Modellazione in VHDL del sistema crittografico XTEA

Mirko Albanese - VR431583

***Sommario*—Questo documento descrive il lavoro svolto durante la seconda parte del progetto. Il suo obiettivo è illustrare le scelte progettuale effettuate durante lo sviluppo del modulo XTEA nel linguaggio VHDL.**

I. Introduzione

Il progetto consiste nello sviluppare un cifrario il cui funzionamento è basato sull'algoritmo wXtended TEA fornito.

- Il cifrario dovrà essere implementato come modulo VHDL, testato automaticamente tramite uno script `stimuli.do` per il software proprietario *Modelsim*.
- Il modulo deve essere importato in *Vivavo*, simulato tramite un testbench e sintetizzato per la piattaforma `Xilinx PYNQ`.
- I sorgenti SystemC del cifrario dovranno essere sintetizzati tramite `Vivado HLS`, ed il risultato della sintesi confrontato con quello del modulo VHDL.

II. Architettura del Modulo

Il modulo cifratore presenta gli ingressi e le uscite visibili in Figura 1.

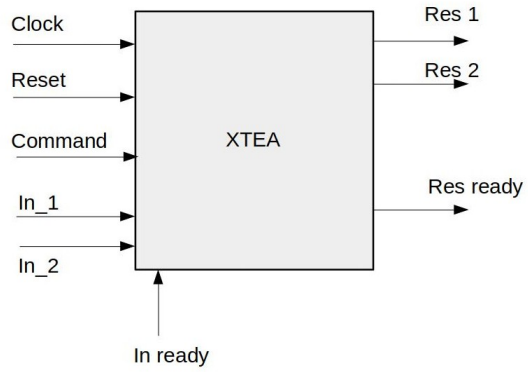


Figura 1. Interfaccia XTEA

In input abbiamo:

- Clock: Segnale di sincronia;
- Reset: Quando alto resetta il modulo;
- Input ready;
- Command: Operazioni da seguire;
- In1: Primo operando;
- In2: Secondo operando.

In output abbiamo:

- Res-ready: Quando alto segnala la completa operazione;
- Res1: Primo risultato;
- Res2: Secondo risultato;

Il modulo per poter criptare o decriptare ha bisogno in input di 6 parole a 32 bit: quattro appartenenti alla chiave di cifratura, e due appartenenti al messaggio. Abbiamo quindi bisogno di una prima fase di configurazione, nel quale impostiamo la chiave. Successivamente possiamo passare il messaggio.

III. Implementazione

Il modulo è implementato nel file xtea.vhd. La sua struttura è quella di una FSMD, in figura 2.

IV. Simulazione con Modelsim

Per poter simulare il progetto con il software *Mentor Modelsim* creiamo un nuovo progetto all'interno del quale importiamo il file `xtea.vhd`. Dopodichè nel file `stimuli.do` viene automatizzata la parte di setup e di test del modulo. Il software permette di salvare le forme d'onda nel fornito proprietario `wlf`. Una volta caricati i file nel progetto `Modelsim` viene caricato il modello e eseguito lo script di simulazione mediante i comandi:

```
#carica il modello in sessione
$vsim work.xtea
#lancia lo script di simulazione
$do stimuli.do
```

Di seguito viene visualizzata una parte del sorgente `stimuli.do`

```
echo "Apertura file stimuli.do"

# fai ripartire la simulazione (se non lo fai,
# continua dal punto nel quale si era fermato)

vsim work.xtea

restart -f

# tolgo e ri-aggiungo i segnali del modulo alla sc
add wave *

# RESET DEL MODULO
# il clock parte con 0, resta basso 1ns, viene alz
# resta alto 1ns, e ripeto il processo ogni 2ns
force clock 0 0 ns, 1 1 ns -r 2ns
force reset 1 0 ns, 0 2 ns
run 5ns

# CONFIGURAZIONE _ 1 Input keys
```

