www.bsc.es

Combining High Performance Hardware and Software with High Software Assurance:

Is it Possible?

Dr. Leonidas Kosmidis

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

September 24, 2025

Aerospace systems require high performance

- Modern and upcoming space systems require increasing levels of computing power
- Traditional space processors cannot provide this performance level
- Need for higher performance hardware in space systems

Increase in Complexity

- Modern aerospace systems require new, advanced functionalities
  - Artificial Intelligence (AI)
  - High Resolution Sensors
  - Optical communications
  - Advanced Robotics…

- Advanced functionalities require complex hardware and software compared to the existing space technologies
- High Performance Hardware technologies: Advanced Multi-cores, GPUs, AI accelerators
- Programming high performance hardware requires complex software: parallel and GPU programming

Need for High Assurance

- Safety Critical systems
  - used in automotive, avionics and **aerospace** industries
  - **correct** and **timely** execution is important
  - any malfunction may be dangerous
  - Compliance with functional safety and quality standards e.g. ISO 26262, DO-178C, ECSS, NASA-STD standards

How can we ensure high assurance when complexity increases?

- At the High Performance Embedded Systems Laboratory and in the HArdware Dependability for Embedded Systems (HADES) group we address this issue
  - Build on standardised technologies and reuse
  - Take advantage of different standards across different industries
  - Modify hardware and software when possible to facilitate standard compliance
  - Model-Based Design
  - Formal Methods
  - Safe Languages

Challenge: DO-178 C Certification for GPU code

- Highest Criticality software (DAL-A) needs to follow software design and coding standards used in the development of safety-critical systems such as MISRA-C, Ada SPARK, JPL Institutional Coding Standard and others:
  - Restricted use of Pointers
  - No dynamic memory allocation
  - Static verification of program properties
  - Resilience to faults
  - Fault isolation

Challenge: DO-178 C Certification for GPU code

- Highest Criticality software (DAL-A) needs to follow so… …gn and coding standards used in the development of safety-cri… …such as MISRA-C, Ada SPARK, JPL Institutional Coding St… …ners:
  - Restricted use of Pointers
  - No dynamic memor…
  - Static verific… …gram properties
  - Resi… …ults
  - Fau… …olation

Violated by EVERY CUDA/OpenCL program!

Glass Cockpit

Modern aircraft use LCD screens which have

- replaced analog instruments
- A320: 4 displays
- A350: 6 very large displays
- A380: 10 large displays
- Newer aircraft feature touch screens

- Driven by avionics-grade GPUs
- Rely on DO-178 B/C certified graphics software stacks

Glass Cockpit

Modern aircraft use LCD screens which have

- replaced analog instruments
- A320: 4 displays
- A350: 6 very large displays
- A380: 10 large displays
- Newer aircraft feature touch screens

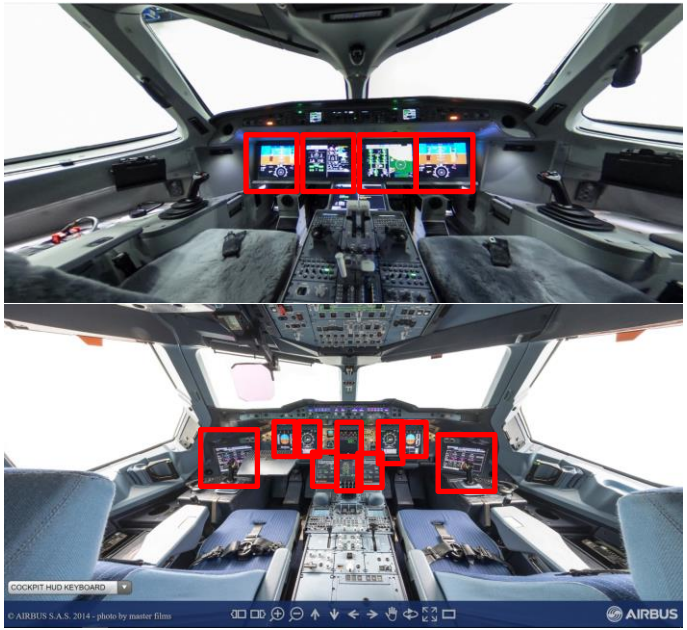
- Driven by avionics-grade GPUs
- Rely on DO-178 B/C certified graphics software stacks

