AES Encryption and Side Channel Key Extraction

Alexander Ingare, Hamza Iqbal

Project Responsibilities

Name	Responsibilities
Zander	AES Encryption & Decryption ● Software and hardware implementations of EBC AES ● Hardware optimizations
Hamza	Power Monitor & Key Recovery ● Hardware Implementation of ring oscillator based power monitor ● Power trace collection and analysis in software

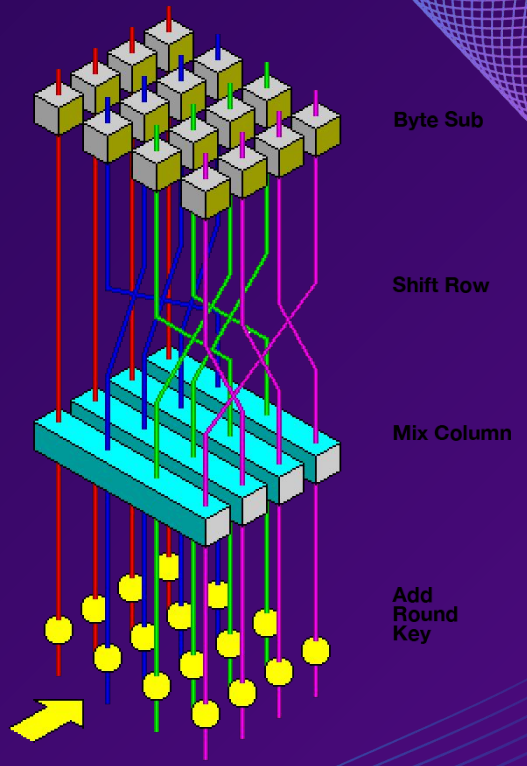
Advanced Encryption Standard - Overview & Features

- Standard encryption scheme for U.S. classified information (and sensitive data worldwide)
  - Adopted by the National Institute of Standards and Technology (NIST) in 2001
- A variant of the Rijndael block cipher (family of symmetric-key ciphers)
  - Rijndael: key and block sizes are multiples of 32 bits, ranging from 128 to 256 bits
  - AES: constant block size of 128 bits, 3 different key sizes (128, 192, 256)
- Built on the substitution-permutation network design principle
- A block is a 4x4 column-major ordered array of the 128-bit block (16 bytes)
- The larger the key, the more transformation rounds required

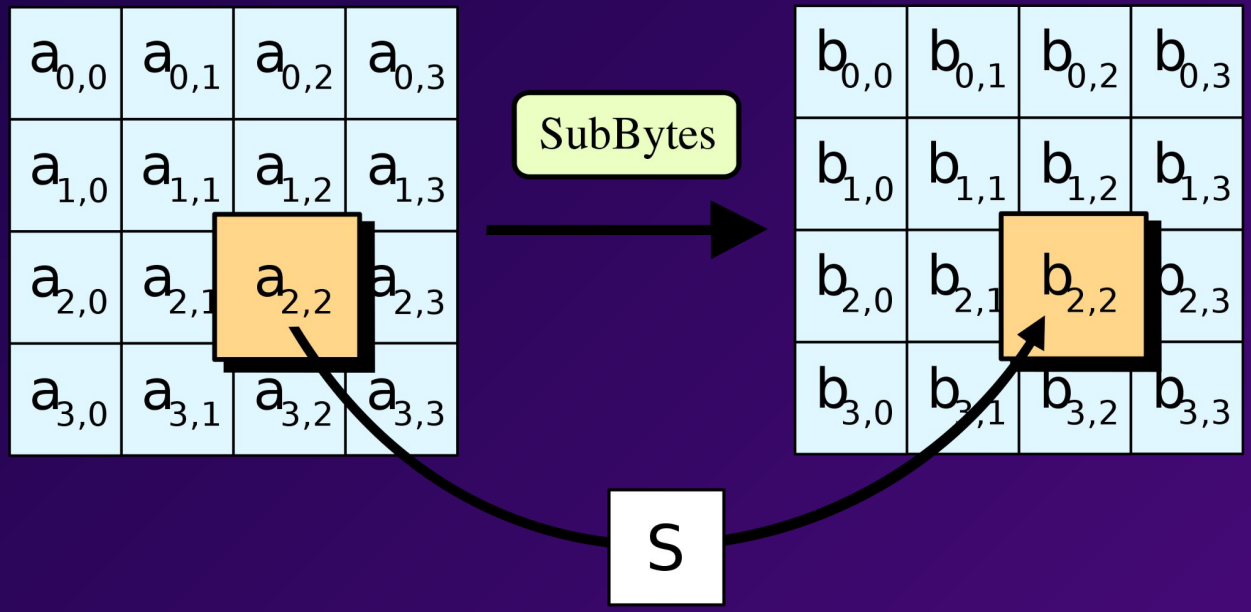
$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

