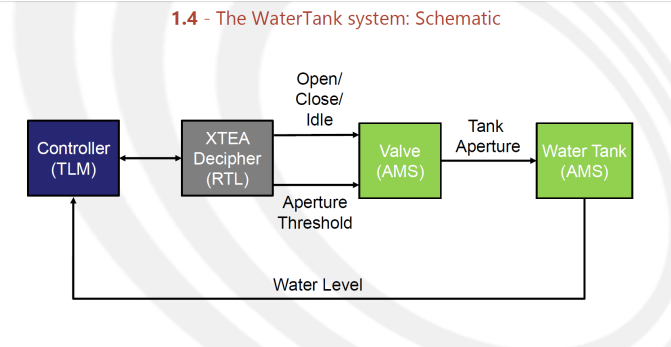
Embedded System Design - First Assignment

Fabio Fiorio - VR422016

***Sommario*—Il primo elaborato di progettazione sistemi embedded è composto da 4 parti: RTL design, TLM design, AMS design e l'Heterogeneous Platform ovvero l'unione delle tre parti precedenti.**
L'obiettivo del progetto è controllare una valvola affinchè il livello di un serbatoio rimanga tra 5 e 8.8, tutto ciò attraverso un controllore che passa il comando e la threshold criptate con l'algoritmo XTEA ad un decifratore, infine il serbatoio riporta il valore del livello dell'acqua al controllore.
Il progetto è stato sviluppato in SystemC, sfruttando le sue estensioni quali TLM ed AMS.

I. Introduzione

Il progetto ha l'obiettivo della realizzazione di un sistema di controllo di una valvola che gestisce l'acqua presente in un serbatoio. Scambiando i dati tra controllore TLM e decifratore RTL criptati con l'algoritmo XTEA. L'algoritmo XTEA è un algoritmo di criptazione con una struttura semplice e la compattezza del codice quindi ideale in molte situazioni dove si hanno vincoli estremi legati alle risorse, ad esempio nei vecchi sistemi hardware dove la quantità di memoria è spesso minima.



Schema

Il sistema è composto dai seguenti componenti:

- Controllore in TLM, si occupa di cifrare il comando open/close/idle e il valore della treshold e di mandare i dati al decifratore, questa modellazione ad alto livello di astrazione è stata fatta in 3 diverse versioni, UT ovvero un modello senza timing, quantunm keeping e temporal decoupling, LT cioè viene introdotto il tempo ed il temporal decoupling, AT4 indica che le operazioni sono suddivise in 4 fasi cioè inizio/fine richiesta e inizio/fine risposta.
- Decifratore RTL, è quel componente che decifra il comando e il valore della valvola e manda i valori alla valvola, strutturato ad FSM (macchina a stati finiti) e Datapath). L'FSM è rappresentato da stati e transizioni, il datapath invece dai registri, multiplexer ed operatori.
- Valvola AMS, è l'attuatore che calcola il valore dell'apertura da mandare al serbatoio, la modellazione AMS è un tipo di modellazione omogenea che ha sia parti digitali che analogiche
- Serbatoio AMS, si occupa di calcolare il livello dell'acqua e di passarlo al controllore chiudendo così il cerchio tra i componenti.

Per implementare il tutto si è partiti dalla modellazione RTL del testbench e della componente digitale, creando un cifratore-decifratore con l'algoritmo XTEA, testando così il corretto funzionamento dell'ESFM.
Successivamente si è passati ad implementare in TLM in versione UT del cifratore -decifratore, quindi non considerando il tempo. Poi si è implementato a livello LT, aggiungendo quinid il tempo. In fine si è passati ad implementare il modulo TLM a livello AT4, cioè utilizzando 4 fasi.
La parte composta da controllore, attuattore e impianto fisico, ovvero controllore, valvola e serbatoio è stata implementata in AMS, realizzato separatamente per verificare l'integrità dei dati e la correttezza dello scambio dei dati.
Testati e verificati i risultati, si è unito il tutto in un sistema eterogeneo composto quindi da TLM, RTL ed AMS, inoltre sono stati aggiunti dei transattori e delle interfacce per collegare i vari componenti tra loro.

