Model-based design for embedded systems with CPAL

Tingting Hu, Nicolas Navet

University of Luxembourg – FSTC, Esch-sur-Alzette, Luxembourg

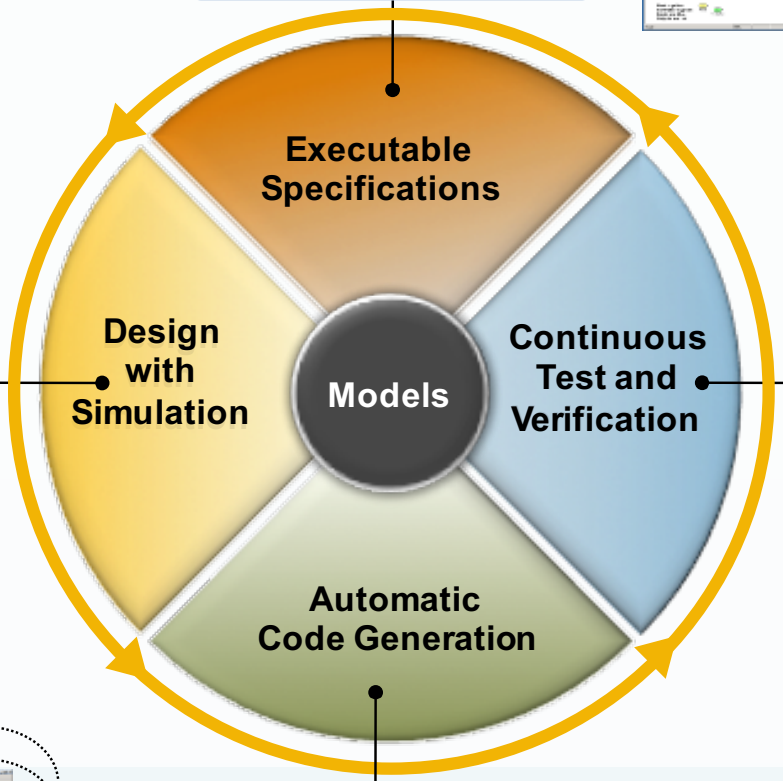
October 4th, 2017, Turin, Italy

Table of Content

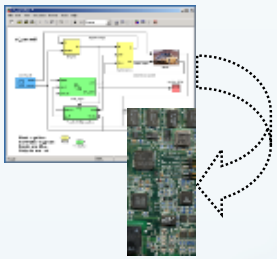
Model-based design

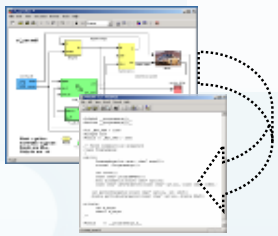
Executable models
- Unambiguous
-"One Truth"

Simulation
- Reduces "real" prototypes
- Iterative "what-if" analysis



Test with Design
- Detects errors earlier





Automatic code generation
- Minimizes coding errors

Source: Analog Device

Domain specific language

Model-based design, when coupled with domain-specific language, permits to achieve higher software productivity and obtain trustworthy system.

- General purpose languages suppose to be used across domains
- A DSL captures semantics specific to a particular domain
- Less comprehensive than GPL, but more expressive in domain knowledge
- Reduce program complexity

Why a new DSL: CPAL

General-purpose programming languages do not offer all the right abstractions for today's real-time embedded systems

- scheduling periodic activites
- time as first class citizen
- safe inter-process communication
- native support for finite-state machines
- high-level interfaces to I/Os
- support for timing and formal verification

Why a new DSL: CPAL

Synchronous languages, such as Esterel, Lustre, Signal, for reactive systems

- Impose constraints and specific programming style
  $\Rightarrow$ initial learning curve steep
- Some are actually Architecture Description Language (ADL), e.g. Prelude, Giotto
  $\Rightarrow$ a different language should be used for development
- Offers formal proof support in both the time domain and value domain
  $\Rightarrow$ Suitable for critical applications