Glass Cockpit

Modern aircraft use LCD screens which have

- replaced analog instruments
- A320: 4 displays
- A350: 6 very large displays
- A380: 10 large displays
- Newer aircraft feature touch screens

- Driven by avionics-grade GPUs
- Rely on DO-178 B/C certified graphics software stacks



DO-178C Certification-ready Graphics-based GPGPU Methods

General Idea:

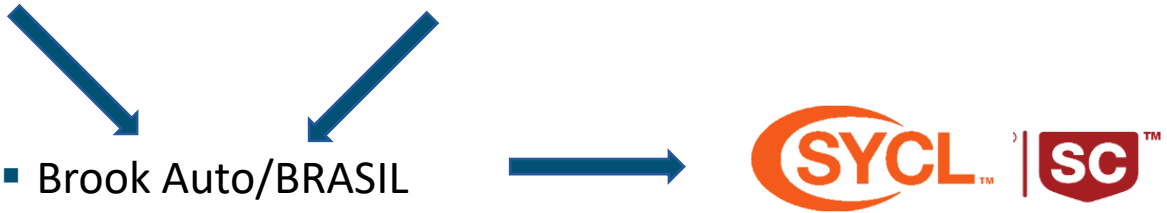
- Leverage certified graphics-based solutions for the acceleration of general purpose computing
  - Use them directly or
  - **Build higher level abstractions on top of them**

Existing Methods:

- OpenGL SC 1.0.1
- OpenGL SC 2.0

Upcoming Methods:

- Vulkan SC



Brook Auto/BRASIL

- **Addresses certification at language level**
- Model-based design / correct-by-construction for error prone GPU operations
  - Same approach followed nowadays by SYCL
- Open-source GPU programming language for safety critical systems [2]
- Subset of the Brook programming language [1], similar to CUDA
  - Restricted subset of C (no recursion, no goto, no pointers, enforcement of loop bounds)
- Source-to-source compilation to safety critical graphics APIs
- Multiple backends (CPU, multicore, vectorisation, embedded graphics)
  - Supports **almost any** parallel platform considered for safety critical systems

[1] I. Buck et al, Brook for GPUs, SIGGRAPH 2004

[2] Brook Auto: High-Level Certification-Friendly Programming for GPU-powered Automotive Systems [DAC'18], https://github.com/lkosmid/brook

Brook Auto/BRASIL vs CUDA Example

```
#define MAX_ITERS 10000

kernel void foo(float a<>, float b[], out c<>){
 float acc=0.0;
 for(int i<0; i < a || i < MAX_ITERS; i++){
   acc +=  b[indexof(c).x];
 }

 c = a + acc;
}

int main(void){
 float a_h[100], b_d[100], c_h[100];
 float a_d<100>, b_d<100>, c_d<100>;

 streamRead (a_d, a_h);
 streamRead (b_d, b_h);
 foo (a_d, b_d, c_d);
 streamWrite (c_d, c_h);
}
```

```
__global__ void  foo(float * a, float * b, float * c){
 unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;
 float acc=0.0;
 for(int i < 0; i < a[tid]; i++)
   acc += b[tid];

 c[tid] = a[tid] + acc;
}

int main(void){
 float a_h[100], b_d[100], c_h[100];
 float * a_d, * b_d, * c_d;

 cudaMalloc(&a_d, 100*sizeof(float));
 cudaMalloc(&b_d, 100*sizeof(float));
 cudaMalloc(&c_d, 100*sizeof(float));

 cudaMemcpy(a_d, a_h, 100*sizeof(float), cudaMemcpyHostToDevice);
 cudaMemcpy(b_d, b_h, 100*sizeof(float), cudaMemcpyHostToDevice);
 foo<<<1,100>>>(a_d, b_d, c_d);
 cudaMemcpy(c_h, c_d, 100*sizeof(float), cudaMemcpyDeviceToHost);
}
```

