A synthesizable VHDL XTEA Cypher Decypher

Francesco Alessandro Picchirallo - VR432225

***Abstract*—This document reports the synthesizable VHDL hardware modeling of the eXtended TEA (XTEA) Algorithm.**

I. Introduction

The project described in this report consists on develop in synthesizable $VHDL$ an hardware module based on the $XTEA$ Algorithm, which is a a block cipher, starting from the first version developed using $SystemC$ at different levels of abstraction. Furthermore, using the main tools, it is necessary to perform the synthesis on FPGA, such as the Xilinx PYNQ Z1.

II. Background

The $XTEA$ Algorithm has been developed in $VHDL$[1] language, which is an Hardware Description Language (HDL). $VHDL$ is a standard for all stages of hardware design that provides the possibility to develop at different levels of abstractions, also used for the synthesis tools. The $VHDL$ abstraction levels are:

- Behavioral: high level behavioral description.
- Data-Flow: description as a data flow through the input / output flow and arithmetic relations.
- Structural: low level description.

The main feature that differentiates the $VHDL$ language from a software language is competitiveness. Once different parts of $VHDL$ code are translated into an electronic circuit, they work simultaneously, because they have dedicated Hardware.
The High-Level Synthesis (HLS)[2] is a process converting an high-level description of a design (in C/C++) to RT Level, which can be synthesized in order to obtain a circuit with logic gates.. The software used is Xilinx Vivado HLS.
The Logic Synthesis, instead, is a process that takes a RT level description in input and produces, through a synhtesis tool, a circuit logic with gates. The software used is Xilinx Vivado. Mentor Graphics Modelsim instead is used for the design and simulation of the hardware module described in $VHDL$.

III. Applied Methodology

A. VHDL implementation and simulation

$VHDL$ implementation of the $XTEA$ is similar to the previous in $SystemC$, where the module is composed from an FSM and a Datapath, designed in the $SystemC$ implementation. Figure 1 show the $XTEA$ module with input and output ports.

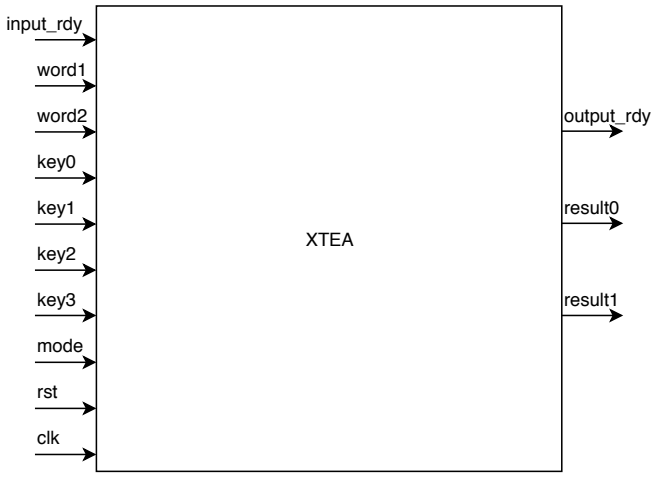

Figure 1. Xtea Module

Composed from six 32 bit input ports, such as the four keys used for the encryption/decryption, the two word that will be encrypt/decrypt, and four 1 bit input ports representing clock, reset, input_rdy and mode ports, where the input_rdy is used when all the input are ready to use, and the mode port used to select the module execution mode. Internal signals are described as:

- $v0$: 32 bit signal, used to cipher and decipher the word0
- $v1$: 32 bit signal, used to cipher and decipher the word1
- $counter$: 6 bit signal, used for the 32 iterations necessary for the both mode, cipher and decipher.
- $delta$: 32 bit signal, used to memorize the delta value.
- $temp$: 32 bit signal, used to temporarily store the key which will be used.
- $STATUS$: signal representing the actual state of the actual FSM state.
- $NEXT\_STATUS$: signal representing the next state.

