Un cifratore/decifratore XTEA VHDL sintetizzabile

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

***Sommario*—ModelSim [1] e Vivado rappresentano oggi degli strumenti essenziali per chi si occupa di progettazione, simulazione e sintesi di sistemi embedded, nonché sistemi ciberfisici. Il primo scopo di questo progetto é comprendere come, a partire da una descrizione di un modello HW in VHDL [2], sia possibile con l'ausilio di appositi tool di modellazione (nello specifico ModelSim e Vivado), simulare e sintetizzare tale modello su una specifica piattaforma HW. Il secondo obbiettivo riguarda invece un confronto con la sintesi ad alto livello HLS, ottenuta partendo da una descrizione ad alto livello, come ad esempio un'implementazione in C++.**

I. Introduzione

Questo report vuole riassumere il lavoro svolto nell'implementazione dell'algoritmo di cifratura/decifratura XTEA in VHDL sintetizzabile. Successivamente vengono mostrate delle simulazioni dello stesso, ottenute con ModeSim e Vivado fino ad arrivare ad ottenere un modello sintetizzato. Infine viene mostrato un confronto con lo stesso design che, a partire dalla sua versione in C++, viene invece sintetizzato passando attraverso la sintesi ad alto livello.

II. Background

L'importanza del VHDL deriva dal fatto che oltre alle applicazioni iniziali legate principalmente alla descrizione, modellizzazione e simulazione di circuiti digitali, esso é attualmente utilizzato soprattutto per la sintesi automatica dei circuiti. In pratica a partire da una descrizione VHDL del comportamento di un circuito é possibile ricavare automaticamente (utilizzando degli appositi programmi, chiamati "sintetizzatori logici") la sua implementazione in termini di porte logiche, o blocchi funzionali elementari. Questo approccio risulta molto vantaggioso da diversi punti di vista: in primo luogo permette di realizzare velocemente circuiti molto complessi ed ottimizzati a partire da una loro descrizione comportamentale. In secondo luogo il circuito descritto tramite VHDL piuttosto che con metodi tradizionali (schemi elettrici, funzioni logiche, etc.) risulta facilmente documentabile e verificabile. Inoltre la descrizione VHDL puó essere del tutto indipendente dalla tecnologia che si utilizzerá per implementare il circuito, questo rende possibile ad esempio implementare su FPGA circuiti descritti per ASIC o viceversa. Una descrizione VHDL composta essenzialmente da due parti:

- **Entity**: in cui si dichiarano gli ingressi e le uscite del circuito;
- **Architecture**: in cui viene descritto il funzionamento del circuito.

La descrizione del comportamento di un modulo mediante l'architecture puó essere descritta mediante diversi livelli di astrazione:

- **Behavioral**: permette una descrizione algoritmica o procedurale del comportamento di un modulo;
- **Structural**: specifica i componenti che costituiscono un modulo e le interconnessioni tra i componenti;
- **Data-flow**: rappresenta il sistema specificando come i segnali si propagano dai pin di ingresso a quelli di uscita;

Una volta implementato un modulo VHDL, é possibile simularlo attraverso un tool di simulazione (e.g. ModelSim, Vivado etc..) seguendo il diagramma di flusso illustrato in figura 1.