High-Level Algorithm

- Key expansion
- Initial round key addition (AddRoundKey)
- Initial transformation rounds (9, 11, or 13 rounds)
  - SubBytes
  - ShiftRows
  - MixColumns
- Final transformation round
  - SubBytes
  - ShiftRows
  - AddRoundKey

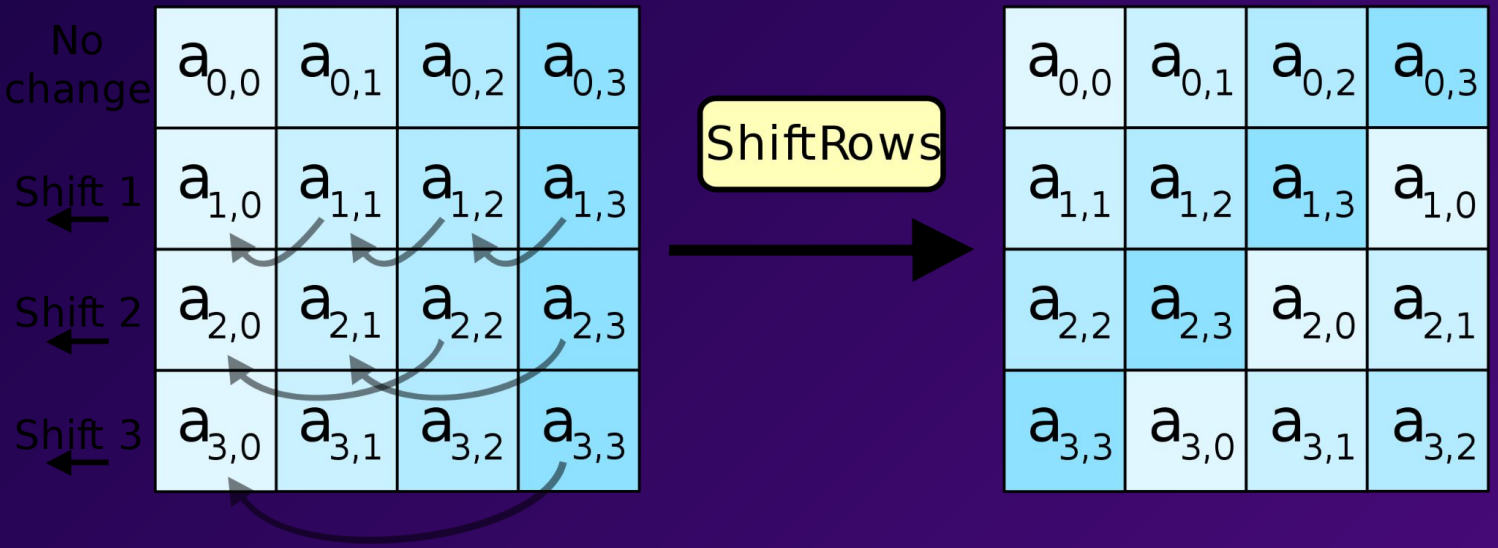


SubBytes



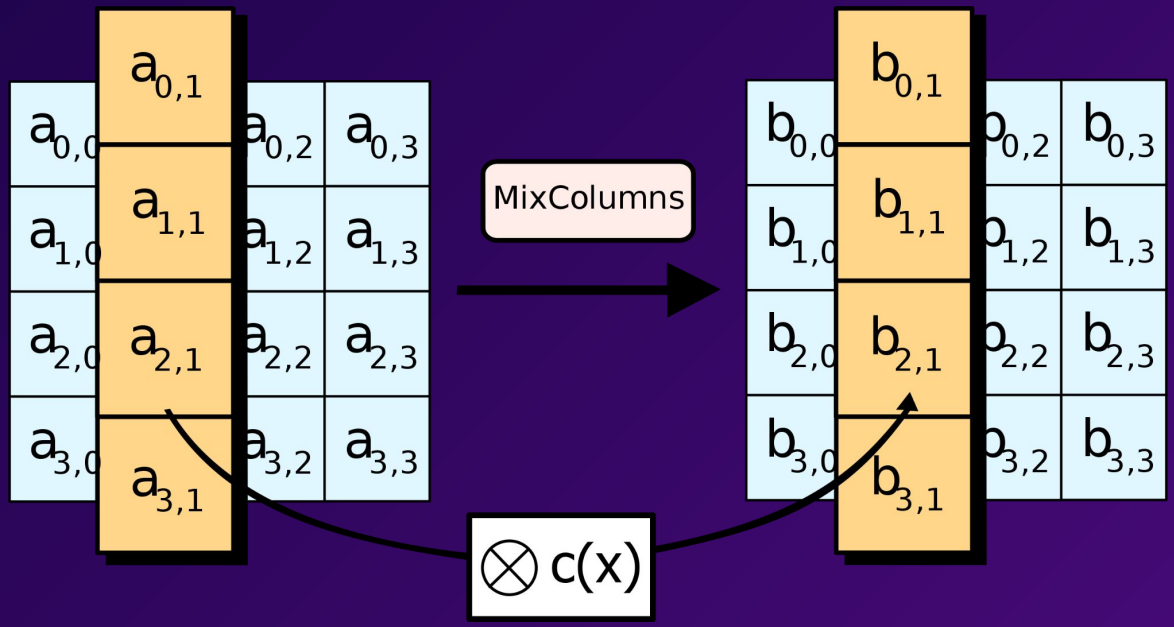
Each byte of the state is replaced with another based on a lookup table (S-box)

