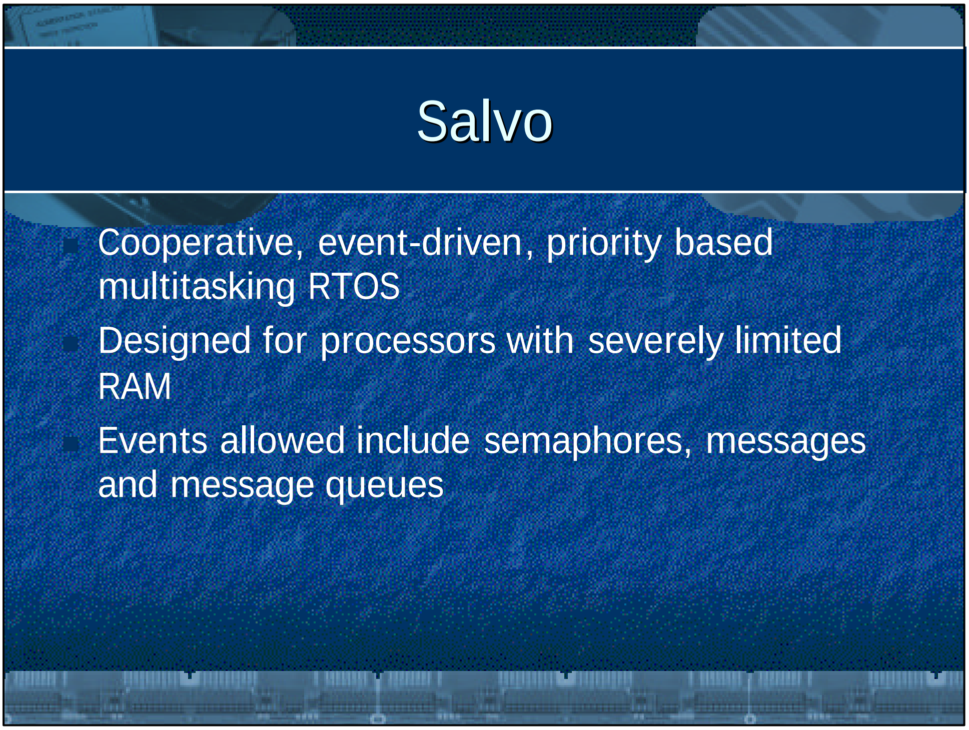
# Real-time Kernel Implantation in a PIC Microcontroller