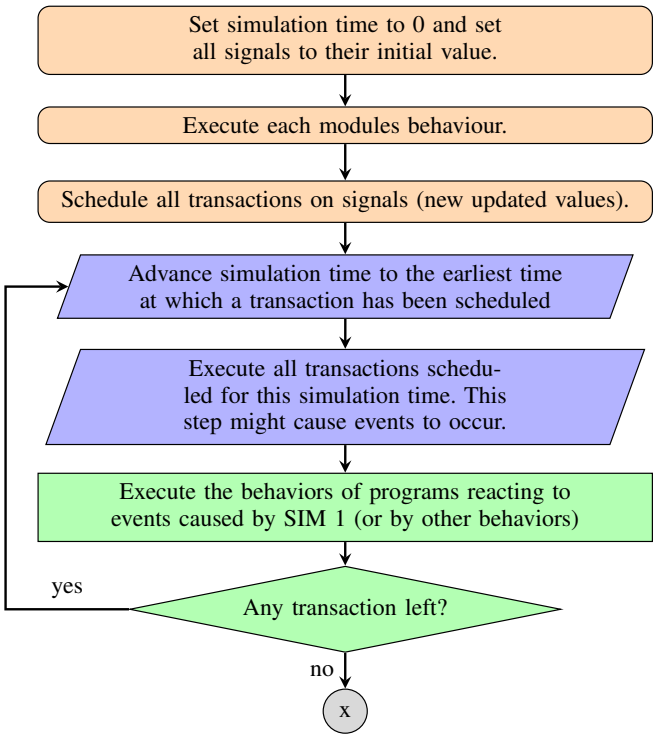

Figura 1: VHDL simulation "scheduler"

Per quanto riguarda la sintesi di un design, esistono diverse metodologie che consentono di arrivare ad ottenere un modello sintetizzato a partire da una sua descrizione ad alto livello (e.g. C++, SystemC, etc ...). L'utilitá nell'effettuare una sintesi ad alto livello risiede nel fatto di abbattere i tempi di sviluppo rispetto ai tempi richiesti con un flusso RTL ed aumentare quindi la produttivitá. Il riutilizzo di blocchi IP in

ambiente C/C++ non solo consente di trasportare facilmente il sottosistema IP da una famiglia all'altra, ma consente anche la riottimizzazione automatica della microarchitettura e dei blocchi RTL associati per i requisiti e per le caratteristiche su silicio dei sistemi di prossima generazione.

III. Metodologia applicata

Il primo passo era volto a comprendere meglio gli aspetti coinvolti durante la creazione di un FMU, che in questo caso era dedicato a svolgere la sola funzionalitá di guadagno di un segnale. In secondo luogo sono state fatte delle operazioni di completamento sul master PyFMI al fine di rendere il tutto simulabile in un unica virtual platform.

A. XTEA: VHDL

Facendo riuso della macchina a stati finiti estesa mostrata in figura 3, é stata effettuata una prima implementazione dello XTEA in VHDL seguendo quanto indicato dalle specifiche del linguaggio. Di seguito viene mostrata quella che é la descrizione della la entity dello XTEA:

```
entity xtea is
  port(
    clk, rst      : in bit;
    din_rdy       : in  bit;
    word0_in      : in  UNSIGNED(SIZE-1 DOWNTO 0);
    word1_in      : in  UNSIGNED(SIZE-1 DOWNTO 0);
    key0_in       : in  UNSIGNED(SIZE-1 DOWNTO 0);
    key1_in       : in  UNSIGNED(SIZE-1 DOWNTO 0);
    key2_in       : in  UNSIGNED(SIZE-1 DOWNTO 0);
    key3_in       : in  UNSIGNED(SIZE-1 DOWNTO 0);
    mode_in       : in  bit;
    result0_out   : out UNSIGNED(SIZE-1 DOWNTO 0);
    result1_out   : out UNSIGNED(SIZE-1 DOWNTO 0);
    dout_rdy      : out bit
  );
end xtea;
```

Mentre per quanto riguarda l'architecture é stato adottato uno stile architetturale di tipo behavioral contenente due processi concorrenti con le senguenti sensitivity list:

```
-- elaborate_XTEA_FSM
process(STATUS, din_rdy, mode_in, i)
```

```
-- elaborate_XTEA
process(clk, rst)
```

il primo implementa il controllore della macchina a stati finiti, mentre il secondo processo gestisce l'elaborazione.

