Rev1 Comm Firmware Development Report (3, Revised)

Gen2 Hardware Call

Bunheng Ty, Kael Hanson

ty@wisc.edu

WIPAC

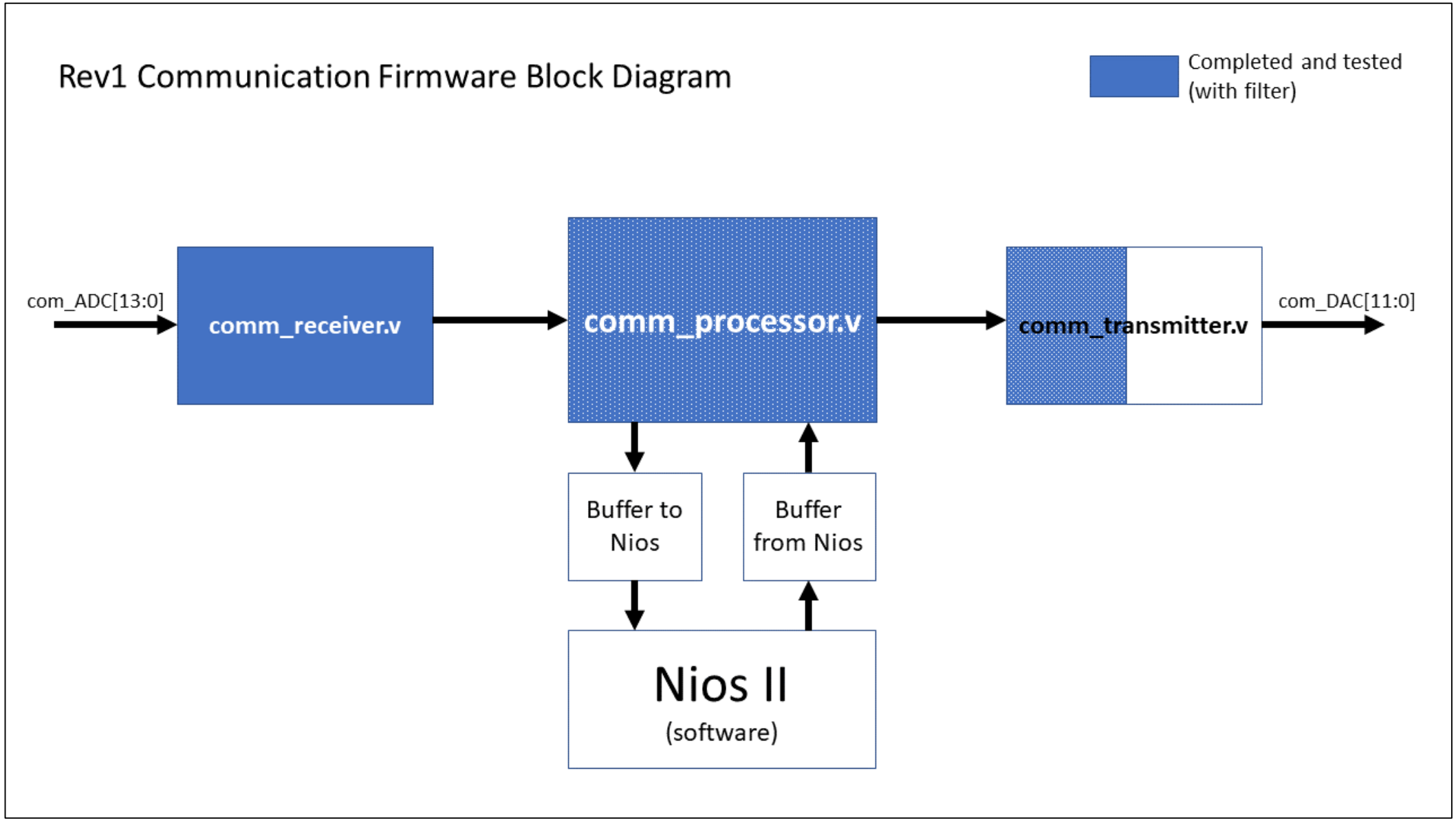
02-08-2018

Overview

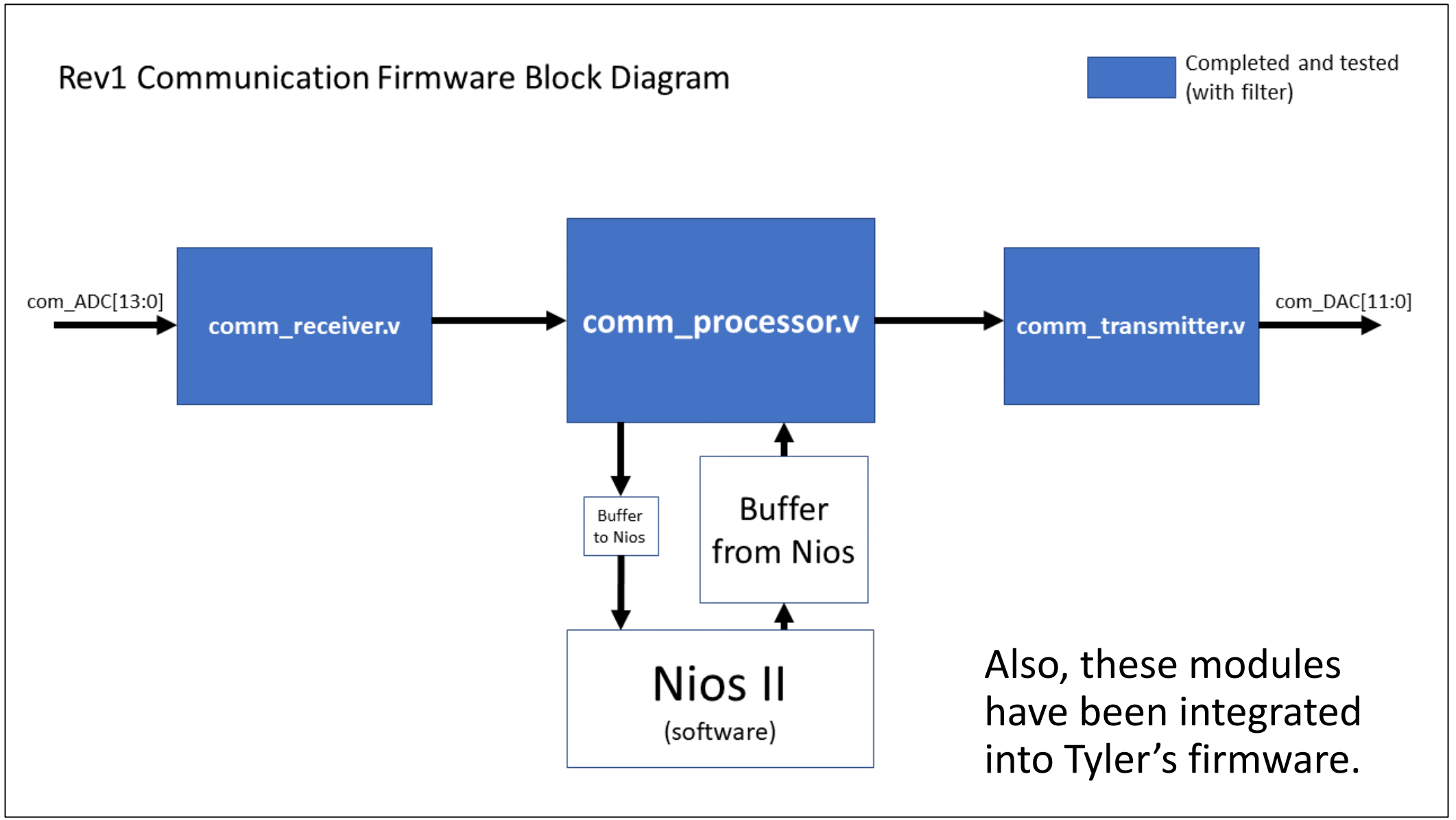
- We now have a working communication firmware – can demonstrate data transfer between Rev1 and DOMHub.

- This presentation:
  - I give the current status of the communication firmware.
  - A summary of the DOMHub-DOM communication scheme and the inner working of the current communication firmware for the Rev1 board.
  - A short test of possible interference with the digitization circuit.

Previously… (before the holiday break)

