Scuola Superiore Sant'Anna di Studi Universitari e di Perfezionamento

Retis Real-Time Systems Laboratory

Form embedded O.S. to Code Generation

Mauro Marinoni [nino@evidence.eu.com]

Retis Lab
Scuola Superiore Sant'Anna

Scilab

Evidence

Aknoledgement

- A special thanks to **Paolo Gai** (Evidence S.r.l.) for the support preparing this presentation.

Objectives

- Embedded O.S.
  - OSEK standard
  - Erika kernel
- Hardware platform
  - FLEX board
  - Demo addon board
- Scilab/Scicos
- Embedded Codegen
  - Structure and implementation
  - Examples

Scuola Superiore Sant'Anna di Studi Universitari e di Perfezionamento

Part I

Embedded O.S.

Scilab

Evidence

Why an embedded O.S. ?

- It reduces the complexity of the application;
- It increases the reusability of the code;
- It simplify the SW debugging;
- It reduces the time to market:
- ...

Why a Real-Time embedded O.S. ?

- An embedded applications tipically presents a lot of interactions with the environment;
- That requires a management of the response time to an external event.

Developement scenario

Typical scenario for an embedded system:

- microcontroller (typically with reduced number instruction)
- lack of resources (especially RAM!!!)
- dedicated HW
- dedicated interaction patterns
  - a microwave oven is **NOT** a general purpose computer

These assumptions leads to different programming styles, and to SW architectures different from general purpose computers.

The footprint problem...

- Considering typical multiprogrammed environments:
  - a full-fledged POSIX footprint is around 1 Mb
  - use of profiles to support subset of the standard
  - a profile is a subset of the full standard that lists a set of services typically used in a given environment
  - POSIX real time profiles are specified by the ISO/IEEE standard 1003.13
- The system we want to be able must fit on a typical system-on-chip memory footprint
  - that is, around 10 Kb of code and around 1 Kb of RAM...

POSIX top-down approach

- POSIX defines a **top-down** approach towards embedded systems API design
  - the interface was widely accepted when the profiles came out
  - these profiles allow easy upgrades to more powerful systems
  - possibility to reuse previous knowledges and code
- PSE51 systems around 50-150 Kbytes
  - that size fits for many embedded devices, like single board PCs
  - **SHaRK** is a PSE51 compliant system

SoC needs bottom-up approaches!

- we would like to have footprint in the order of 1-10 Kb
- the idea is to have a bottom-up approach
- starting from scratch, design
  - a minimal system
  - that provides a minimal API
  - that is able to efficiently describe embedded systems
    - with stringent temporal requirements
    - with limited resources

Results:

- RTOS standards (OSEK-VDX, uITRON)
- 2 Kbytes typical footprint

What is OSEK/VDX?

- It is a standard for an open-ended architecture for distributed control units in vehicles
- The name:
  - **OSEK**: Offene Systeme und deren Schnittstellen für die Elektronik im Kraft-fahrzeug (Open systems and the corresponding interfaces for automotive electronics)
  - **VDX**: Vehicle Distributed eXecutive (another french proposal of API similar to OSEK)
  - **OSEK/VDX** is the interface resulted from the merge of the two projects (`http://www.osek-vdx.org`)
- Motivations:
  - high, recurring expenses in the development and variant management of non-application related aspects of control unit software.
  - incompatibility of control units made by different manufacturers due to different interfaces and protocols.

Objectives

- portability and reusability of the application software
- specification of abstract interfaces for RTOS and network management
- specification independent from the HW/network details
- scalability between different requirements to adapt to particular application needs
- verification of functionality and implementation using a standardized certification process

Advantages

- clear savings in costs and development time.
- enhanced quality of the software
- creation of a market of uniform competitors
- independence from the implementation and standardised interfacing features for control units with different architectural designs
- intelligent usage of the hardware present on the vehicle
  - for example, using a vehicle network the ABS controller could give a speed feedback to the powertrain microcontroller

