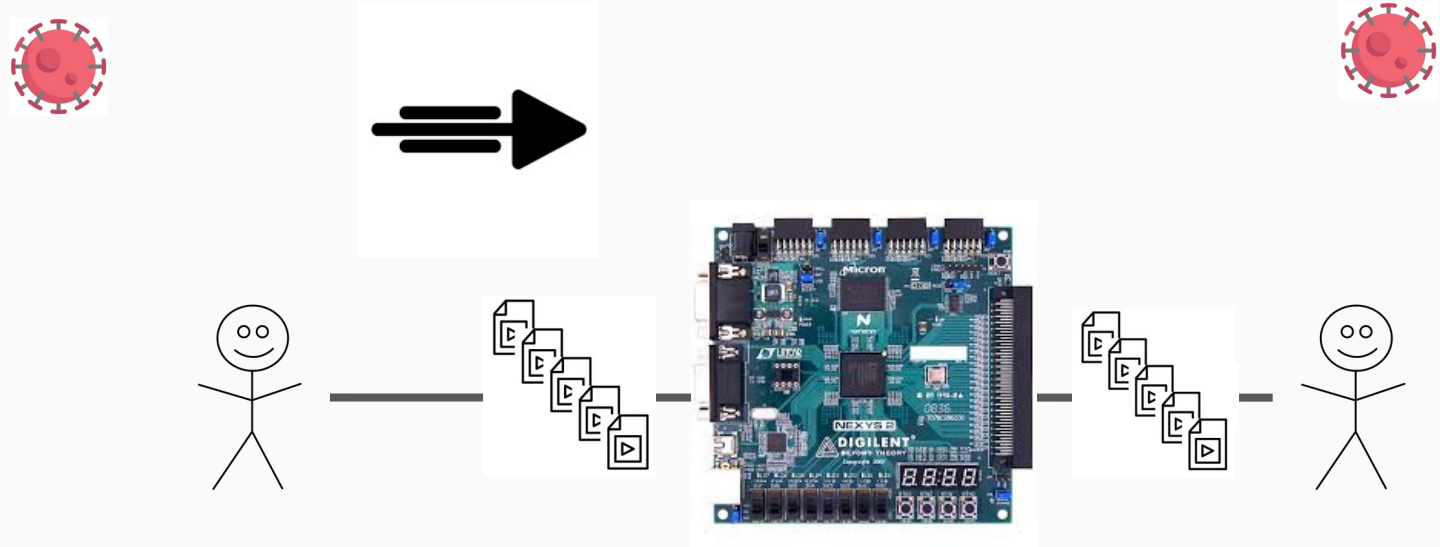
H.263 Media Compression Broadcast on FPGAs
Final Demo

Group 4 (Justin Hai, Yufeng Zhou, Isamu Poy)

Project Summary

About the Project

Why Compression?



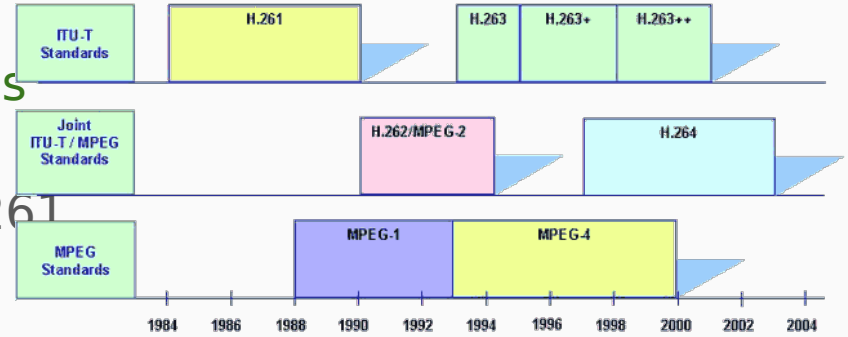
What is H.263?

What is Video Codec?

- compresses and decompresses digital video

What is special about H.263?

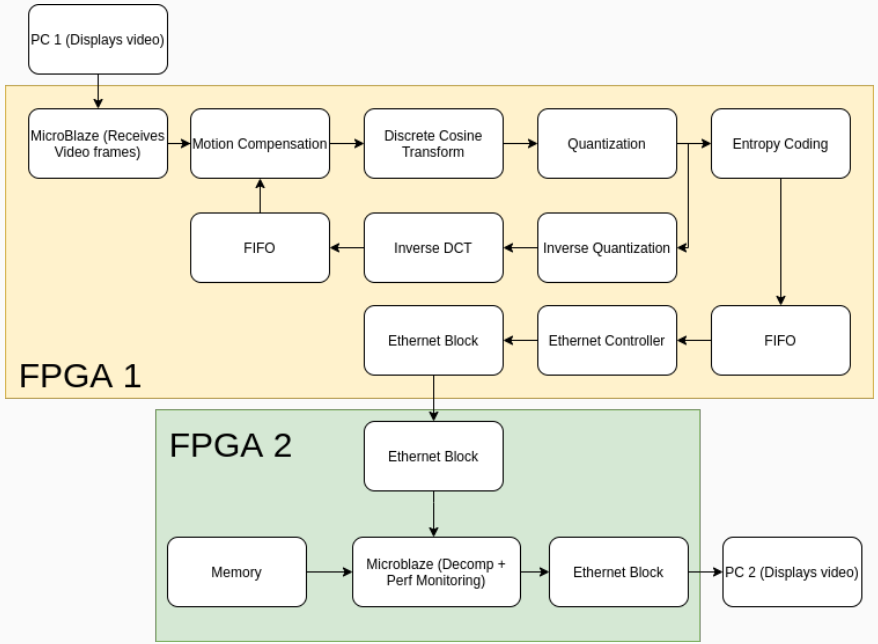
- H.263 is optimized for low data rates
- Better quality than H.261
- H.263 is an advancement of the H.261