Yves Gréalou, Pierre Guéant,
Wilfried Jouve, Mickael Le Baillif

ENSEIRB 2004

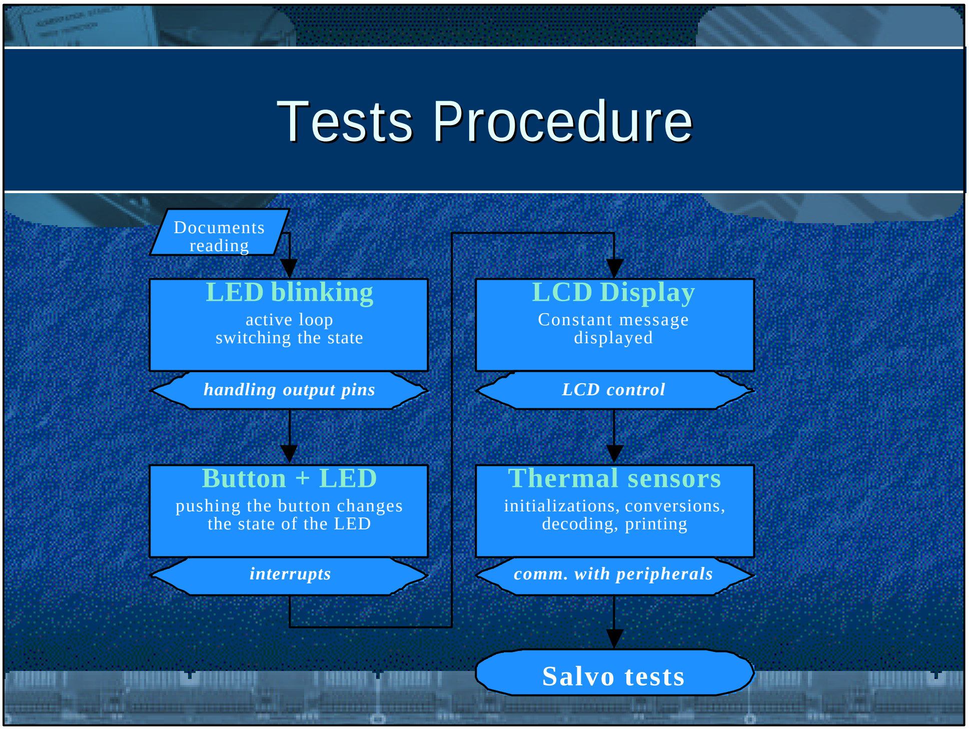
# 1. Introduction

- Project goals
  - Displaying temperatures in real time
- Real time specifications



# Salvo

- Cooperative, event-driven, priority based multitasking RTOS
- Designed for processors with severely limited RAM
- Events allowed include semaphores, messages and message queues



# Tests Procedure

Documents reading

**LED blinking**
active loop switching the state
*handling output pins*

**Button + LED**
pushing the button changes the state of the LED
*interrupts*

**LCD Display**
Constant message displayed
*LCD control*

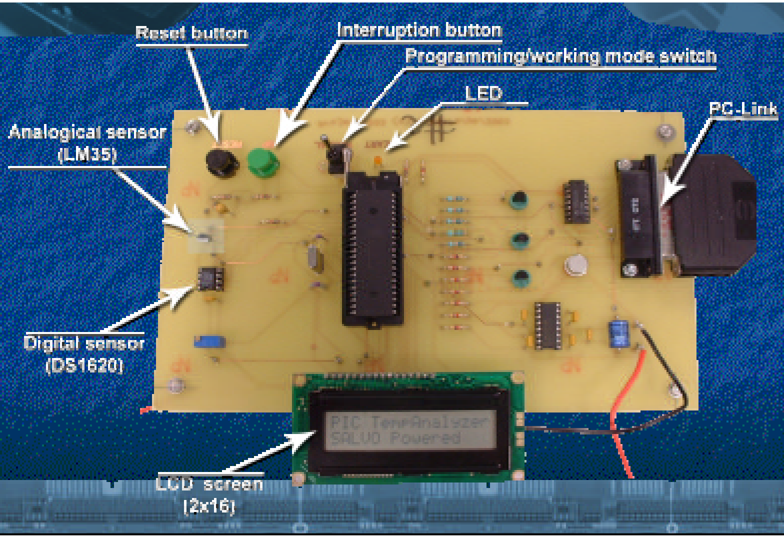
**Thermal sensors**
initializations, conversions, decoding, printing
*comm. with peripherals*