ShiftRows



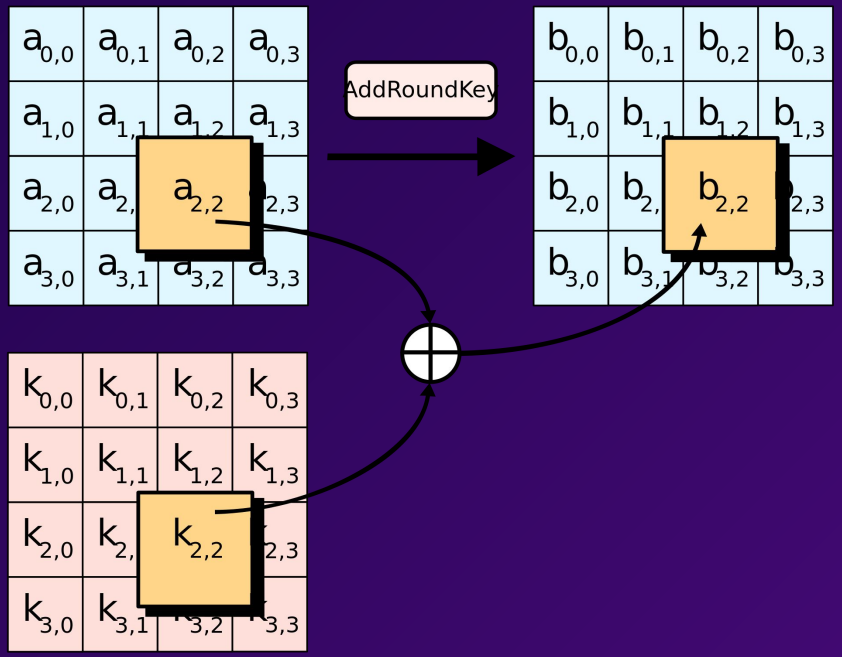
Each row in the state is cyclically shifted a number of times equal to its row index

MixColumns



The four bytes of each column of the state are combined using an invertible linear transformation

AddRoundKey



Each byte of the state is bitwise XORed with a byte of the round key

Effective in Software & Hardware

How Secure is AES?

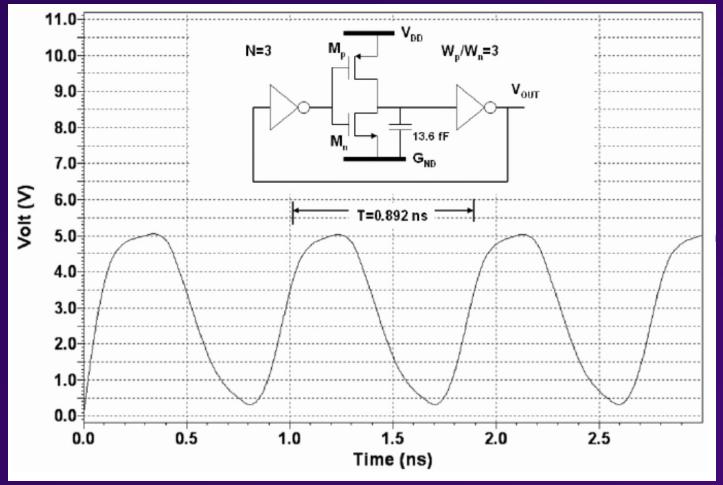
- With current computing capabilities, even if using a supercomputer it would take 1 billion years to brute force just a 128 bit AES key.

Does that mean it's COMPLETELY SECURE????

- Nope
- AES has proven vulnerable to multiple different types of side channel attacks, power monitoring based attacks being one of the easiest to execute.
- Traditionally power monitoring based side channel attacks required complete physical access to the victim device in order to hook up an oscilloscope and collect power traces.
- One FPGAs we can use inherent properties of the PL to collect power traces remotely.