B. XTEA: SIMULAZIONE

Per verificare il corretto comportamento del modulo cosí implementato, é stata effettuata una prima simulazione con ModelSim, utilizzando uno script "stimuli.do" in grado di stimolare il design fornendo alle porte di ingresso dei valori costanti. Una volta lanciata la simulazione, il tool ci permette di visualizzare il comportamento di ogni segnale tramite l'apposita sotto-finestra "wave", il cui risultato é riportato in figura 2.

É da notare che il risultato della cifratura (evidenziato in magenta) risulta visibile in uscita dopo 2630 nanosecondi dall'inizio della simulazione.

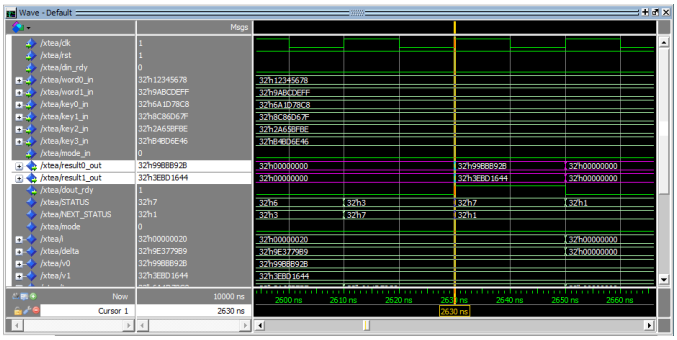

Figura 2: XTEA enchipering simulation

C. XTEA: SINTESI

Successivamente, con l'ausilio di Vivado, e prendendo come board di riferimento la PYNQ-Z1, sono stati effettuati tutti i passi previsti dal flusso per arrivare alla sintesi del design. Purtroppo, una volta arrivati alla fase di I/O plannig, in cui si richiedeva il placement delle porte del design sui pin messi a disposizione dalla board, ci si é accorti che il numero di porte da collegare (261) andava ben oltre il numero di pin disponibili sulla PYNQ. Di conseguenza é stato necessario modificare il design in modo da ridurre il numero di porte utilizzate,in particolare sono state rimosse: le porte di ingresso per la lettura delle chiavi e una porta di uscita per il risultato. Infine é stata modificata la macchina a stati finiti come mostrato in figura 4. É da notare che rimuovendo le porte di ingresso per le chiavi di cifratura, si é reso obbligatorio ricorrere alla condivisione delle porte di ingresso dedicate alla lettura delle parole da cifrare (sfruttando gli instanti di tempo successivi), per effettuare anche la lettura delle chiavi a due a due. Lo stesso principio é stato applicato per le porte di uscita: scrivendo inizialmente la prima parola cifrata/decifrata, e poi la seconda. Dopo ulteriori simulazioni, e ripercorrendo tutti i passi per arrivare alla sintesi, é stato prodotto un vero e proprio design potenzialmente sintetizzabile sulla PYNQ-Z1. I risultati ottenuti dalla sintesi e la sua successiva implementazione sono indicati nelle tabelle I e II rispettivamente.

Resource	Estimation	Available	Utilization %
LUT	493	53200	0.93
FF	327	106400	0.31
IO	101	125	80.80
BUFG	1	32	3.13

Tabella I: Post-Synthesis

Resource	Utilization	Available	Utilization %
LUT	450	53200	0.85
FF	327	106400	0.31
IO	101	125	80.80
BUFG	1	32	3.13

Tabella II: Post-Implementation

I dati riportati nelle tabelle confermano ancora una volta che non sarebbe stato possibile implementare il design di partenza senza effettuarne le opportune modifiche, in quanto il numero di I/O risulta ora essere, tutt'ora, al limite delle risorse messe a disposizione della board.

