Hardware Accelerated Encryption Cracking

Team Members:

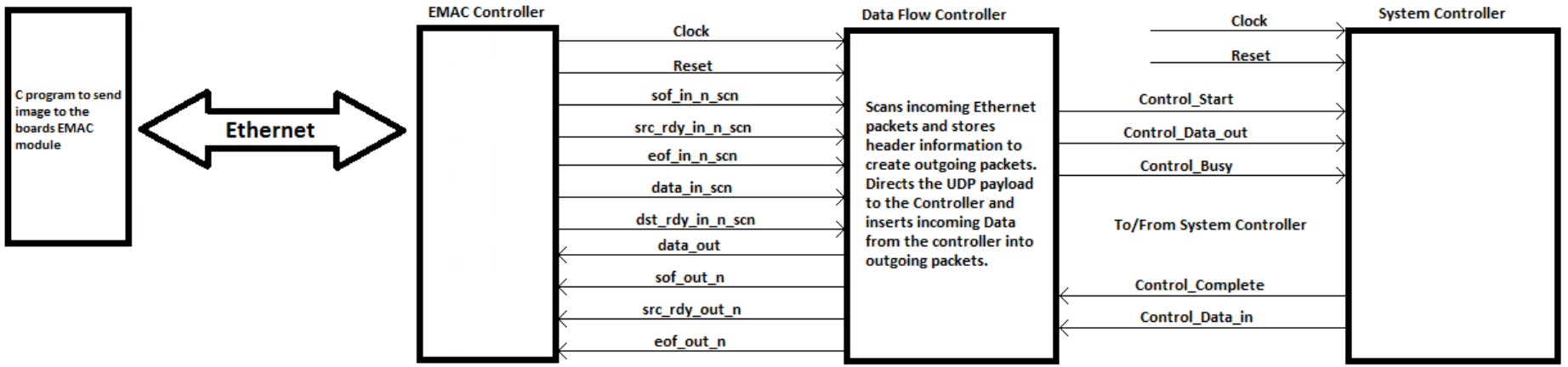
Bryant Smith
Mathew Bitterman
Daniel Zundel

Introduction

- Data security is a #1 priority for every field of communications
- Millions of Dollars is spent in breaking encryption standards
- Brute force attacks are the most common and most time consuming
- Exploring speed up of hardware accelerated brute force attacks vs conventional (software) attacks
- Test will be performed on a single encrypted image file using the blowfish encryption method
- Upon successful attack the decrypted image will be returned