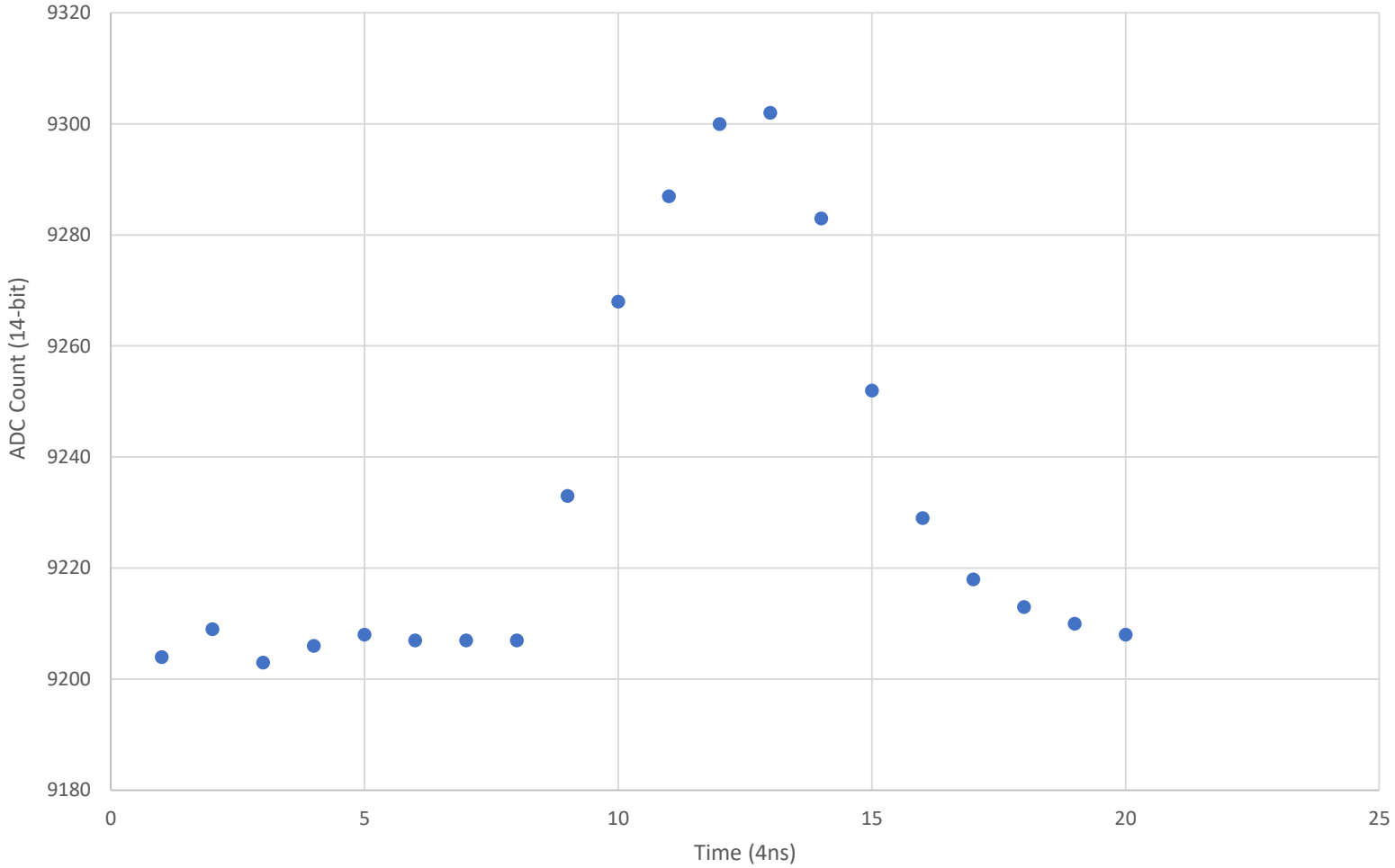


Now


Rev1 Communication Firmware Block Diagram
Completed and tested (with filter)
com_ADC[13:0]
comm_receiver.v
comm_processor.v
comm_transmitter.v
com_DAC[11:0]
Buffer to Nios
Buffer from Nios
Nios II
(software)
Also, these modules have been integrated into Tyler's firmware.

Now

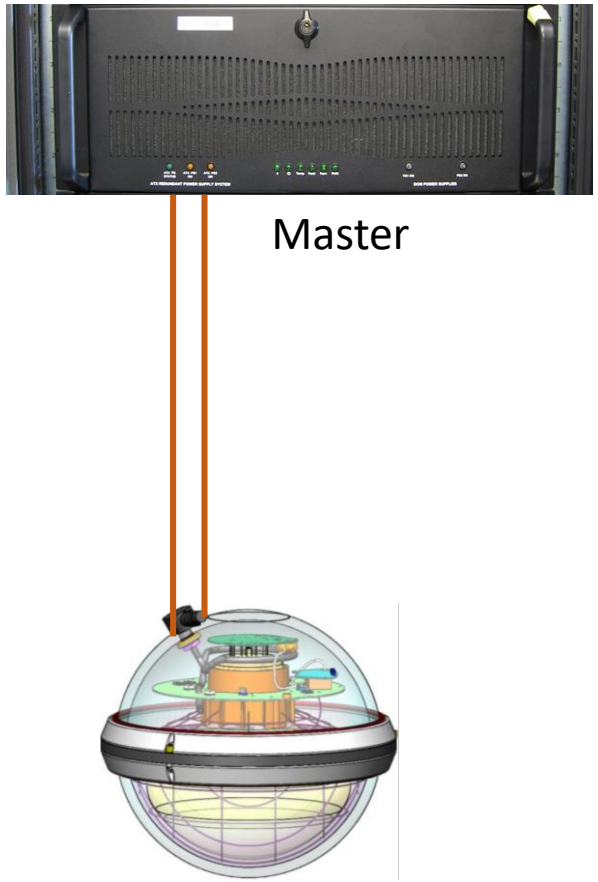
First* Waveform Sent from Rev1 to DOMHub



- Waveform generator (Keysight 33600A), -20 mV pulse, 20ns wide
- Note there are only 20 data samples. For some unknown problem, I can't send anything longer than ~500 bytes. (The max limit of the communication scheme is 4096 bytes.)