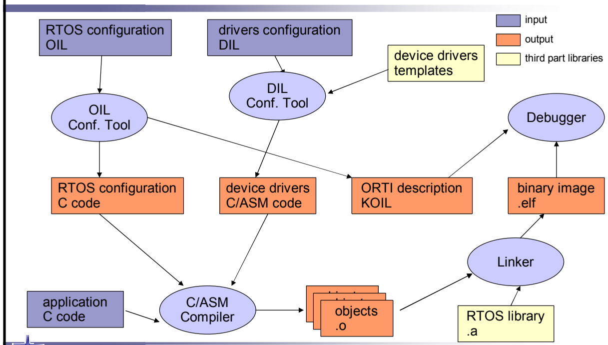
System philosophy

- standard interface ideal for automotive applications
- scalability
  - using conformance classes
- configurable error checking
- portability of software
  - the firmware on an automotive ECU is 10% RTOS and 90% drivers
- **Static is better:**
  - everything is specified before the system runs
  - static approach to system configuration
    - no dynamic allocation on memory
    - no dynamic creation of tasks
    - no flexibility in the specification of the constraints
  - custom languages that helps off-line configuration of the system
    - OIL: parameters specification (tasks, resources, stacks…)
    - KOIL: kernel aware debugging

Support for automotive requirements

- the idea is to create a system that is
  - reliable
  - with real-time predictability
- support for
  - fixed priority scheduling with immediate priority ceiling
  - non preemptive scheduling
  - preemption thresholds
  - ROM execution of code
  - stack sharing (limited support for blocking primitives)
- documented system primitives
  - behavior
  - performance of a given RTOS must be known

Application building process

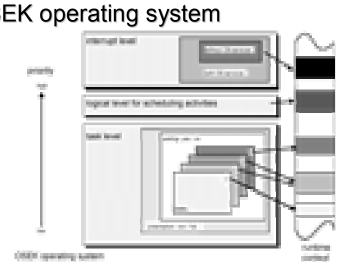


OSEK/VDX standards

- The OSEK/VDX consortium packs its standards in different documents
  - OSEK OS — operating system
  - OSEK Time — time triggered operating system
  - OSEK COM — communication services
  - OSEK FTCOM — fault tolerant communication
  - OSEK NM — network management
  - OSEK OIL — kernel configuration
  - OSEK ORTI — kernel awareness for debuggers

Processing levels

- the OSEK OS specification describes the processing levels that have to be supported by an OSEK operating system

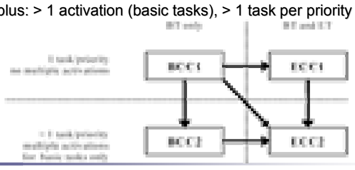


Conformance classes

- OSEK OS should be scalable with the application needs
  - different applications require different services
  - the system services are mapped in Conformance Classes
- a conformance class is a subset of the OSEK OS standard
- objectives of the conformance classes
  - allow partial implementation of the standard
  - allow an upgrade path between classes
- services that discriminates the different conformance classes
  - multiple requests of task activations
  - task types
  - number of tasks per priority

Conformance classes

- There are four conformance classes
  - BCC1
    basic tasks, one activation, one task per priority
  - BCC2
    BCC1 plus: > 1 activation, > 1 task per priority
  - ECC1
    BCC1 plus: extended tasks
  - ECC2
    ECC1 plus: > 1 activation (basic tasks), > 1 task per priority



Basic tasks

- a basic task is
  - a C function call that is executed in a proper context
  - that can never block
  - can lock resources
  - can only finish or be preempted by an higher priority task or ISR
- a basic task is ideal for implementing a kernel-supported stack sharing, because
  - the task never blocks
  - when the function call ends, the task ends, and its local variables are destroyed
  - in other words, it uses a one-shot task model
- support for multiple activations
  - in BCC2, ECC2, basic tasks can store pending activations (a task can be activated while it is still running)

Extended tasks

- extended tasks can use events for synchronization
- an event is simply an abstraction of a bit mask
  - events can be set/reset using appropriate primitives
  - a task can wait for an event in event mask to be set
- extended tasks typically
  - have its own stack
  - are activated once
  - have as body an infinite loop over a WaitEvent() primitive
- extended tasks do not support for multiple activations
  - ... but supports multiple pending events

Scheduling algorithm

- the scheduling algorithm is fundamentally a
  - fixed priority scheduler
  - with immediate priority ceiling
  - with preemption threshold
- the approach allows the implementation of
  - preemptive scheduling
  - non preemptive scheduling
  - mixed