The ports are defined in the entity and the signals are defined in the architecture. An entity declaration defines the interface between the entity and the environment in which it is used. The interface includes all inputs, outputs and bidirectional signals defined in the port declaration. An architecture instead defines a body for a component entity, just the implementations of the entity. The entity is defined as follows:

```
ENTITY xtea IS
    generic(SIZE  :  INTEGER := 32);
    port(
    clk       :IN  bit;
    rst       :IN  bit;
    input_rdy :IN  bit;
    word1   :IN  unsigned (SIZE-1 downto 0);
    word2   :IN  unsigned (SIZE-1 downto 0);
```

```
    key0    :IN  unsigned (SIZE-1 downto 0);
    key1    :IN  unsigned (SIZE-1 downto 0);
    key2    :IN  unsigned (SIZE-1 downto 0);
    key3    :IN  unsigned (SIZE-1 downto 0);
    mode    :IN  bit;
    result0 :OUT  unsigned(SIZE-1 downto 0);
    result1 :OUT  unsigned(SIZE-1 downto 0);
    output_rdy :OUT  bit;
    );
end xtea;
```

The module is described as a thirteen states EFSM, the first process is sensible to STATUS and input_rdy signals, which task is to update the next state of the module, while the second one is sensible to rst and clk, implemented according to $VHDL$ four style. It has to update the actual state and to perform the operations to execute the $xtea$ algorithm and write the results in output. Figure 2 below show the $xtea$ FSMD.

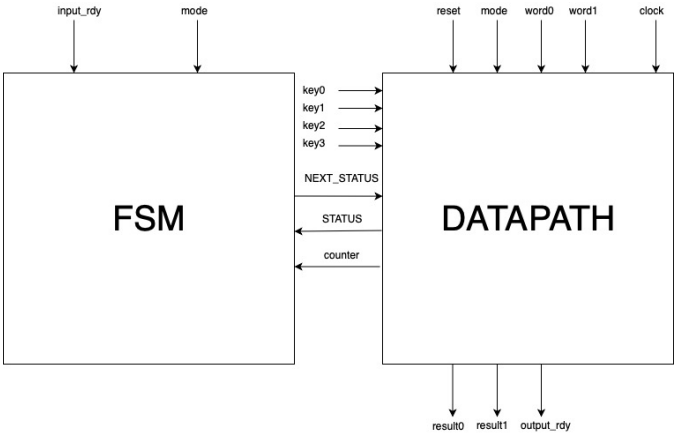

Figure 2. Xtea FSMD

The module has been simulate using Mentor Graphics Modelism, after developing the $stimuli.do$ file, which is a script used for the simulation setup. The command used are the follow:

```
To compile the model:
$ vcom -93 xtea.vhdl
To open the simulator to simulate the model
$ vsim work.xtea
To automate the simulation setup
$ do stimuli.do
```

The stimuli.do file start with a configuration phase where a graph traces are generated from signals and ports. Then a reset of the FSM is done. The words and the keys will be set for the encryption phase.
Then will be set the decryption mode, doing a reset of the FSM and afterwards the encrypted values will be assign to the ports.

As can be seen from the Figure 3 and Figure 4, encryption and decryption modes are run for 300 ns both, precisely the first mode starts at 6 ns and finish at 275 ns ,the second one starts at 316 ns and finish at 583 ns. Figure 7 & 8 shows the crypt and decrypt waves in detail.

```
echo "First invocation: Encryption "

force word1 16#012345678 0 ns
force word2 16#09abcdeff 0 ns

force key0 2#01101010000111010111100011001000 0 ns
force key1 2#10001100100000110110101100111111 0 ns
force key2 2#00101010011001010111111110111110 0 ns
force key3 2#10110100101111010110111001000110 0 ns

force mode 0 0 ns
force input_rdy 1 0 ns

force rst 1 0 ns
run 4 ns
force rst 0 0 ns

run 300 ns
```

Figure 3. Encryption VHDL

```
force rst 1 0 ns, 0 6 ns

force input_rdy 0 0 ns
run 6 ns

echo "Second invocation: Decryption "

force word1 16#99bbb92b 0 ns
force word2 16#3ebd1644 0 ns

force key0 2#01101010000111010111100011001000 0 ns
force key1 2#10001100100000110110101100111111 0 ns
force key2 2#00101010011001010111111110111110 0 ns
force key3 2#10110100101111010110111001000110 0 ns

force mode 1 0 ns
force input_rdy 1 0 ns

run 300 ns
```

Figure 4. Decryption VHDL

B. Logic Synthesis

$Logic\ Synthesis$ is necessary to verify that the $vhdl$ module is synthesizable, this is made possible using the Xilinx Vivado tool. The synthesis is performed simulating the $Pynq$ board, which reference code is $xc7z020clg400$-1, figure 5 show the logic synthesis results, resource utilization in particular.