*Not true

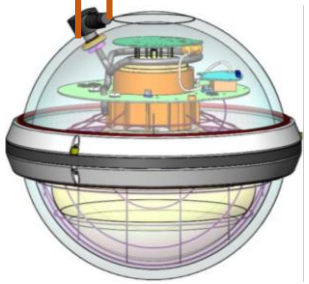
DOMHub-DOM Communication Scheme



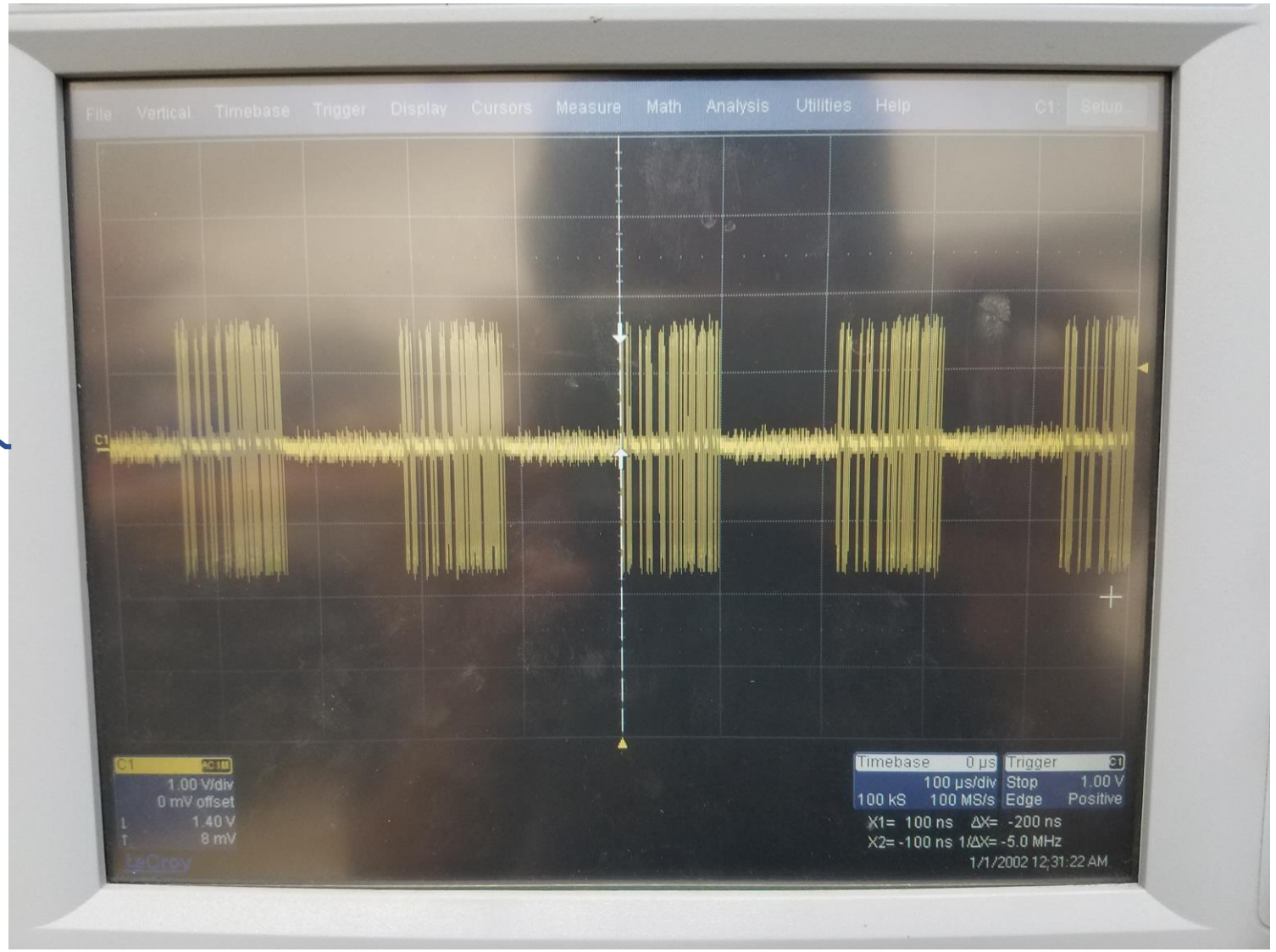
- DOMHub sends a packet of bits to DOM. DOM responses ten µs later. This repeats every ~220 µs (most of the time), and never stops.
- Most of the time, these packets are “empty”, meaning that they are exactly 100 bits and thus 100 µs long.
- Occasionally, a packet is longer than 100 bits (can be as long as 41060 bits). These packets contain “data” – information originated from the software.

