Leveraging hardware emulation to accelerate SoC verification in multi-physics automotive simulation environment via the Functional Mock-up Interface

Pierre-Guillaume Le Guay, CEA, LIST
Henrique Vicente De Souza, CEA, LIST
Caaliph Andriamisaina, CEA, LIST
Emmanuel Molina Gonzalez, CEA, LIST
Tanguy Sassolas, CEA, LIST

accellera SYSTEMS INITIATIVE

list cea tech

SYNOPSYS®

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

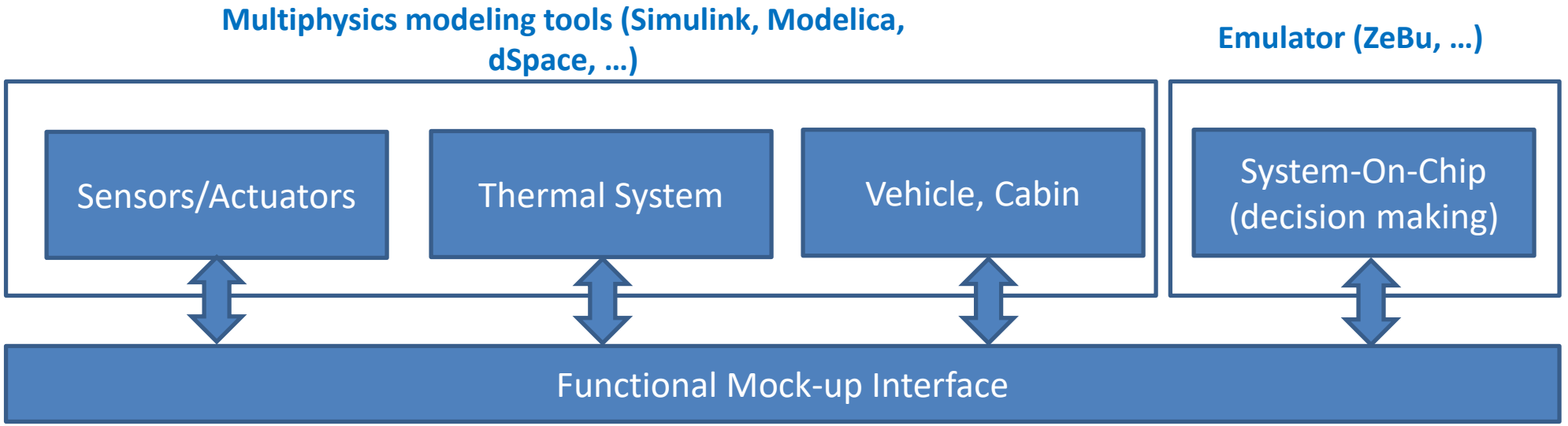
- Context
- Integration of hardware emulation in automotive validation flow
- OpenModelica and ZeBu coupling

Context (1/2)

- The growing complexity and functional demand of automotive HW/SW requires extensive verification effort
  - Verification effort increases to guarantee fault-free functionality to satisfy qualification requirements
  - Development cycles shorten and restrictive time-to-market obligations require advancements for SW development and verification.
  - To meet these needs, hardware emulation solutions have emerged as verification solutions.

- The design of modern automotive SoCs requires also the use of multiple simulation domain tools to validate the system in its future environment with all its external interactions.
  - It is now becoming necessary to build hybrid co-simulation models.

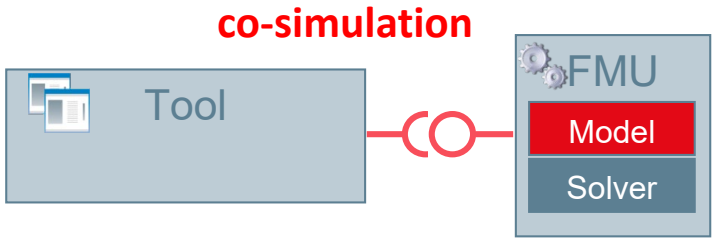
Context (2/2)

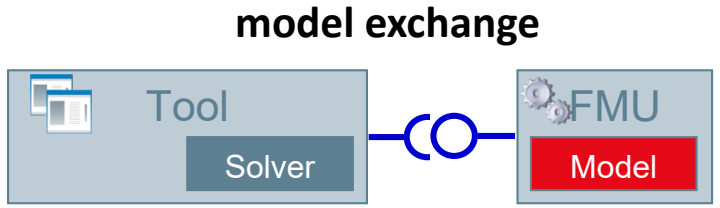


How to integrate hardware emulation like ZeBu in the automotive validation flow?

What is FMI?