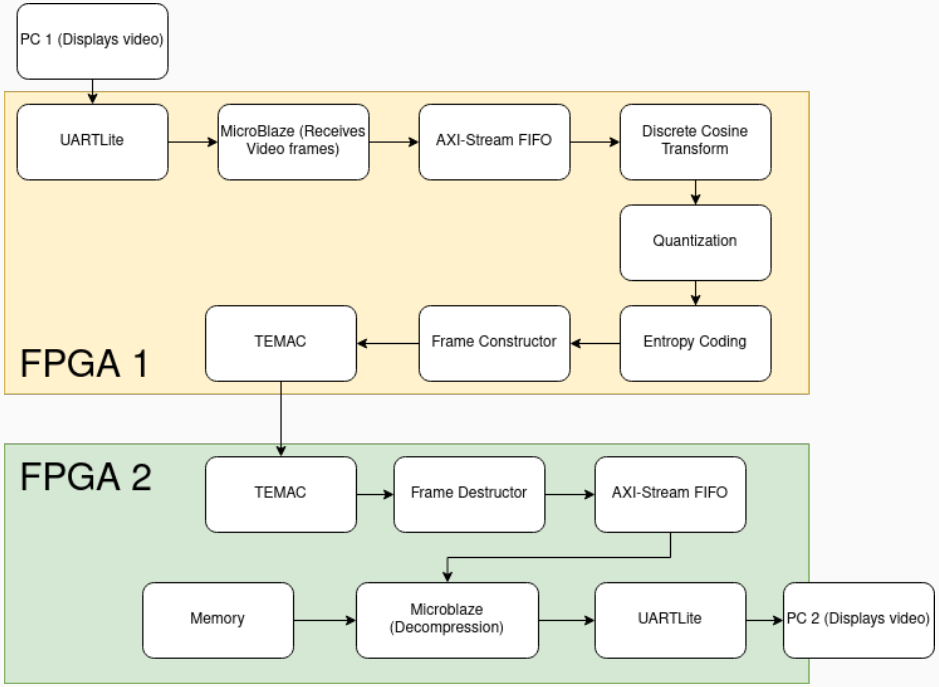
Initial Project Goals vs Final Project Demo

Initial Goals vs Final Demo (Overall System Block Diagram)

Initial

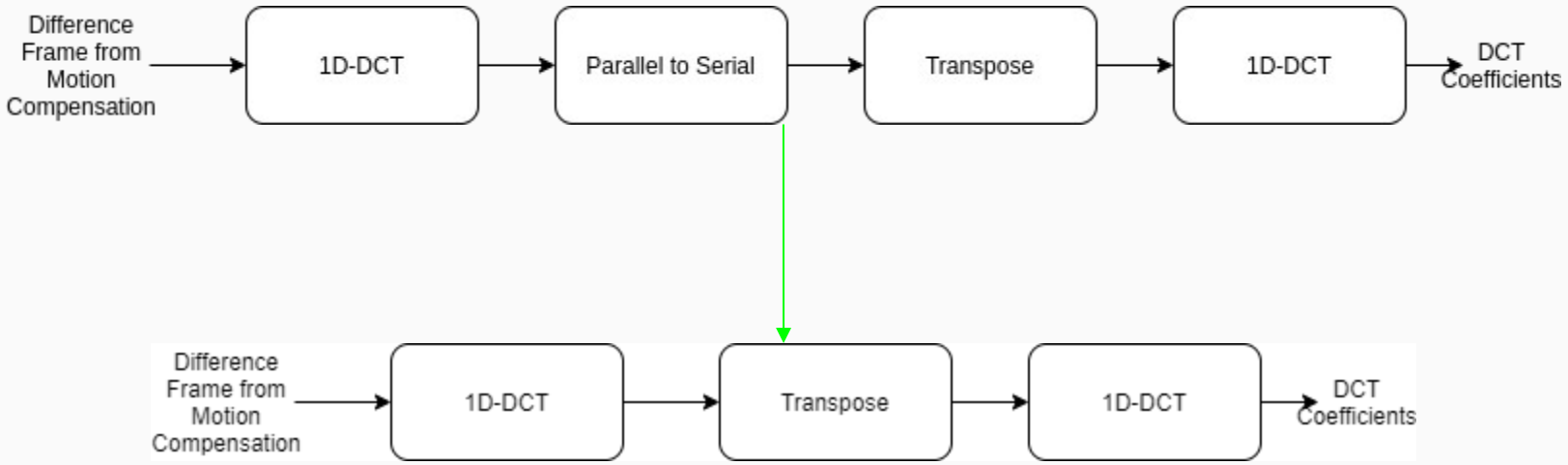


Final



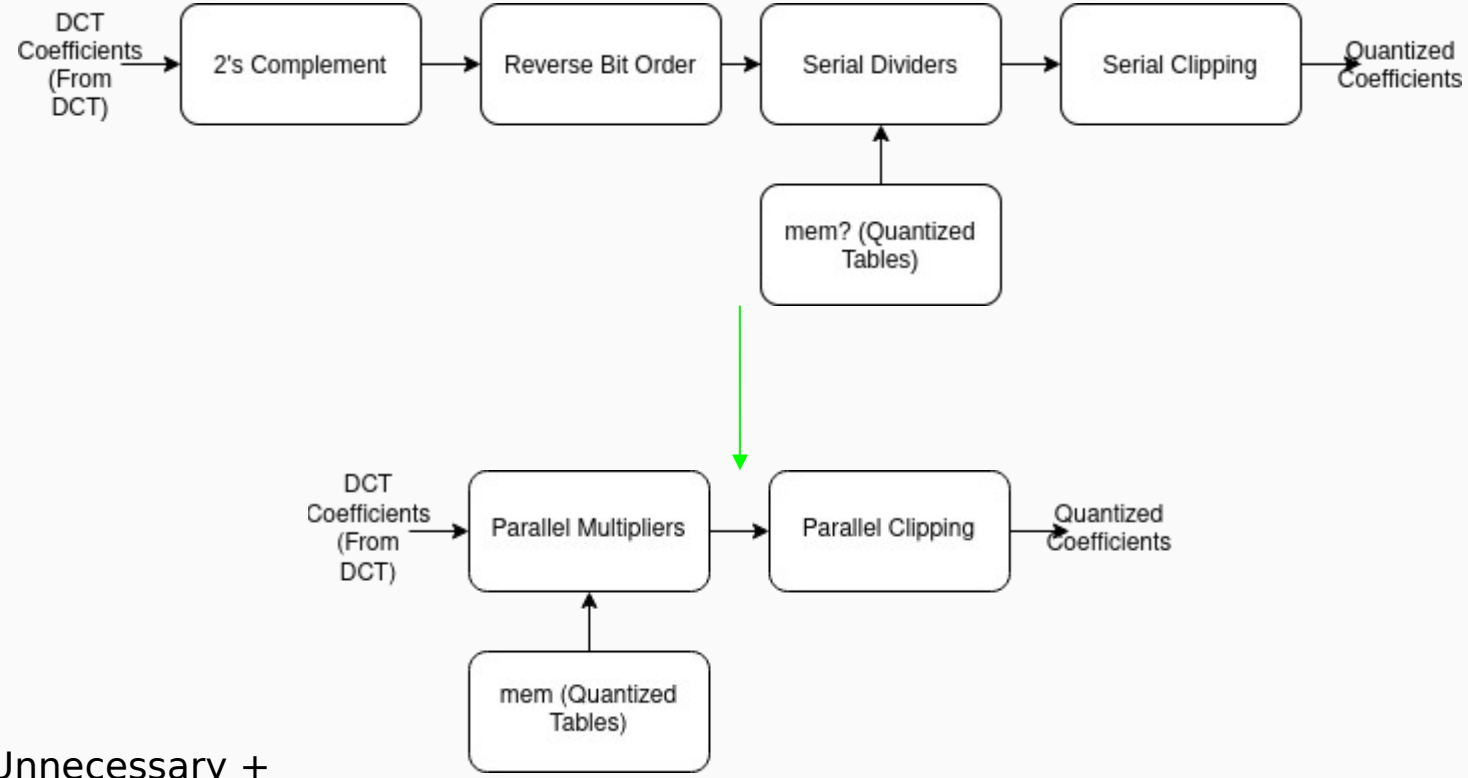
Reason: Lack of time + implementation details

