Cryptographic Engineering 2021-22

Exercise 2

In this exercise you will implement an AES-128 hardware which can perform AES encryption and decryption with 128-bit key on the Xilinx PYNQ-Z2 platform following a hw/sw co-design methodology.

The exercise has a total of 20 points. See 'Marking Scheme' for more information.

The deadline for code submission is: 29th November 23:59.
See 'Code Submission' for more information.

Download 'CE_assignment2.zip' from https://www.iaik.tugraz.at/ce and open the cryptoprocessor project using Vivado 2019.1. In this exercise you will implement an AES unit and make two new instructions for the cryptoprocessor.

Please refer to the Youtube channel of this course for viewing the 'class lecture' explanation of the assignment: https://www.youtube.com/watch?v=13fYqrm2dXc

Task 1 (Total 15 points)

Implement an AES encryption/decryption unit for 128-bit key in the FPGA. The unit should be able to perform both encryption and decryption operations.

Your AES-128 encryption/decryption implementation will be within the module AES128( ) within the ComputeCore( ) module. You are not allowed to change the interface of the module.

```
module AES128(input             clk,reset,
              input             enc_or_dec, // 0 for enc, 1 for dec
              output     [9:0]  addr_rd,addr_wr,
              output            wen,
              input      [63:0] data_in,
              output reg        done,
              output reg [63:0] data_out);
```

There are two new instructions for AES-128 encryption and decryption, INS=19 and INS=20 respectively. For encryption and decryption operations, *'enc_or_dec'* input of AES128( ) module will be set as 0 or 1, respectively. This is already implemented in the ComputeCore( ) module. Similarly, instruction-decoding and communication-related connections are already implemented in the ComputeCore( ) module. The flow of computations is described as follows:

1. When the Cryptoprocessor wants to perform AES encryption/decryption, we first write plaintext/ciphertext and round keys into the memory. Two-words long (hence 128 bit) plaintext/ciphertext block should be written into the memory locations OP1+0 and OP1+1 where OP1 is a part of the instruction specifying the memory address offset for the input data. The 22-words long round keys should be written into the memory locations from OP1+2 to OP1+23.

   In SDK code, OP1 is currently set as 0 by default.
   (Note that an AES block is of size and the memory of the cryptoprocessor has 64-bit word size. Hence, one block occupies two words in the memory.)

2. Then the SDK will send the instruction, which is a 35-bit long string. It consists of (from least significant bit to most significant bit) a 5-bit INS (instruction code), a 10-bit OP1 (operand 1 address), a 10-bit OP2 (operand 2 address) and a 10-bit WT (result address).

   For this assignment, INS should be 19 or 20 for encryption or decryption respectively.
   The OP1 is the memory address of the input data (plaintext followed by round keys).
   The OP2 currently is not used.
   The WT is the memory address offset where the output data will be written.

   For example, an instruction with INS=19, OP1=0, OP2=0 and WT=24 will read the input operands (plaintext and round key words) starting from address 0, then perform one AES-128 encryption and finally write the result (two 64-bit words) in address 24 and 25.

   In the given SDK project, this instruction part is already implemented.

3. When the AES128( ) module is enabled (by making *reset*=0), it will need to read the input data from the memory of the cryptoprocessor. For reading a word from the memory, the AES core should use the '*addr_rd*' port to send the *offset* read address to the memory of the cryptoprocessor. If a read-address is sent in the *i*-th cycle, then the read-data will be available in the '*data_in*' port in the (*i*+1)-th cycle.

   As '*addr_rd*' is an offset read address, it will automatically get added to OP1 to produce the physical address for the memory. E.g., when you want to read
   -- the lowest 64-bits of a plaintext block, just set addr_rd=0.
   -- the highest 64-bits of a plaintext block, just set addr_rd=1.
   -- the lowest 64-bits of the first round-key, just set addr_rd=2.

4. The output result, which is 128 bits or 2-words long, should be written in the memory in the following steps.
   -- addr_wr=0 with wen=1 for writing the lowest 64 bits of the result.
   -- addr_wr=1 with wen=1 for writing the remaining 64 bits of the result.