Cyber physical action language

- C-like
- Interpreted language, offers better code portability
- native support for Finite State Machine (mealy-FSM)
- built-in notation of time:
  - period, offset, activation time, execution time, execution jitter, deadline, etc
- scheduling policies:
  - FIFO, Fixed Priority Non-preemptive (FPNP), Early Deadline First Non-preemptive (EDFNP)
- Easy access to I/Os in the model through high-level hardware abstraction
- Code-generation is currently under investigation

A few exemplar use cases

- Development of the SOME/IP SD automotive protocol on top of Ethernet used in a study with Daimler Cars
- a smart parachute for UAVs
- IoT application in the design of a smart mobility system for two and three-wheelers implemented on an ARM mbed IoT board, with IBM Watson IoT platform as cloud backend
- CPAL is used by ONERA to simulate and prototype a code-upload protocol for the avionics domain.
- CPAL is used to simulate TTEthernet and study by fault-injection its robustness to transient failures
- etc . . .

CPAL sample

```
processdef P(params) {
  common {
    code
  }

  state Warning {
    code
  }
  on (cond) {code} to Alarm_Mode;
  after (time) if (cond) to Normal_Mode;

  finally {
    code
  }
}

process P: inst[period,offset][cond](args);

@cpal:time:inst{
    annotation code
}
```

The annotation mechanism

Timing annotation can be specified at different granularity

- Globally:
  - `@cpal:time`
  - scheduling policy and parameters (priorities, deadlines)
  - executed once at the simulation startup
- Instance-specific
  - `@cpal:time:inst`
  - timing properties regarding a particular process instance
  - executed every time an instance is activated
- Named-block
  - `@cpal:time {block_name.execution_time=}`
  - timing properties regarding any named-block, e.g a state
  - executed every time the referred named-block is executed

Two execution modes

Simulation mode

- Execution is as fast as possible (e.g. periods are not respected)
- Code executed in zero time - except if stated otherwise with timing annotations
- CPAL interpreter is hosted by an OS
- No access to real I/Os

Two execution modes

Simulation mode

- Execution is as fast as possible (e.g. periods are not respected)
- Code executed in zero time - except if stated otherwise with timing annotations
- CPAL interpreter is hosted by an OS
- No access to real I/Os

Suitable for development

Two execution modes

Simulation mode

- Execution is as fast as possible (e.g. periods are not respected)
- Code executed in zero time - except if stated otherwise with timing annotations
- CPAL interpreter is hosted by an OS
- No access to real I/Os

Real-time mode

- Real-time execution
- Code (instructions, read/write I/Os) takes time to execute - depends on the platform
- CPAL can be executed on bare hardware or hosted by an OS

Suitable for development

Two execution modes

Simulation mode

- Execution is as fast as possible (e.g. periods are not respected)
- Code executed in zero time - except if stated otherwise with timing annotations
- CPAL interpreter is hosted by an OS
- No access to real I/Os

