2st Italian Workshop on
Embedded Systems (IWES 2017)

# HEPSYCODE-RTMC: a Real-Time and Mixed Criticality Extensions for a System-Level HWSW Co-Design Methodology

Author:

**Vittoriano Muttillo**, Vincenzo Stoico, Daniele Ciambrone, Giacomo Valente, Luigi Pomante

vittoriano.muttillo@graduate.univaqit, vincenzo.stoico@student.univaq.it, daniele.ciambrone@student.univaq.it
giacomo.valente@graduate.univaq.it, luigi.pomante@univaq.it

*University of L'Aquila*
*Center of Excellence **DEWS***
*Department of Information Engineering, Computer Science and Mathematics **DISIM***

# Summary

1. Introduction
2. Mixed-Criticality Scenario
3. Mixed-Criticality Classification
5. ESL Reference Methodology
6. HepsyCode-RTMC
7. Conclusion and Future Works

# 1.

# Introduction

"Brief Introduction to Embedded and Mixed Criticality Systems"

# Mixed-Criticality Embedded Systems

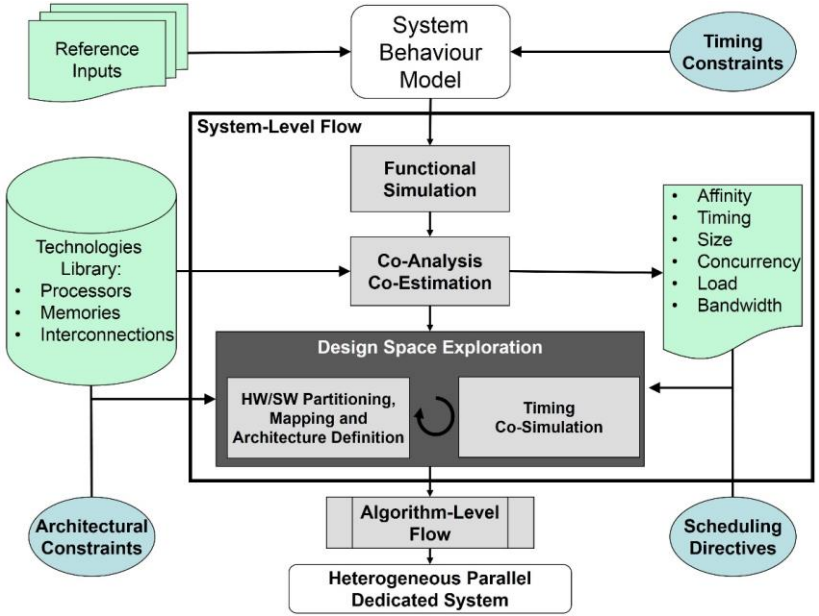
- The growing complexity of embedded digital systems based on modern System-on-Chip (SoC) adopting explicit heterogeneous parallel architectures has radically changed the common design methodologies.

- HW/SW co-design methodologies are of renovated relevance

- A growing trend in embedded systems domain is the development of mixed-criticality systems where multiple embedded applications with different levels of criticality are executed on a shared hardware platform (i.e. Mixed-Criticality Embedded Systems)

# Goals

- This work focus on a Framework (and related tool) for modeling, analysis and validation of mixed critical systems, through the exploitation of an existing "Model-Based Electronic System Level (ESL) HW/SW Co-Design" methodology (called Hepsycode), improved consider both real-time (RT) and mixed-criticality (MC) requirements




Hepsy Code

www.hepsycode.com

# 2.

# Mixed-Criticality Scenario

"Criticality is a designation of the level of assurance against failure needed for a system component"

# MCS State-Of-The-Art Model

- Almost 200 papers treating of the scheduling of MCS have been referenced in Burns and Davis* paper, and tens of related papers are still published every year. Most of the works about MCS published by the real-time scheduling research community are based on a **model proposed by Vestal*** paper.

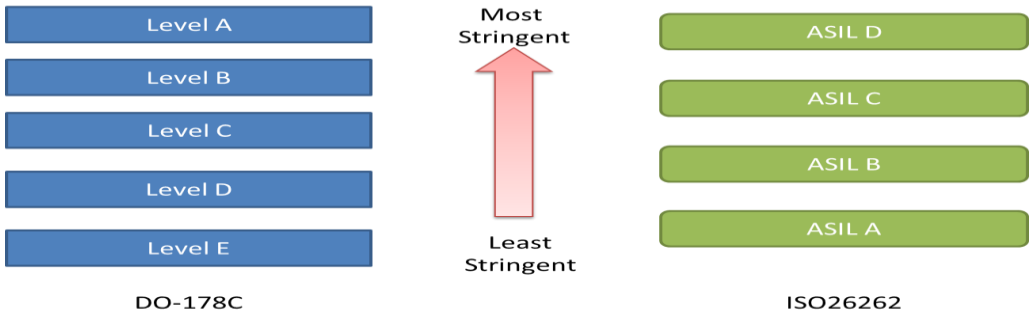
- This model assumes that the system has several modes of execution, say **modes {1, 2, … , L}**. The application system is a **set of real-time tasks**, where each task $\tau_i$ is characterized by a period $T_i$ and a deadline $D_i$ (as in the usual real-time task model), an assurance level $l_i$ and a set of **worst-case computational estimates** $\{C_{i,1}, C_{i,2}, \dots, C_{i,l_i}\}$, under the assumption that $C_{i,1} \leq C_{i,2} \leq \dots \leq C_{i,l_i}$

- The different WCET estimates are meant to model estimations of the **WCET at different assurance levels**. The worst time observed during tests of **normal operational scenarios** might be used as $C_{i,1}$ whereas at **each higher assurance level** the subsequent estimates $\{C_{i,2}, \dots, C_{i,l_i}\}$ are assumed to be obtained by more conservative **WCET analysis techniques**.

** Burns, A, Davis, R.I.: "Mixed Criticality Systems - A Review", University of York, 4 March 2016.*
*** S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," Real-Time Systems Symposium (RTSS) 28th IEEE International on, Tucson, AZ, 2007, pp. 239-243.*

# Integrity Level

- Most safety standards use the concept of an **integrity level**, which is assigned to a system or a function. This level will be based on an initial analysis of the consequences of software going wrong. Both standards have clear guidance on how to identify integrity level.
  - **DO-178C** has **Software Development Assurance Level (DAL)**, which are assigned based on the outcome of "anomalous behavior" of a software component – **Level A** for "Catastrophic Outcome", **Level E** for "No Safety Effect".
  - **ISO26262** has **ASIL (Automotive Safety Integrity Level)**, based on the exposure to issues affecting the controllability of the vehicle. ASILs range from **D** for the highest severity/most probable exposure, and **A** as the least.



# Safety Assurance Standards

- **GENERAL** (IEC-61508) based on **SIL (Safety Integrity Level)**: Functional safety standards (of electrical, electronic, and programmable electronic)
  - **AUTOMOTIVE** (ISO26262) based on **ASIL (Automotive Safety Integrity Level)** (Road vehicles - Functional safety)
  - **NUCLEAR POWER** (IEC 60880-2)
  - **MEDICAL ELECTRIC** (IEC 60601-1)
  - **PROCESS INDUSTRIES** (IEC 61511)
  - **RAILWAY** (CENELEC EN 50126/128/129])
  - **MACHINERY** (IEC 62061)

- **AVIONIC** based on **DAL (Development Assurance Level )** related to ARP4761 and ARP4754
  - DO-178B (Software Considerations in Airborne Systems and Equipment Certification)
  - DO-178C (Software Considerations in Airborne Systems and Equipment Certification, replace DO-178B)
  - DO-254 (Airborne - Design), similar to DO-178B, but for hardware
  - DO-160F (Airborne - Test)

- **MEDICAL DEVICE**
  - FDA-21 CFR
  - IEC-62304

# 3.

# Mixed-Criticality Classification

*"A major industrial challenge arises from the need to face cost efficient integration of different applications with different levels of safety and security on a single computing platform in an open context"*

# MCS Classification

**Separation technique:**

- SW separation: scheduling policy, partitioning with HVP, NoC
- HW separation: one task per core, one task on HW ad hoc (DSP, FPGA), spatial partitioning with HVP, NoC

➢ **HW:**

- Temporal isolation: Scheduling HW
- Spatial isolation: separated Task on dedicated components

➢ **Single processor:**

- Temporal isolation: Scheduling policy with SO, RTOS, or HVP
- Spatial isolation : MMU, MPU, HVP Partitioning

➢ **Multi-processor (MIMD)**

- Architecture: shared memory systems, UMA (SMP), NUMA, distributed systems, NoC
- Temporal isolation : Scheduling policy con SO, RTOS, or HVP
- Spatial isolation : MMU, MPU, HVP partitioning

**Tecnologies:**

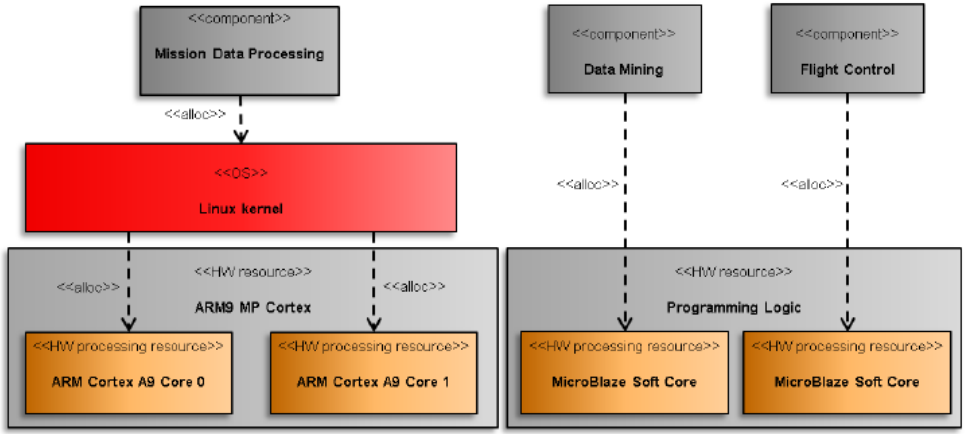
- HW: DSP, FPGA, HW ad hoc, Processor
- SW: OS, RTOS, HVP, Bare-metal
- PROCESSORI: LEON3, ARM, MICROBLAZE
- HVP: PikeOS, Xtratum, Xen
- RTOS: eCos, RTEMS, FreeRTOS, Threadx, VxWorks, Erica
- OS: Linux

| Separation Technique | HW | Single core | Multi-core |
|---|---|---|---|
| Spatial | 0-level scheduling [10] | 0-level scheduling [11][16] | 0-level scheduling [15][16] |
| | | 1-level scheduling [2][5][10][13][16] | 1-level scheduling [4][9][15][16] |
| | | 2-level scheduling [6][11] | 2-level scheduling [3][4][6][7] [8] [9][14] |
| Temporal | 0-level scheduling [10] | 0-level scheduling [11][16] | 0-level scheduling [15][16] |
| | | 1-level scheduling [1][2][10][13] [16] | 1-level scheduling [4][9][12][15][16] |
| | | 2-level scheduling [6][11] | 2-level scheduling [1][4][6][7] [8] [9][14] |

# Multi-core Implementation

## EMC² WP2 - *4-Copter Demonstrator* [16]

- Flight and Position control
  - Execution on Soft-Cores in FPGA
  - Bare metal, no OS support
  - Interfaces for I2C, PPM and GPIO used
- Object tracking
  - Execution on Dual ARM-Core
  - Needs Linux as OS
  - Multimedia Libraries
  - Needs interfaces  USB und Network