Interrupt service routine

- OSEK OS directly addresses interrupt management in the standard API
- interrupt service routines (ISR) can be of two types
  - Category 1: without API calls simpler and faster, do not implement a call to the scheduler at the end of the ISR
  - Category 2: with API calls these ISR can call some primitives (ActivateTask, ...) that change the scheduling behavior. The end of the ISR is a rescheduling point
- ISR 1 has always a higher priority of ISR 2
- finally, the OSEK standard has functions to directly manipulate the CPU interrupt status

Counters and alarms

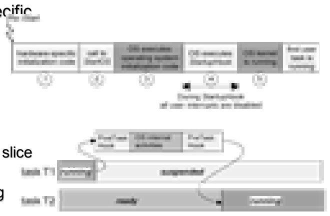
- Counter
  - is a memory location or a hardware resource used to count events
  - for example, a counter can count the number of timer interrupts to implement a time reference
- Alarm
  - is a service used to process recurring events
  - an alarm can be cyclic or one shot
  - when the alarm fires, a notification takes place
    - task activation
    - call of a callback function
    - set of an event

Application modes

- OSEK OS supports the concept of application modes
- an application mode is used to influence the behavior of the device
- example of application modes
  - normal operation
  - debug mode
  - diagnostic mode
  - ...

Hooks

- OSEK OS specifies a set of hooks that are called at specific times
  - StartupHook
    when the system starts
  - PreTaskHook
    before a task is scheduled
  - PostTaskHook
    after a task has finished its slice
  - ShutdownHook
    when the system is shutting down
  - ErrorHook
    when a primitive returns an error



Error handling

- the OSEK OS has two types or error return values
  - standard error
    (only errors related to the runtime behavior are returned)
  - extended error
    (more errors are returned, useful when debugging)
- the user have two ways of handling these errors
  - distributed error checking
    the user checks the return value of each primitive
  - centralized error checking
    the user provides a ErrorHook that is called whenever an error condition occurs
    - macros can be used to understand which is the failing primitive and what are the parameters passed to it

OSEK OIL

- goal
  - provide a mechanism to configure an OSEK application inside a particular CPU (for each CPU there is one OIL description)
- the OIL language
  - allows the user to define objects with properties
    (e.g., a task that has a priority)
  - some object and properties have a behavior specified by the standard
- an OIL file is divided in two parts
  - an implementation definition
    defines the objects that are present and their properties
  - an application definition
    define the instances of the available objects for a given application

OSEK OIL objects

- The OIL specification defines the properties of the following objects:
  - CPU
    the CPU on which the application runs
  - OS
    the OSEK OS which runs on the CPU
  - ISR
    interrupt service routines supported by OS
  - RESOURCE
    the resources which can be occupied by a task
  - TASK
    the task handled by the OS
  - COUNTER
    the counter represents hardware/software tick source for alarms.

OSEK OIL objects (2)

- EVENT
  the event owned by a task. A
- ALARM
  the alarm is based on a counter
- MESSAGE
  the COM message which provides local or network communication
- COM
  the communication subsystem
- NM
  the network management subsystem

OIL example: implementation definition

```
OIL_VERSION = "2.4";

IMPLEMENTATION my_osek_kernel {
[...]
    TASK {
     BOOLEAN [
         TRUE { APPMODE_TYPE APPMODE[]; },
         FALSE
        ] AUTOSTART;
        UINT32 PRIORITY;
        UINT32 ACTIVATION = 1;
        ENUM [NON, FULL] SCHEDULE;
        EVENT_TYPE EVENT[];
        RESOURCE_TYPE RESOURCE[];

        /* my_osek_kernel specific values */
        ENUM [
            SHARED,
            PRIVATE { UINT32 SIZE; }
        ] STACK;
    };
[...]
};
```


OIL example: application definition

```
CPU my_application {
    TASK Task1 {
        PRIORITY = 0x01;
        ACTIVATION = 1;
        SCHEDULE = FULL;
        AUTOSTART = TRUE;
        STACK = SHARED;
    };
};
```


OSEK ORTI

- once having defined the OIL objects and the OS API, there is the need to let the debugger know informations about the kernel and about the application
- ORTI is basically a language that provides the so-called kernel awareness
- benefits
  - OS implementers easily gain support from debugging tools
  - tool vendors can develop only one tool to implement awareness
  - customers have free choice over OSes and and debuggers
- the language is again divided in two parts
  - declaration section
    defines the objects that are present and their properties
  - information section
    describes the objects really present in the system

