# FPGA-based QPU interface unit for fast qubit control and readout

## General notes on development and testing procedures

Christian Križan[1,3] and Mats Tholén[2,3]

[1] Quantum Technology Laboratory, MC2, Chalmers University of Technology
krizan@chalmers.se

[2] Intermodulation Products AB

[3] WACQT, Wallenberg Centre for Quantum Technology

**Abstract.** This document outlines general hints on development and testing procedures, as well as a prioritisation list, for usage during the 2020 DAT096 project 'FPGA-based QPU interface unit for fast qubit control and readout.'

## 1 Hints on development and the verification thereof

This section outlines suggested development methods for this project.

The key take-home message is to use loopback development to the greatest extent possible, as your access to the target hardware is limited during the verification process. The loopback methodology is in turn supported additionally through the upstream/downstream nature of the device design.

Your available development hardware (mainly, the Nexys 4 DDR) was constantly considered during the specification of this project. Implying that every block should be developable and verifiable with hardly any access to the target hardware platform.

### 1.1 Suggestions on development methodology

At more or less every point in your signal path, you may find that attaching some output of the upconversion datapath to some equivalent input of the downconversion path allows for verification of any subsection in this project. Meaning, you may develop the entire datapath step-by-step, and verify your incremental progress by linking output to input.

Developing the central stream controller might be more challenging, as it will require generation of data stream packets at seemingly random FIFOs. One idea could be to clock the central stream controller at a very low speed, and use the

toggle switches on the board to fake incoming/outgoing stream data. However, this will not alone provide you with the very real memory access constraint of your design. Memory read/write times will be a crucial constraint in this module as memory access is considerably slower than your internal communication.

You should think about designing the central stream controller's management on paper before development, or possibly even use tricks learnt in real time scheduling to set internal FIFO read/write deadlines and the management thereof. Perhaps a very basic EDF scheduling plan is in order to manage the flow of data?

Remember that all FIFOs should indicate when they have one memory slice ready for readout, or when they have space for fitting an outbound memory slice.

### 1.2 Expected outputs and testing of DSP portions

A key question during testing should be: 'How do I know whether I'm seeing the correct output?'

In the DSP portions of your board, ie. the up- and downconversion stages, you will undoubtedly come across a substantial amount of human-unreadable data. HDL simulators such as Modelsim or Vivado Simulator will in turn at their basic configuration merely show sinusoidals of the waveforms described by the data.

A suggested approach would be to export the simulated data generated in the DSP portions to some text file format, and import said data into Matlab for FFT analysis. Or, you may do an approach more similar to that done in industry; design the DSP datapath in Simulink and export ready-made VHDL for implementation on the Nexys 4. From personal experience, this approach is very resource intense relative to what you currently have available. But, we've been assured that such an approach can be made very resource effective indeed. Yet, do note that there are several groups who have failed at such an approach in DAT096. While there are groups who have made complex DSP datapaths using Modelsim and Matlab FFT analysis. Also, considering Mats' commercial affiliations, such an export would likely yield unusable VHDL for his applications.

Testing and verification on the non-DSP portions of your design, ie. the Ethernet and datastream management portions, will be much more similar to what you have already done in earlier course work. Meaning, simulating and verifying that certain data words are available at certain ports at the correct strike of the system clock etc. Remember that do-files and testbenches are not just for passing DAT093, but are in fact useful.

### 1.3 Test sequences and how development is done at QTL

At QTL, we start by setting up the experiment in a multi qubit pulse generator. This generator is a software plugin in a large instrument framework based on Python.

In this project, a plugin will very likely be made for this framework for generating the test sequences you might want to use - your test signals would thus contain actual quantum experiment data. At the least, these sequences will consist of data stream packages as well control words. Depending on the time available by the product owner, a full Ethernet frame may be put around said data. Although, this might prove excessive depending on how you figure out how to send data onto the Ethernet interface from the host PC onto the FPGA.