Xilinx Zynq 7020:

- ARM dual-core Cortex-A9 (866MHz)
- Artix-7 FPGA (85k Logik Zellen)

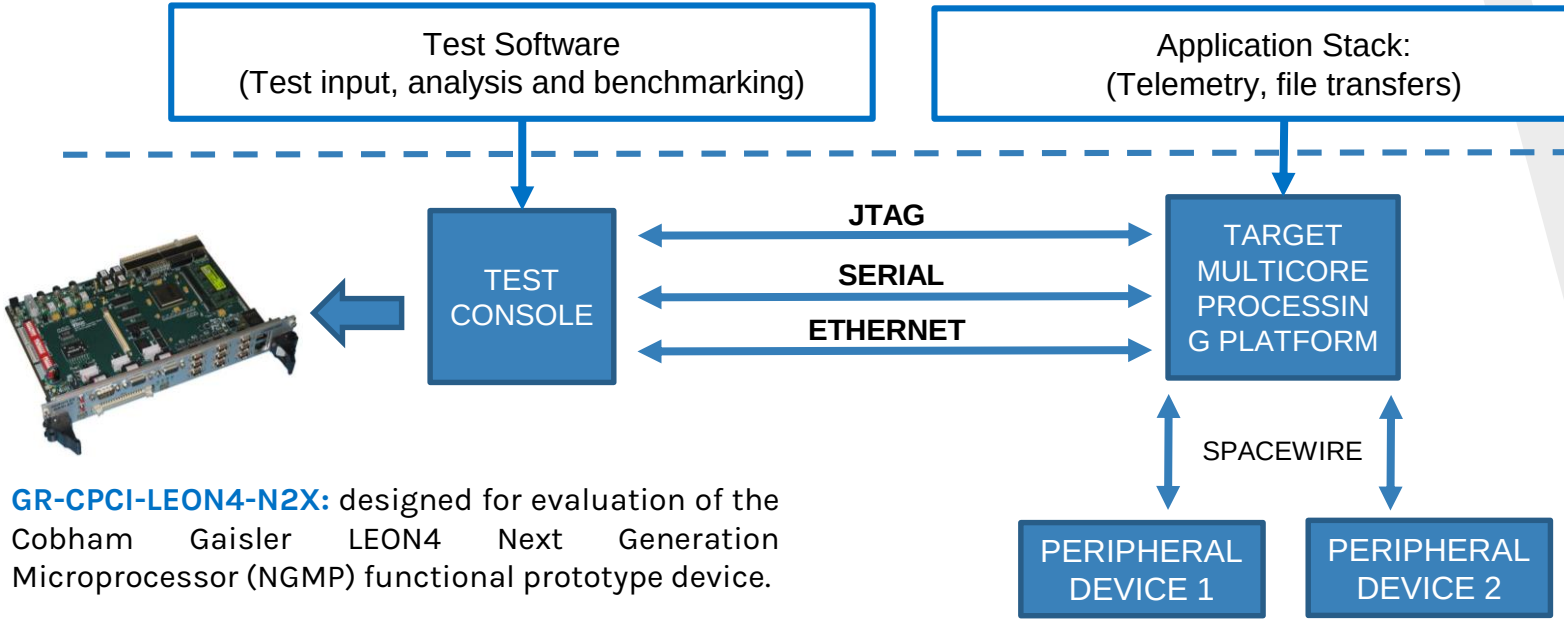
**Safety critical tasks**: All tasks which are needed for a stable and safety flight of the multi-rotor system, e.g. the flight and navigation controllers. An error, like missing a deadline, will cause a crash-landing!

**Mission critical tasks**: All tasks which are not needed for a safe flight, but may also have defined deadlines, e.g. tasks which are belonging to the payload processing, like video processing.

**Uncritical tasks**: All tasks which are not needed either for a safe flight or a correct execution of the mission task, e.g. control of the debug LEDs or transmission of telemetry data.

# Multi-core Implementation

## Univaq EMC² UC – Satellite Demo Platform (Hardware and Software) [8]




**GR-CPCI-LEON4-N2X:** designed for evaluation of the Cobham Gaisler LEON4 Next Generation Microprocessor (NGMP) functional prototype device.

**Processor:** Quad-Core 32-bit LEON4 SPARC V8 processor with MMU, IOMMU

XtratuM

pikeOS

- Migrate a typical aerospace application over a modern multicore platform
- Benchmarking hypervisors
- Compare different virtualization solutions

*F. Federici, V. Muttillo, L. Pomante, G. Valente, D. Andreetti, D. Pascucci,: "Implementing mixed-critical applications on next generation multicore aerospace platforms", CPS Week 2016, EMC² Summit, Vienna, Austria*

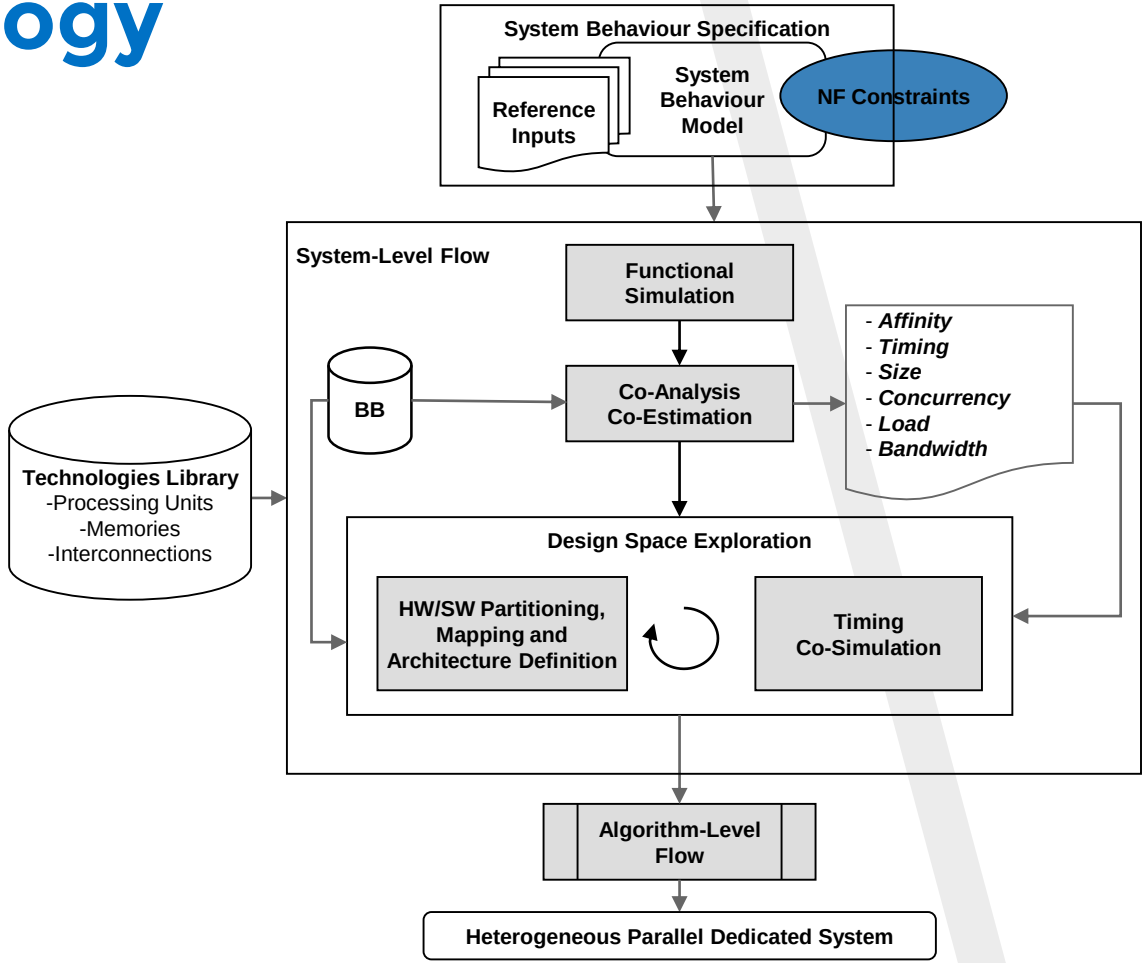
# 4.

# ESL Methodology

“You will never strike oil by drilling through the map! - *Solomon Wolf Golomb*”

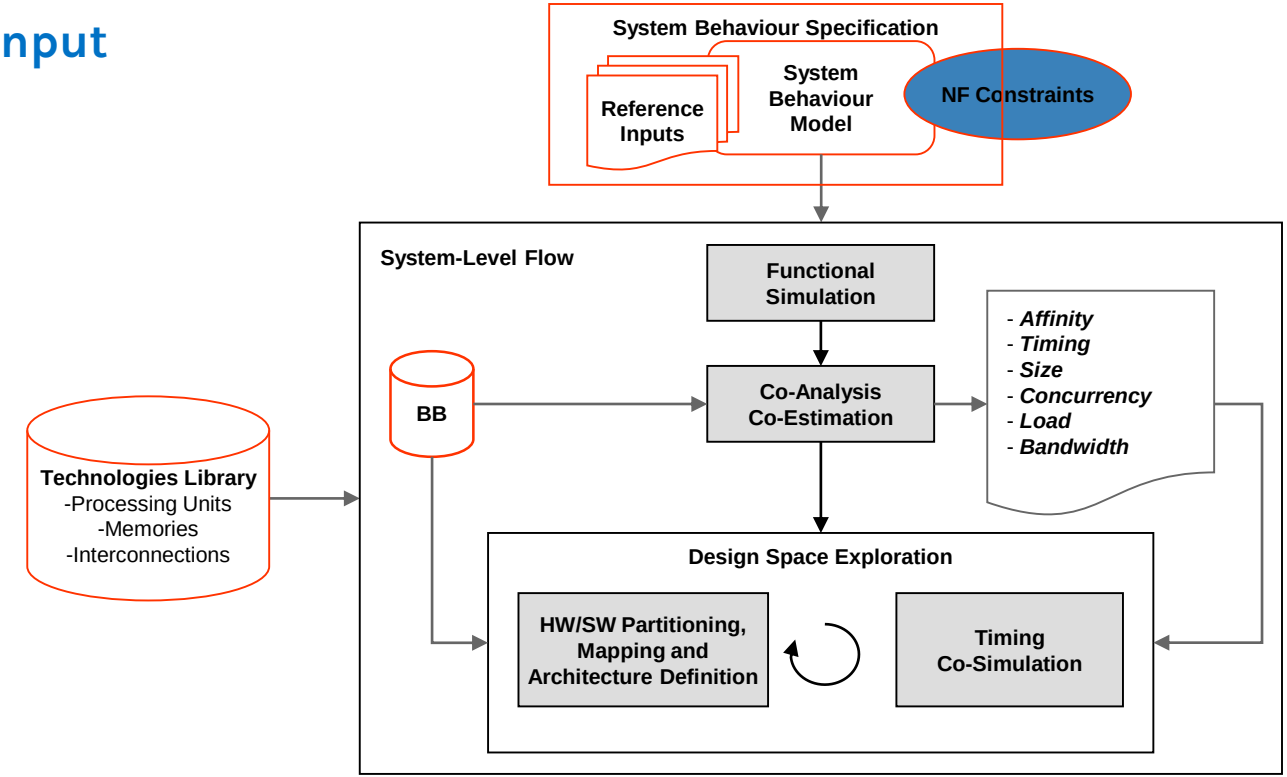
# ESL Methodology

- In the context of real-time embedded systems design, this work starts from a specific methodology (internally called **HEPSYCODE**: HW/SW **CO**-**DE**sign of **HE**terogeneous **P**arallel Dedicated **SY**stems) [www.hepsycode.com], based on an existing System-Level HW/SW Co-Design methodology, and introduces the possibility to specify real-time requirements in the set of non-functional ones (the new framework is so called HEPSYCODE-RT).



