Modellazione in VHDL di un cifratore/decifratore XTEA

Mori Samuelele - VR439256

***Sommario*—In questa relazione verranno descritte le scelte progettuali e i metodi effettuati per la progettazione dell'algoritmo XTEA (eXtended TEA) utilizzando il linguaggio di descrizione VHDL.**

I. Introduzione

L'XTEA (eXtended TEA) è un cifrario a blocchi che fu presentato nel 1997 da David Wheeler e Roger Needham, del dipartimento informatico dell'Università di Cambridge, per correggere le vulnerabilità scoperte nel loro algoritmo TEA.[1] Lo scopo del progetto è sviluppare in VHDL sintetizzabile un chip in grado di crifrare e decifrare due parole a 32 bit data una specifica chiave. Sono state sviluppate due versioni: una scritta a livello comportamentale per un design sintetizzabile sull'FPGA PYNQ™ ed una creata utilizzando un tool di sintesi ad alto livello fornito da Xilinx® partendo da un codice C++. Questo ha permesso la comparazione tra le due metodologie, accentuando il motivo per cui le tecniche di high level synthesis non siano ancora così diffuse.

II. Background

In questo progetto sono stati utilizzati i seguenti software:

- Mentor Graphics Modelsim, per la progettazione e la simulazione del modulo hardware,
- Xilinx Vivado, per verificare che il modulo sia sintetizzabile,
- Xilinx Vivado HLS, per l'high level synthesis del codice scritto in C++.

Inoltre è stato utilizzato il linguaggio di descrizione hardware VHDL per il design e la progettazione del modulo.

III. Metodologia applicata

Il progetto può essere diviso in quattro fasi: implementazione, simulazione, sintesi e sintesi ad alto livello.

A. Implementazione

L'implementazione in VHDL segue la versione SystemC sviluppata in precedenza (FSM mostrata in figura 4).
La macchina a stati finiti presenta i seguenti 8 stati:

- START: stato iniziale;
- INITIAL: fase di inizializzazione, rimane nello stato finché non sono pronti nuovi input;
- ASSIGN: vengono assegnati i valori ai segnali w0, w1 ed anche a sum se è stata selezionata la fase di decifratura;
- ENCRYPT_1, ENCRYPT_2: fasi di cifratura;
- DECRYPT_1, DECRYPT_2: fasi di decifratura;
- TERM: vengono scritti i risultati su output0 e output1 e viene messo output_ready a 0.

L'entità VHDL presenta i seguenti ingressi:

- clk: bit per il clock;
- rst: bit per il reset;
- mode: bit per la selezione della modalità;
- input_ready: bit che indica quando l'input è pronto;
- key0, key1, key2, key3: chiavi a 32 bit da utilizzare durante le fasi di cifratura e decifratura;
- word0, word1: 32 bit dove vengono memorizzate le parole da cifrare/decrifrare.

L'entità VHDL presenta i seguenti output:

- output0, output1: 32 bit dove vengono memorizzate le parole dopo essere state cifrate/decifrate;
- output_ready: bit che indica quando l'output è pronto.

In seguito viene riportata l'interfaccia in VHDL:

```
entity xtea is
    port (
        clk             : in    bit;
        rst             : in    bit;
        mode            : in    bit;
        input_ready     : in    bit;
        output_ready    : out   bit;
        key0            : in    unsigned (31 downto 0);
        key1            : in    unsigned (31 downto 0);
        key2            : in    unsigned (31 downto 0);
        key3            : in    unsigned (31 downto 0);
        word0           : in    unsigned (31 downto 0);
        word1           : in    unsigned (31 downto 0);
        output0         : out   unsigned (31 downto 0);
        output1         : out   unsigned (31 downto 0));
end xtea;
```

B. Simulazione

Per simulare il modello è necessario utilizzare il software Mentor Graphics Modelsim. I comandi da eseguire sono i seguenti:

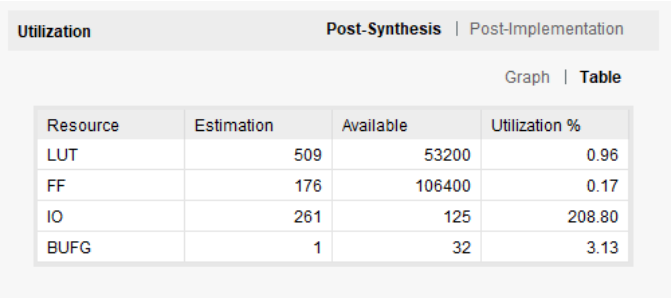
```
# Caricamento del modello
$ vsim work.xtea
# Script di simulazione
$ do stimuli.do
```

Il file xtea.vhd contiene il codice VHDL mentre il file stimuli.do contiene lo script per la simulazione tramite il quale è possibile testare il funzionamento del sistema.
In figura 5 viene mostrata la fase di cifratura, mentre in figura 6 la fase di decifratura.

C. Sintesi

Per verificare che il modulo VHDL sia sintetizzabile sulla piattaforma PYNQ xc7z020clg400-1 è stato necessario l'utilizzo del tool Xilinx Vivado. Il risultato mostrato nella figura 1 ci

indica che il modulo richiede 261 porte I/O, numero superiore rispetto alle porte presenti sulla piattaforma, e quindi non è implementabile sulla FPGA scelta. Tale limite è possibile raggirarlo grazie all'utilizzo dei registri presenti sulla board.



Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	509	53200	0.96
FF	176	106400	0.17
IO	261	125	208.80
BUFG	1	32	3.13

Figura 1: Tabella dell'utilizzo delle risorse.

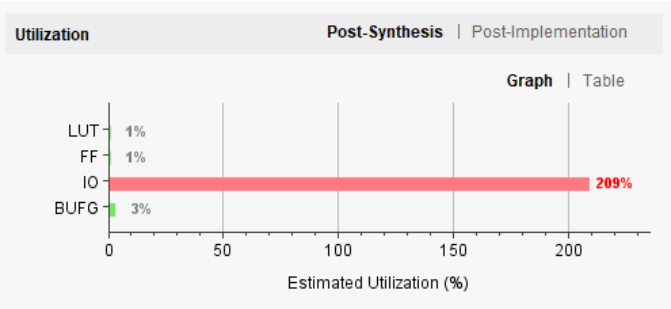


Figura 2: Grafico dell'utilizzo delle risorse.

D. Sintesi ad alto livello

Tramite l'utilizzo del tool Xilinx Vivado HLS è stato possibile generare automaticamente il codice VHDL a partire dal codice sorgente C++. Il tool ha effettuato una sintesi ad alto livello, la figura 3 ne mostra il risultato. Anche in questa versione il numero di porte I/O supera il limite, risultando quindi non implementabile sulla PYNQ.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1478	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	128	-
Register	-	-	628	-	-
Total	0	0	628	1606	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	3	0

Figura 3: Utilizzo delle risorse su piattaforma PYNQ del codice risultante dalla sintesi ad alto livello.

IV. Risultati

I risultati sono già stati in parte trattati nei punti precedenti. Nelle figure 5 e 6 viene mostrata la simulazione del codice VHDL sviluppato.
Nelle figure 1 e 2 vengono mostrati i dati riguardanti l'utilizzo delle risorse hardware su piattaforma PYNQ del codice VHDL sviluppato, mentre in figura 3 quelli della sintesi ad alto livello.

V. Conclusioni

Il confronto tra le due versione ha mostrato quanto il codice risultante può cambiare andando a sintetizzare un codice più o meno a basso livello. Il codice ad alto livello, dando informazioni più generiche rispetto alla versione VHDL sviluppata, ha portato alla creazione di un codice piú complesso, piú arduo da gestire e mantenere, di difficile lettura e soprattutto piú oneroso dal punto di vista dell'utilizzo delle risorse hardware. Questi sono alcuni dei motivi che hanno portato a far preferire la sintesi di codice VHDL/Verilog, partendo comunque da una base in SystemC/C++, alla sintesi automatizzata ad alto livello.

Riferimenti bibliografici

[1] "Xtea," https://it.wikipedia.org/wiki/XTEA.

