Politecnico di Torino

DUMAREY

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Integrating SystemC TLM into FMI 3.0 Co-Simulations with an Open-Source Approach

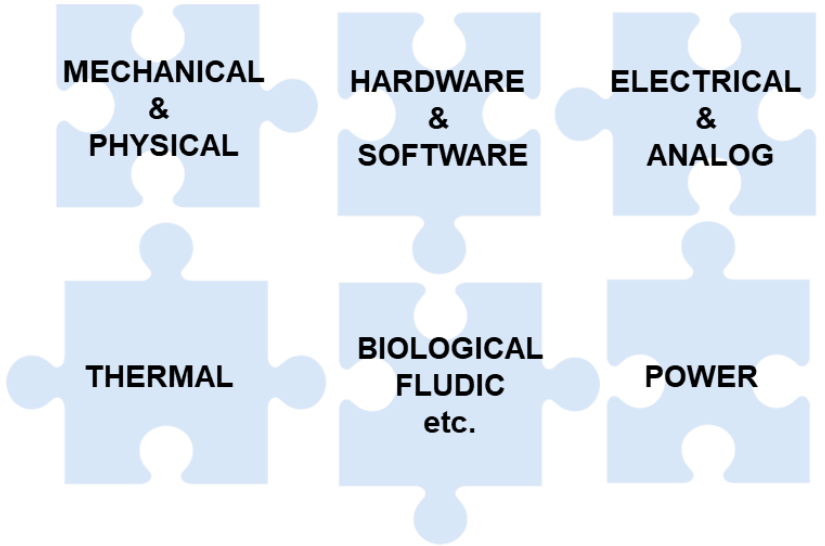
A. M. Albu, G. Pollo, A. Burrello, D. J. Pagliari, C. Tesconi, L. Panaro, D. Soldi, F. Autieri, **S. Vinco**

Politecnico di Torino – Dumarey Group

accellera SYSTEMS INITIATIVE

Motivation

- Growing complexity and heterogeneity of modern systems
  - Cosimulation as a winning strategy
    - ✘ Definition of custom ad-hoc integration strategies
    - ✓ Integration standards
- Functional Mockup Interface (FMI)
  - Widely used in control systems and system dynamic
  - Not fully supported in HW-SW co-design flows

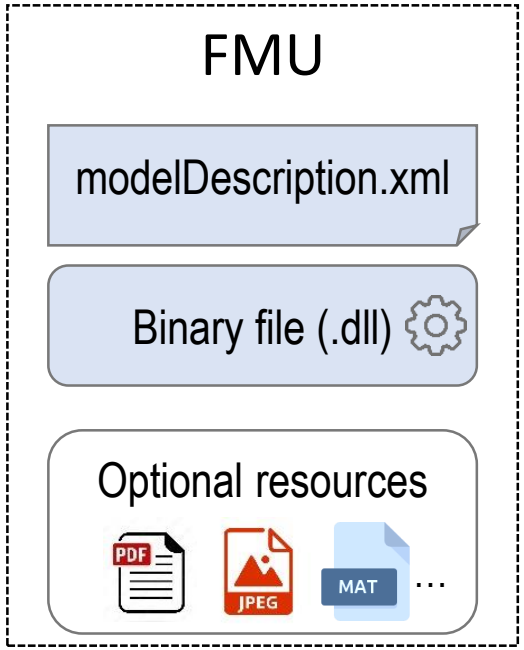


Goal: FMI + SystemC TLM

- SystemC TLM excellent for HW/SW
  - Limited interoperability with other domains
- FMI lacks SystemC TLM support
  - Some solutions proposed for RTL
- **Goal:** enable seamless integration of SystemC TLM into FMI-based co-simulation workflows
  - Fully open-source and standard-compliant
  - No modifications of the initial SystemC TLM code

FMI standard

- Developed to allow co-simulation with a standardized interface
- Encapsulation of models into self-contained components
  - Functional Mockup Units, FMUs
  - An XML-based model description file
  - Platform-specific binary files
  - Executable must comply with the defined FMI API



FMI standard

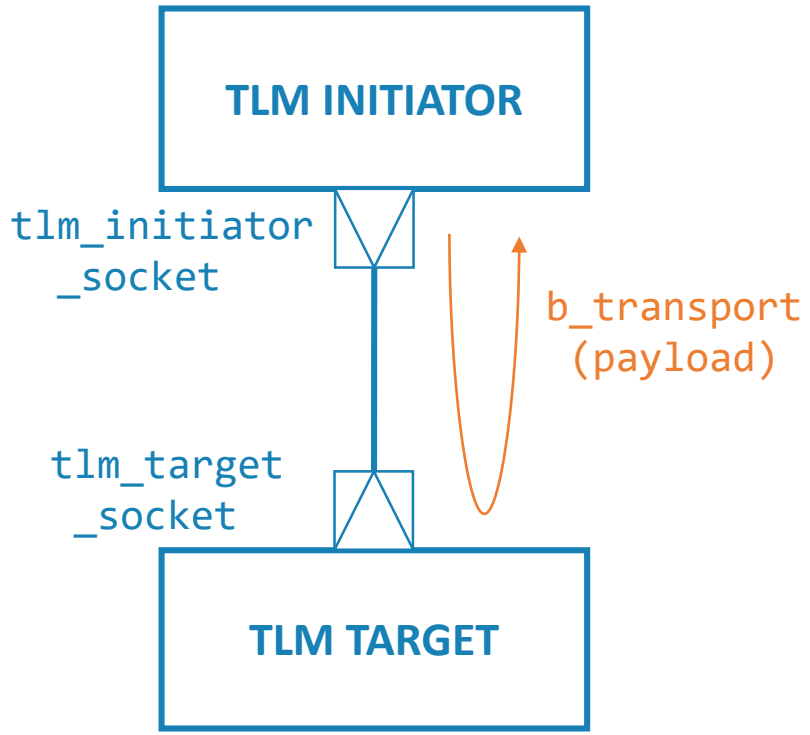
- Subset of API:
  - `fmi3Instantiate`
    - Creates a new instance of an FMU
  - `fmi3SetXXX` / `fmi3GetXXX`
    - Set/Get the value of a variable (XXX = type)
  - `fmi3DoStep`
    - Advance simulation by a specific time interval
  - `fmi3FreeInstance`
    - Release allocated resources

