GSoC 2021 Final Evaluation Report

apertus° Association

Remote Test System for AXIOM Remote

Vinayak Sankar (@vnksnkr)

Mentors : Herbert Poetzl (@Bertl)
Robin Heinemann (@vup)

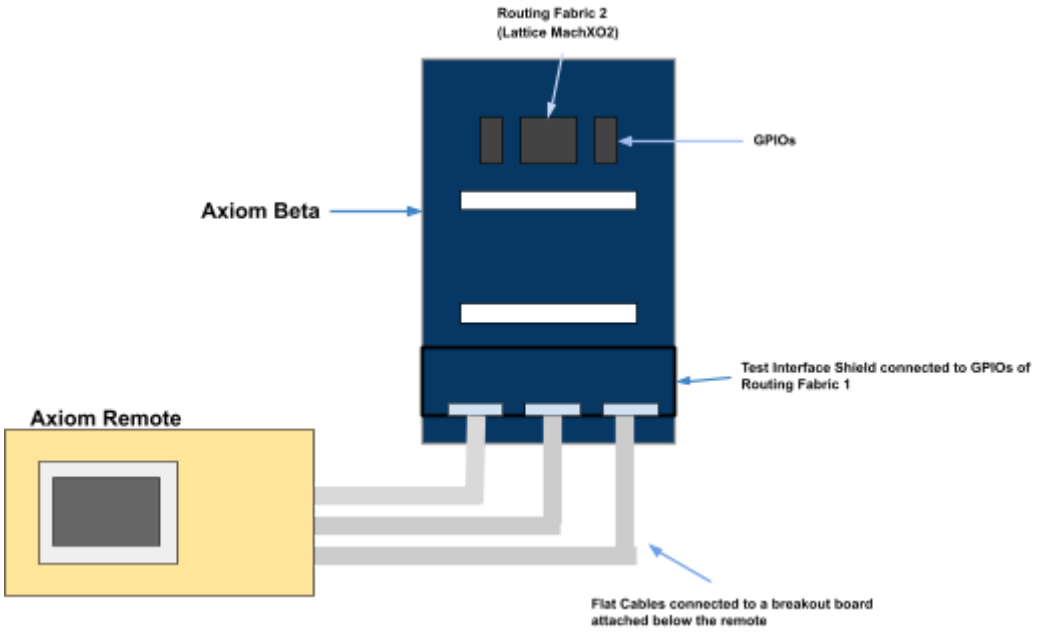
Overview:

The AXIOM Remote is a general purpose input device used for controlling the AXIOM devices. Since there has been significant development on the AXIOM Remote from its initial stage, it is necessary to have a provision to test the remote, in a remote and automated way.

The aim of this project was to develop a Remote Test System to simulate button and rotary encoder action on the AXIOM Remote via external hardware.

Hardware Setup :

- The Axiom Remote to be controlled is hooked up to a breakout board, which connects the various components on the remote to the Test Interface Shield via flat cables.
- The Test Interface Shield is populated with MOSFETs that are used to control the signals for buttons and encoders .
- This shield is plugged onto the Axiom Beta Camera, where the Gates of the MOSFETs are connected to the GPIOs available on the routing fabric FPGAs.



The main goals of this project was:

- Implement and test the HDL (gateware for the routing fabrics) to simulate button presses and encoder turns via simple commands.
- Add advanced actions for testing (button bounce, encoder skipping, etc)
- Design a test framework to run automated tests.

Work Done:

All code produced by me during GSoC 2021 can be found in this

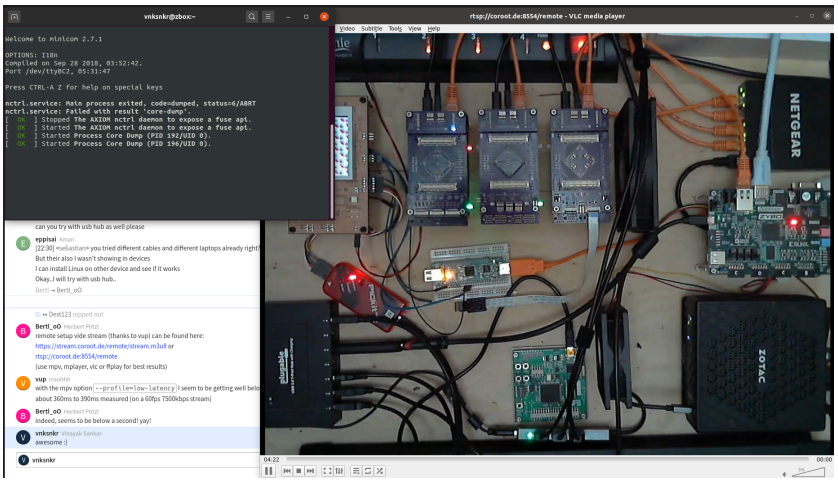
Github Repository

Tasks Implemented:

Community Bonding Period:

The community bonding period was mainly spent on planning how to develop the remote test system. This mainly included :

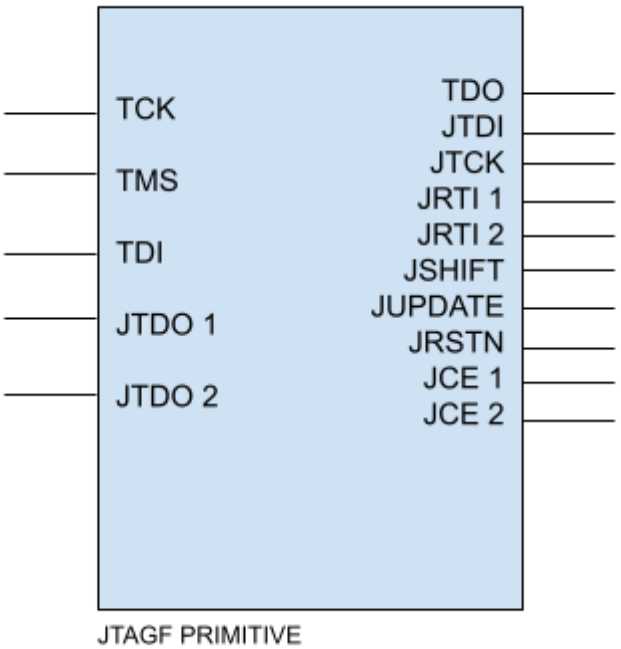
- Introduction to the remote access setup for the AXIOM Beta Camera and the Axiom Remote
- Coming up with a solution to generate bouncy signals that simulate realistic button presses and encoder turns.
- Finalizing on a communication interface to control the Remote from the Beta via the routing fabrics (Lattice MachXO2) available on the Beta.



The remote access setup with live streaming

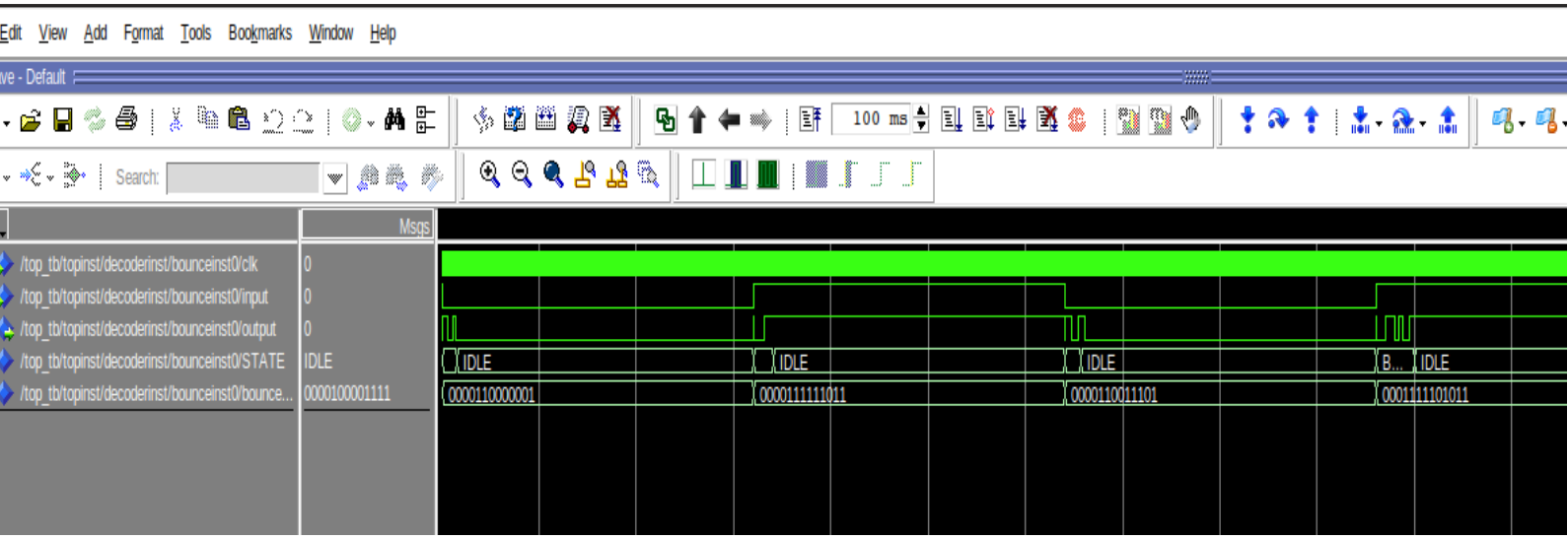
It was decided that the bounces can be generated using a Pseudo Random Number Generator (PRNG) which would give pseudo-random bounces for the buttons. The initial seed value required for this PRNG can be sent from the Beta as this would allow the user to configure the seed,which would allow reproducible test sets to be run on the Remote.

