Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

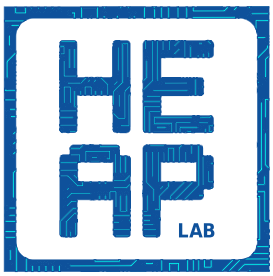
HEAPLab Project Report

Generation of a Matlab FMI Model

Author:
Andrea Amer

Supervisor:
Dr. Federico Terraneo

August 30, 2020



Abstract

Purpose of the presented work is to understand how to generate a FMI (https://fmi-standard.org/) compliant model using Mathworks Simulink and interface this model in a C++ written program using fmi-interface-cpp.

1 Introduction

The Functional Mock-up Interface (FMI) is a free standard that defines a container and an interface to exchange dynamic models using a combination of XML files, binaries and C code zipped into a single file (https://fmi-standard.org/).
Many tools can be used to generate this kind container, in this work two tools will be used: OpenModelica and Simulink.
Since a sample FMI Model generated used Modelica is already available (included in fmi-interface-cpp repository) and this model has been previously tested and loaded using fmi-interface-cpp, this one will be used as reference for validation purposes. An equivalent model will be developed using Simulink (at least MATLAB R2017b is required) and exported as FMU, finally this new developed model will be imported in a C++ program using fmi-interface-cpp and simulation results between this and the reference will be compared.
To summarize, the objectives are:

- Develop an equivalent Model (in respect to the reference Modelica model) using Simulink,
- Generate a FMU from this model using Simulink generation tool,
- Load this model in a C++ written program using fmi-interface-cpp library, to fulfill this step a modified version of the test program will be used,
- Compare the results obtained with the Simulink Model and the Modelica model.

1.1 Reference Model

The reference Model is a simple RC series electrical model developed using OpenModelica:

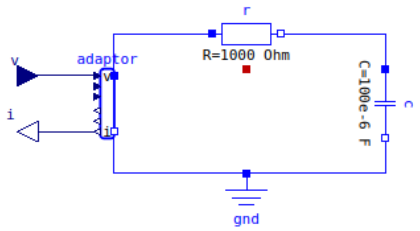


Figure 1: Reference RC Series Model.

The parameters used for the model are:
R = 1000 ohm
C = 100 uF
Input of the model is voltage v(t) and output is the current i(t) flowing in the circuit.

1.2 fmi-interface-cpp

This library written by Dr. Terraneo can be used to load one or more generic FMI compliant models in a C++ written program. Complete source files can be found at (https://github.com/fedetft/fmi-interface-cpp).

2 Design and Implementation

2.1 Equivalent Model

As a first step a model equivalent to the reference one. Some RC series sample model is already available in Matlab environment (ssc_rc_circuit_sl) so this one will be used as starting point for developing the equivalent model. After some modifications the final model is reported in Figure 2.

As the reference model, the input is the voltage $v_i(t)$ and the output is the current flowing $i(t)$ in the circuit.

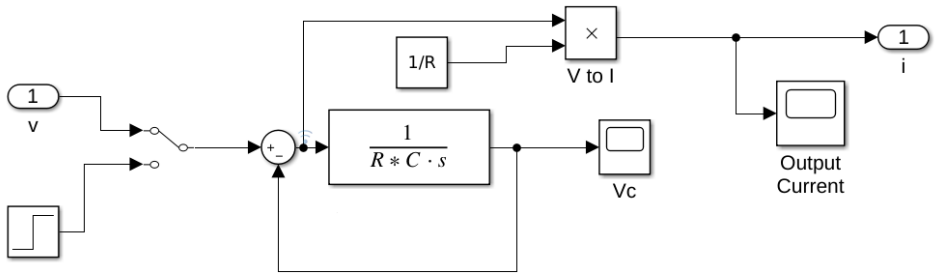


Figure 2: Equivalent RC Series Model.