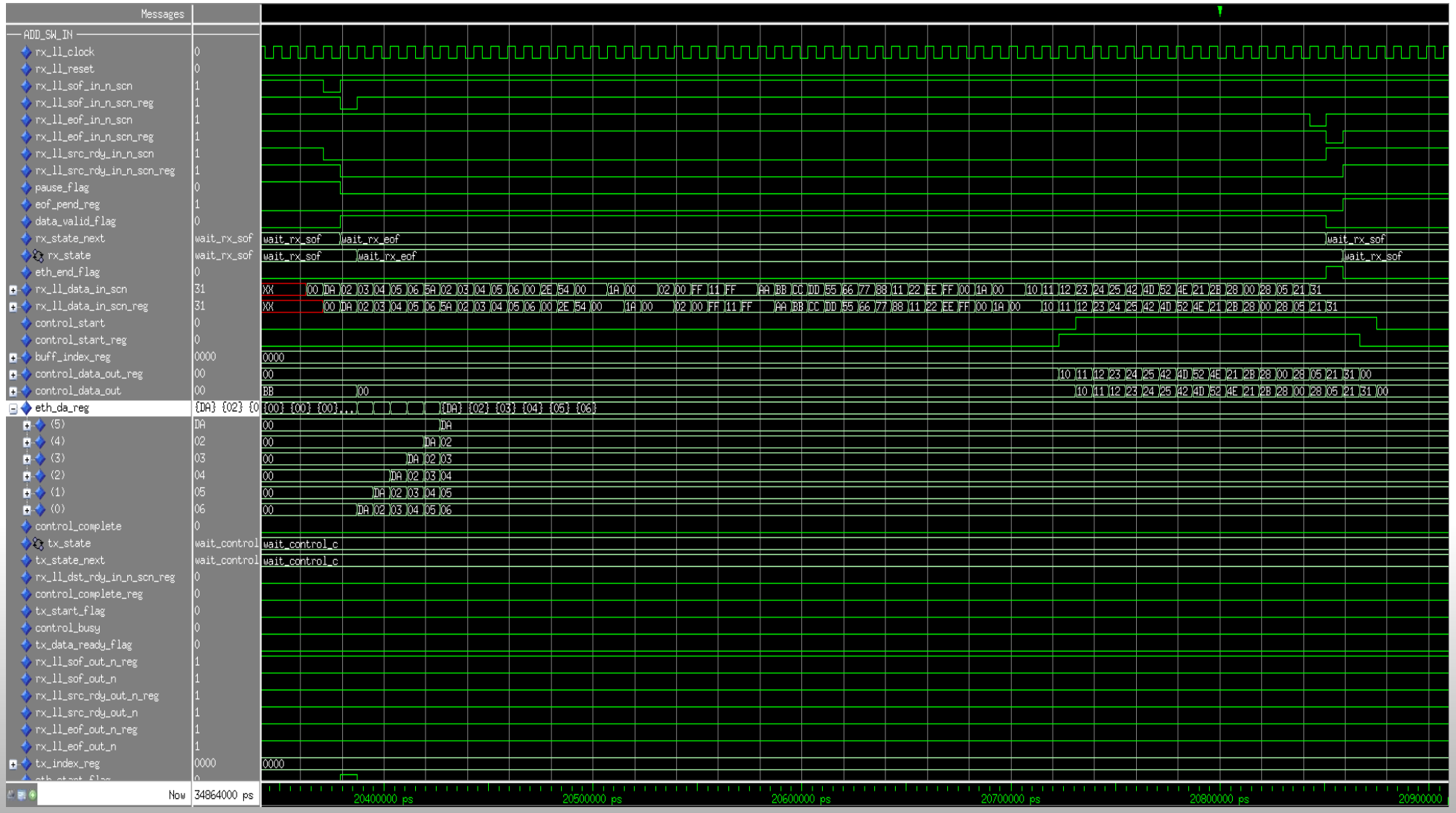
Data Flow Control

- Receives incoming image via Ethernet
- Parses through Ethernet packet information
- Sends Payload data to the System Controller