Initial Goals vs Final Demo (DCT)



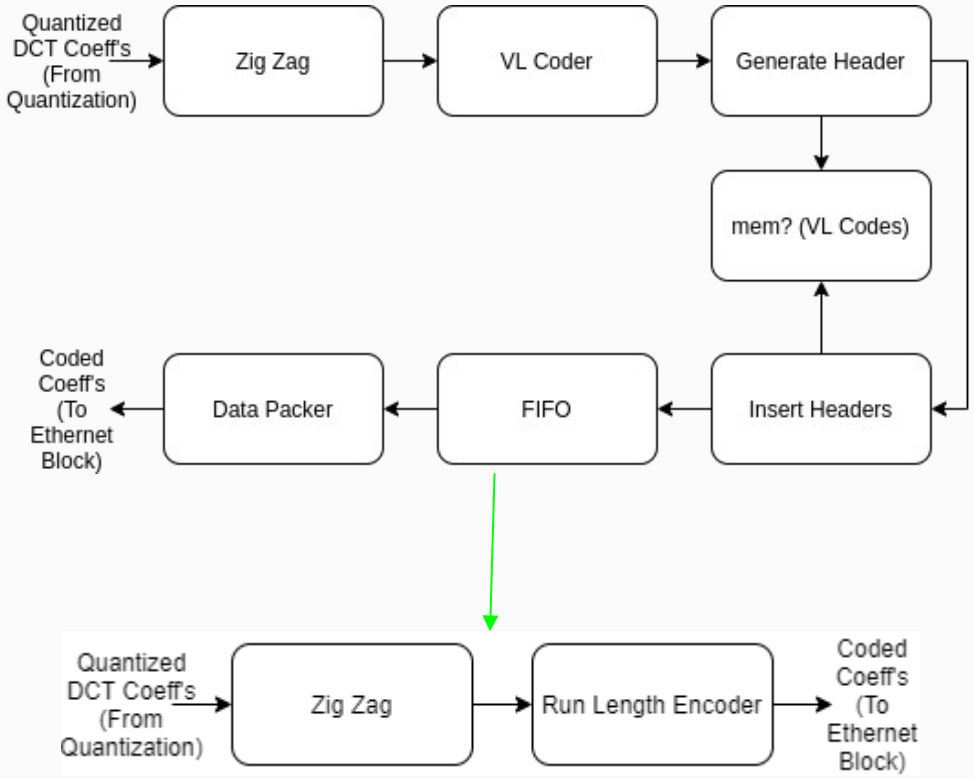
Reason: Lack of Need

Initial Goals vs Final Demo (Quantizer)



Reason: Unnecessary + Performance

Initial Goals vs Final Demo (Encoder)



Reason: Lack of time, unnecessary

```
[ 1,  2,  3,  4,  5,  6,  7,  8,
  9,  10, 11, 12, 13, 14, 15, 16,
  17, 18, 19, 20, 21, 22, 23, 24,
  25, 26, 27, 28, 29, 30, 31, 32,
  33, 34, 35, 36, 37, 38, 39, 40,
  41, 42, 43, 44, 45, 46, 47, 48,
  49, 50, 51, 52, 53, 54, 55, 56,
  57, 58, 59, 60, 61, 62, 63, 64
]
```

```
0000010000010000
0000011111110010
0000011111111110
1111010000000000
```