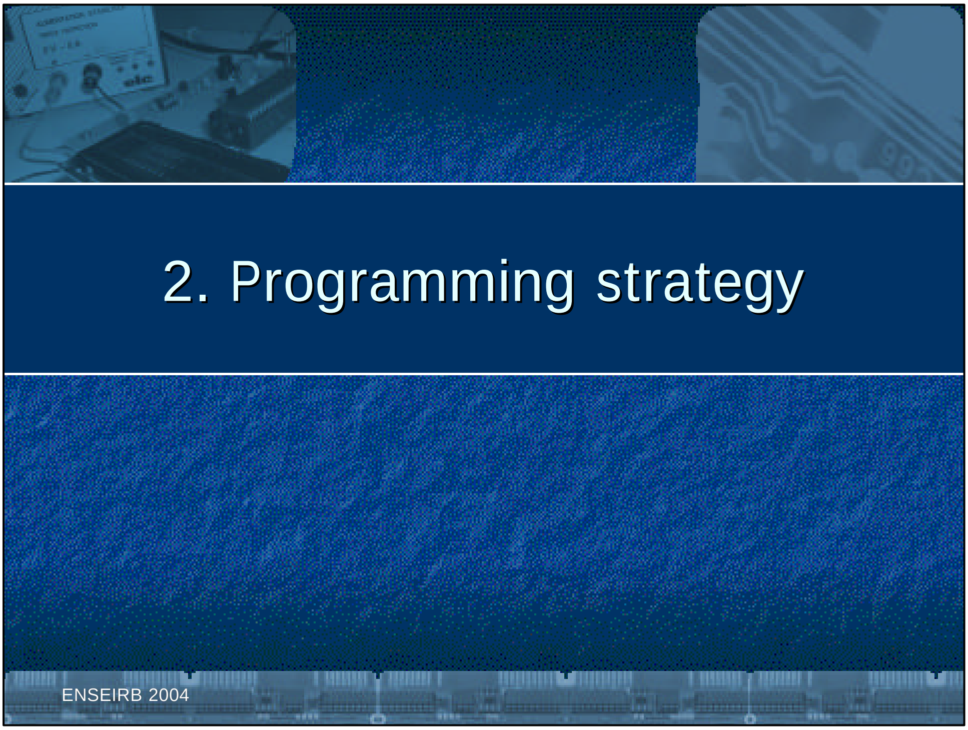
**Salvo tests**

# Implementation Design

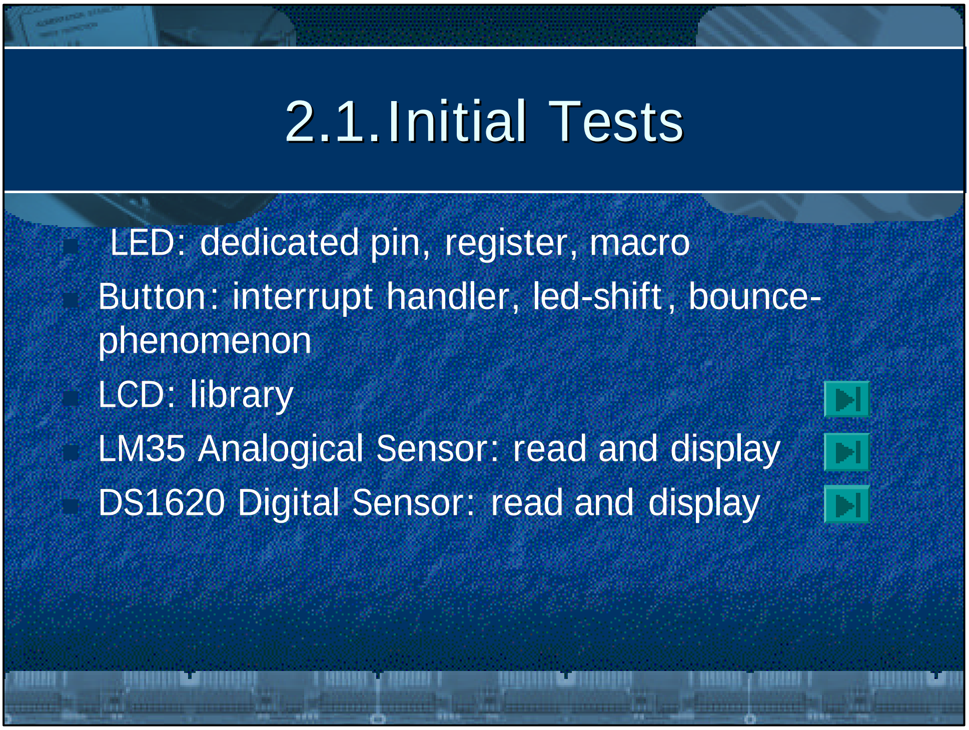
- Listing of required functionalities
- Adaptation with Salvo possibilities
- Functions grouped within 3 tasks, synchronization using 1 semaphore
- Task priority analysis