Power Monitor Main Concepts

- Due to the dynamic power consumption characteristics of transistors the dynamic power consumption will always be higher than the static power consumption, aka when the transistors are switching.
- The increased power consumption slows down the whole network.
- Using a ring oscillator you can measure the frequency of its oscillation and based on the increased power consumption the frequency should get lower proportional to increased power consumption.
- When a one is being processed there is higher dynamic power consumption

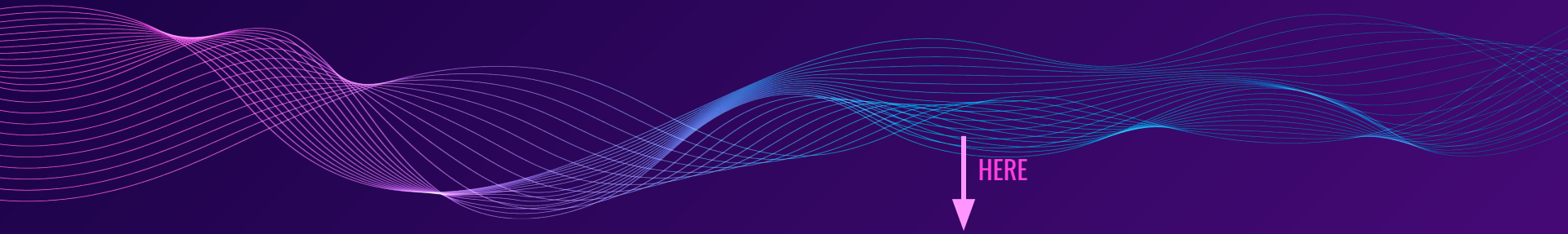


Project Goals

- Get Electronic Codebook (ECB) AES working in software and in hardware
  - Send input cipherkey, plaintext, and AES mode
  - Receive output ciphertext and decrypted text (encryption and decryption)
  - Get all key size modes (AES-128, AES-192, AES-256) working
  - Optimize hardware implementation

- Successfully implement a remote power based side channel attack on our AES accelerator.
  - Create a power monitor on the FPGA PL using ring oscillators.
  - Use power trace generated by onboard power monitor to find AES key

Project Timeline



Initial Python Algorithm

PyCryptodome & raw Python implementation

Initial Vitis Algorithm & Power Monitor

AXI4 & AXI-S implementations. SystemVerilog Ring Oscillator

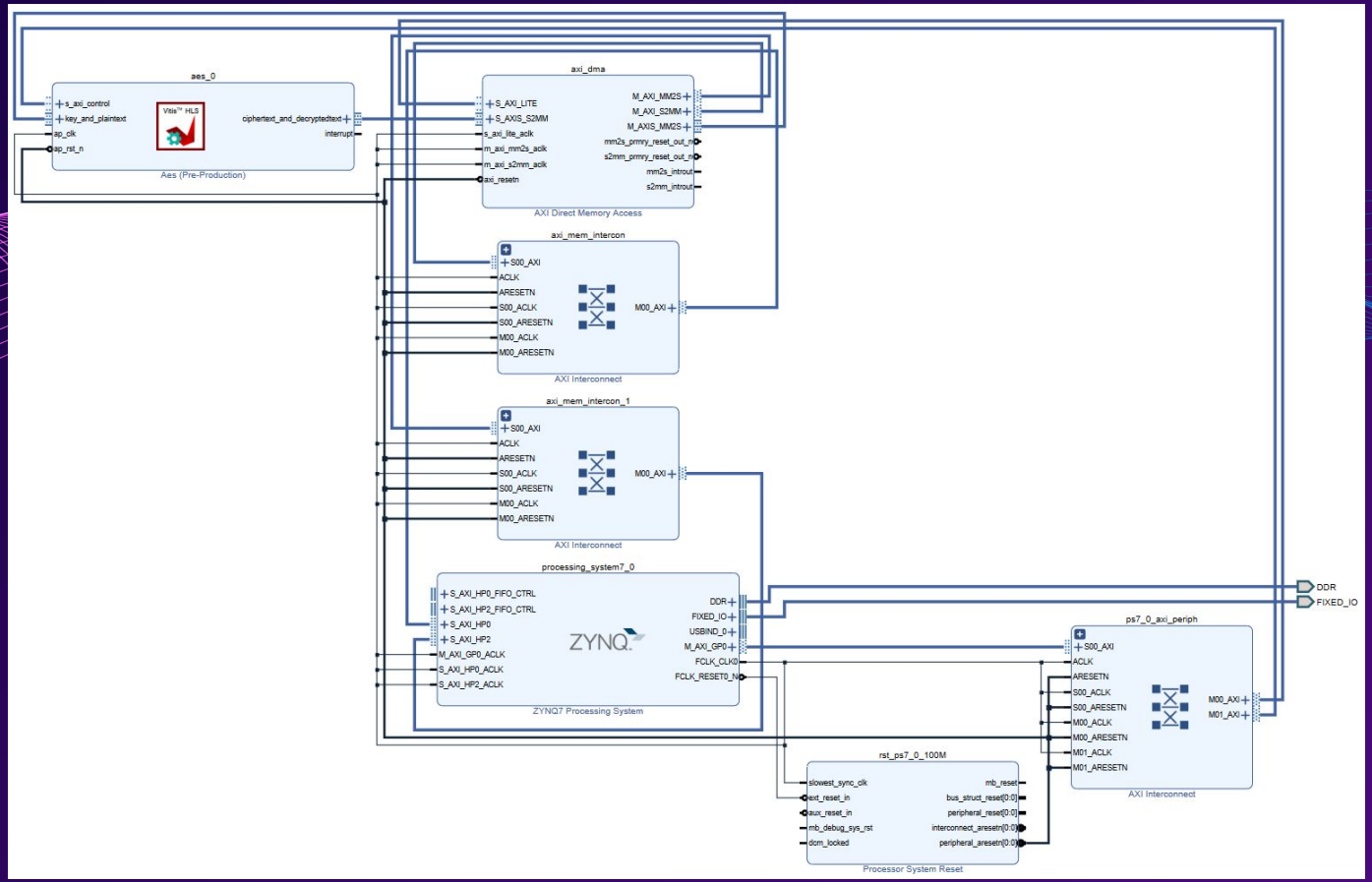
Final Vitis, C, & Python Algorithm, Power Monitor Implementation

