AES-128 Encrypter/Decrypter in VHDL

What the standard is

The Advanced Encryption Standard (AES) encryption standard set forth by the National Institue of Technology, serves as a standardized method to complete block ciphers using a known algorithm that permutes data with a key.

In order to maintain data security, a reliable algorithm must be used to scramble data in a way difficult to decipher without the key. The AES outlines that algorithm with a series of discrete, non-singular permutations that allow data to be encrypted and decrypted reliably, along with additional procedures to generate pseudo random cipher text.

Understanding the process

The process of encryption is done using a series of arithmetic operations, all using Galois Field arithmetic (GF) in the finite field GF().

Beginning with the encryption process, the cipher data is obtained by completing ten rounds of mixing (encrypting) data with the provided key. Each round uses a different key. Each of the ten keys required are derived from the original provided encryption key.

The data is XOR’d with the original key, then 10 loops of encryption are enacted. In each loop, that data is broken into 16 bytes. Each byte is originally substituted with an inverse value equal to its inverse. Following substitution, the data is intermixed in a pattern equivalent to placing it in a matrix and then shifting its rows. Finally the data is further altered by “mixing columns” or conducting matrix multiplication with a fixed matrix.

Decryption is an identical operation to encryption, but with the rounds and functions run in reverse order.

The AES also outlines a procedure for generating pseudo-random data for chains of data blocks to prevent repeat patterns. The procedure stores the cipher of the preceding block as an initializing vector to be XOR’d with the current block of data. This ensures repeat data blocks are all encrypted uniquely, based on the data that precedes it.

Design constraints

Accomplishing this task can lead to designs that significantly vary in specification. The specification constraints for this project include memory, clock range, simulated delays, and interfacing pins.

The design needs able to function within a clock frequency range of 10MHz to 500MHz. It also must utilize no more than 200 8,192 bit memory units with a bus width of 64, and must have no more than 100 inputs and outputs.

Solution

Given that there the design must adhere to the constraint of only consisting of both and input and output bus of only 32 bits (4 bytes), four clock cycles will be needed for ingesting the full encryption key, the full data, and full set of initial values. As well, output of cipher data or decrypted original data will also consume four clock cycles by outputting one of the four bytes at a time.

Deviations from constraints

While the initial project constraints call for independent entities for encryption and decryption, for the sake of resource efficiency, consolidating both functions into a single entity is ideal. As such, the decision was made to use a single entity and consolidate the signals for encryption and decryption at an early point in the project's planning.

Additionally, an input was added to select between encryption and decryption, and the EBC\_mode input was removed and the CBC\_mode pin was implemented with dual functionality (toggling between EBC and CBC modes).

Behavioral Model

The behavioral model serves as the gold standard for the design, as it is not concerned with clock accuracy, memory management, or structure.

While the ingestion and output of data required four clock cycles per individual piece of data, the entire process of generating round keys and calculating the cipher data (or original data) can be calculated in only a single clock cycle. Unfortunately, though it is possible to structure the model in this way, for unknown and untested reasons, our current model requires three clock cycles between conclusion of the ingest, and the initiation of outputting the calculated data.

The primary entity is structured as a state machine that re-uses states for operations that require four clock cycles. Within each of the states, logic to determine the intended function is utilized. States for reset and start are also accounted for.

Functions that handle most of the logic of the key generation and encryption are stored in the accompanying function package. Additionally, round keys are stored as a two-dimensional array of std\_logic\_vector.

Behavioral Testing

The included testbench includes tests for the behavioral model that cover several operations. Encryption of single blocks of data, “chained” blocks of data (CBC mode), as well as decryption of blocks are all included. A few hard-coded data examples are written in-line within the test bench, while most of the test data is pulled in a loop from the accompanying aes\_128\_test\_package file.

Dataflow Model

While dataflow may be a loosely defined term, the objective of a dataflow model is to be clock cycle accurate. As such, clock cycle timing of the ingest and output of data should be identical. Changing from the behavioral model is the clock cycle accuracy of the key generation and encryption/decryption process. Use of loops and variables is also abandoned for further state machine assignments and signals.

Dataflow Design Information

Breaking apart the responsibilities of multiple entities is the most significant design change between the behavioral and dataflow model. The primary entity in the dataflow model is only responsible for any actions that interact with the external interfaces (or buses). All internal work is delegated down to the encrypter\_decrypter entity. This entity functions as another state machine that works through the process of generating round keys, and passing along the round keys as well as they input data to the blackBox entity.

The blackBox entity is responsible for using combinational logic to complete the cipher or decipher. The block entity generates each of the functions to mix columns, shift rows, and substitute, all in parallel. By doing so, the entire cipher or decipher process can occur in a single clock cycle without an state changes required. In initial testing, this process required approximately 5ns to complete.

Dataflow Testing

Similar to the testing of the behavioral model, single block encryption, “chained” block encryption, as well as decryption are all tested. The differing factor here is the clock cycles. As it no longer takes a single cycle (or three cycles in our case), to complete the encryption/decryption process, each test will require more clock cycles.

In this model's testing, it appears that the number of cycles between completing ingest and beginning out is seventeen clock cycles. In total the number of clock cycles required to complete a block of encryption or decryption may take up to 31 cycles to receive the entire solution across the 32 bit bus.

{Other sections here}

Further Work

Given further time for completion various tests and features could be added, as well as optimizations. Tests to be added may include reset at different points in the process for each level of the design. Forms of error checking could also be added with a fault state to fall into if an error is detected.

{Fill in more here}