# Reference Co-Design Flow

- Input



# Modelling Language

- The system behavior modeling language introduced in HEPSYCODE-RT, named **HML** (**HEPSY Modeling Language**), is based on the well-known **Communicating Sequential Processes** (**CSP**) Model of Computation (MoC)

- By means of HML it is possible to specify the System Behavior Model (SBM)

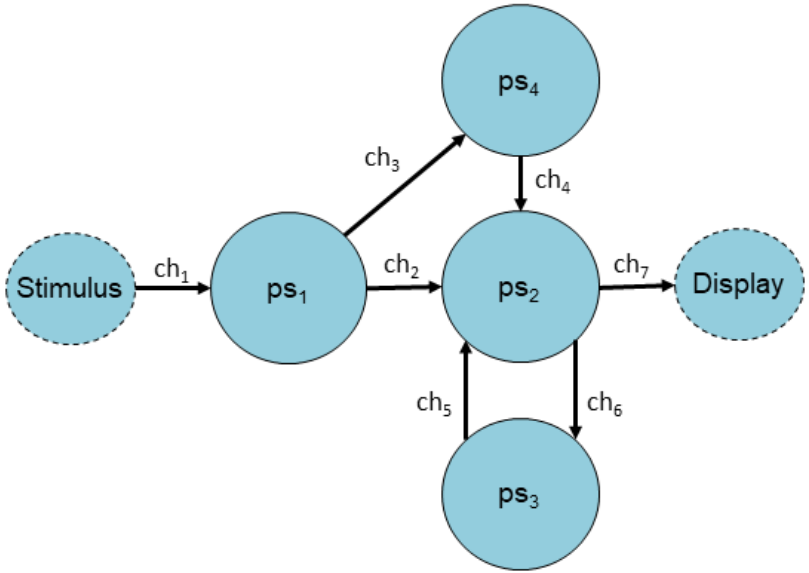
  *SBM = {PS, CH} is a CSP-based executable/simulatable model of the system behaviour based on a Concurrent Processes MoC that explicitly defines also a model of communication) among processes (PS) using unidirectional point-to-point blocking channels (CH) for data exchange (i.e. CSP channels).*

  *PS = {$ps_1$, $ps_2$, .. , $ps_n$} is a set of concurrent processes that communicate each others exclusively by means of channels and use only local variables. Each process is described by means of a sequence of statements (an init section followed by a neverending loop) by using a suitable modeling language. Each process can have a priority p: 1 (lower) to 100 (higher) imposed by the designer*

  *CH = {$ch_1$, $ch_2$, .. , $ch_n$} is a set of channels where each channel is characterized by source and destination processes, and some details (i.e. size, type) about transferred data. Each channel can have also a priority p: 1 (lower) to 100 (higher) imposed by the designer*

# System Behaviour

- An example of a possible SBM in shown in Figure, where the process PS = {$ps_1$, .. , $ps_4$} exchange data using channel CH = {$ch_1$, .. , $ch_7$}



# Constraints

- *Non–Functional Constraints*
  - ✓ Timing Constraints (TC)
    - **Time-To-Completion Constraint (TTC)**
  - ✓ Real-Time Constraints (RTC)
    - **Time-To-Reaction Constraint (TTR)**
  - ✓ Mixed-Criticality Constraints (MCC)
    - **Constraint in the DSE cost function**
  - ✓ Architectural Constraints
    - **Target Form Factor (TFF)**
      - On-chip: ASIC, FPGA, SO(P)C
      - On-Board: SOB (PCB)
    - **Target Template Architecture (TTA)** (related to type of available Basic Blocks BB)
  - ✓ Scheduling Directives (SD) - Available scheduling policies for SW processors:
    - **Round Robin (RR)**, Round Robin (no overhead), Round Robin (Time Stretching)
    - **Fixed Priority (FP)**
    - **Hypervisor (HVP - WIP)**

# Timing Constrains

- ***Time-to-Completion (TTC)**: time available to complete the SBM execution from the first input trigger to complete output generation. This constrain should be satisfied by each (ii, oi) couple.*

- ***Time-to-Reaction (TTR)**: real-time constraints related to the time available for the execution of leaf CSP processes (i.e. the time available to execute the statements inside the input/output pair that delimits the never-ending loop of a CSP process). This constrain should be satisfied by each input and output*

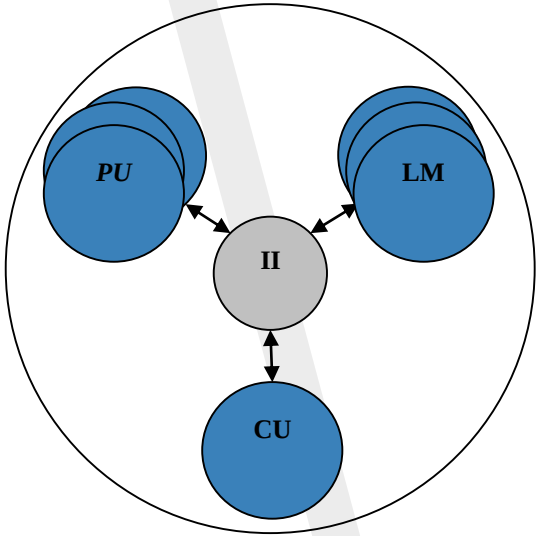
# Target Architecture

- The target HW architectures are composed of different basic HW components. This components are collected into a **Technologies Library** (TL). TL can be considered a generic "database" that provides the characterization of all the available technologies used in industry and academic world.

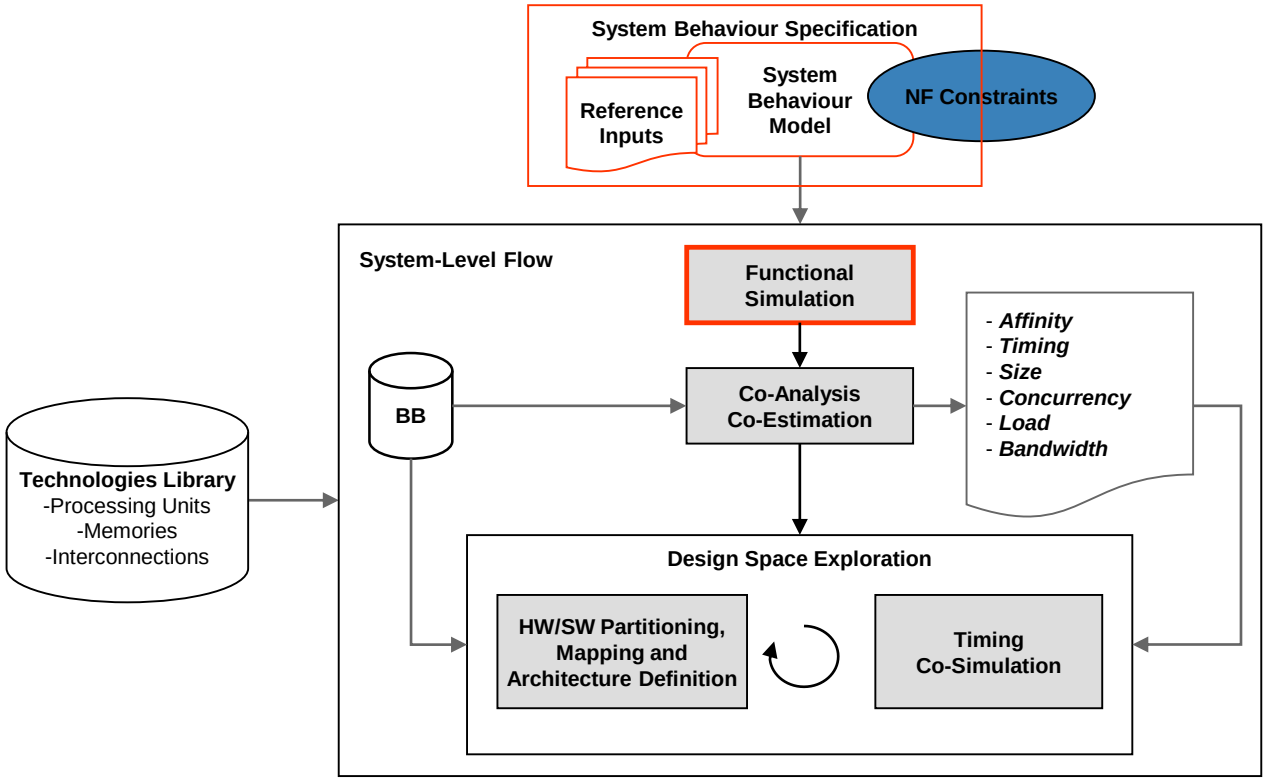
- TL = {PU, MU, CU}, where PU = {$pu_1$, $pu_2$, .. , $pu_p$} is a set of Processing Units, MU = {$mu_1$, $mu_2$, .. , $mu_m$} is a set of Memory Units and CU = {$cu_1$, $cu_2$, .. , $cu_c$} is a set of Communication Units

- Blocks built by the designer starting from the TL are called Basic Blocks (BB)

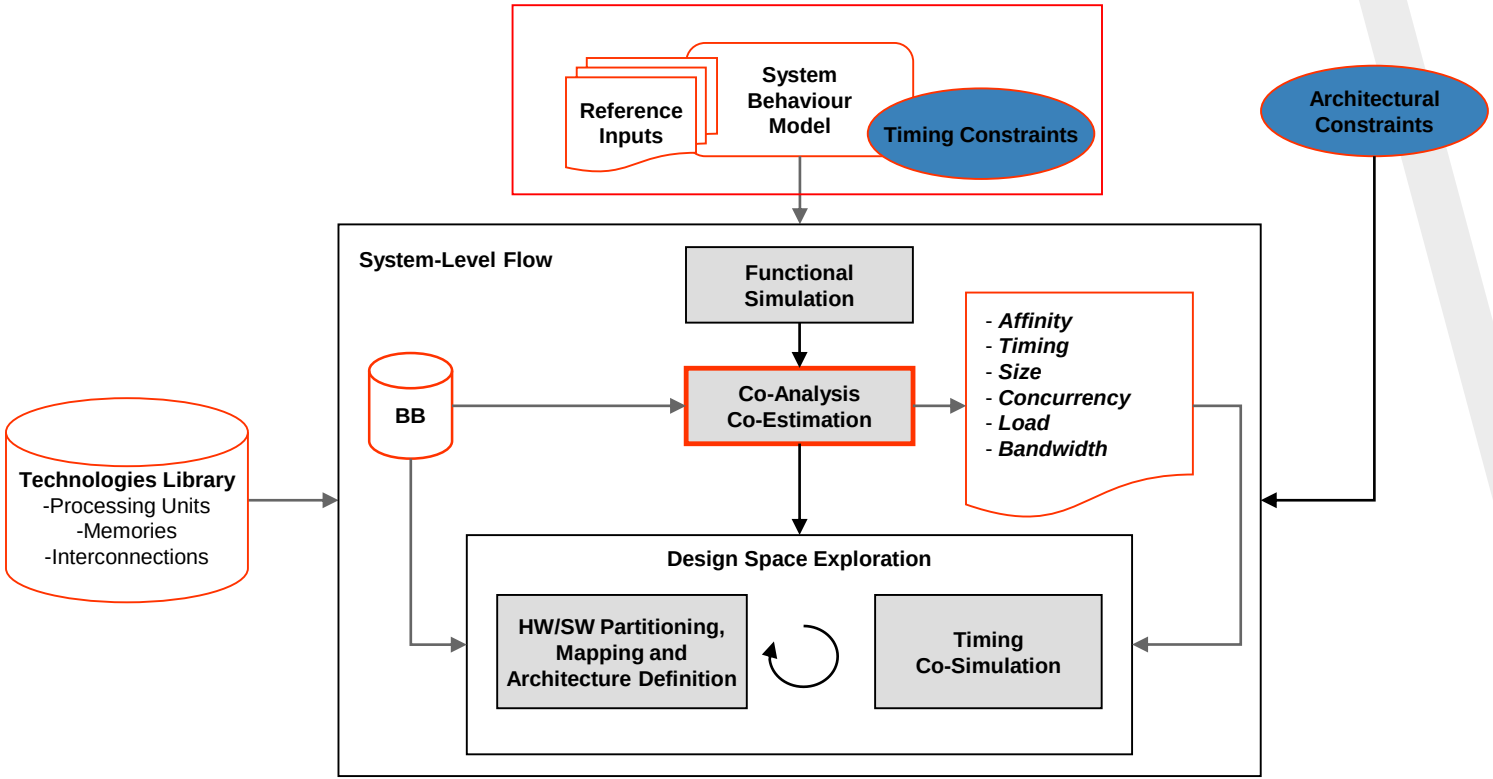
- They are the basic components available during DSE step to automatically define the HW architecture. A generic BB is composed of a set of Processing Units (PU), a set of Memories Units (MU), an Internal Interconnection (IIL) and a Communication Unit (CU)



