1st Italian Workshop on
Embedded Systems (IWES 2016)

An Integrated ESL Methodology for Developing Embedded Parallel Systems in Mixed-Criticality Scenarios

Author:
Vittoriano Muttillo
vittoriano.muttillo@graduate.univaqit

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

Summary

1. Introduction
2. Safety Assurance Standards
3. Mixed Criticality Systems (MCS)
   - MCS state-of-the-art Model
   - MCS Design
   - Industrial and Academic MCS Implementations
5. ESL Methodology
   - Concurrency and process calculi
   - Communicating sequential processes (CSP), model checking and Timed CSP
   - HW/SW Co-Design Methodology for MCS
   - UML/MARTE profile for MCS
6. Conclusion and Future Works

1.

Introduction

“Brief Introduction to Embedded and Mixed Criticality Systems”

Embedded Systems

- In contrast to generic reprogrammable general purpose computer, an embedded system is composed of a set of tasks already known during the development. This make possible to identify a hardware/software combination specifically designed for such an application.

General Purpose Computer

Embedded Systems

- In contrast to generic reprogrammable general purpose computer, an embedded system is composed of a set of tasks already known during the development. This make possible to identify a hardware/software combination specifically designed for such an application.

- Hardware can be reduced to a minimum in order to contain space thus limiting consumption, processing times (higher efficiency) and manufacture cost, considering F/NF requirements.

- Many embedded systems are real-time systems, in which "the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced"

Embedded/**real-time**

Mixed-Criticality Systems

- A mixed criticality system is "**an integrated suite of HW, OS, middleware services and application software that supports the concurrent execution of safety-critical, mission-critical, and non-critical software within a single, secure computing platform**", i.e. a system containing computer hardware and software that executes concurrently several applications of different criticality (such as safety-critical and non-safety critical).

- Different criticality applications are engineered to different levels of assurance, with high criticality applications being the most costly to design and verify.

- Mixed-Criticality systems are typically embedded in more complex systems such as an aircraft whose safety must be ensured.

Embedded/real-time/ safety-critical/**mixed-critical**

2.

Safety Assurance Standards

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

Safety Related Standards

- Industry has shown a growing interest in integrating and running independently-developed applications of different "criticalities" in the same (often multicore) platform. Such integrated systems are commonly referred to as **mixed-criticality systems (MCS)**.

- Most of the MCS-related research cite the **safety-related standards** associated to each application domain (e.g. aeronautics, space, railway, automotive) to justify their methods and results. However, those standards are not freely available and do not always clearly and explicitly specify the requirements for mixed-criticality

- New MC task model is in essence the result of combining the **standard hard real-time requirements** (studied by the real-time research community since the 70's) with the **notion of "criticality" of execution**.

System Design and Development Assurance Process

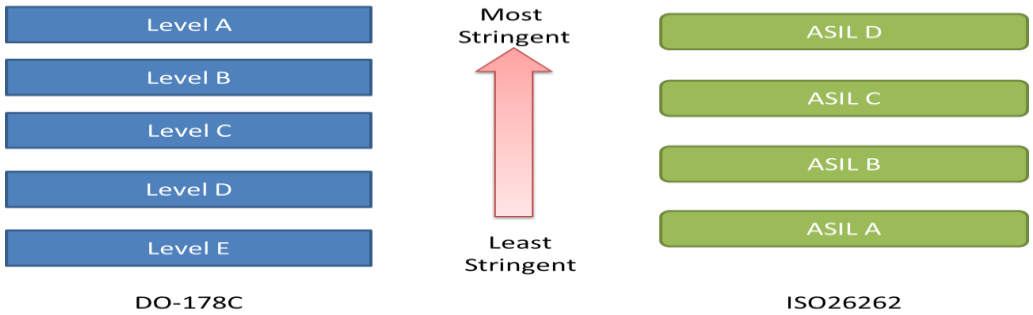
- During a typical **development life cycle of a safety-critical system**, the behavior and characteristics that are expected from the system are expressed in the form of a **list of requirements**
  - based on the system operational requirements (what the system is expected to do) and also considering non-functional properties related to safety, security and performance, including timing and energy constraints.

- **System safety assessment process** must be carried out as part of the development life cycle to determine and categorize the failure conditions of the system (e.g. through a hazard analysis).
  - safety-related requirements are derived as a result of the system safety assessment process, which may include functional, integrity, dependability requirements and design constraints.

- Safety-related requirements are allocated to hardware and software components, thereby specifying the mechanisms required to prevent the faults or to mitigate their effects and avoid the propagation of failures.

