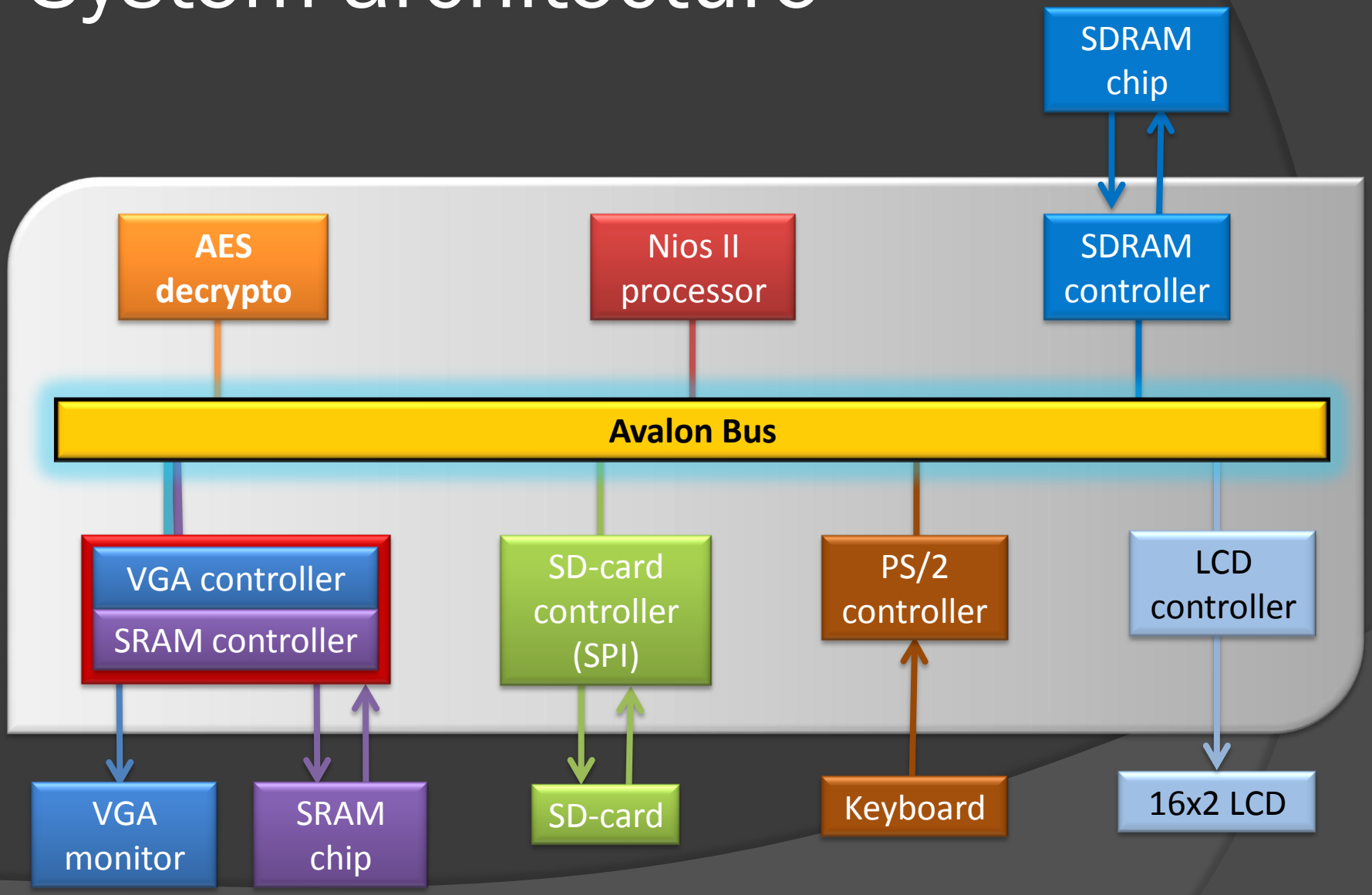
Shrivathsa Bhargav
Larry Chen
Abhinandan Majumdar
Shiva Ramudit