For the communication interface, JTAG communication was found to be a suitable option. The MachXO2 has a primitive available (JTAGF) which gives access to two internal JTAG registers which allows us to access the internal logic on the MachXO2.



Phase I:

Phase I began with the implementation of the HDL for the gateware. A ***PRNG*** module was developed which would output a pseudo-random number. This was fed to a ***programmable pulse generator*** module that would output a pulse having a length and delay determined by this pseudo-random number. The number of pulses generated is also pseudo-randomized. These two modules were incorporated into a single ***bounce*** module. An input signal is fed into the ***bounce*** module.Whenever this input transitions from high to low or low to high, it would initialize the bounces. This results in a bouncy signal near the transitions and a clean signal otherwise,thus simulating a button bounce or encoder skip.The input signals to this module are fed from two other modules that send clean (without bounces) ***encoder and button signals*** depending upon the configuration. Tests and Simulations were carried out during this phase for the above mentioned modules.



random bounces can be observed only near the transition

The next step was to handle the communication interface between the Arm cores on the Zynq and the MachXO2.The ***top*** entity handles the communication interface. Instructions are shifted in via the JTAG TDI pin into the internal logic through the ***JTAGF*** primitive.Thirty Eight bits of data are captured and sent to a ***decoder*** that decodes the instructions and controls the signals sent to the remote. After decoding the instruction, an acknowledge flag is sent back to the Zynq via the TDO pin so that it can send the next instruction to be executed.

The instruction comes in packets of 38 bits. They are decoded as follows:

CONFIGURATION BITS	*COMMAND BITS*
33 bits	5 bits

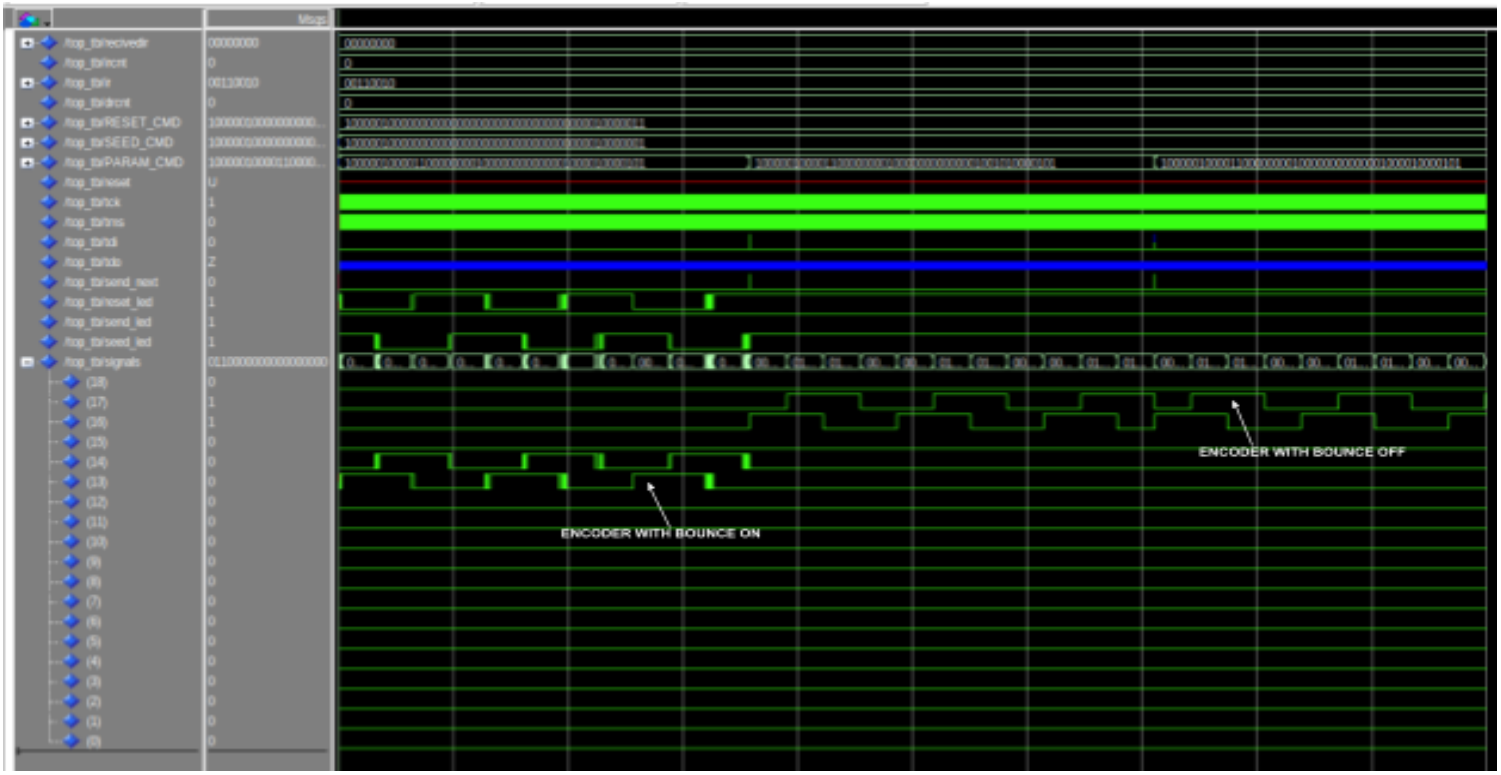
- The *COMMAND BITS* determine the type of command to be executed. This can be :
    - SEED : loads a seed value to the ***PRNG***
    - PARAM : loads signal parameters (which component to be actuated, signal duration .etc) and sends the signal
    - RESET : resets the internal FSM of the decoder
    - POLL : When an instruction contains a POLL command as the 5-bit command, the JTAG interface does not send the CONFIGURATION bits to the decoder, rather it just sends back the current state of the decoder i.e if there is an instruction in queue to be decoded or not. This can be used, for example, to check whether there is an ongoing operation in the gateware, before sending an instruction.
- The *CONFIGURATION BITS* are interpreted depending upon the *COMMAND BITS* associated with the instruction:
    - If the command is SEED, then the first 13 bits of the *CONFIGURATION BITS* are taken as the seed value to be loaded while the other bits are ignored.

- If the command is PARAM, then the *CONFIGURATION BITS* are interpreted as follows

PULSE_NO	*COUNT* VALUE	*ADDRESS*
6 bits	22 bits	5 bits

  - where *PULSE_NO* is the number of pulses in the signal to be sent. This is set to 1 when pressing a button, and the number of ticks when turning the knob of the encoder.*COUNT VALUE* is used to divide the clock frequency of the signal depending upon how long a button is pressed or how fast an encoder is turned.

After developing the ***decoder*** module and the JTAG interface, functional simulations were carried out for the rest of Phase I.



Simulation results when commands are sent to the gateware to turn encoder 1 with bounce on and encoder 2 with bounce off

Phase II:

Phase II involved testing the code on the remote hardware available and developing the python framework.