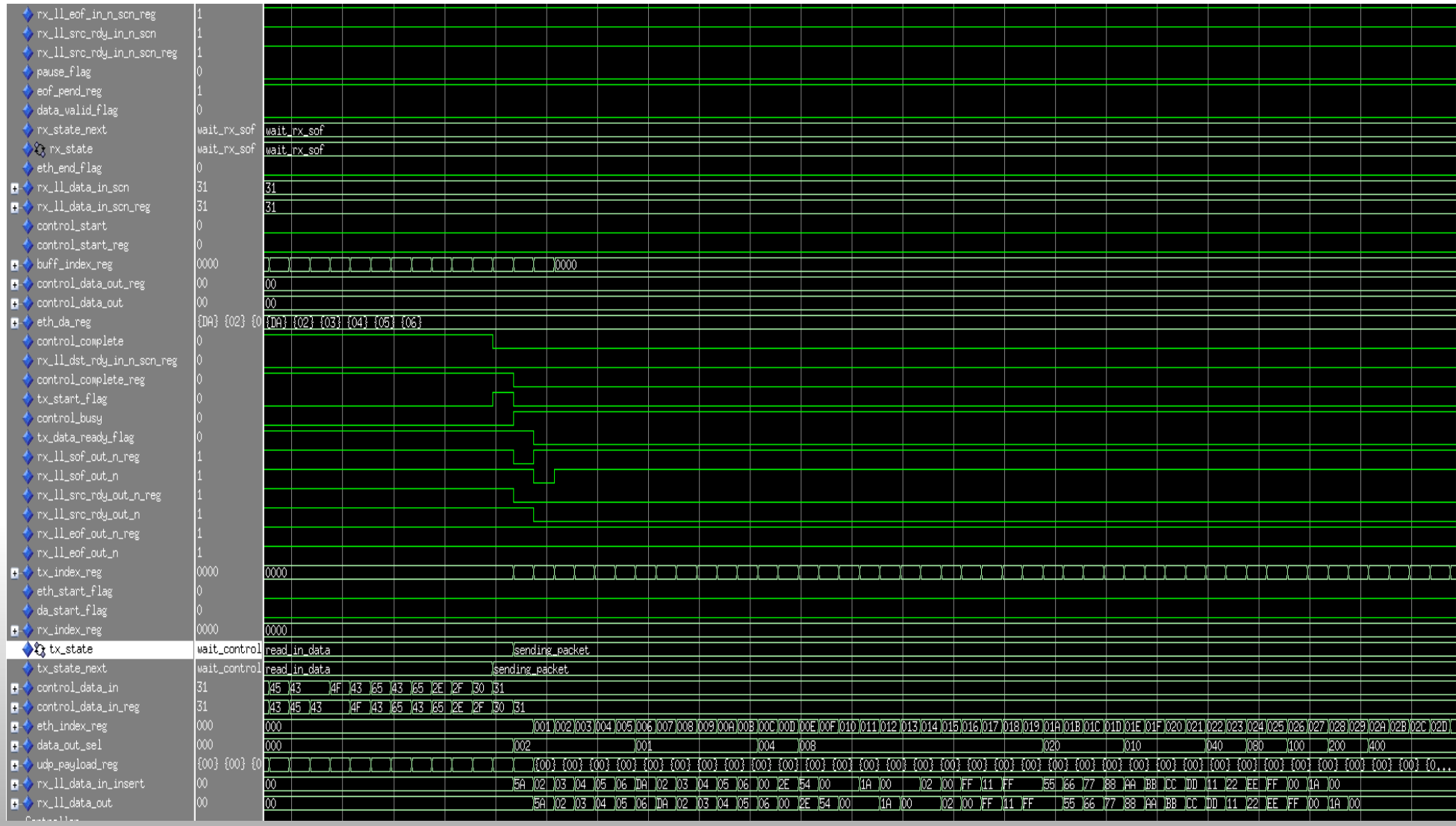
Data Flow Simulation

- Incoming Data



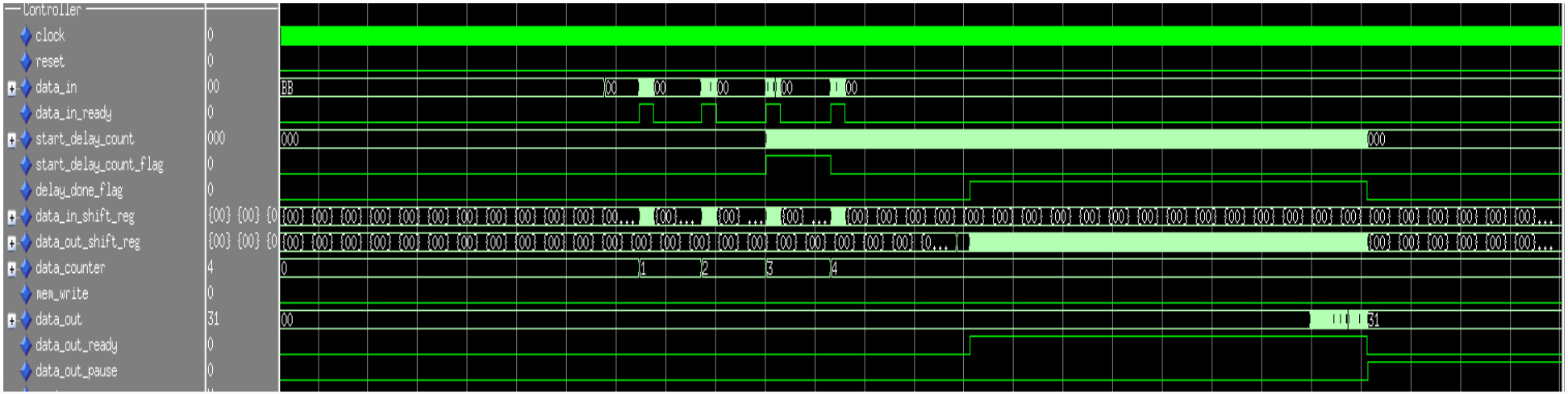
Data Flow Simulation

- Outgoing Data



Data Flow Simulation

- System Controller Simulation



DDR2 Memory

- The on board 256 MB DRAM is used to store the encrypted image during key cracking.
- MIG (Memory Interface Generator) was used to produce a soft core memory interface. A single controller was generated with a 128 bit data-width and burst length of 4.
- Memory is run at 200 MHz using a differential clock provided by the on board frequency generator.
- A User Interface buffers data and issues the memory interface control signals, piping and writing data from the Ethernet component in bursts.
- Data received from the Ethernet component is buffered until the data required for a burst is full or the end of the packet is reached at which point a write occurs.
- Any unused writes in a burst are masked.
- The memory is addressed linearly; a register stores the last address and increments the next address after every write - maintaining the address space of the stored image.
- Data is read back in bursts, incrementally reading each memory address until the last written address is reached.

