EE4218 Major Project AES

DOU YOUZHE A0100229X & GUO YAWEN A0100892L

This project requires to perform AES decryption in one C version and two VHDL version in order to compare the performance. The VHDL versions are expected to be much faster than the C version while using more resources from the FPGA. The two VHDL versions will be compared to each other for their timing and resources. Pareto curve will be used to predict the best combination of performance and resourced used.

AES Background

The National institute of Standards and Technology (NIST) adopted Advanced Encryption Standard (also known as Rijindael) as a Federal Information Processing Standard for block-cipher algorithm. It provides high portability and reasonable security.

The input and output for the AES algorithm consists of sequences of 128, 192 or 256 bits. In this project, we use both key and input of 128 bits and perform only the decryption of a ciphered image. The image is firstly processed by an encoding software and a sequence of 30720 byte of information is produced. Each 128 bits is taken and arranged into a 4*4 block where each cell contains 8 bits of data. Several steps are performed to this block of data, including add round key, inverse shift row, inverse mix column and inverse sub-byte. Different round key is generated for each round through key expansion algorithm.

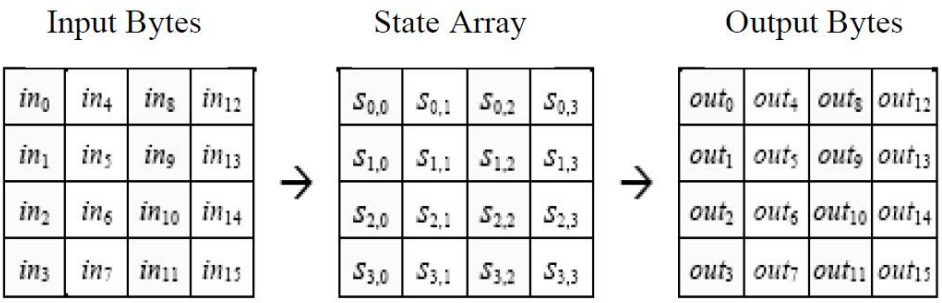


Figure: State Array Input and Output

Key expansion:

1. The 4th column of i-1 key is rotated such that each element is moved up one row.
2. Each cell is passed through S-box (perform forward sub-byte).
3. The 1st column of ith key is produced by XOR the result from step 2 with the 1st column of i-1th key as well as a constant (Row constant).
4. The 2nd, 3rd and 4th row is generated by XOR the previous column from the ith key with the respective column of i-1th key.
5. The entire process continuously iterate to generate all keys for 10 round and stored.

Add round key:

The cipher text is XOR-ed with the round key. The inverse add round key process produce the same result as the forward add round key process because XOR the same bit twice will give you the original value.

Inverse shift row:

This step rotate i^{th} row by i-1 elements rightwards.

Inverse sub-byte:

This step replace each cell in the block with content from the inverse s-box. Each 8 bits of data is grouped and performed substitute through a constant s-box.

Inverse mix column:

The inverse mix column process treat each column as a polynomial over Galois Filed (2^8) and then multiplied modulo x^4+1 with a fixed inverse polynomial is $c^{-1}(x) = 11x^3+13x^2+9x+14$.
The following matrix is used for this process. For faster process, the results of all combination are generally computed before and stored in a look up table.

The decryption process is the exact inverse of the encryption process with the same set of keys. The flow of the decryption is shown below.