# Electronic Board



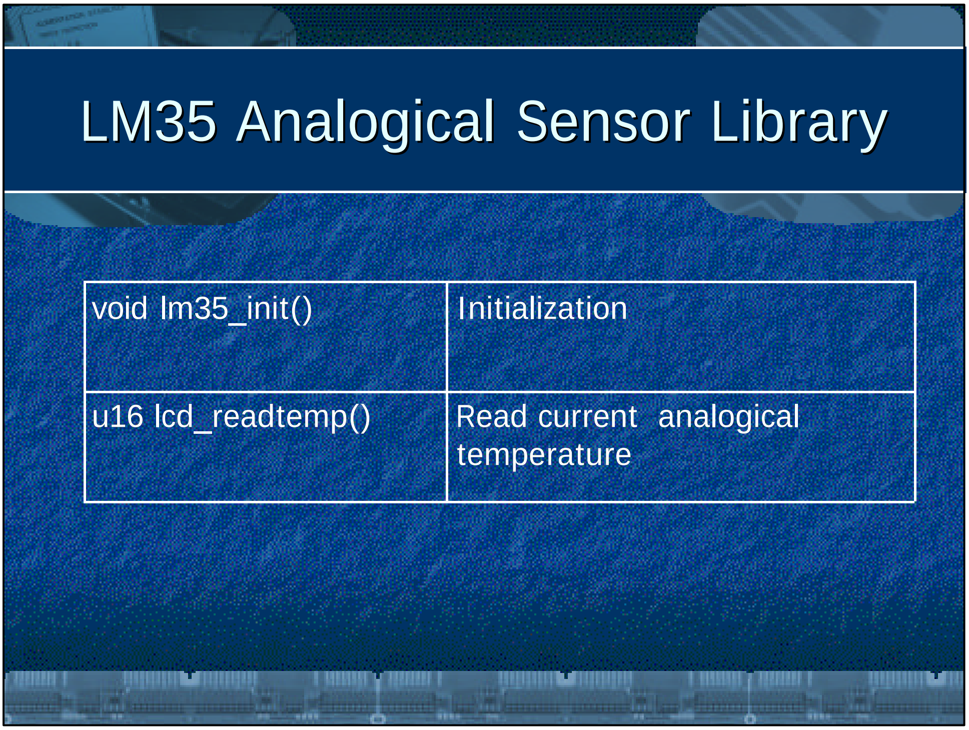


# 2. Programming strategy



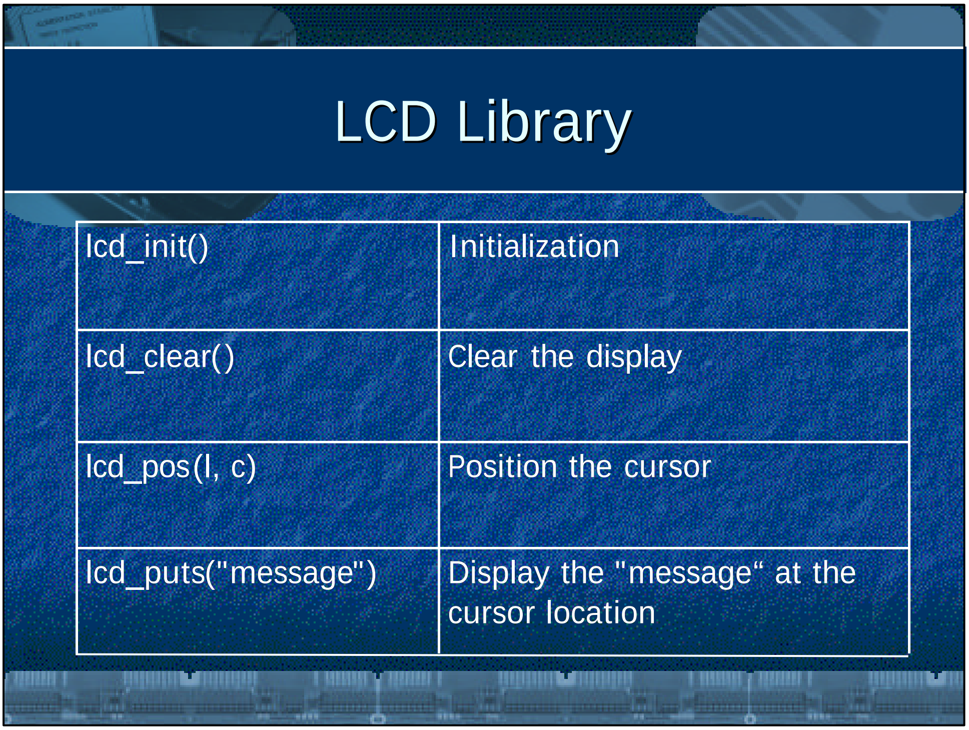
# 2.1.Initial Tests

- LED: dedicated pin, register, macro
- Button: interrupt handler, led-shift, bounce-phenomenon
- LCD: library ⏭
- LM35 Analogical Sensor: read and display ⏭
- DS1620 Digital Sensor: read and display ⏭



# LM35 Analogical Sensor Library

| | |
|---|---|
| void lm35_init() | Initialization |
| u16 lcd_readtemp() | Read current analogical temperature |



