**University of Pennsylvania**
**Department of Electrical and System Engineering**
**System-on-a-Chip Architecture**

ESE532, Fall 2018

# I/O and Energy Milestone

Wednesday, November 7

**Due:** Friday, Nov. 16, 5:00PM

**Group:** Integrate I/O. Measure and estimate energy.

**Individual:** Writeup is an individual task.

1. Integrate I/O and validate operation with the I/O in place. Report the maximum real-time throughput the current design can sustain.
2. Move some part of your design onto the FPGA for acceleration.
   Writeup should identify what you moved onto the FPGA, how you validated it, and how you tuned it. Identify the current throughput achieved and the current bottleneck(s).
3. Turn in a tar file with your I/O integrated and FPGA accelerated code to the designated assignment component in canvas.
4. Measure and report the energy to encode the Linux source code on the two implementations corresponding to Problems 1 and 2 (if you have had a chance to explore multiple design options using the FPGA for acceleration, measure the highest performing design that you have currently found.)
5. Estimate the energy to encode the Linux source code for the same two designs based on metrics provided by the Xilinx tools.

We don't expect significant FPGA acceleration on this milestone, but we do want you to become familiar with how to measure and estimate the energy including that of the FPGA mapping so you will be prepared to characterize your more mature designs.

# Ethernet Packet Format and Ethernet MAC Interface

You can find designs for an ethernet sender and an ethernet receiver here. The provided code contains two XILINX SDK projects. One of them is used to send packet and the other is to receive.

## Sender

The sender is a complete design that you can use to send raw data at a fixed data rate over ethernet. You only need to use this design to provide data to your receiver-deduplication-compression design.

1. In order to build the project, you can either choose to open them using the SDK tool or you can create a new empty SDSoC project and copy all the source files under the src folder into the new project. (The difference between SDK design flow and SDSoC flow is that the SDK is pure software development. It doesn't support cross-compilation with Vivado-HLS).

2. The code is modified from the example of the xemacps driver provided by XILINX. You can find the detail description of the driver here. The main function for the send project is in the file `"testperiph.c"`. It contains two functions. The first one is used to setup the interrupt, and the second one is the actual function used to send the frames. In that function, there are several setup routines for the drivers. Most of them are configuring the `TxBDs` (i.e., Transmit Block Descriptors). The main function to send data is the `EmacPsDmaSent_test`.

   Each TxBD contains details on the actual transmit buffer. A single TxBD is an 8-byte block. It specifies the length of the frame and the actual address of the frame. In the example case, the TxFrame is the array that stores the actual frames. The function `XEmacPs_BdSetAddressTx` is used to assign the address of the data to send. You can find the detail of Block Descriptor in Chapter 16 of the ZYNQ TRM.

3. You can change the send and target MAC addresses by modifying the array `EmacPSMAC` and `RemoteMAC` array.

4. The function `XEmacPs_SetOperatingSpeed` is used to configure the speed of the MAC. It is called in the `EmacPsDmaIntrExample` function. You can modify the speed as you want. *This is the only thing you should need to change on the sender.*

Note: At the moment, we are not certain if this design will consistently send data at the specified speed. From the observation we have, the 1Gb/s setting appears to be achieving a lower average transmission rate (perhaps as low as 600Mb/s). However, it does send data at peak speeds around 1Gb/s. This may just be an artifact of our current measurement. We are exploring further and may provide an updated design and further clarity on the speed it provides.

5. The encoder can read a file to be sent and compressed from the SD-Card. It will pickup the file `raw.dat` in the top-level of the SD-Card and use that as the send data. (**Updated Nov. 10**)

   Relaunch the project to get the sender to run again (rereading `raw.dat` from the SD-Card and sending it to the receiver).

## Receiver

The receiver is a sample design showing you how to use ethernet input. The design we give you simply stores the data received into DRAM then discard the frames. You will integrate this with your design to include your deduplication and compression between the data reception and storage onto DRAM.

1. You can follow the same build process for the project discussed for the sender.
2. The basic structure of the code is the same as the send project. However, unlike the send project, the recv project uses a ring buffer instead of queue. When the MAC receives a frame, it will trigger an interrupt. The interrupt will increase the `FrameRx` in `XEmacPsRecvHandler`. The main thread retrieves the buffer using `XEmacPs_BdRingFromHwRx`. This function will also return a count of the number of packets that were actually received. Your code will need to process all these packets. In the example, the packets are visited in the `for` loop using the pointer `FrameP`. It then frees the buffer and returns it to the hardware. If your processing speed is slower than rate at which the data is arriving, the hardware will run out of receive buffers and not be able to store the input data. As a result, data will be lost, and your code won't see the complete data stream.

## Ethernet Packet Format

The ethernet frame packet uses the standard ethernet frame header. It contains the source and destination MAC address (6B each) and the length of the payload (2B; does not include header bytes). Also, there is an extra 2B field 0x0129 after the ethernet header , which is a specific identifier for the frame. This is designed to distinguish any packets that are not part of the sending stream.

| Header | | | Payload | |
|---|---|---|---|---|
| dst MAC | src MAC | length | identifier (0x0129) | data to compress |
| 6 | 6 | 2 | 2 | 1–1498 |

## Measure Energy

The ZedBoard provides a current sense probe (J21). You may find this posting useful on how to use and interpret. The current sense measurement and calculation gives you power. You will need to also consider the total end-to-end runtime to compute energy. Note that this includes everything on the ZedBoard, not just the Zynq chip.

## Estimate Energy

You can obtain the power by opening the Vivado project file generated by SDSoC in Vivado (like we did before to show the block diagram), and opening the *Project Summary*. The power consumption is given as *Total On-Chip Power* in the *Power* section at the bottom right. The same section also has a tab called *On-Chip* that shows the dynamic and static power. Dynamic power is further divided into clocks, signals, logic, BRAM, and PS7.