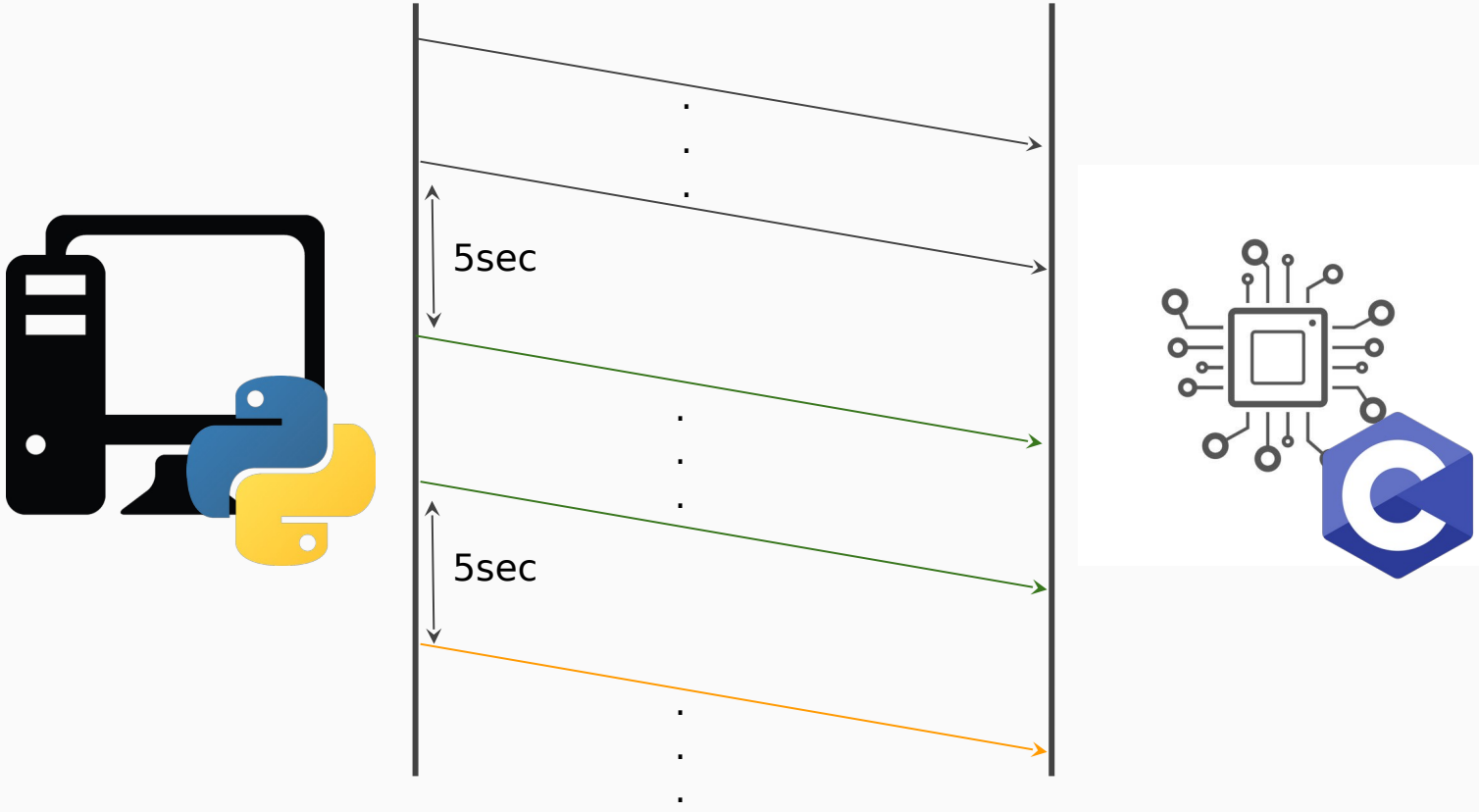
Initial Goals vs Final Demo (Other Changes)

- Switched to **Fixed Point Arithmetic** rather than **Floats**
  - **Reason**: Floats take up more memory, which is not abundant on FPGA
- Demo with 1 image rather than a demo
  - **Reason:** Time constraints, and showing functionality with 1 image means that it will work with video frames, since that will be just a collection of images
- Switched to using **TEMAC** adapter rather than **Ethernet Lite**
  - **Reason**: System required data to be sent from the microblaze through the pipeline. Thus, in order to do so, we had to facilitate FPGA connection via IP cores

Detailed Project Implementation

Desktop to FPGA communication

UART



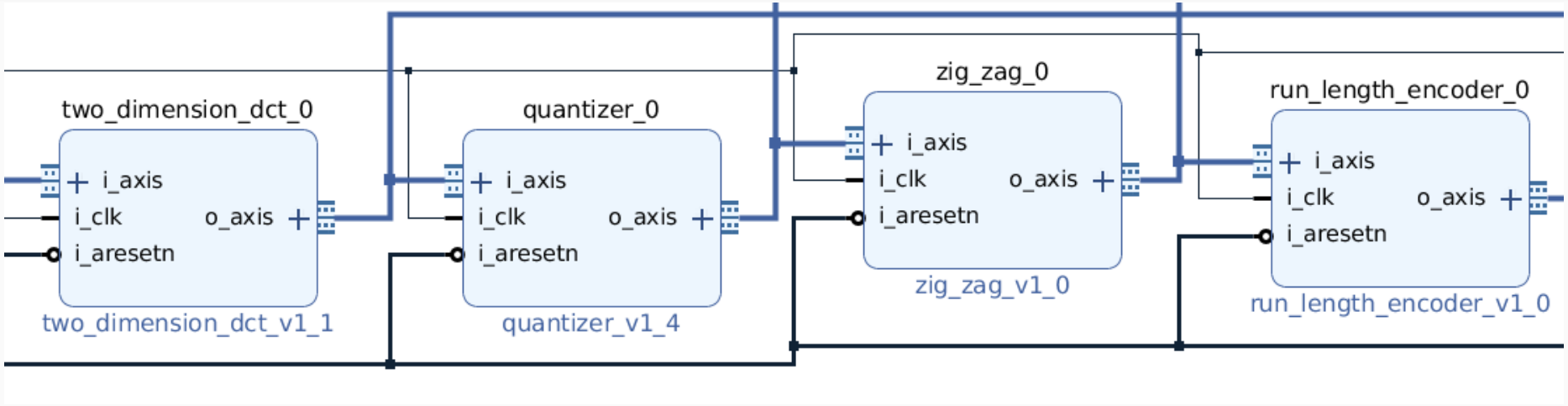
UART number conversion

```python
for i in lists[x]:
    # print("second")
    if (i >= 10) and (i < 100):
        n += 3
    elif (i >= 100) and (i < 1000):
        n += 4
    else:
        n += 2
    serialPort.write(str(i).encode())
    serialPort.write('\n'.encode())
    if(serialPort.in_waiting > 0):
        serialString = serialPort.readline()
        print("read... " + serialString.decode('Ascii'))

    # print(str(i) + "for" + "n = " + str(n))

serialPort.write('\r'.encode())

if(serialPort.in_waiting > 0):
    serialString = serialPort.readline()
    print("read" + serialString.decode('Ascii'))
```

```c
int BUFFER[65];
int n = 0;
int value = 0;
for(int i = 0; i < TEST_BUFFER_SIZE; i++){
    if(RecvBuffer[i] == '\r'){
        // xil_printf("breaking");
        break;
    }
    if(RecvBuffer[i] == '\n'){
        // xil_printf("storing value");
        BUFFER[n] = value;
        value = 0;
        n++;
        // xil_printf("n is: %d\n", n);
    }
    else{
        value *= 10;
        value = (RecvBuffer[i] - 48) + value;
        //xil_printf("values: %d", value);
    }
}
```

Video Encoding Hardware IP Cores in FPGA 1

Video Encoding Hardware



Name	Slice LUTs (63400)	Block RAM Tile (135)	D ... 1	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)	BSCANE2 (4)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)
two_dimension_dct_...	3060	0	16	0	0	0	0		54	2	1151	2988	72
quantizer_0 (design_...	1015	0	8	0	0	0	0		160	80	922	1015	0
zig_zag_0 (design_1_...	301	0	0	0	0	0	0		80	0	260	301	0
vio_0 (design_1_vio_...	102	0	0	0	0	0	0		0	0	63	102	0
system_ila_0 (design...	1646	4.5	0	0	0	0	0		56	0	837	1355	291
run_length_encoder...	299	0	0	0	0	0	0		80	40	244	299	0