# LCD Library

| | |
|---|---|
| lcd_init() | Initialization |
| lcd_clear() | Clear the display |
| lcd_pos(l, c) | Position the cursor |
| lcd_puts("message") | Display the "message“ at the cursor location |

# DS1620 Digital Sensor Library

| | |
|---|---|
| void ds1620_init() | Initialization |
| void ds1620_start() | Start a new conversion |
| u16 ds1620_readtemp() | Read current temperature |

# 2.2.Description of the Tasks

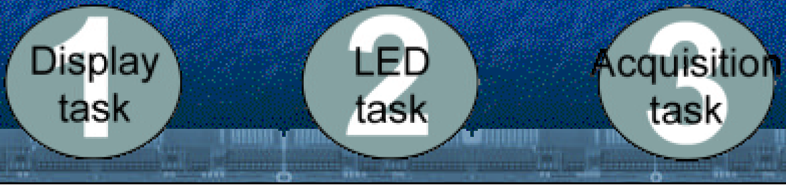
- Display Task
  - Function print()
  - Three print modes: CURRENT, MINI, MAXI
- Temperatures Acquisition Task
  - Function acquisition()
  - Retrieves analogical and digital temperatures computed by the sensors
- LED Blinking Task
  - Function born_to_be_alive ()
  - Switches the LED state

# 2.3.Choice of Priority (1)

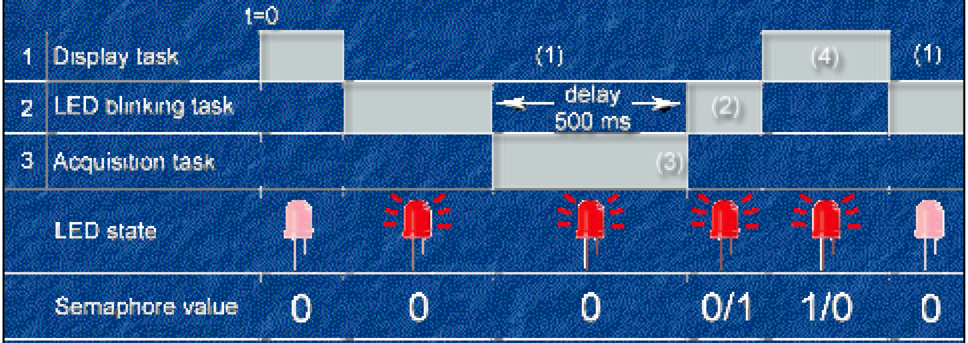
- Call to the scheduler : OS_YIELD, OS_DELAY, OS_WAITSEM, OSSIGNALSEM
- Display Task : the highest priority task
  - Not a background task
  - Waits for the semaphore
- LED Blinking Task
  - LED task establishes the rhythm of the display
  - Releases the semaphore

# 2.3.Choice of Priority (2)

- Temperatures Acquisition Task
  - Acquisition task has to be a background task
  - Minimal and maximal temperatures
- LED Blinking Task Priority > Temperatures Acquisition Task Priority
  - If LED task priority lower => never be scheduled
- Conclusion



# Tasks progression



(1) OS_WaitSem(): The task waits for the semaphore
(2) OSSignalSem(). The task releases the semaphore
(3) OS_Yield(): The scheduler choose a new task
(4) The task Displays new set of temperatures