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 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 Systems Analysis

“The more confidence one needs in a task execution time bound (the less tolerant one is of missed deadlines), the larger and more conservative that bound tends to become in practice”

MCS state-of-the-art Model (1)

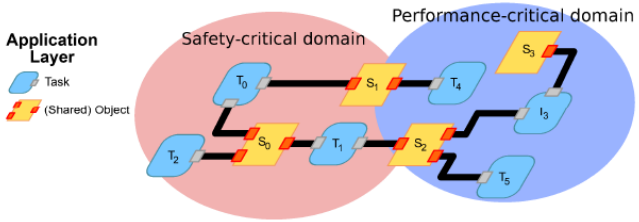
- Almost 200 papers treating of the scheduling of MCS have been referenced in York report, and tens of related papers are still published every year. Most of the works about MCS published by the real-time scheduling research community are based on a **model proposed by Vestal**.
  - System has several modes of execution, say **modes $\{1, 2, \dots, L\}$**. The application system is a **set of real-time tasks**, where each task τ_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**.

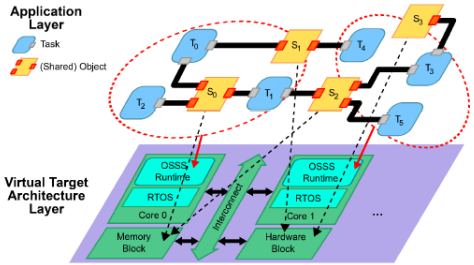
MCS state-of-the-art Model (2)

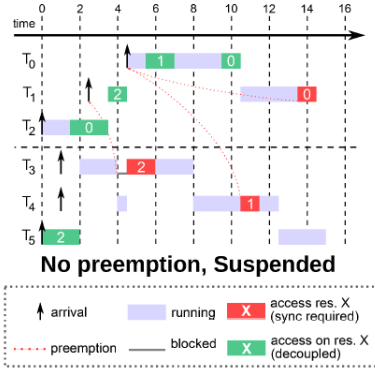
- The system starts its execution in **mode 1** and **all tasks are scheduled** to execute on the core[s]. Then at runtime, if the system is running in **mode *k*** then each time the **execution budget $C_{i,k}$ of a task τ_i is overshot**, the **system switches to mode *k+1***. It results from this transition from mode *k* to mode *k+1* that **all the tasks of criticality not greater than *k*** (i.e., $l_i \geq k$) are suspended. Mechanisms have also been proposed to eventually re-activate the dropped tasks at some later points in time.
  - one of the simplifications of this model is the **Vestal's model** with **only two modes**, usually referred to as **LO** and **HI** modes (which stand for **Low- and High-criticality modes**).
- Multiple variations of that scheduling scheme exist, some for single-core, others for multicore architectures. In the case of multicore, both global and partitioned scheduling techniques have been studied and solutions for **fixed priority scheduling (RM)**, **Earliest Deadline First (EDF)** and **time triggered scheduling** have been proposed in literature.
  - some works also propose to **change the priorities or the periods of the tasks during a mode change** rather than simply stopping the less critical ones.

MCS Design - OFFIS

Mixed-criticality principles applied to application layer modelling objects:







- Safety-critical domain T_0, T_1, T_2, S_0: high-critical tasks and shared objects
- Performance-critical domain T_3, T_4, T_5, S_3: low-critical tasks and shared objects
- mixed-critical shared objects: S_1, S_2

- Ports
  - Connections to shared object **interfaces**
- Deadline
  - might affect scheduling decisions

Mixed-Criticality aware properties:

- Task **periods**
  - Safety-critical tasks might have increased **activation frequencies** in high criticality levels
- **Vector** of computation times
  - platform-dependent, → later
  - One for each criticality level (e.g. *LO*, *HI*)
- **Criticality level**
  - statically defined at design time

Task Definition

Task $T_i \in \tau$,
$T_i = \left(\vec{T}_i, D_i, \vec{C}_i, \pi_i, L_i\right)$

- a **vector** of periods $\vec{T}_i$ (minimum arrival interval)
- D_i: deadline
- $\vec{C}_i$: **vector of computation times** (one for each criticality level)
- π_i: **ports** for connecting to communication objects
- L_i: **criticality** level (e.g. *LO*, *HI*)

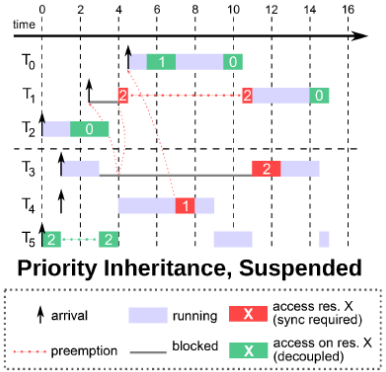
Possible Shared Object realisation should include:

- Cached **internal** state Σ, Σ'
  - before performing the call, create copy: $\Sigma' \leftarrow \Sigma$
  - active calls work on **state copy** Σ'
  - after the call, **update** original state: $\Sigma \leftarrow \Sigma'$
- **Preemptible scheduling** of access, policy defined by attribute Φ (e.g. round-robin, fixed-prio)
  - if criticality level L increases, **abort** active calls $< L$
  - discard Σ'
  - only **allow** calls from tasks with criticality level $\geq L$

