apertus
open source cinema

hand built
constantly evolving
absolute camera control and total freedom

AXIOM Beta

by apertus

USB 3.0 Plugin Module

Apurva Nandan

Supervised by Herbert Pötzl

Google Summer of Code '19

APERTUS.ORG

First edition, August 2019

Contents

Part One

1. GSoC'19 Project Report

1.1 Overview

The AXIOM Beta features PCIe x1 slots on the main board to allow optional plugin modules to extend the IO capabilities of the camera. One such module, the USB 3.0 plugin module, is designed by apertus° Association to allow live stream of 4K video at 20+ FPS to a PC for video processing & recording purposes. My aim in this GSoC project was to develop gearwork HDLs for the USB 3.0 Plugin Module that can provide a throughput of at least 3.0 Gbps from the AXIOM Beta camera to the PC with Bit Error Rate (BER) lower than 10^{-12}.

All the source code written by me during the period of GSoC'19 can be found here:
https://github.com/apurvanandan1997/usb_plug_mod_ber

For detailed documentation of the gearwork of the USB 3.0 Plugin Module, please go to Part Two of this report

1.2 Community Bonding Period

- This period was mainly spent in setting up the work environment, both in terms of hardware & software. Significant time was spent in getting acquainted with the existing code base of AXIOM Beta firmware and going through all the documentation relevant to the project.
- On the hardware side, I learnt how to operate the Remote Beta Camera available at Apertus° Remote Testing Lab, Vienna. I also set up the USB 3.0 Plugin delivered by apertus° locally with a XUPV5-LX110T FPGA. The local set up comprised of USB 3.0 Plugin attached to a breakout board powered by LDO regulator and connected to Virtex-5 using 6 LVDS connection pairs made using dupont cables.
- On the software side, all the manufacturer's tool, namely Xilinx Vivado, Xilinx ISE, Lattice Diamond and Modelsim, were installed on Ubuntu Linux, thereby allowing easy exchange of project source files with the mentors. Also, urJTAG 2018.9 was installed, which allowed programming the USB 3.0 Plugin module using Digilent XUP-USB-JTAG cable, to facilitate testing on the local hardware.

1.3 First Coding Period

- In this period, my major focus was on developing a FTDI FT601 controller for the plugin module that can receive 32-bit words at 100 MHz and transmit them to the D3XX driver running on a PC through USB 3.0 port, thereby providing 3.2 Gbps throughput.
- The FT601 controller developed by me finally was able to provide a maximum throughput of 3.0 Gbps (limit of the FT601 chip as per mentioned in it documentation). The catch here was that though the clock from FT601 is at 100 MHz but the internal FIFOs of this chip soon get full and the FIFO_FULL output signal from the chip is asserted. This implies we get frequent pauses in data transfer of roughly 0.2us at about every 3ms which reduces average throughput from 3.2 Gbps down to 3.0 Gbps. In the case of the internal FIFO of the FT601 chip getting full, we need to stop sending data to the chip or else our gearwork's FIFO gets full and there is significant loss of data due to overflow. I wasn't able to resolve this in my project as the current technique of data transfer between the FPGAs is purely unidirectional and only some kind of feedback to the Transmitting FPGA (Zynq) from the Receiver FPGA (MachXO2) can resolve the issue.
- However, the fraction of time the internal FIFOs of the FT601 chip remains full is largely dependent on the CPU core resources and also on the efficiency of D3XX driver code. Using a light D3XX code with a powerful CPU setup, I was able to reach 3.0 Gbps with significantly less errors, although pausing the data input to the gearwork's FIFOs was able to provide 100% accurate data throughput.
- All tests were made on local and remote hardware. Code written by me in this period can be found here:
https://github.com/apurvanandan1997/usb_plug_mod_ber/blob/master/MachXO2/src/ft601.vhd

1.4 Second Coding Period

- In the second coding period, I implemented the HDLs for Transmitter/ MicroZed side of gearwork for Zynq XC7020 and Virtex-5 LX110T FPGAs. These HDLs comprised of the OSERDES blocks, 8b10b encoder (8b10b encoder and decoder were taken from OpenCores.org under GPLv3) and Fibonacci Pseudo-Random Number Generator for testing the BER of the LVDS connections.
- The transmitter HDLs have been completed, yet there are additional features that can be incorporated in them, namely error reduction techniques like ECC CRC and use of more control symbols available for extending out of the band communication.
- All transmitter HDLs written by me can be found in these two folders:
https://github.com/apurvanandan1997/usb_plug_mod_ber/tree/master/Zynq
https://github.com/apurvanandan1997/usb_plug_mod_ber/tree/master/Virtex-5