2-D Discrete Cosine Transform (DCT)

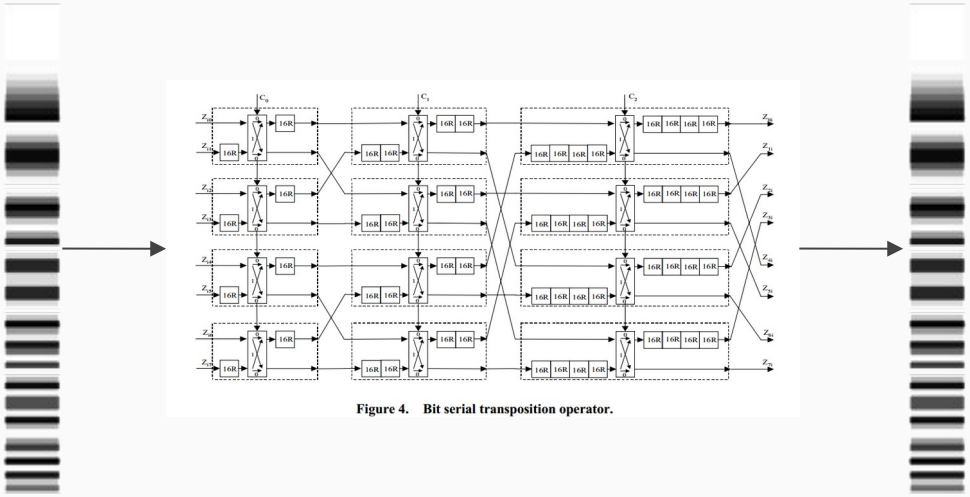
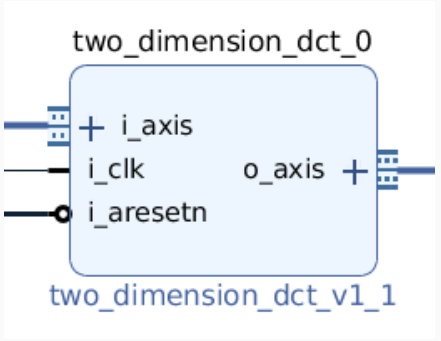
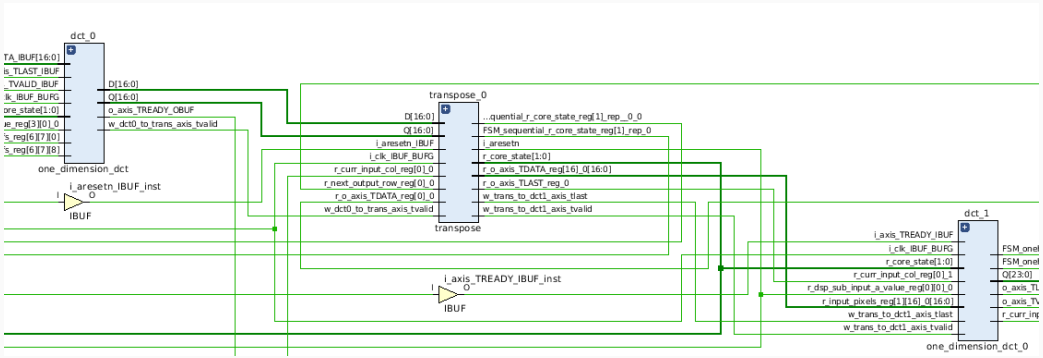
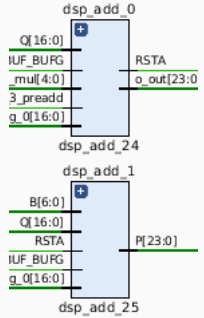


Figure 4. Bit serial transposition operator.



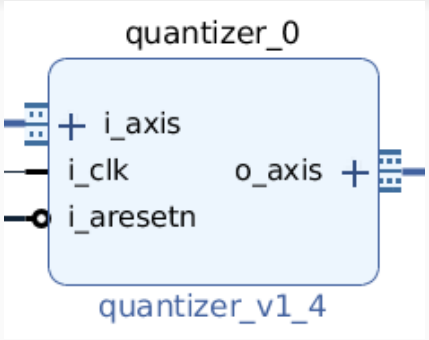


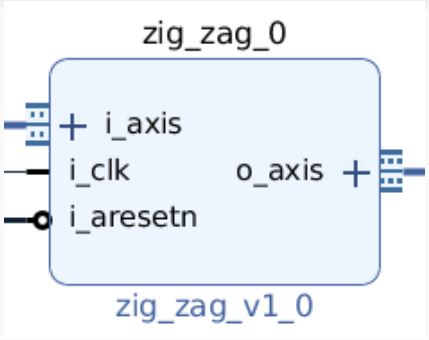


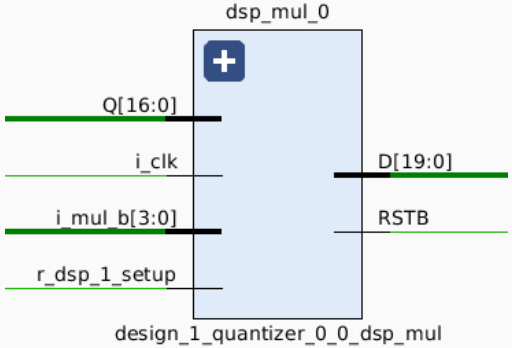
Quantizer and Zig-Zag

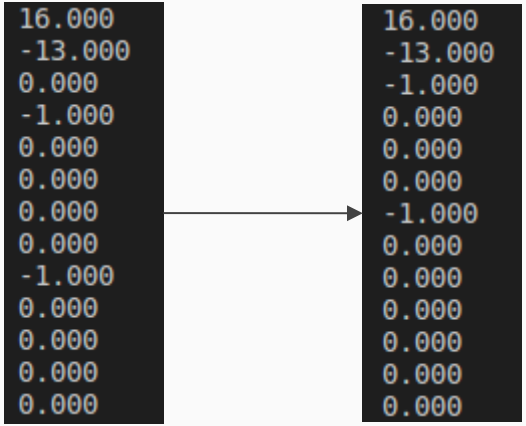
Quantization Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