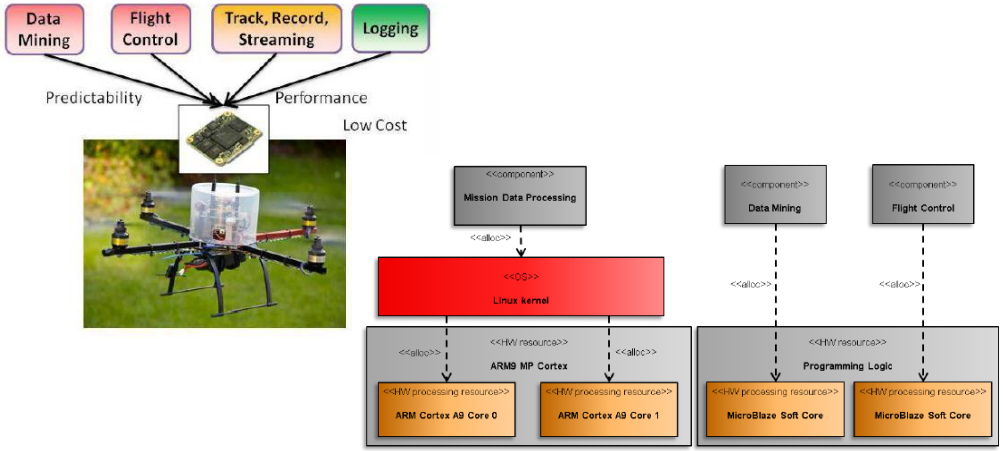
Shared Object

$S_i = (\Sigma, \Sigma', L, M, I, \Phi)$

- $\Sigma_{0,1}$: **inner states** (containing abstract data types),
- L: current criticality level (e.g. *LO*, *HI*)
- $M \subseteq \Sigma \times \Sigma$: a set of **methods** or **services** (e.g. *read*(), *write*())
- $I \subseteq \mathcal{P}(M)$: Interfaces for grouping methods
- Φ: **resource arbitration policy**



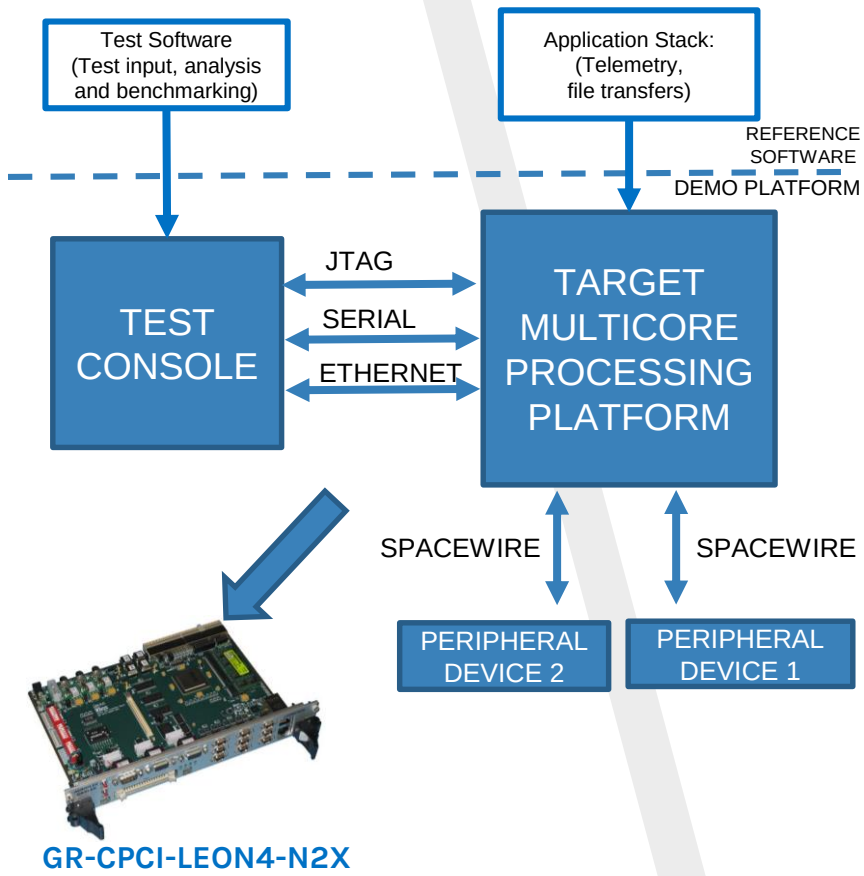
Industrial and Academic MCS Case Study



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.



GR-CPCI-LEON4-N2X

MCS Summary

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

	HW	Single core	Multi-core	Many-core
Spatial	0-level scheduling	0-level scheduling	0-level scheduling	0-level scheduling
		1-level scheduling	1-level scheduling	1-level scheduling
		2-level scheduling	2-level scheduling	2-level scheduling
Temporal	0-level scheduling	0-level scheduling	0-level scheduling	0-level scheduling
		1-level scheduling	1-level scheduling	1-level scheduling
		2-level scheduling	2-level scheduling	2-level scheduling

4.

ESL Methodology

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

Concurrency, Process Calculi and CSP

- Concurrency is the **decomposability property** of a program, algorithm, or problem into order-independent or partially-ordered components or units.