Some remarks: the step block before the switch is normally disconnected and has been used for testing purposes only.
The voltage drop on the capacitor $v_c(t)$ is computed as follows:

$$v_c(t) = v_i(t)(1 - e^{-t/RC})$$

Which in Laplace domain becomes:

$$V_c(s) = V_i(s)\frac{1}{1 + sRC}$$

In the model this is implemented as:

$$V_c(s) = (V_i(s) - V_c(s))\frac{1}{sRC}$$

After some simple algebraic manipulation:

$$V_c(s) = \frac{V_i(s)}{sRC}\frac{sRC}{1 + sRC}$$

Simplyfing the quantity sRC we get the first expression.
The summing node computes the quantity

$$v_r(t) = v_i(t) - v_c(t)$$

To obtain the current flowing we can apply Ohm law as:

$$i(t) = \frac{v_r(t)}{R}$$

Substituing all the previously obtained relation we obtain in time domain:

$$i(t) = \frac{v_i(t)e^{-t/RC}}{R}$$

In Figure 3 the model ouput is reported. The output is obtained by injecting a Step input signal of 1 V a time t = 1s. As expected at t = 1s the current peaks and is equal to 1 mA since for very high frequency the capacitor can be modeled as a short circuit ($I = V/R$).
As t increases the impedence of the capacitor increases and the current exponentially decreases and becomes zero with a time constant $t = RC$ as the capacitor can be assumed to be an open circuit for very low frequency.

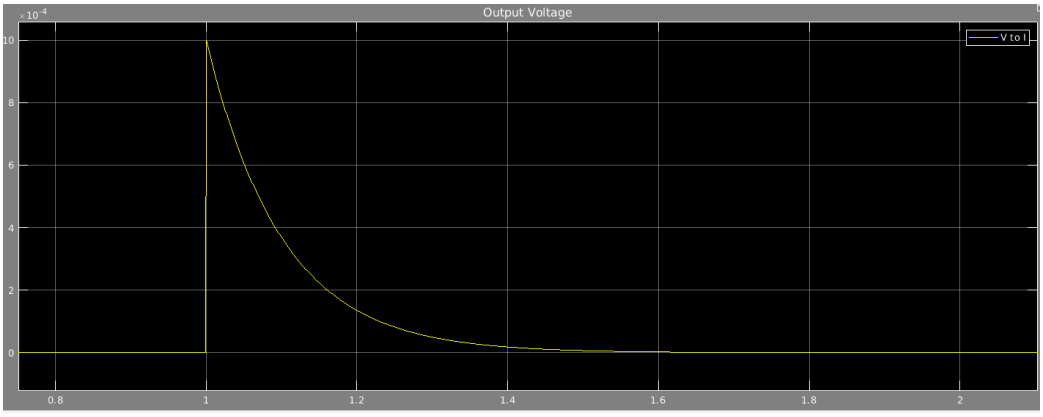
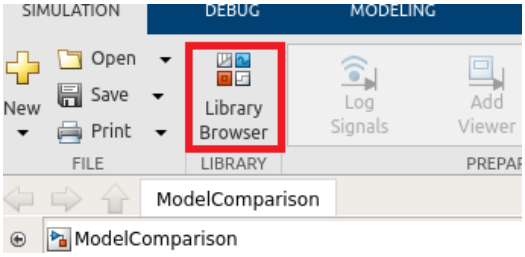
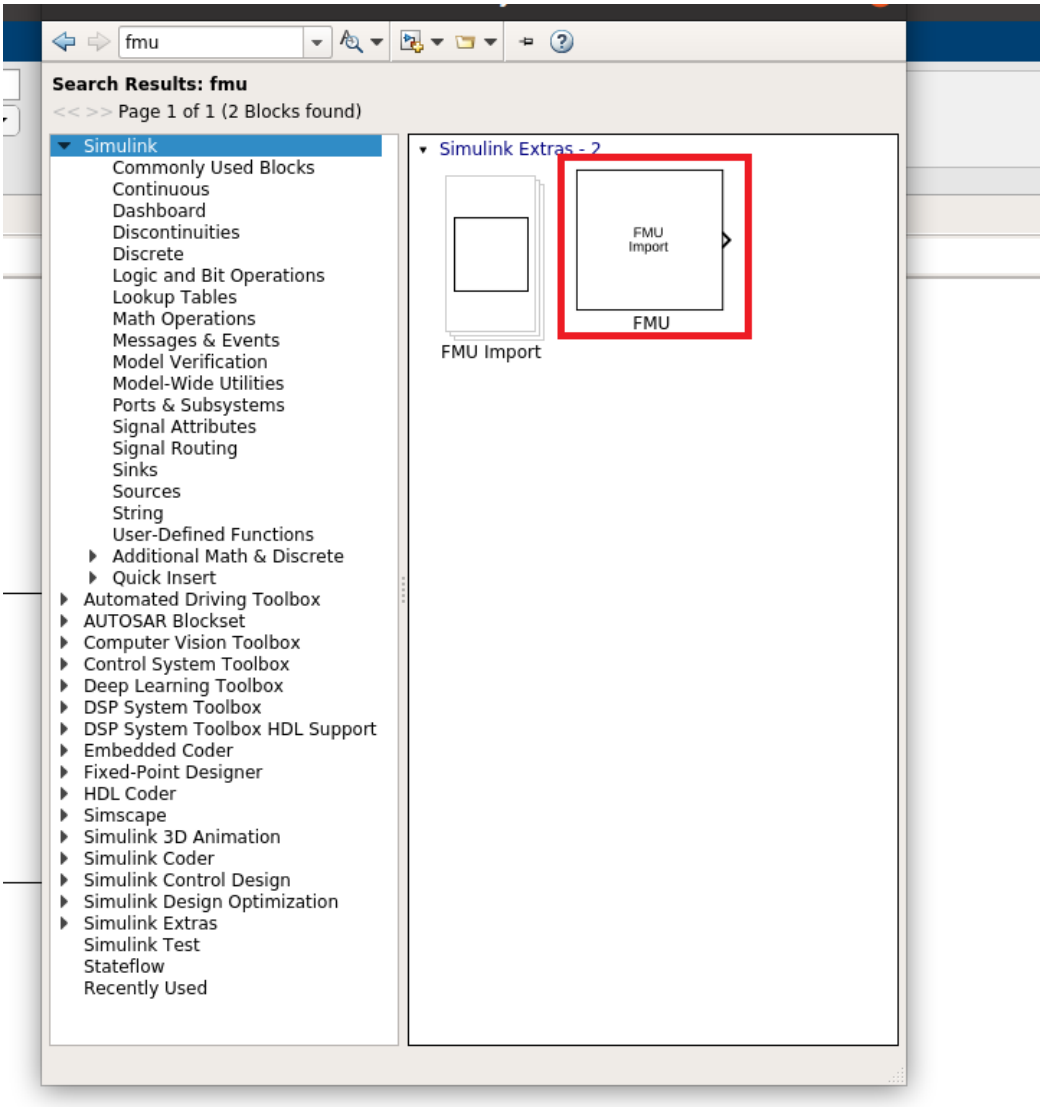


