

GSoC 2020 Final Report

Linux Kernel Driver for Lattice MachXO2 programming/debugging

Swaraj Hota (@bluez_)

<swarajhota353@gmail.com>

Mentor: Herbert Pötzl (@Bertl)

August 31, 2020



Contents

1	Description	3
2	Work Product	3
3	Work Summary	4
4	Current Status and Future Work	6
5	Challenges faced	7
6	In Conclusion...	7

1 Description

The aim of this project (task [#T729](#)) was to make a Linux Kernel Driver to program and debug the Lattice MachXO2 FPGAs present in AXIOM Beta. These FPGAs are used as routing fabrics to handle all the low speed GPIO communications that are required for plugin modules, shields and the centre area without sacrificing valuable Zynq GPIOs.

Two main goals of this project were:

- To implement an “upload” interface to program the FPGAs
- To implement a debug (JTAG) interface to interface OpenOCD

Previously, both the MachXO2 FPGAs were programmed with python scripts which didn’t allow easy update and development. The Linux Kernel Driver acts as a central entity to manage the FPGAs, provides a JTAG interface to any application like OpenOCD to make SVF replays possible, and essentially makes testing and debugging new code much easier.

2 Work Product

All the code produced by me during GSoC 2020 can be found in this Github repository:

<https://github.com/Swaraj1998/axiom-beta-rfdev>

3 Work Summary

Initially I had some idea about Linux Kernel but I never had a chance to actually dive into the development. I found this project on GSoC and saw this as an opportunity to finally do that. I started with the qualification task (task [#T884](#)) and eventually started learning more about driver development. Code can be found [here](#). In the ***pre-community bonding period***, I learned a lot about the hardware of AXIOM Beta and about I2C and JTAG protocols.

I spent much of my time during the ***community bonding period*** studying about the I2C subsystem in Linux Kernel, and also about the whole driver subsystem in general. I built my own kernel and setup the uboot configurations for (partial) AXIOM Betas in the remote setup where I did all my testing. I got a basic skeleton driver up and running in the remote betas during this period.

In ***Phase I***, my milestone was to finalize the “upload” interface in the driver, which I was able to achieve in time. The driver claimed the PIC addresses (0x40 - 0x4f) of the “selected” PIC (PIC West or PIC East), with the help of a simple devicetree description. The driver was then integrated with [Linux FPGA Manager Framework](#), which provides a central sysfs interface for the driver at `/sys/class/fpga_manager/fpga#`, and also some callback functions to manage the programming of FPGAs. These callback functions were then filled in with appropriate device specific implementation. For testing, I built my own compressed bitstream using Lattice Diamond tools which was successfully uploaded into the FPGA’s SRAM. I also added some useful sysfs attributes to read out idcode and status register bits within this period.

In ***Phase II***, my milestone was to figure out a debug interface for OpenOCD and implement at least some debug functionalities, which I was not able to meet within this phase due to some unexpected issues and some underestimation on my part. I spent much of this phase improving various

aspects of the driver. More useful sysfs attributes, more abstract functions, and printing proper debug info were among them. The important part (which I got stuck on) was to make the driver work with both the PICs (and hence both the FPGAs) by *seamlessly switching* between them. This switching (or “selecting”, as mentioned before) was previously done with a python script (rf_sel.py), and hence the driver could only work with one FPGA at a time, whichever was selected before loading the driver. Now, doing this *seamless switching* turned out to be more difficult than it looked, but it was nonetheless done with less changes on the driver part but more on the devicetree part.

Depending on the power board version of the AXIOM Beta, there were two different muxes (PCA9540 & PCA9543) and another analog switch (TS3A4751) to work with. For the muxes, there was an existing Linux Kernel Driver (i2c-mux-pca954x) which was used through appropriate device-tree entries. This driver created two new multiplexed I2C buses which my driver could use to talk to each PIC. For the analog switch also there was an existing Linux Kernel Driver (i2c-mux-gpio) which was used similarly (albeit through GPIO expander present at I2C bus 0 in AXIOM Beta). All these were eventually implemented though I could not complete it by the end of this phase. The issue turned out to be an error in the devicetree entry which, due to lack of proper documentation, was very difficult to find and fix. But in the way I did learn a lot about devicetree and Linux Kernel debugging.