- A number of **mathematical models** have been developed for general concurrent computation (Petri nets, process calculi, the Parallel Random Access Machine model, the Actor model etc.).

- **Process Calculi** (or Process Algebras) are a diverse family of related approaches for formally **modelling concurrent systems**.
  - Communicating Sequential Processes (CSP)
  - The Calculus of Communicating Systems (CCS)
  - The Algebra of Communicating Processes (ACP) and so on.

- **CSP** is based on **message passing via channels** and was highly influential in the design of the OCCAM programming language.

CSP, Timed CSP and model checking

- **Timed CSP** was first proposed in 1986 by **Reed and Roscoe** as a **real-time extension of the process algebra CSP**.
  - Timed CSP **added a single primitive** to the language CSP - **WAIT t, for any time t** - yet differed at the denotation level from the CSP. By syntactically transforming a Timed CSP process into a CSP one (**dropping all WAIT t terms**), **much information is preserved**, and under appropriate conditions **a number of properties can be formally established** of the original Timed CSP process **by studying its untimed counterpart**.
- A number of tools for analyzing and understanding systems described using CSP have been produced.
  - Failures/Divergence Refinement 2 (**FDR2**), which is a commercial model and refinement checker, converts two CSP process expressions into **Labelled Transition Systems** (LTSs), and then determines whether one of the processes is a refinement of the other within some specified semantic model (traces, failures, or failures/divergence).

MCS ESL Context

- **Electronic System level (ESL)** Design Flow for Embedded Systems
  - The main goal is to **model F/NF requirements** and to validate their satisfaction before final implementation
- Use **system-level models** to check HW/SW resources allocation by simulating system behavior
  - Block diagrams, UML, SystemC etc.
- **No mature general methodology** is available to **reduce costs and complexity** of systems realization
  - Use of virtualization
- A critical industrial challenge is to integrate multiple applications with different criticality on a single computing platform
  - **Mixed-Critical Systems (MCS)**

Univaq EMC² – WP2 – T2.4.3

- UNIVAQ-DEWS CONTRIBUTIONS
  - **Semi-Automatic DSE** at the System-Level of Abstraction
  - **Extension of an existing HW/SW Co-Design Methodology** for Parallel Embedded Systems

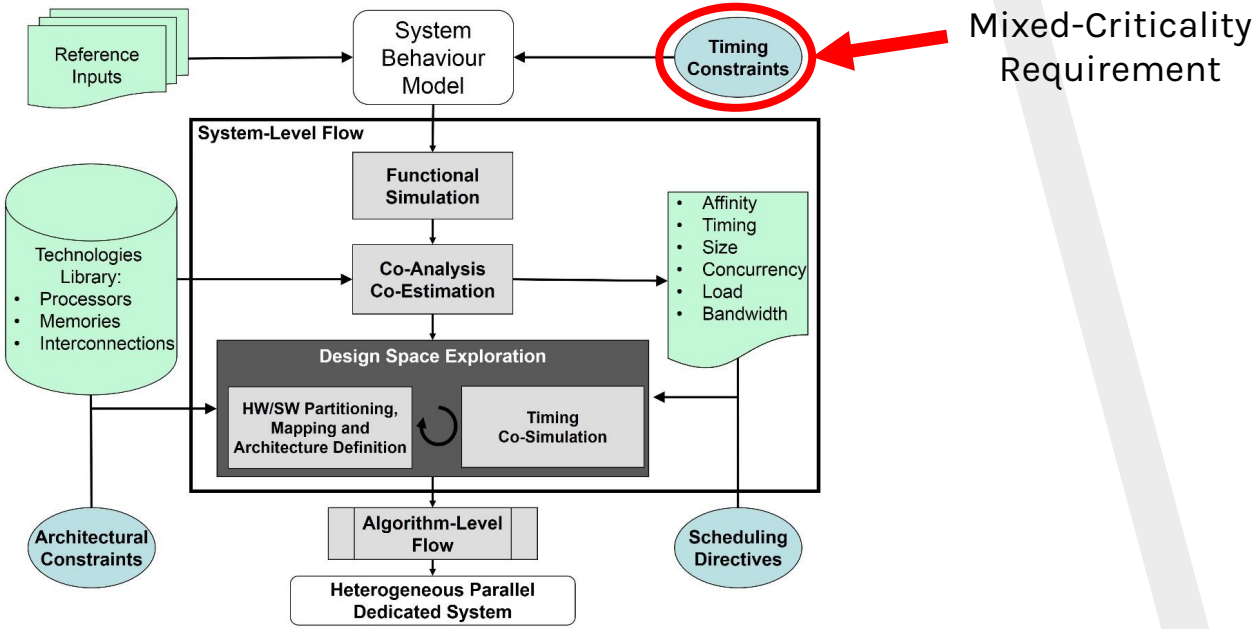
Reference Co-Design Flow

- A System-Level Methodology for HW/SW Co-Design of Parallel Dedicated Systems
  - The proposed methodology **starts from a model of the system behaviour**, based on a **Concurrent Processes MoC** (i.e. a **CSP-like**), and lead to an parallel dedicated system (on-chip or on-board) able to satisfy given F/NF requirements. In particular, the goal is to suggest to designer
    - How **to partition processes** between HW and SW
    - Which **kind of heterogeneous parallel architecture** to use
    - How **to map processes on processor**