Optimized AXI-S, final Python algorithm, Implement power monitor with rest of design

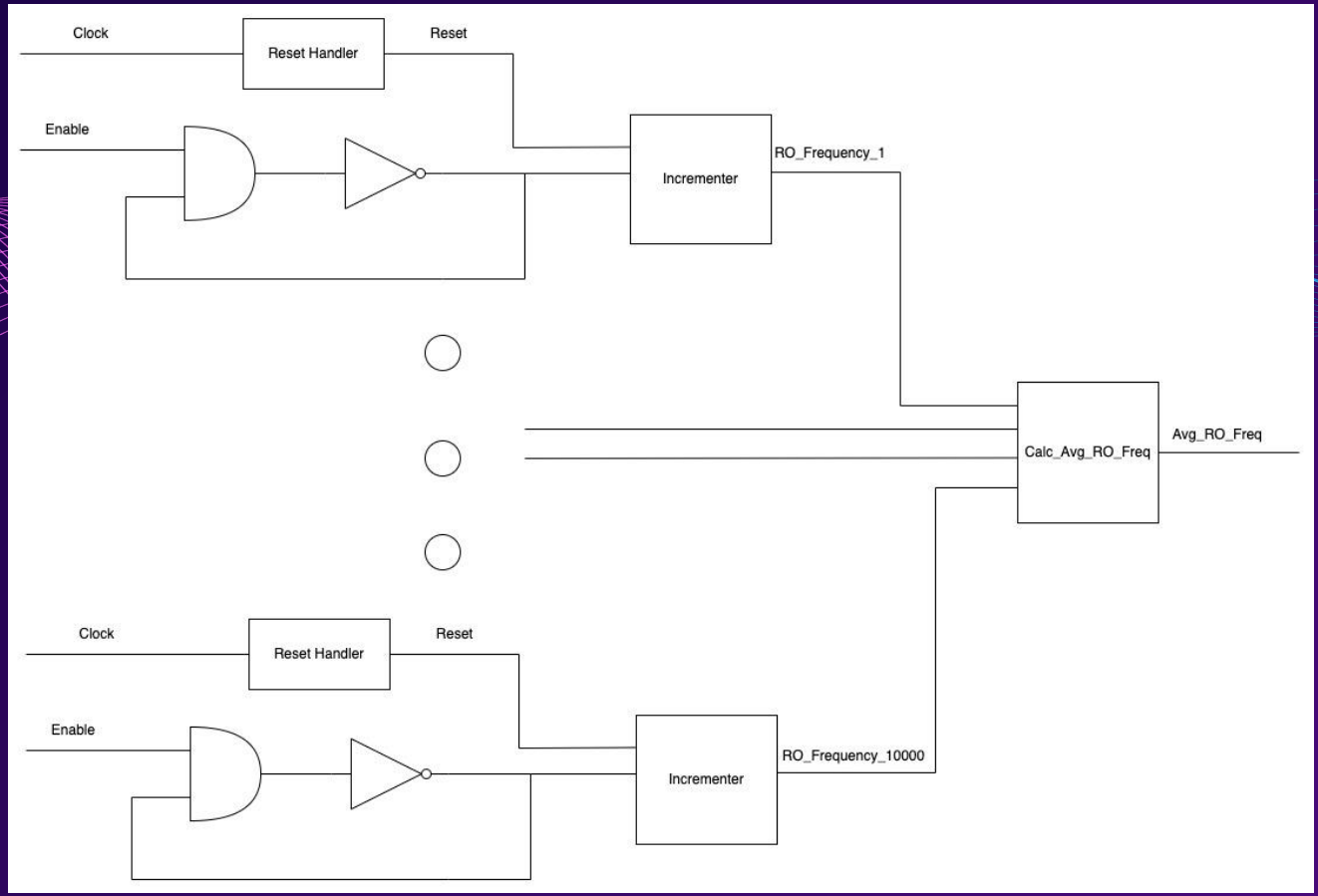
Final Project Report Power Side Channel Attack