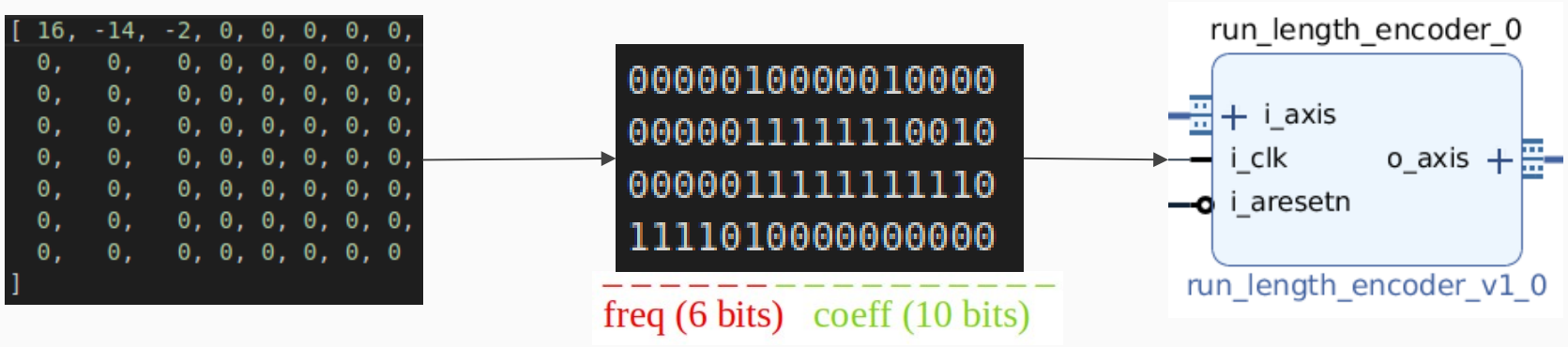








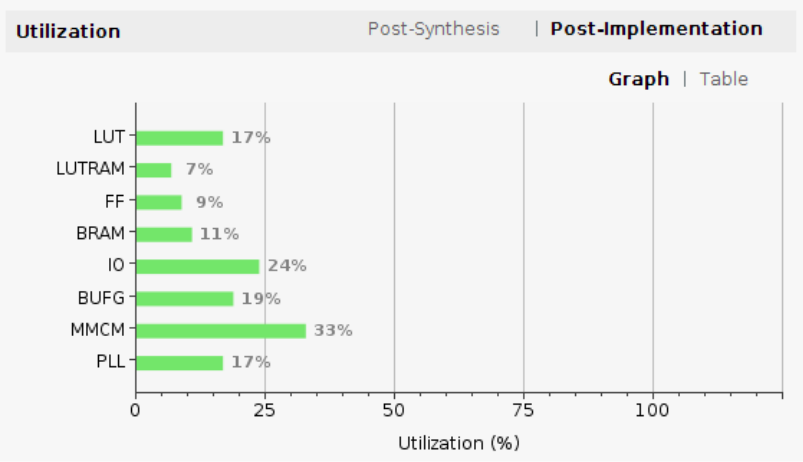
Entropy Coding



Video Decoding Using Microblaze in FPGA 2

Decoding on Microblaze (FPGA 2)

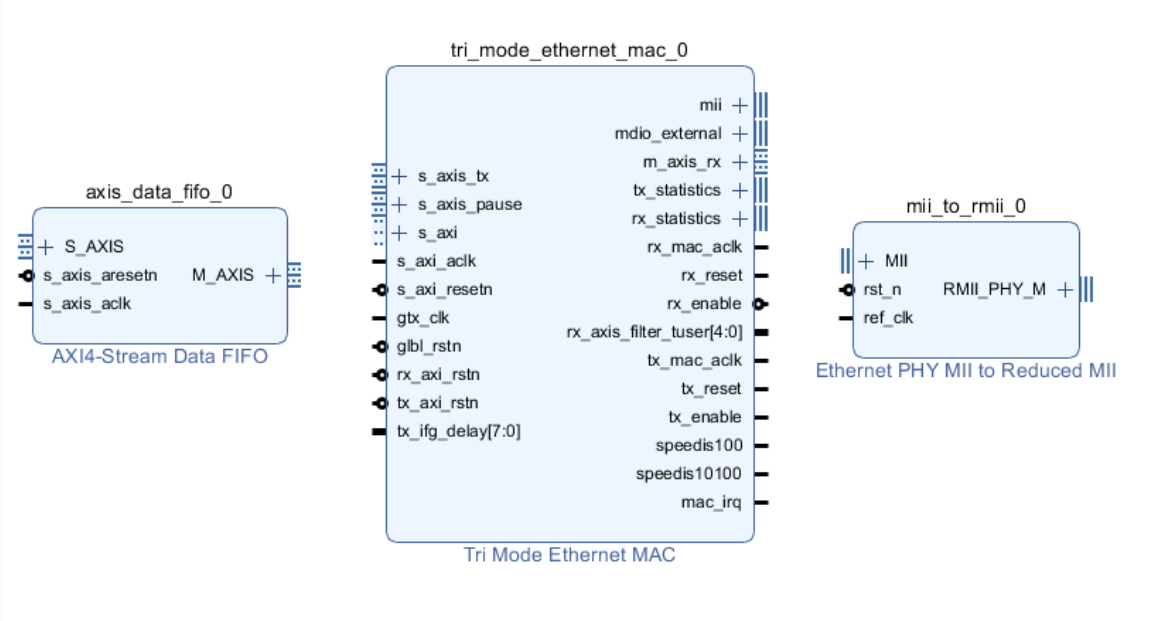
```
case STATE_COMPUTATION:
      xil_printf("Computation!\n");
       for (i=0; i<MAX_PACKET_LEN; i++)
     {
        xil_printf("Zig-Zagged Value %d: %d\n", i, zig_zagged_matrix[i]);
     }
     de_zig_zag(zig_zagged_matrix, quantized_matrix);
     for (i=0; i<MAX_PACKET_LEN; i++)
     {
        xil_printf("De-Zig-Zagged Value %d: %d\n", i, quantized_matrix[i]);
     }
     dequantizer(quantized_matrix, quantization_table, dequantized_matrix);
       for (i=0; i<MAX_PACKET_LEN; i++)
     {
        xil_printf("Dequantized Value %d: %d\n", i, dequantized_matrix[i]);
     }
     compute_inverse_dct(al_values_for_ul, al_values_for_vl, dequantized_matrix, image);
       for (i=0; i<MAX_PACKET_LEN; i++)
       {
           xil_printf("Image Value %d: %d\n", i, image[i]);
       }
    core_state = STATE_OUTPUT_TRANSMISSION;
    break;
case STATE_OUTPUT_TRANSMISSION:
    for (i=0; i<MAX_PACKET_LEN; i++)
    {
        num_bytes_sent = XUartLite_Send(&myUart, image[i], WORD_SIZE);
        while(XUartLite_IsSending(&myUart)) {}
        num_bytes_sent = XUartLite_Send(&myUart, '\n', 1);
        while(XUartLite_IsSending(&myUart)) {}
```