   Here add_wr is an offset write address. It will automatically get added to WT to produce the

physical address of the memory. When ‘wen’ is 1, the 64-bit data in the ‘data_out’ bus gets written in the memory location with the address (WT + addr_wr).

You may have a look at the recorded class lecture https://www.youtube.com/watch?v=13fYqrm2dXc where this exercise was explained.

In Task-1, you need to implement the AES128( ) module and verify its correctness using the C++ implementation of AES in the SDK part. Your AES hardware should produce the same output as software implementation in SDK.

There could be various optimization directions in your design, e.g., low area, or high speed. Choose your optimization goal and implement accordingly.

Task 2 (Total 5 points)

Implement the Cipher Block Chaining (CBC) mode (see Fig.1) in the SDK using a hw/sw co-design methodology. It must use the AES core that you have designed in Task 1.

The AES-128 encryption/decryption part will be performed using the cryptoprocessor while the rest of the operations will be performed in the software. You should modify the SDK code to provide CBC mode functionality. The length of the chain should be a variable and all blocks should use the same keys.

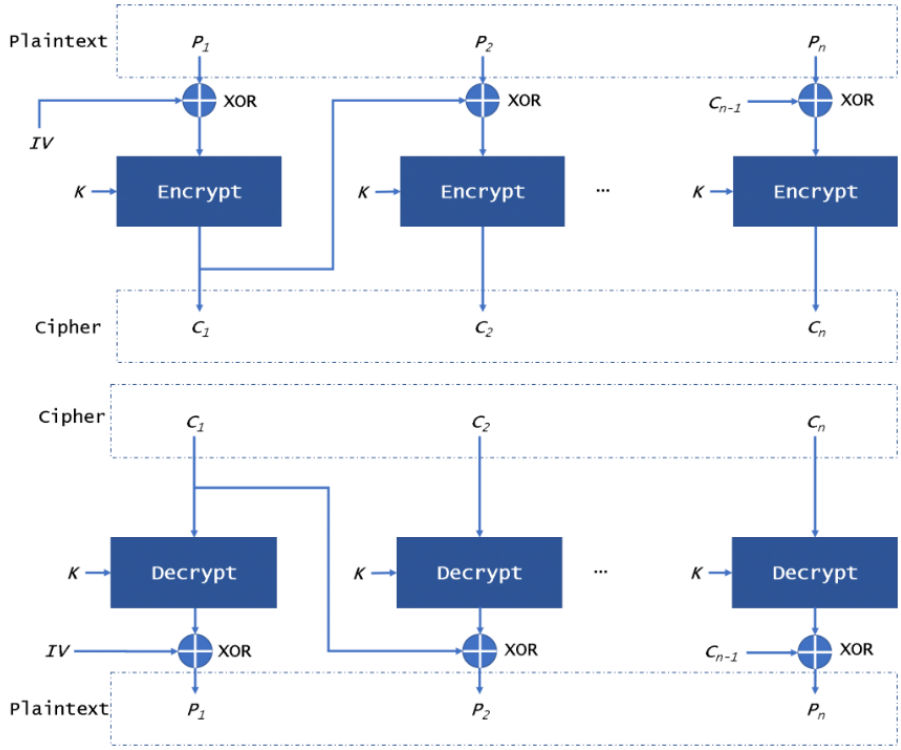

Fig.1. CBC Mode of AES*

* Image is retrieved from https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/

Submission guidelines

- The deadline for submitting your project is 29th November 23:59.
- You will upload your project to the git repository of your team.

Marking scheme

Task 1 (15 points)

- You get 10 points in Task 1 if you (1) have a working implementation satisfying the requirements and (2) you defend your work successfully.

  Otherwise you do not get any points for Task 1 overall.

- You get 0 to 5 points based on how well your implementation of Task 1 performs (in terms of resource requirements, speed) compared to the implementations of other teams and our own reference implementation.

Task 2 (5 points)

- You get any points for Task 2 only if you get points from Task 1.
- You get 5 points in Task 2 if you (1) have a working implementation of CBC mode of AES-128 and (2) you defend your work successfully.

  Otherwise you do not get any points for Task 2 overall.