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

Mirko Albanese - VR431583

***Sommario*—Questo documento descrive il lavoro svolto durante la seconda parte del progetto. Il suo obiettivo è quello di illustrare passo per passo la procedura che genera i pacchetti `fmu` a partire da sorgenti eterogenei forniti e simulare il comportamento dell'intero sistema composto 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*.

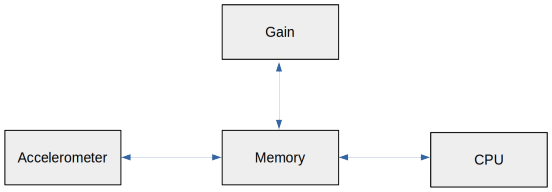

Figura 1. Sistema

La comunicazione tra le componenti è basa su memory mapping.

II. Protocollo FMI

FMI (`Functional Mockup Interface`) 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).[M. FMI, 2014] Esso definisce:

- API scritta in linguaggio C per eseguire le funzioni di una FMU;
- `FMI Description Schema`: file XML-Schema che descrive come deve essere il documento contenente le informazioni di interfaccia della FMU.

III. Modulo Gain

Questa sezione di riferisce al contenuto della directory `/fmi_lesson/mmodels/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 seguente funzione

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

Processando il file con `cmake` e `make` otteniamo un eseguibile binario `.so`. All'interno della cartella `/fmu` vi è un ulteriore file in formato XML da modificare. Viene aggiunta una porta definita nel seguente modo:

```
<ScalarVariable
causality="output"
description="result data port"
name="result"
valuereference="1"
variability="discrete">
<integer start="0"/>
</ScalarVariable>
```

Ogni variabile è identificata dalla coppia (tipo,n).
Effettuando la compilazione del modello e generando il file `.so`, esso viene impacchettato insieme al documento XML per generare la *functional mockup unit*(.fmu).

IV. Moduli da Compilare

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

I sorgenti che devono essere compilati sono i seguenti:

- Accelerometer: Sensore descritto in Verilog-A;
- m6502: Processore descritto in Verilog;
- mem: Memoria scritta in Verilog.

Ognuno di questi viene elaborato in un pacchetto `fmu` attraverso le utility del software HIFSuite. In primo luogo viene avviato il frontend `verilog2hif`, che a partire da un file Verilog restituisce lo stesso file secondo la rappresentazione interna del tool. Di seguito vengono applicate ottimizzazioni e successivamente viene avviato il backend `hif2sc` che a partire dal'HIF restituisce la rappresentazione del modulo in C++. Il quarto step consiste nell'avviare l'utility `hif2vp` che restituisce il documento XM relativo alla rappresentazione del modulo, e, infine, lo step finale consiste nel compilare il codice C++ e comprimerlo in un archivio `zip` insieme al documento XML.

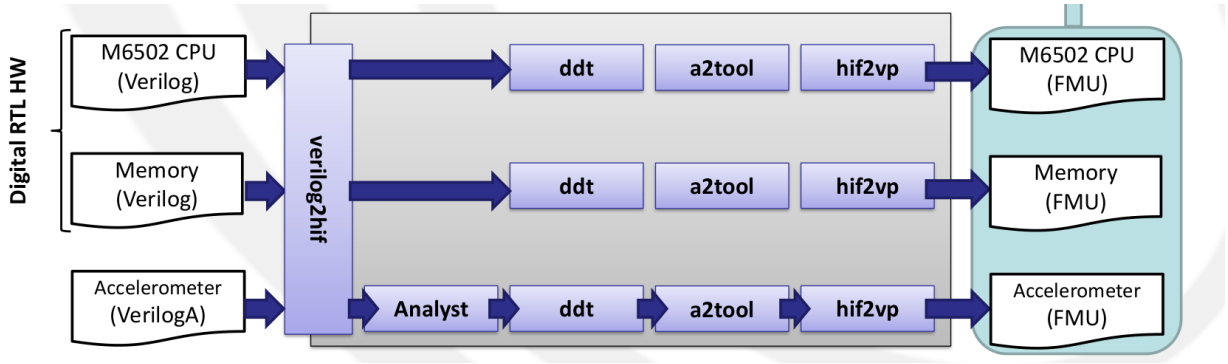

Figura 2. Comandi necessari per gestire i file a partire dai sorgenti.

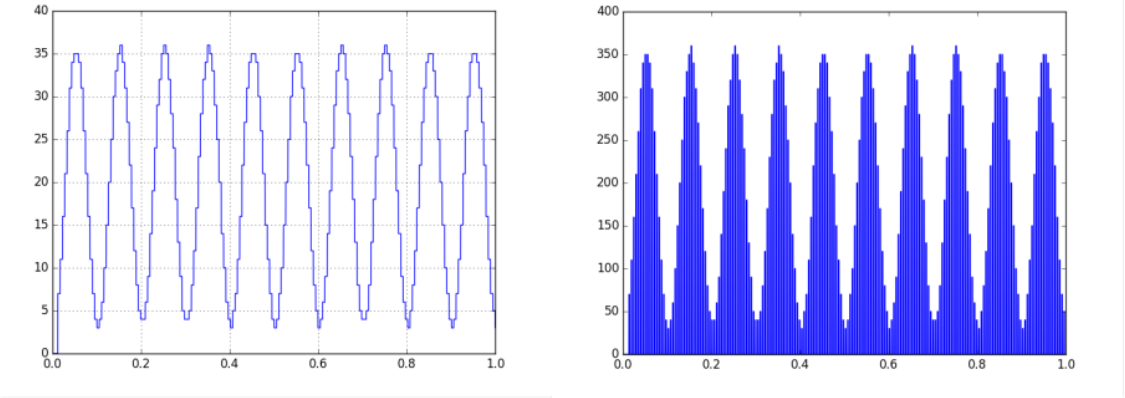

Figura 3. Risultato della simulazione con PyFMI. A sinistra il segnale in input, a destra il segnale in output.

V. Simulazione del Sistema

Questa sezione si riferisce al contenuto della directory `/fmi_lesson/models/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, prendiamo come esempio il modulo Gain, vengono caricati nel modo seguente:

```
#Carico FMU
gain = load_fmu('./gmus/gain.fmu')
#Init FMU
gain.initialize()
#Eseguo la computazione
gain_do_step(....)
#Leggo dalle porte di output
gain.get_integer(GAIN_RESULT)
```

Possiamo vedere in figura 3, nella parte a sinistra abbiamo il segnale in input al gain, a destra, il segnale in output. Come possiamo notare i valori vengono moltiplicati per un fattore 10 rispetto ai valori di input.