▨ CPU monopolization

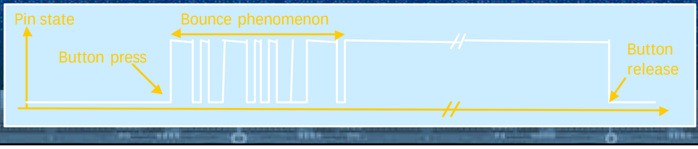
# 3. Development & Experimentation

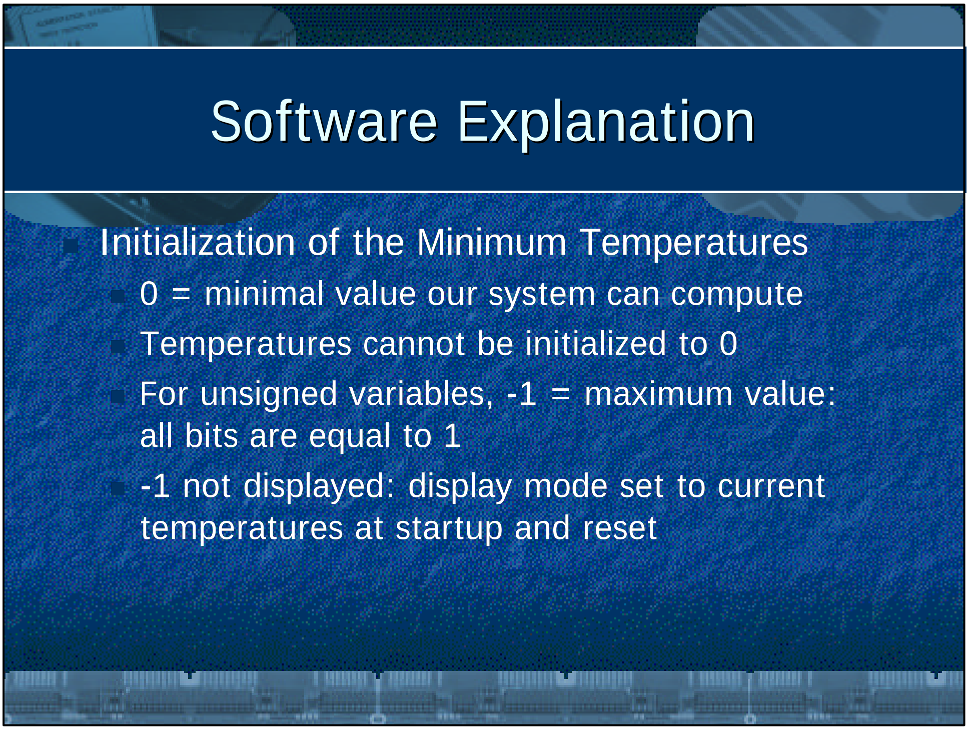
# Development and Experimentation: Details (1)

- Interruption enabling
  - Problem: Start displaying the minimal temperature
  - Cause: Pending interrupt at startup
  - OSEi() reset global interrupt mask, INTE =1 allows external interrupts
  - Solution: enabling interrupts before init_value()

# Development and Experimentation: Details (2)

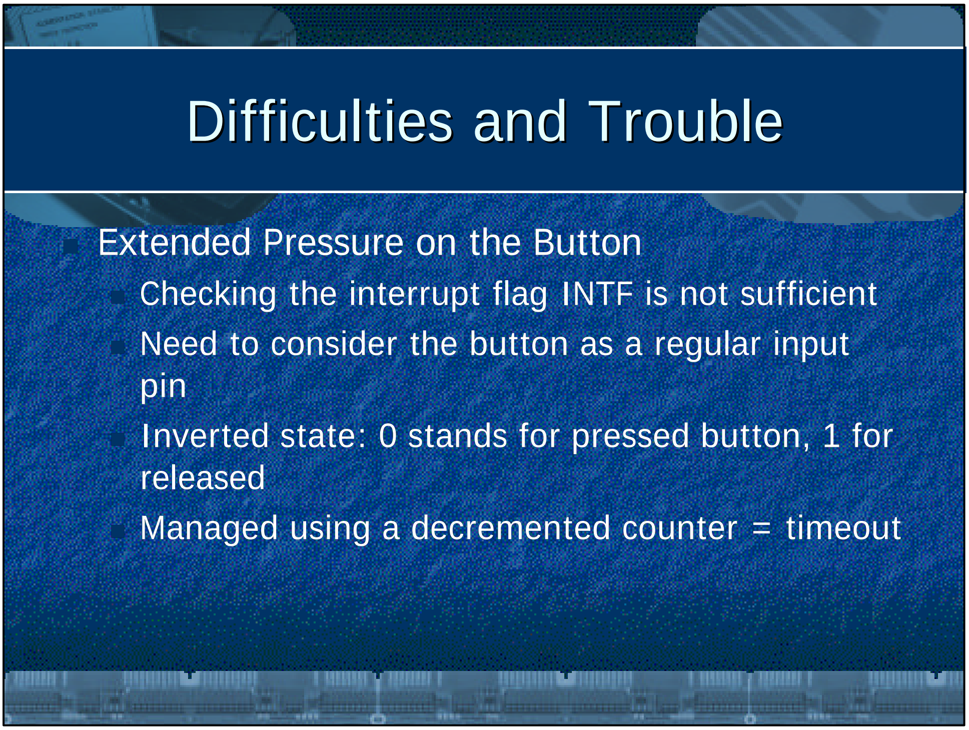
- External Interrupt
  - Goal: changing the display mode
  - Pressing the button: call to the interrupt handler, INTF=1
  - Bounce phenomenon avoidance
  - Right after a request for erasing extrema values, no change in display mode