OSEK ORTI objects

- The ORTI specification defines the properties of the following objects:
  - OS
  - TASK
  - CONTEXT
    subset of the informations saved by the OS for a task
  - STACK
    a stack memory area
  - ALARM
  - RESOURCE
  - MESSAGECONTAINER
    a communication message

ORTI example: declaration section

```
VERSION {
    KOIL = "2.1";
    OSSEMANTICS = "ORTI", "2.1";
};
IMPLEMENTATION OSTEST1_ORTI {
[...]
    TASK {
        ENUM "int" [
            "Not Running (0)" = 0
            "0x1" = 0x1
            "0x2" = 0x2
            "0x4" = 0x4
            "0x8" = 0x8
        ] PRIORITY , "Actual Prio";
        ENUM "unsigned char" [
            "RUNNING"=0,
            "WAITING"=1,
            "READY"=2,
            "SUSPENDED"=3
        ] STATE, "Task State";
        CTYPE "int" CURRENTACTIVATIONS, "Current activations";
    };
[...]
};
```

ORTI example: information section

```
TASK thread0 {
    PRIORITY = "(ERIKA_ORTI_th_priority[0])";
    STATE = "(ERIKA_th_status[0])";
    CURRENTACTIVATIONS = "(1 - ERIKA_th_rnact[0])";
};
```


OSEK COM

- The OSEK COM standard provides an interface for communication inter- and intra- ECU
- main features
  - four conformance classes
  - user can send message objects (defined in the OIL file)
  - message objects can be queued or non queued
  - queued and unqueued message objects
  - one-to-one and one-to-many communication supported
  - support for filtering undesired messages
  - support for transparent network support using IPDUs
  - support for Network Management (OSEK NM)
    - NM is related to the safety and reliability of the network

OSEK NM

- describes node-related (local) and network-related (global) management methods
- services provided
  - initialization of ECU resources, e.g. network interface
  - start-up of the network
  - network configuration
  - management of different mechanisms for node monitoring
  - detecting, processing and signaling of operating states for network and nodes
  - reading and setting of network- and node-specific parameters
  - coordination of global operation modes
    - (e.g. network wide sleep mode)
  - support of diagnosis

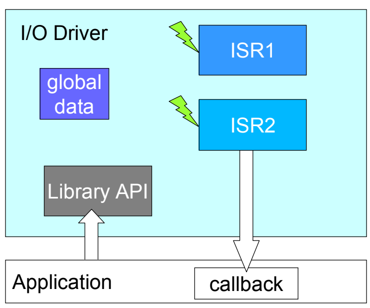
I/O Management architecture

- the application calls I/O functions
- typical I/O functions are non-blocking
  - OSEK BCC1/BCC2 does not have blocking primitives
- blocking primitives can be implemented
  - with OSEK ECC1/ECC2
  - not straightforward
- the driver can use
  - polling
    - typically used for low bandwidth, fast interfaces
    - typically non-blocking
    - typically independent from the RTOS

I/O Management architecture (2)

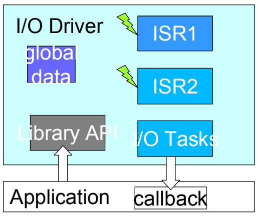
- Interrupts
  - there are a lot of interrupts in the system
  - interrupts nesting often enabled
  - most of the interrupts are ISR1 (independent from the RTOS) because of runtime efficiency
  - one ISR2 that handles the notifications to the application
- DMA
  - typically used for high-bandwidth devices (e.g., transfers from memory to device)

I/O Management: using ISR2



I/O Management architecture (3)

- another option is to use the ISR2 inside the driver to wake up a driver task
- the driver task will be scheduled by the RTOS together with the other application tasks



ERIKA Enterprise

ERIKA Enterprise
- OSEK-like RTOS for minimal embedded systems
- 1-4 Kb ROM footprint
- enhanced scheduling algorithms
- support for Lauterbach Trace32

RT-Druid
- RTOS configuration using OSEK OIL
- schedulability analysis
- integrated in eclipse.org

Supported MCU

availability

Currently available as a product for:
- Microchip dsPIC
- Atmel AVR
- Altera NIOS II (with multi-core support!)