- Co-simulation algorithm:

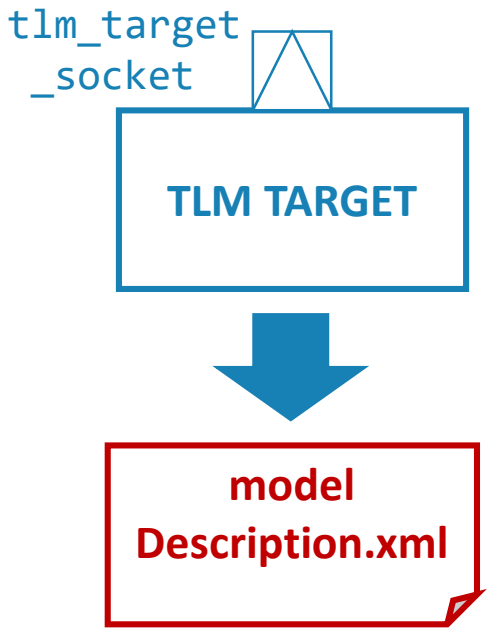
```
fmi3Instantiate
repeat many times{
    fmi3SetXXX
    fmi3DoStep
    fmi3GetXXX
}
fmi3FreeInstance
```

SystemC TLM

- Extension of SystemC standard
- Higher-level transaction-based communication
  - Interface supporting blocking and non-blocking communication
  - Focus on blocking
    - Initiator directly calls the target `b_transport` method
    - Processes the request immediately



A. DESIGN ANALYSIS



Design Analysis

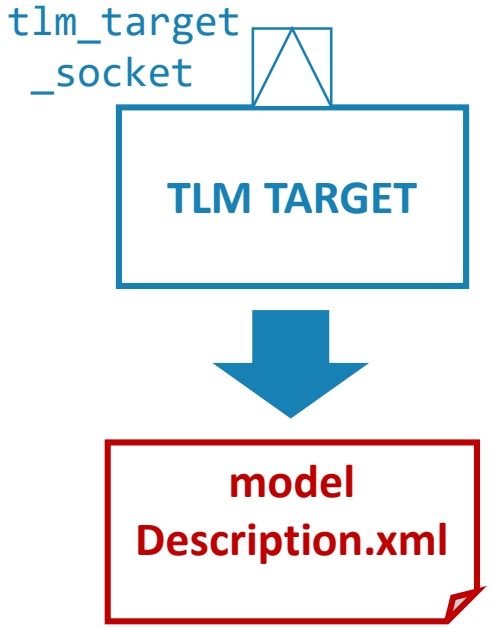
- Scan data structure uploaded to payload
  - Detect which fields are read/written
  - Map type onto FMI types
- Generate `modelDescription.xml` file

```
1   <fmiModelDescription fmiVersion="3.0" modelName="tlm"/>
2     ...
3     <ModelVariables>
4       <Int32 name="fmi_data_in" valueReference="1"
5            causality="input" start="0"/>
6       <Int32 name="fmi_result" valueReference="2"
7            causality="output"/>
8     </ModelVariables>
9     ...
10  </fmiModelDescription>
```