Figure 3: Simulink Model Output.

To ensure the two models (Modelica and Simulink) equivalence, a further validation step is performed exploiting Simulink capabilities to import an FMI compliant model. The two model shall be imported in a project then the same voltage signal will be injected and the two output currents will be then compared.
To import an FMU select Simulink Library browser:

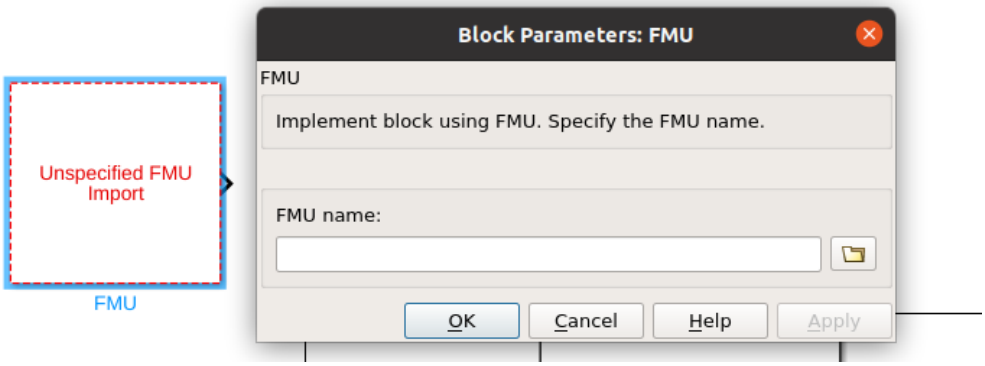
In the Library browser look for the FMU import block





After adding the module into the workspace simply insert the path of .fmu

archive that you need to import.
NOTE: since OpenModelica does not output a .fmu, first add into a compresed archive to generated Model(.xml and .so are needed) then change to extension to .fmu.