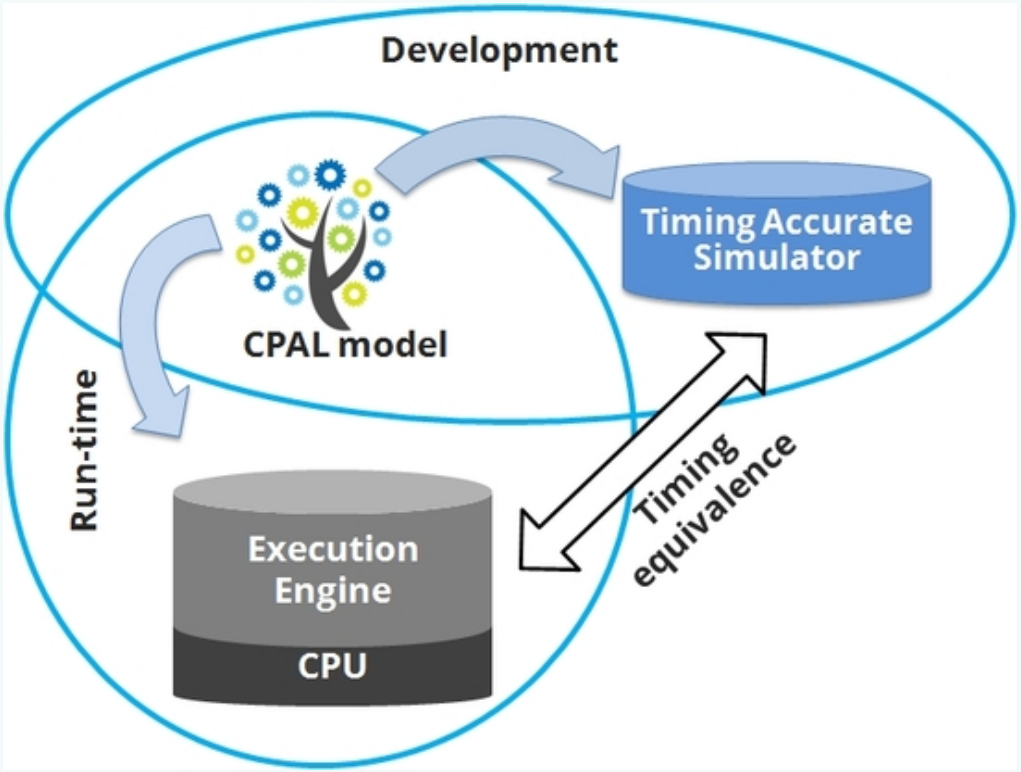
Suitable for development

Real-time mode

- Real-time execution
- Code (instructions, read/write I/Os) takes time to execute - depends on the platform
- CPAL can be executed on bare hardware or hosted by an OS

Suitable for deployment

Two execution modes



Execution order of processes remains the same in simulation and in real-time mode under FIFO scheduling policy

The interpreter

- Signle interpreter:
  - Ready for use in both simulation mode and real-time mode
  - Timing annotation will be ignored in real-time mode, except scheduling policies and parameters
  - –stats option: monitoring the WCET of processes at run-time
- Multi-interpreter:
  - multiprocessors, multicores, or distributed environment
  - One interpreter for each resource
  - Currently just supported in the simulation mode