Resource	Utilization	Available	Utilization %
LUT	368	53200	0.69
FF	202	106400	0.19
IO	261	125	208.80

Figure 5. Resource utilization on Pynq board

As can be seen from the last picture, this $vhdl$ implementation

requires 261 I/O ports while the $Pynq$ board has 255 ones, it means that this $vhdl$ implementation of the $xtea$ algorithm is not directly implementable on this board. The main problem has been maintaining the same $SystemC$ implementation, when instead a specific version for this board should have been developed.

C. High Level Synthesis

The High Level Synthesis is provided from the Xilix Vivado HLS tool. Starting from an high level description in C++, this synthesis allows to generate a $VHDL$, $SystemC$ and $Verilog$ implementation based on $Pynq$ board. This generated version is anyway not implementable in the reference board, overuse of I/O ports is the problem. Figure 6 shows the utilization estimates resulting of the High Level Synthesis on $Pynq$ board.

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

Figure 6. Utilization estimates HLS on Pynq board

IV. Results

As explained in the previous sections, the choice of re using the $SystemC$ implementation of the $xtea$ algorithm allowed a considerable saving of time. The resulting $vhdl$ version is correct and functional, as shown in the simulation part, synthesizable but not directly implementable in the reference board.

V. Conclusions

The code generated using High Level Synthesis tools is much more complex and expensive from an hardware resources viewpoint, resulting not directly implementable on the board. To describe an hardware module direct development is therefore preferable, moving from a C++ or $SystmeC$ to a $VHDL$ version.

References

[1] "Ieee standard vhdl language reference manual," *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pp. c1–626, Jan 2009.

[2] A. M. Philippe Coussy, *High-Level Synthesis*. Springer, Dordrecht, 2008.