FPGA BASED 128-BIT AES DECRYPTION

System architecture

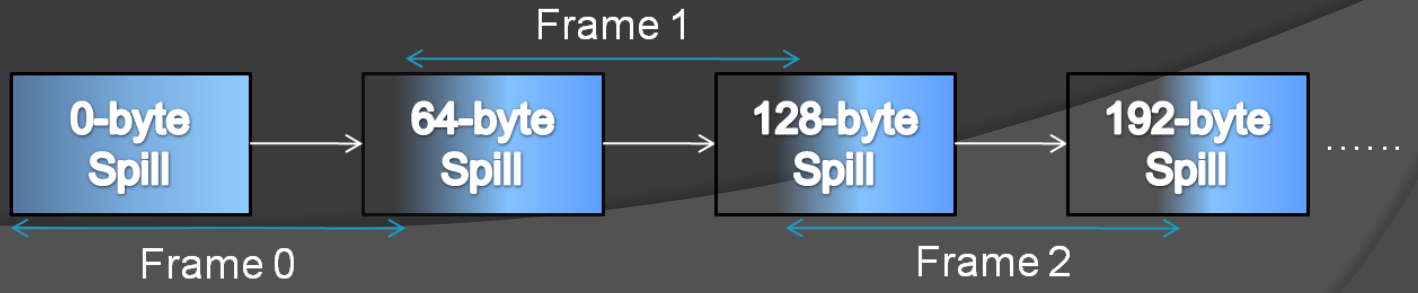


SD-Card SPI Interface

- The SD-Card SPI interface communicates with the MMC/SD card via SPI protocol
- The SPI interface interacts with the card through a sequence of commands such as reset, initialize, set block length, and data read request
- This interface was difficult to simulate and debug since the MMC/SD card protocol is proprietary
- Modified Professor Edwards' SPI interface implementation from APPLE2FPGA

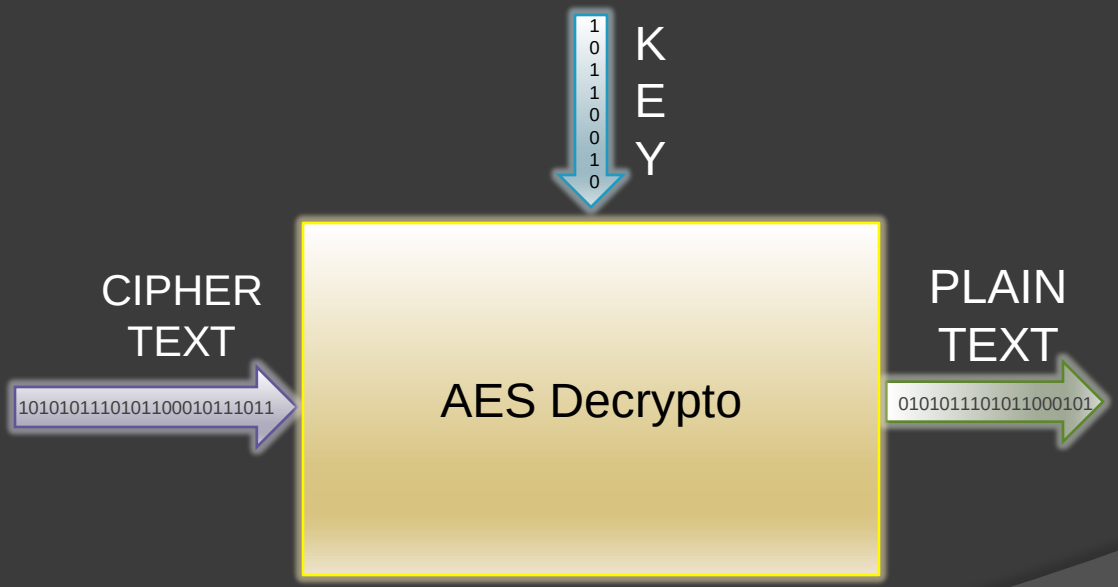
SD-Card SPI Interface

- Increased compatibility
  - Applied a patch to send additional pulses to the SD to wake it up
  - Increased wait clock cycles to successfully read consecutive blocks of data
- Increased performance
  - Set block length to 512-bytes and correspondingly sized buffer to avoid issuing unneeded number of data read requests
- Reduced duplicate reads
  - Issuing 512-byte block reads causes buffer spill for consecutive frames
  - A single frame is 77888 bytes, which is not divisible by 512-byte blocks
  - A check in software is implemented to monitor the frames and offset it by 64*(frame % 8) to read the correct data contents
  - The spill will be multiples of 64-bytes, and it will takes 512-byte/64-byte = 8 spills to go back to a 0-byte spill block