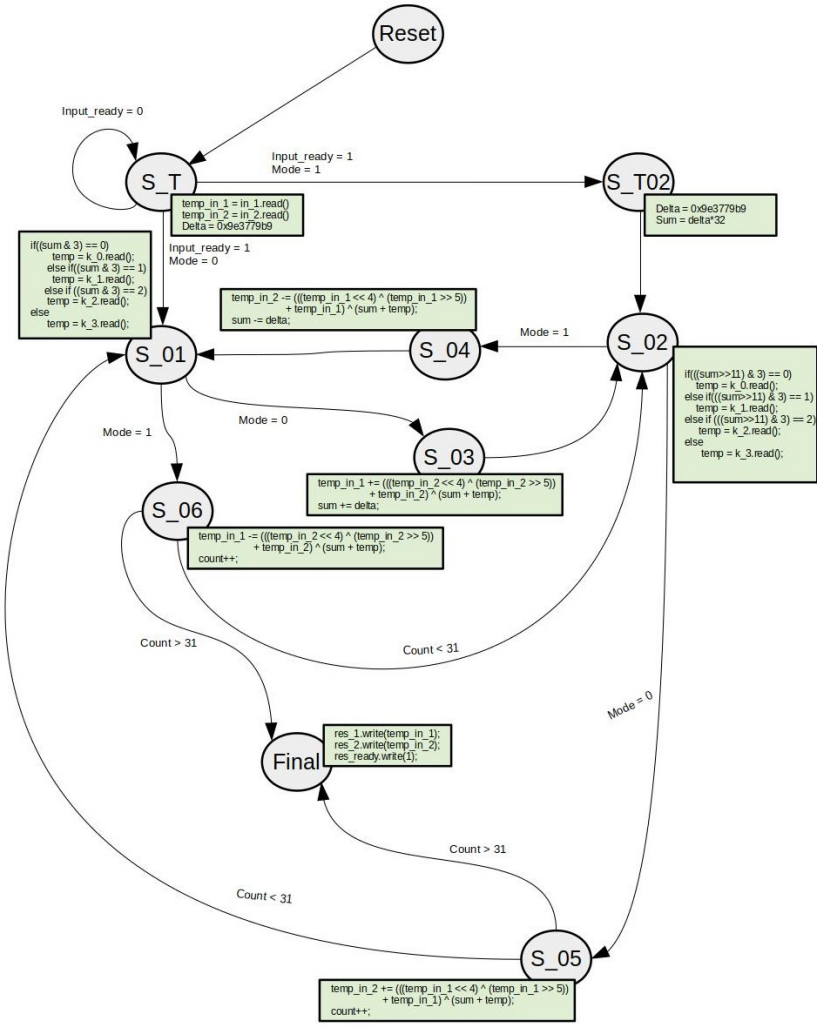


Figura 2. Diagramma degli stati. Le transizioni sono comandate dalla FSM. In verde sono presenti le elaborazioni fatte dal datapath.

```
force in1 16#6a1d78c8
force in2 16#8c86d67f
force input_ready 1
run 2 ns
force input_ready 0
run 4 ns

# CONFIGURAZIONE _ 2 Input keys
force in1 16#2a65bfbe
force in2 16#b4bd6e46
force input_ready 1
.
.
```

Le figure 6 e 7 illustrano le forme d'onda prodotte dal software `Modelsim`, si noti che gli input in alto a sinistra della figura 6 corrispondono ai valori di output posti in basso a destra della figura 7.

V. Sintesi

Per verificare che il modulo progettato fosse sintetizzabile è stato necessario utilizzare il tool `Xilink Vivavo`. Una volta impostata la piattaforma PYNQ `xc7z020c1g400-1` è stato possibile lanciare la sintesi, di cui ne viene riportato il risultato. Dai risultati raccolti nella schermata acquisita in figura 3 si può notar che la scelta dell'interfaccia definita richieda un utilizzo di 133 porte I/O su 125 porte della board, risultando così non implementabile sulla FPGA scelta.