- Current NF requirements are related only to timing and some architectural ones but the methodology can be extended to consider other ones (e.g. power/energy, reliability, etc.)
  - In this project we are considering **Mixed-Criticality** requirements

Reference Co-Design Flow

- Set of **models, metrics and tools** that **drives** a designer from specification to implementation
  - The path to be followed is called **Co-Design Flow**



Reference Co-Design Flow – Input

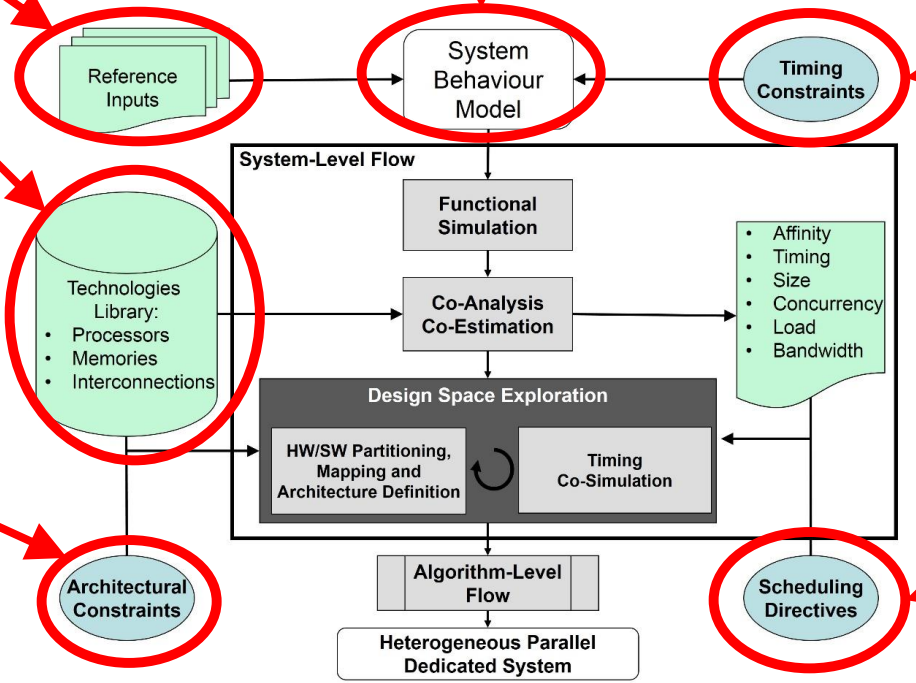
Relevant
Reference Inputs

An executable/simulatable
System Behaviour Model
based on a *Concurrent Processes MoC*

Technologies Library, characterization of:
- Processors, Memories, Links

Non Functional Requirements (**Timing**)
- REAL-TIME Constraints (Time-to-task completion)

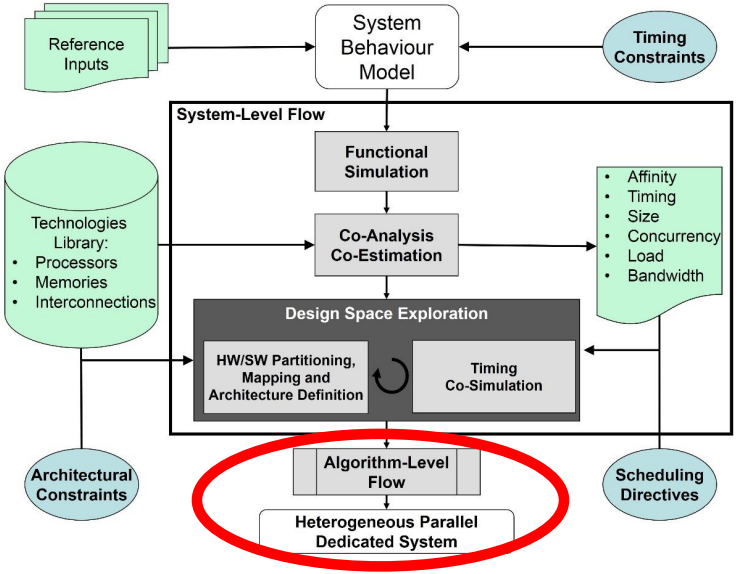
Non Functional Requirements (**Architectural**):
- min/max # processors and links instances
- Total area (or equivalent metrics for FPGA)
- Template architecture



Non Functional Requirements (**Scheduling**):
- Available scheduling policies

Reference Co-Design Flow – Output

- A heterogeneous parallel dedicated system
  - HW/SW partitioning of processes
  - HW/SW Architecture
  - How many processors, which kind, how to connect them, which scheduling policies on SW ones
  - How to map processes on processors