Also available for:
- ARM7TDMI (Samsung KS32C50100, Triscend A7, ST Janus, ST STA2051)
- Hitachi H8 (RCX/Lego Mindstorms)
- Tricore 1
- PPC 5xx (PPC 566EVB)
- C167/ST10 (Ertec EVA 167, tiny/large mem. model)

Erika Enterprise

features

Supported API
- OSEK OS (BCC1, BCC2, ECC1, ECC2)
- OSEK OIL 1.4.1
- OSEK ORTI 2.1.1 for Lauterbach Trace32
- reduced OSEK API for EE Basic

Features
- preemptive fixed priority multithreading
- Immediate Priority Ceiling to avoid priority inversion
- stack sharing to reduce stack RAM usage
- support for multicore devices
- small footprint
- fast execution time

Erika Enterprise

features

Support for guaranteed Real-Time Scheduling
- Fixed Priority (OSEK)
- EDF schedulers and Resource Reservation
- multiprocessor implementation of Fixed priority and EDF (MSRP)

Reduced memory requirements
- stack sharing using preemption thresholds and one shot task model
- stack optimization algorithms available

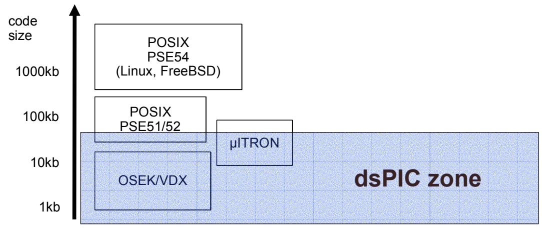
Erika Enterprise: *footprint on dsPIC30*

- FP kernel, monostack, 4 tasks, 1 resource

Code footprint (24-bit instructions):	244 (732 bytes)
ISR2 stub (for each IRQ)	24
IRQ end	23
kernel global functions	67
ActivateTask	43
GetResource + ReleaseResource	42
Task end	45

Data footprint (bytes)	
ROM	26
RAM	42

An RTOS for dsPIC *… but which footprint?*



Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

Part II

Hardware platform

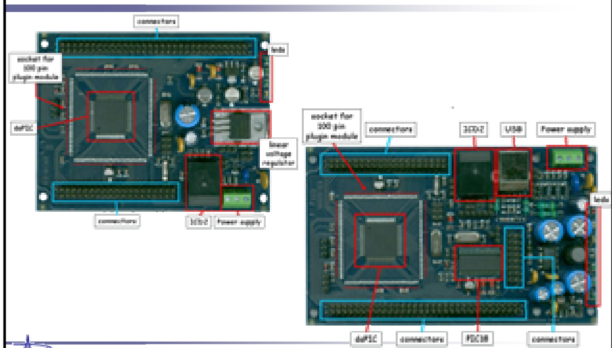
Why another Evaluation Board?

- Tipically, demo boards are:
  - big!
  - limited pin counts MCU
  - most of the pins used for LEDs, buttons, …
  - difficult to expand!
  - poor connection with developement PC
- **FLEX:**
  - small size (7x10 cm)
  - 100 pin dsPIC
  - all pins free on connectors
  - 2.54 pitch, no SMD expertise required!
  - PIC18 for USB connection

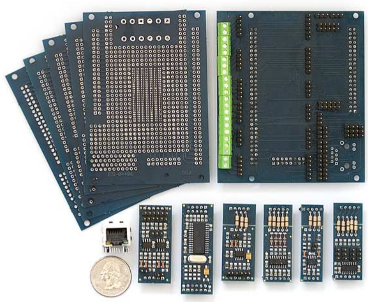
FLEX: other features

- switching power supply
- resettable fuses
- dsPIC programming from USB
- daughter boards (Thru Hole, CAN, SPI, Ethernet, RS232, RS485, RS422, …[other coming soon])
- support fot ERIKA O.S.