DOMHub-DOM Communication Scheme

Master

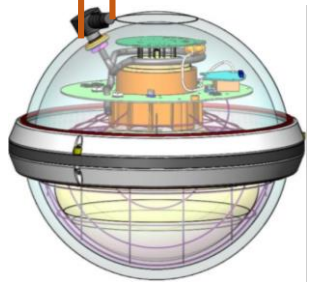


Slave



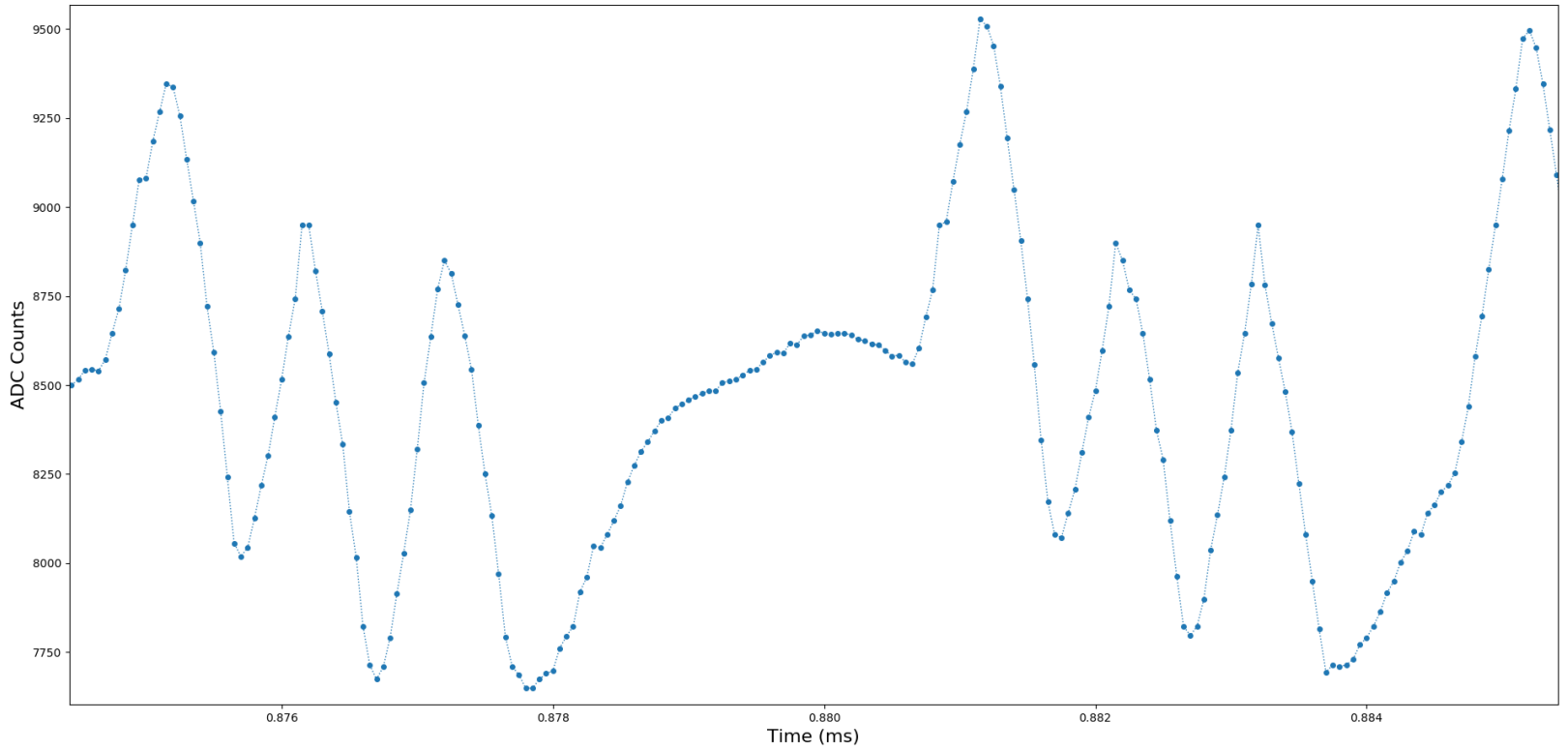
DOMHub-DOM Communication Scheme

Master



Slave

First 10 bits of every packet: 1110001110



DOMHub-DOM Communication Scheme

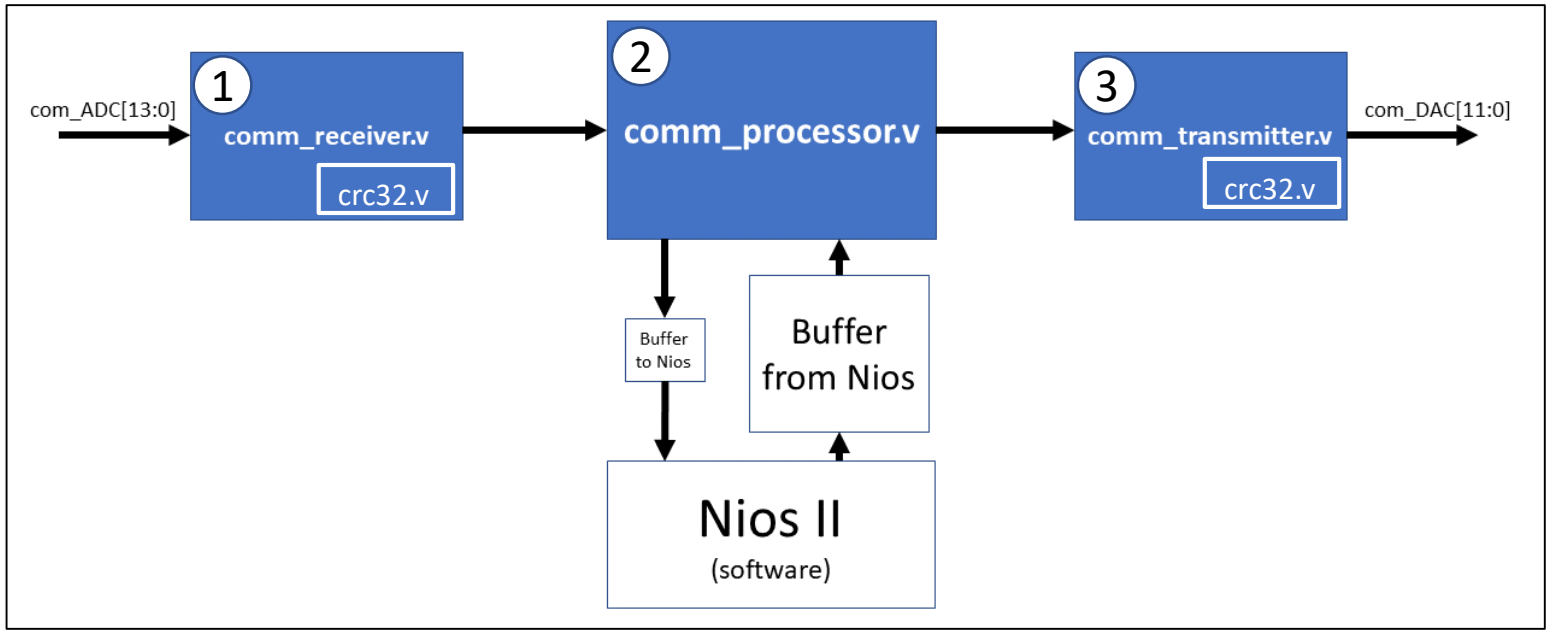
A visual representation:

- An “empty packet”. (Empty with regard to software, but not to the firmware.)

- A packet containing 32 bits of “data”