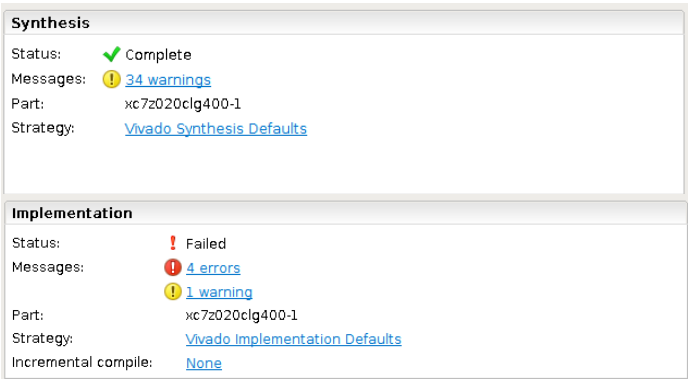
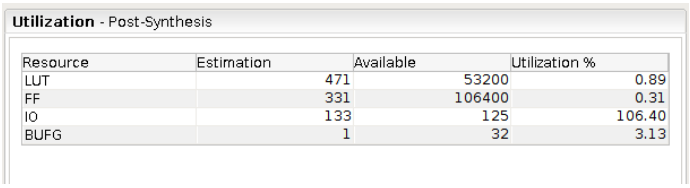


Figura 3. Vivado, risultato sintesi e implementazione

In figura 4 si illustra il risultato della sintesi.



Utilization - Post-Synthesis

Resource	Estimation	Available	Utilization %
LUT	471	53200	0.89
FF	331	106400	0.31
IO	133	125	106.40
BUFG	1	32	3.13

Figura 4. Vivado, risultato sintesi e implementazione

VI. Sintesi ad Alto Livello

Per confrontare il codice sviluppato con uno generato automaticamente grazie alla sintesi ad alto livello, è stato necessario utilizzare il tool `Xilink Vivado HLS` già citato nelle sezioni sopra. Questo tool ha permesso di generare automaticamente codice VHDL a partire da un sorgente in C++ in cui l'algoritmo veniva descritto ad un alto livello. Il risultato della sintesi è visibile in figura 5. Questa sintesi soffre del problema relativo al sovrautilizzo delle porte di I/O, risultando non implementabile direttamente sulla PYNQ.d

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1282
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	280
Register	-	-	628	-
Total	0	0	628	1562
Available	280	220	106400	53200
Utilization (%)	0	0	~0	2

Figura 5. Vivado HLS, risultato hls

VII. Conclusioni

Il confronto tra la versione del modulo XTEA sviluppato in VHDL e la versione derivante da una sintesi ad alto livello ha permesso di capire un codice automatico generalmente si presenta essere più complesso, e sopratutto molto più oneroso dal punto di vista dell'utilizzo delle risorse hardware. Per questo motivo si preferisce utilizzare una strategia non automatizzata partendo da un modello ad alto livello C++/SystemC, ad una versione più dettagliata in VHDL/Verilog, verificando la correttezza del prodotto con simulazioni e/o test, almeno fino a quando tecniche di *high level synthesis* non permettano di raggiungere la versatilità di un codice costruito a mano.