FPGA 1 to FPGA 2 Communication

TEMAC

Data coming out of encoder



Data going into the PHY layer

Frame Structure

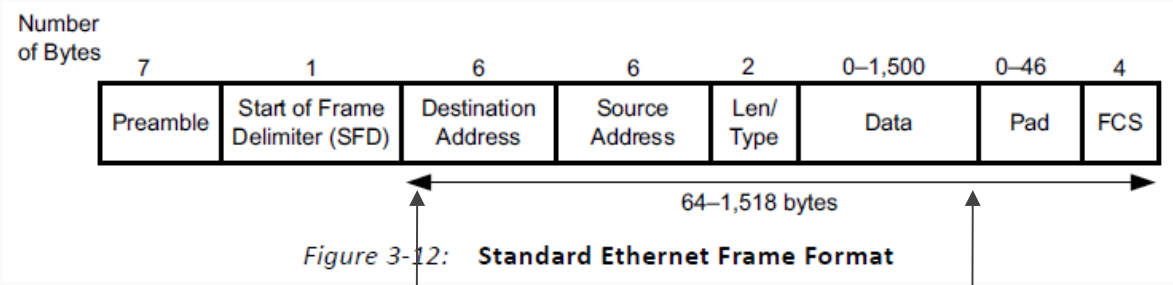
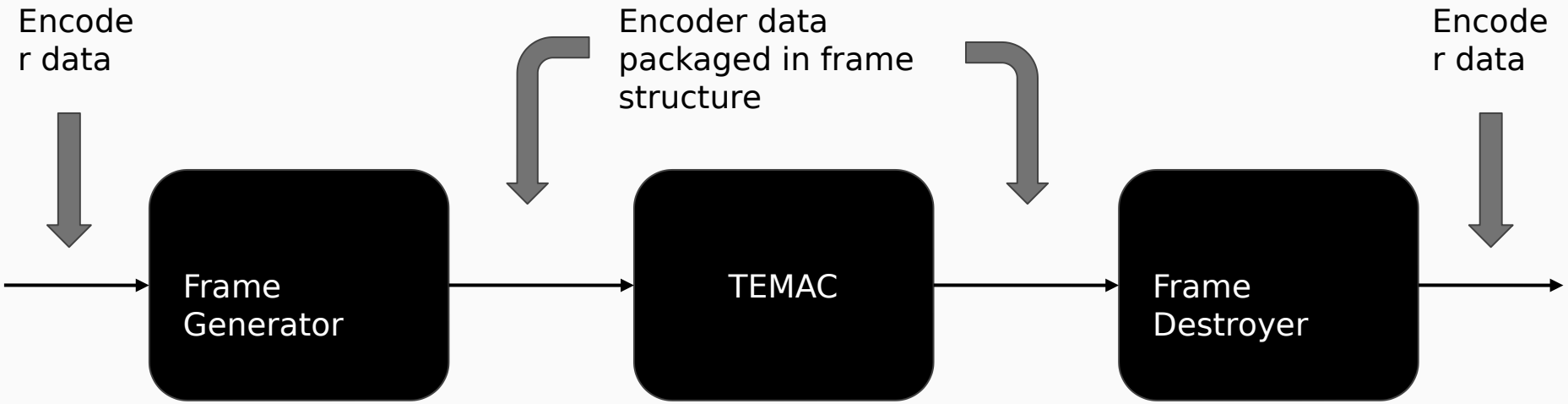


Figure 3-12: **Standard Ethernet Frame Format**

Frame structure going into TEMAC

Frame Generator and Frame Destroyer



Project Management

Task Assignment and Risk Management

Justin:

- Wrote compression algorithm for FPGA1 and decompression algorithm for FPGA2 **(Complete)**
- Risk: Wrote compression algorithm in C before implementing in Verilog, so that a Microblaze could be a potential substitute for the hardware algorithm

Bob:

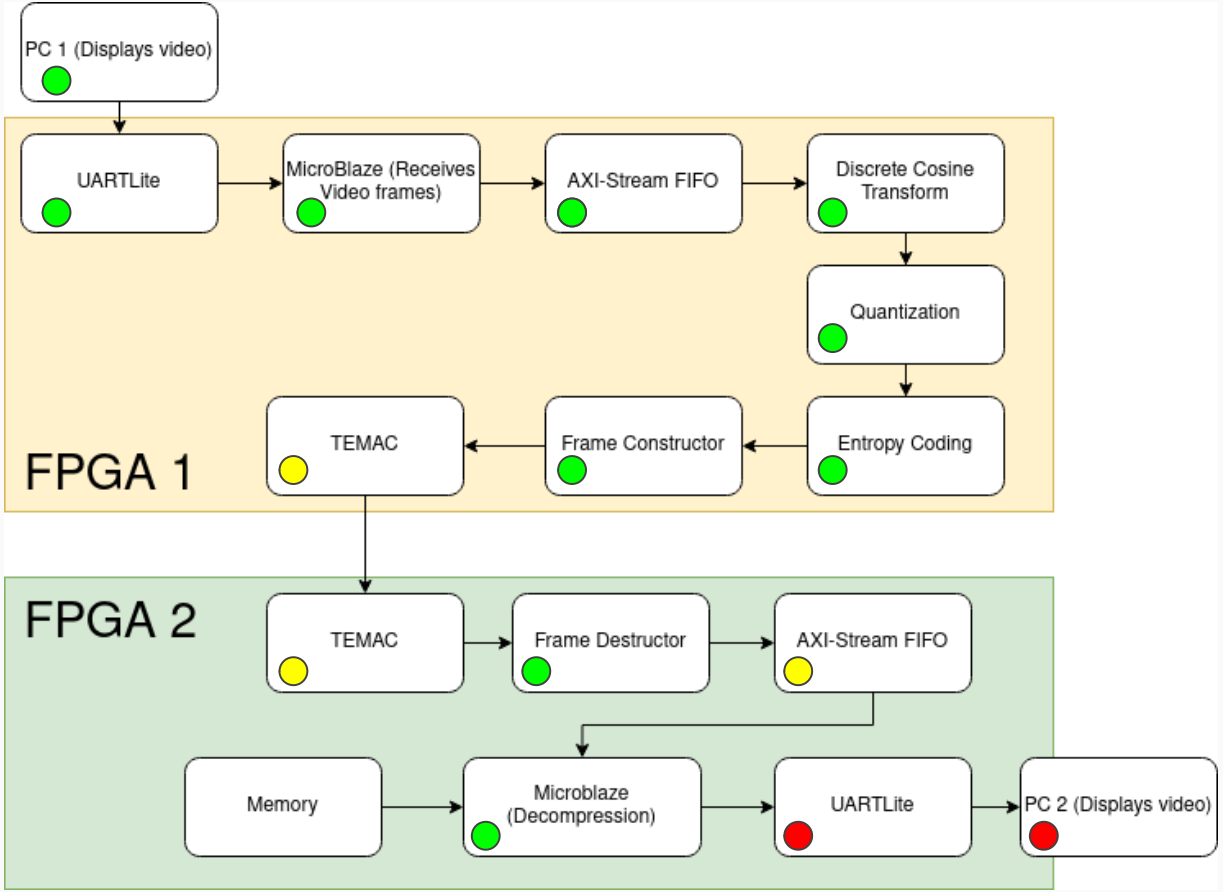
- Modified the TEMAC example design to transmit data from FPGA 1 to FPGA 2 **(Complete)**
- Created first iteration of “frame generator” **(Complete)**
- Risk: Alternative is to use LWIP, however attempting to use 2 Microblazes in one project is a also a big risk.