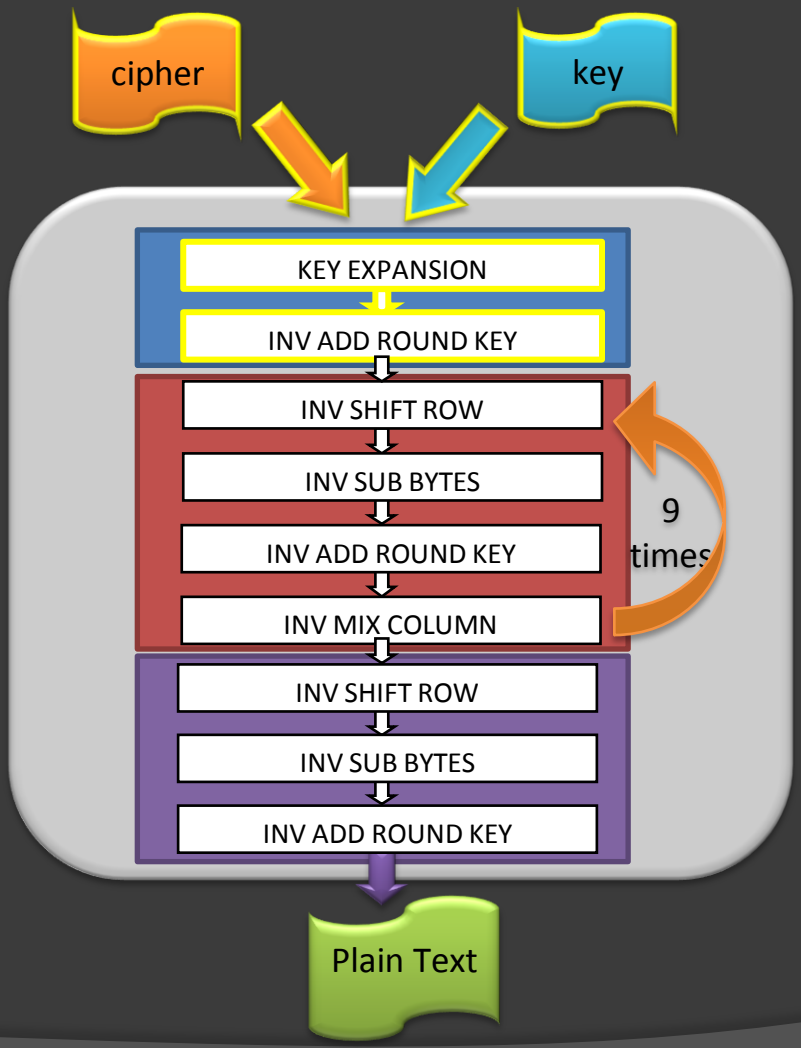


AES Decryption

- AES (Advanced Encryption Standard) Decryption is a Symmetric Key Cryptographic Algorithm that accepts the cipher text and the key as input, and generates original text as output