- Open interface standard for model exchange between different modeling and simulation environments
  - A component implementing the FMI interface is known as **Functional Mock-up Unit (FMU)**
- The FMI standard supports





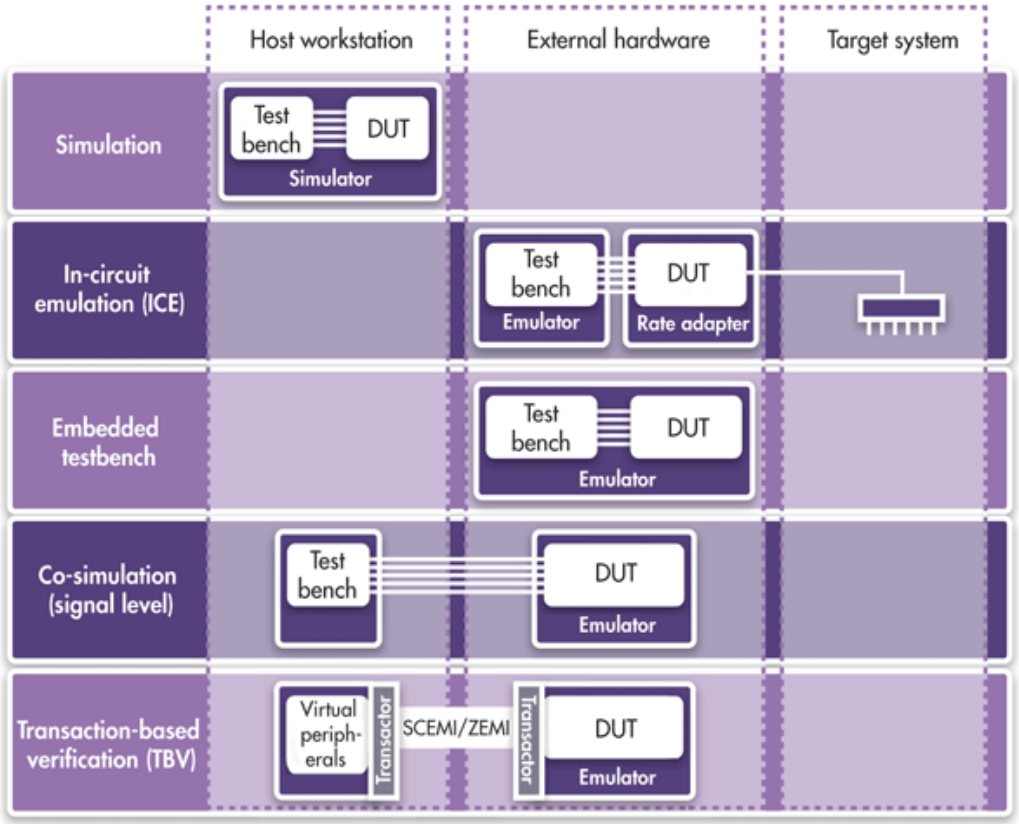
  - Tool: a master which controls the data exchange between FMUs
- A FMU package consists of
  - Model description file
    - a XML file containing the definition of all exposed variables in the FMU and other static information
  - FMU model implementation
    - in form of source code and/or pre-compiled shared libraries.
  - Additional files
    - model icon (bitmap file), documentation files, maps and tables needed by the FMU, and/or all object libraries or dynamic link libraries that are utilized.

FMI execution steps

Steps	Functions	Description
Instantiation	*fmi2Instantiate()*	FMU instance creation
Initialization mode	*fmi2EnterInitializationMode()*	• FMU notification to perform its internal model's initialization • Possibility to set input variables with ***fmi2SetXXX*** and to get output variables with ***fmi2GetXXX*** (XXX corresponds to the variable type)
Runtime	*fmi2DoStep(fmi2Component c, fmi2Real currentCommunicationPoint, fmi2Real communicationStepSize,…)*	• Slave initialization and co-simulation computation. • The calculation is performed until the next communication point • fmi2DoStep function is called periodically (communication step) by the master until the simulation ends
Termination	*fmi2Terminate()*	Retrieving the simulation solution and terminate the simulation

Commonly used emulation modes (1/2)