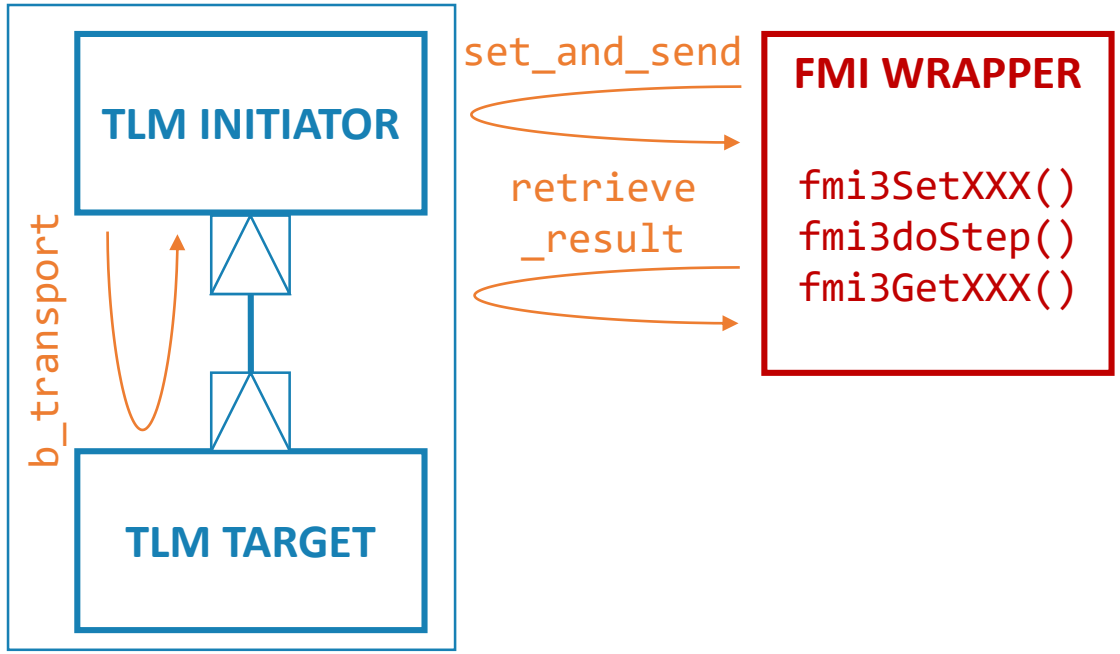
modelDescription.xml

```
1   struct payload {
2      sc_dt::sc_int<32> data_in;
3      sc_dt::sc_int<32> data_out; };
```

A. DESIGN ANALYSIS

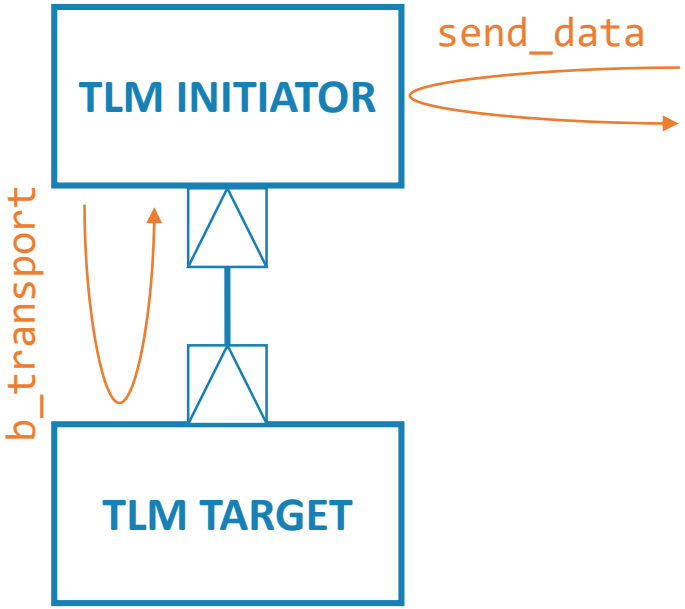


B. FMI WRAPPER GENERATION



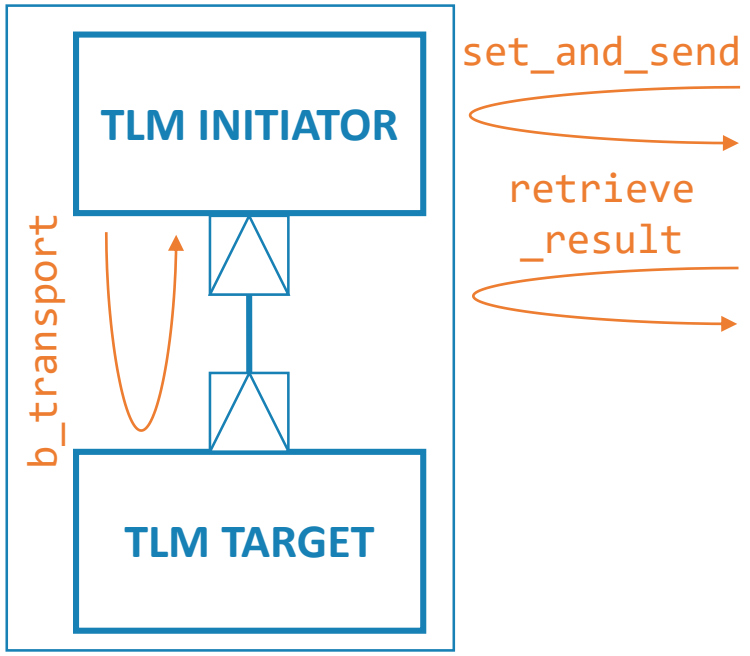
TLM Subsystem Generation

- TLM initiator
  - `tlm_initiator_socket` for handling communication with TLM target
  - Manages communication with the target
    - `sending_thread` process
      - Repeatedly sets the payload fields
      - Invokes the `b_transport` primitive of the target
    - `send_data` function
      - Public, can be invoked by top level
      - Wakes up the `sending_thread` process