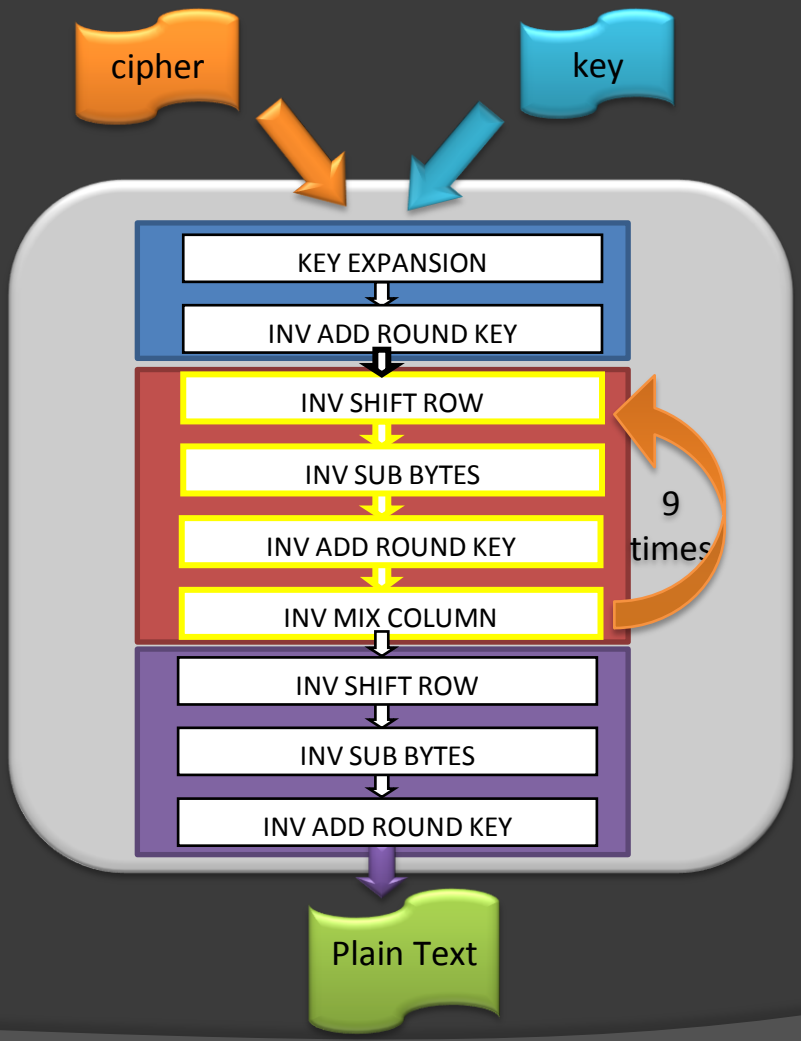


AES Decryption Algorithm



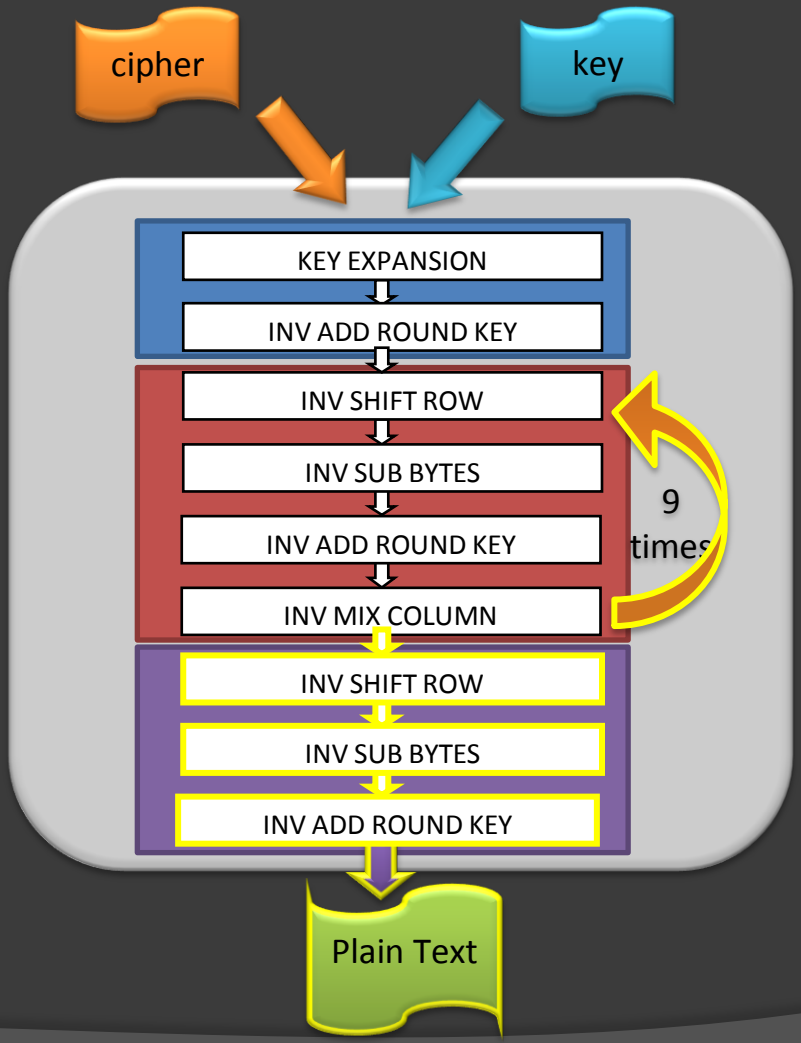
- Key Expansion
  - Generates Intermediate Keys required for each iteration
- Inv Add Round Key
  - XORs the generated key for that particular iteration with the cipher text