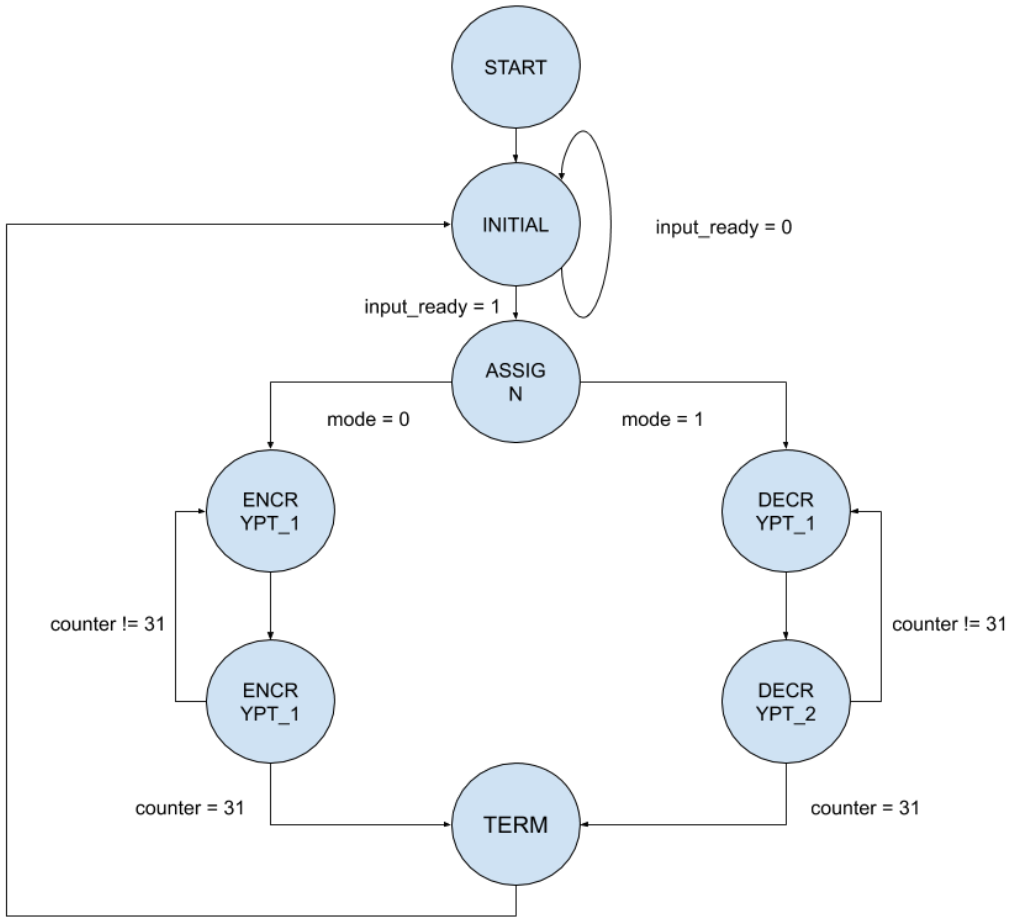


Figura 4: FSM.