Brook Auto/BRASIL

BRASIL

- **Improvement of the Brook Auto toolchain to address tool qualification**
- Possible thanks to its small code base
- Assessment according to ISO 26262 [2]
  - ASIL (Automotive Safety Integrity Level): High (D)
  - Tool Confidence Level (TCL) 3
- Not same as DO-330 for but it relies on the same concepts:
  - Extensive checks of the generated code
  - Full source code traceability to facilitate manual code inspection

[1] Brook Auto: High-Level Certification-Friendly Programming for GPU- powered Automotive Systems [DAC'18], https://github.com/lkosmid/brook

[2] BRASIL: A High-Integrity GPGPU Toolchain for Automotive Systems [ICCD'19]

Prototype Avionics Application on Brook Auto/BRASIL [1]

- Prototype GPU application provided by Airbus Defence and Space, Madrid within the Airbus TANIA-GPU Project ADS (E/200)
  - Consists of both graphics and compute parts
  - OpenGL SC 2 and Brook Auto implementations
  - Both provide identical outputs and exceed the screen refresh rate (60fps)
- Realistic Industrial Experimental Setup provided by CoreAVI:
  - Commercial, certified OpenGL SC 2 driver
  - Avionics-grade AMD E8860 GPU
- HIPEAC Technology Transfer Award 2019
- Bronze Medal ACM Student Research Competition at ICCAD 2020

[1] Comparison of GPU Computing Methodologies for Safety Critical Systems: An Avionics Case Study [DATE'21]

Formal Methods for GPU Software Development Using Ada SPARK

- ESA funded project
- Focused on the use of Ada SPARK backend for NVIDIA GPUs and AdaCore's formal methods tools in order to increase GPU software assurance
- Prove code correctness
- Find hidden bugs
- Prove code properties

For more details check our SWPA 2023 presentation and DATE 2024 publication

Leveraging Automotive Standards and Components

- ASIL2ECSS - Reusing Automotive Certification and Qualification Standards (https://nebula.esa.int/4000136128) ESA funded project
- Analysis of Automotive (ISO 26262, AEC-Q) and European Space Standards (ECSS) to identify the additional steps required to qualify automotive qualified hardware and software components for use in space

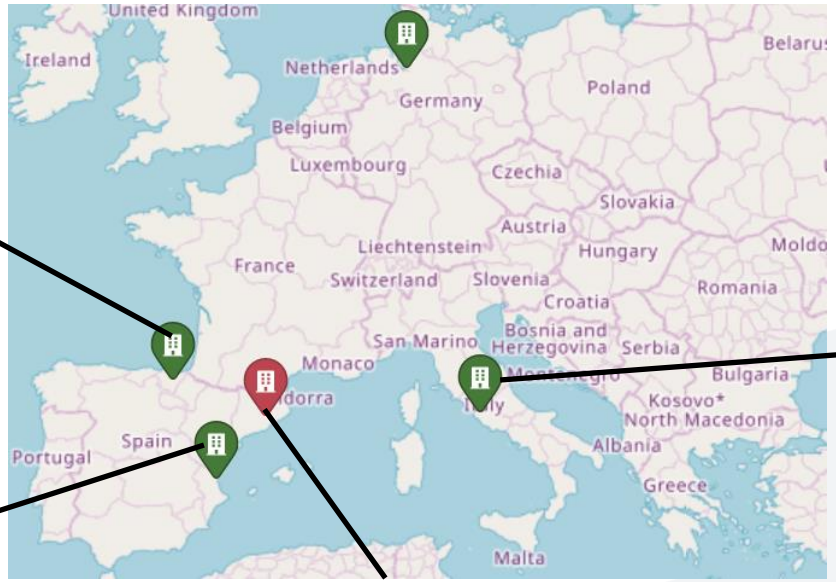
For more info check our upcoming ASIL2ECSS paper and presentation at ESA's European Data Handling & Data Processing Conference (EDHPC) 2025 in October

