Periodic waveform used to synchronise the operation engineering essay

Engineering

The main task of this coursework is to design hardware in VHDL to execute both encryption and decryption of data using Tiny Encryption Algorithm. Altera Quartus 12. 1 is used as main tool for system development and CodeLight is used as an environment to run the C code to check the correctness of the system. Designed implementation of the system is supposed to be able to carry out 16 rounds of encryption and decryption for word lengths 4, 8 and 16 bits. Modular design approach is used to implement the system, that is, components that are used in complete system are built and tested before further use.

2. Introduction

2. 1 Coursework Aims

Design TEA hardware in VHDLDevelop VHDL skills

2. 2 Coursework Objectives

Produce components that are used in final system implementationProduce the design that is capable of carrying a single round of encryption and decryption for word lengths 4, 8 and 16 bits. Produce a design that is capable of carrying 16 rounds of encryption and decryption.

3. System Implementation

3. 1 System Architecture

The system has 6 inputs and one output. The detailed description of all of them is summarised in Table 1 below. NoNameType(Input/Output)Bus width (bits)Description1clkInput1Periodic waveform used to synchronise the operation of the system. 2RoundsInput6Determines a number of rounds of encryption (used when a single component does multiple

rounds)3Data_inInput64Either encrypted or decrypted data input port. If the word length is less than 64 bits, the appropriate number of MSBs are not used. 4keyInput128The 128 bit key is split into 4 32 bit long keys that are used for encryption and decryption. k0-[31.. 0], k1-[63.. 32], k2-[95.. 64], k3-[127.. 96]. 5word_lengthInput2Defines the word length of the data:" 00"-4 bit," 01"- 8 bit, " 10" - 16 bit, " 11"- 64 bit. 6encrypt_decryptInput2Defines whether the encryption or decryption is executed: '0' for decryption and '1' for encryption. 7Data_outOutput64Either encrypted or decrypted data output port.

Table 1: Input/Output Description

Two ways of implementing TEA hardware are presented and compared. First implementation is using one TEA component to perform one or more rounds of encryption. This is achieved by feeding the output of the system back to input port and performing multiple rounds of encryption or decryption. This approach does not provide an excellent data throughput, that is, only one set of data can be either encrypted or decrypted at a time. However, the overall complexity is reduced greatly. The other implementation includes 16 separate TEA components (each performing one round of encryption and decryption) and 16 registers between them. This approach takes more time to get first result after 16 rounds, compared to a single component doing 16 rounds. However using this approach multiple words of data can be encrypted and decrypted simultaneously. TEA hardware consists of the following components: xor_3 executes xor logical operation on 3 inputs. splitter splits the input data in halves depending on the word length for example if the input is 16 bit long, then the splitter splits the data into two 8

bit words. Implemented using " case" statement. shift_4_left shifts 4 bits left. shift_5_right shits 5 bits right. The " case" is used to shift different word length data correctly. key_splitter splits 128 bit key into four 32 bit keys. data_merger puts together two halves of data. adder_subtractor adds or subtracts data.

3. 2 Simulation results

To make waveforms readable, some of the are split in half

3. 2. 1 xor_3

The output from the XOR gate is HIGH when both inputs are different. The same applies to 3 input xor gate. To demonstrate the correctness of this component all bits are set either to '1' or '0'.

3. 2. 2 splitter

It can be seen on the simulation result below, that the random data is split in halves.

3. 2. 3 shift_4_left

One hexadecimal number represents 4 bits, therefore it can be seen that random data is shifted one symbol to the left.

3. 2. 4 shift_5_right

If the word length of the data to be encrypted or decrypted is 4, 8 or 16 bits, it means that when split, it will be 2, 4 or 16 bits long. In first case bits [1.. 0] will set to 0, in second case bits [3.. 0] will set to 0 and finally bits [7.. 3] will be set to zero.

3. 2. 5 key_splitter

The simulation results below proves that key splitter's correct operation. The component splits the keys depending on encryption and decryption input.

3. 2. 6 data_merger

The simulation results below demonstrate the correct operation of the components that is respopnsible for merging encrypted and decrypted data.

3. 2. 7 adder_subtractor

This component is performing as expected, unless the output value exceeds . The y is the output, x defines whether addition or subtraction is done

3. 2. 8 TEA

The simulation results below demonstrate one round of encryption and decryption for 64 bit word. The 4, 8 and 16 bit words are manipulated in similar manner. The only difference is that instead of encrypting all 64 bits, for example only 8 bits [7.. 0] are encrypted or decrypted. It takes approximately 65 ns to obtain the resultThe next simulation shows the same component with minor modifications performing 16 rounds encryption and decryption. It takes approximately 515 ns to obtain the result of encryption and decryption. Only one set of data can be manipulated at a time! Finally, to improve the throughput, the component that includes 16 TEA components is implemented. Each component performs one round of encryption/decryption. Registers are put between each TEA component to enable the pipelining. Thus, after first set of data has been fed in, 30 ns later next set of data can be fed in and so on. The first result is ready in 660 ns,

which is slightly longer than a single component performing 16 rounds, however the second result is ready 15 ns later of the first result. All results have been compared to the output of the C program that implements the Tiny Encryption Algorithm. It proves that the system is working correctly. All simulation were targeted to Cyclone II device family and the shortest clock period that produces reliable results is 30 ns for the first implementation and 20 ns for the second.