The two models are then imported in the workspace as shown in figure 4

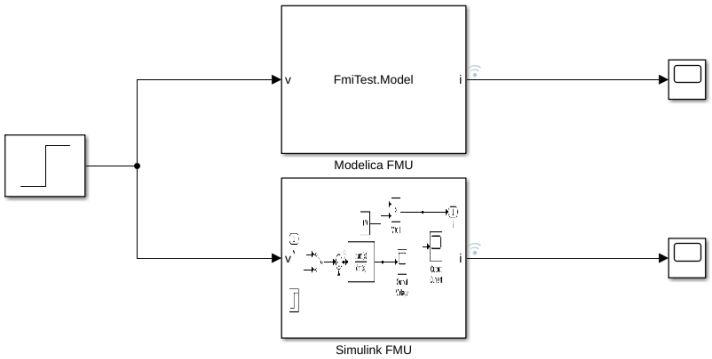


Figure 4: Model Comparison.

By injecting the same voltage signal no notable difference can be observed on the two outputs as reported in figure 4.

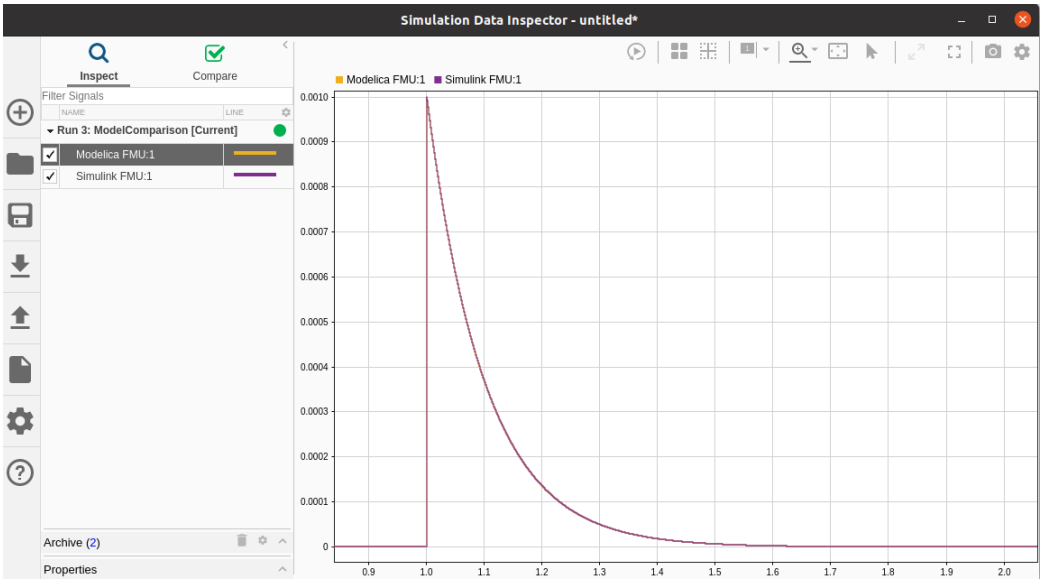
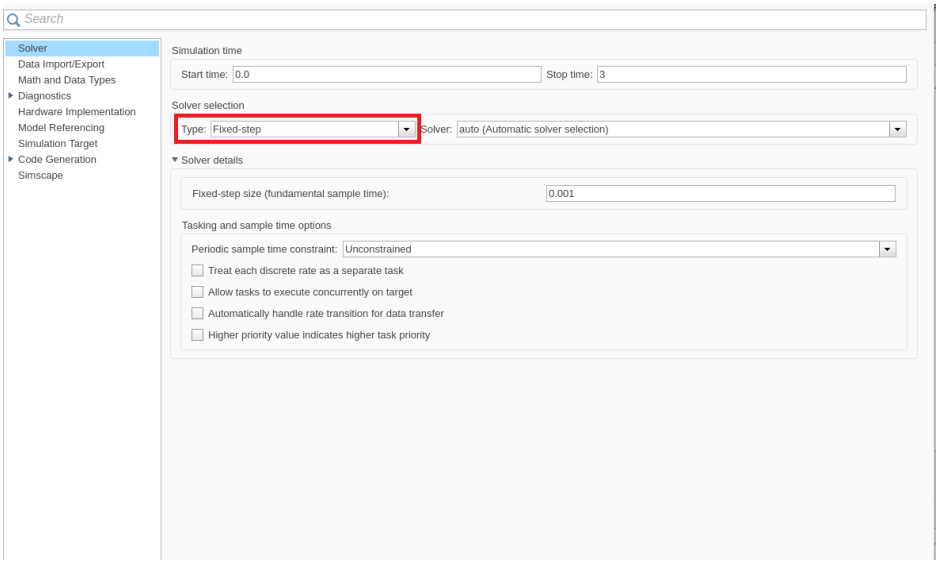


Figure 5: Model Comparison Output.