- Outcomes:
  - Automotive products have much higher quality than regular COTS
  - Reproducible production and similar tests with space but with less margins (e.g. smaller temperature ranges or samples)
  - Reliability features included for functional safety (lockstep, ECC, watchdogs etc) are good for radiation performance
  - But Automotive products are only tested with neutrons, so radiation tests need to be repeated with space relevant radiation sources: proton, heavy ions

The METASAT Project Consortium

- 2-year Horizon Europe project: January 2023-December 2024
- TRL 3-4

OHB



ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Collins Aerospace

fentISS

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

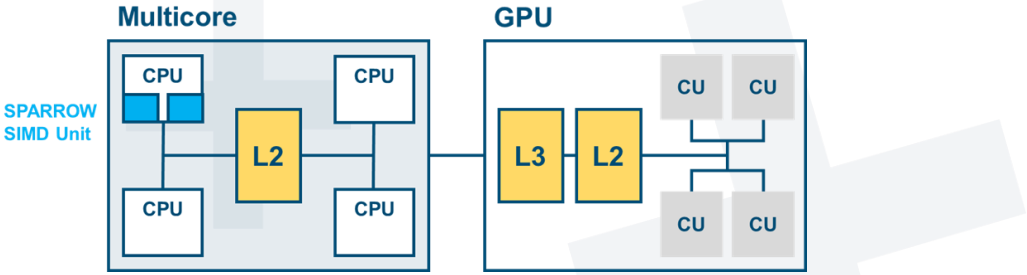
METASAT has received funding from the European Union's Horizon Europe programme under grant agreement number 101082622.

METASAT Overview

- Use a complex, highly capable space processor SoC
- Integrate multiple functionalities in a single platform
  - Similar to the Integrated Modular Avionics concept (IMA) in avionics
- Hardware cost reduction
- Mixed Criticality support through time and space partitioning
  - Software qualification cost reduction
- Use Model-Based Design to manage complexity

Hardware Selection

- No hardware with high-performance and architectural complexity exists for the space domain
- COTS Embedded Multicore and GPU devices provide these features but depend on non-qualifiable software stacks
  - GPU drivers available only for Linux
  - Blocking point for use in institutional missions where high assurance is required
- Design a prototype hardware platform based on the RISC-V ISA



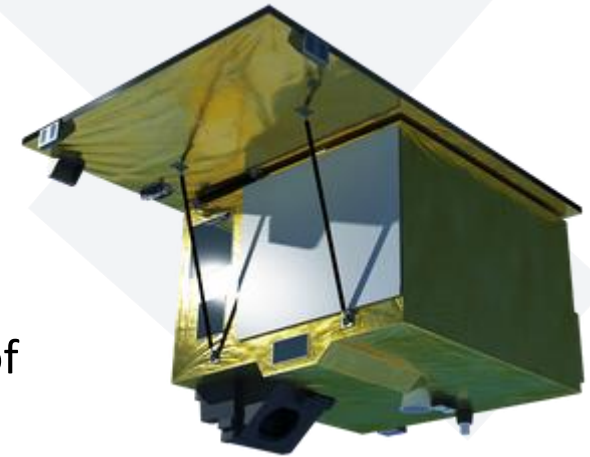
METASAT Use cases and final demonstrator [1]

- Several independent use cases
  - Different processing and acceleration requirements
- Representative of different flight software criticalities
- All use cases were integrated in a single platform
  - High degree of integration was achieved

[1] Mixed-Criticality Flight Software Integration In a High Performance RISC-V Space Platform, SMC-IT 2025

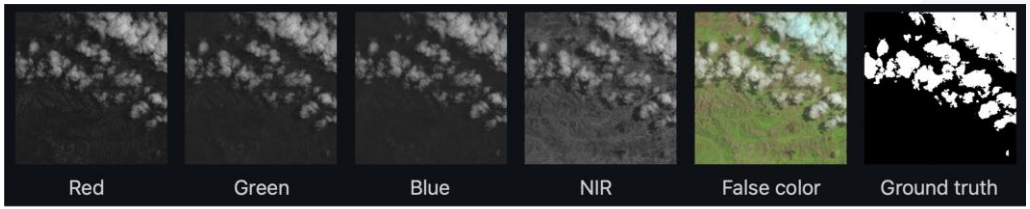
Project Use Cases

- 3 Project Use cases were implemented
  - High degree of integration
  - Distributed over 8 partitions executed together
- OHB/DLR Use Case - #UC1
  - Hardware interlocking – ILSWA, ILSWB
    - Protect against 2 types of wrong software behaviour
  - Implemented interlocks at software level instead of hardware
    - Reduced cost
  - Instrument Control Software
  - Implemented AI Based FDIR
  - Housekeeping data from ENMAP