Successfully recover AES key using power monitor

Vivado Block Design



Power Monitor Schematic

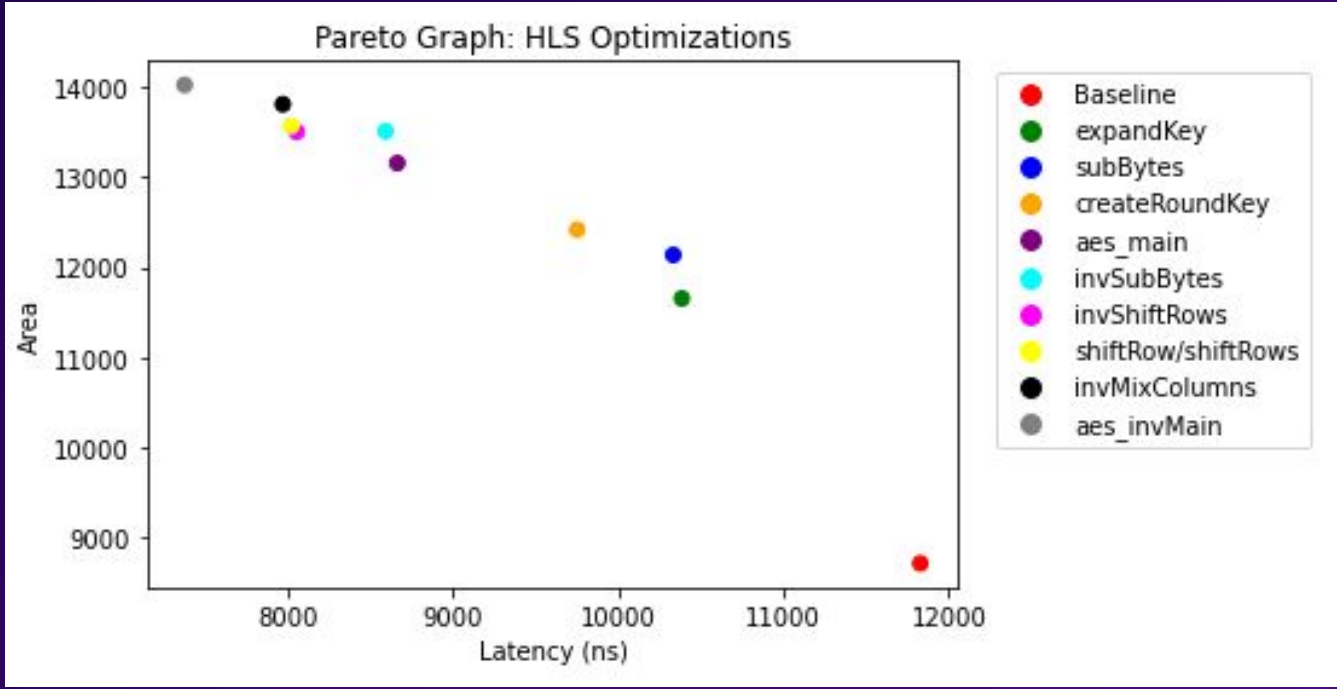


Vitis HLS Optimization Results

Compounding Optimizations

Optimization	Latency (cycles)	BRAM	FF	LUT
Baseline	11828	9	1785	7806
expandKey	10386	9	4405	10751
subBytes	10334	9	4555	11235
createRoundKey	9753	9	4548	11516
aes_main	8665	9	4811	12256
invSubBytes	8595	8	4929	12711
invShiftRows	8057	8	4990	12703
shiftRow/shiftRows	8025	8	5003	12770
invMixColumns	7973	8	5018	13011
aes_invMain	7379	9	5169	13126

Pareto Graph



AES Timing Results

HW Algorithm	Time (seconds)
Baseline: AES-128	0.006011486053466797
Baseline: AES-192	0.015700578689575195
Baseline: AES-256	0.011939287185668945
Optimized: AES-128	0.0018124580383300781
Optimized: AES-192	0.0017862319946289062
Optimized: AES-256	0.0017783641815185547

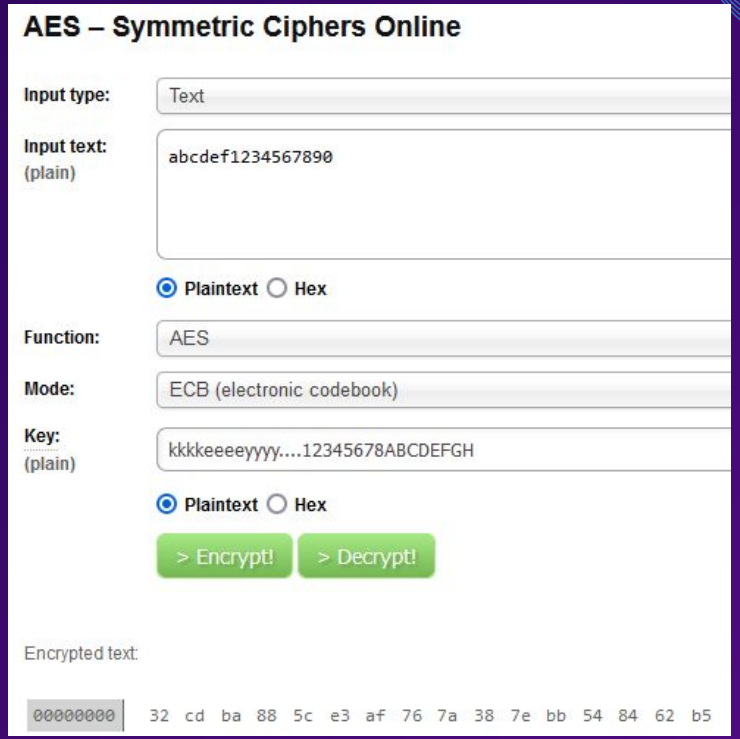
SW Algorithm	Time (seconds)
Python: AES-128	0.061778
Python: AES-192	0.065350
Python: AES-256	0.079375
C: AES-128	0.000043
C: AES-192	0.000050
C: AES-256	0.000054