II. Background

Per la realizzazione del progetto è stato utilizzato il linguaggio SystemC [1], il quale è un'estensione della libreria di C++ tale da permettere un'efficente descrizione e progettazione del sistema hardware/software assegnato.
La parte Hardware del sistema è stata raffinata fino al livello più accurato del SystemC, ovvero l'RTL: Register Transfer Level, con cui è stato implementato un ESFM (macchina a stati finiti, FSM e Datapath) per il calcolo dell'XTEA.
Invece per quanto riguarda la parte Software sono state utilizzate le seguenti estensioni del linguaggio:

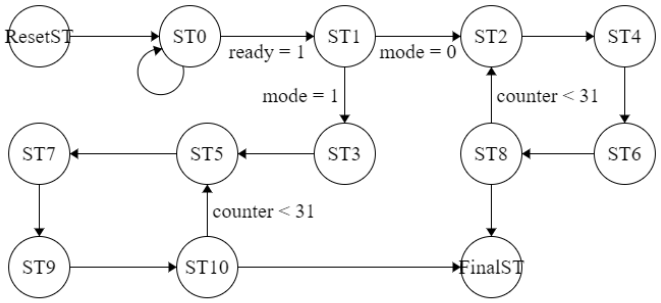
- TLM: Transaction Level Modeling, fondamentale per la comunicazione dei moduli software e hardware e l'utilizzo dei Transactor, i quali servono per la comunicazione a diversi livelli di astrazione. (TLM con RTL e AMS con TLM)
- AMS: il quale offre diversi modelli per la realizzazione di sistemi a tempo continuo, discreto, conservativo, non conservativo. L'elaborato si è concentrato sul modello LSF (Linear Signal Flow) e TDF (Time Data Flow) per la progettazione del controllore, valvola e serbatoio.

III. Metodologia applicata

Analizziamo punto per punto i componenti che sono stati implementati e come sia stato effettuato il binding di tutti i moduli per la realizzazione del progetto finale, vedendo in dettaglio i passi per arrivare al progetto finito.

A. RTL

Per prima cosa è stato sviluppato l'algoritmo XTEA in RTL, sia la parte che cifra che quella che decifra. Quindi è stato necessario la costruzione di un FSM e del relativo Datapath come mostrato in figura.



EFSM

Lo stato ResetST è dove vengono inizializzate le porte dei dati in input e del risultato.
Nello stato ST0 viene inizializzato il valore del "delta" necessario per l'algoritmo XTEA, e vengono inizializzate le variabili temporanee utilizzate. Nel quale resto in attesa del segnale "ready".
Quindi passo nello stato ST1 dove vengono letti i due dati da criptare.
Successivamente passo nello stato ST2 o ST3 in base alla modalità criptazione o decriptazione, in questi stati si inizia con l'algoritmo XTEA.

```
case ST_2:
if((sum.read() & 3) == 0)
    temp0.write(key0.read());
else if((sum.read() & 3) == 1)
    temp0.write(key1.read());
else if((sum.read() & 3) == 2)
    temp0.write(key2.read());
else temp0.write(key3.read());
break;
```

```
case ST_4:
v0.write(v0.read() + (((((v1.read() << 4)
^ (v1.read() >> 5)) + v1.read()) ^
(sum.read() + temp0.read()))));
sum.write(sum.read() + delta.read());
break;
```

```
case ST_6:
if(((sum.read() >> 11) & 3) == 0)
    temp0.write(key0.read());
else if(((sum.read() >> 11) & 3) == 1)
    temp0.write(key1.read());
else if(((sum.read() >> 11) & 3) == 2)
    temp0.write(key2.read());
else temp0.write(key3.read());
break;
```

```
case ST_8:
v1.write(v1.read() + (((((v0.read() << 4) ^
(v0.read() >> 5)) + v0.read()) ^
(sum.read() + temp0.read()))));
if(counter.read() < 31){
    counter.write(counter.read() + 1);
}
break;
```

Quando si termina l'algoritmo e si arriva nello stato FinalST si procede a scrivere i risultati e imposto "output rdy" a 1 per segnalare la terminazione.

B. TLM

Successivamente è stato modellato il cifratore-decifratore XTEA a livello TLM a tre differenti livelli. L'idea che sta alla base è la nozione di transazione rappresentata da un oggetto "payload". Il Transaction Level Modeling prevede l'utilizzo di due moduli, l'Initiator che inizializza la transazione, ovvero crea il payload e richiama le funzioni del target per inviarlo. Il Target che elabora il payload.
Passando da un livello all'altro la nozione di tempo acquisisce significato, partendo dall'UT dove il tempo non è presente, passando dall'LT si acquisisce la nozione di tempo ed infine con AT4 si suddividono le operazioni in 4 fasi, quindi il tempo è ancora più preciso. Tutte le implementazioni utilizzano il concetto di payload come pacchetto, la sua definizione è la seguente;