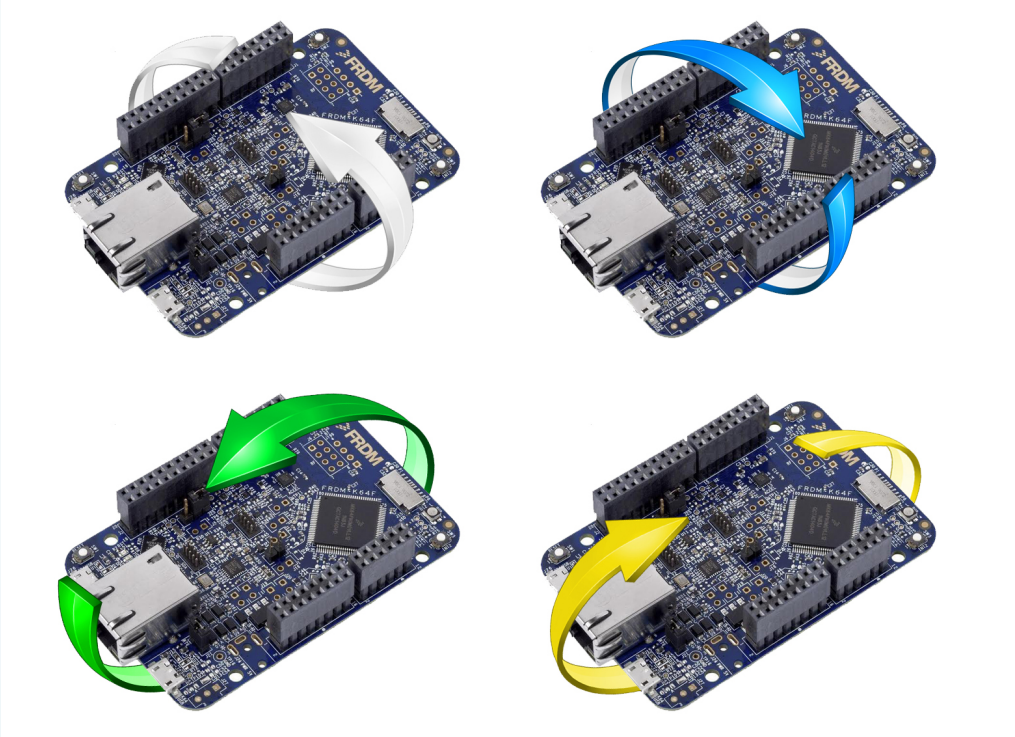
Supported platforms

Platforms	Execution mode	Access to HW?	Executable
Windows 32/64bit	Simulation	No	cpal_interpreter
Embedded Windows 32/64bit	Real-time and Simulation	No	cpal_interpreter_winmbed
Linux 64bit	Simulation	No	cpal_interpreter
Embedded Linux 64bit	Real-Time and Simulation	Yes	cpal_interpreter_linuxmbed
Mac OS X	Simulation	No	cpal_interpreter
Freescale FRDM-K64F	Real-Time	Yes	NA, no OS, image is uploaded
Raspberry Pi	Real-Time and Simulation	Yes	cpal_interpreter_raspberry

Use case

CPAL for the development and execution of real-time applications

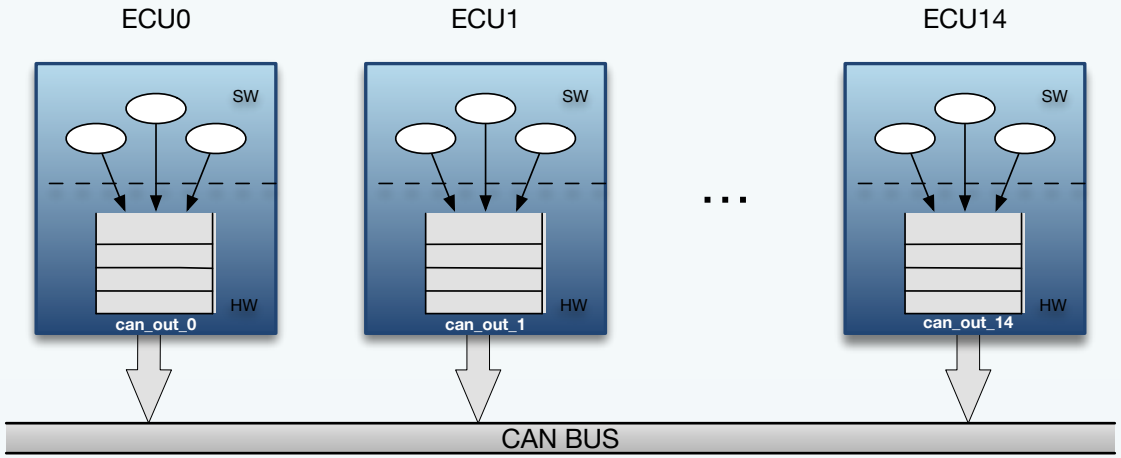
Control the LED color according to the orientation of the FRDM-K64F board, leveraging the on-board accelerometer.



Use case

CPAL as a modeling and timing-accurate simulation environment

A distributed system based on the Controller Area Network (CAN) consists of 15 nodes, each subject to typical automotive traffic.



Each node is associated with an output buffer: FIFO queue or priority queue. Given the traffic, which assures better real-time performance, e.g. schedulability?

Main components of CAN model

The multi-interpreter is handy for the modeling of distributed system

- CAN controller, in particular the transmission side
- Individual nodes: the CAN controller + the application tasks
  - `ecu0.cpal, ecu1.cpal, ..., ecu14.cpal`
- The CAN network
  - `can-network-fifo.cpal, can-network.cpal`
- System & dataflow configuration for the multi-interpreter

Multi-interprerter simulation