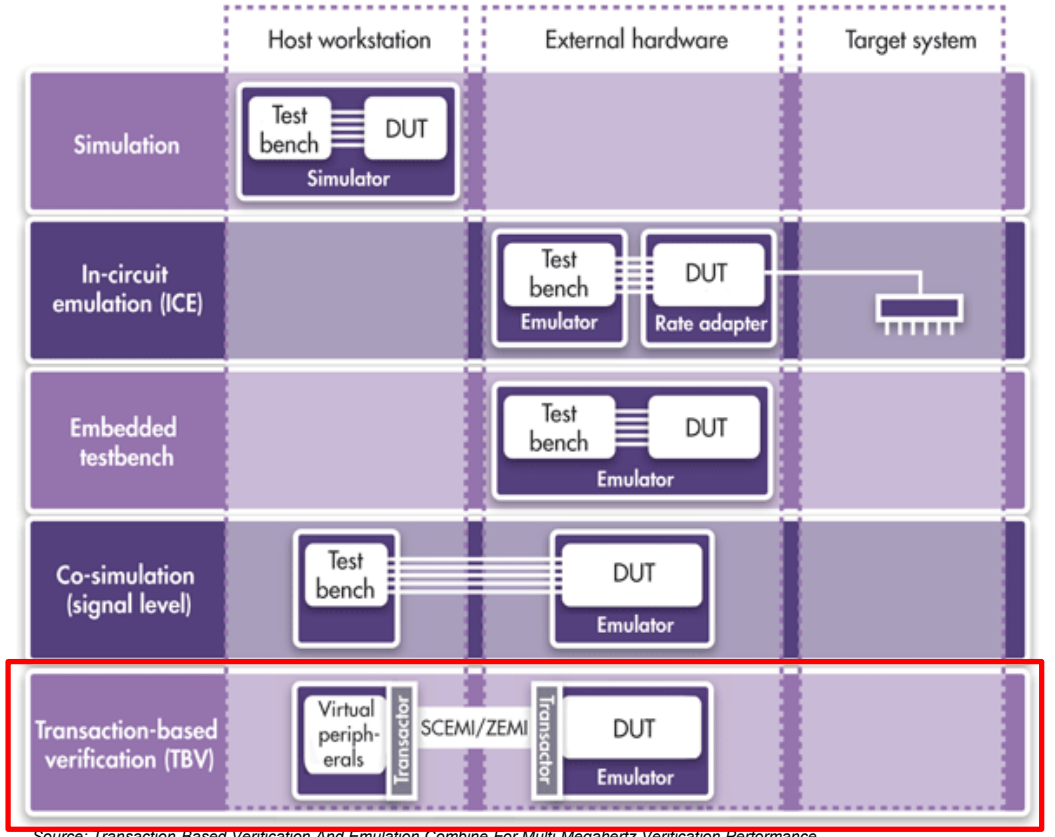
- In-circuit emulation (ICE)
  - Pros
    - Allow to directly connect physical devices to the emulator
    - Eliminate host PC communication
  - Cons
    - Rate adapter development can be complex
- Embedded testbench
  - Pros
    - Eliminate host PC communication
  - Cons
    - Testbench has to be synthesizable



Source: Transaction-Based Verification And Emulation Combine For Multi-Megahertz Verification Performance

Commonly used emulation modes (2/2)

- Co-simulation
  - Pros
    - Easy to set up => use the existing testbench
  - Cons
    - Performance limited by the number of signals to exchange between host and emulator
- Transaction-based verification (TBV)
  - Pros
    - Raises the level of verification abstraction
    - Simplifies the communication between the testbench and DUT
  - Cons
    - Need to develop protocol-specific transactors



Source: Transaction-Based Verification And Emulation Combine For Multi-Megahertz Verification Performance

ZeBu transaction-based verification (TBV) execution steps

Steps	Functions	Description
Initialization	*open(location);* *init();* *getClock("clock");* *connect();*	• Mandatory operations • Open and initialize the ZeBu board • Get the clock driver • Depending on the component used in the design, configuration phases might be required • AXI transactor configuration phase (setting the data bit-width, the address bit-width) etc of the AXI interface. • Several operations, like memory load, signals dump and so on, can be included in this step.
Runtime	*run(cycles);* *speed = InVar;* *OutVar = accel;*	• Running the emulation for a defined amount of cycles. • The current value of variables can be get or set during this step.
Termination	*free(pt);* *close();*	Free pointers, …, close the ZeBu board

FMU to ZeBu adaptation: functions

	FMI functions	ZeBu functions

Initialization	`fmi2Instantiate(…)`
	`fmi2EnterInitializationMode(…)`
		`zebu = Board::open(workLocation);`
		`zebu->init();`
		`zebu->getClock("top.clk");`
		`zebu->getDriver("top.dut_cosim");`
		`driver->connect();`
		`*reg_speed = zebu->getSignal("top.reg_speed");`
		`*reg_accel = zebu->getSignal("top.reg_accel");`
		`*reg_brake = zebu->getSignal("top.reg_brake");`