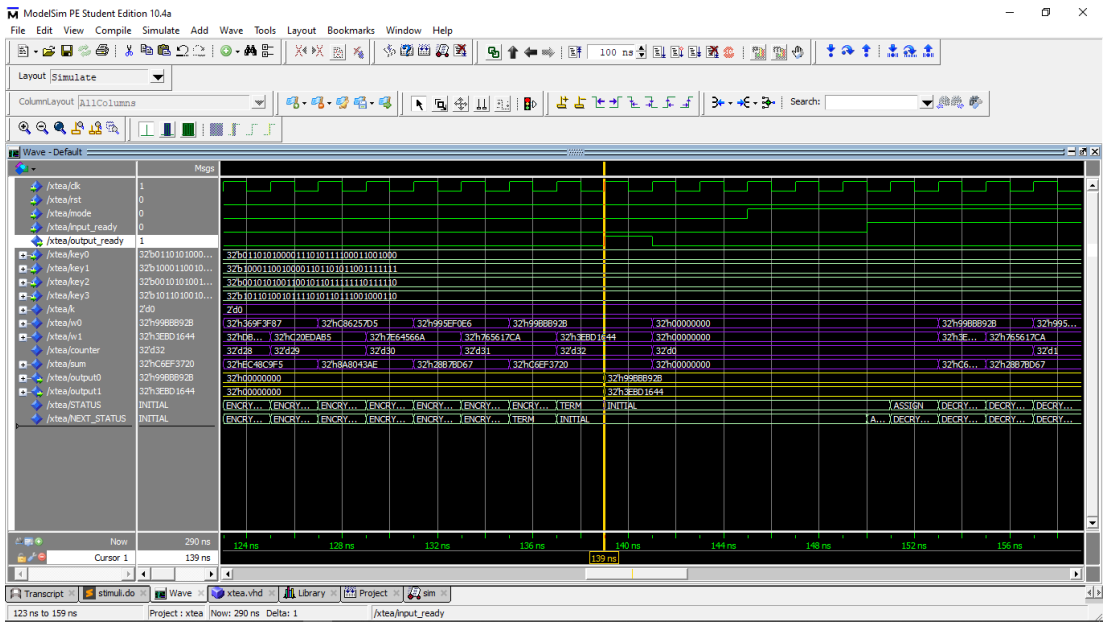


Figura 5: Esecuzione del codice VHDL, cifratura.

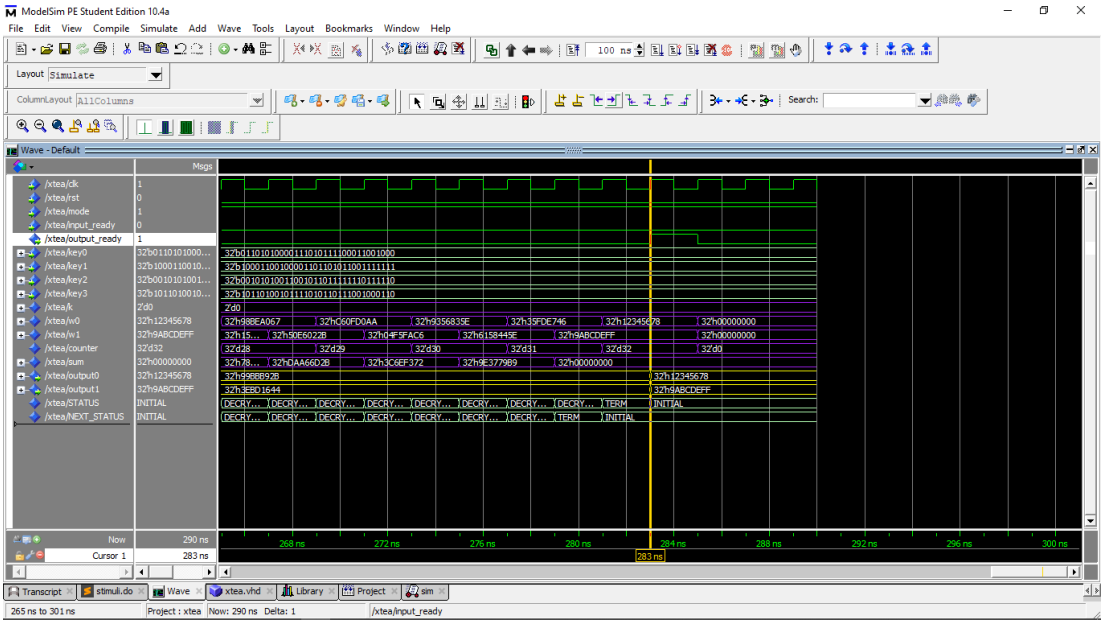


Figura 6: Esecuzione del codice VHDL, decifratura.