4. Conclusion

Both proposed systems are capable of encrypting and decrypting data of different length (4, 8, 16 and 64 bits). Both systems are synchronous. The first option is easy to implement and it is flexible, because it has Rounds input that defines the number of encryption decryption cycles. Changing this input, the system is capable of carry out different number of rounds. However the throughput is very low: 1 result in 515 ns. The second proposed system has far better throughput because of the pipelining, however it is more complex and it is not very flexible. Single TEA component consists of 7 major components: xor_3, splitter, shift_4_left, shift_5_right, data_ merger, key_splitter and adder_subtractor. All subsystems were simulated to ensure their correct functionality. Overall, this coursework has been very challenging, however the understanding of digital electronics, knowledge of VHDL and Altera software have improved significantly.

Appendix - VHDL code

tea16library ieee; use ieee. std_logic_1164. all; entity tea16 IS PORT

(

encrypt_decrypt: IN std_LOGIC; word_length: IN Std_Logic_vector(1 downto 0); key: IN Std_Logic_Vector(127 downto 0); clk : IN std_logic; dataInput : IN std_logic_vector (63 downto 0); dataOutput : OUT std_logic_vector (63 downto 0)

);

end tea16; ARCHITECTURE structural of tea16 ISCOMPONENT reg PORT

(

clk: IN std_logic; d : IN std_logic_vector(63 downto 0); q: OUT std_logic_vector(63 downto 0)

);

END COMPONENT reg; COMPONENT tea_com PORT

(

sum_input: IN Std_LOGIC_VECTOR(31 downto 0); clk : IN Std_Logic; Rounds: IN Std_Logic_Vector(6 downto 0); Data : IN Std_Logic_Vector(63 downto 0); key: IN Std_Logic_Vector(127 downto 0); word_length: IN Std_Logic_vector(1 downto 0); encrypt_decrypt: IN Std_Logic; Encrypted_data: OUT Std_Logic_Vector (63 downto 0)

);

END COMPONENT tea_com; SIGNAL Data1 : Std_Logic_Vector(63 downto 0); SIGNAL out1, out2, out3, out4, out5, out6, out7, out8: Std_Logic_Vector (63 downto 0); SIGNAL out9, out10, out11, out12, out13, out14, out15, out16: Std_Logic_Vector (63 downto 0); SIGNAL out17, out18, out19, out20, out21,

out22, out23, out24: Std_Logic_Vector (63 downto 0); SIGNAL out25, out26, out27, out28, out29, out30, out31, out32: Std_Logic_Vector (63 downto 0); SIGNAL sum1, sum2, sum3, sum4, sum5, sum6, sum7: Std_Logic_Vector(31 downto 0); SIGNAL sum8, sum9, sum10, sum11, sum12, sum13, sum14, sum15, sum16 : Std_Logic_Vector(31 downto 0); CONSTANT numberOfRounds: Std_Logic_Vector(6 downto 0) := " 0000001"; SIGNAL k: Std_Logic_Vector(127 downto 0); SIGNAL wl: Std_Logic_vector(1 downto 0); SIGNAL ed: std_LOGIC; BEGINtea_com_1: tea_comport map (sum1, clk, numberOfRounds, Data1, k, wl, ed, out1); reg1: regport map (clk, out1, out2); tea_com_2: tea_comport map (sum2, clk, numberOfRounds, out2, k, wl, ed, out3); reg2: regport map (clk, out3, out4); tea_com_3: tea_comport map (sum3, clk, numberOfRounds, out4, k, wl, ed, out5); reg3: regport map (clk, out5, out6); tea_com_4: tea_comport map (sum4, clk, numberOfRounds, out6, k, wl, ed, out7); reg4: regport map (clk, out7, out8); tea_com_5: tea_comport map (sum5, clk, numberOfRounds, out8, k, wl, ed, out9); reg5: regport map (clk, out9, out10); tea_com_6: tea_comport map (sum6, clk, numberOfRounds, out10, k, wl, ed, out11); reg6: regport map (clk, out11, out12); tea_com_7: tea_comport map (sum7, clk, numberOfRounds, out12, k, wl, ed, out13); reg7: regport map (clk, out13, out14); tea_com_8: tea_comport map (sum8, clk, numberOfRounds, out14, k, wl, ed, out15); reg8: regport map (clk, out15, out16); tea_com_9: tea_comport map (sum9, clk, numberOfRounds, out16, k, wl, ed, out17); reg9: regport map (clk, out17, out18); tea_com_10: tea_comport map (sum10, clk, numberOfRounds, out18, k, wl, ed, out19); reg10: regport map (clk, out19, out20); tea_com_11: tea_comport map (sum11, clk, numberOfRounds, out20, k, wl, ed, out21);