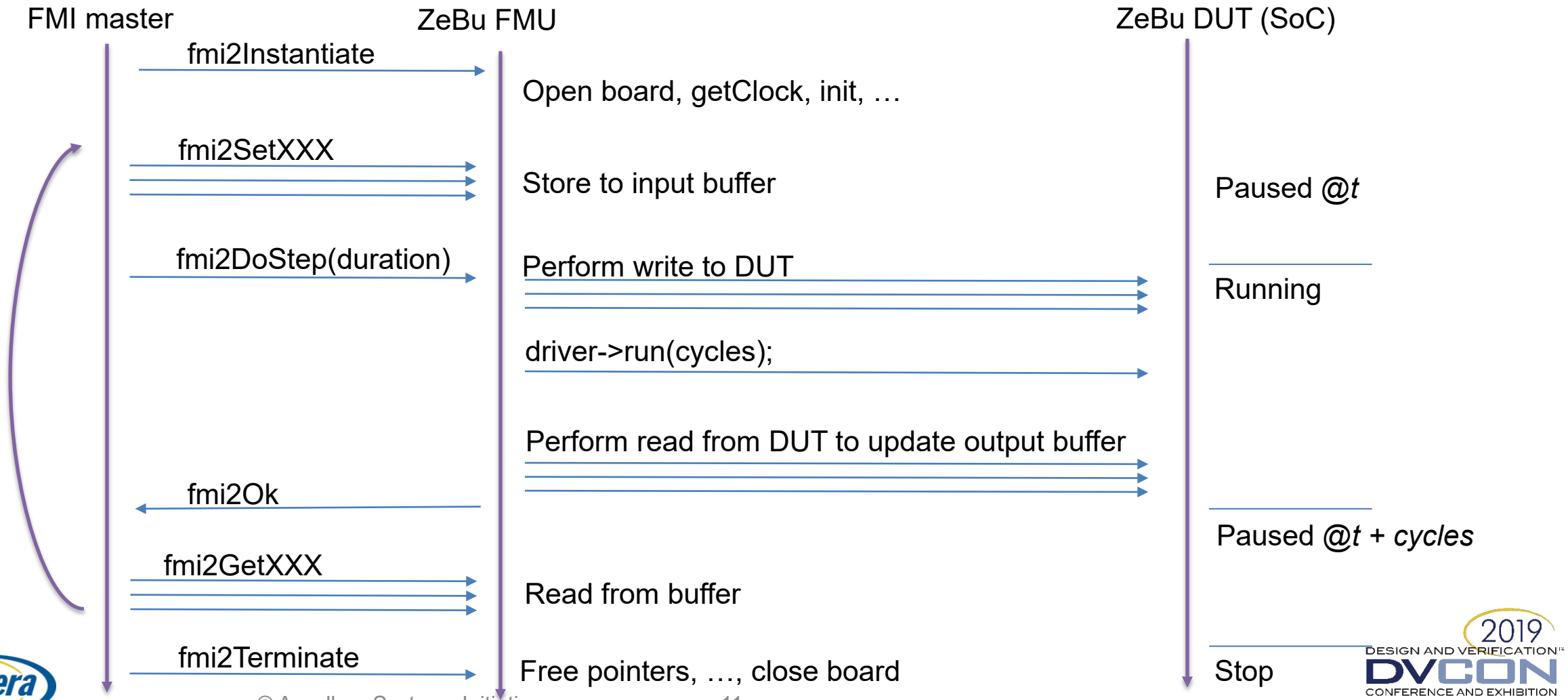
Runtime	`fmi2DoStep(…,duration,…)`	`driver->run(cycles);`

Reading data	`fmi2GetInteger(…)`
	`fmi2GetReal(…)`
	…
		`OutVar[1] = *reg_accel;`
		`OutVar[3] = *reg_brake;`

Writing data	`fmi2SetInteger(…)`
	`fmi2SetReal(…)`
	…
		`speed = InVar[2];`
		`*reg_speed = speed;`

