# Embedded System Design - Second Assignment

Fabio Fiorio - VR422016

***Sommario*—L'elaborato si propone come obiettivo l'implementazione in VHDL di un cifratore/decifratore con l'algoritmo XTEA.**
**Il progetto è stato realizzato seguendo la traccia dell'XTEA già realizzato precedentemente in SystemC [1] RTL, facendo attenzione comunque a nuove scelte implementative.**
**I risultati evidenziano come, una volta effettuata la sintesi logica, l'area del circuito aumenti sensibilmente e come esso possa essere ottimizzato.**

## I. Introduzione

Il progetto si pone come scopo la modellazione di un cifratore/decifratore XTEA. Il tutto è stato realizzato in VHDL, uno dei linguaggi per la descrizione dell'HW più usati al mondo per la sua versatilità ed efficienza.
Per la progettazione si è fortemente seguita la traccia utilizzata per la realizzazione dell'XTEA in SystemC RTL, con alcune opportune modifiche.
La simulazione è stata effettuata utilizzando il programma ModelSim, e la conseguente sintesi logica con il programma Vivado.
Quest'ultima mostra chiaramente come l'area del circuito aumenti nel momento in cui si passa da una realizzazione non ancora sintetizzata ad una con sole porte logiche.

## II. Background

Per la realizzazione del progetto si è utilizzato il linguaggio VHDL (VHSIC Hardware Description Language, dove VHSIC rappresenta Very Hign Speed Integrated Cirtuits) che, insieme a VERILOG, rappresenta uno dei linguaggi per la descrizione dell'HW (HDL) più usati al mondo.
VHDL offre diverse scelte di progettazione dell'HW che si basano sul modello di architettura scelto:
- Behavioral: il cosiddetto livello comportamentale, consente di modellare l'HW ad un livello più alto, sfruttando processi e, di conseguenza, i segnali per l'interazione tra essi. Si possono inoltre utilizzare anche variabili per la memorizzazione di dati. Legati al behavioral nascono poi diversi stili per la scrittura del codice, che andremo poi a trattare.
- Structural: è il livello più basso dove si istanziano i componenti che si vogliono utilizzare e si creano i collegamenti circuitali tra di essi.
Per la sintesi logica del codice è stato utilizzato ModelSim su piattaforma Windows.
ModeSim si presenta semplice ed intuitivo, ed offre la possibilità di visualizzare tutti i segnali raccolti in un grafico, in modo tale da ricercare passo a passo tutti i punti della simulazione.
Possiamo dire quindi che VHDL risulta essere molto vantaggioso per la progettazione di un sistema HW grazie alla sua versatilità, indipendenza dalla tecnologia e offre multiple metodologie di progettazione.

## III. Metodologia applicata

Come accennato nell'introduzione, l'implementazione ha seguito la traccia del cifratore/decifratore XTEA già realizzata in SystemC RTL. Abbiamo quindi un componente che prende in input i due valori da criptare o decriptare, la modalità e le 4 key, e restituisce i due valori dopo aver effettuato l'algoritmo XTEA nella modalità data in input.

```
entity XTEA is
    port (
        clk : in BIT;
        rst : in BIT;
        word0 : in UNSIGNED (31 downto 0);
        word1 : in       UNSIGNED (31 downto 0);
        KEY0 : in UNSIGNED (31 downto 0);
        KEY1 : in UNSIGNED (31 downto 0);
        KEY2 : in UNSIGNED (31 downto 0);
        KEY3 : in UNSIGNED (31 downto 0);
        input_ready : in BIT;
        mode : in        BIT;
        result0 : out UNSIGNED (31 downto 0);
        result1 : out UNSIGNED (31 downto 0);
        output_ready : out       BIT
    );
end XTEA;
```

Inoltre sono presenti due porte, una in input e una in output di tipo BIT, che vengono utilizzate per dire se i dati sono pronti per essere criptati/decriptati e se il risultato è pronto.
Nell'Architecture, ovvero l'architettura del sistema da un punto di vista interno, indica il funzionamento, si è usato lo stile behavioral per la modellazione. L'architettura si basa principalmente su due processi.
Un processo sensibile al clock e al reset, che implementa l'elaborazione della macchina a stati finiti estesa (Datapath). Il processo è stato scritto seguendo il seguente stile, sensibile ai segnali sopracitati e caratterizzato dal seguente frammento di codice:

```
process (clk, rst)
    begin
        if rst = '0' then
            STATUS <= Reset_ST;
        elsif clk = '1' then
```