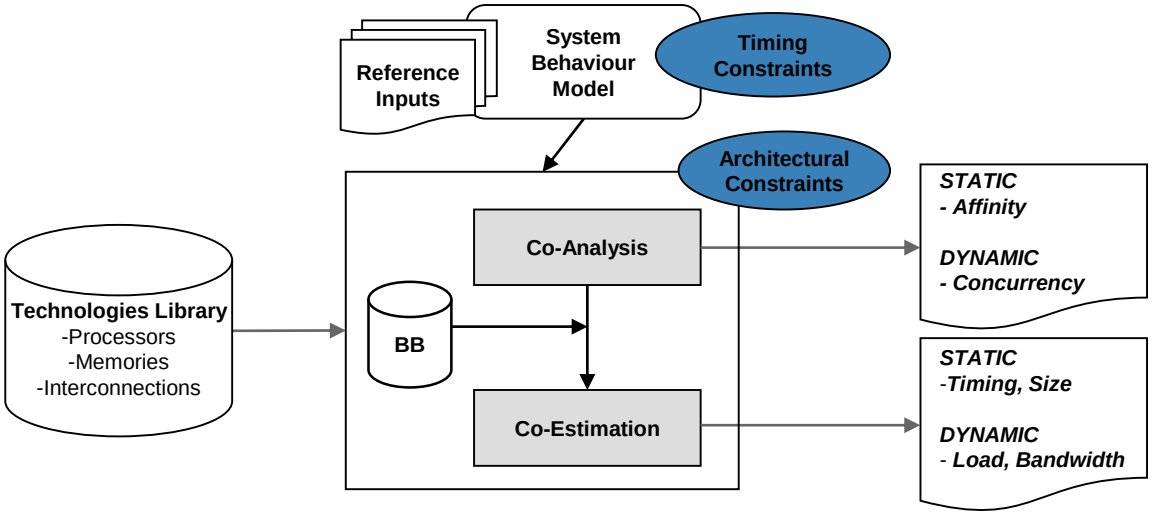
# Functional Simulation



# Co-Analysis & Co-Estimation (1)



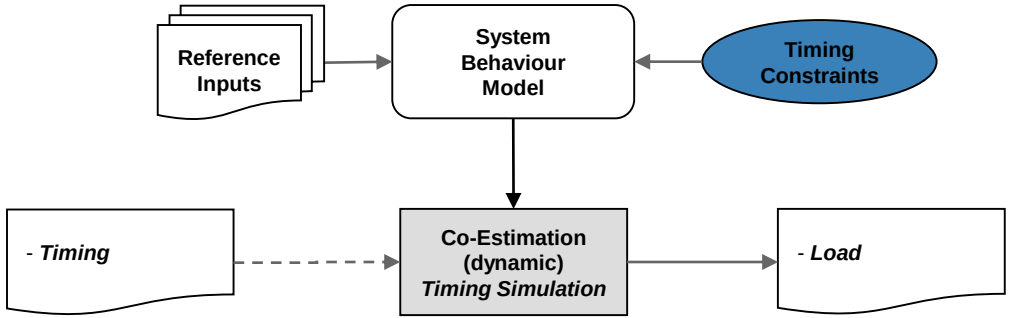
# Co-Analysis & Co-Estimation (2)



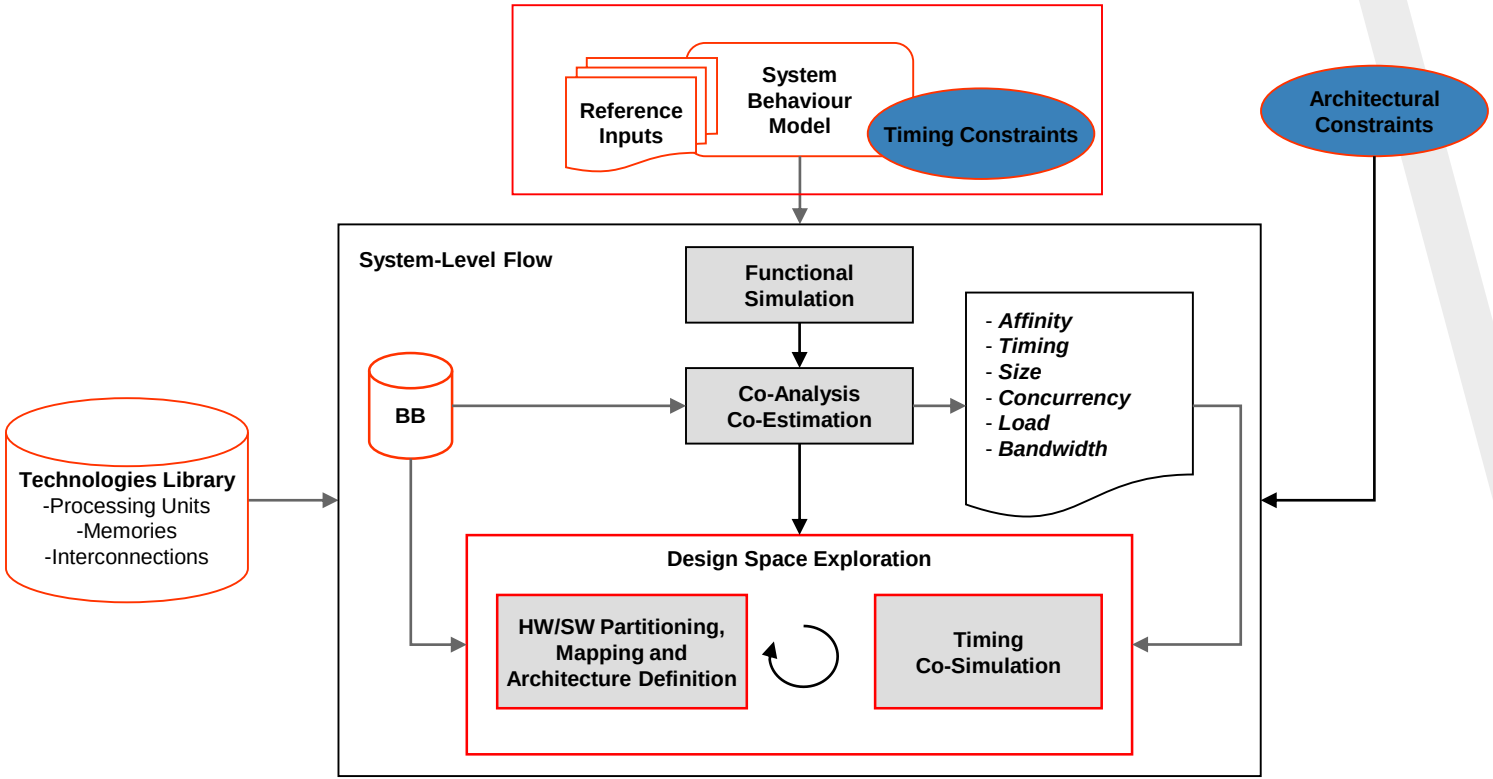
# Co-Estimation

## Load

***L*** *is the Load (i.e. the processor utilization percentage) that each process would impose to each not-SPP processor to satisfy imposed timing constraints*



# Design Space Exploration (1)



# Design Space Exploration (2)

- The goal is to **extend the existing HW/SW co-design methodology** for parallel embedded systems to consider also mixed-criticality applications.

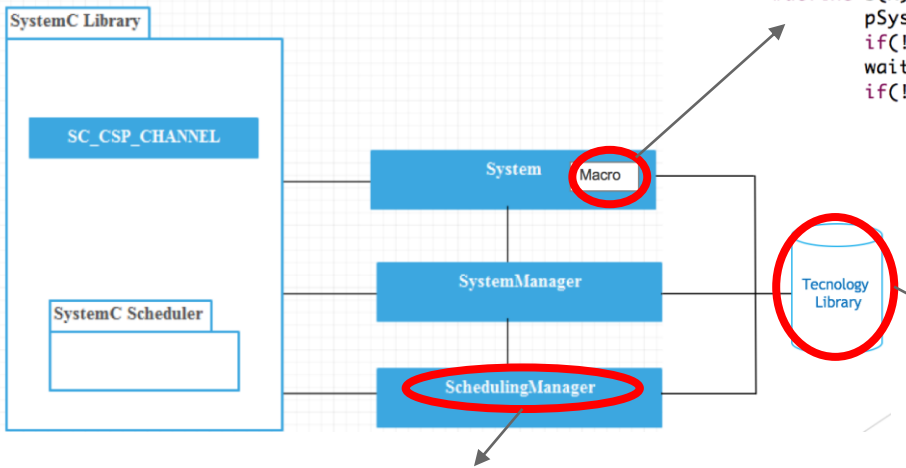
- In order to support incremental DSE for mixed-criticality, UNIVAQ is investigating two iterative activities:
  - First step: starts from system behavior and timing constraints, **provide a suitable architecture/mapping item**
  - Second step: starts from an architecture/mapping item and some mixed-criticality constraints in order to **suggest needed modifications to the HW/SW architecture or to the mapping**

- The final mixed-critical architecture/mapping item is early validated by means of a system-level HW/SW Timing Co-Simulation.