Termination	`fmi2Terminate()`	`zebu->close();`

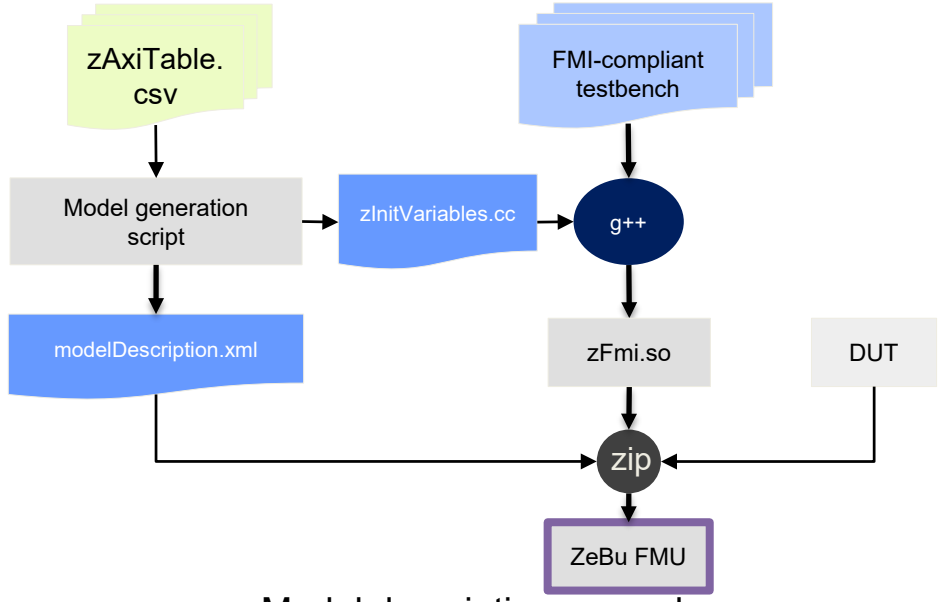
FMU to ZeBu adaptation: synchronization



Automatic generation of ZeBu FMU

- Inputs
  - zAxiTable.csv contains
    - Name, address, direction (input or output), type, intial value, description and dependency between variables
  - FMI-compliant testbench
    - Specialization of the FMI functions to implement the testbench
- Model generation script generates
  - C++ file (zInitVariables.cc)
    - Set the address and initialize the value of all variables
  - XML file (modelDescription.xml)
    - FMI description of available interfaces
- ZeBu FMU package
  - zFmi.so – shared library
  - modelDescription.xml – XML-based model description
  - DUT backend folder – optional other resources
    - FPGA bitstreams

Automatic FMU generation flow



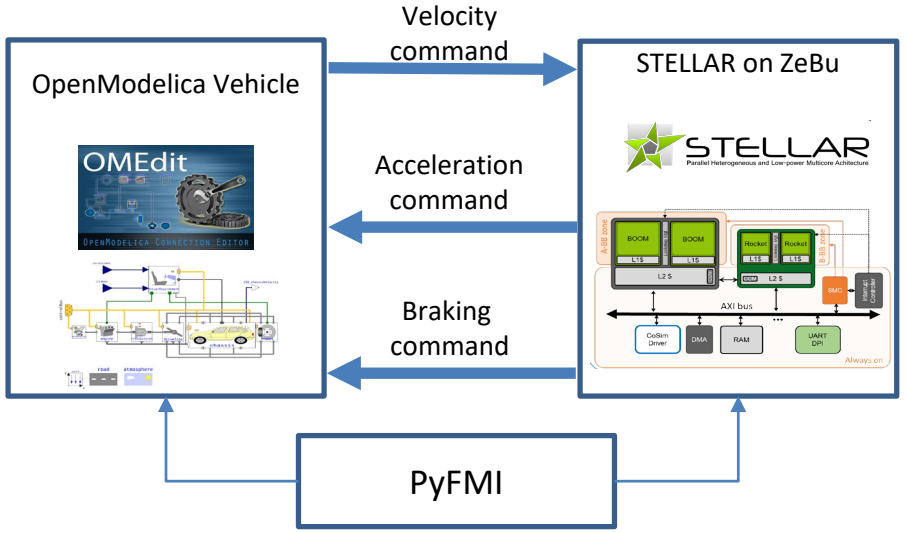
Model description example

```
<fmiModelDescription variableNamingConvention="flat" modelName="zFMU"
generationDateAndTime="2018-11-30T13:49:27.249948" description="FMU ZeBu" numberOfEventIndicators="0"
fmiVersion="2.0" guid="737a13def6464e0191e5704890233ad1" generationTool="zFmiCsvParserToXml.py">
  <CoSimulation canHandleVariableCommunicationStepSize="true" canSerializeFMUstate="false"
canGetAndSetFMUstate="false" canInterpolateInputs="false" maxOutputDerivativeOrder="0"
providesDirectionalDerivative="false" canNotUseMemoryManagementFunctions="true" canRunAsynchronuously="true"
canBeInstantiatedOnlyOncePerProcess="true" modelIdentifier="zFMU" needsExecutionTool="false"/>
  <ModelVariables>
    <ScalarVariable name="velocity" valueReference="1" description="Vehicle Chassis Velocity" causality="input"
variability="discrete" initial="approx">
      <Real start="0"/>
    </ScalarVariable>
    <ScalarVariable name="brake" valueReference="2" description="Vehicle Brake" causality="output"
variability="discrete" initial="approx">
      <Real start="0"/>
    </ScalarVariable>
    <ScalarVariable name="accelerate" valueReference="3" description="Vehicle Acceleration" causality="output"
variability="discrete" initial="approx">
      <Real start="0"/>
    </ScalarVariable>
  </ModelVariables>
  <ModelStructure>
    <Outputs>
      <Unknown index="2" dependencies=""/>
      <Unknown index="3" dependencies=""/>
    </Outputs>
  </ModelStructure>
</fmiModelDescription>
```