TLM Subsystem Generation

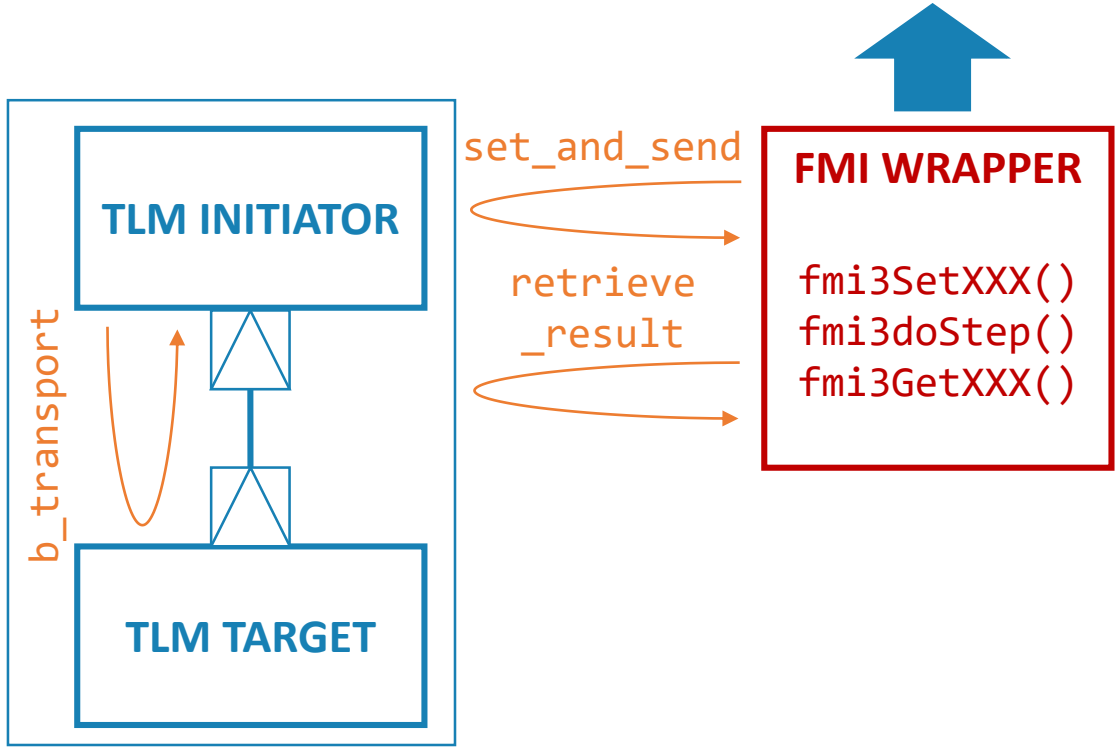
- TLM top level
  - Instantiates and binds initiator and target
  - Makes TLM data updatable and visible + allows TLM activation
    - `set_and_send` function
      - Receives as parameter data to be transferred to the target
      - Copies it to the initiator
      - Invokes the send_data function to start communication
    - `retrieve_result` function
      - Collects target output once the transaction has completed



Wrapper generation

- FMI wrapper
  - Bridge between FMI and TLM
    - Coordinate execution
    - Manage data exchange
  - Declares:
    - Pointer to the TLM top level
    - FMI interface variables defined in the XML model description
    - `sc_time` variable for simulation time

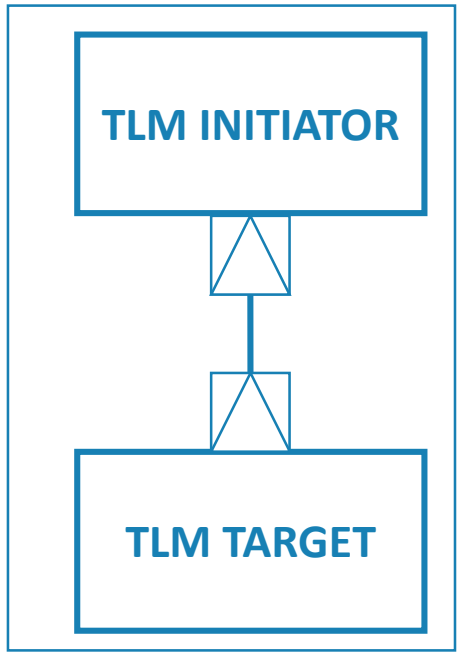
```
1  struct WRAPPER_STRUCT {
2     Top *top; // Pointer top-level module
3     sc_time current_time; // Current simulation time
4     fmi3Int32 fmi_data_in; // Input data from FMI domain
5     fmi3Int32 fmi_result; // Output data to FMI domain
6  };
```

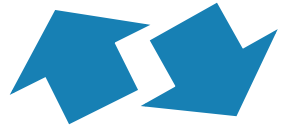


FMI API implementation

- Instantiation and initialization functions
  - Declare and instantiate the top level entity
  - Issue a `sc_start(SC_ZERO_TIME)` to construct SystemC TLM objects
- Setter and getter functions
  - Copy or retrieve values of FMI variables to local variables

```
1  struct WRAPPER_STRUCT {
2     Top *top; // Pointer top-level module
3     sc_time current_time; // Current simulation time
4     fmi3Int32 fmi_data_in; // Input data from FMI domain
5     fmi3Int32 fmi_result; // Output data to FMI domain
6  };
```