# Design Space Exploration (2)

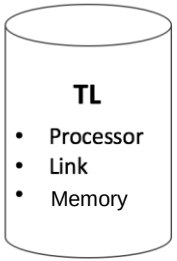
- Main issues:
  - **Extension of** the first-step of **the DSE methodology** for a better management of timing requirements in order to consider also classical RT ones
  - **Analysis of existing HW/SW technologies to** support **mixed-criticality management** (with focus on **hypervisors** technologies) to be exploited in the second-step of the DSE methodology
  - Extension of the **system-level co-simulation approach** to consider also two-levels scheduling policies typically introduced by hypervisors technologies

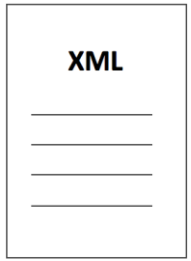
# Co-Simulator




```
#define P(X) pSystemManager->Profiling(X);
#define S(X)                                \
        pSystemManager->Increase(X);    \
        if(!pSystemManager->checkSPP(X)) wait(pSchedulingManager->schedule[X]); \
        wait(pSystemManager->upSimTime(X));    \
        if(!pSystemManager->checkSPP(X)) pSchedulingManager->release[X].notify(SC_ZERO_TIME);
```



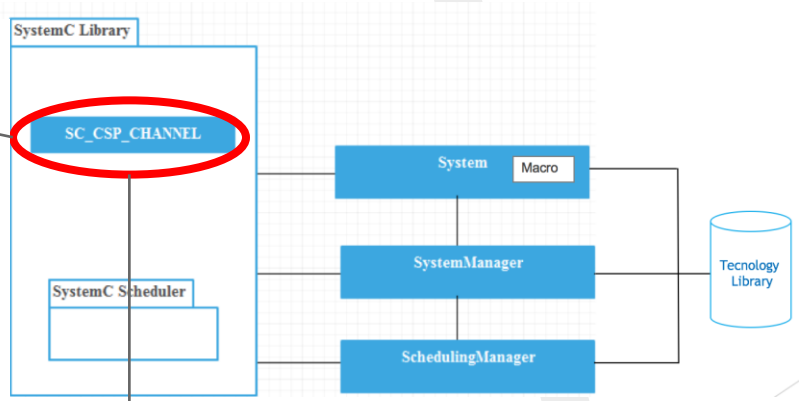

```
wait(0.001*p.getOverheadCS());
sched_oh[pinstance-1]+=(0.001*p.getOverheadCS());

if (ready[ps.id]==true)
{
    if (ps.id!=old_id)
    {
        old_id=ps.id;
        wait(0.999*p.getOverheadCS());
        sched_oh[pinstance-1]+=(0.999*p.getOverheadCS());
    }
    schedule[ps.id].notify(SC_ZERO_TIME);
    wait(release[ps.id]);
}
```

# Co-Simulator (2)

```
sc_csp_channel<T>::write( const T& val_ )
{
    if (w_id>=2 && w_id<pSystemManager->getVPS().size()) pSchedulingManager->ready[w_id]=false;

    if( ready_to_read==true)
    {
        csp_buf=val_;

        ready_to_write=true;
        ready_to_write_event.notify(SC_ZERO_TIME);
        sc_core::wait(ready_to_read_event);

        wait(waiting_time);
        working_time+= waiting_time;

        ready_to_write=false;
        ready_to_write_event.notify(SC_ZERO_TIME);
    }
    else
    {
        ready_to_write=true;
        ready_to_write_event.notify(SC_ZERO_TIME);
        sc_core::wait(ready_to_read_event);

        csp_buf=val_;

        wait(waiting_time);
        working_time+= waiting_time;

        ready_to_write=false;
        ready_to_write_event.notify(SC_ZERO_TIME);
        sc_core::wait(ready_to_read_event);
    }

    // profiling
    num++;

    if (w_id>=2 && w_id<pSystemManager->getVPS().size()) pSchedulingManager->ready[w_id]=true;
}
```

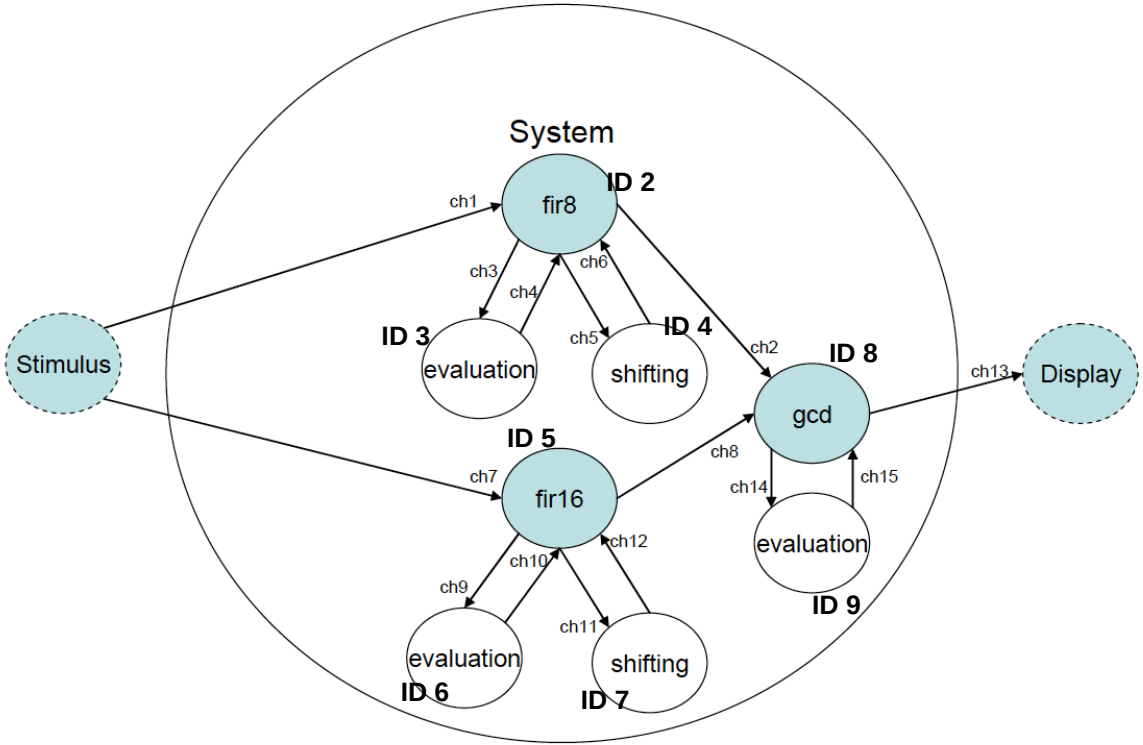
Full handshake methods

waiting_time=IPC*ceil((float)width/pSystemManager>getLink().physical_width) *pSystemManager->getLink() .tcomm +pSystemManager->getLink().tacomm;



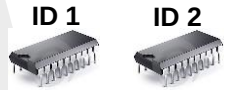

```
sc_csp_channel<T>::read( T& val_ )
{
    if (r_id >=2 && r_id<pSystemManager->getVPS().size()) pSchedulingManager->ready[r_id]=false;

    if(ready_to_write==true)
    {
        ready_to_read=true;
        ready_to_read_event.notify(SC_ZERO_TIME);
        sc_core::wait(ready_to_write_event);

        val_=csp_buf;

        ready_to_read=false;
        ready_to_read_event.notify(SC_ZERO_TIME);
    }
    else
    {
        ready_to_read=true;
        ready_to_read_event.notify(SC_ZERO_TIME);
        sc_core::wait(ready_to_write_event);

        val_=csp_buf;

        ready_to_read=false;
        ready_to_read_event.notify(SC_ZERO_TIME);
        sc_core::wait(ready_to_write_event);
    }

    if (r_id>=2 && r_id<pSystemManager->getVPS().size()) pSchedulingManager->ready[r_id]=true;
}
```

# Reference Application




## Processors

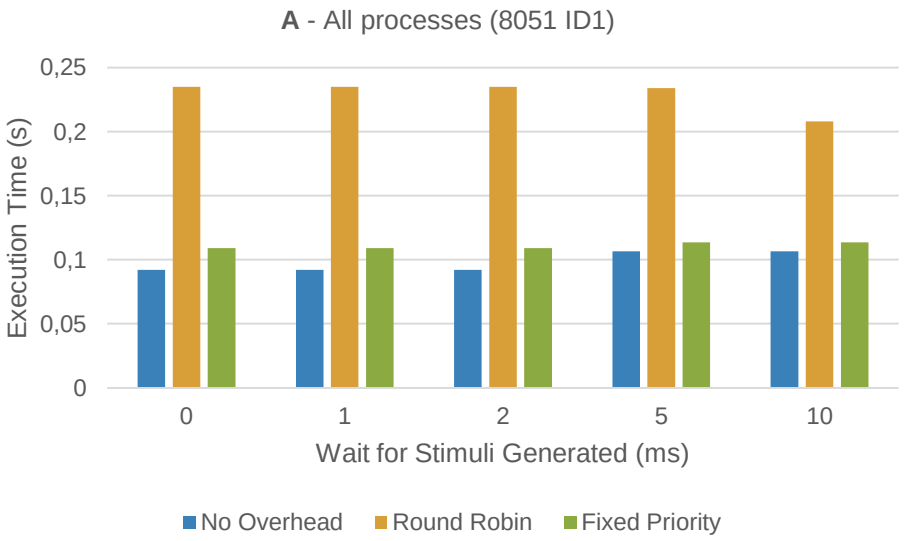
Intel **MPU 8051** with frequency 20 MHz