Finally, in **Phase III**, I was able to fix the issue regarding the *seamless switching*, after lots and lots of debugging. I provided a devicetree overlay for each of the muxes/switch, the appropriate one of which must be loaded before loading the driver. The driver can then automatically take care of which FPGA to talk to. Two different FPGA managers are registered (one for each FPGA), having separate entries under `/sys/class/fpga_manager/`. Once this was taken care of I could finally focus on the debug interface of the driver. We decided to go for an ioctl interface very similar to one mentioned in a new (currently unpatched) [Linux Kernel Driver patchwork for JTAG support](#). The interface could be accessed via `/dev/rfw` and `/dev/rfe` for routing fabric west and east respectively.

After I had the ioctl interface finalized (and tested with sample application code), I started working on OpenOCD codebase to write a debug adapter for it. In the very last week, after some rounds of testing and debugging, I was finally able to produce a patch for OpenOCD that worked with my driver. I tested it with SVF replays to flash each of the FPGAs, which was successfully executed by OpenOCD using my driver.

4 Current Status and Future Work

The initial goal of the project, as mentioned in GSoC 2020 Task [#T729](#), is successfully completed. The driver can currently upload a compressed bitstream produced by Lattice Diamond tools through the FPGA Manager Framework (using "firmware" attribute). Some useful sysfs attributes like idcode, stat, statstr, digest, traceid, and usercode are added (details can be found in README.md of the Github repository). Using the OpenOCD patch provided in the repository, OpenOCD can be used to run SVF replays on the MachXO2 FPGAs through the driver's ioctl interface.

Even though the initial goal is met, there is still a lot to improve in terms of code design, performance and code quality. For instance, the driver is "hardcoded" in many ways to work with just RFWest and RFEast. As my mentor suggests, more MachXO2 FPGAs could also be added in the setup in future and hence it should be made generic. Also as it does not make sense for the driver to know whether the FPGA is "West" or "East", just referencing them through their device instance will be a good idea. Further, the reads and writes to the PIC can also be optimized in many ways. I will be working after GSoC on all these and finally deploy a polished driver in-tree with the AXIOM Beta Kernel.

5 Challenges faced

Working on this project was quite challenging, and yet rewarding. Many challenges I faced were based on lack of documentation or explanation of things, especially when working with device-trees and MachXO2 FPGA. Also, writing kernel code is very critical, as the system could break if anything goes wrong. Keeping that in mind, I had to think a lot before writing even a single line of code (and even that wasn't enough). I reused well tested code wherever possible. Debugging the kernel, and specifically the devicetree, was also quite challenging as there is no good upstream support. I had little time to understand and implement the OpenOCD side of things, as it was an unknown codebase for me, and I haven't really ever had used it before.

All in all, the learning curve was quite steep, as there were so many things involved, from Python scripts to Kernel code to OpenOCD code, and there also was hardware (and sometimes its weird issues). Nevertheless, my mentor was always helpful and ever available for any issues I faced. We also debugged a lot of issues together (@IRC) and I learned a lot from him in the process.

6 In Conclusion...

Initially, my perception of the project was that I would learn about Linux Kernel Development and in general about hardware stuff. I eventually ended up getting (at least some) exposure to VHDL, Lattice Diamond tools, FPGAs, MCUs, Muxes/Switches, Python, TCL, Makefiles, Bash, Kernel configuration and build process, devicetree (and overlays), protocols like JTAG & I2C, reading PCB schematics, OpenOCD, and \LaTeX , among other things. My mentor also always managed push me beyond my comfort zone, and I think that is why I got exposure to so many things. I would like to thank him and other developers who helped me in various ways throughout the program. Working with apertus^o Association this year at GSoC was really fun, and I would like to continue to be a part this organization even after GSoC ends.