1.5 Third Coding Period

- During the third coding period, I implemented most challenging part of this gearwork which were the HDLs for the receiver FPGA (MachXO2). These HDLs comprised of DDR x4 input gearing, link training, channel bonding of 5 LVDS lanes, 8:10 gearing, 8b10b decoder(8b10b encoder and decoder were taken from OpenCores.org under GPLv3), Clock Domain Crossing (CDC) FIFO and BER calculator.
- After completing all these HDLs, I was able to run various important tests and measurements on USB 3.0 Plugin Module, among which the most significant was the BER testing of the LVDS lanes. The LVDS connection and the DDR x4 gearing primitive have been tested and

verified using the BER testing model made by me. I hereby report the Bit Error Rate (BER) measured by my code on both local Virtex-5 hardware and remote AXIOM Beta hardware:
BER measured on AXIOM Beta over 5 LVDS lanes at 375 MHz DDR $\approx 10^{-12}$
BER measured by Virtex-5 FPGA over 5 LVDS lanes at 375 MHz DDR $\approx 10^{-9}$

Static delay have been used on the LVDS data lanes for aligning the edge of data transition with the high speed DDR clock edge. These static delays have been adjusted to work perfectly on the AXIOM Beta camera.

- All receiver HDLs written by me, for the MachXO2 on the USB 3.0 plugin module, can be found on the following link:
https://github.com/apurvanandan1997/usb_plug_mod_ber/tree/master/MachXO2

1.6 Tasks Pending

- Implementing 40-bit to 32-bit gearing between the 5 8-bit LVDS lanes and the 32-bit CDC FT601 FIFO to allow data transfer along with the 32-bit BER.
- Incorporating feedback signals from the MachXO2 FPGA on USB 3.0 Plugin Module to the Zynq FPGA on the AXIOM Beta which would control the throughput of the data over the 5 LVDS lanes.

2. Future Goals

2.1 Error Correction Techniques

Although the Bit Error Rate (BER) achieved over the 5 LVDS lanes on the AXIOM Beta is very low ($\approx 10^{-12}$), yet implementing the widely used error correction techniques like Error Correction Codes (ECC) & Cyclic Redundancy Check (CRC) and can even allow us to overclock the DDR x4 primitives a little, thereby increasing the throughput through the LVDS lanes.

2.2 Featuring the FT602

FTDI FT602 is a proprietary solution for plug and play type USB devices, so it doesn't require any special software/drivers on most computers and is supported by many third party applications. FT602 takes the data input in form of 32-bit video data in YUV422 format at 100 MHz clock and also provides the option of compressed video data streaming. Hardware for USB 3.0 Plugin Module featuring the FT602 has recently been designed by apertus° Association and is ready to allow furhter gearwork development.

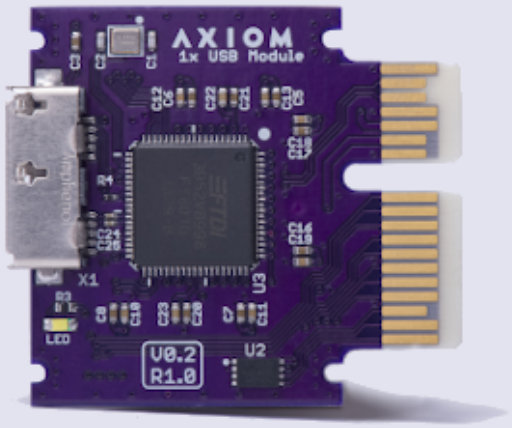
2.3 Higher FPS at Same Throughput

A 4K 12-bit video at 20+ FPS is definitely not sufficient as the CMV12000 sensor sends 4096x3072 sensel at up to 150 FPS. In GSoC'19, a Lossless JPEG 1992 Encoder on AXIOM Beta has been implemented by one of the students under apertus° Association. To get higher frame rates, I would like to integrate my current gearwork with the JPEG encoder core thereby allowing video streaming at even higher FPS at same throughput by the USB 3.0 Plugin Module.