Proposed Framework (1)

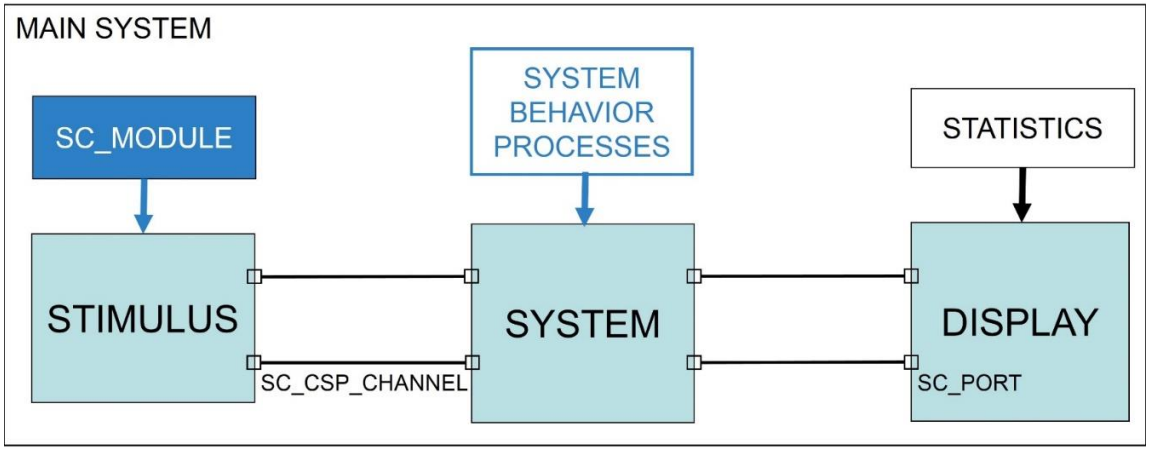
- **ESL Modelling**: The System Behaviour Model (**SBM**) is based on the **Communicating Sequential Processes MoC** (i.e. **a CSP-like**), that allows modeling system behavior as a **network of processes communicating through unidirectional synchronous** (i.e. rendez-vous based) **channels**. Three main CSP subsystems:
  - **Stimulus**: single instance process activation
  - **System**: System Behavior Model (SBM)
  - **Display**: output feedback for offline analysis

- **SystemC Model**: The System Behaviour Model, formed by CSP processes and CSP channels, is described in **SystemC**
  - The single **CSP process is a set of SystemC statements** divided into an **init** and **while(1) loop** sections that realize the functional logic.
  - CSP channels are modeled by a proper **sc_csp_channel**

- **HW/SW Co-Design Flow**: SBM, NF constraints, reference input and technologies library, functional simulation, co-analysis and co-estimation etc…

CSP for Real-Time Application (1)

- CSP-like notation adopted in the reference HW/SW co-design methodology doesn't match very well with the RT world, in term of:
  - **timing constraints**
  - **relations constraints** (e.g. DAG representation of task)
  - mutual exclusion constraints on **shared resources**



Proposed Framework (2)

- The idea is to identify in the SBM a set of elements so classified:
  - s (**statement**): statement with C/SystemC data types
  - p (**process**) or J (job): process or job consisting of a set of s divided into two sections (init and while1) encapsulated in a SystemC SC_THREAD
  - t (**task**): set of potentially competing/cooperating p (with communication rules implemented through CSP-like SC_CSP_CHANNEL) with a given **criticality**
  - k (**component**): component (or subsystem) composed of one or more t encapsulated in a SystemC SC_MODULE with a given **criticality**
  - a (**application**): application (or system) composed of one or more k (mixed-criticality system)

- With these particular objects it is possible to model CSP/SystemC representation of system by means of **a set of tasks** such that:

$t_i = \{J_{i,1}, J_{i,2}, \dots, J_{i,n}\} \equiv \{p_{i,1}, p_{i,2}, \dots, p_{i,n}\}$, n := number of task instance or processes (jobs)

In this way it is possible to apply classical approaches found in the RT world

CSP for Real-Time Application (2)

- **Timing constraints**
  - The real-time parameters can be entered by the designer, and mapped on the CSP-like model
- **Relations constraints** (e.g. DAG representation of task): In general, the CSP approach never create DAG. For this, there are two possible solutions:
  - put all the s of a loop in a super_s so that the while section of each p becomes a DAG that is repeated (a) periodically
  - force the designer to write p as a DAG (by separating init and while into two p) and work on (a) periodical sequence of p in t
- **Mutual exclusion constraints** on shared resources is not yet considered
- **Preemption** has been introduced by **points of preemption** included in **p**, with a wait in the scheduling policy simulation step and proper interactions with a scheduler manager

Design Space Exploration (1)

- 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 behaviour 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**

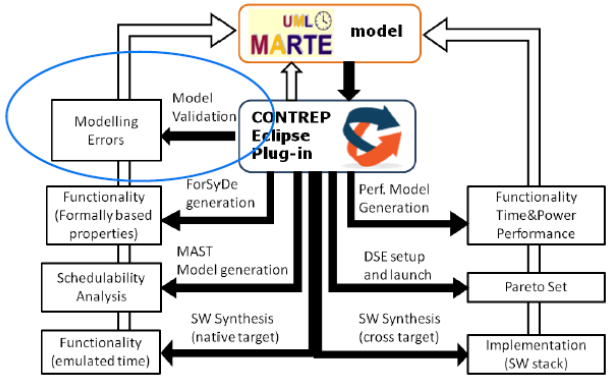
- The final mixed-critical architecture/mapping item is early validated by means of a system-level HW/SW Timing Co-Simulation.