FLEX: versions

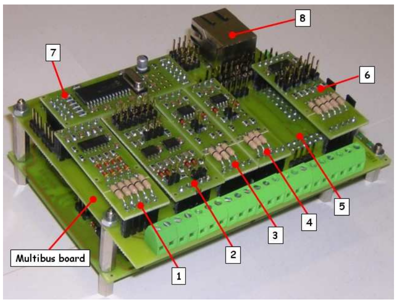


FLEX: *add-on boards*



Available:
- **Thru Hole**
- **Multibus**
  (CAN, SPI, I2C, Serial, Ethernet, Konnex)
- **DemoBoard**

FLEX: Multibus board



1 - Serial port 2
(RS232 / RS422 / RS485 / TP-UART)

2 - Serial port 1
(RS232 / RS422 / RS485)

3 - CAN port 1

4 - CAN port 2

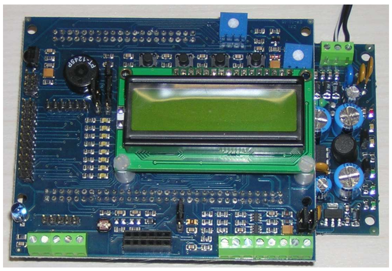
5 - I2C port

6 - SPI port

7 - 10Mbit Ethernet

8 - RJ45 Ethernet

FLEX: Demo Board



LCD 2x16
8 LED
4 buttons
Accelerometer
DAC
Temperature sensor
Light sensor
Infrared I/O
RS232/485/422 socket

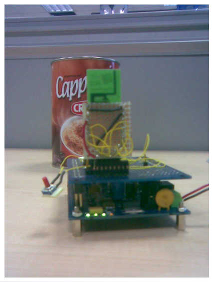
HW demos

- XBee, compass, ultrasound receiver

- TCP/IP demo
- DC Motor identification

HW demos

- Image transfer using 802.15.4
  - FLEX Board
  - ERIKA Enterprise
  - Chipcon 2420 Transceiver
  - CMOS Camera
  - Microchip MAC Layer



Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

Part III

Scilab & Scicos

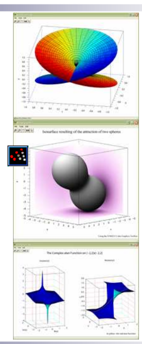
What is Scicos?

Scilab is a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications.

Scilab is OS independent:
* Windows (*including Vista*)
* Unix, Linux,
* Mac OSX
* … your custom O.S. ☺

Scilab is hardware platform independent.

Scilab is an open source software. Since 1994 it has been distributed freely along with the source code via the Internet. It is currently used in educational and industrial environments around the world. Scilab is now the responsibility of the Scilab Consortium, launched in May 2003.

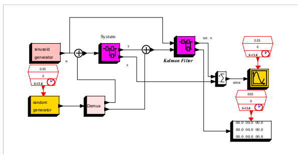


A number of toolboxes are available with the system:

* 2-D and 3-D graphics
* Linear algebra
* sparse matrices
* Polynomials
* Rational functions
* Interpolation
* approximation
* Simulation:
  ODE solver and
  DAE solver
* Scicos: a hybrid dynamic
  systems modeller and
  simulator
* Classic and robust
  control, LMI opt.
* Signal processing

Scicos is …

Scicos is a graphical dynamical system simulator toolbox built in Scilab.



With Scicos you can create block diagrams to model and simulate the dynamics of hybrid systems, control real system in real time with Scicos Hardware In the Loop (Scicos-HIL) and compile your models into executable code for faster simulation and ***stand alone embedded applications.***

With Scicos you can:

* Graphically model, compile, and simulate dynamical systems
* Combine continuous and discrete-time behaviours in the same model
* Select model elements from Palettes of standard blocks
* Program new blocks in C, Fortran, or Scilab Language
* Run simulations in batch mode from Scilab environment
* Generate C code from Scicos model using a Code Generator
* Run simulations in real time with real devices using Scicos-HIL
* Generate hard real-time control executables with Scicos-RTAI
* Simulate digital communications systems with Scicos-ModNum
* Use implicit blocks developed in the Modelica language

Scicos is used for signal processing, control systems, queuing systems, and to study physical and biological systems.
New extensions allow generation of component based physical modelling of electrical and hydraulic circuits using the Modelica language.

Scilab: technology roadmap

- **Objectives (2008 – 2011):**
  Planned major technical developments:
  - Graphics, Scilab GUI and GUI builder
  - Scicos industrialization (GUI, quality,...)
  - Documentation
  - New kernel, 64 bits and 128 bits technology
  - Improvement and updating of algorithms: control, signal processing, identification,...
- Excellence domains:
  - Interoperability (with standard scientific software) and services architecture
  - HPC (High Performance Computing), Grid Computing, parallel computing, multi-core
  - C code generation, embedded systems
  - R & D: developments in collaboration with research
- **Scilab 5.0 (2008):**
  - New license: CeCILL (GPL2 compatible) and GPL2.
  - Modularization
  - New and graphics rendering GUI

What do you need for your embedded RT applications?

