Modellazione in SystemC di un sistema hardware/software per il controllo del livello dell'acqua di una tanica forata

Simone Girardi - VR421147
Dipartimento di Informatica
Universitá Degli Studi Di Verona
Email: simone.girardi@studenti.univr.it

***Sommario*—Lo standard SystemC [1] pubblicato a Dicembre 2013 e lo standard SystemC AMS 2.0 [2] pubblicato a Marzo 2013 dal gruppo di lavoro AMS Working (AMSGW) forniscono strumenti di modellazione Hardware e Software che permettono di modellare, e simulare, sistemi complessi composti dall'insieme di componenti digitali e analogici. L'obbiettivo di questo progetto é dimostrare le potenzialitá di questi due strumenti, modellando un sistema in grado di mantenere stabile il livello dell'acqua di una tanica avente una perdita costante. Il sistema é quindi composto da un controllore e un decifratore (parte digitale), e poi da una valvola e una tanica forata (parte analogica).**

I. Introduzione

Le principali caratteristiche che un sistema di controllo deve avere sono le seguenti:

- Una parte di sensoristica, in grado di rilevare i dati utili e necessari per monitorare lo stato del sistema.
- Una parte di controllo che riceve i dati dai sensori, li elabora, e cerca di mantenere il sistema stabile.
- Una parte di attuazione che risponde ai comandi inviati dal controllore e agisce sull'impianto fisico del sistema.

Per poter modellare un sistema embedded minimamente complesso, sono necessari strumenti di progettazione in grado di modellare sistemi hardware digitali per la parte di controllo, e componenti analogici per la parte di sensoristica e di attuazione. Questi strumenti sono dettagliati nel corso del report, mostrando quali siano le caratteristiche principali e le motivazioni all'adozione degli stessi.

Durante la progettazione di un sistema di questo tipo le problematiche principali da affrontare riguardano soprattutto: la definizione formale delle specifiche, la suddivisione hardware e software dei componenti di cui il sistema é composto, nonché la verifica ed il test del sistema prodotto. Infine vi sono da tenere in considerazione anche problematiche legate ad aspetti commerciali quali il *time to market*.

Questo report vuole riassumere il lavoro svolto nella progettazione di un sistema che comprende tutte le caratteristiche sopracitate mostrando nel dettaglio il flusso di progettazione e tutte le problematiche riscontrate.

II. Background

SystemC é insieme di librerie e macro scritte in C++, volte ad estendere il linguaggio stesso in modo da permettere una co-modellazione ad alto livello di componenti hardware e software. Durante il flusso di progettazione, adottare una strategia a livello transazionale (TLM) permette di avere vantaggi sui tempi di progettazione e simulazione. A questo livello di astrazione di SystemC, viene data molta enfasi alla comunicazione tra moduli dove entra in gioco il concetto di transazione. Inoltre é da notare che, secondo quanto definito dallo standard TLM 2.0, le definizioni delle interfacce si distinguono dalle descrizioni degli stili di codifica riportati in figura 1.

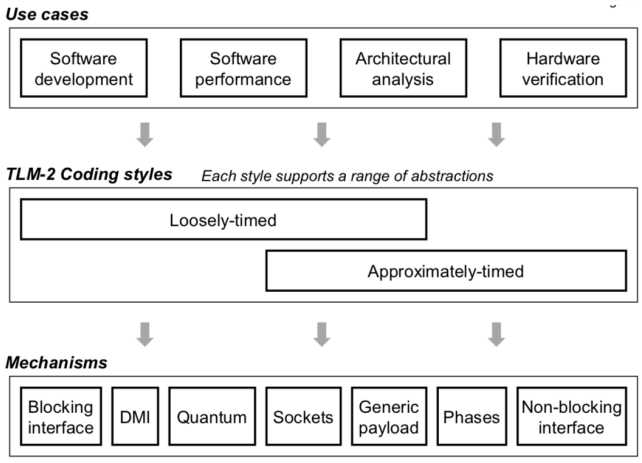

Figura 1: TLM Coding Style

Ogni stile di codifica puó supportare diverse astrazioni tra funzionalitá, tempistica e comunicazione. Uno stile di codifica Loosely-timed (LT) é appropriato quando si sviluppa utilizzando un modello di piattaforma virtuale di un MPSoC, in cui il contenuto del software puó includere uno o piú sistemi operativi. Questo stile inoltre, supporta la modellazione di timer ed interrupt, ed é quindi sufficiente per avviare un sistema operativo ed eseguire del codice arbitrario sulla macchina di destinazione.

