Pulse Sequence Generation and Input Echoing using ARTIQ and the Observer Pattern

Aaron Vontell, Michael Walsh, Dirk Englund
Quantum Photonics Group, Massachusetts Institute of Technology, Cambridge MA, 02139, USA
vontell@mit.edu

Abstract: Here we present a suite of tools for real time quantum information processing using ARTIQ, for the purpose of generating pulse sequences and echoing input signals in the context of Nitrogen Vacancy Centers in Diamond.

OCIS codes: 200.3050, 270.5585

1. Introduction

The current laboratory hardware for quantum information processing experiments are most often not suitable for real-time re-configuration as a result of information received from a qubit. Here we present a suite of tools for real-time quantum information processing using ARTIQ (Advanced Real-Time Infrastructure for Quantum physics), a Python-based FPGA control architecture [1] developed by M-Labs in partnership with the Ion Storage Group at NIST as an attempt to handle many of the issues with real-time quantum information processing systems (Fig 1). These tools can be used to generate pulse sequences which adapt to new input with a lag of about 9 nanoseconds if using Pipistrello gateware, or a lag of 0.05 seconds if using Python software. In the near future, we hope to demonstrate the utility of this in state preparation of the negatively charged Nitrogen-vacancy center in diamond (NV).

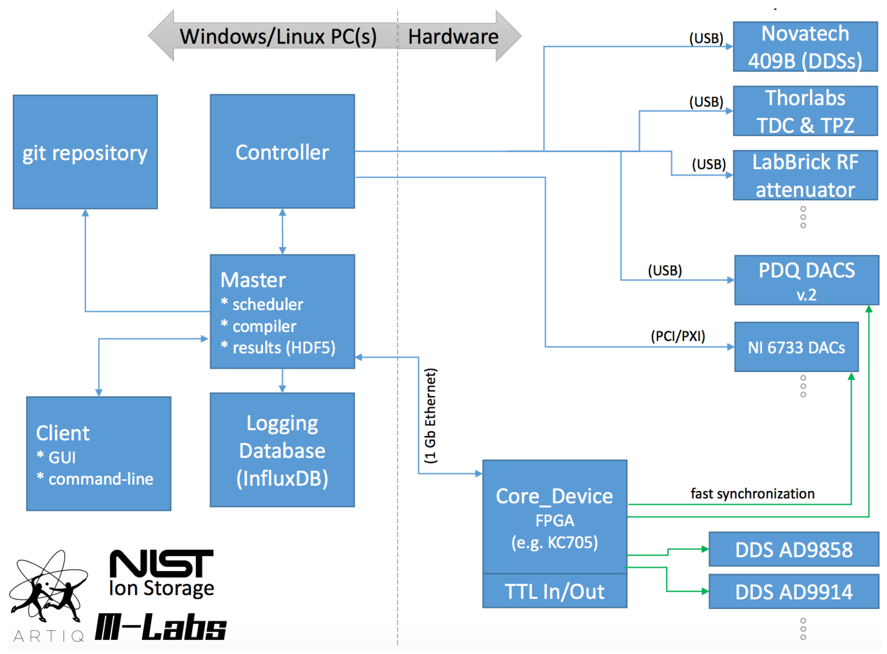

Fig. 1. Overview of the ARTIQ system

Due to the low coherence times of qubits with today's technology, there is a need for low latency instruction execution, all the while maintaining a manageable complexity when coordinating and scaling pulse sequences which drive quantum information experiments. These pulse sequences are currently created by commercial TTL pulse generators such as the PulseBlaster [2]; however, these devices do not provide real time instruction execution in response to an input signal. With ARTIQ's abstraction of FPGA hardware and gateware into a Python-based control system, and by extending it with an abstraction which features an observer pattern, physicists and developers are able to write code for FPGAs for the purpose of controlling quantum systems with this real time feedback.

The focus of this work is to validate the use of ARTIQ for pulse sequence generation and real-time response to input signals with NV centers in diamond, which have proven to be a promising medium for quantum information processing as a solid-state qubit. Additionally, we present an abstraction which uses ARTIQ to record an input pulse sequence or signal for playback at a later time, using asynchronous callbacks.

2. ARTIQ-Enabled Abstraction

Using ARTIQ as an architecture for interfacing with various FPGA boards, we created an abstraction of the Pipistrello LX45. This abstraction provides multiple methods which can be used in various quantum information processing experiments, in both setup, manipulation, and readout of data from an experiment. This abstraction provides a set of instructions which are reusable across experiments, universal across various setups, and customizable in its input and output operations.