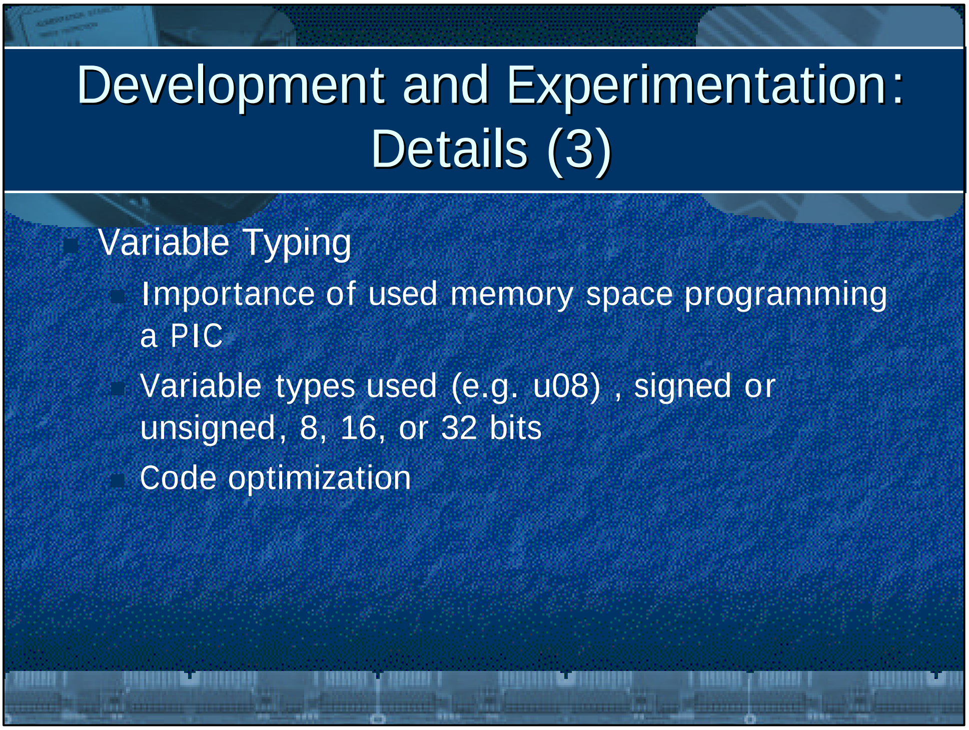
# Software Explanation

- Initialization of the Minimum Temperatures
  - 0 = minimal value our system can compute
  - Temperatures cannot be initialized to 0
  - For unsigned variables, -1 = maximum value: all bits are equal to 1
  - -1 not displayed: display mode set to current temperatures at startup and reset