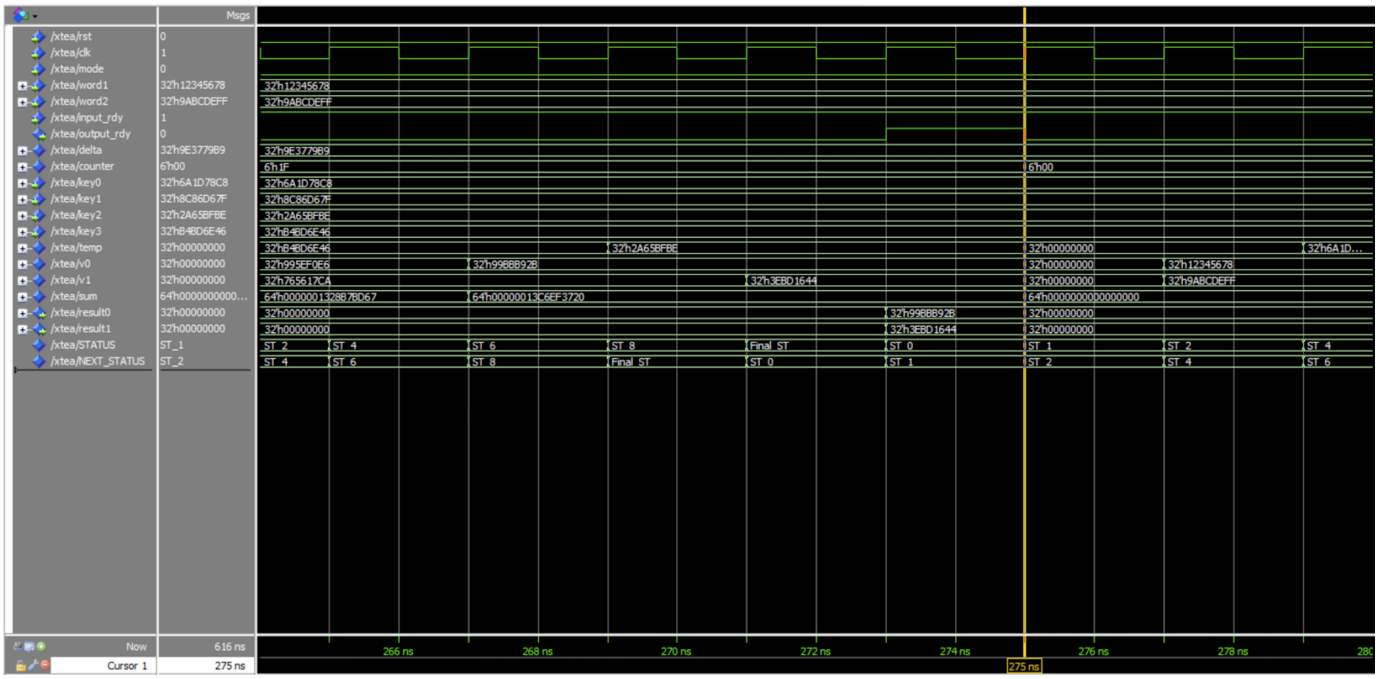

Figure 7. Encryption

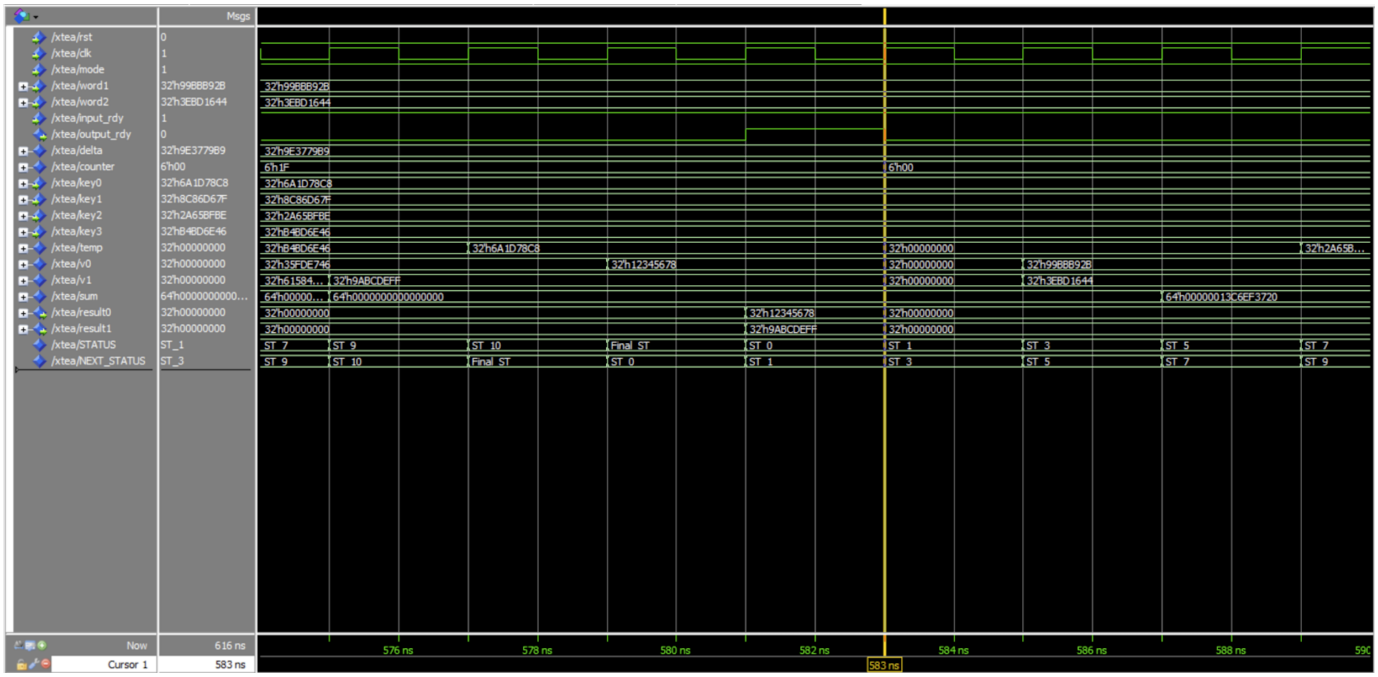