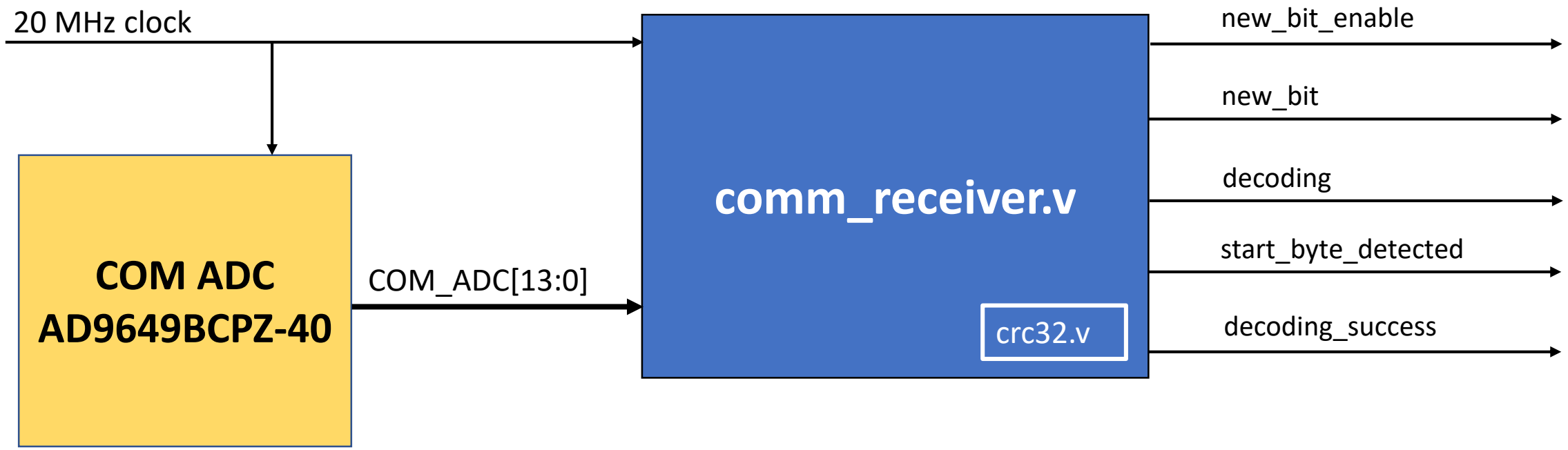
Rev1 Communication Firmware Overview

- So to be able to communicate with the DOMHub, need to
  1. Recognize the start of a packet and convert the waveform into 1's and 0's.
  2. Determine the packet's type and intended destination. Deposit data bits into a buffer for the onboard CPU (Nios II) to read. Also prepare the proper response packet, include bits from Nios II, if there are any.
  3. Control the comm_dac to convert the response packet into the right waveform on the communication line.



Also need to do crc32 check. A fourth Verilog module, crc32_calculator.v was written and is used in comm_receiver.v and comm_transmitter.v, to verify and generate crc32 checksum.

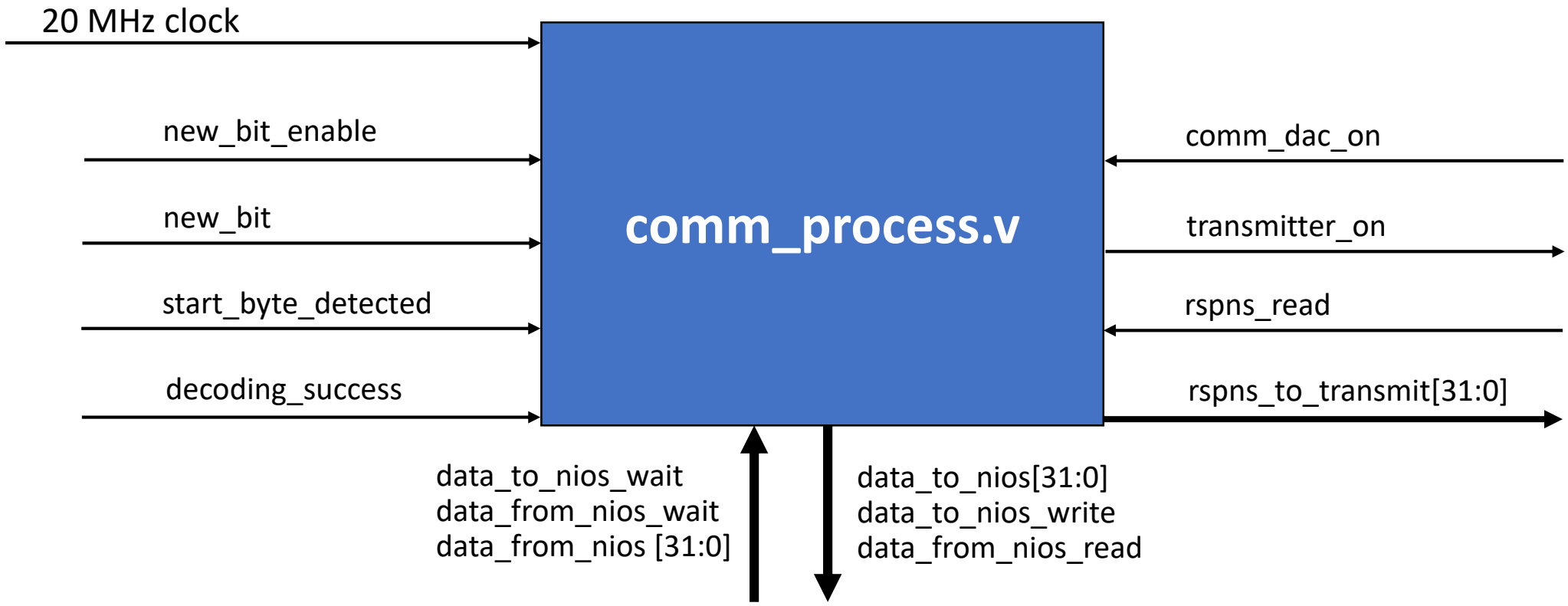
comm_receiver.v



Note:

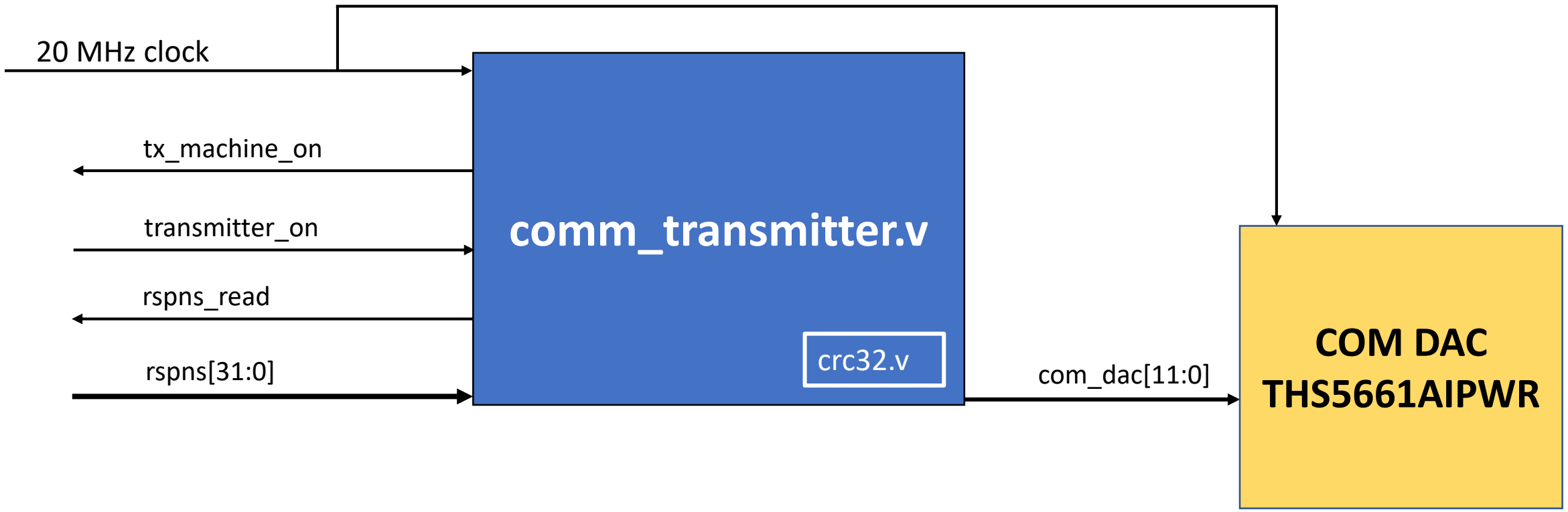
- start_byte_detected goes up if the first ten bits of the packet matches the correct values.
- decoding_success goes up if the last ten bits decoded matches the end-of-packet pattern while at the same time, the crc32 checksum is verified.

comm_process.v



- Wait until start_byte_detected goes up before starting processing: looking at the new_bit and new_bit_enable lines.
- Wait until decoding_success goes up before transmitting the response.

comm_transmitter.v



- A “zero” is 011111111111 for 20 clock ticks.
- A “one” is 111111111111 for 10 clock ticks followed by 000000000000 for 10 clock ticks, then return to 011111111111 at the end.

Sending Data to DOMHub

- How to get data into the data buffer of the DOMHub? Here's how:

#

1.	DOMHub	→	"com chan reset"	→	Rev1
2.	DOMHub	←	"idle"	←	Rev1
		(Rev1 now appears as "communicating" on the DOMHub)			
3.	DOMHub	→	"data read request"	→	Rev1
4.	DOMHub	←	"data req ack, no data"	←	Rev1
.					
.		[open the dev file on DOMHub terminal]			
n.	DOMHub	→	"initiate connection"	→	Rev1
n+1.	DOMHub	←	"acknowledged"	←	Rev1
.					
.					