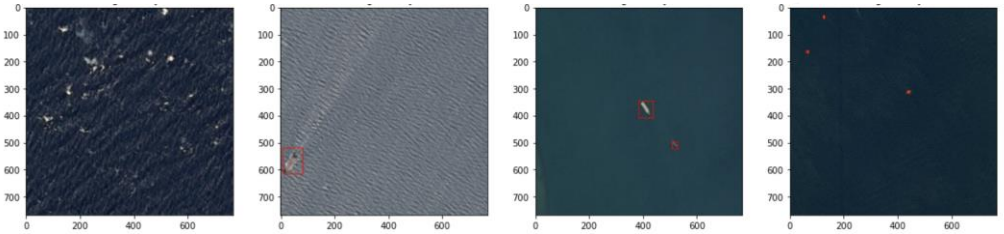
Project Use Cases

- 2 BSC provided use cases based on ESA's OBPMark-ML Open Source Benchmarking suite
- Cloud screening



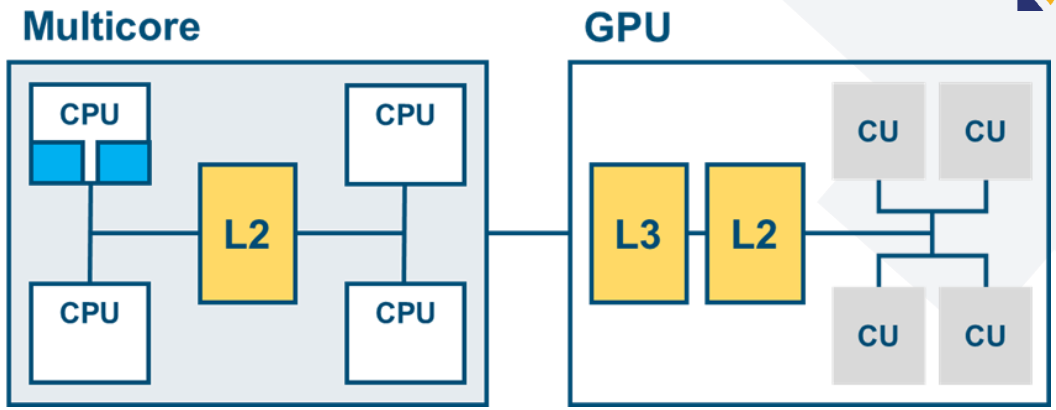
4 Channels RGB/NIR mapped to binary mask (cloud/no cloud)

