Sessione hands-on del tutorial "Exploiting Automatic Abstraction and the FMI Standard to build Cycle-accurate Virtual Platforms from Heterogeneous IPs"

Massimiliano Incudini - VR433300

***Sommario*—Il seguente documento documenta il lavoro svolto durante la seconda parte del modulo di laboratorio del corso Progettazione di Sistemi Embedded (aa 2018/2019). Il suo obiettivo è illustrare passo per passo la procedura che genera i pacchetti `fmu` a partire da sorgenti eterogenei forniti e simulare il comportamento dell'intero sistema fatto dalla connessione di questi componenti.**

I. Introduzione

Il progetto consiste nel modificare, compilare e interconnettere gli elementi del sistema in Figura 1 esportati secondo il formato del protocollo *Functional Mockup Interface*.

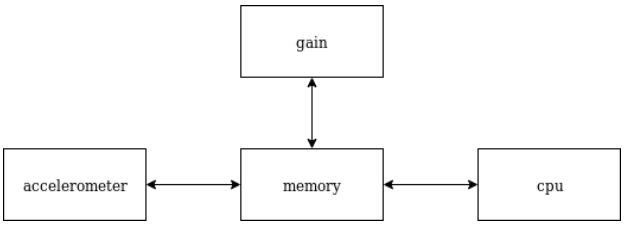

Figura 1. Sistema

Le componenti del sistema comunicano tramite memory-mapping.

II. Protocollo FMI

FMI è uno standard indipendente pensato per supportare sia *scambio di modelli* che *cosimulazione* di modelli dinamici attraverso una combinazione di file XML e codice C (sia compilato che in formato sorgente)[1].

Lo standard, tra le altre cose, definisce:

- una API (C) per eseguire le funzioni di una FMU attraverso codice C;
- l'FMI Description Schema, file XML-Schema che descrive come deve essere il documento contenente le informazioni statiche (di interfaccia) della FMU; queste informazioni sono contenute in un file `modelDescription.xml`

III. Modulo Gain

Questa sezione si riferisce al contenuto della directory `/fmi_lesson/models/gain`.

Il modulo Gain è descritto nel linguaggio C++ all'interno della sottodirectory `/cpp`. La sua interfaccia viene modificata aggiungendo un campo `result` di tipo `int`. All'interno del codice, questa variabile implementa la funzione

$$\texttt{result} = \begin{cases} \texttt{data} * 10, & \texttt{data\_rdy} = 1 \\ 0 & \text{altrimenti} \end{cases}$$

Una volta che il dato è stato scritto poniamo `result_rdy` alto. Tale flag verrà ri-settato basso in altri punti di codice.

Una volta processato con `cmake` e `make` otteniamo un eseguibile binario `.so`. All'interno di `/fmu` è presente il file XML di descrizione della piattaforma che deve essere anch'esso modificato. All'interno viene aggiunta la porta definita nei sorgenti:

```
<ScalarVariable
    causality="output"
    description="result data port"
    name="result"
    valueReference="1"
    variability="discrete">

        <Integer start="0"/>
</ScalarVariable>
```

Come possiamo vedere la variabile è di output, di tipo intero ed il suo identificatore è settato ad 1. Ogni variabile è difatti identificata dalla coppia $\langle \texttt{tipo}, n \rangle$. Essendoci già una variabile intera (campo `data` in input) non utilizziamo il primo numero identificativo per il tipo (0) ma il suo successore. Attraverso il comando `make` viene preso il file `.so` generato dalla compilazione del modello, ed impacchettato insieme al documento XML per generare la *functional mockup unit* (file `.fmu`).

IV. Moduli da compilare

Questa sezione si riferisce al contenuto della directory `/fmi_lesson/models/[nome modello]`.

I moduli del progetto presenti unicamente come sorgenti che devono essere compilati sono:

- accelerometer: sensore analogico descritto in Verilog-A;
- m6502: processore descritto in Verilog;
- mem: memoria scritta in Verilog.

Ognuno di questi progetti viene elaborato in un pacchetto `fmu` attraverso le utility del software HIFSuite.

1) Il primo step consiste nell'avviare il frontend `verilog2hif`, che a partire dal Verilog restituisce lo stesso file secondo la rappresentazione interna del tool;

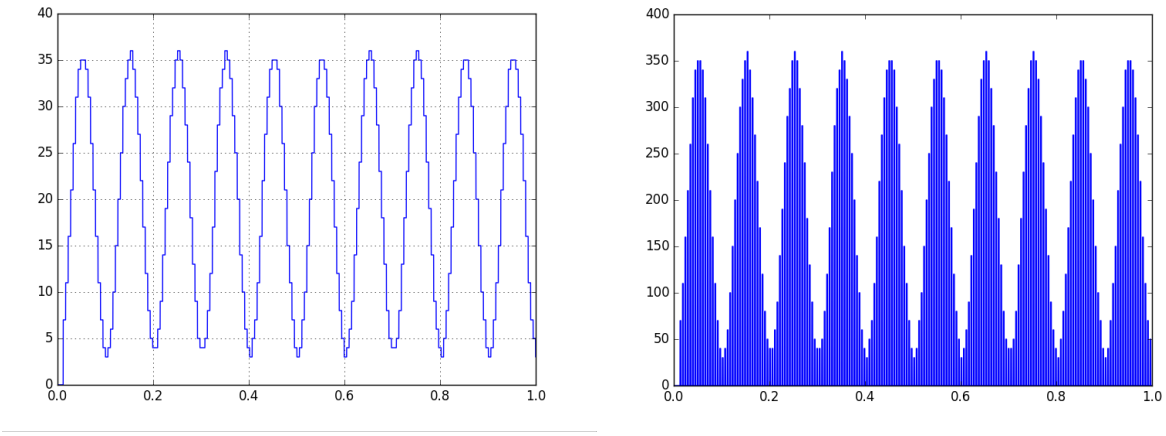

Figura 2. Risultato della simulazione con PyFMI. A sinistra il segnale in input al gain, a destra l'output.

2) il secondo step consiste nell'applicare delle ottimizzazioni;
3) il terzo step consiste nell'avviare il backend `hif2sc` che restituisce la rappresentazione del modulo in C++;
4) il quarto step consiste nell'avviare l'utility `hif2vp` che restutisce il documento XML relativo alla rappresentazione del modulo;
5) il quinto step consiste nel compilare il codice C++ e comprimerlo in un archivio `zip` insieme al documento XML.

V. Simulazione del sistema

Questa sezione si riferisce al contenuto della directory `/fmi_lesson/coordinator`.

Per simulare il sistema viene utilizzato il framework PyFMI basato su linguaggio Python. I comandi necessari per la simulazione sono presenti nel file `coordinator.py`. I moduli, ad esempio il Gain, vengono caricati come segue:

```
# Carico le FMU
gain = load_fmu('./fmus/gain.fmu')
# Inizializzo le FMU
gain.initialize()
# Eseguo uno step di computazione
gain.do_step( ... )
# Leggo le porte di output
gain.get_integer(GAIN_RESULT)
# La scrittura delle porte invece è
# tutta nel metodo data_exchange
```

Possiamo vedere in Figura 2. Nella parte a sinistra abbiamo il segnale in input al gain. A destra abbiamo il segnale. Come possiamo vedere i valori sono moltiplicati $\times 10$ rispetto all'input, meno che alcuni punti: questi sono gli istanti nei quali `data_rdy`.

Riferimenti bibliografici

[1] M. "FMI", "Fmi for model exchange and co-simulation," 2014.