AES Decryption Algorithm



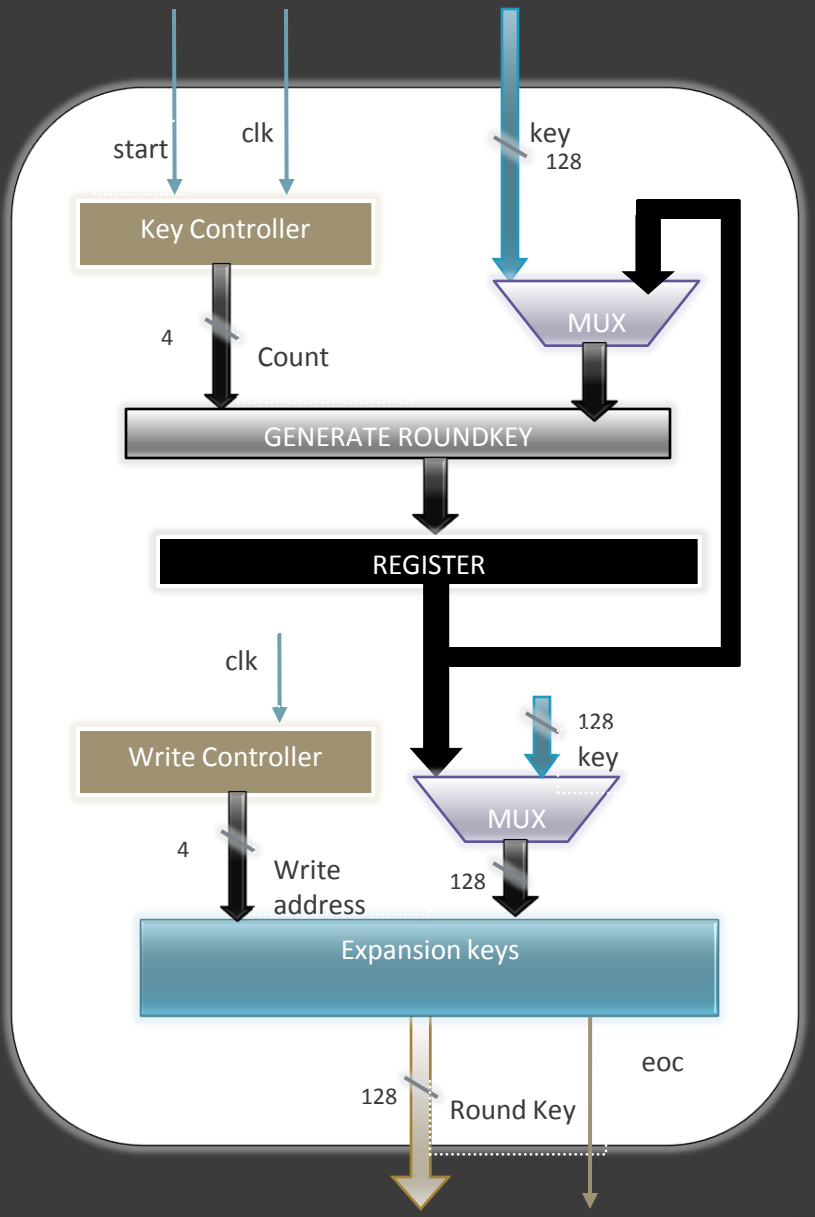
- Inverse Shift Row
  - Shifts each i^{th} row by i elements to the right
- Inv Sub-bytes
  - Replaces each element by corresponding entry from inverse s-box
- Inv Add Round Key
  - XORs the generated values by corresponding intermediate key to that iteration
- Inv Mix Column
  - Performs modulo multiplication with MDS matrix in Rijndael's finite field

AES Decryption Algorithm



- Repeats these four steps for 9 iterations
- As a last iteration, it does inverse shift row, inverse sub-bytes and inverse add round key
- Final output is the plain text