- Ship Detection



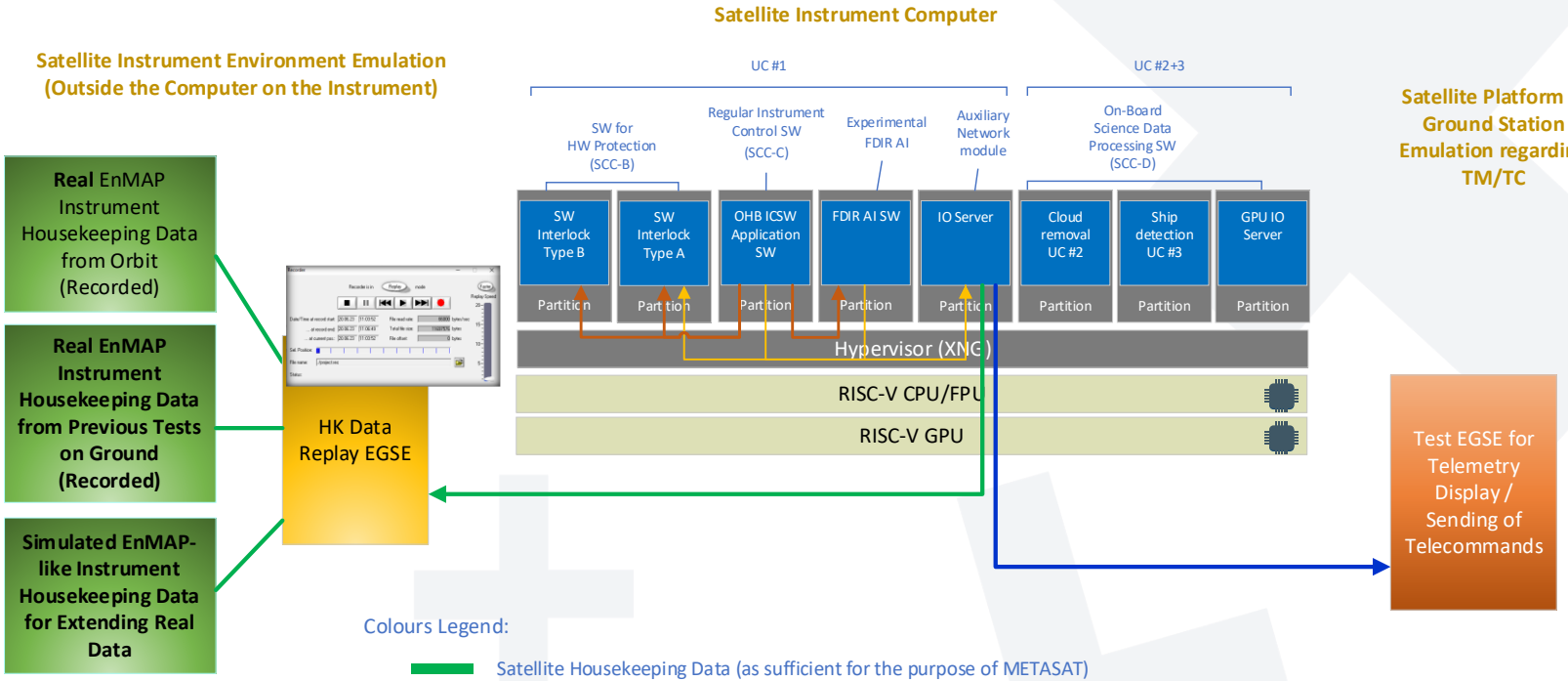
- Accelerated on the SPARROW and GPU

The METASAT Ecosystem Overview

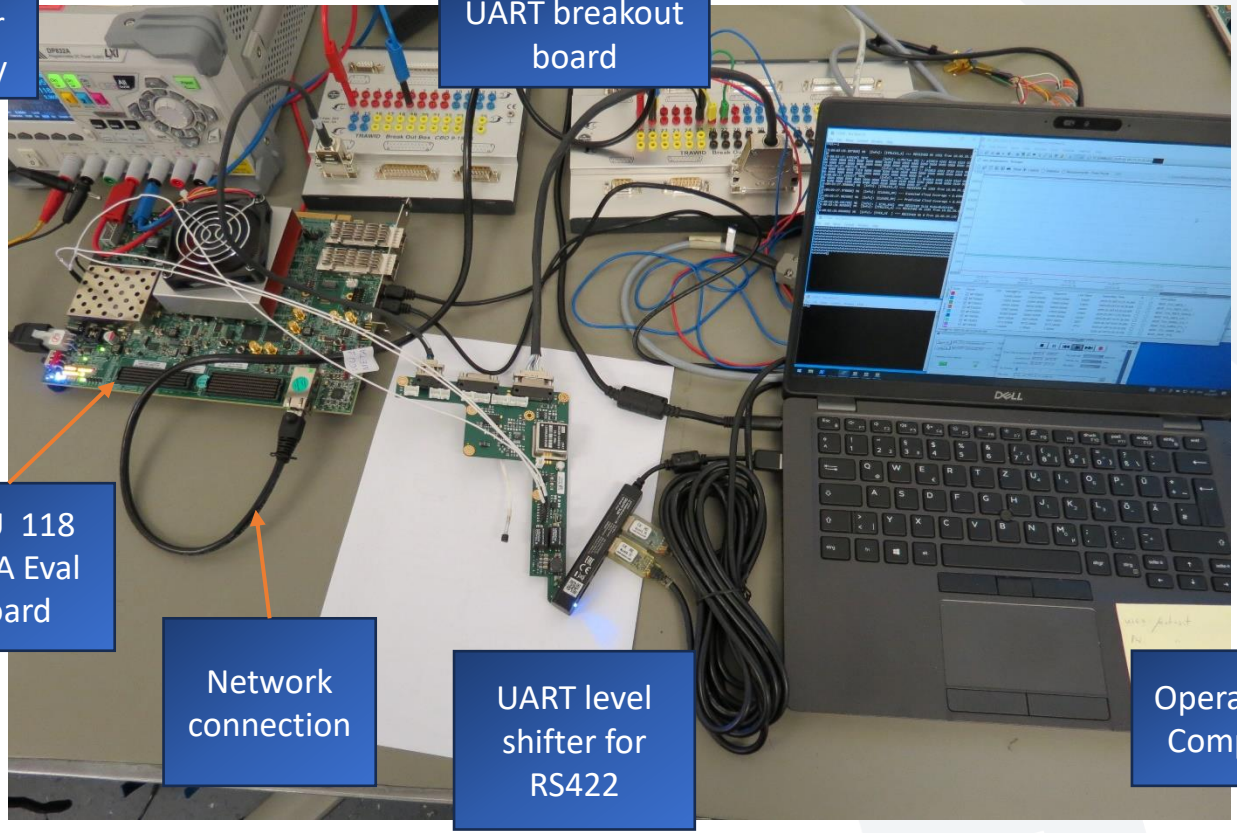


- Qualifiable Software Stack: accelerators can be used from bare metal or RTEMS SMP
- Mixed-criticality: TSP support
- Added support in Model-based design tools
- Standard-based Digital Twin framework

METASAT Use Cases Architecture



METASAT Platform Laboratory Setup



Addressing Hardware Qualification Challenges

Reuse:

- Multicore NOEL-V
- SPARROW AI accelerator
  - Minor modifications on core, retain backwards code compatibility
  - Reuse functional and timing results of existing code
- Vortex GPU
- GRETH ethernet controller
- 2 UARTS: emulation of controlled devices through I/O

- Fully functional FPGA prototype on AMD/Xilinx VCU118
- Fully functional Digital Twin