Lo stile di codifica Approximately-timed (AT) risulta appropriato nel caso dell'esplorazione dell'architettura e dell'analisi delle prestazioni. Esso prevede l'utilizzo di interfacce non bloccanti che forniscono annotazioni temporali e piú fasi distinte, durante la vita di una transazione.

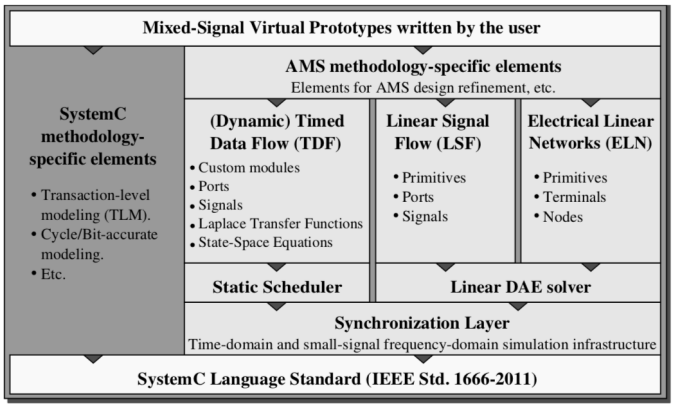

Figura 2: Architecture of SystemC AMS 2.0

Una soluzione a livello di registro (RTL), permette invece di avere una maggiore accuratezza nella descrizione di un modello, a discapito di un'implementazione piú onerosa e una simulazione decisamente piú lenta. SystemC AMS fornisce una metodologia uniforme e standardizzata per la modellazione e la simulazione di sistemi analogici (Analog/Mixed-Signal) a livelli di astrazione piú alti. Le estensioni AMS, sono costruite sulla base dello standard del linguaggio SystemC [1] e supportano i modelli di computazione (MoC) mostrati in Figura 2. Il MoC Timed Data Flow (TDF) consente la modellazione a tempo discreto, ed un'efficiente simulazione a livello funzionale degli algoritmi di elaborazione di segnali. Il MoC Linear Signal Flow (LSF) supporta la modellazione di comportamenti a tempo continuo. Infine il MoC Electrical Linear Network (ELN) consente la modellazione di reti elettriche.

III. Metodologia applicata

Il primo passo era volto a comprendere meglio gli aspetti coinvolti durante la modellazione di un modulo HW, che in questo caso era dedicato a svolgere la sola funzionalitá di cifratura/decifratura : XTEA [3].

A. XTEA: RTL

Facendo riferimento all'implementazione sequenziale scritta in C++ dello XTEA, é stata definita l'EFSM (Extended Finite State Machine) mostrata in figura 9, in grado di descrivere completamente e deterministicamente tutti i passi dell'algoritmo stesso. Una volta chiaro il flusso di controllo dello XTEA, si é passati ad un'implementazione RTL dello stesso, realizzando cosí il modulo rappresentato in figura 3.

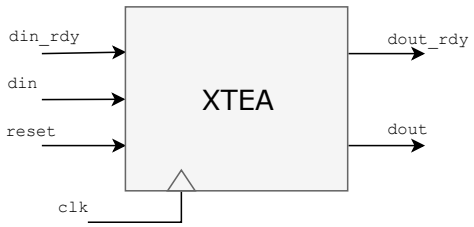

Figura 3: Architecture of XTEA

dove *din* e *dout* rappresentano l'insieme di input {*word0*, *word1*, *key0*, *key1*, *key2*, *key3*, *mode*}, e l'insieme di output {*result0*, *result1*} rispettivamente.

Per verificare il corretto funzionamento dello XTEA RTL é stato costruito un apposito testbench in grado di stimolare il modulo con diversi input, poi, tramite un tool di visualizzazione grafica dei segnali (e.g. GTKWave), é stato quindi verificato che il comportamento simulato rispecchi effettivamente quello atteso.

Per dividere il controllo dall'elaborazione é stata implementata una rappresentazione dell'FSMD a 2 processi:

1) computa lo stato prossimo e le uscite (datapath)
2) aggiorna lo stato corrente (controllore)

I segnali utilizzati per la comunicazione tra il testbench e il modulo XTEA sono i seguenti:

```
1 sc_clock                      CLOCK;
2 sc_signal < bool >            rst;
3 sc_signal<sc_uint<32> >       word0, word1;
4 sc_signal<sc_uint<32> >       k0, k1, k2, k3;
5 sc_signal< bool >             mode;
6 sc_signal<sc_uint<1> >        din_rdy;
7 sc_signal<sc_uint<1> >        dout_rdy;
8 sc_signal<sc_uint<32> >       result0, result1;
```