FMI WRAPPER

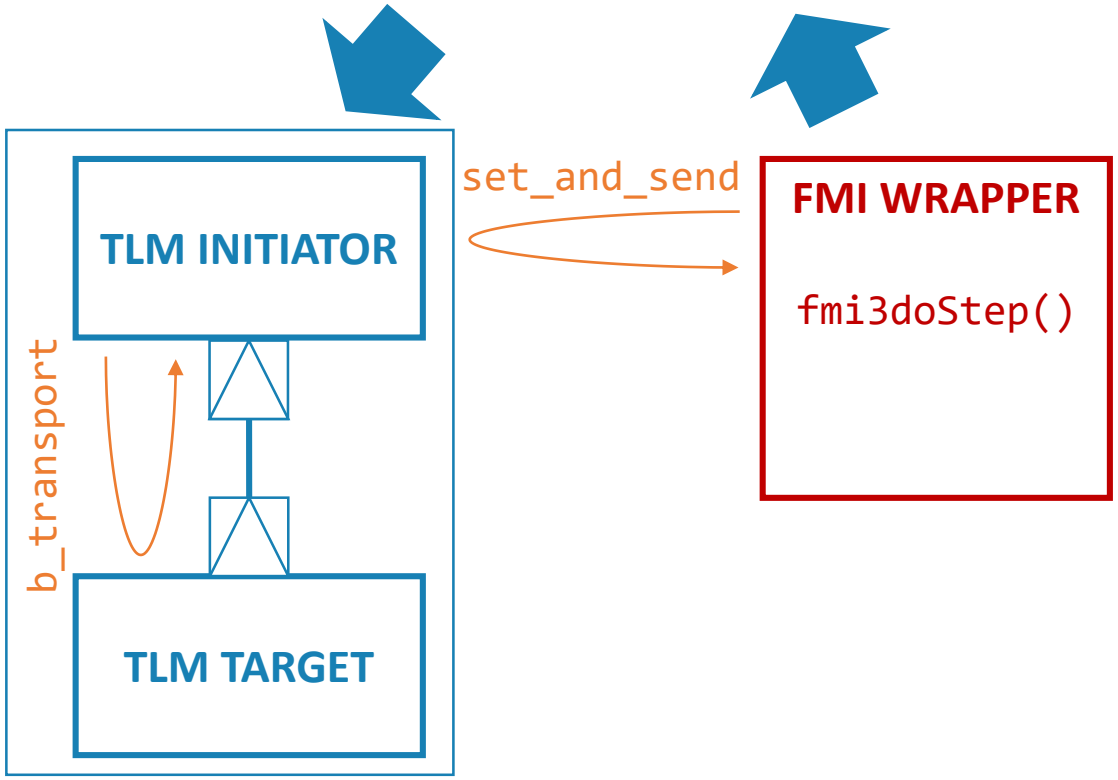
`fmi3SetXXX()`
`fmi3GetXXX()`

FMI API implementation

- Instantiation and initialization functions
  - Declare and instantiate the top level entity
  - Issue a `sc_start(SC_ZERO_TIME)` to construct SystemC TLM objects
- Setter and getter functions
  - Copy or retrieve values of FMI variables to local variables
- `fmi3doStep` function
  - Advance simulation time

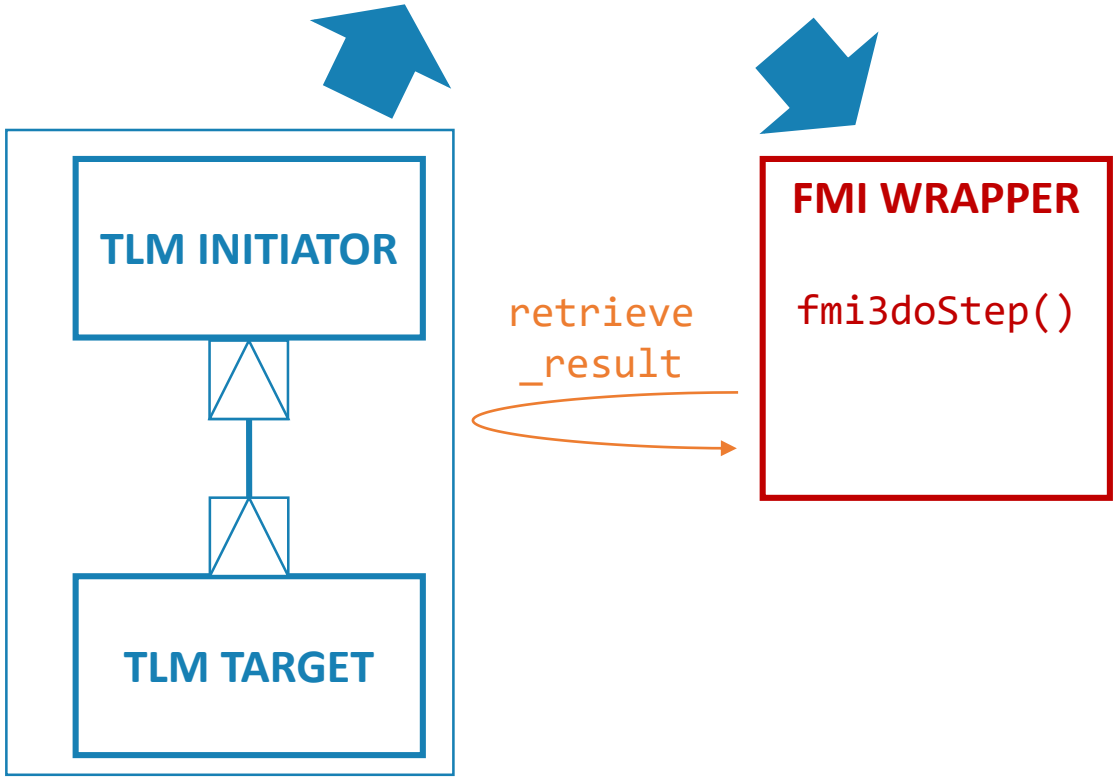
```
1   fmi3DoStep(fmi3Instance instance,
2           fmi3Float64 currentCommunicationPoint,
3           fmi3Float64 communicationStepSize, ...
4           fmi3Boolean* earlyReturn,
5           fmi3Float64* lastSuccessfulTime) {
6
7      WRAPPER_STRUCT* fmu = static_cast<WRAPPER_STRUCT*>(
           ↪ instance);
8      sc_time step_size(communicationStepSize, SC_SEC);
9
10     fmu->top->set_and_send(static_cast<int32_t>(fmu->
           ↪ fmi_data_in));
11     sc_start(step_size);
12
13     int32_t result;
14     fmu->top->retrieve_result(result);
15     fmu->fmi_result = result;
16
17     sc_time next_time;
18     if (!(*earlyReturn))
19       next_time = fmu->current_time + step_size;
20     else next_time = fmu->current_time + (*
           ↪ lastSuccessfulTime);
21     fmu->current_time = next_time;
22
23     return fmi3OK;
24  }
```

```cpp
struct WRAPPER_STRUCT {
   Top *top; // Pointer top-level module
   sc_time current_time; // Current simulation time
   fmi3Int32 fmi_data_in; // Input data from FMI domain
   fmi3Int32 fmi_result; // Output data to FMI domain
};
```