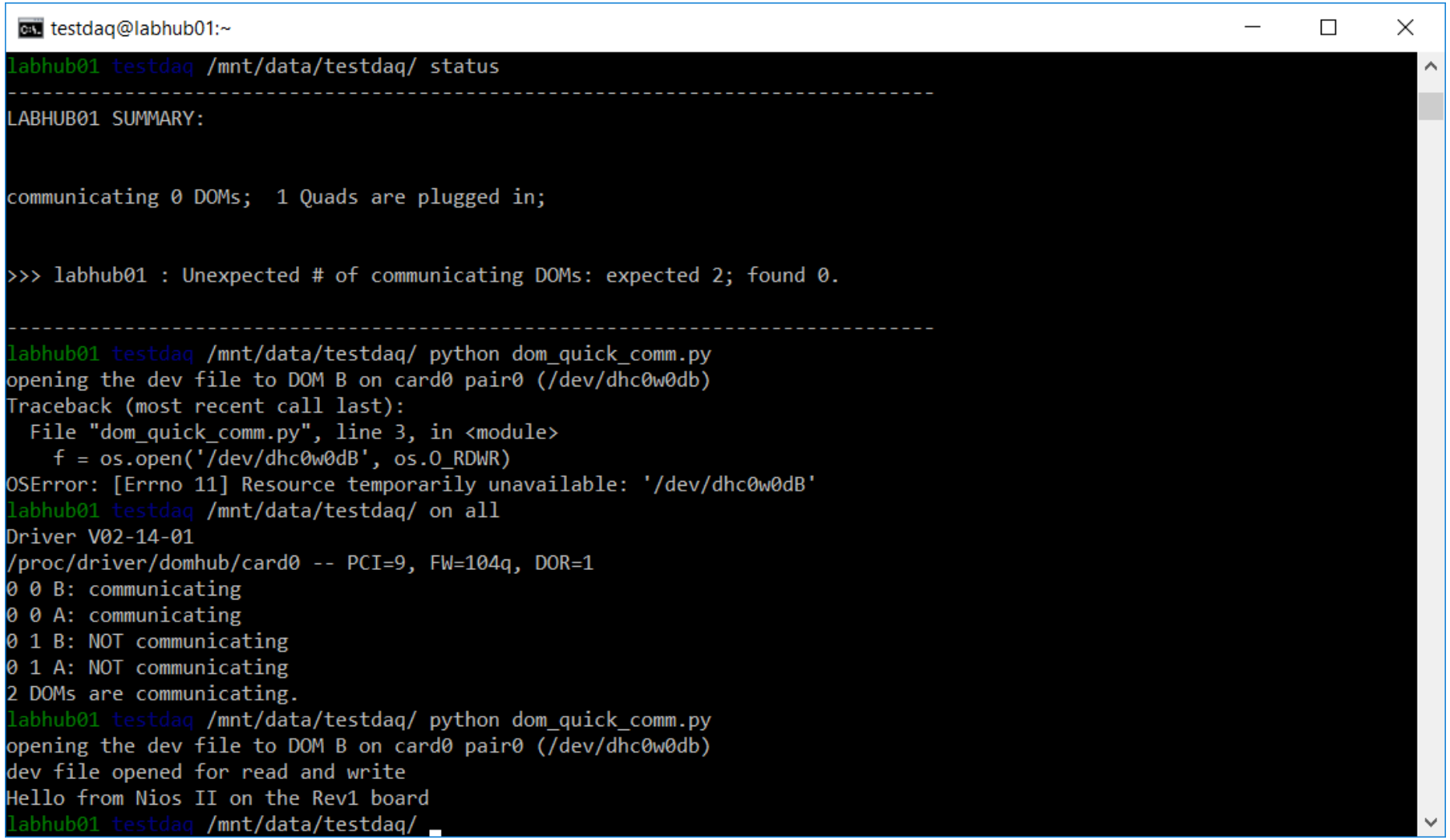
Sending Data to DOMHub

- How to get data into the data buffer of the DOMHub? Here's how:

#

n+m.	DOMHub	→	"data read request"	→	Rev1
n+m+1.	DOMHub	←	"initiate connection"	←	Rev1
.					
.					
n+m+l.	DOMHub	→	"connection initiated"	→	Rev1
n+m+l+1.	DOMHub	←	"acknowledged"	←	Rev1
			(this repeats for 9 more times)		