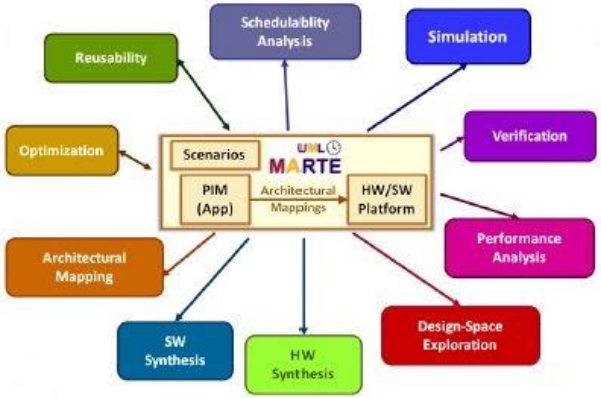
Design Space Exploration (2)

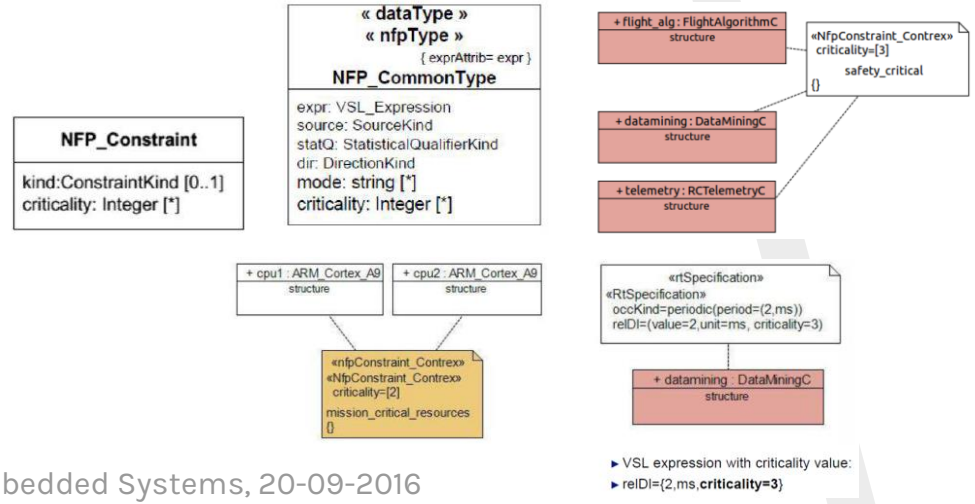
- Main issues:
  - **Extension of** the first-step of **the DSE methodology** for a better management of timing requirements in order to consider also classical RT ones
  - **Analysis of existing HW/SW technologies to** support **mixed-criticality management** (with focus on **hypervisors** technologies) 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