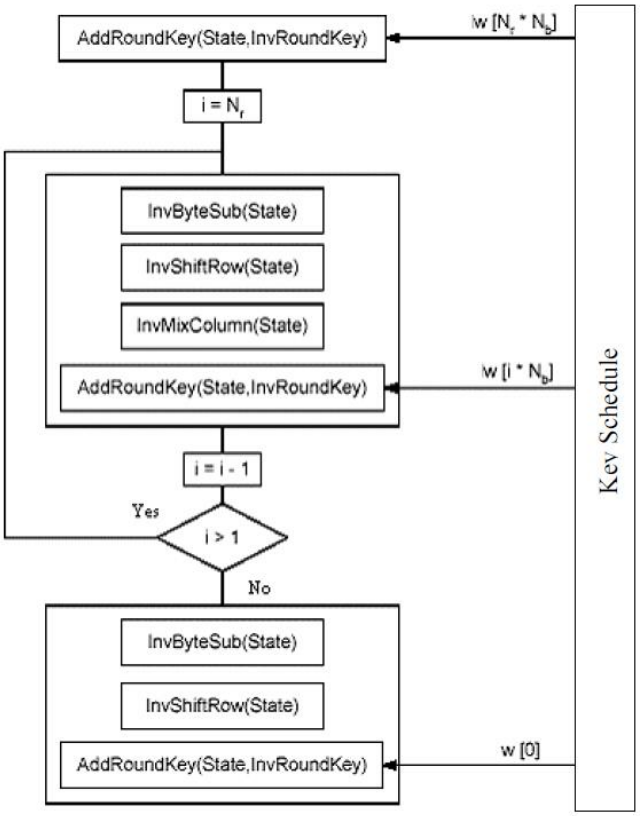


Figure: Decryption Process

C Implementation

The C implementation is written in C language in SDK. There are three parts in C implementation, including receiving the data using UART, decryption cipher text and displaying the image through VGA using TFT controller.

- A Zynq-7000 processor and TFT controller are created by adding IP in the block design.
- Function XUartPs_RecvByte() is used to receive data byte by byte from PC.
- The C program is designed to collect, process and display 16 byte each cycle. Each two byte is assigned for 5 pixels and the last bit is unused. A total loop of 30720/16 times will be needed for the entire process.

```
for(k = 0; k < 16; k++){
        ecodedimage[k] = XUartPs_RecvByte(XPAR_PS7_UART_1_BASEADDR);
}
for(k = 0; k < 8; k++) {
        two_byte_for_color = (((u16)ecodedimage[2*k]<<8)) | ecodedimage[2*k+1];
                for(pixel = 0; pixel < 5; pixel++){
                                color = two_byte_for_color & (mask << (12 - pixel*3));
                                color = color >> (12 - pixel*3 + 1);
                                RGB = 0x0;
                                if(color & 0x00000004)
                                  RGB = RGB | 0x00FF0000;
                                if(color & 0x00000002)
                                  RGB = RGB | 0x0000FF00;
                                if(color & 0x00000001)
                                  RGB = RGB | 0x000000FF;
                                col = j*40 + k*5 + pixel;
                                XTft_SetPixel(&TftInstance,col,i,RGB);
                }
}
```

VDHL Implementation Version 1 (128 bits)

The first version process 128 bits of data every time. It uses the least amount of resources while the processing time is expected to be longer. Since AXI-Stream (AXIS) channels are dedicated 32-bit point-to-point communication interfaces, we need four clock cycles to read or write one block of data. The first 128 bit read is the original key which will be passed to the key schedule component and generate ten more sets of round keys. The 128 bits block is then passed through the decryption process and the output will be produced after 59 clock cycles. The decryption process is written in Finite State Machine and an internal signal called Round_number is used for the control logic to decide the next state.

The system is in Idle state by default. When the data is ready, S_AXIS_TVALID will become one and the system will go to Read_Input state. After four clock cycles, 128 bits of key is stored and the state will be Key_Schedual. In Key_Schedual state, 10 sets of round key is generated and the Boolean signal HasKeyBeenGenerated will be true. Then the first block of data is read and the system will go to AddRoundKey for the pre-round. Then the system will loop InverseSubByte, InverseShiftRow, InverseMixColumn and AddRoundKey nine times and the Round_number will increment by 1 in each

loop. At the tenth round, the Round_number will be 10 and the system will jump from directly from InverseShiftRow to AddRoundKey (exclude InverseMixColumn in round 10).

