ARTIQ induction - Realtime Input and Output (RTIO)

Charles Baynham

2024-02-05

---

1 Realtime Input and Output (RTIO)

You have now become familiar with the ARTIQ Experiment structure and the way that ARTIQ interacts with devices. One special device is the ARTIQ core - this is the crate of hardware which motivated the whole ARTIQ infrastructure. In this module, we will learn how to communicate with the ARTIQ core device and become familiar with some of the devices available to us.

1.1 The core

1. Execute your first kernel on the core device! Try the features of the artiq compiler:
    a. Add some numbers together and `print` the output
    b. Repeat your plotting dataset experiment from the previous module, but this time build the dataset on the core. You will need to use an *array* on the core device.
2. Turn a TTL on and off and watch it on the scope

N.B. In the lab, there is an RTB scope connected to the ARTIQ core above the sidearm. The `_device_db.py` file contains aliases for the ARTIQ devices which are connected to it, as well as a list of the connections that are present. You should use devices by their alias instead of their raw address, i.e. refer to "ttl_output_to_scope" instead of "ttl12". This is connected to ch3 on the scope. You can access the scope at its IP address on the lab LAN (192.168.1.14 at time of writing). One caveat - only one person can be using the scope at once. So take turns!

3. Output a pulse train with user-configurable number of pulses, width of pulses and time between pulses.
4. Use the signal generator to create a square wave from 0 to 5V (be careful not to go negative!). Use "ttl_input_from_scope" to count these pulses coming from the signal generator.
5. Repeat exercise 4, but now use your code from exercise 3 to generate the pulses instead of the scope, whilst simultaneously reading them back in on another TTL ("ttl_input_from_other_ttl"). Try to use ARTIQ `parallel`/`sequential` blocks to keep your code tidy. Also consider breaking your code into functions instead of storing it all in `run()`.

6. Write a simultaneous pulse on all ttl lines from ttl8 to ttl15. Now add one more, ttl7, to make 9 simultaneous pulses. This will fail. Explain why (you may find this file useful)

1.2 Input

7. Configure the scope to output an interesting function (e.g. a sine). Read this in on the sampler (channel 0) and plot it

8. a. Use your square-wave code to generate a square wave on "ttl_output_to_sampler".
   b. Use sampler channel 1 to take a high-sample-rate dataset of the rising or falling edges of your pulse train.
   c. From this, calculate the rise-time that you observe on the Sampler data and use this to estimate the bandwidth of the Sampler's analog input stages.
   d. See how fast you can take data - push the Sampler to its maximum. The ADC claims to achieve 1MS/s - can you realise this performance?

1.3 RPCs

So far you have run code in two places. In the previous module you ran code on the host pc. In this one, you have run code from ARTIQ kernels. Often, however, it's useful to run code in both these places. You have already encountered code that calls kernel methods from the host pc: your `@kernel` decorated `run()` methods are such examples. Here, we will encounter code that works the other way around, i.e. code that runs on kernels which calls code on the host pc. This is useful e.g. for communicating data to the host pc (you have already used an RPC when you called `.set_dataset()` in exercise 1 of this module) or for controlling devices other than the core, connected via USB or ethernet to the host pc.

9. Use the `@rpc` decorator to create a method that runs on the core and e.g. uses numpy to calculate an equation and prints the answer. Call it from a kernel.

10. Make an rpc which uses `time.sleep()` to wait for some time, configured by a parameter which is passed from the kernel. See how the kernel code waits for the RPC to complete.

11. Convert the previous RPC to "asyncronous" mode - see that the kernel no longer waits for the response before it continues.

12. Run the Experiment in `test_rpcs.py`.

    a. Uncomment the print statements one at a time and rerun the experiment, but try to guess in advance what they will print.
    b. Explain why the value of `my_variable` differs on the host and the core at different times in the experiment.
    c. Ensure you understand this Experiment's results fully before you move on