AES Key Expansion– RTL Design

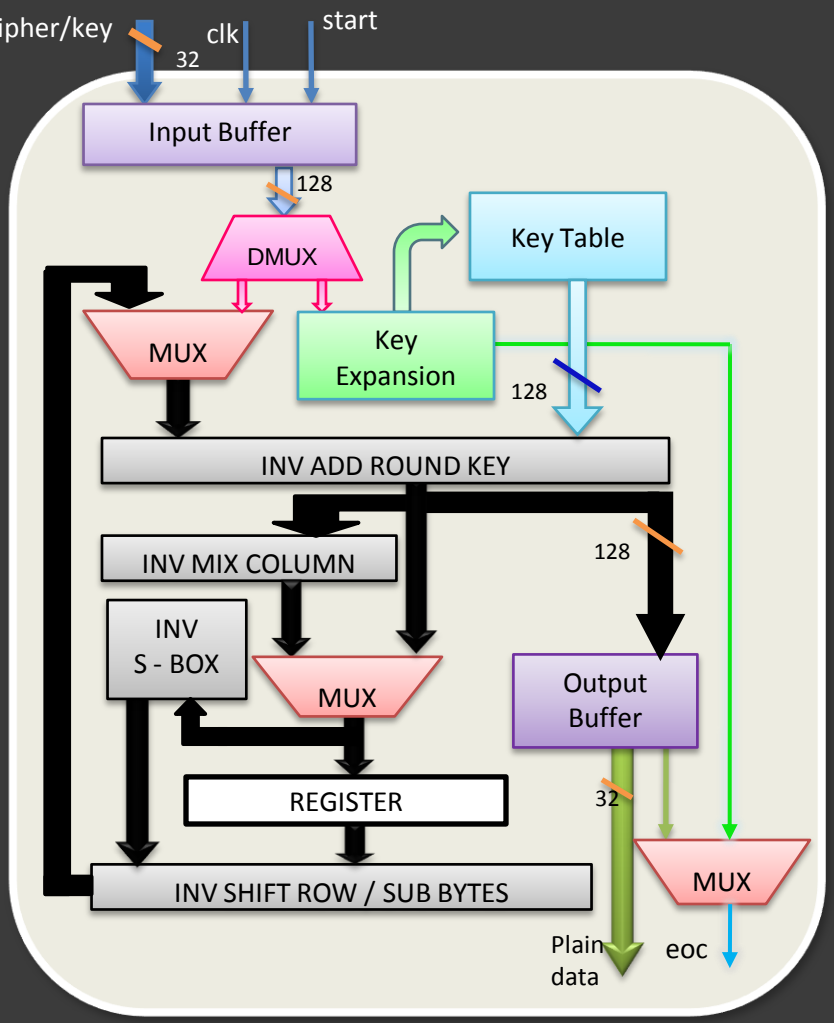


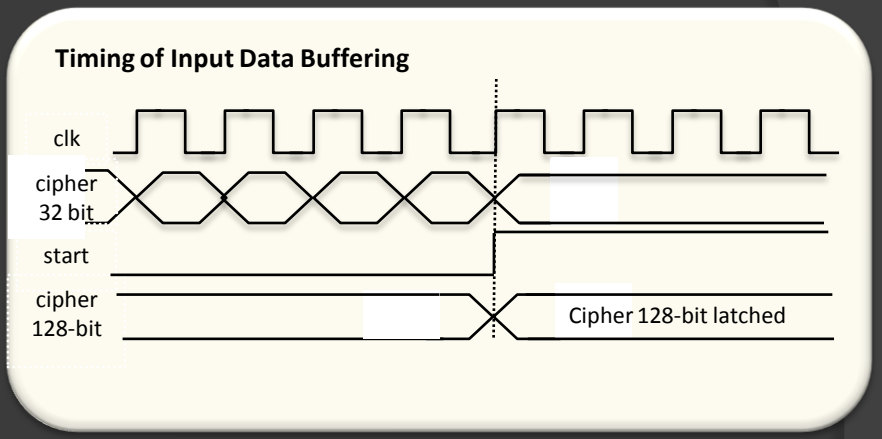
Key expansion required to generate the roundkeys required for each round of encryption

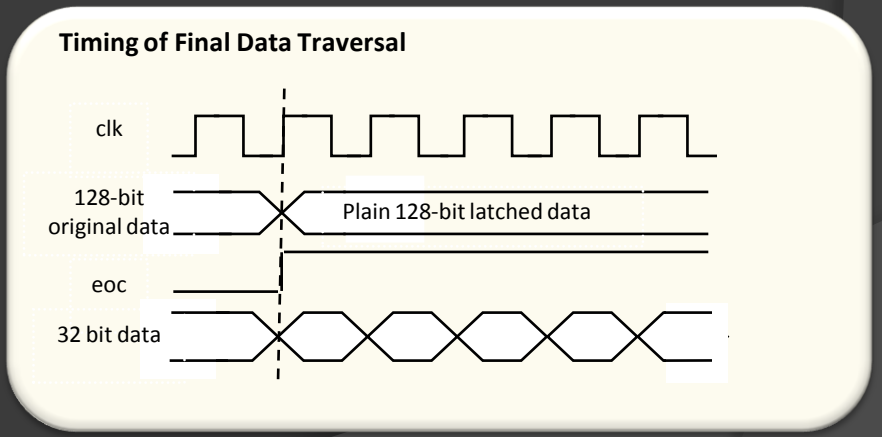
Generate roundkey module contains all combinational logic to perform the key expansion algorithm

Takes 11 clock cycles to generate the 10 roundkeys

AES Decrypto – RTL Design



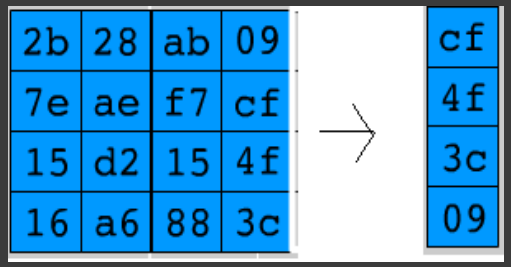




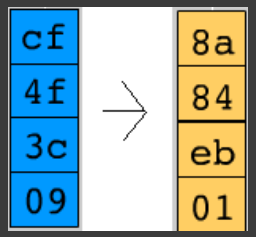
Takes 10 clock cycles to generate the plain text. Runs at 88.31 MHz and occupies 17% of the FPGA Logic Elements.

AES Key Expansion Algorithm

The algorithm for generating the 10 rounds of the round key is as follows: The 4th column of the i-1 key is rotated such that each element is moved up one row.