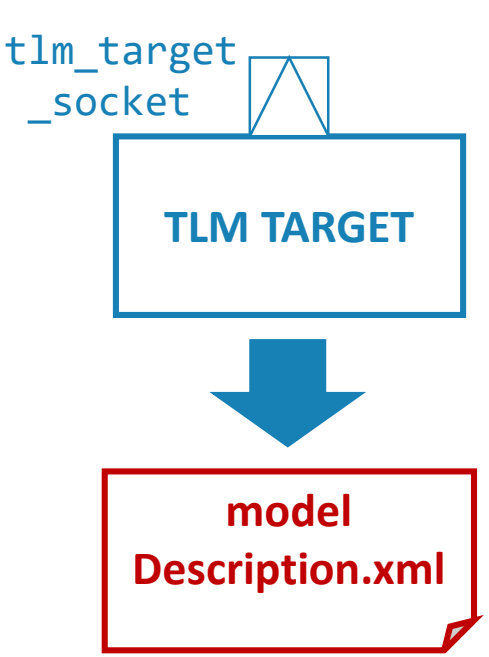

```cpp
fmi3DoStep(fmi3Instance instance,
           fmi3Float64 currentCommunicationPoint,
           fmi3Float64 communicationStepSize, ...
           fmi3Boolean* earlyReturn,
           fmi3Float64* lastSuccessfulTime) {

   WRAPPER_STRUCT* fmu = static_cast<WRAPPER_STRUCT*>(
        ↪ instance);
   sc_time step_size(communicationStepSize, SC_SEC);

   fmu->top->set_and_send(static_cast<int32_t>(fmu->
        ↪ fmi_data_in));
   sc_start(step_size);

   int32_t result;
   fmu->top->retrieve_result(result);
   fmu->fmi_result = result;

   sc_time next_time;
   if (!(*earlyReturn))
     next_time = fmu->current_time + step_size;
   else next_time = fmu->current_time + (*
        ↪ lastSuccessfulTime);
   fmu->current_time = next_time;

   return fmi3OK;
}
```

```
1  struct WRAPPER_STRUCT {
2     Top *top; // Pointer top-level module
3     sc_time current_time; // Current simulation time
4     fmi3Int32 fmi_data_in; // Input data from FMI domain
5     fmi3Int32 fmi_result; // Output data to FMI domain
6  };
```