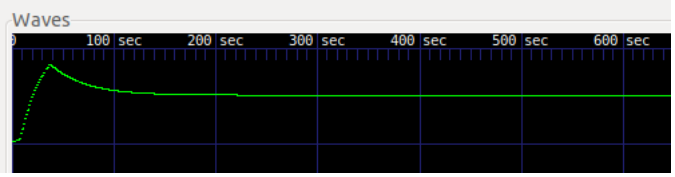
```
struct iostruct {
    sc_uint<32> datain_word1;
    sc_uint<32> datain_word2;
    sc_uint<32> datain_key0;
    sc_uint<32> datain_key1;
    sc_uint<32> datain_key2;
    sc_uint<32> datain_key3;
    sc_uint<32> result0;
    sc_uint<32> result1;
    sc_logic mode;
};
```

Il codice a livello UT è implementato dal modulo xtea_UT, nella funzione xtea_function(). l'initiator è il modello che inizia la transazione e comunica con il target attraverso il metodo nb_transport_fw. Il target è il modulo che riceve i dati ed elabora il payload. In TLM esistono 2 percorsi, il "forward path" e il "backward path", dove il primo indica il percorso che viene chiamato dall'initiator in direzione del target, mentre il secondo è il percorso che viene chiamato dal target per tornare all'initiator. Il meccanismo è blocking ovvero l'initiator completa l'esecuzione di una funzione assieme al target e non attraverso una successioni di operazioni tra initiator e target che è caratteristico di AT4. Si può notare che a livello UT la nozione di tempo è assente, ed è quindi il simulatore che gestisce il tempo. Per questo tra tutte le simulazioni questa è quella più veloce.
IL livello Loosely Timed prevede l'introduzione di nozione di tempo, la procedura è medesima a quella dell'UT, ovvero l'initiator (testbench) inizializza il payload e richiama il target (modulo xtea) per inviare il payload. Qui il meccanismo è blocking ovvero l'initiator completa l'esecuzione di una funzione assieme al target e non attraverso una succesione di operazioni tra initiator e target, com'è caratteristico di AT4. Il livello AT4 prevede che la nozione di tempo sia completamente specificata, questo permette di aggiungere dettagli ma si perde velocità, infatti dopo l'RTL è il meccanismo più lento. Questo livello deve effettuare uno scambio in 4 fasi,

Begin request, L'initiator () comunica al modulo (xtea) che intende iniziare una nuova richiesta e quindi gli vuole fornire i dati. End request, Il target una volta letta la richiesta e i dati inviati comunica all'initiator che ha ricevuto correttamente la richiesta e quindi inizia l'elaborazione. Begin response, Il target a seguito della computazione avverte l'initiator che intende ritornare il risultato. End resonse, L'initiator una volta letta la risposta conferma il target della ricevuta e quindi una computazione termina.

C. AMS

Infine si è implementato il sistema "WaterTank" suddiviso tra valvola e serbatoio, dove la prima ha lo scopo di riempire il serbatoio, il quale tiene traccia del livello dell'acqua, spetta quindi ad un controllore gestire il comando di open/close/idle e mandare il livello della treshold alla valvola. Si è progettato il modulo del serbatoio in LSF Linear Signal Flow, quindi a tempo continuo non conservativo. Si è implementato utilizzando LSF poiché si vuole descrivere un'equazione differenziale e l'unico modo per realizzarlo é utilizzando il mondo continuo, l'equazione che si vuole realizzare è x' = 0.6 * a - 0.03 * x. Lo schema a blocchi prende in ingresso il valore di a al tempo t e dopo aver sommato l'ingresso derivante dalla valvola calcola l'uscita x ad ogni istante di tempo. L'ingresso derivante dalla valvola é discreto TDF quindi si è trasformato per renderlo continuo come il modulo LSF. Successivamente il segnale di uscita si converte in TDF dato che il controllore è progettato in TDF. Si è progettato il modulo del serbatoio dell'acqua in TDF Timed Data Flow, quindi a tempo discreto non conservativo. E' stato scelto questo poiché da specifiche si richiedono due metodi differenti e il LSF lo si era già utilizzato per l'implementazione del serbatoio e il ELN viene usato solamente per le reti elettriche elettriche lineari, e non è il nostro caso, quindi si è implementato con TDF. La realizzazione del modulo viene fatta nel file valvola.cc: si è realizzata una MSF implementata attraverso un switch case, lo stato iniziale è IDLE, in ingresso riceve il valore di in_flag e in_treshold e in uscita da il valore di a al modulo serbatoio. Il valore di a può oscillare solamente tra il valore di treshold e 0. Il modulo controller è realizzato in TDF Discrete-time non-conservative quindi è più lento dell'impianto quindi si producono più dati di quelli che il controllore riesce ad analizzare, quindi alcuni campioni che riceve li ignora. Il suo compito è verificare il livello dell'acqua nella valvola, invia il segnale di flag e di treshold e in base al valore calcolato regola la valvola, inviando gli opportuni segnali.