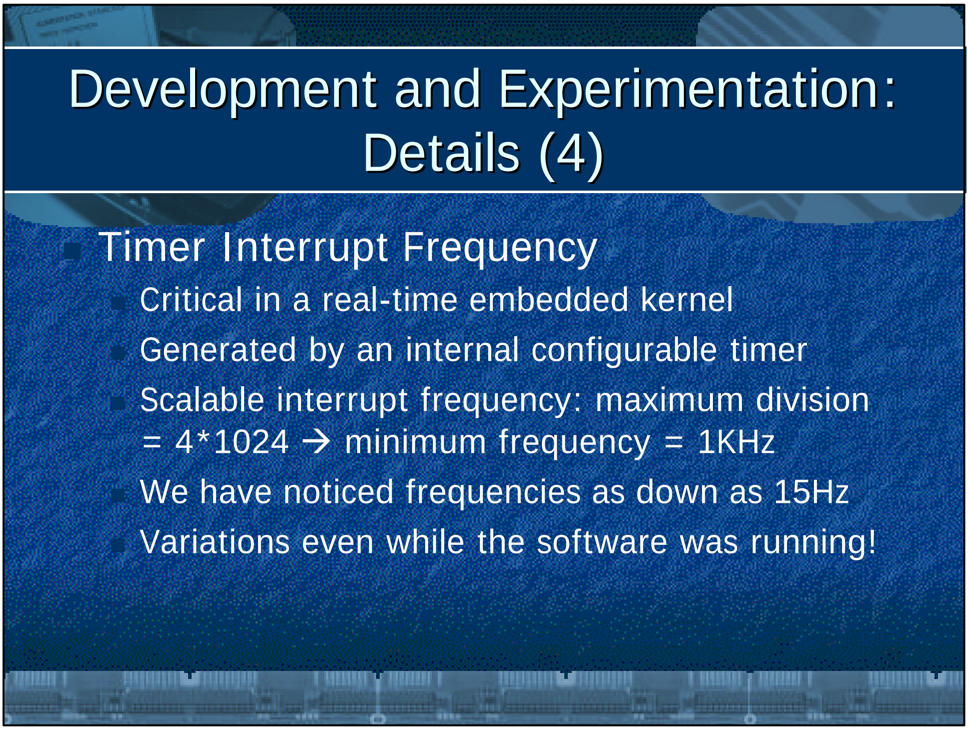
# Difficulties and Trouble

- Extended Pressure on the Button
  - Checking the interrupt flag INTF is not sufficient
  - Need to consider the button as a regular input pin
  - Inverted state: 0 stands for pressed button, 1 for released
  - Managed using a decremented counter = timeout



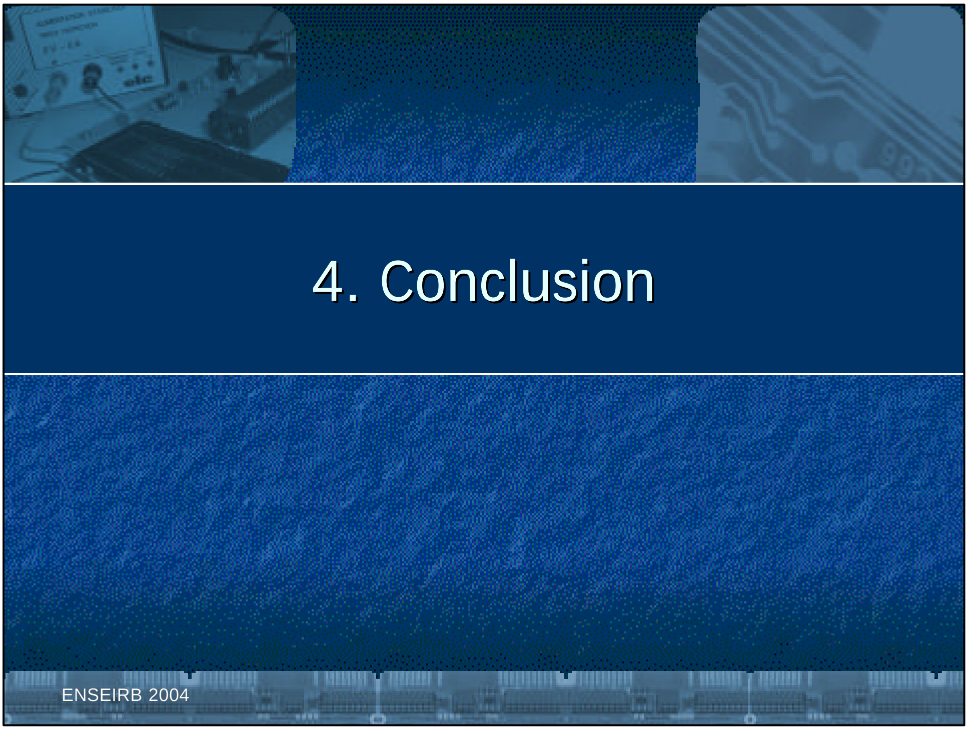
## Development and Experimentation: Details (3)

- Variable Typing
  - Importance of used memory space programming a PIC
  - Variable types used (e.g. u08) , signed or unsigned, 8, 16, or 32 bits
  - Code optimization



## Development and Experimentation: Details (4)

- Timer Interrupt Frequency
  - Critical in a real-time embedded kernel
  - Generated by an internal configurable timer
  - Scalable interrupt frequency: maximum division = 4*1024 → minimum frequency = 1KHz
  - We have noticed frequencies as down as 15Hz
  - Variations even while the software was running!



# 4. Conclusion