D. XTEA: HLS

Una volta effettuata la sintesi seguendo un flusso RTL, é stato effettuata una nuova sintesi a partire da un implementazione in C++ dello XTEA. In particolare, con l'ausilio della suite Vivado HLS, é stato dapprima creato un nuovo progetto, quindi é stato importato il sorgente C++ e infine ancora un volta é stata selezionata come board target, la PYNQ-Z1 in modo che Vivado possa configurare le librerie necessarie da utilizzare per la sintesi. Una volta lanciato il comando per la sintesi, vengono generati cos i rispettivi modelli RTL (VHDL, Verilog, ecc ...). La tabella III riporta una stima dell'utilizzo delle risorse necessarie per la vera e propria sintesi su sull'FPGA.

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1478
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	117
Register	-	-	217	-
Total	0	0	217	1595
Available	280	220	106400	53200
Utilization (%)	0	0	˜0	2

Tabella III: C++ High-level synthesis

Con i modelli cosí generati, é stato effettuato un tentativo di sintesi e sua successiva implementazione, che peró a causa dell'eccessivo numero di porte e quindi l'impossibilitá di effettuare l'operazione di placement, quest'ultima non é andata a buon fine. Nonostante tutto é stato comunque possibile effettuare un analisi post sintesi i cui risultati sono riportati nella tabella IV

Resource	Estimation	Available	Utilization %
LUT	698	53200	1.31
FF	206	106400	0.19
IO	265	125	212.00
BUFG	1	32	3.13

Tabella IV: Post-Synthesis

Notiamo che la stima delle risorse richieste é chiaramente superiore a quella ottenuta dalla sintesi partendo da un design RTL implementato manualmente (vedi tabella I). Questo perché con un implementazione manuale si ha un maggiore controllo sull'utilizzo delle risorse e quindi generalmente risulta piú ottimizzato rispetto ad una soluzione automatica. Una possibile soluzione a questo problema si potrebbe ottenere modificando opportunamente il codice C++, oppure agire direttamente sul design RTL generato dalla sintesi ad alto livello, con lo scopo di ridurre il piú possibile in numero di porte del design stesso.

IV. Conclusioni

Grazie allo sviluppo di questo progetto é ora ben chiaro come effettuare una simulazione utilizzando strumenti professionali come ModelSim e Vivado, e quale sia il flusso da seguire per arrivare ad ottenere un modello sintetizzato per una specifica FPGA.

Inoltre, risulta evidente come la sintesi ad alto livello ci fornisca soluzione molto utile per riuscire ad avere un prototipo di quello che sará poi il design finale, specifico per una certa board, prima ancora di effettuarne una vera e propria implementazione RTL, consentendoci cosí di verificarne la funzionalitá con largo anticipo rispetto all'intero flusso di modellazione.

Riferimenti bibliografici

[1] M. G. Corporation, "Modelsim pe student edition," *XXX*, XXX.
[2] Accellera Systems Initiative *et al.*, "1076-2008 - ieee standard vhdl language reference manual," *Online, January*, 2009.

Appendice

Di seguito sono riportate alcune le figure citate nel report:

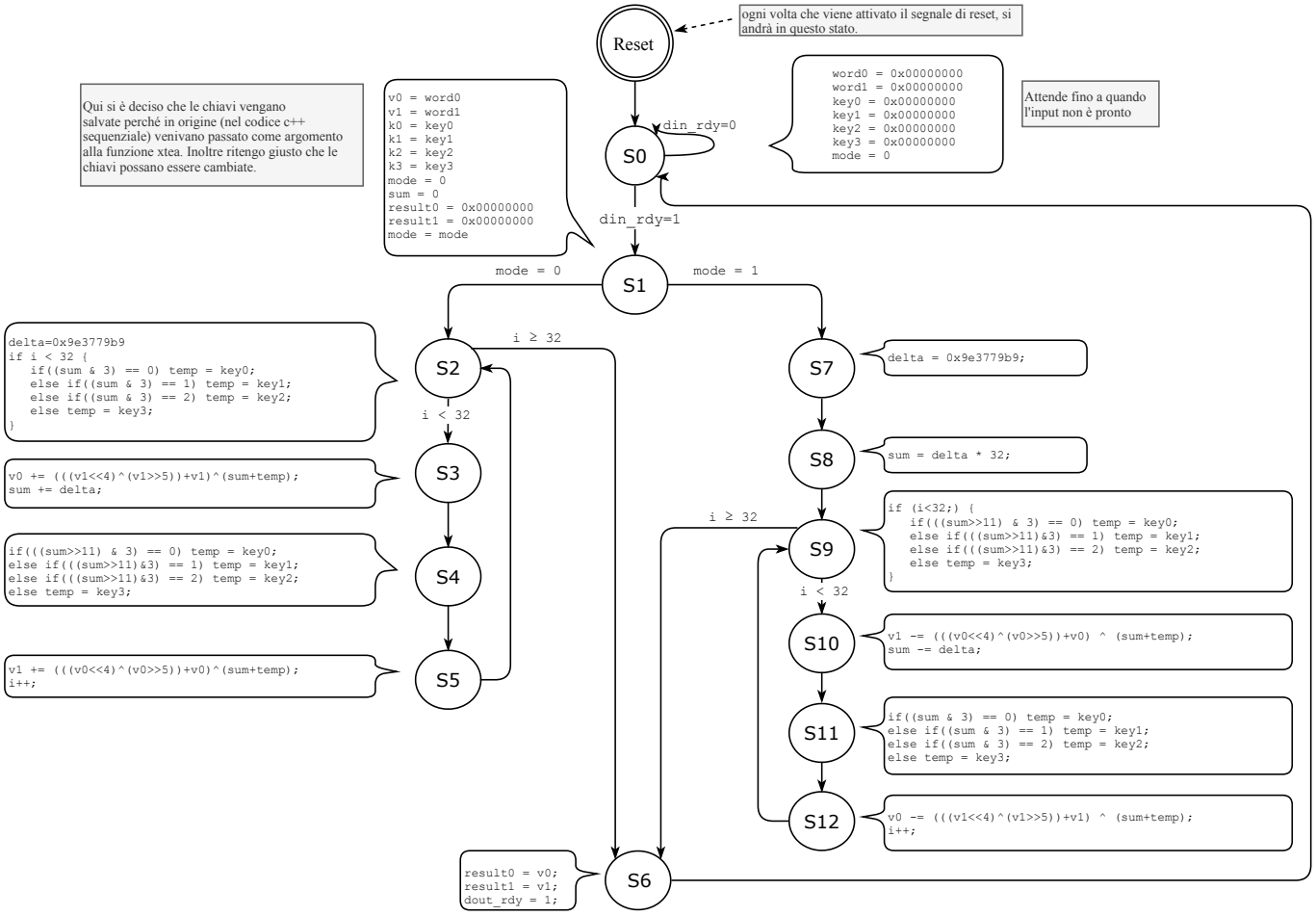

Figura 3: Extended finite state machine of XTEA

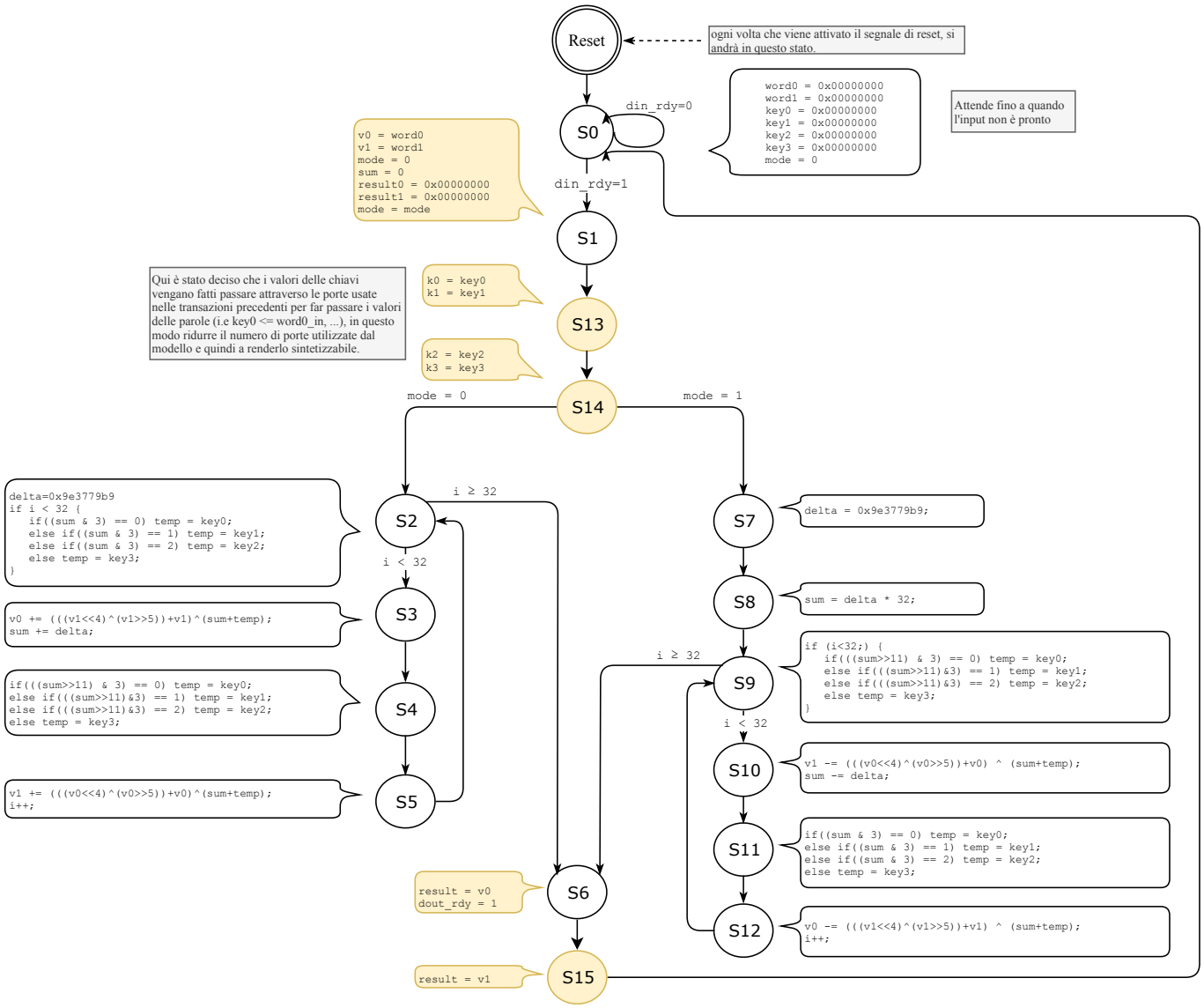

Figura 4: Synthetizable extended finite state machine of XTEA

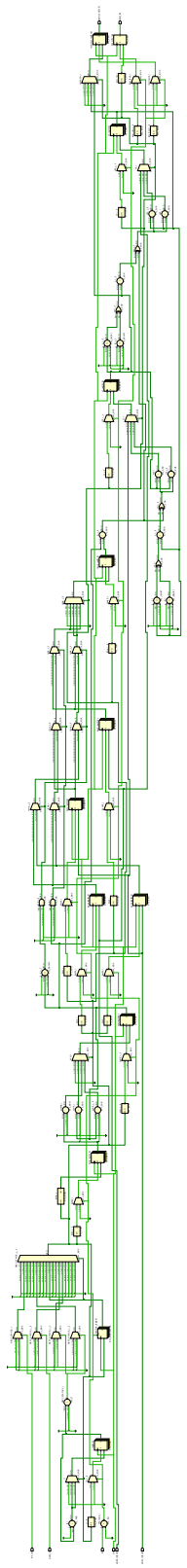

Figura 5: Netlist schematic after synthesis