Isamu:

- Prepare Python script that will break apart an image and prepare packets which contain the pixel values of each block **(Complete)**
- Figuring out the UART connection and interface between command line and Microblaze via SDK **(Complete)**
- Connect data on Microblaze and feed to IP cores via AXIS FIFO **(Complete)**

Project Progress

- Complete
- Partially Working
- Incomplete



What's Left

TEMAC:

1. **Fix bug: Transmitter FIFO is not outputting data (Incomplete)**

FPGA2 AXI-Stream FIFO-to-Microblaze (AXI-Stream FIFO):

2. **Fix bug: Reading packet data interferes with FIFO's Receive Length Register (Incomplete)**
    a. **Workaround: Reconstruct packets in Microblaze using encoded coefficients (Complete)**
3. **Fix bug: AXI-Stream FIFO sometimes stops transmitting data (Incomplete)**

FPGA2-to-PC2 Connection (UARTLite):

4. **Include carriage return characters in packets to separate pixel values (Incomplete)**

PC2 Post-Processing (PC2):

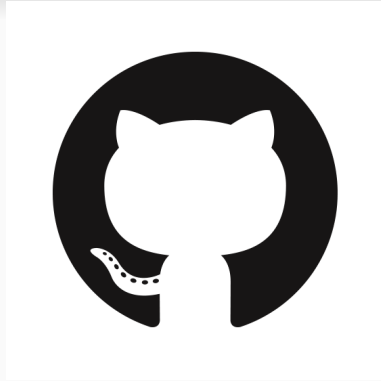
5. **Receive data from COM port (Incomplete)**
6. **Perform fixed->float conversion and display image (Incomplete)**

Problems We Encountered

- **IP performance changed when implemented on hardware**
  - Used ILA to track signals and isolate issues
  - Followed coding practices for synthesizable Verilog
- **Hardware cores don’t meet timing constraints**
  - Use Vivado to find critical paths, then alter them or switch them to DSP’s
- **Hardware cores use too many resources**
  - Reduce parallelism to reduce utilization at cost of performance
- **Unexpected behaviour due to equipment malfunctions**
  - When unexpected behaviour occurs, switched machines or worked locally
- **Time-Consuming to implement UART and FIFO**
  - Reference designs lacked detail, and no existing examples sent large amounts of data over UART using Pyserial

Open Source

Open Source



Observation: Online there is little access to hardware implementations of image/video compression. The following will be very useful to future developers:

1. Custom hardware IP cores for each component of the H.263 algorithm with DSP usage
2. UART connection to send large numbers of packets over serial port without loss of information
3. AXIS FIFO integration with UART
4. TEMAC implementation between connecting between 2 FPGAs

The above code and documentation explaining how to use them will be released publicly in our following repository: https://github.com/isamumu/ECE532

IDEA: With the above implementations released, it will give developers an easier time debugging and allow them to build more complex systems due to less risk of debugging.

What We Learned

- **Implementing UART to send numerous packets over serial communication is not trivial**
  - Not guaranteed that UART will catch the packets in order over the port
  - Catching the packets at different times cause data to get caught halfway through transmission

- **UART FIFO requires designer to know how much data to send**
  - Easier to use Xilinx drivers to send data from UART -> FIFO -> IP cores
  - Troublesome to send data from IP core -> FIFO -> microblaze

- **H.263 hardware Implementation requires many resources**
  - Should employ DSPs to reduce LUT utilization and to meet timing constraints
  - Reduce parallelism to reduce utilization at cost of performance

- **TEMAC Implementation is non-trivial**
  - IP Core examples were virtually absent online, so it was very difficult to debug
  - One obstacle is to connect two FPGAs with the TEMAC. Another obstacle is to get it working with other IP cores
- **Bottlenecks:**
  - Third party cores (TEMAC and AXIS FIFO) can be very difficult to integrate due to

Demo

Q & A

Project Complexity

Data transfer over USB

- Transfer data from Desktop to FPGA using UART + MicroBlaze + python script (0.25 points).

Desktop to FPGA network connection

- Connect using raw IP/MAC packet from python script (0.25 points).

~~**FPGA network connectivity**~~

- Connect to the network without using MicroBlaze, direct connection from hardware (0.5 point).

IP Core implemented in FPGA Hardware

- Implement the IP core with a complexity point of at least (1 point).

Performance monitoring

- Implement performance monitoring in MicroBlaze with multiple network connectivity types (0.25 points).

Software algorithm implemented on MicroBlaze

- Difficulty scales with complexity of algorithm (assume 0.1 points).

Meaningful visualization of program run/statistics gathered/results obtained

- Visualize meaningful results with a GUI (0.75 points).

Total complexity score : 3.1 points.