- ~38.4 X Speedup (optimized hardware vs. Python software)
- ~6.3 X Speedup (optimized hardware vs. baseline hardware)

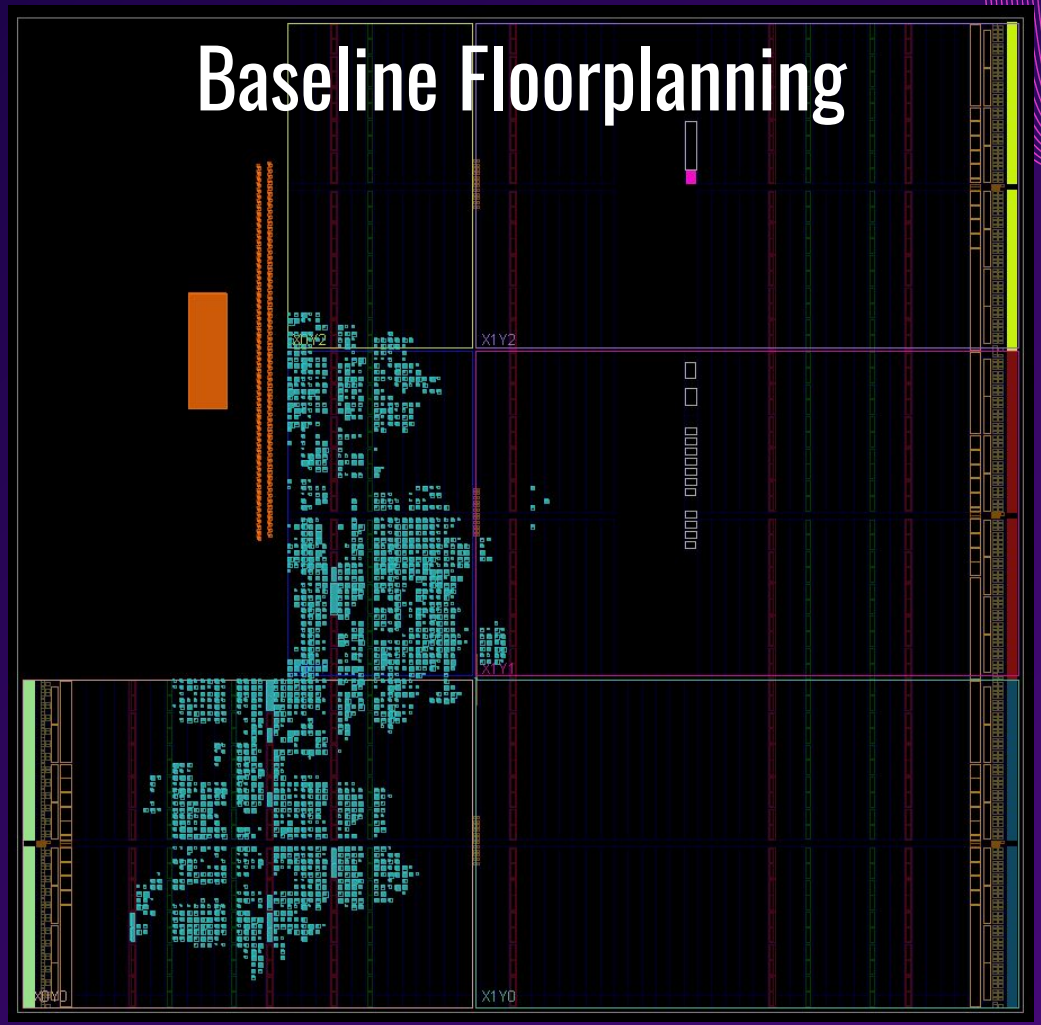
Validation

- Validated the output decrypted text is the same as the input plaintext
- Validated the same inputs and outputs across Vitis, C, and Python implementations
- Validated inputs and outputs against external AES implementations (AES-256 in the figures)