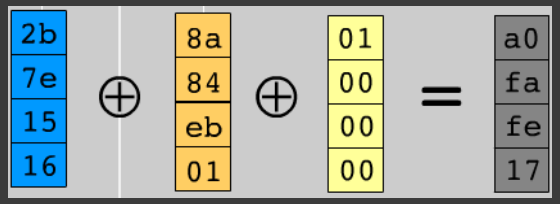


This result goes through forwards Sub Box algorithm which replaces each 8 bit value of this column with a corresponding 8-bit value.



AES Key Expansion Algorithm

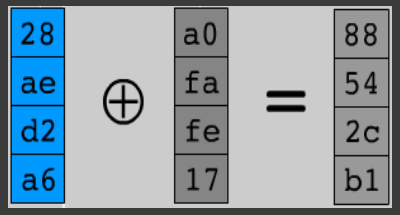
To generate the first column of the i^{th} key, this result is exclusive-or-ed with the first column of the $i\text{-}1^{th}$ key as well as a constant (Row constant or Rcon) which is dependent on i.



01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

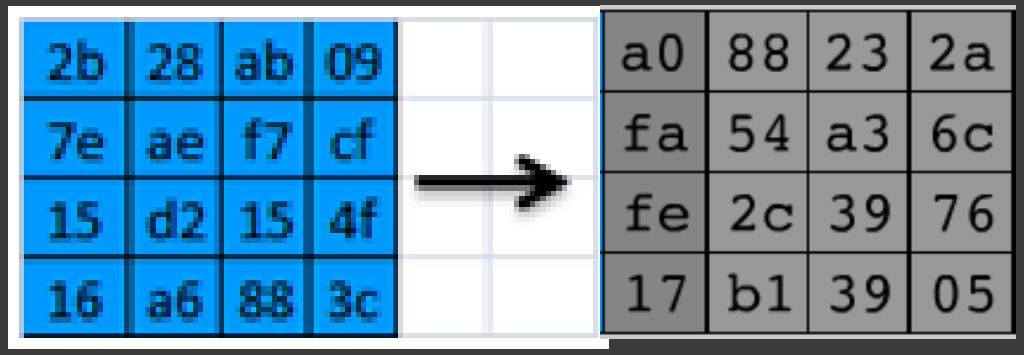
Rcon

The second column is generated by exclusive-or-ing the 1st column of the i^{th} key with the second column of the $i\text{-}1^{th}$ key.



AES Key Expansion Algorithm

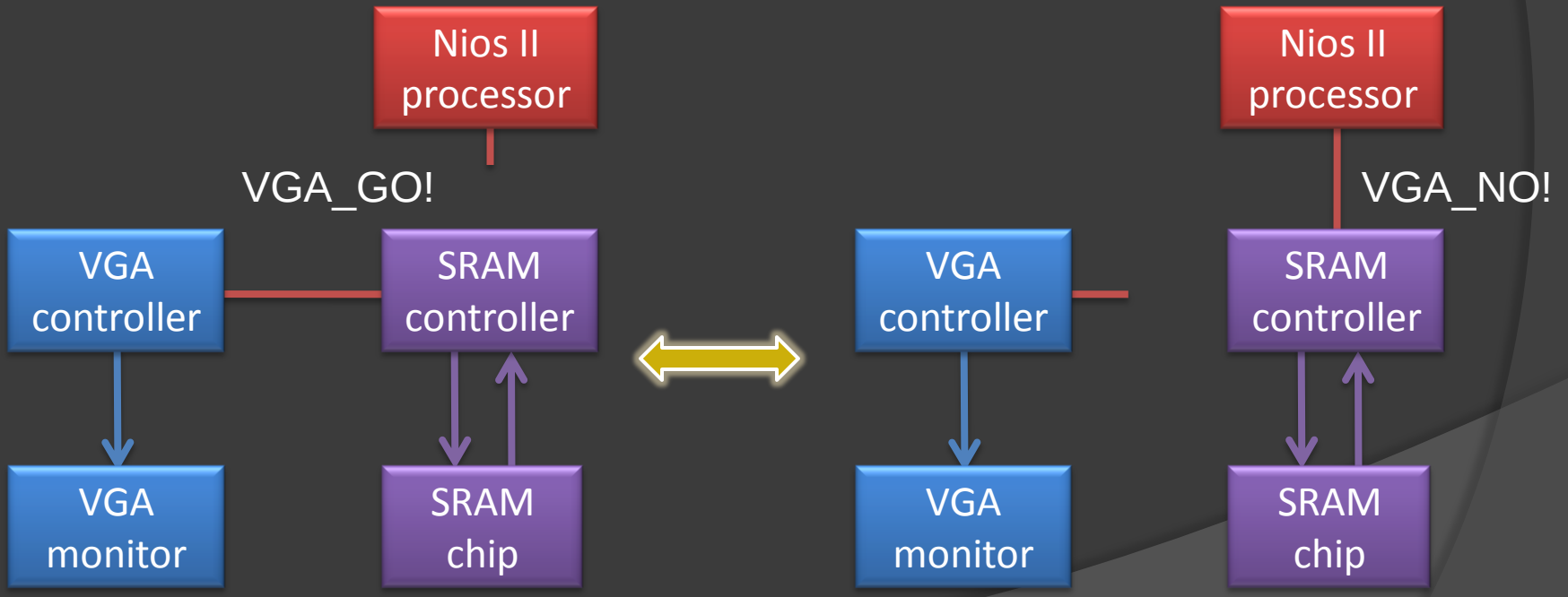
This continues iteratively for the other two columns in order to generate the entire i^{th} key.



