Streaming Infrastructure for Frequency Multiplexed Sensors

Sarah Marmolejos
URA - FRA Undergraduate Women in STEM Intern 2023
Wake Forest University

Supervisor: Leandro Stefanazzi

ABSTRACT

The recent development of a firmware, Qick (Quantum Instrumentation Control Kit), has brought the need for the continuous streaming of data from devices utilizing it. It was initially created for the readout and control of frequency multiplexed sensors called Microwave Kinetic Inductance Detector (MKID). A number of larger scale projects have begun utilizing the firmware, and it is beneficial to provide remote data streaming. To address this, I have programmed a Field Programmable Gate Array (FPGA) board to take in data produced by the memory buffer, have it packetized, then sent over the network using User Datagram Protocol (UDP). The lack of handshaking involved in this communication protocol is what makes it ideal for transferring large amounts of data from a device, as it would drastically reduce transport time. This process required that I develop a basic server to capture the data from the buffer, so that it may be sent to a client device.

I. INTRODUCTION

Microwave Kinetic Inductance Detectors are photon detectors that are commonly used in astrophysics and other astronomical sciences due to the fact their sensors can be easily multiplexed into larger array structures. This allows them to be able to read out thousands of pixels over one microwave cable.

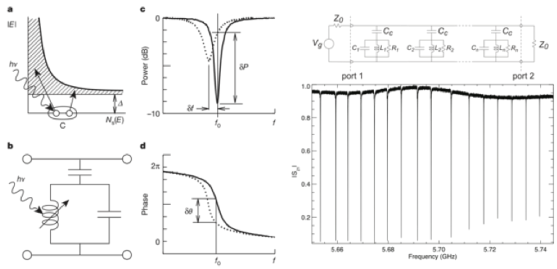

Fig. 1. (a)Activity of quasiparticles in superconducting film, (b) High Frequency planar resonant circuit. (c) resonator response in phase (d) of microwave signal [2]. .

When an MKID is hit by a photon, its superconducting properties change the kinetic inductance of the device and result in a frequency shift on the resonator. This is shown by the phase shift in (a) and (b) of Fig.1.

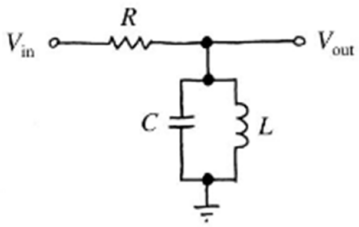

Fig. 2 Parallel resonator circuit, using an inductor (L), a capacitor (C), and a resistor (R).

In this way, MKIDs behave similarly to resonator circuits, as the frequency of some oscillations are allowed to pass, while others are limited by the resistance. The integration of hundreds, or

thousands of resonators on 2Gz of bandwidth requires a more complex system of readout and control. This need is what led the Quantum and Astrophysics department at Fermilab to develop the Quantum Instrumentation Control Kit (QICK). Although this controller is broad in the coverage of its potential usage, it's worth noting that it can be used to excite many MKIDS using a single frequency feed line.

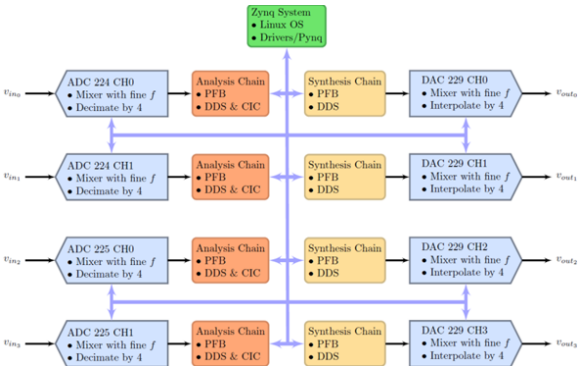

Fig. 3. Firmware implementation with 4 independent RF feed-lines [1].

Cutting-edge FPGA devices allow high logic integration of complex blocks, which allows for more than one independent RF feed line per FPGA chip. This reduces the overall complexity of a system with a large number of channels or pixels, like a telescope. It is in the interests of several of the larger scale projects at Fermilab to have the ability to stream data continuously from an FPGA, all of which would be using an Ethernet network connection.

III. METHODS

A. FPGA Programming

Field Programmable Gate Arrays (FPGAs) are integrated circuits that have a programmable fabric made up of logic blocks. However, some can make use of higher level programming by including additional interface functions and components.

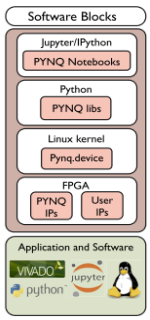

Fig. 4. This project mostly involved higher level programming of the FPGA using a Jupyter Notebooks environment,
several python libraries, and the Ubuntu linux OS [1].

Modern FPGAs typically include a processing system, which is a multicore system with DDR memory, and run a standard operating system like Linux.

For this project, the ZCU111 evaluation board was programmed with its own server that would be able to send data to a client, in this case, a PC.