## 2 Development prioritisation

As repeatedly noted, completion of any of the four major specified modules yields a result usable by somebody else once this project has finished. The follow-up questions would be 'What is then *more* usable?' as compared to what is less usable in the end. It can be foreseen that some modules may receive less attention than others when racing to achieve a system good enough to perform final functional verification, as getting to run measurements on a real quantum processor is pretty cool.

Hence, this section will outline a bare minimum system. Without the modules specified herein, you will simply not be capable of running the device on actual quantum hardware. [4]

### 2.1 Prioritisation list

In this subsection, the major modules have been 'ranked' from most prioritised to least prioritised.

[4] You'd be unable to communicate with the qubits in the QPU.

1. Ethernet communications module
   Without the Ethernet communications module, you have no ability to interface with the device other than going beyond the scope of this project.

2. Central stream controller
   Not having any stream management would imply sending arbitrary data at random times to the QPU, which will result in quantum gibberish. Thus it is not suggested that you prioritise a bulletproof up-/downconversion lane system until you have at least a very basic stream management up and running.

3. Up- and Downconversion lanes (equal prioritisation)
   Finally, once you have the ability to receive streamed data in a non-panicked fashion, you should strive to make the up- and downconversion lanes operable. It is suggested that you begin working on the upconversion lane, as the downconversion lane is simpler and will re-use many of the assets that you have developed for the upconversion lane.

### 2.2 A minimal system

Loosely, the bare minimum system consists of something which may set and read data samples at some defined frequency band. You must thus be able to send data to your device, put that sample at some frequency, and read back the results onto the host PC. The bare minimum system must thus feature the following:

1. Practically every module specified for the Ethernet communications module, except for a control word host, and the ability to process ARP requests. Meaning that you have to link MAC and static IP at the OS level, and that modules will have to be configured at synthesis, using buttons on the board or by similar methods. AXI bus compatibility can also be discarded for some custom bus format.

2. The central stream controller can be made very dumb and naïve, in practice without conflict handling ie. some packages may even be dropped because a FIFO is full or similar. The memory management in turn does not have to be DDR-based, it can be made using BRAM cells or even LUTRAM. In fact, there might be no need for advanced memory management depending on the implementation, and the quality (or a lack thereof) of your stream manager. Although, the key principle must still be held that packages can arrive at any FIFO at any time. Yet again, AXI buses can also be neglected in favour of some custom format.

3. The upconversion stage can be made very minimal with a FIFO linked to the DAC directly. The upconversion must then happen by hard-setting a register on the physical DAC, which enables a built-in mixer. This mixer is very inflexible during runtime, which is why your mixer is part of the specification in the first place. Alas, if the operator has cheated in setting up the measurement and already swept the QPU using a standalone VNA, then the operator already knows at what frequencies the qubits can be heard from. The operator thus knows at what frequency the built-in DAC/ADC mixers can be set to. The minimal upconversion stage would not feature RLE decoding, summation etc. And since there is no IQ mixer block, no NCO.

4. The downconversion stage can be made equally simple as the upconversion stage, while also skipping the skip/store block. And in turn, setting the internal ADC mixer to one specific downconversion frequency. Implying that the operator has to perform a lot of post-processing on a large dataset.[5].

At the very least, this minimal system can be used to sweep a readout resonator spectroscopy. That would prove for instance whether the chip readout resonators were successfully manufactured. By sending higher waveform amplitudes, it should also be possible to check whether the qubits exert a dispersive shift on these resonators. This shift in turn shows that "the qubits are alive."

If you can show that you can generate approppriate data "at the DAC input" port, and receive appropriate data at the corresponding eight ADC FIFOs, we may instigate synthesis of your design on the target hardware platform so that you can run your design on an actual quantum processor unit.

[5] For comparison, in a real comparable case at QTL, one Ph.D. student is by himself generating 250 gigabytes per week, which is why QTL is the largest generator of data on the Chalmers intranet according to the MC2 IT-department.