VidAccel

A FPGA Accelerated Video Streaming Service

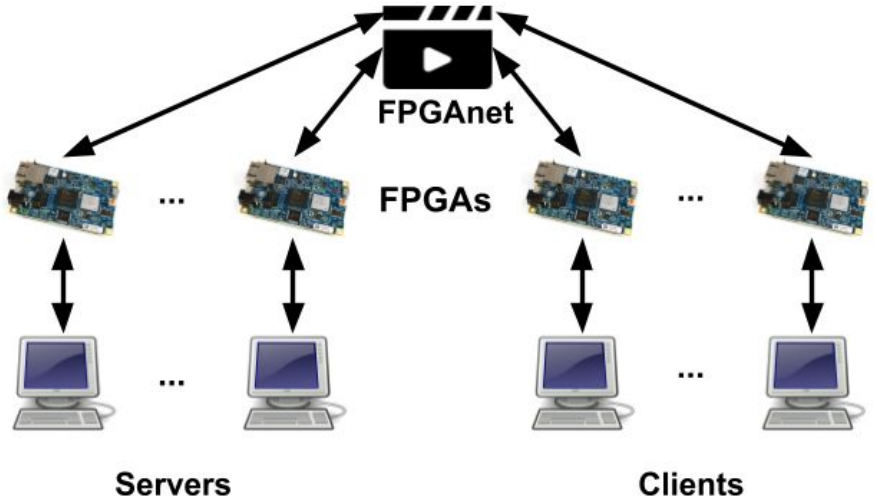
Team 7
Tianyi Zhang, Bruno Almeida, Kanver Bhandal

Project Overview

- A video streaming platform
- Accelerated encoding and decoding
  - FPGAs > CPUs
- Why?
  - Minimize buffering and quality drops while streaming videos
  - We watch videos in our daily lives