Memory Interface Architecture

User Component

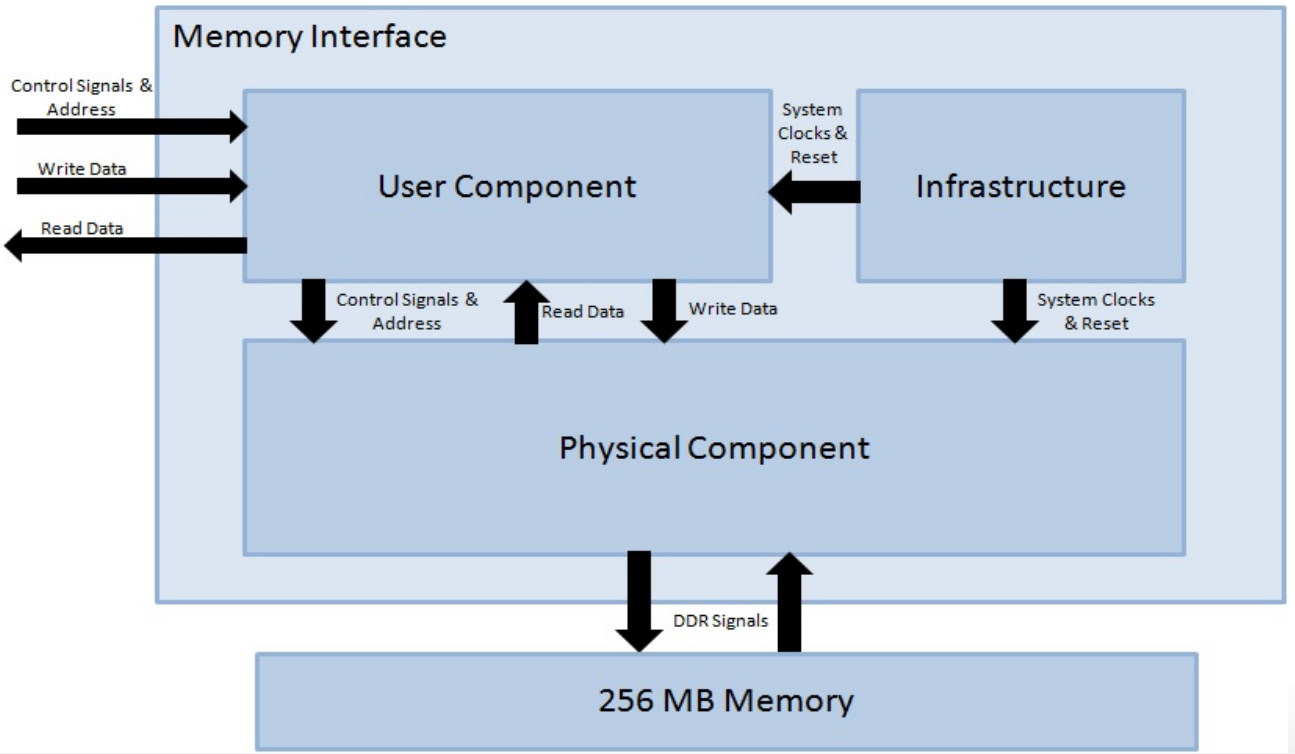
- Controls memory read, write, and addressing signals for the user operation of the memory interface

Infrastructure

- Includes memory control clock and reset signals, this component includes idelay blocks to compensate for environmental effects on the memory clocks, such as voltage a temperature variations.

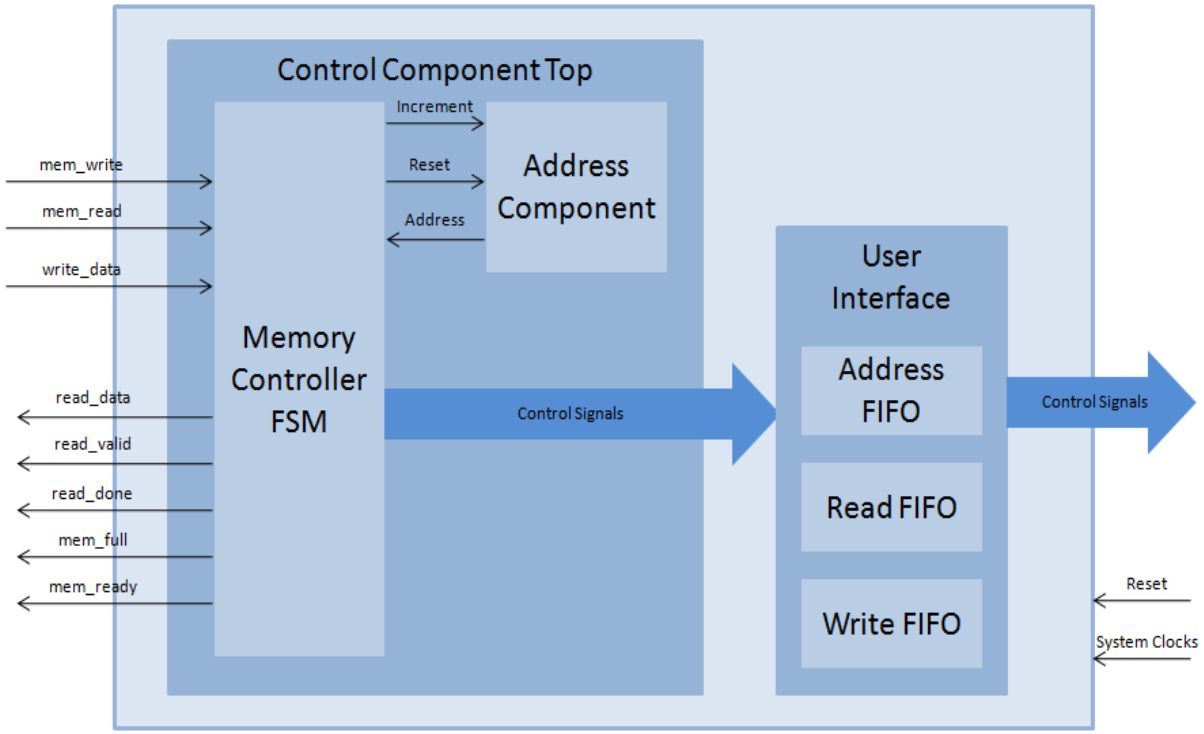
Physical Component

- Comprises all I/O signals to memo including buffer and delay primitives. This component also performs memory initialization and calibration.