Faster
- Reduced development time = Minimum Time To Market
  - Better tools
  - Access to source code and development chain

Better
- High quality, flexibility, market superiority
  - Knowledge
  - Collaboration
  - Independence

Cheaper
- Competitive
  - No patent
  - No royalties
  - No hidden costs
  - Protected by OS licenses

Scilab / Scicos

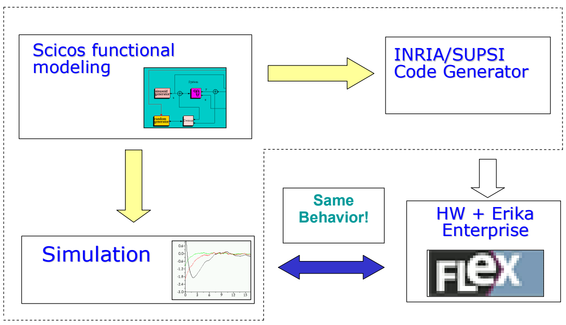
For modeling and Simulation
- Scilab: Scilab language (script)
- Scilab: integration with other programming languages (C/C++, Java, FORTRAN, etc.)
- Scicos: Scicos diagram (visual programming)
- Scicos: integration with other simulation platform (Modelica, GHDL, etc)

But also for embedded applications

Real Time simulation with real plants
Scicos Hardware In the Loop
- Scicos-HIL: the Scicos simulator executes the control section in real time and uses data acquisition cards for the connection with the real plant

Code Generation from Scicos diagram
- Scicos internal GP code generator
- Scicos-RTAI: for Linux RTAI systems
- Scicos-FLEX: for micro controllers and DSPs

Code Generation with Scilab/Scicos



Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

Part IIII

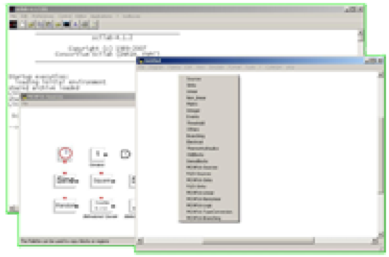
Code Generator

Code Generator

- It converts a **Scicos** superblock into a **dsPIC** application ready to be executed by the MCU.
- It supports single-rate single-task controller.
  - *One and only one* source of time is required.

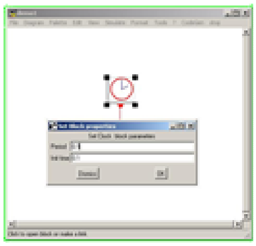
dsPIC and FLEX support

- A set of palette for the specific Hardware has been produced.



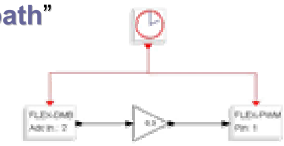
Code Execution - Time

- The **Scicos superblock** is mapped to an **Erika periodic** task.
- The task is executed with a period which is equal to the Scicos Timesource period.
  - Some tricks are needed in order to improve performancies.



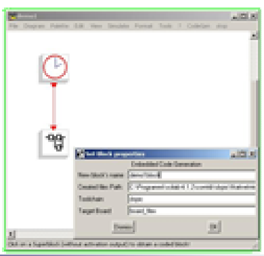
Code – Execution Order

- To each block is connected a function
  - **Init**, **IoOut** or **Close** depending on the system status
- The application is executed block by block (function by function) following the "**data path**"

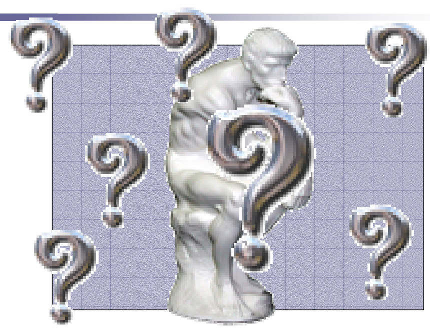


Generation Parameters

- Some parameters are required by the generation engine.



Questions



Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

Mauro Marinoni
[nino@evidence.eu.com]

- Retis:
  - http://retis.sssup.it
- Evidence:
  - http://www.evidence.eu.com
- Scilab:
  - http://www.scilab.org

Scilab

Evidence