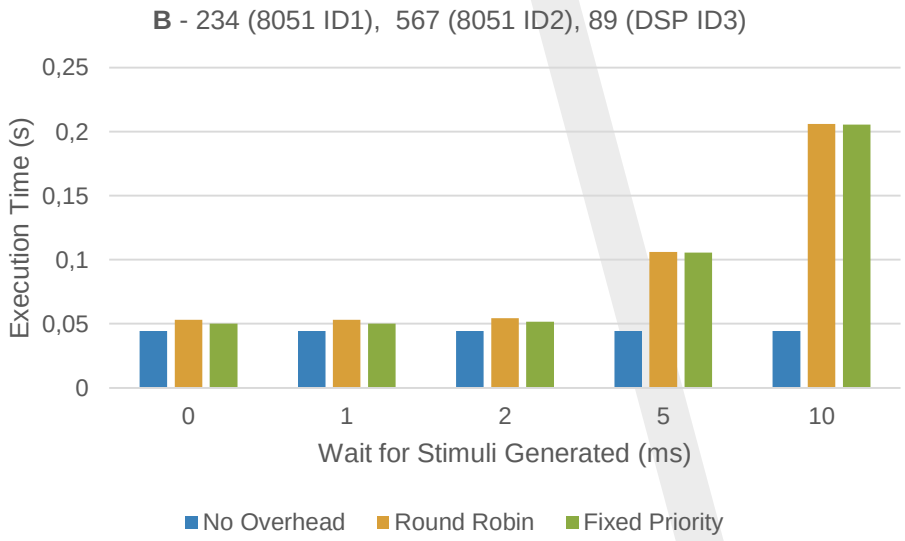



Microchip DSPIC or **PIC24** with frequency 20 MHz

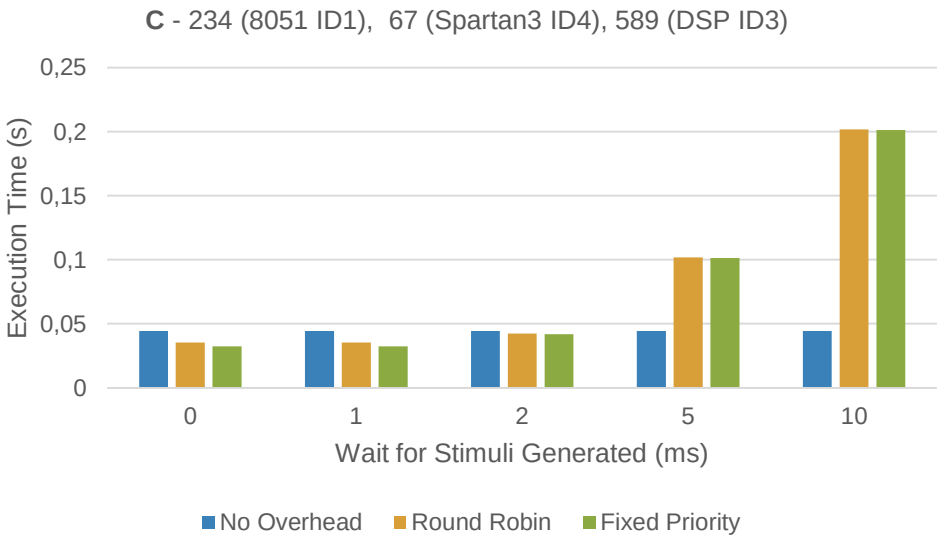

Xilinx **Spartan3AN** with frequency 50 MHz

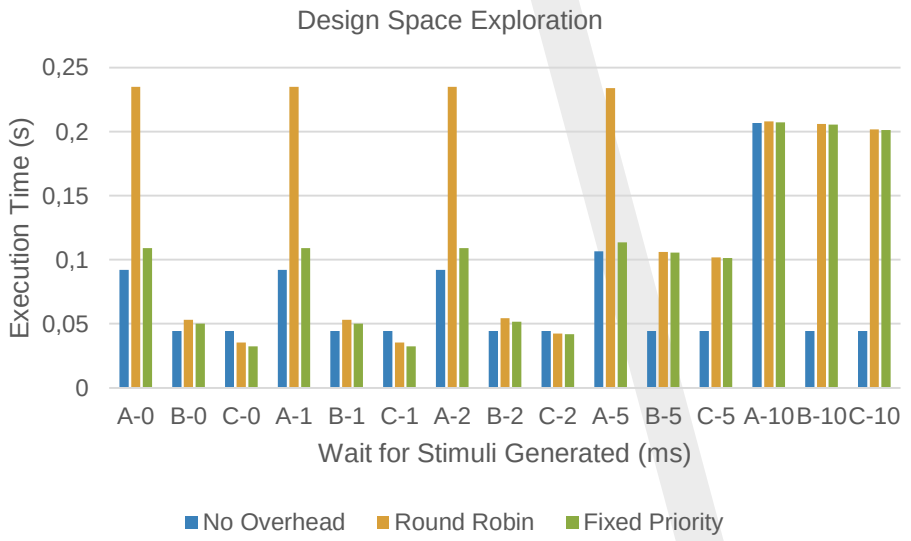
# Preliminary Validation



# Preliminary Validation



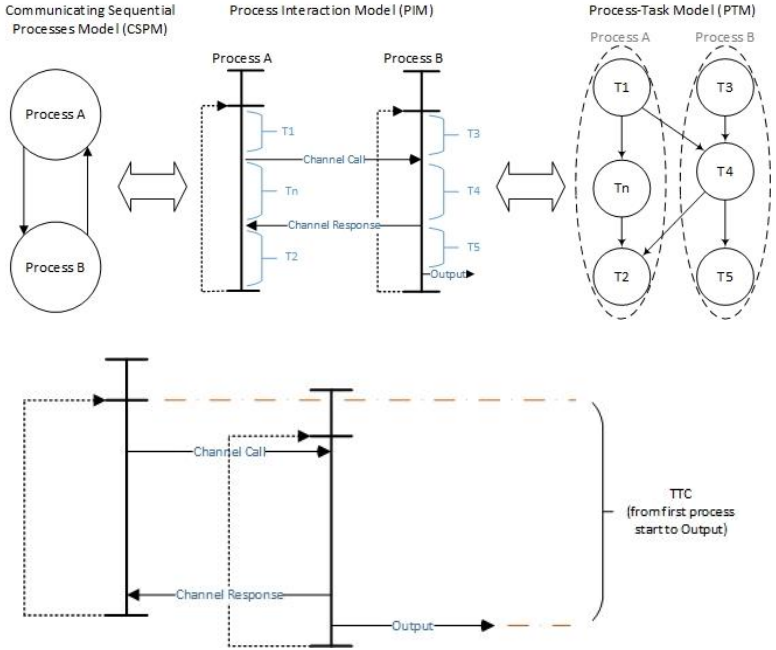
# 5.

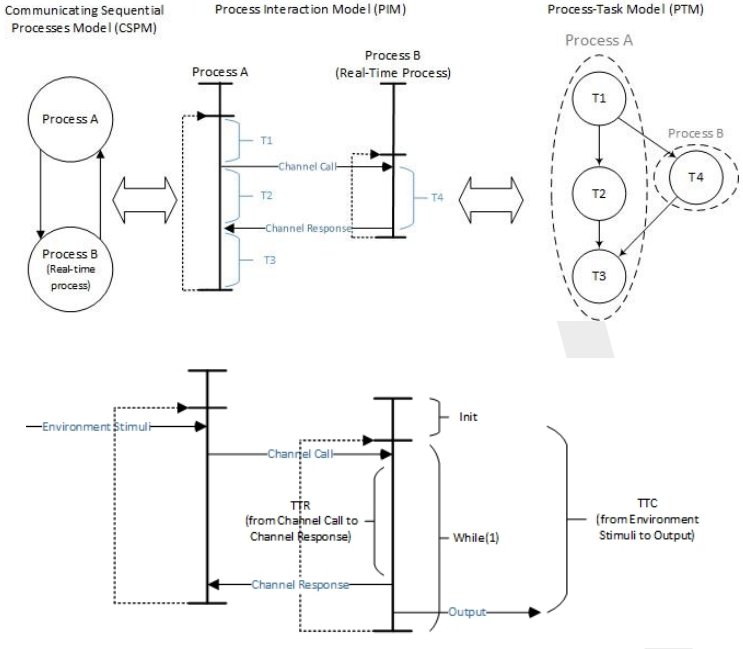
# HepsyCode-RTMC

“The fundamental issue with MCS is how to reconcile the differing needs of separation (for safety) and sharing (for efficient resource usage)”

# SBM with Real-Time Constrains

- With respect to the SBM model, it is now possible to identify two class of CSP processes: classical CSP process and real-time CSP processes



# Co-Estimation

## Load

*The Load (i.e. processor percentage) $L_i$ is the load that each non real-time $ps_i$ process would impose to each s software processor to satisfy input timing constrain. $L_i$ is estimated by allocating all the n processes to a single-instance of each s software processor ($pu_i \subseteq \{[pu_1, .. , pu_s]\}$ with $s \leq n$) and performing simulations while considering the need to satisfy input constrain*

Three parameters has been computed:

- ***$FRT_j$ (Free Running Time)*** *: total application simulation time on processor $pu_j$*
- ***$t_i$*** *simulation time for each process psi in a lap on processor $pu_j$*
- ***N****: number of simulation lap*

Starting from this estimated parameters, the load Li is calculated by the equation:

- $L_i = t_i / (FRT_j/N)$

Designer can impose a timing constraint (TTC) to the system, so that each $FRT_j$ is lower than a certain percentage:

- $TTC = x_j * FRT_j$

New load parameter:

- $L_i = t_i / ([x_j * FRT_j]/N)$

# Co-Estimation

## Load in RT scenario

*The Load Li that each real-time process psi would impose to each s software processor to satisfy input real-time constrain* $TTR_i$ *is set equal to:*

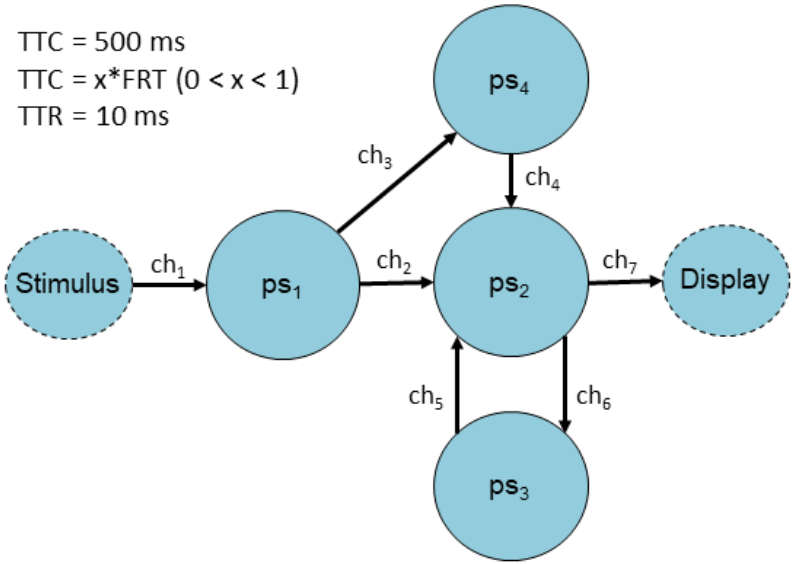
- $L_i = t_i / TTR_i$

$TTR_i$ is the real-time constrain related to the process $ps_j$ and it is the input real-time constrains related to the $pu_j$ processor

In this mode it is possible to consider three different situation:

- **Hard real-time process**: if $t_i < TTR_i$ the constrains is fulfilled and it is possible to consider the value Li as an input to the DSE step
- **Soft real-time process**: if $(TTR_i) < t_i < (TTR_i + \delta(t))$ then constrains could be considered as a soft real-time constrains. It is possible to apply the classical method for the DSE step and load analysis or relaxing the input requirements to match RT constrains.
- **Firm real-time process**: if do not consider real-time constrains it is possible to apply the classical method for the DSE step and load analysis

# Case Study




$L_1 = t_1 / ([x_1 * FTR_1]/N)$, $L_2 = t_2 / ([x_2 * FTR_1]/N)$,
$L_4 = t_4 / ([x_4 * FTR_1]/N)$
$L_3 = t_3 / TTR_3$ *(real-time process load)*

# Results

| Allocation | FINAL SIMULATED TIME (ms) | PS1 (ms) | PS2 (ms) | PS3 (ms) | PS4 (ms) | LAPS | TTC (ms) | TTR (ms) | Check Constraint |
|---|---|---|---|---|---|---|---|---|---|
| All MPU_1 | 425,6755 | 20,67015 | 30,94065 | 4,22915 | 20,62915 | 10 | 600 | 10 | YES |
| All MPU_2 | 545,344375 | 25,8376875 | 38,6758125 | 5,2864375 | 25,7864375 | 10 | 600 | 10 | YES |
| All MPU_2 | 545,344375 | 25,8376875 | 38,6758125 | 5,2864375 | 25,7864375 | 10 | 600 | 5 | NO |
| All FPGA | 7,1078 | 0,352905 | 0,528255 | 0,072205 | 0,352205 | 10 | 600 | 5 | YES |
| 1 and 4 on MPU_2, 2 and 3 on MPU_1 | 473,8755 | 25,8376875 | 30,94065 | 4,22915 | 25,7864375 | 10 | 500 | 5 | YES |
| 1 and 4 MPU_2, 2 and 3 on MPU_1 | 473,8755 | 25,8376875 | 30,94065 | 4,22915 | 25,7864375 | 10 | 500 | 4 | NO |
| 4 on MPU_2, 1, 2 and 3 on MPU_1 | 446,338875 | 20,67015 | 30,94065 | 4,22915 | 25,7864375 | 10 | 500 | 4 | NO |
| 1 on MPU_2,  2, 3 and 4 on MPU_1 | 447,0755 | 25,8376875 | 30,94065 | 4,22915 | 20,62915 | 10 | 500 | 4 | NO |
| 1, 2 and 4 on MPU_2, 3 on MPU_1 | 525,304 | 25,8376875 | 38,6758125 | 4,22915 | 25,7864375 | 10 | 500 | 4 | NO |
| 1 and 4 on MPU_2, 2 on MPU_1, 3 on FPGA | 449,58505 | 25,8376875 | 30,94065 | 0,072205 | 25,7864375 | 10 | 500 | 4 | YES |

# 6.

# Conclusion and Future Works

“The fundamental issue with MCS is how to reconcile the differing needs of separation (for safety) and sharing (for efficient resource usage)”

# Conclusions and future work

- This talk presents the MC domain, respect to RT model, criticality and safety requirements and high system-level design methodologies
- An extended ESL Electronic Design Automation (EDA) methodology (and related tools) that will help designers to develop Mixed-Criticality Embedded Systems has been discussed
- After defined a CSP to RT model transformation, the next step is to further enhance the DSE step to suggest to the designer how to manage different criticality levels of applications, components, and tasks, by means of relevant available technologies (e.g. hypervisors, physical partitioning, etc.).
- Introduce multiple scheduling levels to simulate Hypervisor behavior

# Reference


Main References

L. Pomante, D. Sciuto, F. Salice, W. Fornaciari, C. Brandolese, “Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC”, IEEE Transactions on Computers, Vol. 55, Iss. 5, May 2006.