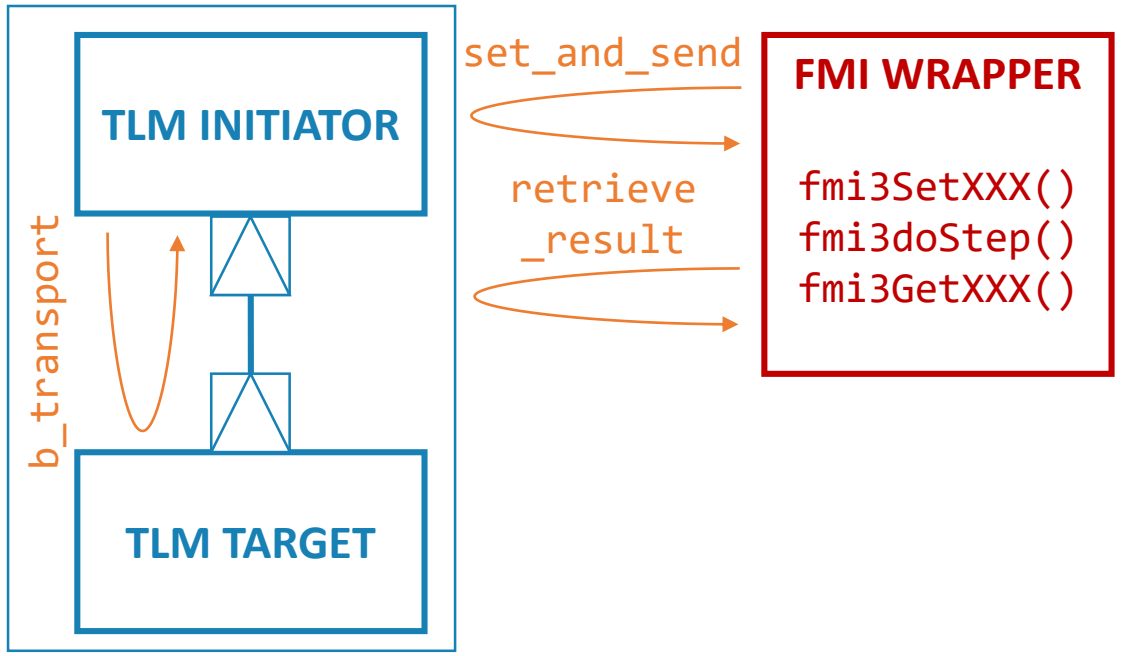



```
1  fmi3DoStep(fmi3Instance instance,
2          fmi3Float64 currentCommunicationPoint,
3          fmi3Float64 communicationStepSize, ...
4          fmi3Boolean* earlyReturn,
5          fmi3Float64* lastSuccessfulTime) {
6
7     WRAPPER_STRUCT* fmu = static_cast<WRAPPER_STRUCT*>(
          ↪ instance);
8     sc_time step_size(communicationStepSize, SC_SEC);
9
10    fmu->top->set_and_send(static_cast<int32_t>(fmu->
          ↪ fmi_data_in));
11    sc_start(step_size);
12
13    int32_t result;
14    fmu->top->retrieve_result(result);
15    fmu->fmi_result = result;
16
17    sc_time next_time;
18    if (!(*earlyReturn))
19      next_time = fmu->current_time + step_size;
20    else next_time = fmu->current_time + (*
          ↪ lastSuccessfulTime);
21    fmu->current_time = next_time;
22
23    return fmi3OK;
24 }
```

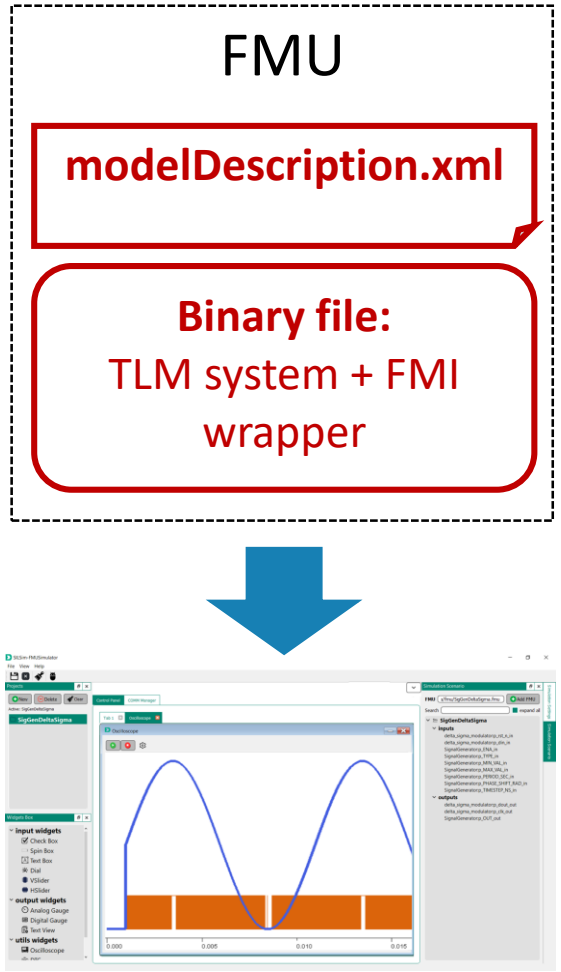
A. DESIGN ANALYSIS



B. FMI WRAPPER GENERATION



C. SIMULATION



A. DESIGN ANALYSIS

B. FMI WRAPPER GENERATION

C. SIMULATION

tlm_target
_socket

TLM TARGET

model
Description.xml

AUTOMATED WITH PYTHON TOOL + JSON CONFIG FILE

TLM INITIATOR

b_transpo

TLM TARGET

set_and_send

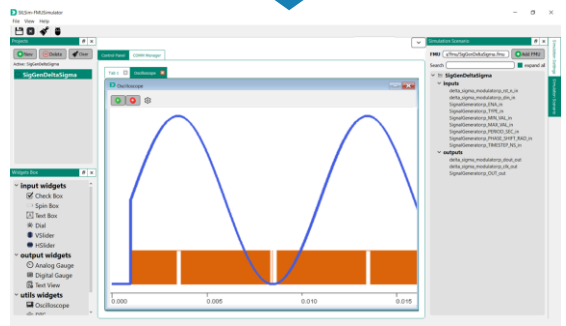
FMI WRAPPER

fmi3GetXXX()