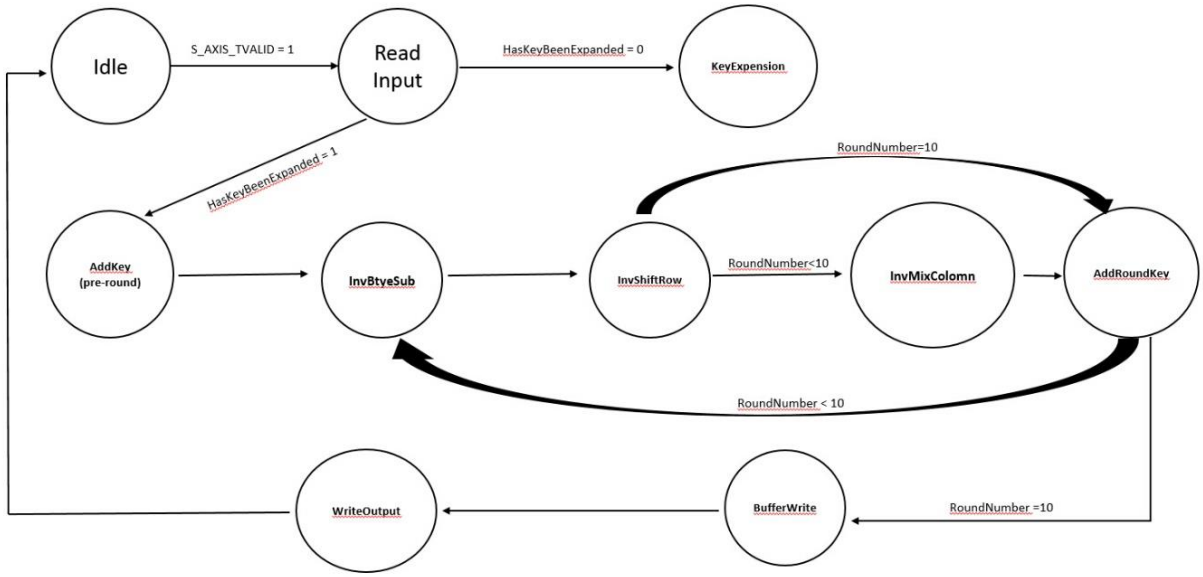
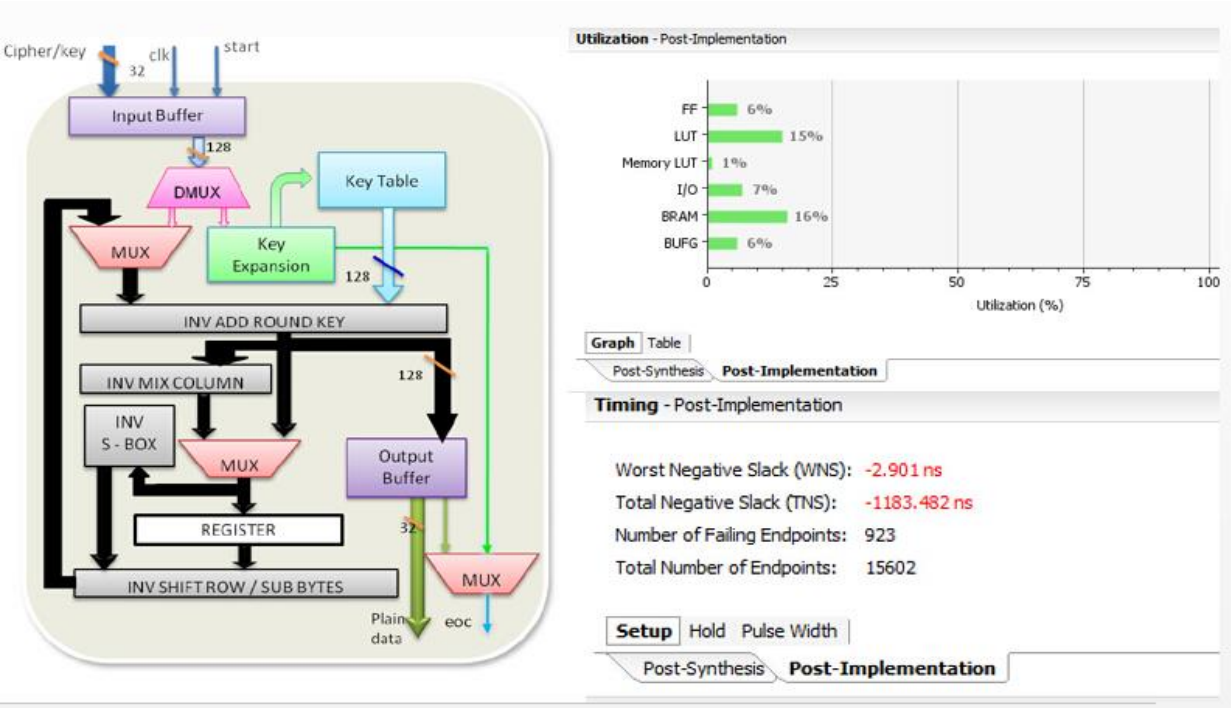
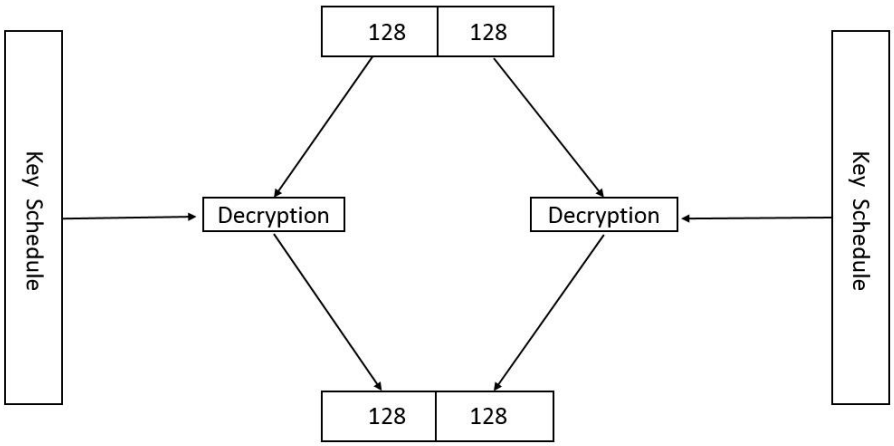