UML/MARTE profile for MCS

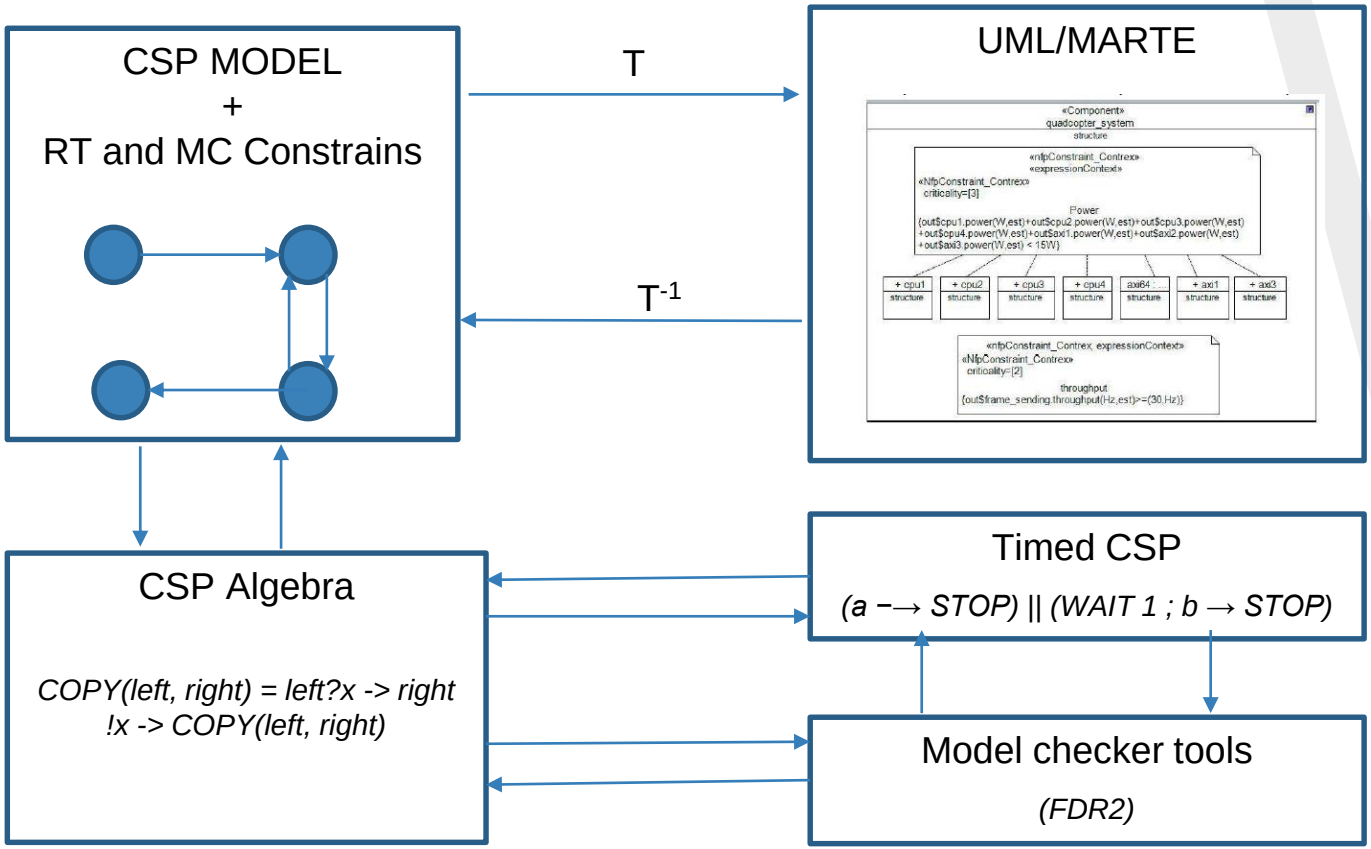
- Criticality: **annotation** that can be associated to:
  - Application (**PIM**) Components
  - **Platform Resources**
  - Extra-Functional Requirements
  - Value annotations
- Enables two basic modelling techniques:
  - **Criticality constraint** associated to **modelling element**
  - Criticality associated to **value**







Possible PhD Work Summary



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 (1)

- 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.).
- The final result will be a methodology able to support mixed-criticality systems developments by suggesting both the platform and mapping solutions for the specific mixed-criticality application

Conclusions and future work (2)

- Work on CSP mathematical model, integrates the Timed CSP in the design model, use model checking techniques in order to validate the initial functional model and perform the static analysis of behavior model.

- Integrate the UML/MARTE performance analysis tool developed by University of Cantabria with CSP Tool in order to validate the estimated analysis and the possible implementation solutions founds during the DSE and Co-simulation step.

- Offer an integrated tool and framework to manage in the right manner MC application.

- Find a meaningful use cases in order to validate the methodologies, comparing outputs with commercial and academic research.

Further work

- Introduce multiple scheduling levels to simulate Hypervisor behavior
- Model GR-CPCI-LEON4-N2X Quad-Core 32-bit LEON4 SPARC V8 processor with MMU, IOMMU
- Model LL3 TASI/UNIVAQ Satellite Application
- Compare mapping between LL3 – UC Platform Application and WP2 – T2.4.3 HW/SW Co-Design Tool

Reference

[1] A. Esper, G. Nelissen, V. Nélis, E. Tovar: "*How realistic is the mixed-criticality real-time system model?*" In: Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS '15). ACM, New York, NY, USA, 139-148, 2015.

[2] 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.

[3] Burns, A, Davis, R.I.: "Mixed Criticality Systems - A Review", University of York, 4 March 2016.

[4] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar. Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In RTNS 2013, RTNS '13, pages 183–192. ACM, 2013.

[5] C. A. R. Hoare. "Communicating sequential processes". Commun. ACM 21, 8 (August 1978), pp. 666-677.

[6] L. Pomante. HW/SW co-design of dedicated heterogeneous parallel systems: an extended design space exploration approach. IET Computers & Digital Techniques, 2013.

[7] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In Proceedings of the Thirteenth International Colloquium on Automata, Languages, and Programming (ICALP 86), pages 314–323. Springer LNCS, 1986.

[8] F. Federici, V. Muttillo, L. Pomante, P. Serri, G. Valente,: "A Model-Based ESL HW/SW Co-Design Framework for Mixed-Criticality System", CPS Week 2016, EMC^2 Summit, Vienna, Austria

[9] F. Herrera, H. Posadas, P. Peñil, E. Villar, F. Ferrero, R. Valencia, and G. Palermo. 2014. The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems. J. Syst. Archit. 60, 1 (January 2014), 55-78

THANKS!

Any questions?