Mentre sono stati aggiunti dei segnali interni allo XTEA per far evolvere la macchina a stati:

```
1 sc_signal<sc_uint<32> > i, delta, v0, v1, temp;
2 sc_signal<sc_uint<32> > key0, key1, key2, key3;
3 sc_signal<sc_uint<64> > sum;
4 sc_signal<bool>         mode;
```

Infine nel costruttore dello XTEA é stata definita la seguente sensitivity list:

```
1 SC_CTOR(xtea_RTL) {
2     SC_METHOD(datapath);
3     sensitive << rst.neg();
4     sensitive << clk.pos();
5
6     SC_METHOD(fsm);
7     sensitive << STATUS;
8     sensitive << din_rdy << mode_in << i;
9 }
```

Mentre il testbench é sensibile solo al fronte di salita del clock.

Quando viene avviata la simulazione il clock attiva entrambi i moduli contemporaneamente: testbench e XTEA. Il testbench inizia a configurare i paramentri necessari alla cifratura/decifratura, quindi setta *din_rdy*= 1; mentre lo XTEA parte dallo stato di *Reset* e si porta nello stato *S_0* in attesa di un input. Dal successivo fronte di salita del clock, fino a quando lo XTEA non ha terminato la cifratura/decifratura, il testbench si mette in attesa dei risultati, mentre lo XTEA continua ad evolvere passando agli stati successivi sotto la guida della macchina a stati. É da notare, che é stato deciso di non creare ulteriori stati per i costrutti di controllo di flusso interni, a quello che in origine era un ciclo for, con lo scopo di ottenere un circuito che occupa meno area. Quando lo XTEA raggiunge lo stato S_6 scrive i risultati sulle porte di uscita e setta $dout_rdy = 1$. A questo punto il testbench si sblocca dall'attesa e prosegue fino a terminare o procedere con l'inizializzazione di un ulteriore computazione (e.g. decifratura), mentre lo XTEA si riporta allo stato di partenza S_0 e rimane in attesa di un nuovo input.

B. XTEA: TLM

Una volta compreso il significato e la complessitá di realizzare un modello RTL, é stato deciso di seguire un flusso di progettazione piú efficiente, ma meno accurato rispetto al modello ”reale”, adottando quindi una soluzione a livello transazionale. Per poter effettuare una transazione tra due moduli é necessario prima di tutto definire chi inizierá la comunicazione (initiator) e chi sará il destinatario finale di questa transazione (target): nel nostro caso é stato deciso che l'initiator sará il testbench mentre il target sará il nostro modulo XTEA TLM. Questo meccanismo é reso possibile associtando all'initiator socket del testbench, la target socket dello xtea nel seguente modo:

```
// bind the TLM ports
initiator.initiator_socket(target.target_socket);
```

A questo punto il flusso prevede di progettare un primo modello che permetta di verificarne la comunicazione, si tratta quindi di un'implementazione che non tiene conto del tempo (UT). Di conseguenza é stato sufficiente attenersi al protocollo standard implementando le funzioni richieste, tra le quali la piú importante é sicuramente la b_transport, in cui di seguito é riportato lo pseudocodice:

```
1  void b_transport(TRANS& trans, sc_core::sc_time&
       t) {
2      // download data from the payload
3      data = trans.get_data_ptr()
4
5      // compute the functionality
6      xtea_function();
7
8      //save result
9      data.result = xtea_result;
10
11     // and load it on the payload
12     trans.get_data_ptr() = data;
13 }
```

Quando questa funzione viene invocata dal testbench passando i dati nel formato standard (generic payload), lo xtea inizia ad eseguire le istruzioni contenute in essa e, fino a quando non termina, il testbench rimane bloccato sulla chiamata. Lo xtea esegue quindi la sua funzionalitá come fosse una normalissima funzione C++, rispettando peró le necessitá di comunicazione imposte dal protocollo. Una volta che il target termina la b_transoport, il testbench si sblocca continuando cosí la sua esecuzione, che in questo caso non fa altro che verificare la correttezza dei dati prodotti dallo xtea, confrontandoli con dei valori obbiettivo.

Una volta verificata la corretta comunicazione dello xtea, si é passati ad un raffinamento dello stesso, associando del tempo alle transazioni, con lo scopo di ottenere una soluzione Loosely-timed. Per fare questo, a partire dall'implementazione UT, é stato sufficiente associare del tempo alle transazioni, introducendo il concetto di: quantum keeper, timing annotation e temporal decoupling. In questo modo é stato possibile abbattere l'ordine esatto degli eventi eseguendoli in ordine diverso, ma senza compromettere la correttezza del modello rispetto alle specifiche. L'intervallo, tra un quanto di tempo e il successivo, per questa implementazione é stato impostato a 500ns.