reg11: regport map (clk, out21, out22); tea_com_12: tea_comport map (sum12, clk, numberOfRounds, out22, k, wl, ed, out23); reg12: regport map (clk, out23, out24); tea_com_13: tea_comport map (sum13, clk, numberOfRounds, out24, k, wl, ed, out25); reg13: regport map (clk, out25, out26); tea_com_14: tea_comport map (sum14, clk, numberOfRounds, out26, k, wl, ed, out27); reg14: regport map (clk, out27, out28); tea_com_15: tea_comport map (sum15, clk, numberOfRounds, out28, k, wl, ed, out29); reg15: regport map (clk, out29, out30); tea_com_16: tea_comport map (sum16, clk, numberOfRounds, out30, k, wl, ed, out31); reg16: regport map (clk, out31, dataOutput); PROCESS (clk)BEGINcase(encrypt_decrypt) iswhen '1' => Data1 <= dataInput; k <= key; wl <= word_length; ed <= encrypt_decrypt; sum1 <= X" 00000000"; sum2 <= X" 9e3779b9"; sum3 <= X" 3c6ef372"; sum4 <= X" daa66d2b"; sum5 <= X" 78dde6e4"; sum6 <= X" 1715609d"; sum7 <= X" b54cda56"; sum8 <= X" 5384540f"; sum9 <= X" f1bbcdc8"; sum10 <= X" 8ff34781"; sum11 <= X" 2e2ac13a"; sum12 <= X" cc623af3"; sum13 <= X" 6a99b4ac"; sum14 <= X" 08d12e65"; sum15 <= X" a708a81e"; sum16 <= X" 454021d7"; when '0' => Data1 <= dataInput; k <= key; wl <= word_length; ed <= encrypt_decrypt; sum1 <= X" 3c6ef372"; sum2 <= X" daa66d2b"; sum3 <= X" 78dde6e4"; sum4 <= X" 1715609d"; sum5 <= X" b54cda56"; sum6 <= X" 5384540f"; sum7 <= X" f1bbcdc8"; sum8 <= X" 8ff34781"; sum9 <= X" 2e2ac13a"; sum10 <= X" cc623af3"; sum11 <= X" 6a99b4ac"; sum12 <= X" 08d12e65"; sum13 <= X" a708a81e"; sum14 <= X" 454021d7"; sum15 <= X" e3779b90"; sum16 <= X" 81af1549"; end case; END PROCESS; END structural; reglibrary ieee; use ieee. std_logic_1164. all; ENTITY reg IS PORT

(

clk: IN std_logic; d : IN std_logic_vector(63 downto 0); q: OUT

std_logic_vector(63 downto 0)

);

END reg; ARCHITECTURE structural OF reg ISBEGIN-- Register with active-high ClockPROCESSBEGINWAIT UNTIL clk = '1'; q <= d; END PROCESS; END structural; tea_comLIBRARY ieee; USE ieee. std_logic_1164. all; USE ieee. std_logic_unsigned. all; USE ieee. numeric_std. all; ENTITY tea_com IS PORT

(

sum_input: IN Std_LOGIC_VECTOR(31 downto 0); clk : IN Std_Logic; Rounds: IN Std_Logic_Vector(6 downto 0); Data : IN Std_Logic_Vector(63 downto 0); key: IN Std_Logic_Vector(127 downto 0); word_length: IN Std_Logic_vector(1 downto 0); encrypt_decrypt: IN Std_Logic; dataOutput: BUFFER Std_Logic_Vector (63 downto 0)

);

END tea_com; ARCHITECTURE structural OF tea_com ISCOMPONENT xor_3 PORT

(

A: IN Std_Logic_Vector(31 downto 0); B: IN Std_Logic_Vector(31 downto 0); C: IN Std_Logic_Vector(31 downto 0); Y: OUT Std_Logic_Vector(31 downto 0)

);

END COMPONENT xor_3; COMPONENT splitter PORT

(

data_input: IN Std_Logic_Vector(63 downto 0); wordLength: IN Std_Logic_Vector(1 downto 0); encrypt_decrypt: IN Std_Logic; data_left: OUT Std_Logic_Vector(31 downto 0); data_right: OUT Std_Logic_Vector(31 downto 0)