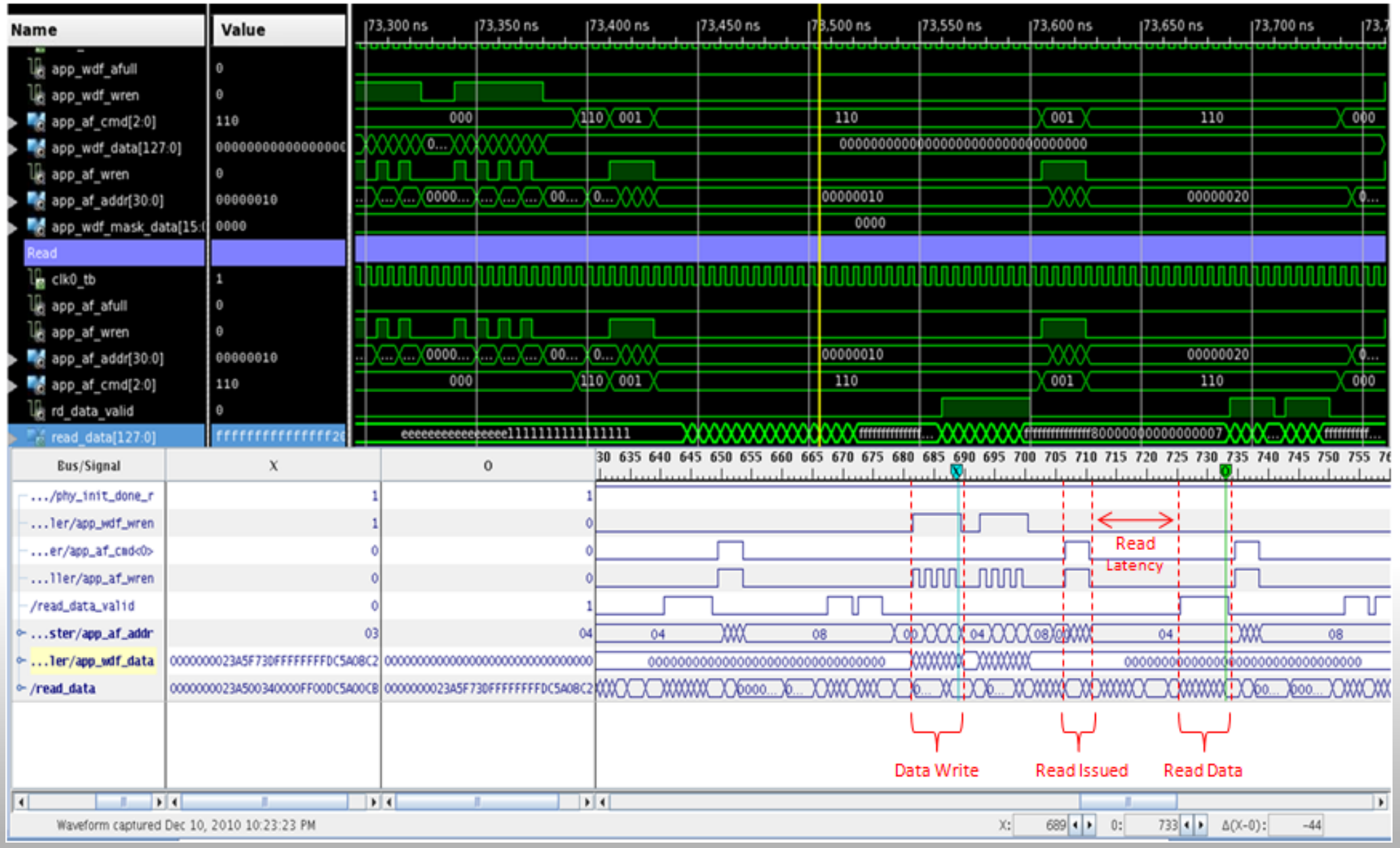


User Component Architecture

- Offers simple input signals to write and read data from memory.
- Insulates the overall system controller from memory control signaling and bursting requirements.
- Utilizes data buffers to isolate different clocks used by the memory and Ethernet components.