Raffinando ulteriormente il nostro modello é stato possibile ottenere una soluzione Approximately-timed, raggiungendo quindi la massima accuratezza possibile secondo lo standard TLM. Per raggiungere questa accuratezza, oltre ad introdurre il concetto di clock, é stato necessario aggiungere piú punti di sincronizzazione, e sostituire la b_transport con l'interfaccia non bloccante di seguito riportata in pseudocodice:

```
1  tlm_sync_enum nb_transport_fw(TRANS& trans,
       PHASE& phase, sc_time& t) {
2    // Verify there is no pending transaction
3    if (pending_transaction != NULL) {
4      trans.set_response_status(tlm::
           TLM_GENERIC_ERROR_RESPONSE);
5      return tlm::TLM_COMPLETED;
6    }
7
8    // Phase must be BEGIN_REQ (begin request)
9    if (phase != tlm::BEGIN_REQ) {
10     trans.set_response_status(tlm::
           TLM_GENERIC_ERROR_RESPONSE);
11     return tlm::TLM_COMPLETED;
12   }
13
14   // download the payload
15   pending_transaction = &trans;
16   data = trans.get_data_ptr()
17
18   // target accepted to process the data
19   phase = tlm::END_REQ;
20
21   // activate the ioprocess process
22   io_event.notify();
23
24   // return control
25   return tlm::TLM_UPDATED;
26 }
```

É da notare che alla riga 22 si prevede l'esistenza di una funzione *ioprocess()* in attesa di un evento. Questa funzione si occupa di gestire le restanti fasi del protocollo e di richiamare la funzionalitá dello xtea.

Arrivati a questo punto, é doveroso confrontare i risultati delle tre diverse implementazioni a livello TLM mostrati dalla tabella I.

	UT	LT	AT4	RTL
real	0,579	0,648	1,293	4,347
user	0,168	0,221	0,342	3,969
sys	0,142	0,158	0,293	0,300

Tabella I: Risultati a confronto

Da questa tabella risulta chiaro che, ogni stile di codifica implica una maggiore accuratezza dei tempi, ma con lo svantaggio di avere una simulazione piú lenta. Questo risultato é del tutto atteso, dal momento che ad ogni livello di raffinamento vengono aggiunti dei punti di sincronizzazione. Il grafico mostrato in figura 4 mostra i tempi di simulazione *real* per ogni stile di codifica al variare del numero di iterazioni. Al grafico é stato inoltre aggiunto il confronto con livello di astrazione RTL, in modo da avere una chiara idea di quanto possa differire il tempo di simulazione rispetto allo standard TLM.

C. WATERTANK SYSTEM: AMS

Compresi gli aspetti legati al mondo digitale, dove il tempo é discretizzato, il passo successivo é stato quello di com-

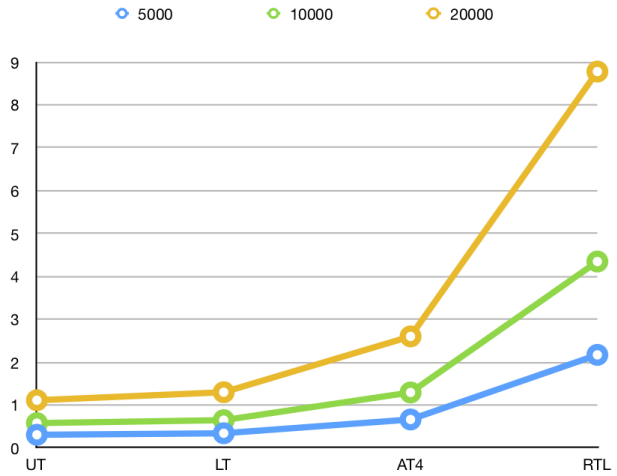

Figura 4: TLM Coding Style

prendere la modellazione di componenti analogici. Come giá citato nel sezione II del presente report, SystemC AMS é lo strumento che supporta la modellazione di componenti a tempo continuo. In particolare é stato deciso di modellare il sistema rappresentato in figura 5, il quale rappresenta l'insieme composto da un controllore, una valvola ed un serbatoio. Il controllore effettua una rilevazione del livello dell'acqua contenuta nel serbatoio ogni 5 secondi, se questo dovesse essere al di sotto del valore 5 o al di sopra del valore 9, il controllore provvederá a mandare gli opportuni input alla valvola in modo che essa possa aprirsi o chiudersi e compensare quindi la perdita costante presente nel serbatoio. Per modellare un sistema di questo tipo, é stato deciso di adottare un modello computazionale TDF per il controllore e la valvola, mentre é risultato piú intuitivo usare un MoC LSF per quanto riguarda il serbatoio.