FMU

modelDescription.xml

Binary file:
TLM system + FMI wrapper



I2C & ECC FMU generation

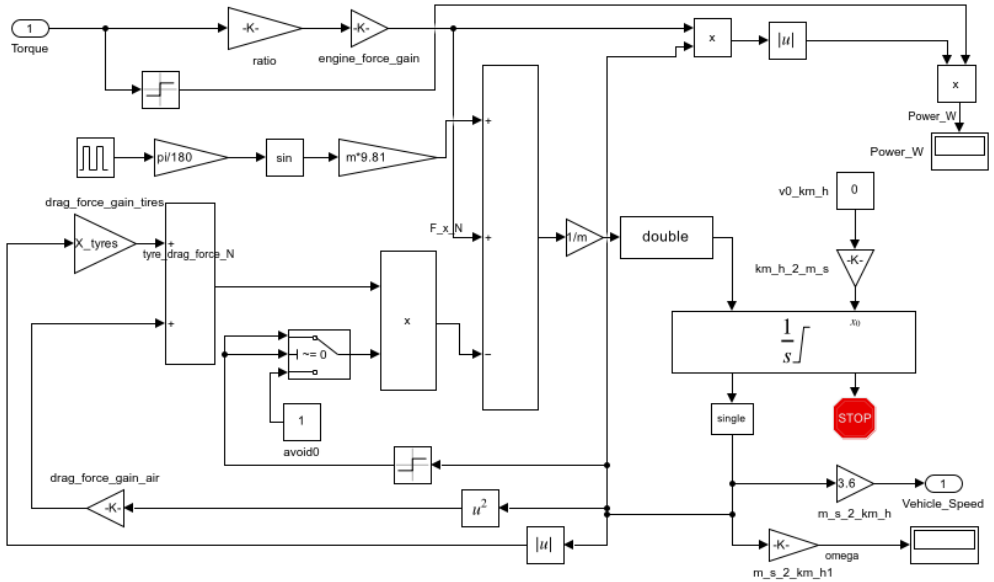
- 2 SystemC TLM designs
  - Simulation time increases with number of FMI interactions
    - And thus length of simulated time
    - Computational and communication overhead
    - Allowed to cosimulate the wrapped TLM DUMAREY Software-in-the-Loop (SIL) environment
    - Stable memory overhead

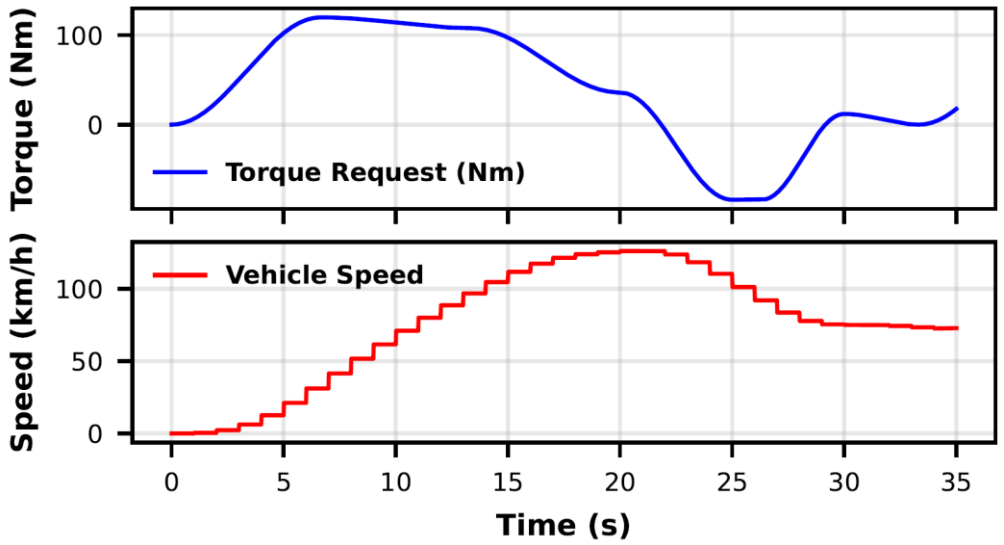
DESIGN	I2C	ECC
MODULES	4	1
PROCESSES	5	3
LINES OF CODE	1,072	1,311
PAYLOAD FIELDS	7	10

		doStep (#)		
DESIGN	TARGET	250	1,000	10,000
I2C	Time	1.14x	2.20x	4.17x
	Memory	22.82x	22.41x	21.45x
ECC	Time	1.02x	6.36x	10.11x
	Memory	14.52x	15.01x	12.83x

Cosimulating with FMUs

- Single-pedal EV implemented in **Simulink**
  - Exported as an FMU using the CATIA FMI-Kit
  - Implemented the Electronic Control Unit (ECU) in SystemC TLM
- Integrated through a FMI master
  - Vehicle speed response shows appropriate dynamic behavior
  - Generated FMUs are **fully compliant** with the FMI standard





Concluding remarks

- **Open-source automated framework** for integrating SystemC TLM models with the FMI standard
  - Automatic generation of FMU and **non-intrusive integration**
  - No modifications to the SystemC TLM code
- **Experimental results** handled different designs efficiently
- Future work will focus on extending support for
  - Additional SystemC TLM features + **mixed RTL and TLM**
  - Integration with Instruction Set Simulator (ISS) based environments

Thank you!

Politecnico di Torino

DUMAREY

accellera SYSTEMS INITIATIVE