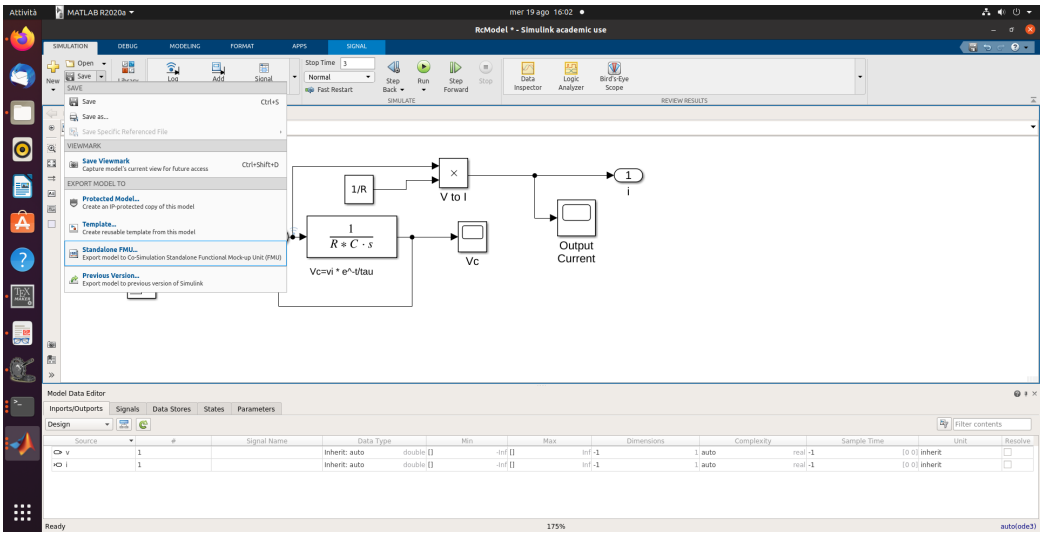
2.2 FMU Generation using Simulink

To generate a standalone FMU from the developed model the following steps needs to be performed. First the solver type must be set to "Fixed Step".

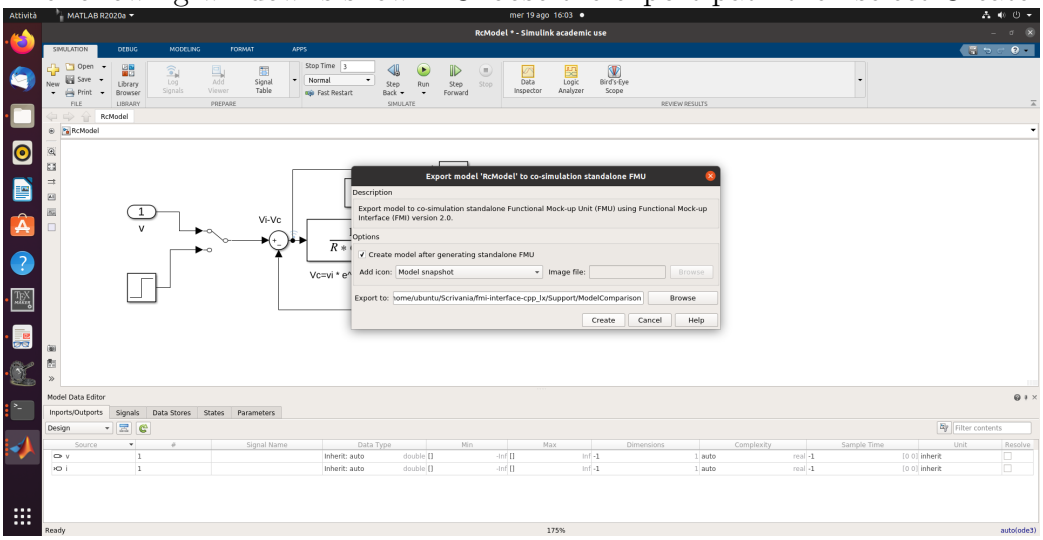


NOTE: When the model is run as FMU in a C++ program, is not possible

to use a step size which is lower then the one selected as Step size in the Simulink Model, so care must be taken regarding the selection of this parameter.
The next step is to generate a FMU standalone using Simulink exporting tool from Simulation → Save → Standalone FMU:



The following window is shown. Choose the export path then select Create.