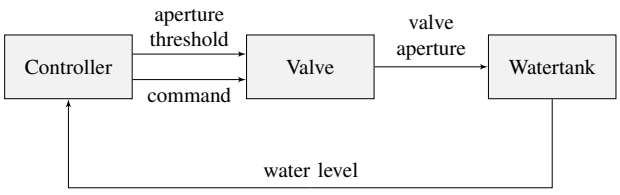

Figura 5: AMS System

Durante l'implementazione dei moduli TDF, una prima osservazione é stata fatta sui tempi di attivazione (*timestep*). In particolare il controllore deve essere decisamente piú lento della valvola (e del serbatoio) per poter dare il tempo alla tanica di riempirsi, ed eventualmente di svuotarsi.

É stato quindi deciso di impostare un timestep di 5 secondi per il controllore ed 1 millisecondo per la valvola.

Secondo lo standard SystemC AMS, ogni modulo di un MoC di tipo TDF dovrá avere un timestep che soddisfa la seguente relazione:

$$\begin{aligned} \textit{module timestep} &= \textit{input port timestep} \cdot \textit{input port rate} \\ &= \textit{output port timestep} \cdot \textit{output port rate} \end{aligned}$$

Questa relazione implica che il modulo attivo deve sempre avere dei dati da consumare pari al numero di campioni che vengono prodotti in quell'istante. Perció al fine di raggiungere gli obbiettivi prefissati, per quanto riguarda la valvola sono stare raggiunte le seguenti impostazioni:

```
1 set_timestep(1.0, SC_MS);
2 command_in.set_timestep(1.0, SC_MS);
3 command_in.set_rate(1);
4 a_threshold_in.set_timestep(1.0, SC_MS);
5 a_threshold_in.set_rate(1);
6 valve_aperture_out.set_timestep(1.0, SC_MS);
7 valve_aperture_out.set_rate(1);
```

mentre per quanto riguarda il controllore si ha che:

```
1 set_timestep(5.0, SC_SEC);
2 water_level_in.set_delay(5000);
3 water_level_in.set_rate(5000);
4 a_threshold_out.set_timestep(0.001, SC_SEC);
5 a_threshold_out.set_rate(5000);
6 command_out.set_timestep(0.001, SC_SEC);
7 command_out.set_rate(5000);
```

Sovracampionando il segnale in uscita dal controllore, vengono generati 5000 campioni, mentre la valvola ne consuma e ne genera 1 ogni millisecondo. Ogni campione consumato e prodotto dalla valvola viene trasferito al serbatoio, il quale lo elabora e genera un nuovo campione da propagare verso il controllore. Di conseguenza, dopo 5 secondi il controllore ha a disposizione 5000 nuovi campioni da elaborare per determinare la nuova threshold e i nuovi comandi da dare alla valvola. Per semplicitá é stato deciso di permettere al controllore di prendere in considerazione solo l'ultimo campione emesso dal serbatoio, e di scartare quindi tutti gli altri. Un implementazione piú smart, avrebbe previsto l'utilizzo di tutti i campioni per stimare il livello dell'acqua negli istanti di tempo successivi, anticipando quindi la chiusura della valvola. Ad esempio se alla seconda attivazione il controllore, sapendo di aver impostato una certa apertura della valvola, rileva che ogni campione differisce di una certa quantitá, puó usare questa informazione per accorgersi che il livello dell'acqua supererá la soglia di 8.8 e quindi potrá iniziare a chiudere la valvola prima di superare tale soglia. Ogni modulo TDF deve implementare una funzione *processing()* che verrá attivata ad ogni time step. Questa funzione rappresenta la funzionalitá del modulo in cui é contenuta. In particolare il controllore, non fa altro che calcolare la nuova threshold e i nuovi comandi da dare alla valvola, sulla base del livello dell'acqua attualmente letto. Di seguito é riportato lo pseudocodice che mostra quanto appena descritto:

```
1  void controller_tdf::processing() {
2      if (water_level > 8.8) {
3          current_threshold *= 0.7;
4          a_threshold_out.write(current_threshold);
5          command_out.write(CLOSE);
6      }
7      else if (water_level < 5) {
8          current_threshold *= 1.1;
9          a_threshold_out.write(current_threshold);
10         command_out.write(OPEN);
11     }
12     else {
13         a_threshold_out.write(current_threshold);
14         command_out.write(IDLE, i);
15     }
16 }
```