Additionally this entire process continues iteratively for generating all 10 keys. All of these keys are stored statically once they have been computed as the i^{th} key generated is required for the $(10\text{-}i)^{th}$ round of decryption.

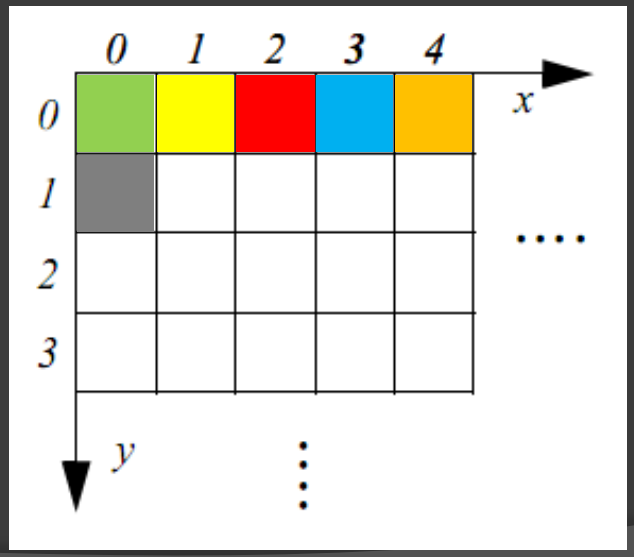
SRAM controller

- Single-ported SRAM poses a problem
- Had to devise a GO/NO switch (Mux)



VGA controller

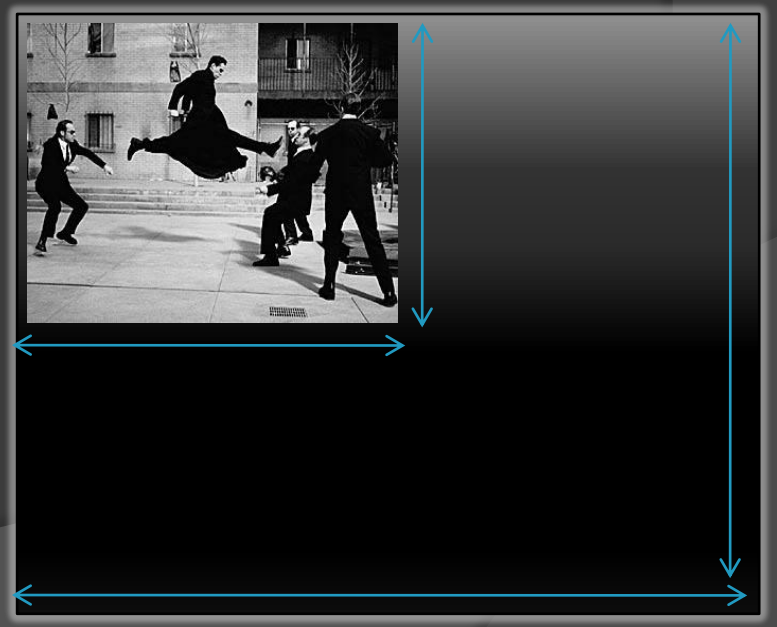
- Bitmap specs
  - 1078-byte header, 8-bit depth, flip row order
- Forcing grayscale (R=G=B=data)
- Address calculation



16-bit/512k SRAM	
Byte 0	Byte 1
10110101	01100100
10000011	10010010
01010011	01001100

VGA controller

- Reading VGA draw location constantly in software
- Writing into SRAM only when outside “rectangle”
- Reduced fps from 8.5 to 6!



Summary

- Results
  - 32% LE, 14% Memory, 3.74 Mbps throughput
- Lessons learned
  - Technical knowledge
  - Hardware behaviors are difficult to visualize without simulations
  - Code reuse saves time and effort to design and debug
  - Start early; Work on modularized tasks parallelly *and* concurrently
- Original goals superseded by **video**
- Future work
  - Color video (there's enough memory)
  - Higher frame-rate (overclock system)
  - Double-buffering to remove scan lines