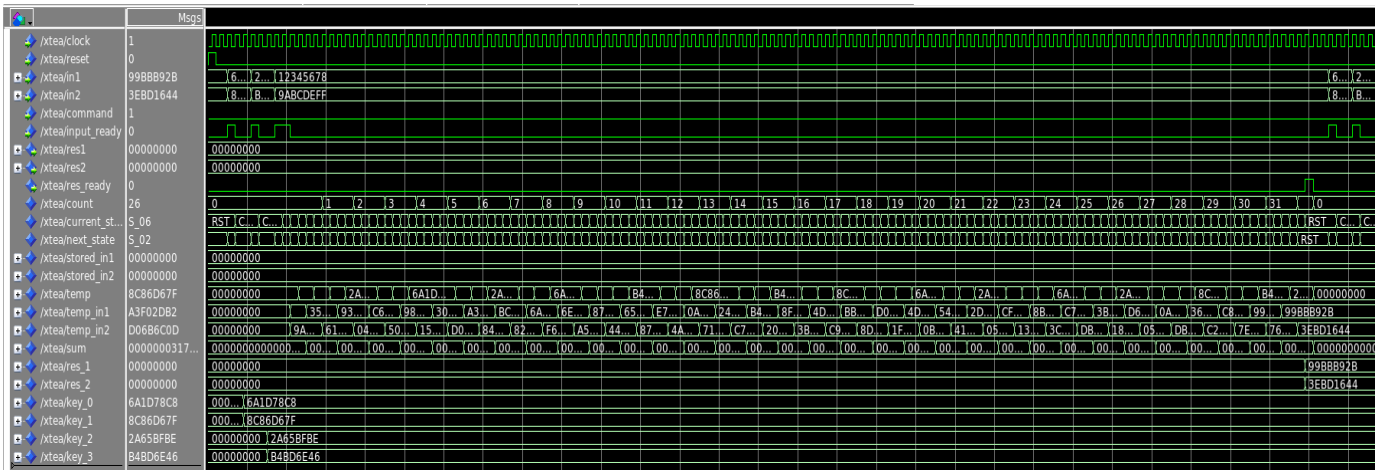


Figura 6. Setup, configurazione delle chiavi e codifica

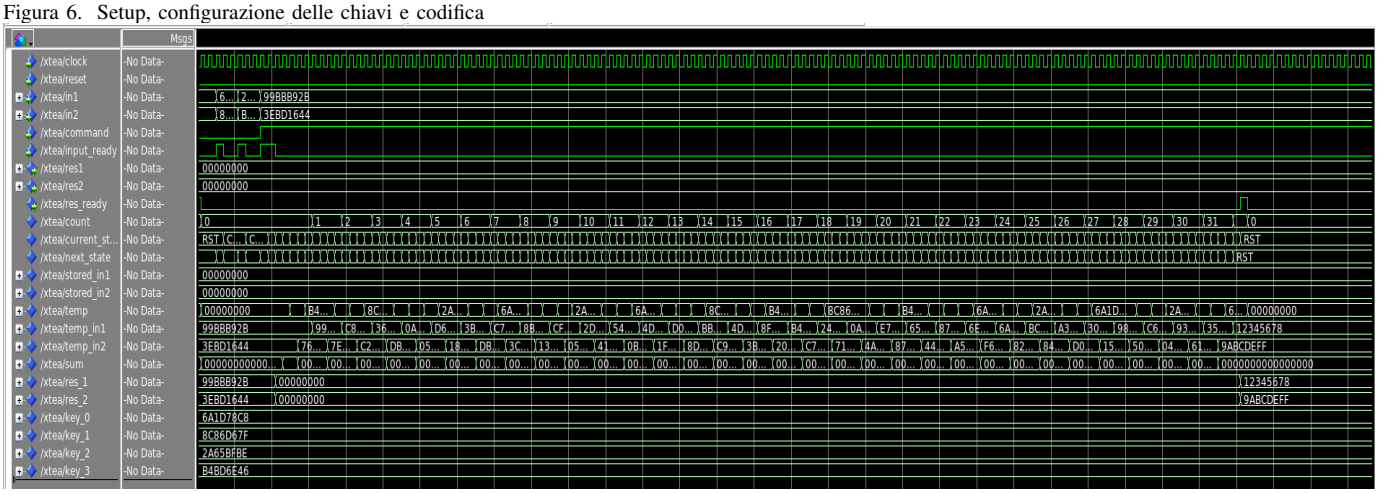


Figura 7. Setup, configurazione delle chiavi e decodifica