Per quanto concerne la valvola invece il suo compito é quello di leggere i valori trasmessi dal controllore e trametterе di conseguenza la giusta apertura al serbatoio. Per rappresentare la variazione di apertura della valvola nel tempo, é stata effettuata un operazione di integrazione sulla threshold come mostrato dal seguente pseudocodice:

```
1  void valve_tdf::processing() {
2      delta = 0.25 * get_timestep().to_seconds();
3
4      switch (command) {
5        case IDLE:
6          valve_aperture_out.write(a_threshold);
7          break;
8        case OPEN:
9          a_threshold = a_threshold + delta;
10         valve_aperture_out.write(a_threshold);
11         break;
12       case CLOSE:
13         a_threshold = a_threshold - delta;
14         valve_aperture_out.write(a_threshold);
15         break;
16     }
17 }
```

Arrivando a descrivere il comportamento del serbatoio, é molto importante aver compreso il significato di effettuare le operazioni nei sistemi lineari a tempo continuo. Per poter rappresentare il livello dell'acqua all'uscita dal serbatoio, in funzione della variazione di apertura della valvola, e tenendo conto della perdita costante, é necessario operare secondo la seguente relazione:

$$\dot{x} = 0.6 * a - 0.3 * x$$

Dove a rappresenta la variazione di apertura della valvola e x é il livello attuale dell'acqua. É da notare che essendo una variazione, questo risultato va sottoposto ad integrazione prima di essere trasferito al controllore. Il diagramma in figura 6 rappresenta l'implementazione del serbatoio.

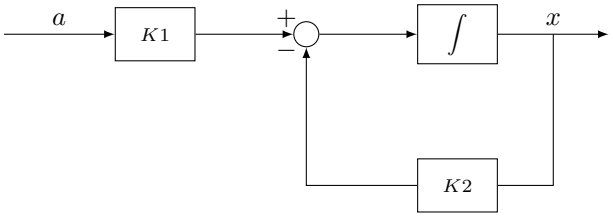

Figura 6: Watertank block diagram

Dopo aver simulato il modello AMS, attraverso GTKWave o tool analoghi, é possibile notare come il livello dell'acqua (inizialmente a zero) inizi a crescere progressivamente fino a superare il valore 5, per poi crescere linearmente fino a 8.8 (o poco piú) e quindi iniziare a scendere. Dopo un pó di secondi si nota che il livello dell'acqua inizia a stabilizzarsi all'interno dell'intervallo 5 - 8.8, questo perché la valvola ha raggiunto un'apertura tale da compensare esattamente la perdita.

D. WATERTANK SYSTEM

Dopo aver compreso sia gli aspetti che riguardano la modellazione di componenti digitali, sia quelli relativi al mondo analogico, non rimane altro che mettere insieme il tutto e realizzare cosí una piattaforma eterogenea. Per raggiungere questo obbiettivo, é necessario ricorrere all'uso di componenti ibridi chiamati transattori, ossia componenti dotati di interfacce *speciali*, che permettono la comunicazione sia tra i diversi livelli di astrazione di SystemC (TLM - RTL), sia tra moduli a tempo continuo e moduli a tempo discreto (TLM/RTL - AMS). Per realizzare questo sistema sono stati inseriti due transattori TLM-RTL ed un'interfaccia RTL-AMS. Il controllore é stato implementato secondo lo standard TLM LT ed é dotato di due initiator socket: una collegata alla target socket del transattore comunicante con lo xtea (*controller transactor*), una collegata alla target socket del transattore comunicante con il serbatoio *watertank transactor*. La configurazione del sistema appena descritto é mostrata nella figura 10. Seguendo quanto indicato dallo standard TLM [4, p. 73], il payload verso il *controller transactor* é stato impostato in modalitá scrittura, mentre il payload verso il *watertank transactor* é stato impostato in modalitá lettura. Per quanto riguarda le interfacce RTL dei transattori, invece, ogni dato che deve essere trasmesso dal controllore verso lo xtea, viene mappato dal transattore sulle rispettive porte RTL come un segnale. Rimane quindi da definire come realizzare l'interconnessione tra il mondo continuo, ed il mondo discreto. Questo é stato reso possibile grazie all'utilizzo di particolari porte messe a disposizione dallo standard AMS. É stata quindi introdotta un interfaccia RTL-AMS con le seguenti porte di ingresso/uscita per permettere la comunicazione tra lo xtea e la valvola:

```
1 sca_tdf::sca_out< int > c_out;
2 sca_tdf::sca_out< double > a_t_out;
3 sca_tdf::sca_de::sca_in< int > c_in;
4 sca_tdf::sca_de::sca_in< uint32_t > a_t_in_0;
5 sca_tdf::sca_de::sca_in< uint32_t > a_t_in_1;
6 sca_tdf::sca_de::sca_in<sc_uint<1> >dout_rdy;
```

É da notare che le righe 4 e 5 rappresentano la dichiarazione delle porte d'ingresso per la mantissa e l'esponente della threshold passata in input dallo xtea come **uint32_t**, e riconvertita in **double** dall'interfaccia, con l'ausilio di una union, prima di inviare il dato al serbatoio.