2.4 Bidirectional Data Transfers

As both FT601Q and FT602 allow bidirectional data transfer over USB 3.0, additional functionality and accessibility can be added to the USB 3.0 plugin module by making the data link from AXIOM Beta to the PC bidirectional by reverse implementation of the current gearwork.

II Part Two



3. Gearwork Documentation

3.1 Overview

The RAW data captured from the CMV12000 sensor on the AXIOM Beta camera gets directly buffered on the DDR3 SDRAM of the MicroZED board. The main board of AXIOM Beta featrues PCIe x1 connectors which exposes 6 LVDS pairs (routed to the MicroZED board through JX connectors) and 8 GPIO pins (routed to West Routing Fabric (RFW)). In order to transmit the RAW video data from the DDR3 SDRAM of the AXIOM Beta, the USB 3.0 Plugin Module has been developed which would act as a 6-Lane LVDS connection to USB bridge between the camera and the PC. The USB 3.0 Plugin Module features a Lattice MachXO2 FPGA, on which the receiver gearwork logic has been implemented. A proprietary FTDI FT601 chip has been featured on this plugin module which inputs the a 32-bit data at 100 MHz through FIFOs and forwards them to the USB 3.0 port, offloading the conversion of data packets to USB 3.0 protocol from the MachXO2 FPGA

The complete data flow from the DDR3 SDRAM on AXIOM Beta to the software driver on the connected PC can be described as follows:

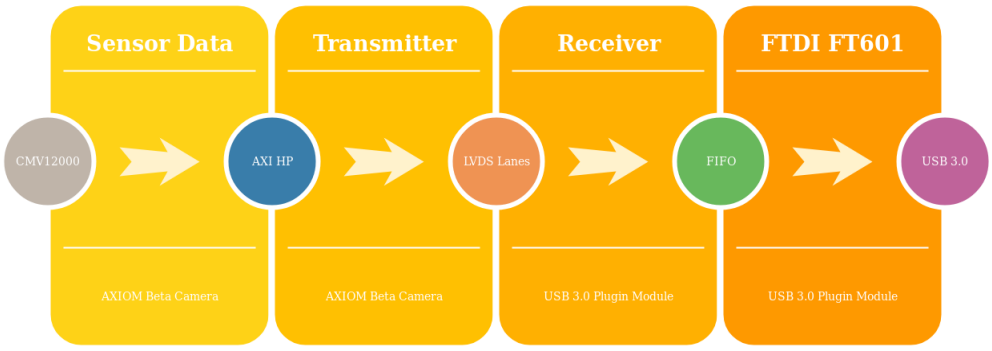


Figure 3.1: Gearwork Overview

3.2 Transmitter Gearwork

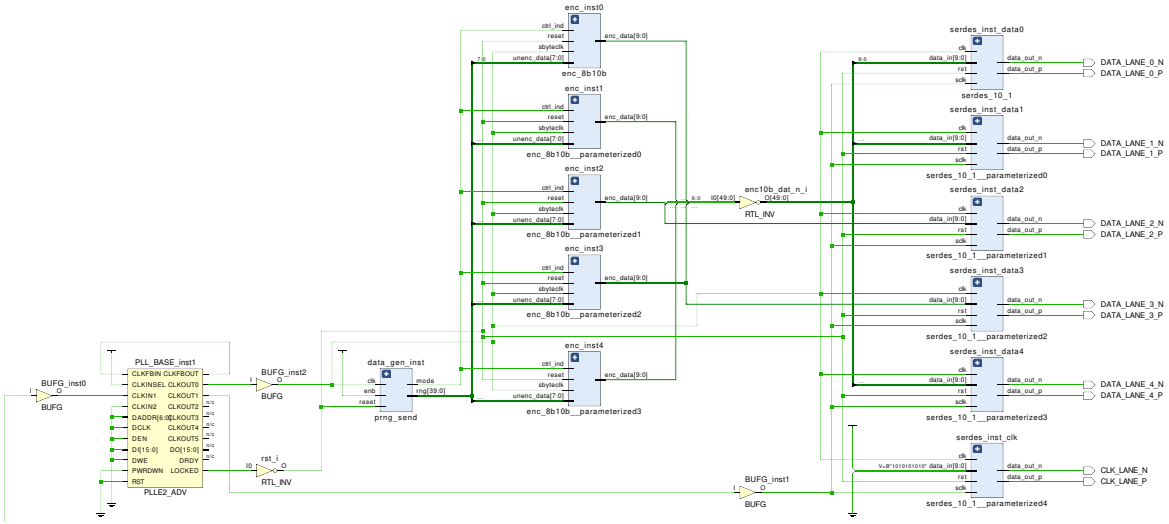
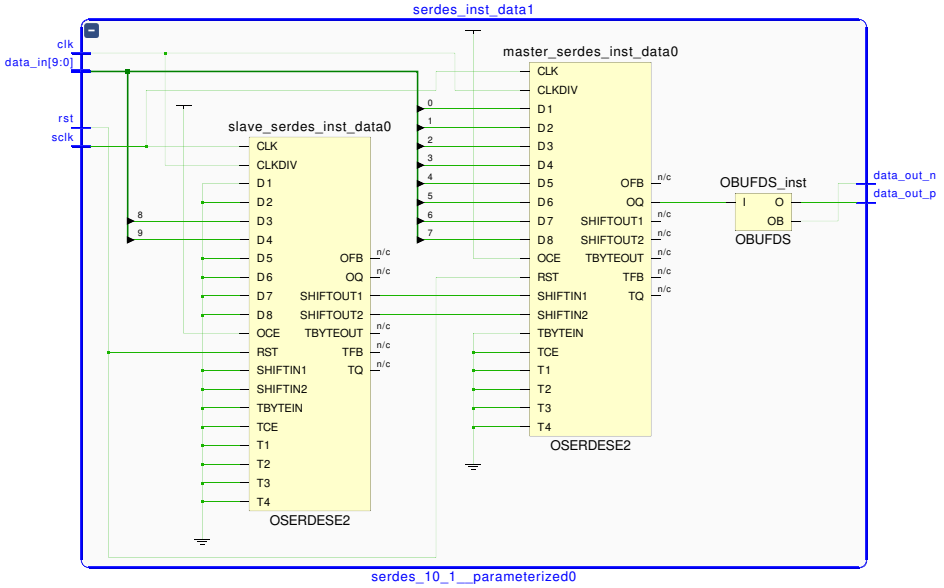


Figure 3.2: Zynq Gearwork Schematic

The above figure describes the transmitter part of model developed for BER testing.

- The PLL_ADV instance does the task of generating the required clocks from the 50 MHz clock being received from the Zynq PS. The sclk is the fast serial clock at 375 MHz while the clk is the slow divided clock at 75 MHz.
- The data_gen_inst has incorporated a 40-bit Pseudo Random Number Generator generating random number clocked at 75 MHz. It also generates the link training/word alignment pattern (K28.5 = BC) until the link training is complete on the receiver side. It uses all zeros as a synchronizing code for the PRNG present on the plugin module.
- The random 40-bit wide data generated from generated from PRNG gets divided into grouping of five, i.e. 8-bit words which get forwarded to the 8b10b encoder.



- The 10-bit encoded data generated from the encoder instances are forwarded to the SerDes units. serdes_inst instances do the task of serializing the 50-bit data over five LVDS data

Lane. The fast serial clock sclk is also sent through LVDS lane using serdes_inst instance to match the delay in clock lane with the data paths. The OSERDESE2 units have been used in the cascading configuration using the SHIFT* pins to expand the data width to 10-bit.