Figure: State Machine

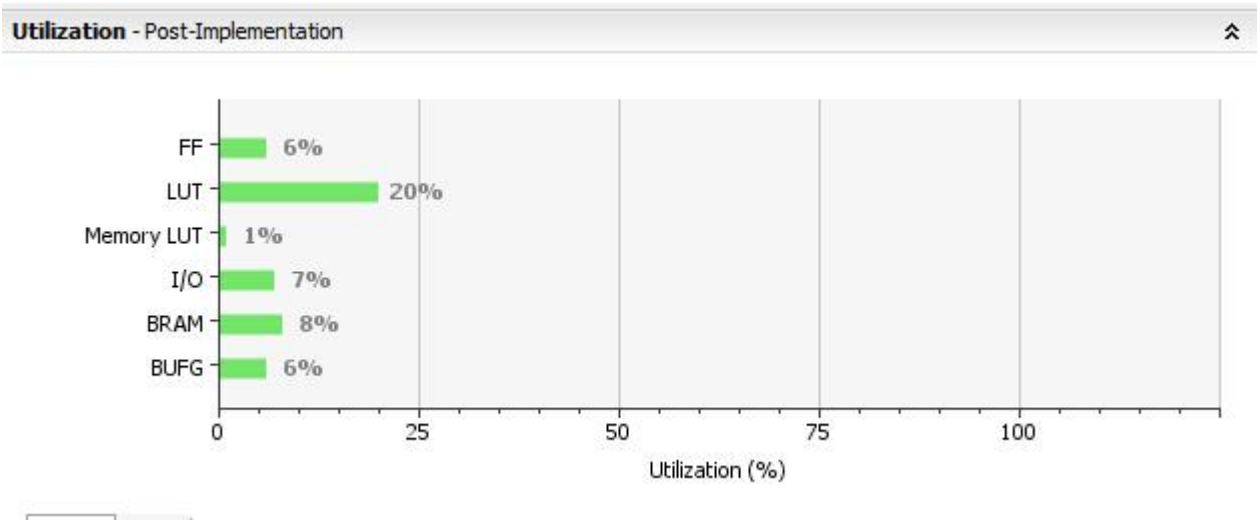
A buffer state is created before Write_Output stage because the first 32 bits will be written when M_AXIS_TVALID change value. Without the buffer, the first 32 bits is always missing and the remaining last 32 bits will distort the data for next 128 bits.



VHDL Implementation Version 2 (256 bits)



This version of VHDL process two set of 128 bits in parallel. Therefore, it is expected to finish the decryption process two times faster while utilizing double the resources. From the previous design, the Worst Negative Slack (WNS) is negative. This means that in some state, the time for computation is longer than one clock cycle for the worst case. In order to reduce that slack, logic need to be simplified. In this design, all states are listed manually and control logic is not needed.



Timing and Resource Comparison

XScuTimer functions are used to measure the number of ARM clock cycle for one process. One ARM clock cycle is 10 ns. The results are shown below.

	LUT	FF	No.of ARM clock cycle for one process	No. of processes	Total Time
128 Version	15%	6%	2100	1920	$4.032*10^7$ ns
256 Version	20%	6%	2200	960	$2.112*10^7$ ns
C Implementation	2%	2%			5.1s