Infine al modulo LSF che descrive il serbatoio, é stata aggiunta la seguente porta di uscita:

```
sca_lsf::sca_de::sca_sink lsf2rtl;
```

la quale puó ora vantare della possibilitá di comunicare con moduli esterni a tempo discreto.

IV. RISULTATI

In questa sezione vengono riportati i risultati delle varie simulazioni effettuate per una durata complessiva di 1000s ciascuna.

I dati mostrano che il sistema AMS riscontra un comportamento atteso: il livello dell'acqua inizia a salire in maniera progressiva per poi stabilizzarsi intorno ai 250s ad un livello pari a 5.41 circa.

Successivamente la simulazione della piattaforma eterogenea realizzata, ha prodotto i seguenti risultati:

i quali, essendo molto simili alla simulazione del sistema AMS, possiamo affermare che il design si comporta correttamente dal punto di vista della simulazione.

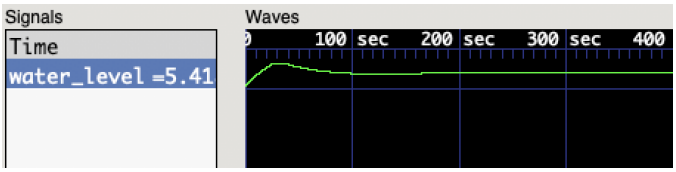

Figura 7: AMS - water level wave

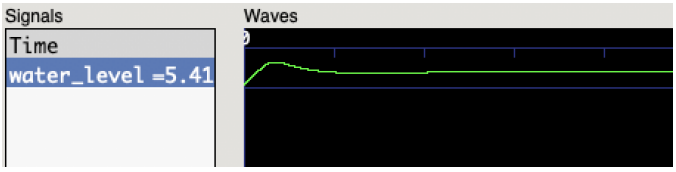

Figura 8: Watertank system - water level wave

Per ottenere questi risultati, un ruolo fondamentale lo giocano i tempi di: moduli AMS e relative porte, componenti TLM e il tempo di simulazione del sistema. É da notare quindi che, diminuendo sempre di piú il tempo di funzionamento dei moduli AMS, aggiustando di conseguenza i tempi di tutti gli altri moduli, e sfruttando ulteriori tecniche di ottimizzazione (e.g. l'utilizzo di tutti i campioni descritti nella sezione III-C), é possibile ottenere un sistema ancora piú reattivo.

V. Conclusioni

Grazie allo sviluppo di questo progetto é ora ben chiaro quale sia il flusso di progettazione e la complessitá di una piattaforma hardware che comprende elementi di natura eterogenea. Inoltre, risulta evidente come SystemC ci fornisca strumenti molto utili per riuscire a progettare un design partendo da una definizione ad alto livello dello stesso, per poi basarsi sul riuso dei componenti e raffinare via via sempre di piú il modello fino ad arrivare ad un'implementazione RTL sintetizzabile.

Riferimenti bibliografici

[1] Accellera Systems Initiative *et al.*, "Systemc," *Online, December*, 2013.
[2] ——, "Asi systemc amswg, standard systemc ams extensions language reference manual," *Online, March*, 2013.
[3] Roger M. Needham, David J. Wheeler, "Computer laboratory, university of cambridge (technical report)," *Online, October*, 1997.
[4] John Aynsley, Doulos *et al.*, "Osci tlm-2.0 language reference version ja32," *Online, July*, 2009.

Appendice

Di seguito sono riportate alcune le figure citate nel report:

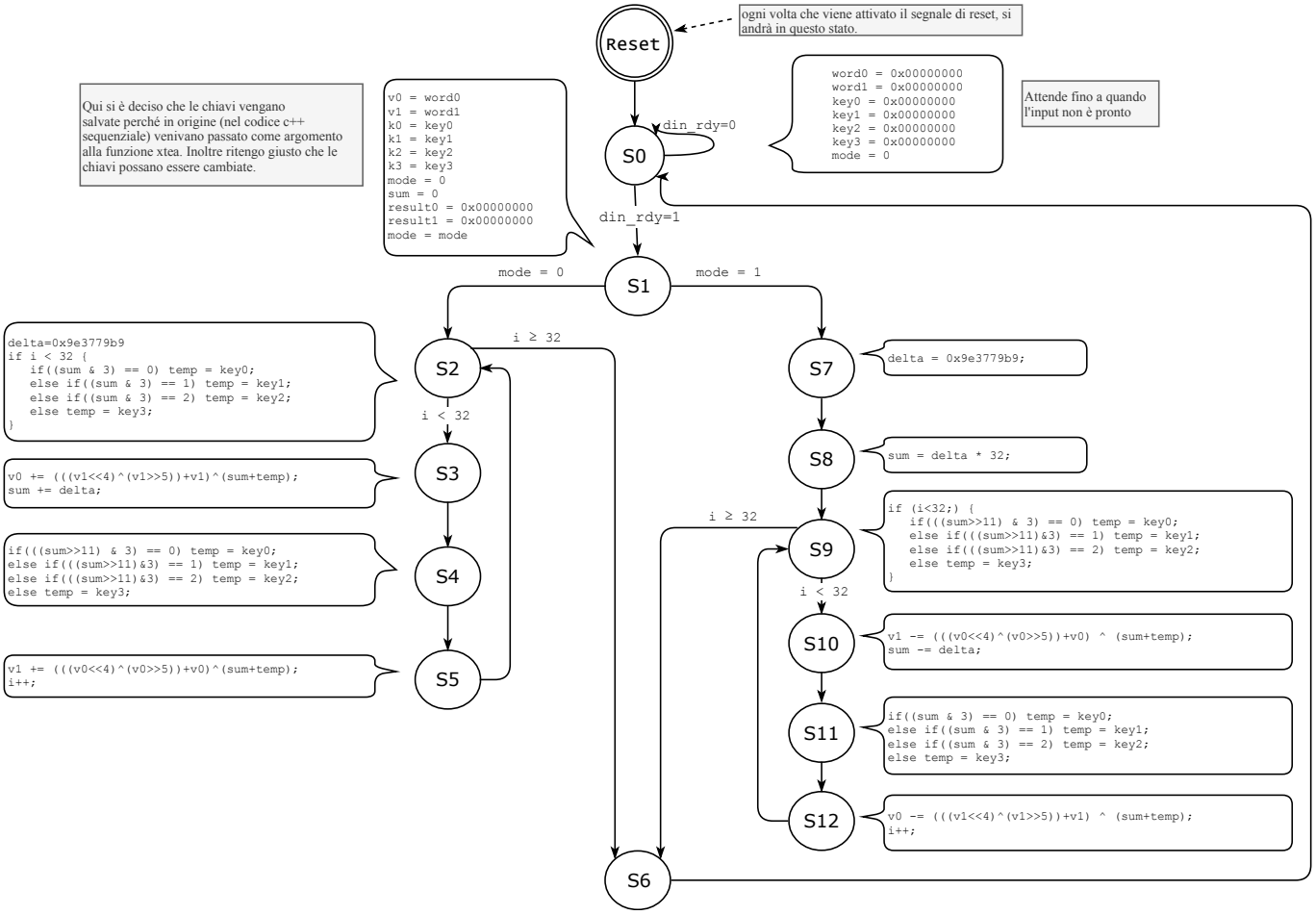

Figura 9: Extended finite state machine of XTEA

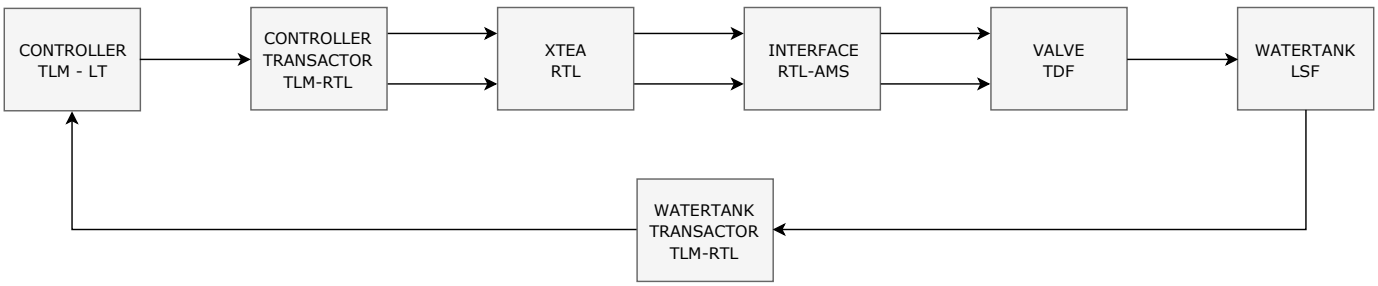

Figura 10: Heterogeneous Watertank System