If no error is detected in the generation phase, the ouput of this export operation shall be a .fmu compressed archive that contains the .xml descriptor and .so library of the model (optionally if selected in the export phase a

.png report a model graphical snapshot)

In this case we get a RcModel.fmu and RcModel.so in the binaries folder (the library is generated with the same name as .fmu).
The last step is to decompress the .fmu archive.

2.3 Model load using fmi-interface-cpp

To load this new obtained model and compare the simulation results with the Modelica model, the start point will be the program fmitest.cpp (available in fmi-interface-cpp project repository). This program loads the sample Modelica RC model and run a simulation of duration 1 s with a step of 10 ms and at t = 100 ms injects a step voltage of 1 V. At every simulation step n the input $v_i(n)$ and the output $i(n)$ is printed on the console.
First a new object of class FmiInterface initialized with RcModel is declared.

```
// Load the Simulink Rc Model
FmiInterface fmiSimulink("RcModel","../",LogLevel::Normal);
```

Then the code used for the Modelica model is simply replicated for the object fmiSimulink (printVariables, getting v and i index and startSimulation)

```
fmiSimulink.printVariables();
auto vSimulinkIndex=fmiSimulink.variableIndex("v");
auto iSimulinkIndex=fmiSimulink.variableIndex("i");
fmiSimulink.startSimulation();
```

After this initial startup the Input is set using the same variable as Modelica model and the step is performed using the same time variables.

```
fmiSimulink.setScalarDouble(vSimulinkIndex,v);
fmi.setScalarDouble(vIndex,v);

fmiSimulink.doStep(time,step);
fmi.doStep(time,step);
```

An additional modification to allow simple csv log of the system variable for post elaboration and verification has been added to fmitest.cpp.