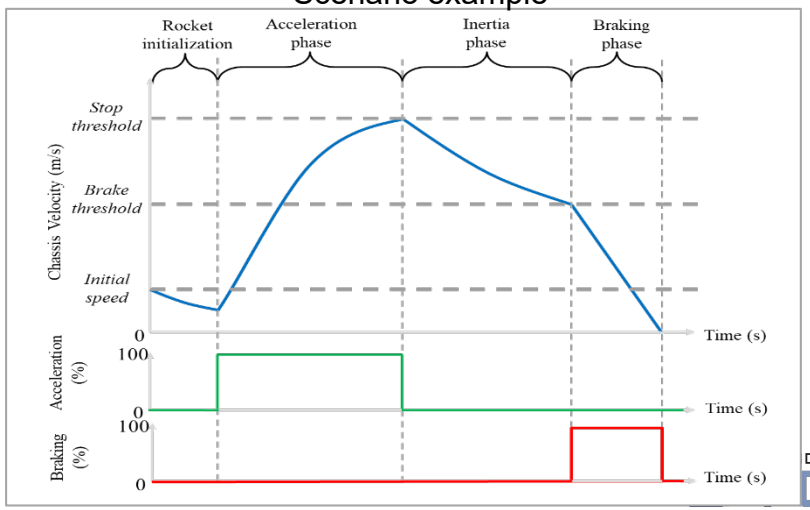
EXPERIMENTS

Overview

- Evaluation based on a multi-physics environment
  - a FMU vehicle model in OpenModelica
  - a multi-core processor based on RISC-V rocket core emulated on ZeBu (ZeBu FMU)
- Data exchange and the synchronization between the two FMUs controlled by PyFMI master
- Experiment's goal
  - To control the vehicle speed of the FMU vehicle with one rocket core