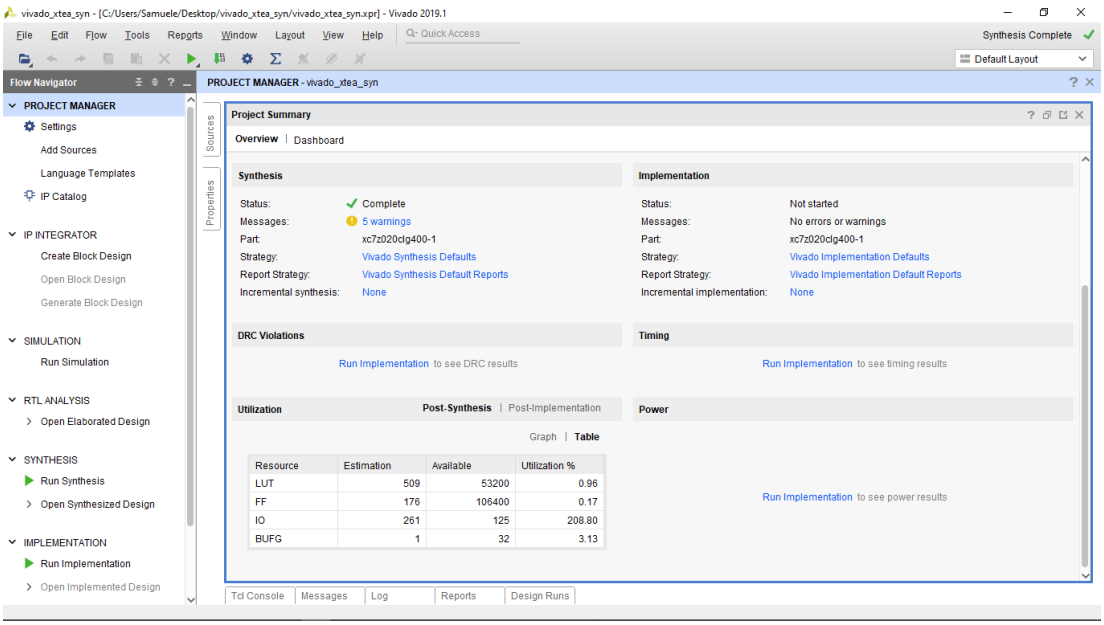


Figura 7: Vivado post sintesi codice VHDL.

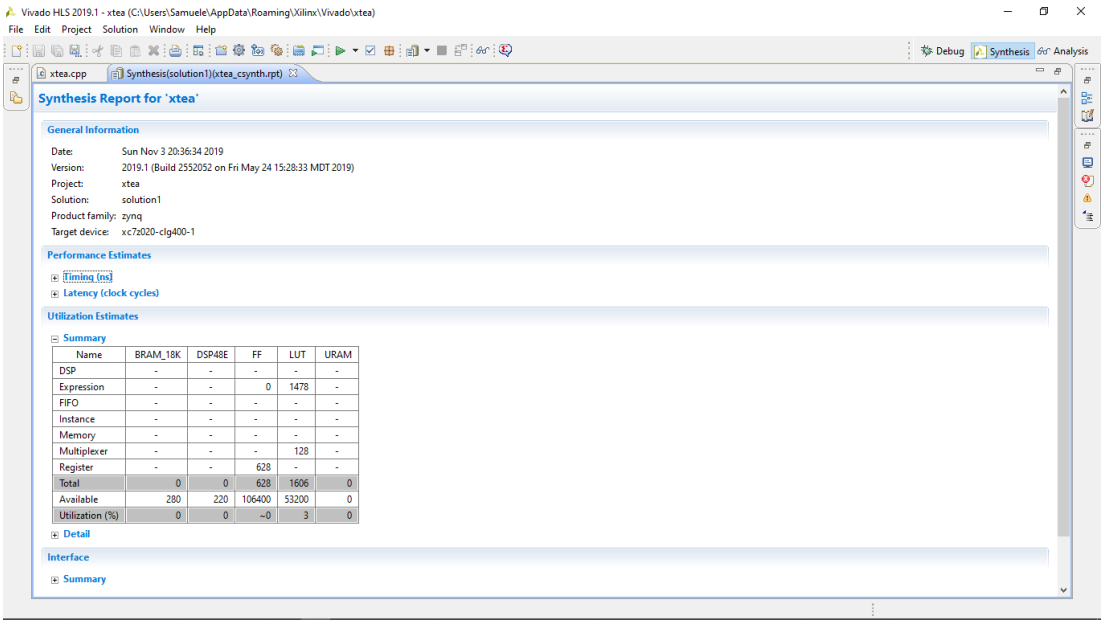


Figura 8: Vivado HLS post sintesi codice C++.