GTKwave

D. Heterogeneous Platform

In fine è stato unito il tutto in unico elaborato riutilizzando le parti create in precedenza, aggiungendo dei transatori ed un interface per far comunicare le diverse parti. Il controllore è stato sviluppato in TLM riutilizzando la parte di cifratura già creata in precedenza, questo componente è necessario per verificare il livello dell'acqua e mandare il comando e il valore della treshold cifrati al decifratore, qui è stato aggiunto un transattore per lo scambio di dati tra TLM e RTL con la seguente struttura:

```
sc_out<sc_uint<1> > input_rdy;
sc_out<sc_uint<32> > word1;
sc_out<sc_uint<32> > word2;
sc_out<sc_uint<32> > key0;
sc_out<sc_uint<32> > key1;
sc_out<sc_uint<32> > key2;
sc_out<sc_uint<32> > key3;
sc_out<sc_uint<1> > mode;
sc_in<sc_uint<32> > result0;
sc_in<sc_uint<32> > result1;
sc_in<sc_uint<1> > output_rdy;
sc_out<bool> rst;
sc_in<bool> clk;
```

Per quanto riguarda la parte RTL è stato riutilizzato l'ESFM del decifratore già sviluppato in precedenza, con l'aggiunta di un interface per passare i dati decifrati alla valvola, convertendo così i segnali digitali in quelli analogici:

```
sca_tdf::sca_de::sca_in <sc_uint<32> >
flag_controller;
sca_tdf::sca_de::sca_in <sc_uint<32> >
threshold_controller;
sca_tdf::sca_out <double> flag_valvola;
sca_tdf::sca_out <double> threshold_valvola;
```

Per la valvola ed il serbatoio sono stati riutilizzati i componenti già sviluppati in AMS, però per far leggere il livello dell'acqua dal controllore è stato aggiunto un transactor.

```
sc_in< double > livello_acqua_in;
```

Così si conclude l'heterogeneous platform. E' stato utilizzato il main.c e il trace.hh per allocare tutti i componenti ed effettuare il binding delle porte, inoltre viene creato il grafico per raccogliere il valore dell'acqua.

!!!!!!!!!! IMMAGINE GTK WAVE !!!!!!!!!!

E. Organizzazione dell'implementazione

La struttura delle cartelle è organizzata così: nell'HW_Subsistem ci sono tutti i raffinamenti a livello RTL e TLM nelle 3 versioni UT, LT e AT4; La cartella Continous_Subsistem invece ha il progetto sviluppato in AMS; Infine nell'Heterogeneous_Platform è presente l'elaborato tutto unito con TLM, RTL e AMS con l'aggiunta dei transattori e dell'interfaccia.

IV. Risultati

Per descrivere i risultati ottenuti si è scelto di analizzare il tempo di simulazione in relazione al raffinamento:

	UT	LT	AT4	RTL
Real Time	0.002s	0.002s	0.002s	0.019s
User Time	0.002s	0.000s	0.002s	0.003s
System Time	0.002s	0.002s	0.002s	0.000s

Il real time indica il tempo dall'inizio alla fine della chiamata. Questo è tutto il tempo trascorso, compresi gli intervalli di tempo utilizzati da altri processi e il tempo trascorso dal processo bloccato (ad esempio se è in attesa del completamento dell'I / O). L'User Time è la quantità di tempo della CPU speso nel codice in modalità utente (al di fuori del kernel) all'interno del processo. Questo è solo il tempo CPU effettivo utilizzato nell'esecuzione del processo. Altri processi e tempi di blocco bloccati non contano ai fini di questa cifra. Il System time indica la quantità di tempo della CPU speso nel kernel all'interno del processo. Ciò significa eseguire il tempo della CPU impiegato nelle chiamate di sistema all'interno del kernel, al contrario del codice della libreria, che è ancora in esecuzione nello spazio utente. Come "user", questo è solo il tempo CPU utilizzato dal processo.

Si può notare come il tempo aumenti sensibilmente nel momento in cui si sceglie l'implementazione in RTL. I risultati sono stati ottenuti con il comando "time".

Per provare il corretto funzionamento nelle parti separate dell'elaborato è stato utilizzato un testbench che prima criptava e poi decriptava lo stesso valore così se il risultato mostrato è uguale a quello di partenza il funzionamento è corretto, invece per quanto riguarda il modello della valvola e del serbatoio è stato controllato sia attraverso la simulazione che controllando i risultati su GTKwave.

V. Conclusioni

In conclusione si può dire che i risultati ottenuti rispecchiano le aspettative premeditate e mostrano, seppur in piccola parte, una buona robustezza del codice.

Riferimenti bibliografici

[1] Accellera Systems Initiative *et al.*, "Systemc," *Online, December*, 2013.