3 Experimental Results

Below the a Matlab generated plot obtained from .csv is reported: The first

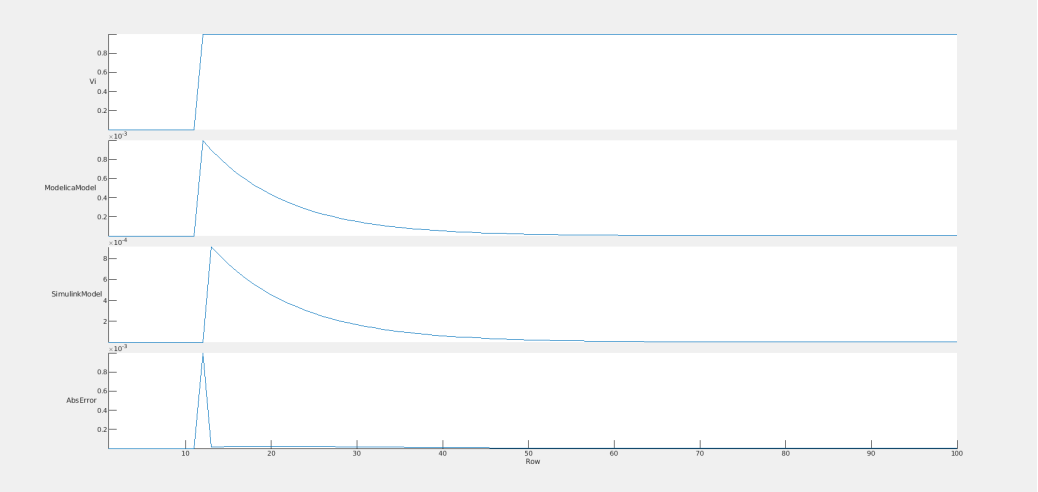
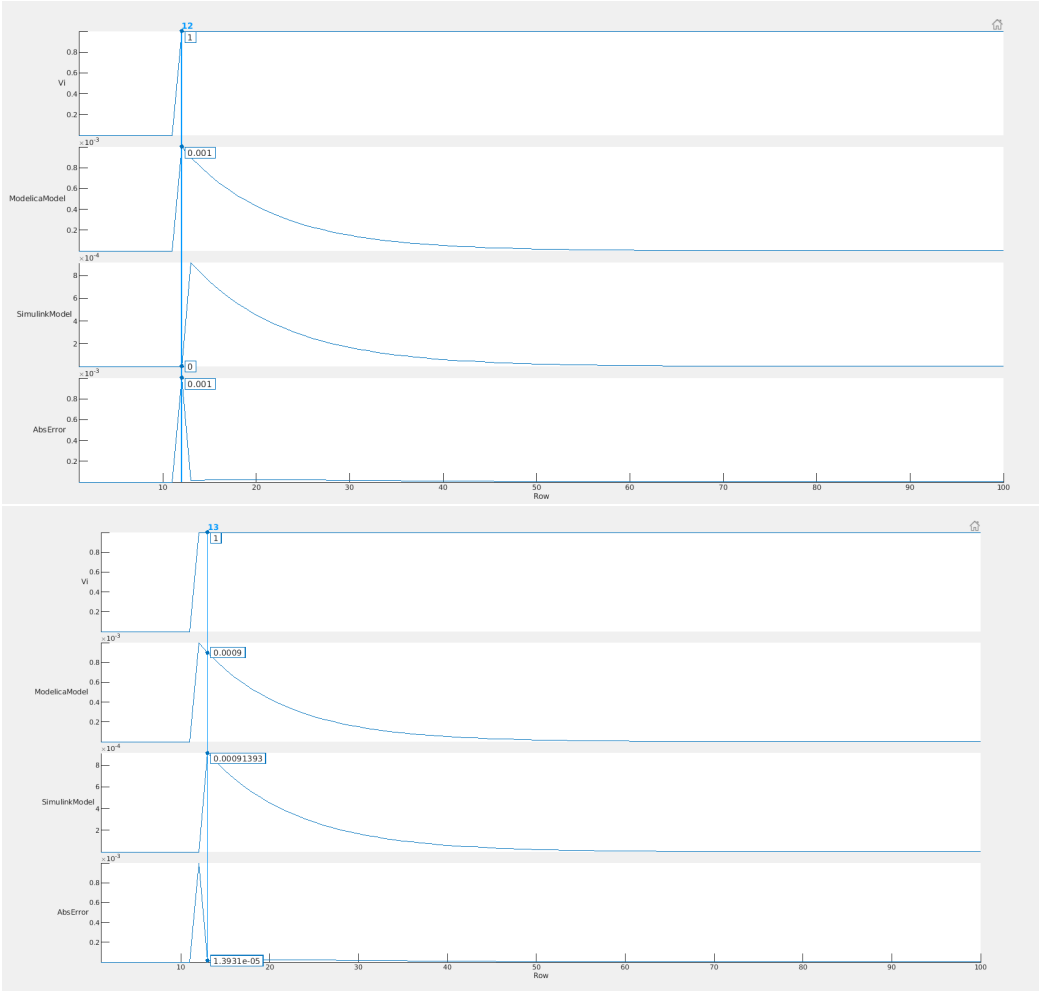