L. Pomante, “System-Level Design Space Exploration for Dedicated Heterogeneous Multi-Processor Systems”, IEEE International Conference on Application-specific Systems, Architectures and Processors, September 2011.

L. Pomante, “HW/SW Co-Design of Dedicated Heterogeneous Parallel Systems: an Extended Design Space Exploration Approach”. IET Computers & Digital Techniques, Institution of Engineering and Technology, 2013, Vol. 7, Iss. 6, pp. 246–254.

Other Journals

L. Pomante. “System-Level Design Space Exploration for Heterogeneous Parallel Dedicated Systems”, DLINE Journal of Electronic Systems, 2013, Vol. 3 , Iss. 2

L. Pomante, P. Serri, "SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems", International Journal of Information Systems, 2014, Vol.1

Books

L. Pomante. “Electronic System-Level HW/SW Co-Design of Heterogeneous Multi-Processor Embedded Systems”, The River Publishers Series in Circuits and Systems, June 2016.

# Reference


Other Conference Papers

L. Pomante, S. Marchesani, P. Serri, “Design Space Exploration for Heterogeneous Multi Multi-Core Processor Dedicated Systems”. 3th Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy'13), Philadelphia, April 2013.

L. Pomante, S. Marchesani, P. Serri, “System-Level Design Space Exploration for Heterogeneous Parallel Dedicated Systems”. ICMAES'2013 - The International Conference on Machines Applications and Embedded Systems, Computer and Information Technology (WCCIT), 2013 World Congress on , Sousse (Tunisia), June 2013.

F. Federici, V. Muttillo, L. Pomante, P. Serri, G. Valente, "A Model-Based ESL HW/SW Co-Design Framework for Mixed-Criticality Systems", EMC$^2$ Summit at CPS Week, Vienna, Austria, 11 April 2016

D. Di Pompeo, E. Incerto, V. Muttillo, L. Pomante, G. Valente, "An Efficient Performance-Driven Approach for HW/SW Co-Design", International Conference on Performance Engineering (ICPE '17), 2017, ACM, New York, NY, USA, 323-326.

Work in Progress Sessions Papers

D. Ciambrone, V. Muttillo, G. Valente, L. Pomante, "HW/SW Co-Simulator for Embedded Heterogeneous Parallel Systems", Euromicro Conference on Digital Systems Design (DSD) - Work in Progress Session, 2017

V. Stoico, V. Muttillo, G. Valente, L. Pomante, F. D'Antonio, "CC4CS: A Unifying Statement-Level Performance Metric for HW/SW Technologies", Euromicro Conference on Digital Systems Design (DSD) - Work in Progress Session, 2017

THANKS!

# Any questions?

# 7.

# Backup Papers

# Hardware Implementation

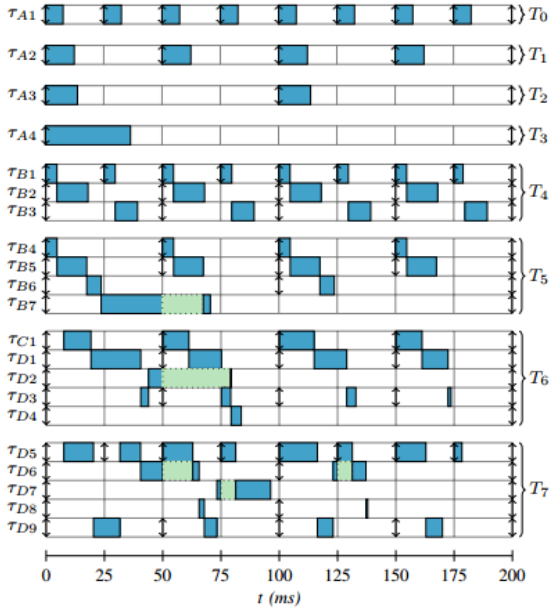
## FlexPRET: Processor Platform for Mixed-Criticality Systems [11]

**FlexPRET** is a 32-bit, 5-stage, fine-grained multithreaded processor with software-controlled, flexible thread scheduling. It uses a classical RISC 5-stage pipeline: instruction fetch (F), decode (D), execute (E), memory access (M), and writeback (W). Predict not-taken branching and software-controlled local memories are used for fine-grained predictability. It also implements the RISC-V ISA [20], an ISA designed to support computer architecture research, that we extended to include timing instructions.

FlexPRET is implemented in Chisel [26], a hardware construction language that generates both Verilog code and a cycle-accurate C++-based simulator.

| Task | Thread ID | Thread Mode | $T_i, D_i$ (ms) | $E_{i,1}$ ($*10^5$) | $E_{i,1/2}$ ($*10^5$) | $E_{i,1/3+}$ ($*10^5$) |
|---|---|---|---|---|---|---|
| $\tau_{A1}$ | 0 | HA | 25 | 1.10 | 1.00 | 0.95 |
| $\tau_{A2}$ | 1 | HA | 50 | 1.80 | 1.64 | 1.55 |
| $\tau_{A3}$ | 2 | HA | 100 | 2.00 | 1.82 | 1.72 |
| $\tau_{A4}$ | 3 | HA | 200 | 5.30 | 4.83 | 4.56 |
| $\tau_{B1}$ | 4 | HA | 25 | 1.40 | 1.27 | 1.20 |
| $\tau_{B2}$ | 4 | HA | 50 | 3.90 | 3.54 | 3.34 |
| $\tau_{B3}$ | 4 | HA | 50 | 2.80 | 2.54 | 2.40 |
| $\tau_{B4}$ | 5 | HA | 50 | 1.40 | 1.28 | 1.21 |
| $\tau_{B5}$ | 5 | HA | 50 | 3.70 | 3.37 | 3.19 |
| $\tau_{B6}$ | 5 | HA | 100 | 1.80 | 1.64 | 1.55 |
| $\tau_{B7}$ | 5 | HA | 200 | 8.50 | 7.75 | 7.32 |
| $\tau_{C1}$ | 6 | SA | 50 | 1.90 | 1.77 | 1.63 |
| $\tau_{D1}$ | 6 | SA | 50 | 5.40 | 5.03 | 4.65 |
| $\tau_{D2}$ | 6 | SA | 200 | 2.40 | 2.33 | 2.28 |
| $\tau_{D3}$ | 6 | SA | 50 | 1.30 | 1.26 | 1.23 |
| $\tau_{D4}$ | 6 | SA | 200 | 1.50 | 1.45 | 1.42 |
| $\tau_{D5}$ | 7 | SA | 25 | 2.30 | 2.14 | 1.98 |
| $\tau_{D6}$ | 7 | SA | 100 | 4.80 | 4.65 | 4.30 |
| $\tau_{D7}$ | 7 | SA | 200 | 13.00 | 12.70 | 12.44 |
| $\tau_{D8}$ | 7 | SA | 100 | 0.60 | 0.57 | 0.56 |
| $\tau_{D9}$ | 7 | SA | 50 | 2.40 | 2.33 | 2.28 |

A mixed-criticality avionics case study

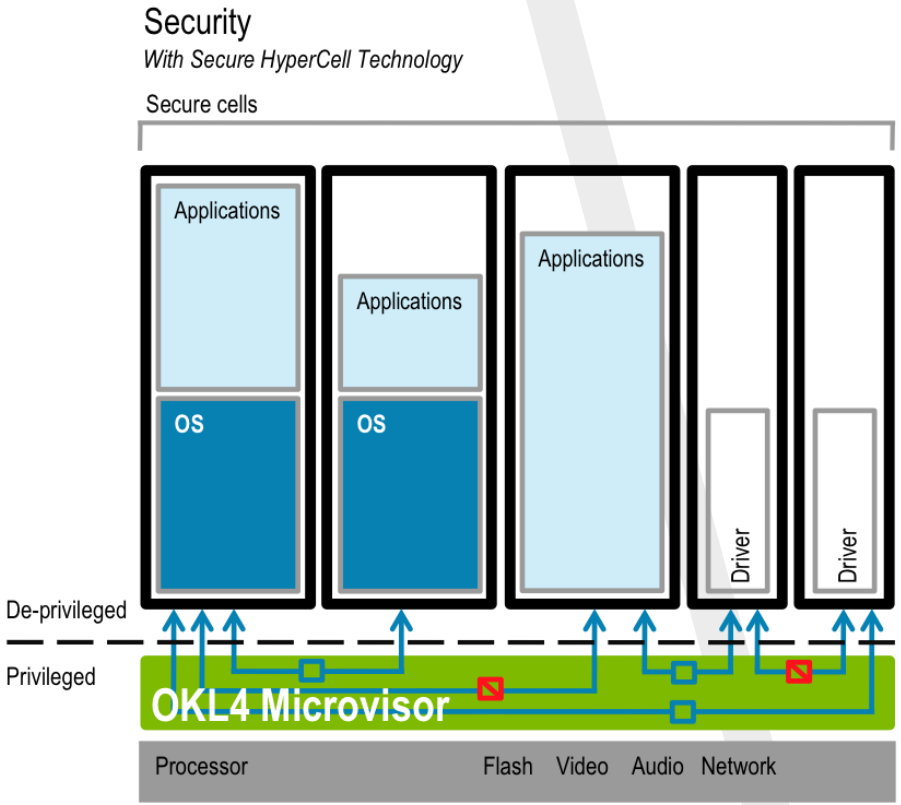



FlexPRET-8T (8 physical number of threads available) executing a mixed-criticality avionics case study

# Single-core Implementation

## OKL4 Microvisor [14]

- The **OKL4** Microvisor is an advanced secure type-1 hypervisor developed by General Dynamics C4 Systems and supports all ARM processors with MMU hardware
- Supporting virtualization with the lowest possible overhead, the microvisor's abstractions are designed with:
- the microvisor's execution abstraction is that of a virtual machine with one or more virtual CPUs (vCPUs), on which the guest OS can schedule activities;
- the memory abstraction is that of a virtual MMU (vMMU), which the guest OS uses to map virtual to (guest) physical memory;
- the I/O abstraction consists of memory-mapped virtual device registers and virtual interrupts (vIRQs);
- communication is abstracted as vIRQs (for synchronisation) and channels. The latter are bi-directional FIFOs with a fixed (configurable per channel) buffer allocated in user space (run also TCP/IP on a channel).