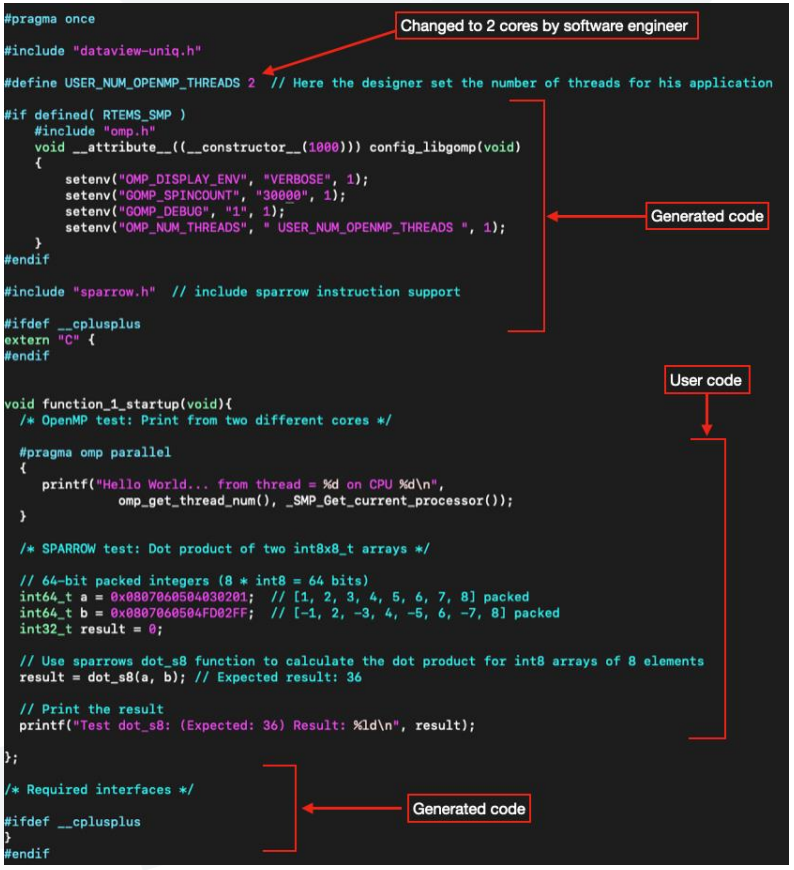
Model Based Design for SW 1/2

taste

METASAT

Model Based Design for RISC-V and Multicores

- Added TASTE support for:
  - RISC-V/NOEL-V
  - SPARROW and OpenMP code generation configuration for RTEMS
  - XtratuM inter-partition communication modeling and multicore partition configuration generation

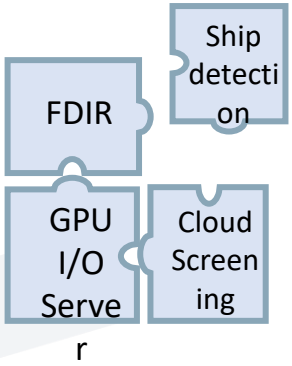



```
#pragma once

#include "dataview-uniq.h"

#define USER_NUM_OPENMP_THREADS 2  // Here the designer set the number of threads for his application

#if defined( RTEMS_SMP )
    #include "omp.h"
    void __attribute__((__constructor__(1000))) config_libgomp(void)
    {
        setenv("OMP_DISPLAY_ENV", "VERBOSE", 1);
        setenv("GOMP_SPINCOUNT", "30000", 1);
        setenv("GOMP_DEBUG", "1", 1);
        setenv("OMP_NUM_THREADS", " USER_NUM_OPENMP_THREADS ", 1);
    }
#endif

#include "sparrow.h"  // include sparrow instruction support

#ifdef __cplusplus
extern "C" {
#endif


void function_1_startup(void){
  /* OpenMP test: Print from two different cores */

  #pragma omp parallel
  {
     printf("Hello World... from thread = %d on CPU %d\n",
              omp_get_thread_num(), _SMP_Get_current_processor());
  }

  /* SPARROW test: Dot product of two int8x8_t arrays */

  // 64-bit packed integers (8 * int8 = 64 bits)
  int64_t a = 0x0807060504030201;  // [1, 2, 3, 4, 5, 6, 7, 8] packed
  int64_t b = 0x0807060504FD02FF;  // [-1, 2, -3, 4, -5, 6, -7, 8] packed
  int32_t result = 0;

  // Use sparrows dot_s8 function to calculate the dot product for int8 arrays of 8 elements
  result = dot_s8(a, b); // Expected result: 36

  // Print the result
  printf("Test dot_s8: (Expected: 36) Result: %ld\n", result);

};

/* Required interfaces */

#ifdef __cplusplus
}
#endif
```

Model Based Design for SW 2/2

Model-Based Design for Accelerators

- TensorFlow Micro Support for SPARROW and Vortex GPU
- TensorFlow Lite code generation from MATLAB/Simulink to TensorFlowMicro
- Accelerated layers in SPARROW intrinsics and Vortex
- Bare Metal, RTEMS and XRE
- Seamless integration with the GPU Server
  - No changes in the integrated partitions
  - Plug and Play



Conclusions

- High complexity in safety critical hardware and software is here to remain
- There are ways to manage complexity and still obtain high assurance
- No need to reinvent the wheel, we can build on existing standards and incremental improvements
- Model based solutions


Thank you!
Questions?
leonidas.kosmidis@bsc.es