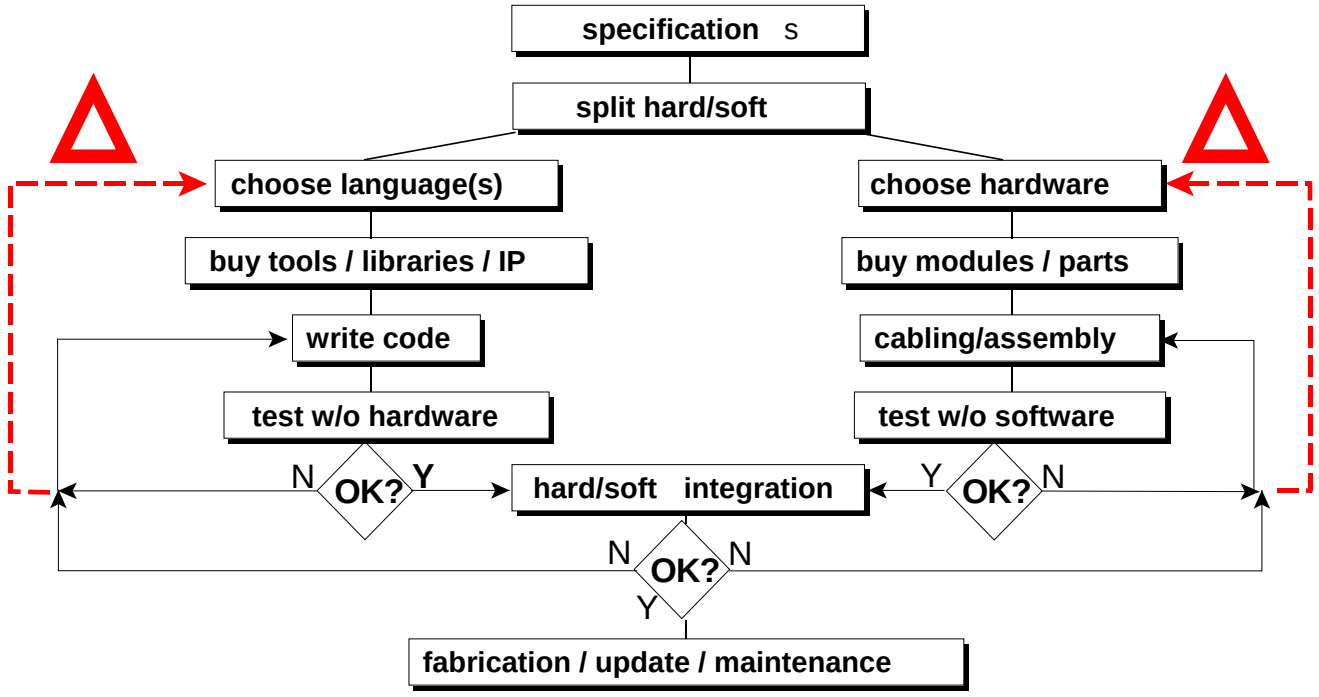
The final stage is of course the test of the debugged controller on the actual process.

The link with the PC can be unplugged, but it is interesting to keep all the functions dealing with the communications for

- updates
- maintenance, diagnostic and repair

Workflow

prototyping of a microcontroller system



The workflow to design a microcontroller begins as usual by careful specifications (remember all previous remarks over the importance of this first stage). In the second stage, it is split into **two parallel branches**, one for the **hardware** and the other one for **software.** (1)

Two further chapters will be devoted to the choices of hardware and software. Here again the "buy-or-build" questions will arise.

Let us insist particularly on the need to test **each module (software or hardware) as exhaustively as possible**, in order to render them "bullet proof". We will see in this course methods to **test the hardware and software separately**, which enables to progress in each branch without being delayed by the development of the other one. We have already seen that rapid prototyping by "real-time boxes" enables to test the software to some extent before developing the final hardware.

A crucial stage is then the integration of the hardware and the software. Problems that will appear at this stage will be more easily solved if the tests have been carefully done in the previous phases.

Debugging is necessarily an iterative process, indicated by the feedback loops on this figure. Try to avoid the red arrows, meaning that you have to go back to fundamental choices, which is time consuming and generates considerable additional costs (remember the "x10" rule of the thumb). This also highlights the importance to invest in good development tools, debugging instruments and training of your staff.

REM :
Historically design teams have worked in a sequential way: develop hardware then write the software. Today the trend is to **parallel development** like in this workflow, or even **co-design** (see lthe last chapter).

(1) this separation is not obvious, e.g. tasks shared by processors and FPGAs and even hardware assistance/services for the operating system

Workflow

modular design

- make the product testable (hard et soft)
  - separate in modules
    - write test for each module
    - foresee physical test points (for probes)
  - progress step-by step
  - keep all traces from the prototypes
    - testing tools
    - test programs
    - to retest a module
    - to design production test
    - for repair and maintenance

It is also necessary to design the product for its testability (hardware as well as software) This will generally be achieved by a **modular design** and step-by-step progression with **intermediate tests points** (physical or virtual).

Each subset must be submitted to test procedures and the hardware and software tools developed for that purpose must be preserved, for example:

- software: write a test program for each function. If a function must be modified later, the complete test protocol can be applied again to validate it

- hardware: generally you will have to design some test set comprising connectors, cables, measuring instruments (including the setup of these)

All those provide a good basis for the future production test, and for repair and maintenance of the equipments.

Workflow

Overview of tools for µC development

CAD Project Management

software

- IDE integrated environment
  - ♦ editor
  - ♦ compiler
  - ♦ linker
  - ♦ debugger
- simulators

hardware

- prototyping
  - ♦ evaluation boards
  - ♦ "starter kits"
- CAO PCB/PLD/ASIC
  - ♦ schematic/VHDL
  - ♦ simulation
  - ♦ placement/routing
  - ♦ measuring instruments
  - ♦ oscilloscopes
  - ♦ logic analysers

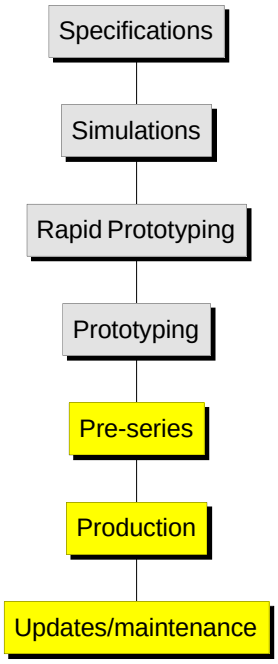
intégr

- monitor/debugger
- emulators ICE or ONCE
- logic analysers
- performance analysers

To conclude, here is an overview of the tools required for the development of classical embedded systems. Most of them will be detailed in the next chapters.

Do not forget that the management of the project itself can be assisted by computer tools to separate the different tasks and teams and coordinate them.

Workflow of embedded systems



These last stages are outside the scope of this course.