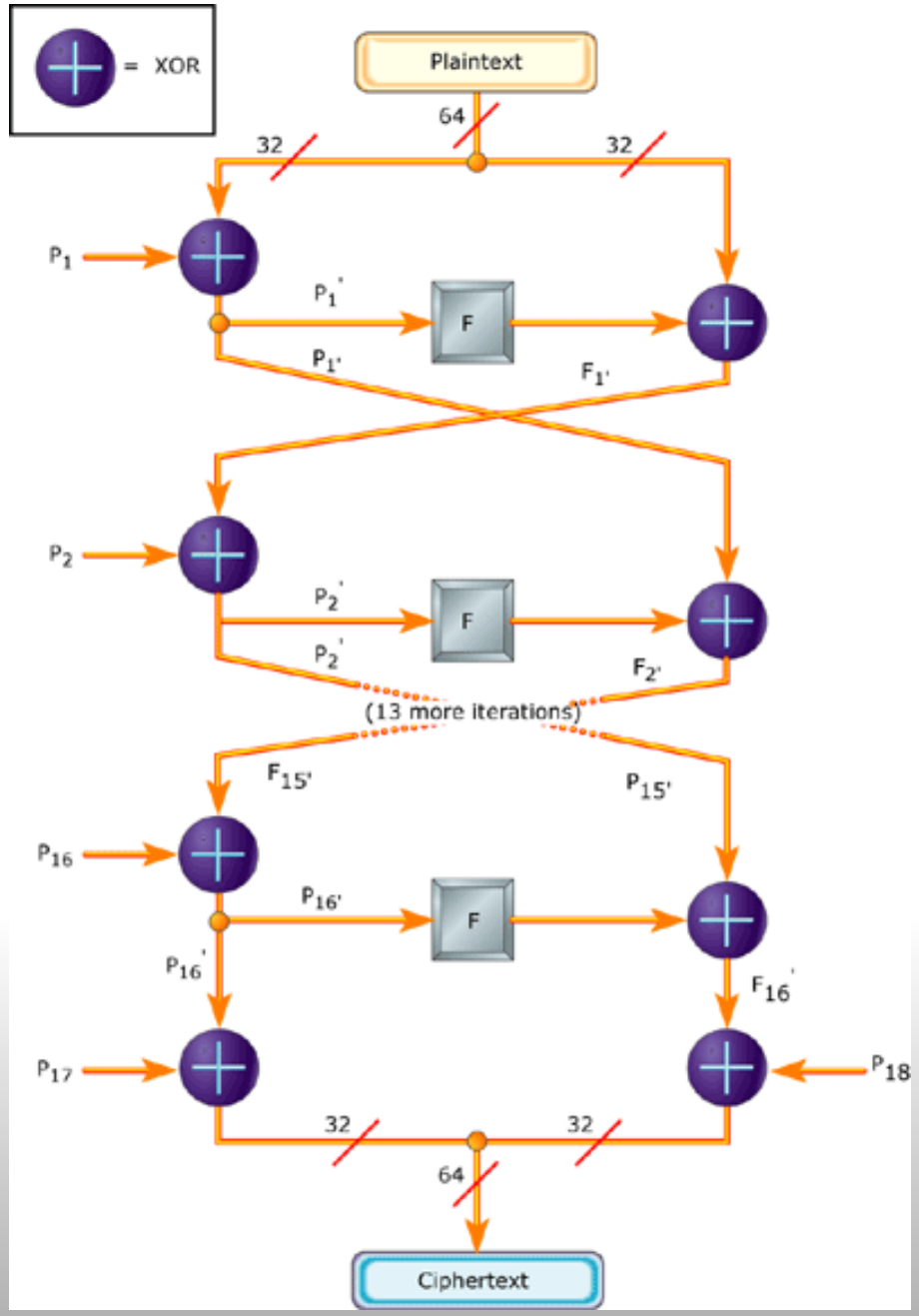
Memory Simulation Results



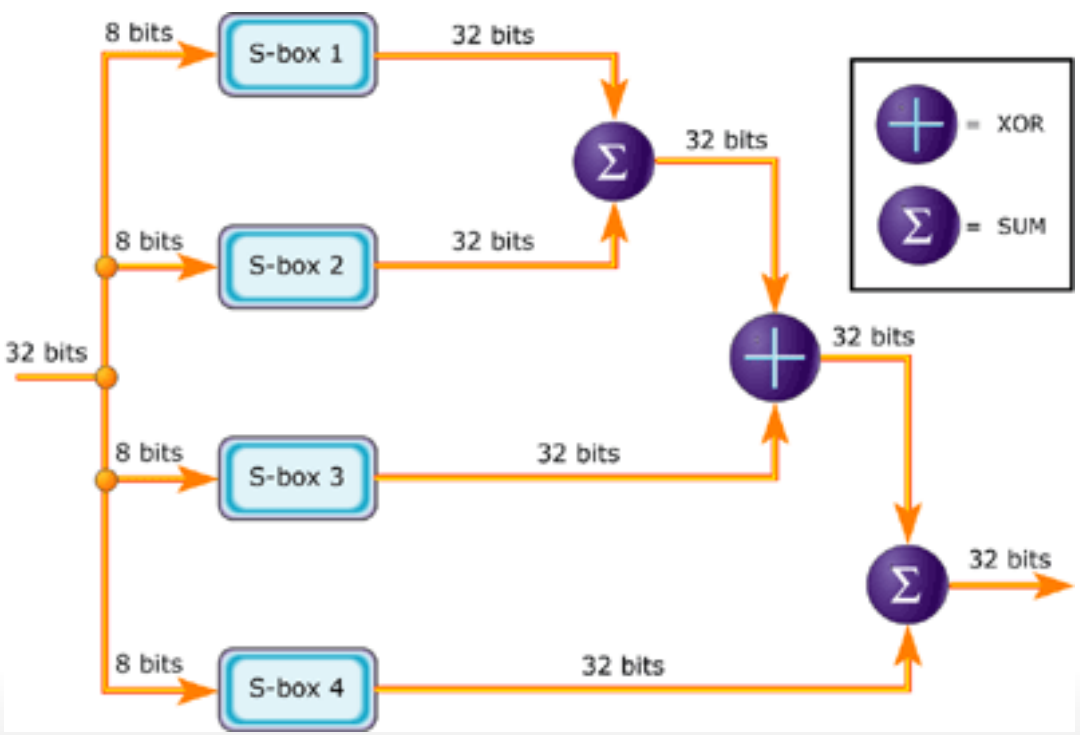
Introduction to Blowfish

- Used to replace DES which was proving to be easy to crack as teams were doing so in under 24 hours (22.25 hours)
- Developed by Bruce Schneier and published on public domain
  - can never be licensed or patented, free for everyone
- utilizes a 16 round feistel network
  - a common operation is repeated 16 times
  - lends itself well to same operations to encrypt and decrypt
- key can be 32-448 bits (in steps of 8-bits)
- no known cryptanalysis attacks have been reported
- considered a "fast cipher" (data in to data out)
- basic operation of data substitution is key dependent which increases security

Introduction to Blowfish cont.