3.3 Receiver Gearwork

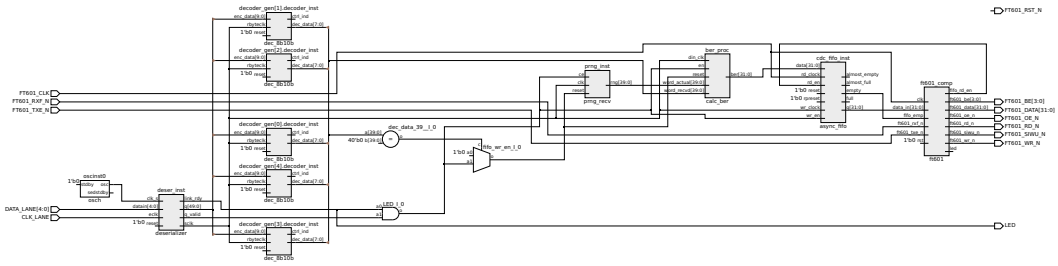
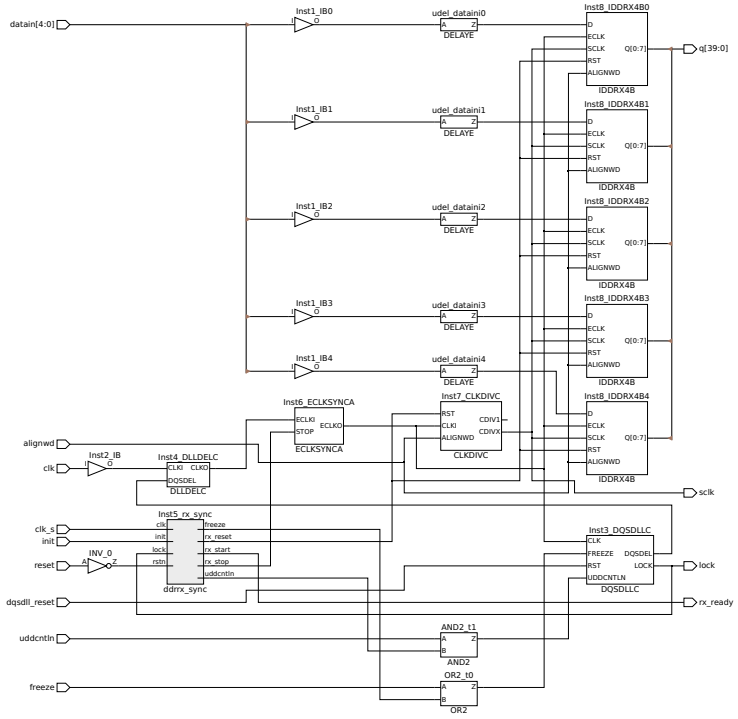


Figure 3.3: MachXO2 Gearwork Schematic

The above figure describes the receiver part of model developed for BER testing.

- The high speed serial links set up between the MicroZED and the plugin module follow the source synchronous system with the assumption that initially data and clock are edge-aligned. Static delays have been adjusted in the data path to achieve the same on the AXIOM Beta.
- The edge aligned clock is forwarded to the DLLDELC primitive which can add dynamic delay to the clock with DQSDEL delay input code from DQSDLLC primitive. DQSDLLC ensure that clock gets a 90° phase shift, thereby making it centre aligned with the data while also adjusting for the PVT variation.



DDRx4 Gearing Schematic

- RX Sync is a soft IP provided by Lattice that ensures proper reset synchronization of all the IDDRX4B and CLKDIVC primitive to maintain the clock domain crossing margin between fast edge and divided word clock, and to avoid bus bit-order scrambling due to the various delay of the reset pulse. ECLKSYNCA is used to send the fast edge clock to the edges of the MachXO2 with minimum skew and also for reset synchronization of IDDRX4B & CLKDIVC. (Refer to Implementing High-Speed Interfaces with MachXO2 Devices-TN1203)
- As MachXO2 doesn't support 10:1 gearing, the data had to be first geared down to 8:1 with the IDDR4XB primitive, then consecutively gear down to 10:8 with a barrel shifter. The IDDR4XB provides an option for bit slip with the input signal ALIGNWD, and as the lengths of all data lanes have difference of few millimeters , i.e. less than few picoseconds in terms of delay, all IDDRX4B instance need equal number of bit slip to get generate aligned words.
- The deserializer instance performs the link training, channel bonding, 10:8 gearing and finally outputs the five 10-bit encoded words received from each LVDS lanes at 93.75 MHz with 4/5 times valid signal.
- The encoded 10-bit data is sent to 8b/10b decoders where after decoding they get compared with the data generated by the PRNG instance (having the same SEED and synchronized to PRNG on Zynq using a all zeros word). The calc_ber instance counts the number of bit errors, i.e. the number of unequal bits between the received data and the locally generated data, and updates the BER result every time after analysing 2^{30} 40-bit words.
- The 32-bit BER results are en-queued in asynchronous FIFO allowing clock domain crossing between the received data clock and the 100 MHz data input clock from the FT601 chip. These 32-bit BER values are then fetched by the FT601 controller instance from the FIFO and sent to the FTDI FT601 chip for conversion to USB 3.0 packets, which are then streamed to the connected PC.
- On the software end, we use a proprietary FTDI D3XX driver, whose documentation and APIs can be found here

---

***