);

END COMPONENT splitter; COMPONENT shift_5_right PORT

(

data_in: IN Std_Logic_Vector(31 downto 0); word_length: IN Std_Logic_Vector(1 downto 0); data_out: OUT Std_Logic_Vector(31 downto 0)

);

END COMPONENT shift_5_right; COMPONENT shift_4_left PORT

(

data_in : IN STD_LOGIC_VECTOR(31 downto 0); data_out: OUT STD_LOGIC_VECTOR(31 downto 0)

);

END COMPONENT shift_4_left; COMPONENT key_splitter PORT

(

key: IN Std_Logic_Vector(127 downto 0); encrypt_decrypt: IN Std_Logic; K0: OUT Std_Logic_Vector(31 downto 0); K1: OUT Std_Logic_Vector(31 downto 0); K2: OUT Std_Logic_Vector(31 downto 0); K3: OUT Std_Logic_Vector(31 downto 0)

);

END COMPONENT key_splitter; COMPONENT data_merger PORT

(

data_left: IN Std_Logic_Vector(31 downto 0); data_right: IN Std_Logic_Vector(31 downto 0); word_length: IN Std_Logic_vector(1 downto 0); encrypt_decrypt: IN Std_Logic; output: OUT Std_Logic_Vector(63 downto 0)

);

END COMPONENT data_merger; COMPONENT adder_subtractor PORT

(

a: IN Std_Logic_Vector(31 downto 0); b: IN Std_Logic_Vector(31 downto 0); x: IN Std_logic; y: OUT Std_Logic_Vector(31 downto 0)

);

END COMPONENT adder_subtractor; CONSTANT delta : Std_Logic_Vector(31 downto 0) := X" 9e3779b9"; SIGNAL keySignal: Std_Logic_Vector(127 downto 0); SIGNAL dataSignal: Std_Logic_Vector(63 downto 0); SIGNAL sumSignal, sumSignal1, K0, K1, K2, K3: Std_Logic_Vector(31 downto 0); SIGNAL wl: Std_Logic_vector(1 downto 0); SIGNAL ed : Std_Logic; SIGNAL a0, a1, a2, a3, a4, a5, a6, a7: Std_Logic_Vector(31 downto 0); SIGNAL b0, b1, b2, b3, b4, b5, b6, b7, sum: Std_Logic_Vector(31 downto 0); BEGINc0 : key_splitterport map (keySignal, ed, K0, K1, K2, K3); c1 : splitter port map (dataSignal, wl, ed, a0, b0); c2 : shift_4_left port map (b0, b1); c3 : shift_5_right port map (b0, wl, b2); c4 : adder_subtractor port map (K0, b1, '1', b3); c5 : adder_subtractor

port map (sumSignal, delta, ed, sumSignal1); c6 : adder_subtractor port map (sumSignal1, b0, '1', b4); c7 : adder_subtractor port map (K1, b2, '1', b5); c8 : xor_3 port map (b3, b4, b5, b6); c9 : adder_subtractor port map (a0, b6, ed, a1); c10: shift_4_left port map (a1, a2); c11: shift_5_right port map (a1, wl, a3); c12: adder_subtractor port map (K2, a2, '1', a4); c13: adder_subtractor port map (sumSignal1, a1, '1', a5); c14: adder_subtractor port map (K3, a3, '1', a6); c15: xor_3 port map (a4, a5, a6, a7); c16: adder_subtractor port map (b0, a7, ed, b7); c17: data_mergerport map (a1, b7, wl, ed, dataOutput); PROCESS(clk, word_length)VARIABLE i, variable_1, variable_2, variable_3 : INTEGER := 0; BEGINif (clk'event and clk = '1') thencase (encrypt_decrypt) iswhen '1' => case (i) iswhen 0 => sum <= sum_input; when 1 => dataSignal <= Data; sumSignal <= sum; keySignal <= key; wl <= word_length; ed <= encrypt_decrypt; when others => dataSignal <= dataOutput; sumSignal <= sumSignal1; end case; i := i + 1; if (i = Rounds + 1) theni := 1; end if; when '0' => case (i) iswhen 0 => variable_1 := conv_integer(delta); variable_2 := conv_integer(Rounds+1); variable_3:= variable_2*variable_1; sum <= STD_LOGIC_VECTOR(TO_UNSIGNED(variable_3, 32)); when 1 => dataSignal <= Data; sumSignal <= sum; keySignal <= key; wl <= word_length; ed <= encrypt_decrypt; when others => dataSignal <= dataOutput; sumSignal <= sumSignal1; end case; i := i + 1; if (i = Rounds + 1) theni:= 1; end if; end case; end if; END PROCESS; END structural;