Center for **Quantum Networks**
National Science Foundation Engineering Research Center

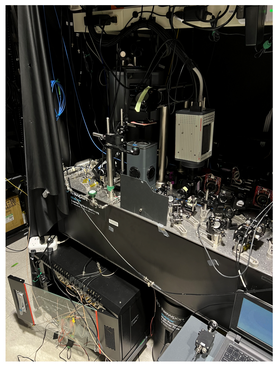
Massachusetts Institute of Technology

Digital Twin of an FPGA-Controlled SnV Color Center

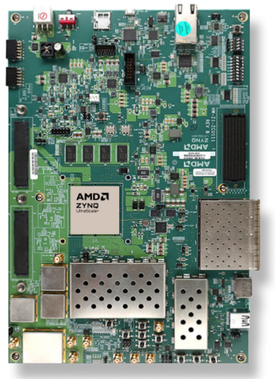
Background: A SnV Color Center Experiment

We have an experimental setup for controlling a **tin vacancy color center**. A pulse of a green (515 nm) laser at 6µw for 10µs can pump an electron out of the vacancy, leaving room for remaining electrons to bounce between the ground and excited states. A red (~620nm) laser at is used to excite one of these electrons, which will re-emit photons at a rate dependent on how close the frequency of the red laser is to the resonant frequency of the vacancy. The vacancy can have multiple optically active states with different resonant frequencies.

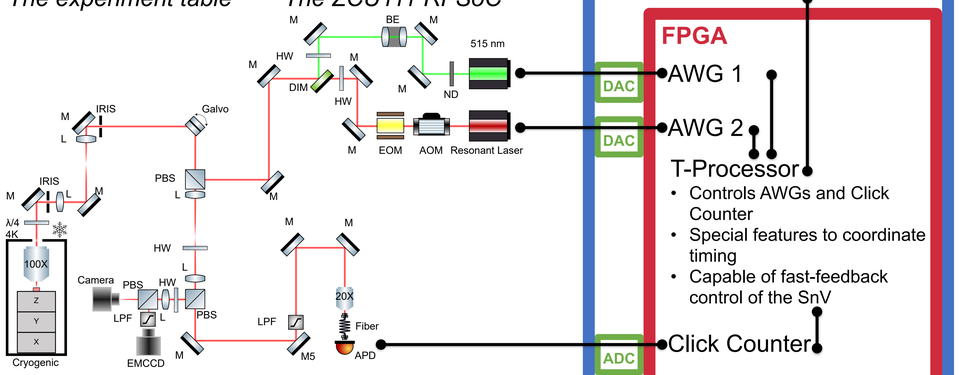
We would like to **choose a state** and **push the emitter into it as quickly as possible**. This is a first step to further manipulation such as producing entangled states with other color centers. To accomplish this, we **monitor the rate of photons detected**, **estimate the state**, **and then send a laser pulse using the AWG**.



The experiment table



The ZCU111 RFSoC



Our project uses a modified version of Qick (see https://docs.qick.dev)

We have replaced the Qick readout system with a click counting and time tagging system.

Controlled through Jupyter Notebook over the internet

Ethernet

ZCU111 RFSoC

ARM CPU
- Hosts the Jupyter Notebook
- Sends programs and commands to the T-Processor

FPGA

DAC — AWG 1

DAC — AWG 2

T-Processor
- Controls AWGs and Click Counter
- Special features to coordinate timing
- Capable of fast-feedback control of the SnV