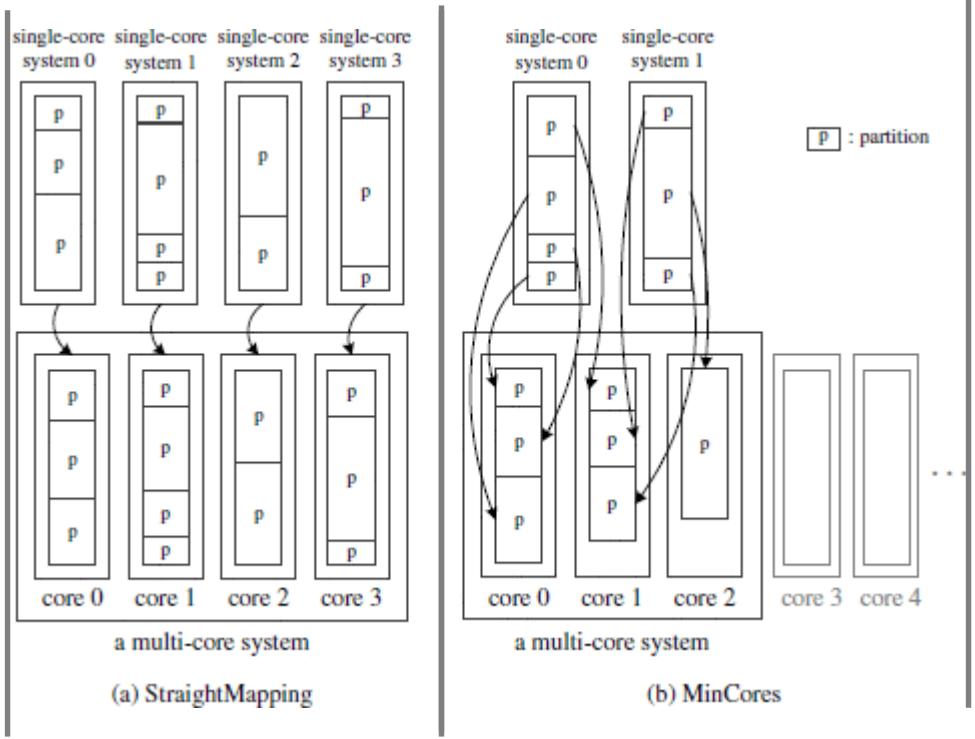
# Multi-core Implementation

## Multi-IMA Partitioning [6]

Given a set of:

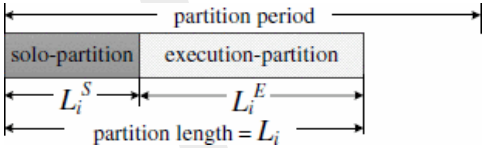
- Single-core IMA systems
- Partitions
- Multi-core system

All partitions from a single core must be scheduled on the same core of the multi-core system (they can be rescheduled within the same core)




Multi-IMA partition scheduling optimization where a partition consists of two logical regions:

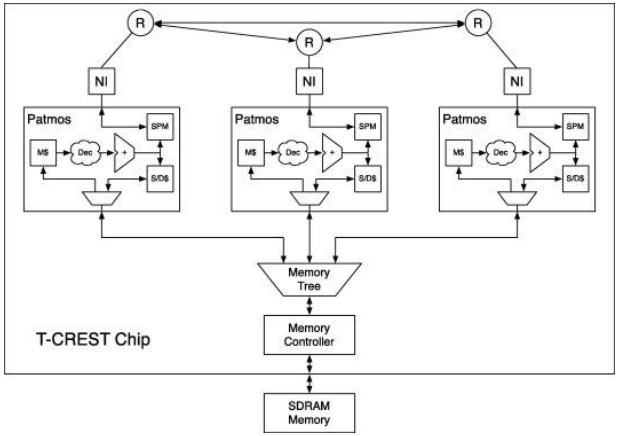
- solo- partition (in avionics systems performing I/O transactions )
- execution-partition

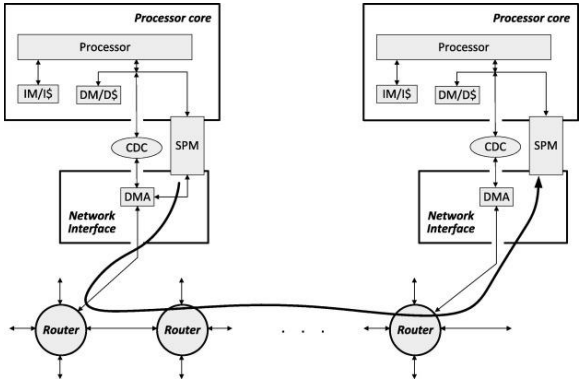


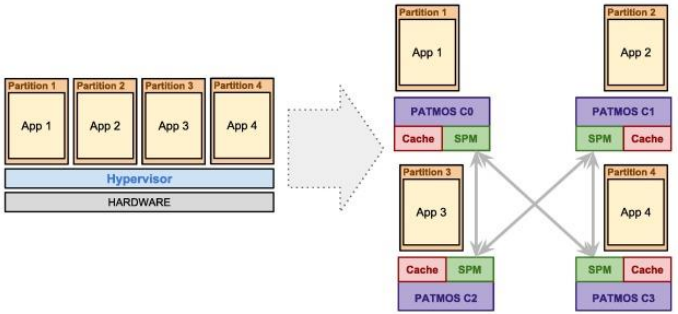

Partition can be scheduled on a core if it doesn't interfere with the solo-partitions of other partitions and doesn't overlap with other partitions assigned to the same core. Each partition is strictly periodic and non-preemptive. This supports **temporal** and **spatial** isolation among partitions.

# Many-core Implementation

## T-CREST Multi-core Architecture [15]



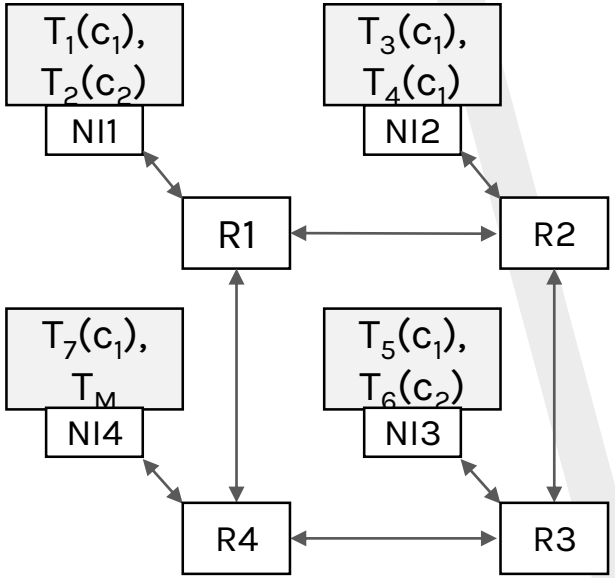




- The ***T-CREST*** platform consisting of Patmos processor nodes that are connected via an on-chip network for message passing communication and a memory tree to a memory controller for shared memory access
- Data transfer in the **T-CREST** core-to-core message passing NoC.
- Mapping of **ARINC 653** partitions to cores onto the **T-CREST** platform.

# Many-core Implementation

## University of L'Aquila CRAFTERS Case Study

- **Hardware mechanisms to support isolation in a Network-on-Chip**
  - Isolation of different application classes on NoC architectures
  - Hardware mechanisms supporting isolation to be introduced into existing network interfaces
  - Support for the execution of multiple applications with different criticality levels
  - Strategy: message exchange supervision




*L. Pomante, C. Tieri, F. Federici, M. Colizza, M. Faccio, R. Cardinali, B. Iorio. "HW Mechanisms to Support Isolation in Mixed-Criticality NoC". Euromicro Conference on Digital System Design - WIP Session, Verona, August 2014.*

# Reference


[1] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in Proc. Int’l Real-Time Systems Symp. (RTSS), 2007

[2] A. Gerstinger, H. Kantz, C. Scherrer: ”TAS Control Platform: A Platform for Safety-Critical Railway Applications”, ERCIM NEWS 75, Oct. 2008

[3] D. Muench, O. Isfort, K. Mueller, M. Paulitsch, A. Herkersdorf: "Hardware-Based I/O Virtualization for Mixed Criticality Real-Time Systems Using PCIe SR-IOV," 2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, NSW, 2013, pp. 706-713

[4] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, J. Nowotsch: "Mixed-Criticality Embedded Systems -- A Balance Ensuring Partitioning and Performance" Digital System Design (DSD), 2015 Euromicro Conference on, Funchal, 2015, pp. 453-461

[5] M. G. Hill, T. W. Lake: "Non-interference analysis for mixed criticality code in avionics systems," Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on, Grenoble, France, 2000, pp. 257-260.

[6] J. E. Kim, M. K. Yoon, S. Im, R. Bradford, L. Sha: “Optimized Scheduling of Multi-IMA Partitions with Exclusive Region for Synchronized Real-Time Multi-Core System” in Proceedings of the 16th ACM/IEEE Design, Automation, and Test in Europe (DATE 2013), Mar. 2013.

[7] J. E. Kim, M. K. Yoon, R. Bradford, L. Sha, "Integrated Modular Avionics (IMA) Partition Scheduling with Conflict-Free I/O for Multicore Avionics Systems," to appear in Proceedings of the 38th IEEE Computer Software and Application Conference (COMPSAC 2014), Jul. 2014.

[8] F. Federici, V. Muttillo, L. Pomante, G. Valente, D. Andreetti, D. Pascucci: “Implementing mixed-critical applications on next generation multicore aerospace platforms”, CPS Week 2016, EMC$^2$ Summit, Vienna, Austria

[9] B. Huber, C. El Salloum, and R. Obermaisser. A resource management framework for mixed-criticality embedded systems. In 34th IEEE IECON, pages 2425–2431, 2008

# Reference


[10] R. Pellizzoni, P. Meredith, M. Y. Nam, M. Sun, M. Caccamo, L. Sha,: “Handling Mixed-criticality in SoC-based Real-time Embedded Systems”, Proceedings of the Seventh ACM International Conference on Embedded Software, 2009

[11] M. Zimmer, D. Broman, C. Shaver and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems" 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), Berlin, 2014, pp. 101-110.

[12] M. Mollison, J. Erickson, J. Anderson, S. Baruah, J. Scoredos: “Mixed-criticality real-time scheduling for multicore systems,” in Proc. of the 10th IEEE International Conference on Computer and Information Technology (CIT), 2010, pp. 1864–1871.

[13] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. B. Nejad, A. Nelson, S. Sinha: Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow. SIGBED Rev. 10, 3 (October 2013), 23-34.

[14] G. Heiser, B. Leslie: “The OKL4 microvisor: convergence point of microkernels and hypervisors”, In: Proceedings of the first ACM asia-pacific workshop on Workshop on systems (APSys '10). ACM, New York, NY, USA, 19-24

[15] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, A. Tocchi: “T-CREST: Time-predictable multi-core architecture for embedded systems, Journal of Systems Architecture”, Volume 61, Issue 9, October 2015, Pages 449-471

[16] W, Weber, A. Hoess, J. van Deventer, F. Oppenheimer, R. Ernst, A. Kostrzewa, P. Dorè, T. Goubier, H. Isakovic, N. Druml, and others: “The EMC2 Project on Embedded Microcontrollers Technical Progress after Two Years”. Digital System Design (DSD), Euromicro Conference on. Pp. 524-531