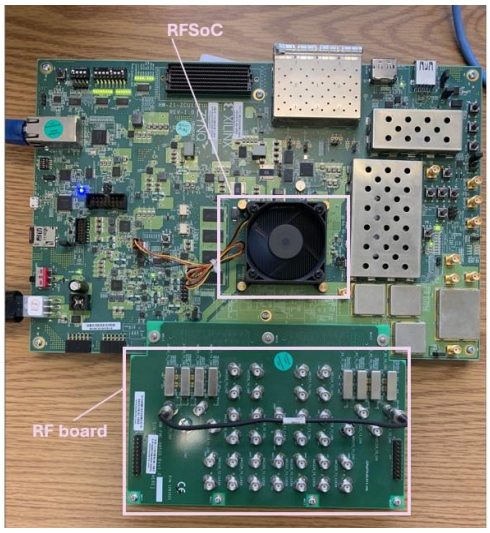

Fig. 5. The ZCU111 board was used in the later stages of this project. Highlighted are the RFSoC (software, firmware, RF, and processor components) and the RF board. The RF blocks include the ADC (analog to digital), DAC (digital to analog), and digital output.

Pyro, a python library that allows objects to interact over a server, was instrumental to this process. Unique to Pyro is the Name Server, a tool that allows the user to keep track of objects by using a logical name rather than an ID [4]

B. User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a type of communication protocol that doesn't require any handshaking when transferring data.

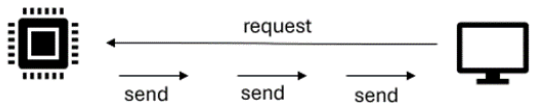

Fig. 6. After an initial request is received, a device will continuously send information until the process is complete.

This communication protocol tends to be less secure, as it does not require that the server checks to see if its message was received. However, it is more convenient to use when taking speed into account.

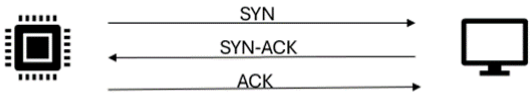

Fig. 7. TCP (Transmission Control Protocol) communication consists of three steps in its handshaking process. First, it establishes a connection (SYN). Then another device synchronizes and acknowledges the message (SYN-ACK). Finally, this device sends the acknowledgement back (ACK).

This differs from TCP, which is typically used for more secure communication such as emails or banking transactions. TCP uses a multi-step handshaking process that would take much more time to send large amounts of data.
For this project, we had used the Pyro library to establish a UDP-network broadcast to send the data generated by the firmware to the PC client.

C. Implementation

Fig. 9. Shows how information moved through the system. It is through the ethernet connection that the PC sends a request to the ZCU111. From there, the server that is running on the ZCU11 will interact with the counter, streamer, and DMA to send the requested information to be handled by the processing system, which will then be sent back to the client PC.

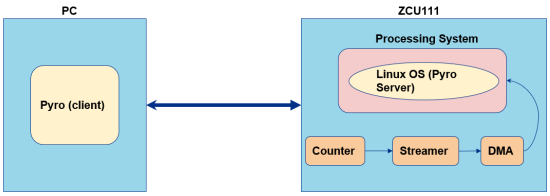

Fig. 9. Block diagram showing the connections between different components. The PC has an ethernet connection with the ZCU111 board, and acts as the client. The ZCU111 firmware utilizes a counter, streamer, and DMA (direct memory access) hardware blocks.

Fig. 10. provides a visual representation of the system validation from the counter block. The streamer is what allows the data to be sent over the network for further processing. While Fig.10. was generated by the QICK firmware on the ZCU11, it does not show the data being sent over the network. However, the plot would have remained unchanged if it had. The final step in this process was to send the data generated by the firmware to the client PC.

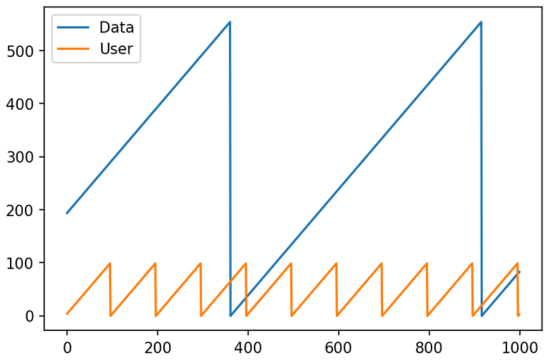

Fig. 10. This plot was obtained after programming the ZCU111 to produce data. The board is plotting points from the counter, which resets to zero after reaching a set limit. The plot shows two counters, with limits of 500 and 100.

IV. CONCLUSIONS

During this process we had been able to successfully establish a constant connection between a server running on the ZCU111 FPGA board and the client PC. The firmware had acted as the testing environment that was needed to simulate the conditions of an experiment. The next steps in the process would be to change the source of the data, rather than have it produced by the firmware.

ACKNOWLEDGEMENTS

I would like to give thanks to my supervisor Leandro Stefanazzi for all of his support during this project. Special thanks to the URA and Fermilab for allowing me to participate in research this summer.

REFERENCES

[1] L. Stefanazzi *et al.*, "The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors," *Review of Scientific Instruments*, vol. 93, no. 4, p. 044709, Apr. 2022, doi: 10.1063/5.0076249.

[2] A.O. El-Rayis *et al*., "Reconfigurable architectures for the next generation of mobile device telecommunications systems, " Nov. 2014.

[3]"http://web.physics.ucsb.edu/~bmazin/public_html/mkids.html," *web.physics.ucsb.edu*. https://web.physics.ucsb.edu/~bmazin/mkids.html

[4] "Pyro - Python Remote Objects - 4.82 — Pyro 4.82 documentation," *pyro4.readthedocs.io*. https://pyro4.readthedocs.io/en/stable/ (accessed Aug. 02, 2023).