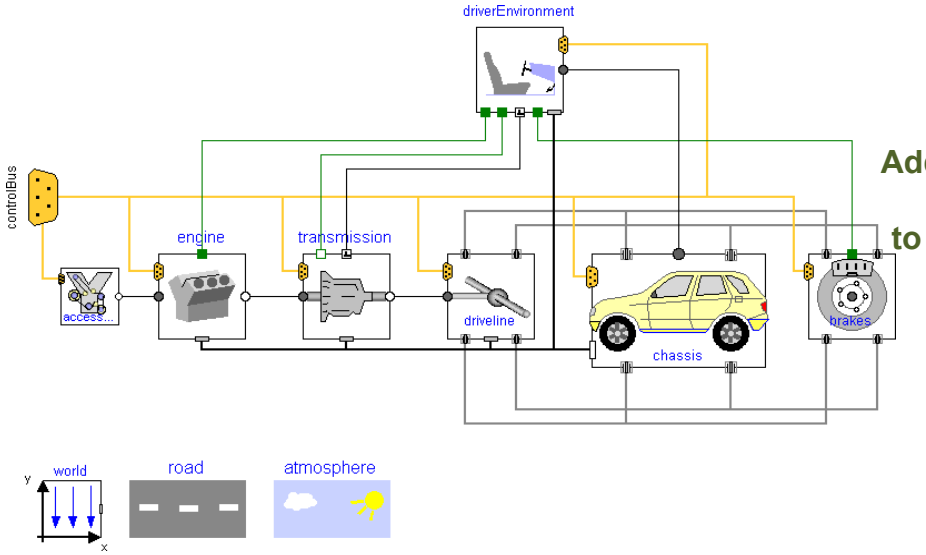


Scenario example



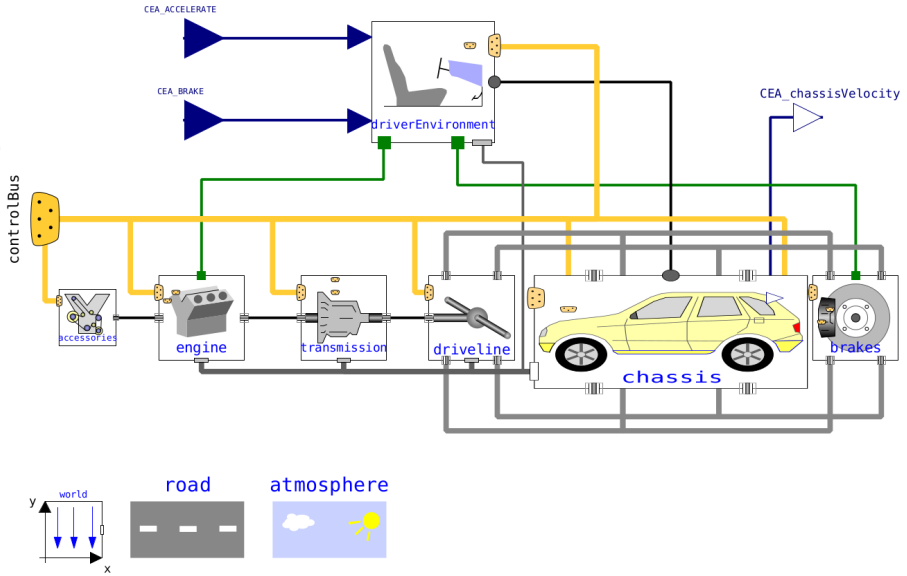
Vehicle model in OpenModelica

Initial model



Adding inputs/outputs to communicate to external components

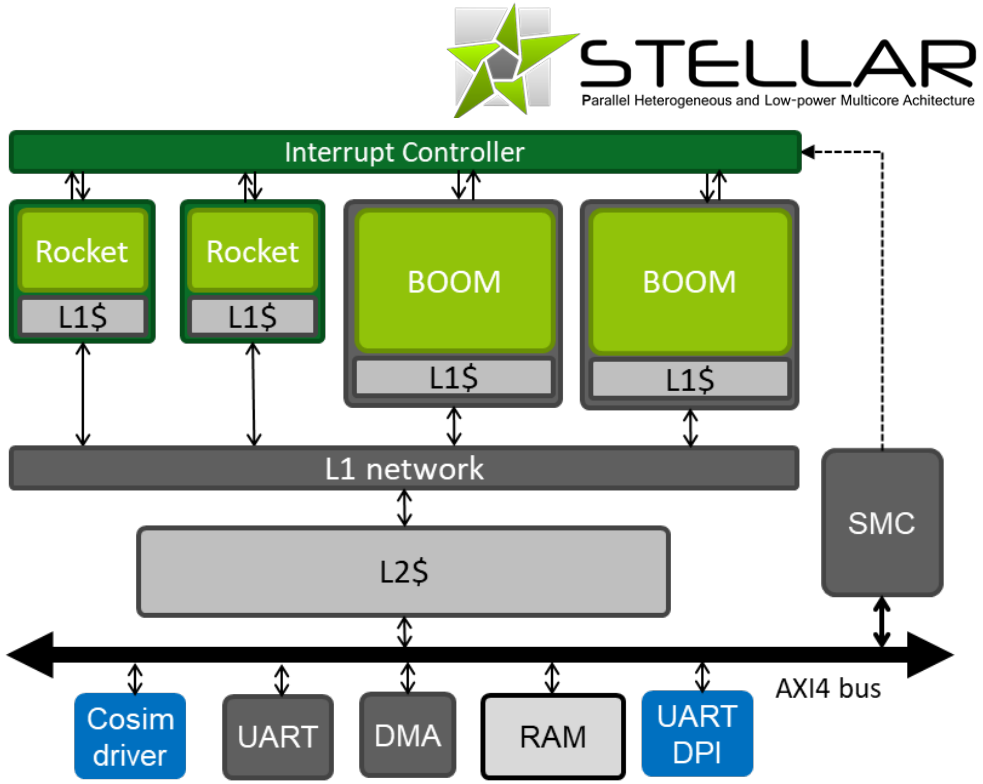
Modified model



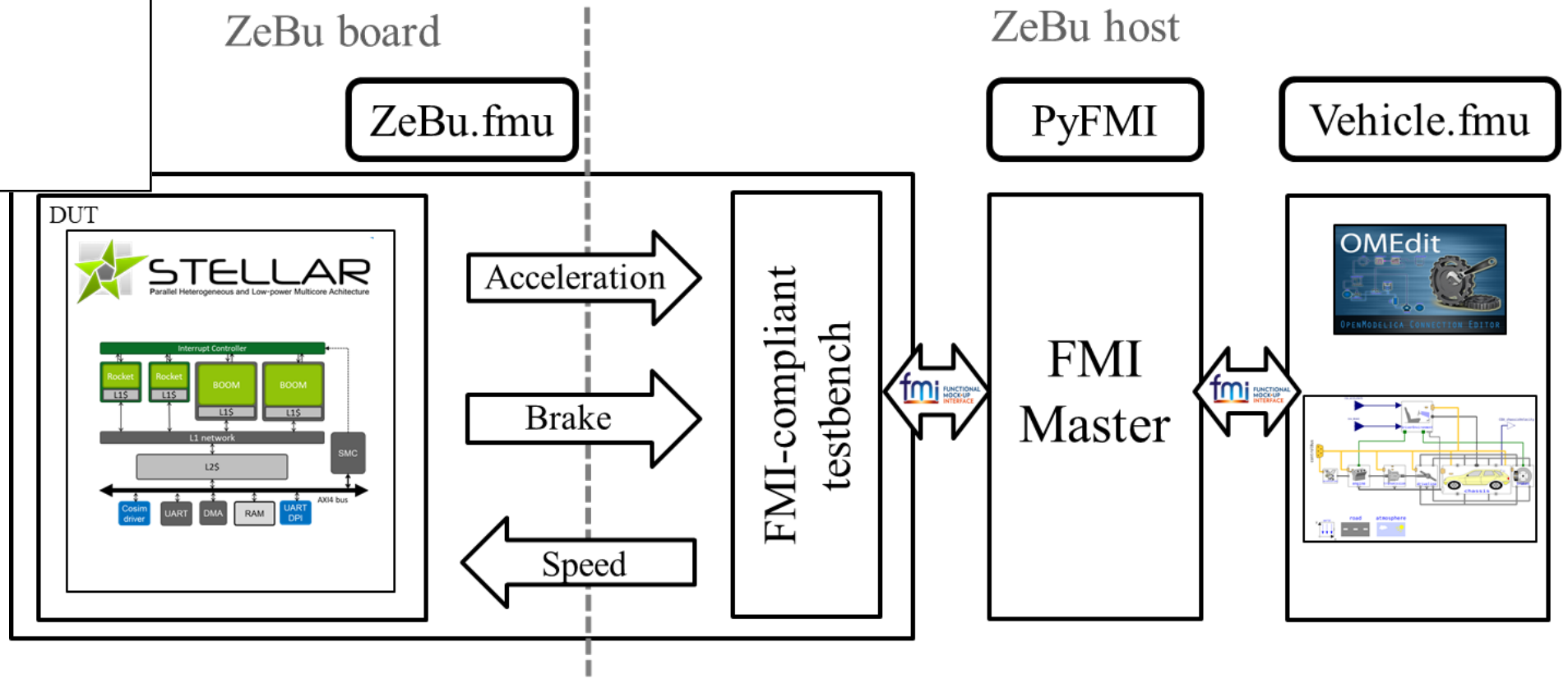
- Complexity of OpenModelica vehicle
  - 150 non-trivial equations and variables
  - Whole system consists of 10 main sub-systems
- Provides standard interface definitions for automotive subsystems and vehicle models.
- Designed to promote compatibility between the various automotive libraries and provide a flexible, powerful structure for vehicle modelling.

STELLAR: Parallel Heteregenous and Low-power Multicore Architecture

- A big.LITTLE like 64-bit heterogeneous multicore
  - Small cores
    - Rocket without FPU based
    - 8ko L1 caches
  - Big cores
    - Triple issue BOOM based
    - 32ko L1 caches
  - L1 cache coherency (MESI)
  - Instructions monitor (ROCC)
- Complete AMBA interconnection
  - Generated by Synopsys Core Assembler
  - AXI4 + AHB + APB network
  - I2C, UART and timers peripherals
- Main features
  - Smart monitoring
    - Performance, ageing, power consumption, BB zones
  - Heterogeneous management (FAMP and HW accelerators)
  - Semi-automatic MPSoC generation



Results



- Master simulation information: final simulation time = 240s and step size = 1s
  - The co-simulation duration is 5s
- Thanks to ZeBu, the hardware-based control accelerates the co-simulation (up to 100x compared to RTL simulation)

Conclusion

- Integration of ZeBu Server-3 emulator platform into a multi-physics automotive simulation environment through the use of FMI
- The proposed approach is based on creating a FMI to ZeBu adaptation functions
  - An automatic FMU generation flow is also proposed.
- Validation of the integration
  - Co-simulation between OpenModelica (modeling a vehicle) and ZeBu (implementing a RISC-V based multicore architecture) for vehicle speed control
  - The hardware-based control accelerates the co-simulation (up to 100x compared to RTL simulation)

Thank you

Any questions?