Questo stile è generalmente il più utilizzato nel momento in cui si vuole descrivere l'ESFM.
L'altro processo è sensibile allo status e all'input_ready, che mi definisce il nuovo next_status sulla base delle condizioni prefissate.
In un Architecture la definizione dei segnali e degli eventuali componenti va fatta prima dell'implementazione del processo.
I segnali STATUS e NEXT_STATUS sono così definiti:

```
subtype STATUS_T is UNSIGNED (3 downto 0);
```

```
signal STATUS          : STATUS_T;
signal NEXT_STATUS     : STATUS_T;
```

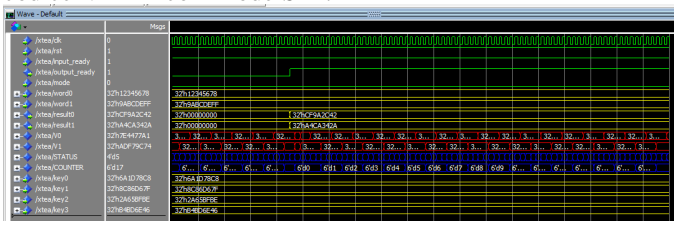
Si noti infine che per gli stati si è utilizzato un package, definito ad inizio codice, che codifica il valore del singolo stato come costante:

```
package XTEA_PACK is
    CONSTANT Reset_ST : UNSIGNED(3 downto 0)
        := "0000";
    CONSTANT ST_0 : UNSIGNED(3 downto 0) := "0001";
    CONSTANT ST_1 : UNSIGNED(3 downto 0) := "0010";
    CONSTANT ST_2 : UNSIGNED(3 downto 0) := "0011";
    CONSTANT ST_3 : UNSIGNED(3 downto 0) := "0100";
    CONSTANT ST_4 : UNSIGNED(3 downto 0) := "0101";
    CONSTANT ST_5 : UNSIGNED(3 downto 0) := "0110";
    CONSTANT ST_6 : UNSIGNED(3 downto 0) := "0111";
    CONSTANT ST_7 : UNSIGNED(3 downto 0) := "1000";
    CONSTANT ST_8 : UNSIGNED(3 downto 0) := "1001";
    CONSTANT ST_9 : UNSIGNED(3 downto 0) := "1010";
    CONSTANT ST_10 : UNSIGNED(3 downto 0) := "1011";
    CONSTANT Final_ST : UNSIGNED(3 downto 0)
        := "1100";
end XTEA_PACK;
```

**Organizzazione dell'implementazione** Nella cartella VHDL_behavioral_implementation sono presenti i file xtea.vhdl che contiene tutta l'implementazione del codice in VHDL e il file stimuli.do il quale è stato utilizzato in ModelSim per verificare il corretto funzionamento del codice tramite il comando "do stimuli.do".

## IV. Risultati

Si è testato il sistema inserendo diversi valori nel file stimuli.do per ricoprire tutte le possibili caratteristiche. Simulando con il comando "do stimuli.do" si può vedere il risultato del codice VHDL con ModelSim:



GTKwave

Dopo aver eseguito la sintesi logica con Vivado, percorrendo quindi la strada più veloce verso il silicio, ci si è fermati all'implementazione dovuta al seguente problema:


GTKwave

Questo poichè il design contiene 261 porte di input/output, invece il device target selezionato (xc7z020clg400-1) ha solo 255 porte disponibili.

Analizzando invece la tabella prodotta dopo la sintesi logica

| 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 |

GTKwave

possiamo vedere come il circuito può essere ampiamente migliorato in termini di LUT e FF in quanto la loro percentuale di utilizzo risulta essere veramente minima.

## V. Conclusioni

In conclusione si può affermare che i risultati ottenuti rispecchiano una corretta, ma non ottimale modellazione del progetto e la scrittura in un HW sintetizzabile. La sintesi può essere ottimizzata per garantire una migliore efficienza.

## Riferimenti bibliografici

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

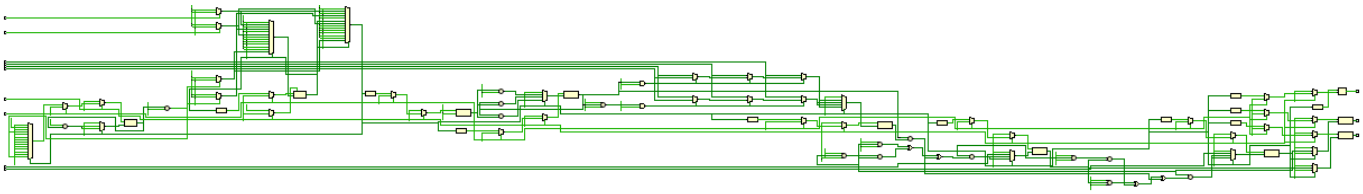


Figura 1. vivado