F-box definition

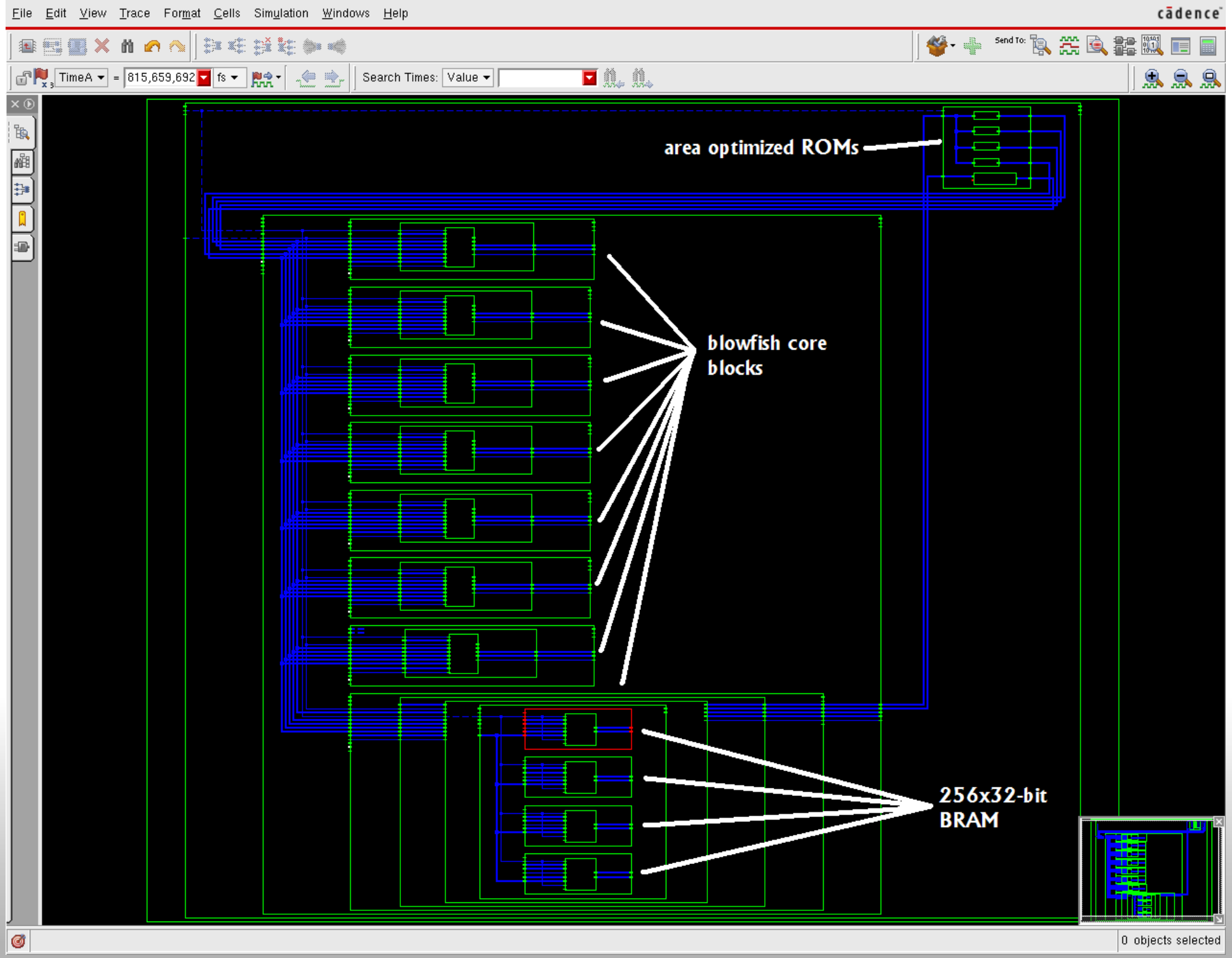


Blowfish Algorithm Primer

- microcontroller/PLD "friendly"
  - operations are substiutions, XORs, ADD (32-bit overflow)
- two main data structures, P-array and 4 S-boxes
- algorithm is started by filling the P-array and S-boxes with the Pi hex data
  - fractional part of Pi (Pi-3.0) in hex (tables exist for this)
- operations are performed to mix the key into the P-array and S-box initialization data
- once complete, data can be cycled through the algorithm
- algorithm is fast, except when keys need to be changed

Blowfish Architecture

- had to restructure the core block 3 times to fit on the FPGA
- utilized area optimized ROMs, and 8Kbyte BRAMs
- each blowfish core block uses 4 8Kbyte BRAMs for S-boxes
- 8 core blocks used so 32 8Kbyte BRAMs utilized (of 296)
- hierarchy shown on following slide


File Edit View Trace Format Cells Simulation Windows Help
cadence
Send To:
TimeA = 815,659,692 fs
Search Times: Value
area optimized ROMs
blowfish core blocks
256x32-bit BRAM
0 objects selected

Key Search Algorithm

- since 8 cores are used, key space is broken into 8 parts
  - block 0 keys: 0x00000000-0x1FFFFFFF
  - block 1 keys: 0x20000000-0x3FFFFFFF
  - block 2 keys: 0x40000000-0x5FFFFFFF
  - block 3 keys: 0x60000000-0x7FFFFFFF
  - block 4 keys: 0x80000000-0x9FFFFFFF
  - block 5 keys: 0xA0000000-0xBFFFFFFF
  - block 6 keys: 0xC0000000-0xDFFFFFFF
  - block 7 keys: 0xE0000000-0xFFFFFFFF
- ensures that no keys are tested twice
- upper n bits of 2^n cores used seperate keys (3 in this case)
- top blowfish controller handles sending keys to the cores

Key Search Algorithm cont.

- Controller enables cores with key to use
- cores initialize data structures and attempt to decrypt data
- if none of the decrypted data has 0xFFD8 as the first two bytes, the keys are incremented and the cores are run again
- if 0xFFD8 is found as the first two bytes in any of the outputs, the cleartext is sent back in to be encrypted, the encrypted output is matched against the original input
- if it doesn't match exactly, the keys are incremented and the cores are run again
- if it matches exactly, core0 is configured to run with the found key
