Exploiting Automatic Abstraction and the FMI Standard to build Cycle-accurate Virtual Platforms from Heterogeneous IPs

Vladislav Bragoi - VR436747

***Sommario*—In questo documento vengono descritte le principali fasi di creazione dei pacchetti FMU, al fine di poter co-simulare un sistema costituito da IP eterogenei.**

I. Introduzione

In questo progetto si vogliono provare e analizzare i principali vantaggi e le potenzialità introdotte dallo standard FMI, andando a toccare con mano una possibile implementazione di un modulo rispettando lo standard, e la sua integrazione in un sistema più complesso costituito di elementi eterogenei.

II. Background

Il Functional Mockup Interface (FMI)[1] è un protocollo che standardizza la comunicazione tra modelli ciberfisici eterogenei scritti in diversi linguaggi di programmazione, supportandone la co-simulazione attraverso la definizione di un'interfaccia del singolo modello definita in un file xml e la cui funzionalità è interamente espressa da un sorgente o compilato in codice C. Il singolo elemento che implementa tale protocollo è detto Functional Mockup Unit (FMU).

III. Metodologia applicata

Poiché questo è un tutorial che andava semplicemente completato, è stato sufficiente completare le parti mancanti nei sorgenti C++ del modulo *gain* e integrarne la funzionalità all'interno del coordinatore definito nel file `coordinator.py`. Di seguito i dettagli.

A. GAIN

Il modulo gain da specifica deve moltiplicare per 10 (costante *GAIN_VALUE*) il valore che gli arriva in ingresso. Per fare ciò, oltre ad aggiungere la variabile `result` nel file header del modulo, che dev'essere inoltre definita anche nel file `modelDescription.xml` come porta di output e inizializzata a 0, è stato necessario modificarne il comportamento come segue:

- se *data_rdy* è a true, il valore del dato dev'essere moltiplicato per la costante *GAIN_VALUE* e memorizzato nella variabile `result` definita precedentemente,
- `result` deve essere settato a 0 altrimenti.

Compilando il modulo con il comando `cmake` viene generato il file `.so` da utilizzare nella successiva generazione dell'archivio `.fmu`.

B. Generazione delle FMU

Per generare gli archivi `.fmu` è sufficiente utilizzare il comando `make` una volta che ci si è posizionati nella directory contenente, come già detto precedentemente, i file `.xml` dell'interfaccia del modulo appunto e `.so` del codice binario che ne racchiude la funzionalità. I makefile che accompagnano ciascun modulo contengono al loro interno i comandi necessari per la generazione delle corrispondenti functional mockup unit, a partire da codice Verilog-A (analog) per il modulo *Accelerometer* e Verilog per il modulo *m6502* e il modulo *memory*. Questi comandi vengono riportati in figura 1, e nello specifico sono:

- `verilog2hif`, per tradurre il codice Verilog nel formato HIF utilizzato dall'insieme dei tool che costituiscono la HIFSuite,
- `analyst`, per generare modelli HIF a eventi discreti a partire da un design analogico quale quello dell'Accelerometer,
- `ddt`, per una conversione dei tipi di dato definiti nel modulo con quelli nativi del codice C,
- `a2tool`, per una traduzione in C++/SystemC a partire dalla rappresentazione HIF-RTL,
- `hif2vp`, per la generazione del documento xml che descrive il modulo, da inserire nell'archivio `.fmu`.

C. Coordinatore

Il coordinatore è l'elemento tramite il quale riusciamo a simulare l'intero sistema sfruttando lo standard FMI. Essendo scritto in Python, il framework PyFMI[2] è di riferimento per implementare le funzionalità del master in modo da poter caricare le varie FMU e farle interagire tra loro. La sua implementazione è definita nel file `coordinator.py` ed è stata aggiornata per includere la funzionalità aggiunta del modulo gain.

IV. Risultati

In figura 2 è possibile verificare come la funzionalità del gain è stata perfettamente implementata sfruttando lo standard FMI. Il valore in input (nell'immagine a sinistra) viene moltiplicato per 10 come da specifica (immagine a destra).

V. Conclusioni

Il tutorial ha permesso di verificare i vantaggi introdotti dallo standard FMI, che risultano dunque concentrarsi sulla

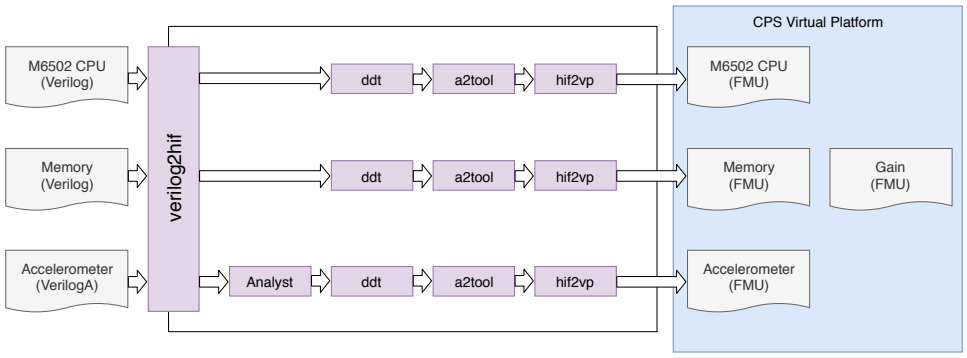

Figura 1: Comandi necessari per generare i file `.fmu` a partire dai vari sorgenti

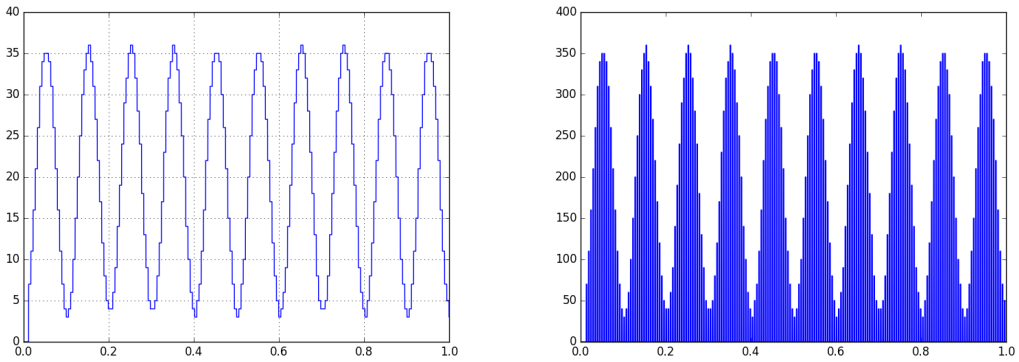

Figura 2: Risultato della simulazione del modulo gain: input a sinistra e output a destra

possibilità di far integrare in una piattaforma virtuale diversi componenti eterogenei, senza preoccuparsi del loro linguaggio di definizione, per permetterne una facile simulazione.

RIFERIMENTI BIBLIOGRAFICI

[1] Modelica, "The functional mockup interface for tool independent exchange of simulation models," https://svn.modelica.org/fmi/branches/public/docs/Modelica2011/The_Functional_Mockup_Interface.pdf, 2011.

[2] M. AB, "Python pyfmi package for loading and interacting with functional mock-up units," https://pypi.org/project/PyFMI/, 2018.