ADC — Click Counter

Marc Davis
Presenter

Yong Hu

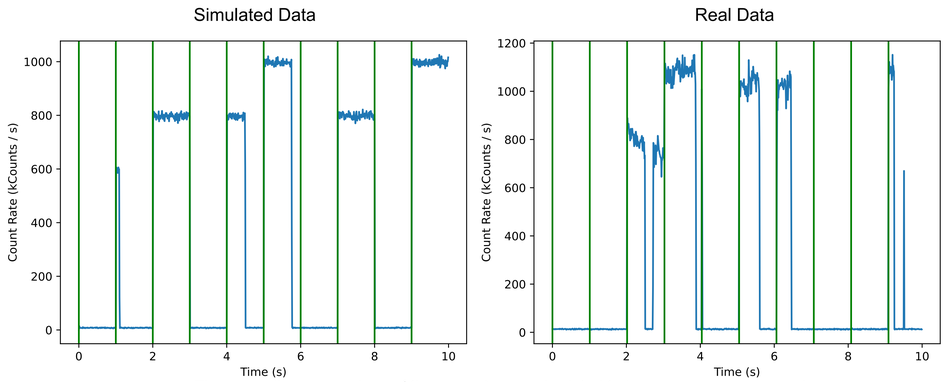
Ian Berkman

Prof. Dirk Englund

Simulating the Experiment

For testing and training strategies for the code that will be run on the FPGA, we need to **simulate the whole system**, with a focus on the behavior of the FPGA and how it interacts with the other devices.

Typical FPGA simulation involves simulating every logic gate during every clock cycle. This is **extremely slow**, and can often result in minutes of compute time for nanoseconds of simulation time. This is far too slow for our needs, so instead we developed our own simulation system with two major improvements over FPGA simulation: **high-level emulation** of FPGA-implemented components, and **event-driven simulation**. Together these result in a simulator that often runs faster than real-time while still offering nanosecond-level detail.



The green lines are pulses of the green laser, scheduled to happen once per second.
The blue lines show the rate of photons detected, averaged over 1ms time bins.

Emulating FPGA Components

Rather than attempt to simulate every detail of FPGA, we instead focus on the **high-level behavior**. We developed a Python library that implements the instruction set of the T-Processor, as well as capturing the behavior of the peripheral components. AWGs are modeled as recording the parameters of the generated waveform and the timing of when the pulse is generated. The click counting component is its own specialized coprocessor with multiple modes, including both counting and time tagging, so its behavior was also described in Python.

We chose to implement our emulator in Python because typically the Qick Python library is used to control the T-Processor. We have designed our simulation library to have an identical interface to the Qick library, such that **the same Python script can be run on either the simulator or the actual device**.

Event -Driven Simulation

The base unit of time in our simulation is the FPGA clock cycle (it is a 384MHz clock, so the clock cycle is about 2.6ns). However, we do not need to simulate every clock cycle. A typical program for the T-Processor involves a 1µs measurement cycle. The T-Processor performs data processing and makes decisions at the beginning of the cycle, and then waits until the next cycle. The SnV in its brightest state emits about 1 million photons per second, which is about 1 photon per µs. Overall, **only about 10-15% of the FPGA clock cycles involve any activity that we need to emulate**.

We have made our simulation **event-driven**, which means that instead of simulating each clock cycle, we skip to the next "event" which is either T-Processor activity or a photon captured. This allows us to skip 85-90% of the clock cycles with no loss in simulation accuracy. We are able to maintain **nanosecond-level detail while often simulating at faster than real-time**.

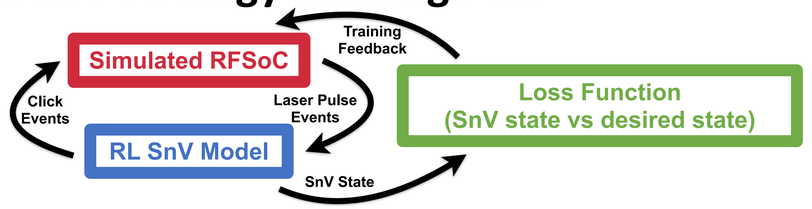
Next Steps:

Learning Optimal Control Strategies

Our simulation setup provides ways to debug T-Processor programs that are not possible on device, and this has already let us r**apidly prototype control strategies**. However, we are working on taking this a step further.

We are working on a **neural network accelerator FPGA component**, which will run a model that we will train in simulation. However, to train this model, we need a high-quality simulation of the SnV. **We can achieve this by using data from the actual experiment to train a model of the SnV!**

RFSoC Strategy Training Plan



SnV Model Training Plan