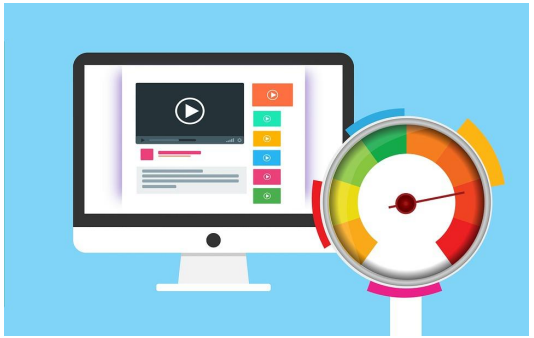
Scaling Up

- Data Centers
- More Clients and Servers
- Reduce Costs
- Faster Encoding

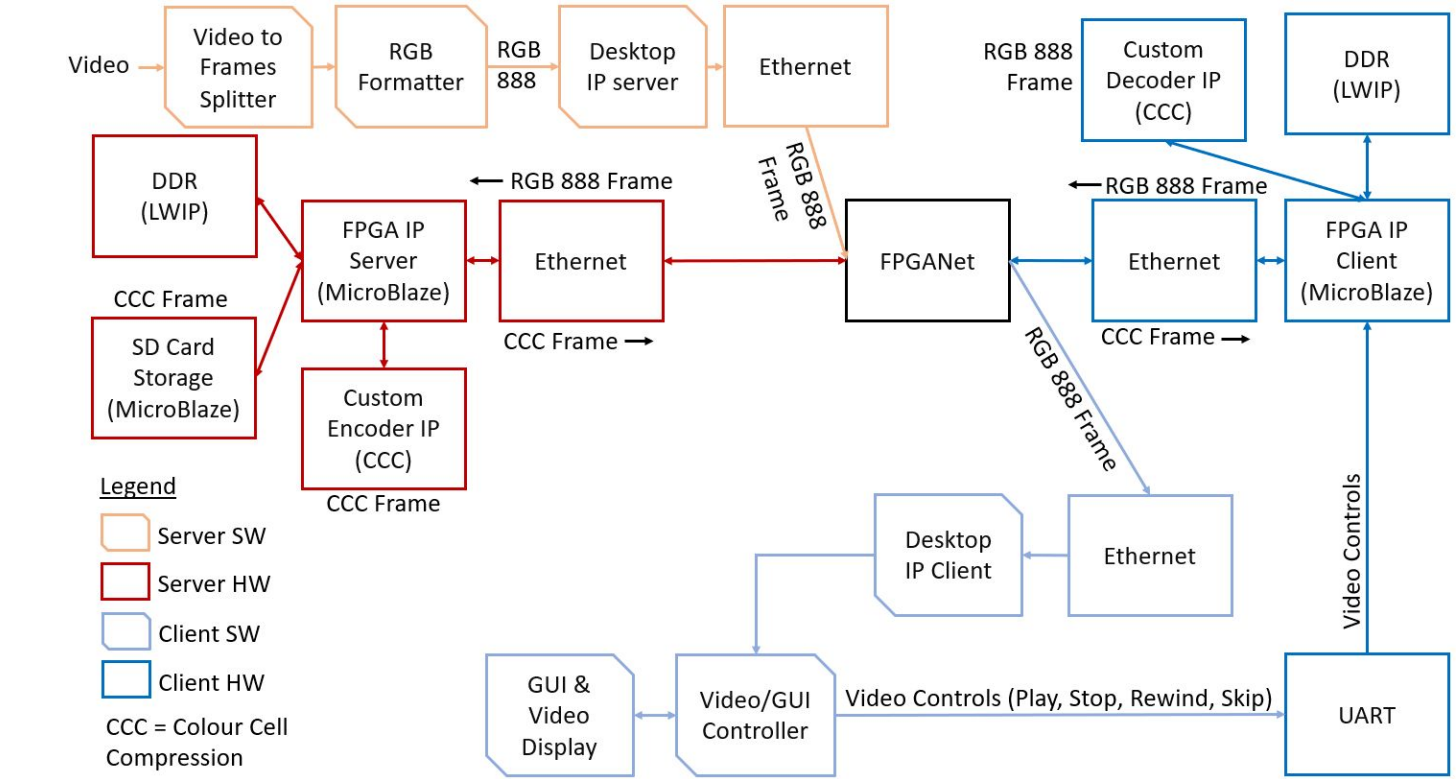


Initial Goals

- Providing faster network connection for video streaming
  - Encoding RGB frames
  - Using a lightweight network stack
  - Reducing bandwidth requirements



Planned System

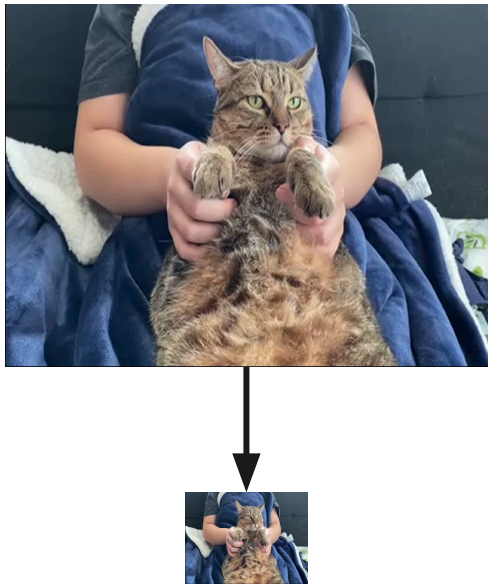


Modifications

- Planned to use IP and build our own transportation layer protocol
  - Never reached this point due to time constraints and only used TCP
- Planned to compare TCP vs UDP
  - Never implemented UDP due to time constraints but gathered TCP performance measurements

Modifications

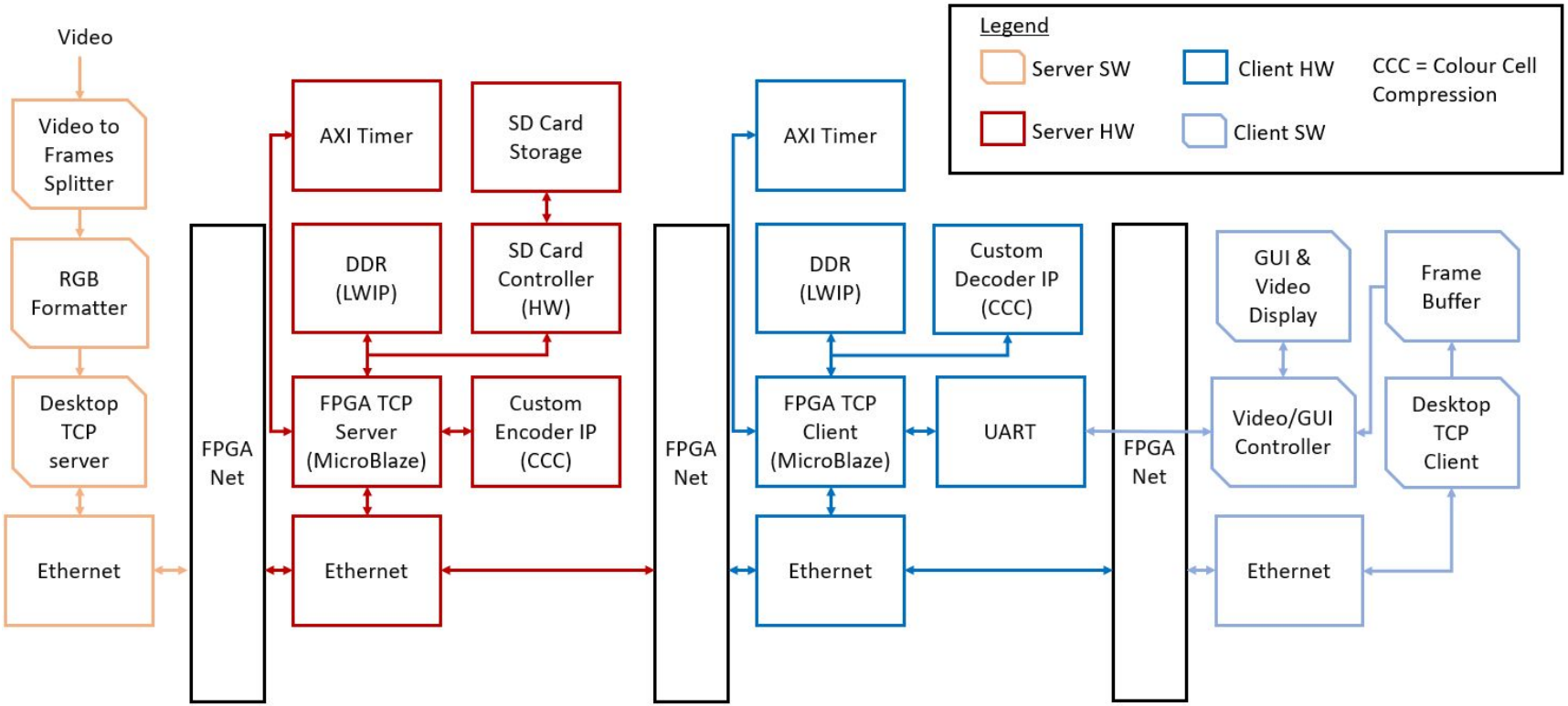
- Planned to stream 640x480p video
  - Reduced to 160x120p video
- Planned to stream the video live
  - Network/code was too slow to send a frame in 42ms (24 FPS)
  - Instead we buffer the whole video then display