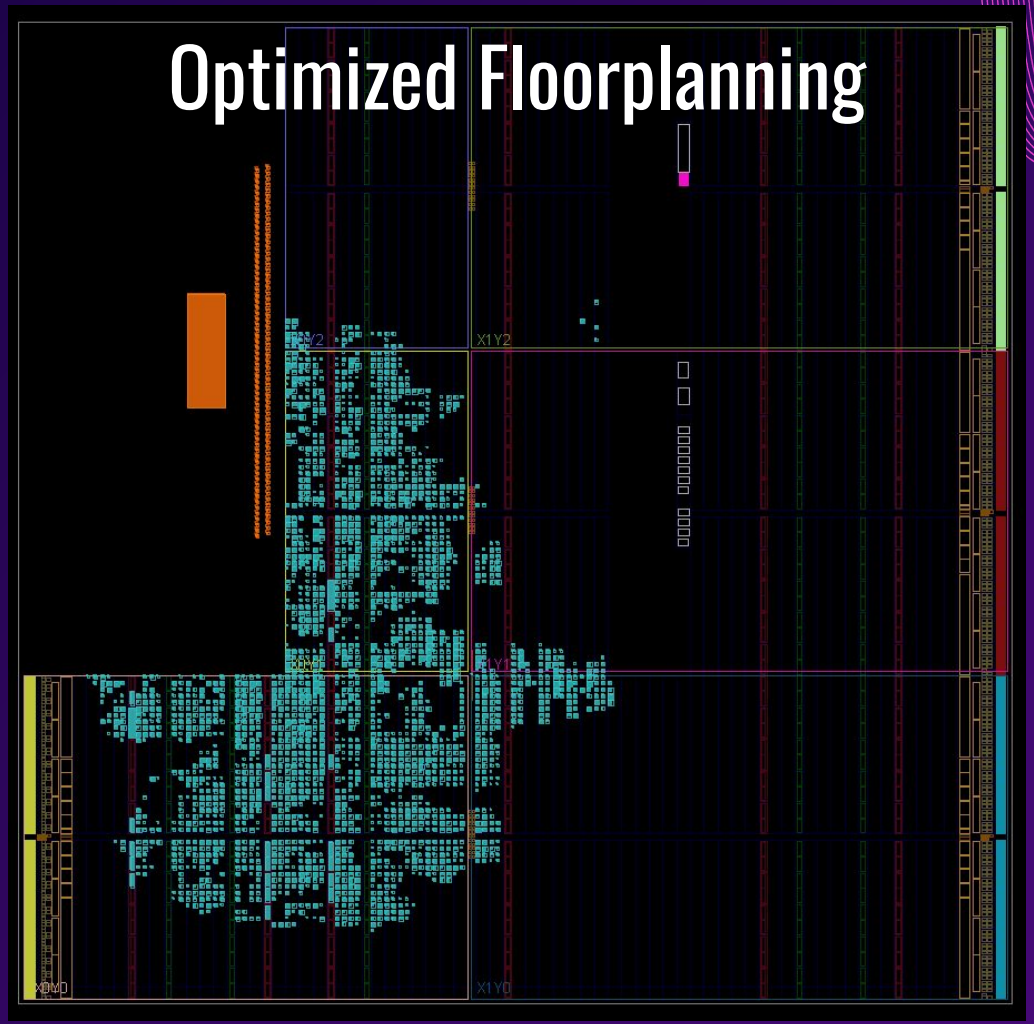
```
Plaintext (hex):
61 62 63 64 65 66 31 32 33 34 35 36 37 38 39 30

Ciphertext (hex):
32 cd ba 88 5c e3 af 76 7a 38 7e bb 54 84 62 b5

Decryptedtext (hex):
61 62 63 64 65 66 31 32 33 34 35 36 37 38 39 30
```



Baseline Floorplanning



Optimized Floorplanning



Lessons Learned

- Begin with a software implementation in C rather than one in Python
  - Python bitwise arithmetic can be obfuscating
- AXI-Stream behavior does not conform to a software-oriented sequential process
  - TLAST must be set after output stream writing occurs
  - Numpy buffer data types remain silent and do not throw errors
- Start working on interfacing power monitor much earlier on
  - Getting two IP’s to talk to each other without AXI is hard

Future Work

- Project Report
- Get design working with power monitor as part of the overlay.
- Use the power monitor to extract the AES key
- More Time?
  - Explore optimized algorithms for AES
  - Explore other AES modes (Cipher Block Chaining (CBC), Cipher Feedback (CFB), Propagating Cipher Block Chaining (PCBC), Output Feedback (CFB), Counter (CTR))
  - Automate key recognition
  - Try implementing key recognition in the presence of additional processes on the board.

References

- https://github.com/m3y54m/aes-in-c
- https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
- https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf#page=1
- https://wikimedia.org/api/rest_v1/media/math/render/svg/63ac3cf2cb47d5a29c1210fca521f9e4e49e39b1
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#/media/File:AES-SubBytes.svg
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#/media/File:AES-ShiftRows.svg
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#/media/File:AES-MixColumns.svg
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#/media/File:AES-AddRoundKey.svg
- https://upload.wikimedia.org/wikipedia/commons/5/50/AES_%28Rijndael%29_Round_Function.png

Q&A