The Pipistrello LX45 abstraction features useful methods for creating quantum information processing experiments: `reset()` permits the ability to reset the instruction execution timeline of the FPGA; `led_test()` tests the connection from the control computer to the Pipistrello board; `pulse()` takes as parameters an output `ttl`, a `period` which defines the period of a signal to output, and a `length` which indicates how long to pulse this signal for. Other methods which find properties of the board, such as instruction execution latency, were also developed and included as part of the abstraction.

3. Using the Observer Pattern

The observer pattern, a software engineering technique used in many event handling systems, can be used to execute instructions in response to an input signal using ARTIQ. Due to the functional programming capabilities of Python, we are able to pass functions as parameters, which is useful in using the observer pattern in that we can call these functions as handlers in response to a series of input events.

An example of this is the `register_rising()` method included within the abstraction, which executes a function upon reading a certain number of rising edge events on an input channel (which can also be adapted to read falling edges). This method takes as inputs a `detector` input channel (which may be a device such as an APD or PMT), a `threshold` which indicates the number of rising edges to count before reacting to the input, a `start` time at which to begin listening for rising edge events, a `length` for which to listen for rising edge events after the start time, and a `handler` function which gets called once `threshold` rising edges are detected. This function observes the number of rising edges by dequeuing a buffer on the FPGA board which stores the timestamps of the rising edges events, counting them until the desired treshold is reached. At this point, the handler function is executed, and the listener for rising edge events is terminated.

The executed method `handler` has the ability to place new FPGA events into the ARTIQ timeline (Fig 2), giving us the ability to respond to signals in real time, after a latency L defined by the board's processing speed if using gateware to handle the observer callback, or defined by the time needed for the control CPU to query the FPGA board for event timestamps if using the while loop as mentioned above. In order to find the gateware latency, the abstraction provides a function which uses a binary search to test various values of latency. In our tests, this procedure found the latency of the Pipistrello LX45 to be 8 nanoseconds. This latency-finding procedure is essential in our experimental setup; it allows us to use different FPGA setups with varying speeds without reconfiguring the underlying software. However, the latency for the dequeuing loop, which must communicate to the board over USB for the rising edge timestamps, was found to be about 0.05 seconds, and was therefore the value for L in our experiments.

4. Echoing Input Sequences

The `register_rising()` method includes functionality which allows us to collect a list of timestamps which indicates rising and falling edges in a signal. By adding these timestamps to a list, we can reference these events to create a new pulse sequence which separates on and off pulses on a channel to mimic the input.

Without gateware logic for responding to input events on a nanosecond scale, the current echo functionality supports taking this list of timestamps and recreating a pulse later in the ARTIQ timeline. However, if gateware were to be developed, we would be able to echo an input in real time. On the arrival of an input from a device, we can place a

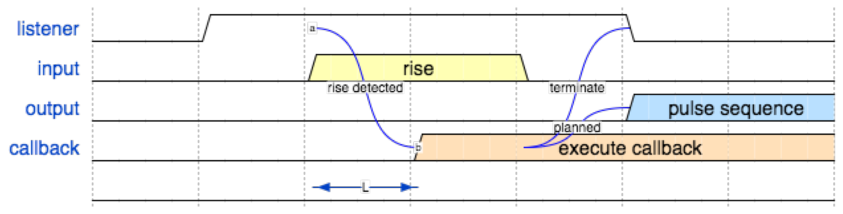

Fig. 2. Example pulse placement created from a callback after rising edge detection

rising edge into the timeline on another TTL channel. This would essentially echo the input from the input device on another channel, with a small phase due to the latency required to execute these new rising and falling edges.

5. Conclusion and Next Steps

From this work we claim that ARTIQ and the developed abstraction would be appropriate for use in the lab. Its universality and automatic dynamic setup would be useful in various settings, whether it be integrated with NV centers in diamond, ion traps, or other types of qubit systems. This can be used to create complex pulse sequences which may respond to inputs within a minimum latency, determined dynamically by a binary search algorithm. Additionally, the use of asynchronous callbacks and the observer pattern makes implementation and use in the lab easy to manage and easy to change.

The next steps for the project would include creating more methods and tools within the abstraction for organizing pulse sequences into reusable and manageable objects. A server which connects to existing FPGA setups in the laboratory would also provide a convenient control center for planning, scheduling, and organizing experiments. Additionally, gateware will need to be implemented which allows for latency dependent on the speed of the FPGA, rather than on the speed of the host computer. Finally, a variety of experiments should be completed to confirm these claims, such as bell state measurements and Rabi oscillation pulse sequence generation.

References

1. Bourdeauducq, Sbastien et al. (2016). ARTIQ 1.0. Zenodo. 10.5281/zenodo.51303.
2. ”PulseBlaster - SpinCore Technologies.” Accessed December 08, 2016. http://spincore.com/products/PulseBlaster/PulseBlaster-Programmable-Pulse-Generator.shtml.