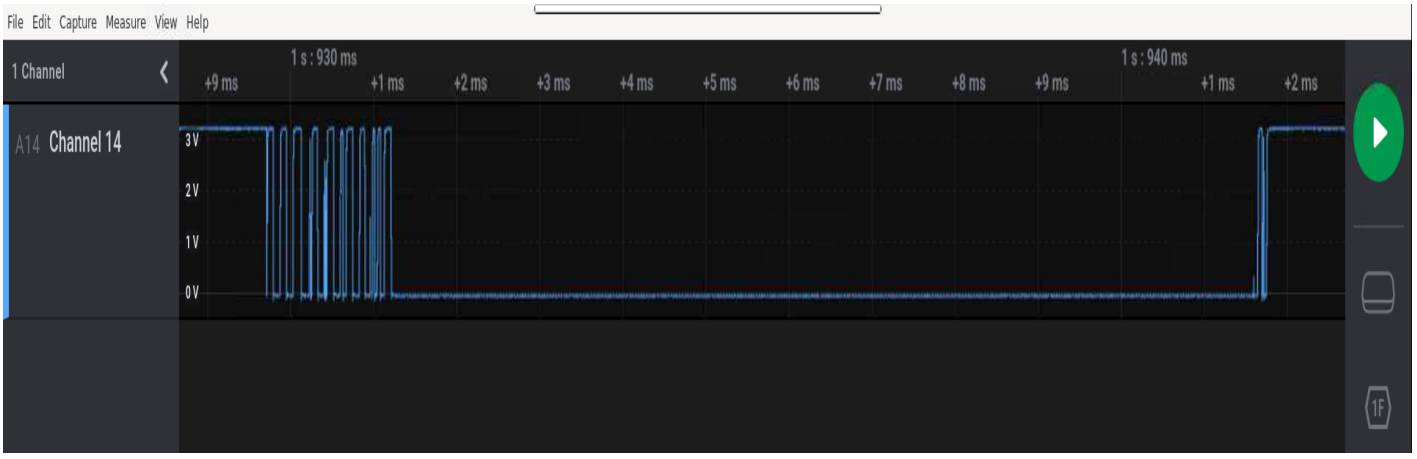
For testing the code on the remote hardware, a bitstream was generated using Lattice Diamond Tools. This bitstream was uploaded using the Axiom Beta Kernel Driver for programming the Routing Fabrics. Testing the code on the hardware, helped me in debugging my code better and optimizing it. The GPIOs of the fabric interface to a shield which is populated with MOSFETs to drive the signals to different components on the Axiom Remote. A demo firmware was uploaded to the remote which would react when a button was pressed or a dial was turned. After days of debugging the code and revising the schematics we were finally able to record presses and turns on the Axiom Remote.

A few of the components were hooked up to a logic analyzer to analyze the input signals sent to the terminals of these components .

OBSERVATIONS :

Signal observed on Push Button,P13 with bouncing on

Signal observed on Encoder 1 Switch with bouncing off



Signal observed on Encoder 1 Switch with bouncing on

Signals observed on Encoder 2 Knob with bouncing on and 2 ticks

A closer look into the same signal shown above

The final task was to develop an easy to use framework that provides the necessary functions to perform basic operations on the remote either as an automated script or in real time. Instructions on how to use the framework, the methods available and an example script have been documented in the Github Repository .

Signals observed when running an automated test script

Project Status:

- The gateware can successfully press buttons and turn dials on the encoder depending upon the instructions sent via JTAG.
- Advanced actions such as bounces have also been added and the feature can be switched on or off depending upon the user's configuration.
- The python framework currently has basic operations needed to send the right instruction for actuating the right component for the desired time interval.

Future Goals:

- Although the major objectives have been met, there might be ways in which the project could be improved in terms of design and communication.Testing the code on the Axiom Firmware would help in finding out any more bugs deep inside the code.

- Another future goal would be to make the framework more intuitive to the user and add more higher level functions to take away the lower level details
    - For example, The parameters deciding the encoder action are the number of ticks, and the frequency of the pulses. This can be made more intuitive like giving other parameter options like RPM with which the dial of the encoder is turned.
    - Maybe even integrating the current Axiom Remote Visualizer with the Remote Test System, could also be a potential project.
- Future iterations of the remote might also require modifications to the gateware and the script.
- While considering options to communicate with the Remote, one of the desired methods, which was not implemented, was to use the kernel driver itself to communicate with the routing fabrics. This could be done using OpenOCD, which is supported by the driver.
- Another enhancement could be to change the duty cycle during the bounce itself i.e the duty cycle could be randomly set to a value between 0 and 100% during a transition in the signal. This would make the bounces even more realistic.
- The communication protocol could be enhanced by providing more functionality to the data being sent back to the Beta.
    - For example, Debug functionality could be introduced that gives more information about the internal states of the gateware.

Conclusion:

Working on this project was overall a great learning opportunity. My experience with FPGAs and HDL before GSoC was limited to Verilog and Labworks in my college. Through this project I was able to learn more about VHDL, Lattice Design Tools, Clock Domain Crossing, JTAG ..etc.

Phase II had the most challenges in store for me, since setting up and testing the code on hardware never goes as planned. My mentor was always there to guide me through, especially with the remote test setup.

One of the best parts of the project was the remote test setup. The remote test setup allowed me to work on the project irrespective of the lack of hardware on my side and I hope my project would also help future contributors who don't have the required hardware to develop and test code from the comforts of their homes.

Overall I had a great experience contributing to apertus° Association and I would love to find new ways to contribute more, especially in areas like the Axiom Remote and the Camera Sensor, in the future.