- once core0 is initialized, the blowfish controller signals the top level controller that a key has been found and is ready for data

Blowfish Issues

- first write of core took <100us to test a key but one core took >160% of resources on FX70T FPGA
- second write of core took ~200us to test a key but one core took ~130% of resources on FX70T FPGA
- third write of core takes ~400us to test a key but eight cores take ~60% of resources on FX70T FPGA
- collectively 8 cores can test ~20,000 keys/second
- this would require ~60 hours to test a 32-bit keyspace

Blowfish Issues Cont.

- hardware testing has shown that false positives can occur
- ex.
  - 0xFFD83344556677 88 encrypted with 0x67AEF891 is 0x55F498A5C51B16AB but this 64-bit block can be decrypted with keys 0x00004BDF, 0x6000AEAE, 0x8000640E, 0x40005010, 0x60003231, 0xC000A891, 0xE000359F to name a few found in simulation
- the false positive keys can't be used to decrypt other data encrypted with the same key, it returns bogus cleartext
- makes finding the correct key difficult
- no solution has been found yet to correct this issue

Current Status

- Integrating EMAC and DDR components together
  - held up at getting clocks straightened out
- Next step is to integrate the blowfish into the project
- Searching for a fix for false keys
- Documenting RTL
- Writing the final report

Conclusion

- FPGA's can be used as a viable option for brute force attacks as their inherent parallelism can be used to test several keys at once
  - The 22.25 hour DES attack used 128 Xilinx Spartans
- This generic structure could be applied to any number of ciphers
- With enough resources (FPGA size, number of FPGA's), even the largest of keys could be cracked in a reasonable timeframe

Lessons Learned

- Set times when the entire group can meet to discuss topics
  - proved to be difficult with 2 time zones, work schedules, family issues, etc.
- Keep communication channels open
  - thank-you google chat
- Utilize a way everyone can work on the same documents
  - thank-you google documents
- Not having physical access to hardware can be troublesome
  - thank-you Dr. Jones and Dr. Zambreno
- Check for system level I/O and clock conflicts early when desigining discrete components for integration down the line.
- Don't be afraid to ask for help

Questions/Comments?