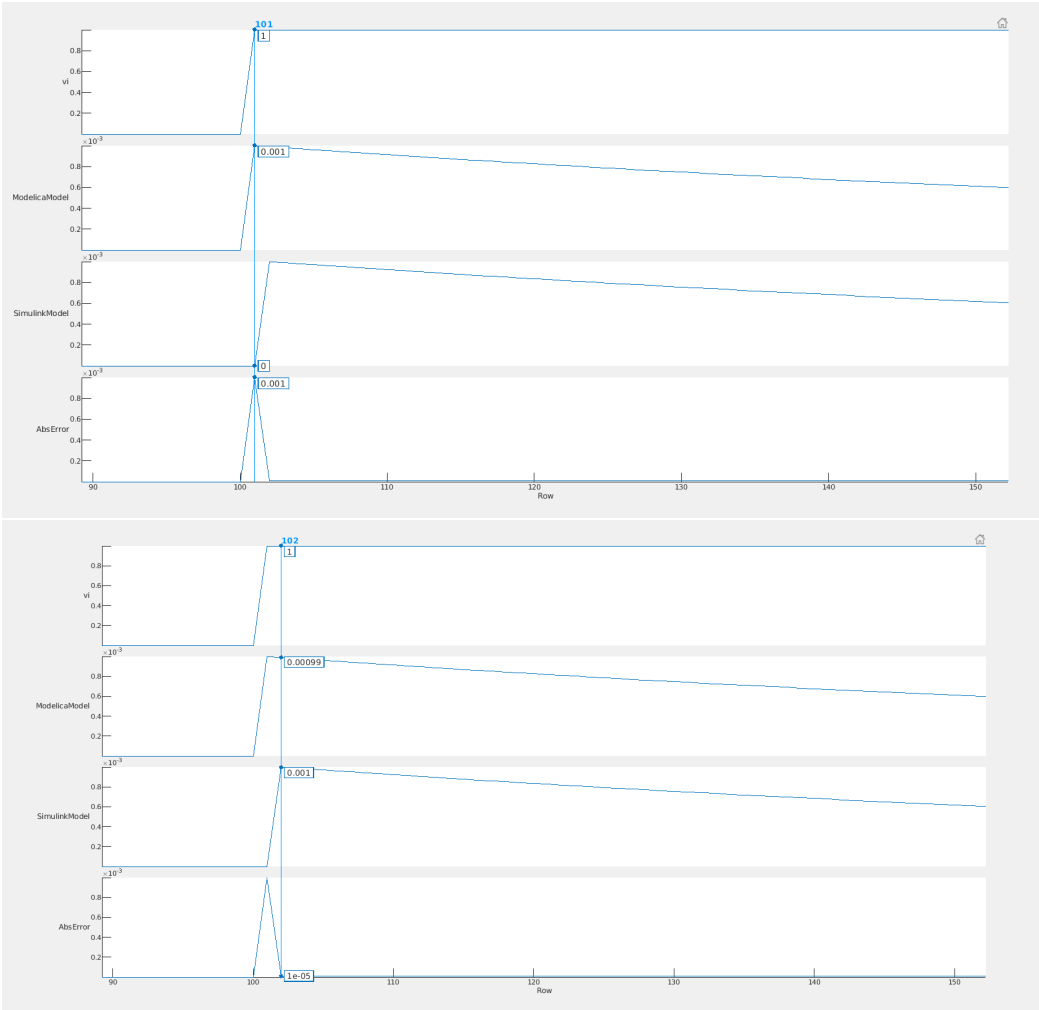
Figure 6: Experimental results.

plot is the input $v_i(n)$, the second and the third plots reports $i(n)$ computed with the reference Modelica Model and the equivalent Simulink Model. The last plot reports the absolute error between the computed $i(n)$ of the two models.
As can be seen from the asolute error plot, the two model behaviour does mostly agree excepts for the simulation step when voltage step is injected. This behaviour is probably due to some difference in the generated Simulink FMU.

Apparently the algebraic part of the system is better managed by the Modelica generated model since when the input voltage step is injected, the current in the circuit should immediatly change, this correctly happens in the reference model, while in the Simulink model it is required that at least a simulation step is completed.
From the plots, we can see that at n=12, the step voltage is injected and the reference model reacts immediatly. The Simulink model reacts at n=13 but the delay is not a simulation step (10 ms), but the chosen fixed step size of the model (1 ms), this explains why the Simulink plotted output does not reach the 1 mA value as expected but nearly 0.9 mA



This conclusion can verified by changing the simulation step in fmitest.cpp from 10 ms to 1 ms. Using this simulation step, we can se that at n = 101



the voltage step is injected, an the behaviour is the same as the previous case. At n = 102 the Simulink model reacts to the input this time reaching the current of 1 mA. After this step, the Simulink model is always beyond one simulation step in respect to the reference model.

4 Conclusions

The generation of a standalone FMU from Simulink has been carried out. The package has been then succesfully imported in a C++ program using the library fmi-interface-cpp. Some discrepancy in the generated model in respect to the reference model has been noted, but this is most probably due to Simulink generated code (needs at least a simulation step before update the input state).
This difference shouldn't be a problem if all the FMU models used are Simulink generated, but even in a mixed-tool model environment the differences could be not noticable (dependes on the modeled systems and simulation objectives).
More complex and advanced models comparations are needed to evaluate and highlights further possible differences between the two tools code generation alghoritms.