Modifications

- Planned to use UART to send video start/stop commands to FPGA Client
  - Unneeded
  - Instead used UART for logging debug messages
- Planned to control SD Card using MicroBlaze and AXI QuadSPI
  - Didn’t work
  - Instead used a HW SD Card Controller

Final System



Challenges

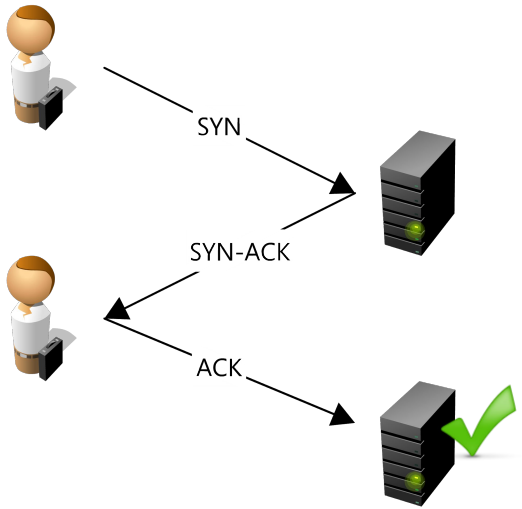
- SD Card
  - Not able to receive responses using AXI QuadSPI
  - Switched to different SD Card Controller IP
  - Was able to communicate but some bytes were randomly incorrect when reading/writing a block
    - Possibly due to different clock domains (100 MHz AXI vs 25 MHz SPI)

Challenges

- Signedness of numbers
  - `char` is signed for arithmetic in C
  - Encoding produced bad results
  - Use `unsigned char` to fix

Challenges

- Using LWIP for TCP
  - Slow performance
    - **35 mins** to send a 6s video…
    - Resolved by increasing TCP packet size, TCP Window, and optimizing code
    - Now **~7s** to send a 6s video!



Existing IP Used

- MicroBlaze
- AXI UART Lite module
- TCP LWIP and sample code
- SD Card controller module (from MIT course website)
- Multiple Python libraries (e.g. OpenCV, PySerial)

New IP Created

- Video to frames splitter and RGB formatter (Python code)
- Desktop Server and Client TCP (Python code)
- FPGA Server and Client TCP (MicroBlaze code)
- Desktop GUI, video display, video controller (Python code)
- CCC Encoder/Decoder modules
- SD Card AXI-Lite interface module

Design Process

- Common Git repository
  - Frequently committed custom RTL files, MicroBlaze code, Python code
  - Sometimes committed Vivado project files
- Simulated hardware blocks, then MicroBlaze testing with ILAs
- Separately developed components, then integrated
  - TCP code, SD card interface, encoder/decoder modules
  - Used MicroBlaze + TCP Vivado project as the base
  - Integrated SD card and encoder/decoder as AXI-Lite IPs

Lessons Learned

- Prepare a project descoping plan in advance
- Debugging custom AXI IPs in simulation using an AXI VIP saves time
- Don’t trust Vivado reports for resources/timing until you know the design is functionally correct
- ILAs are useful for stepping through a MicroBlaze program

Demo and Questions