Figure 8. Decryption

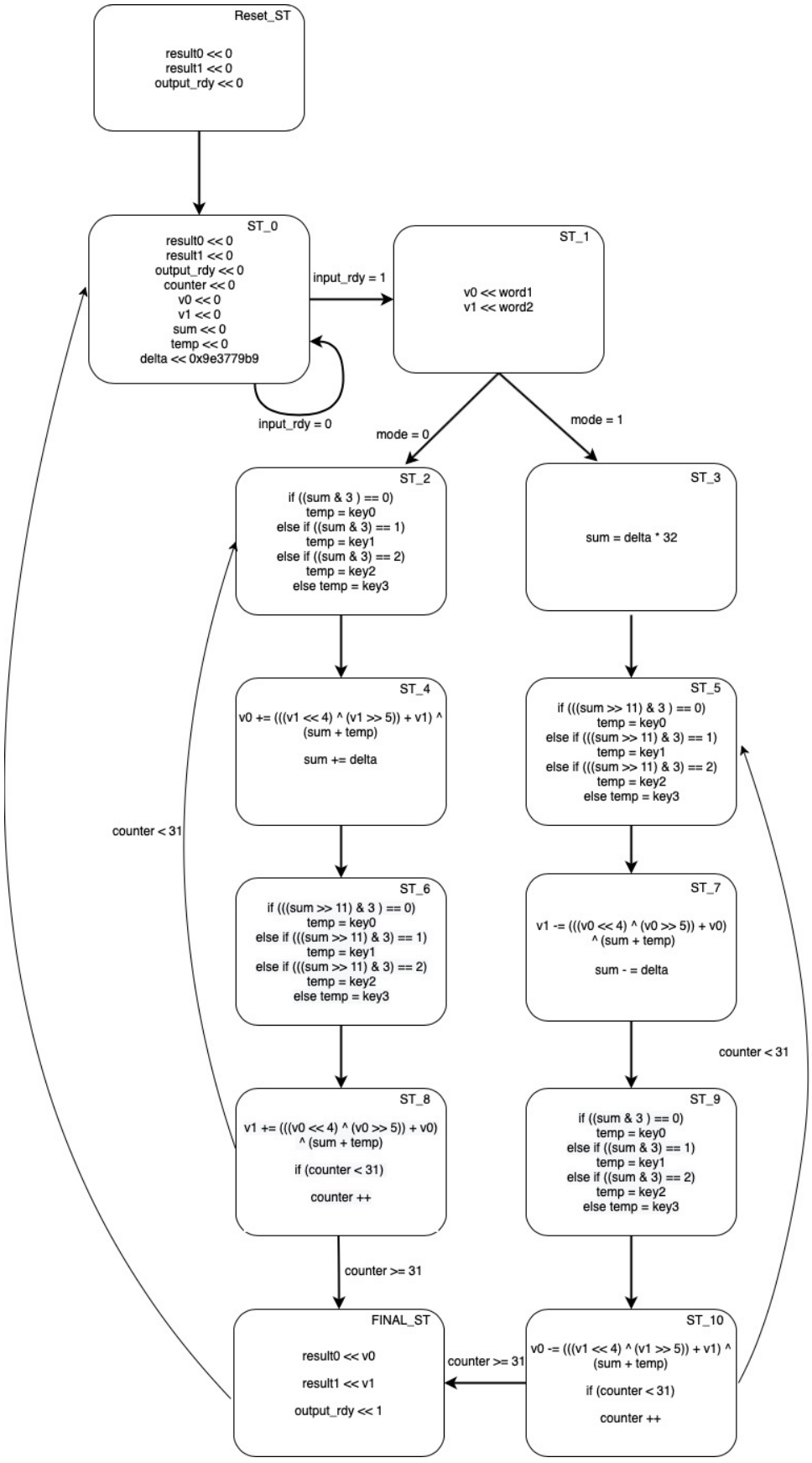

Figure 9. EFSM XTEA module