n+m+l+k.	DOMHub	→	"data read request"	→	Rev1
n+m+l+k+1.	DOMHub	←	"connection initiated"	←	Rev1
			(dev file should now be opened for read and write)		

Sending Data to DOMHub

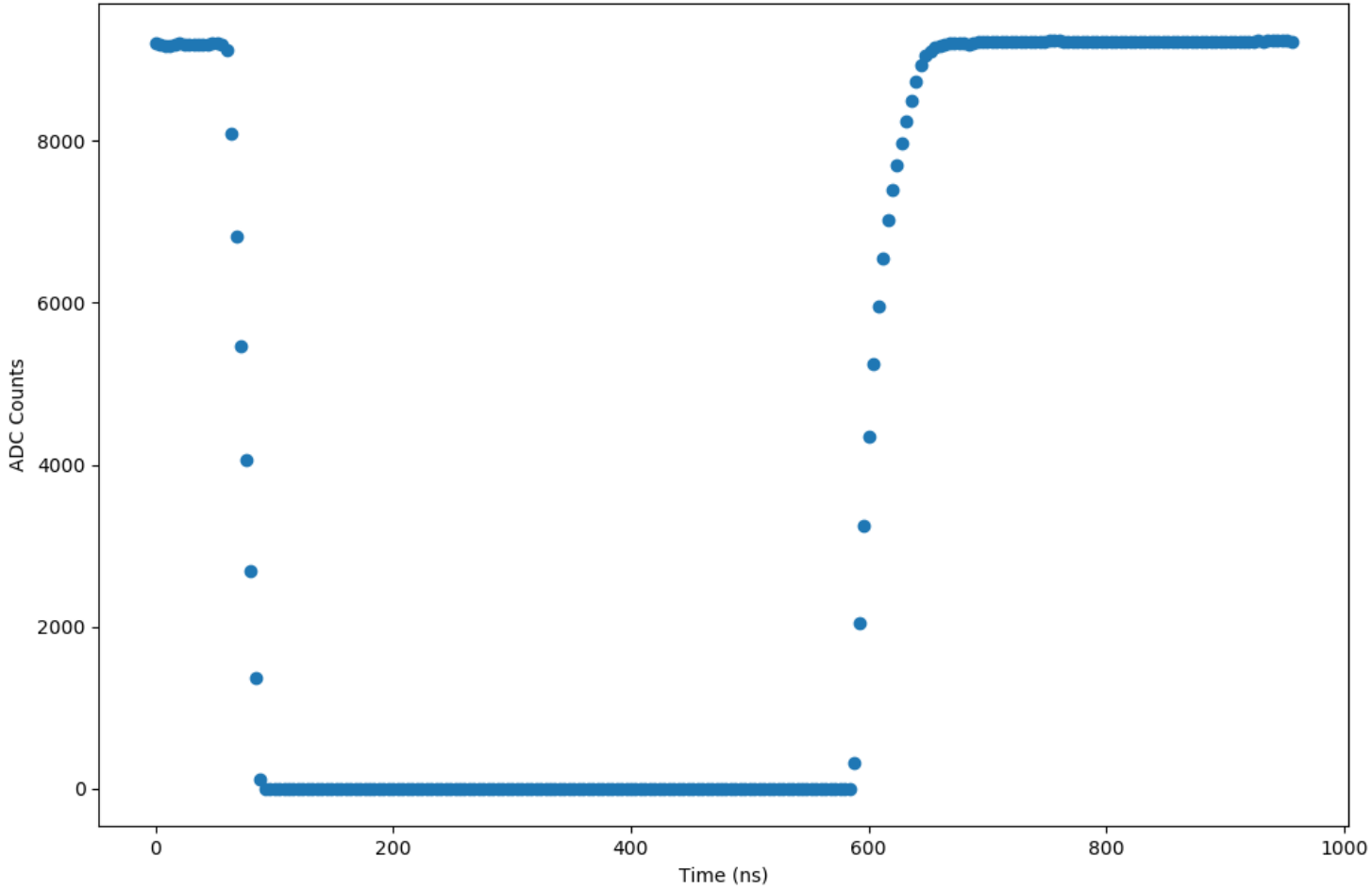


This line came from the Rev1 board

An example of Rev1 sending data to the DOMHub.

A Simple Digitizer Interference Test

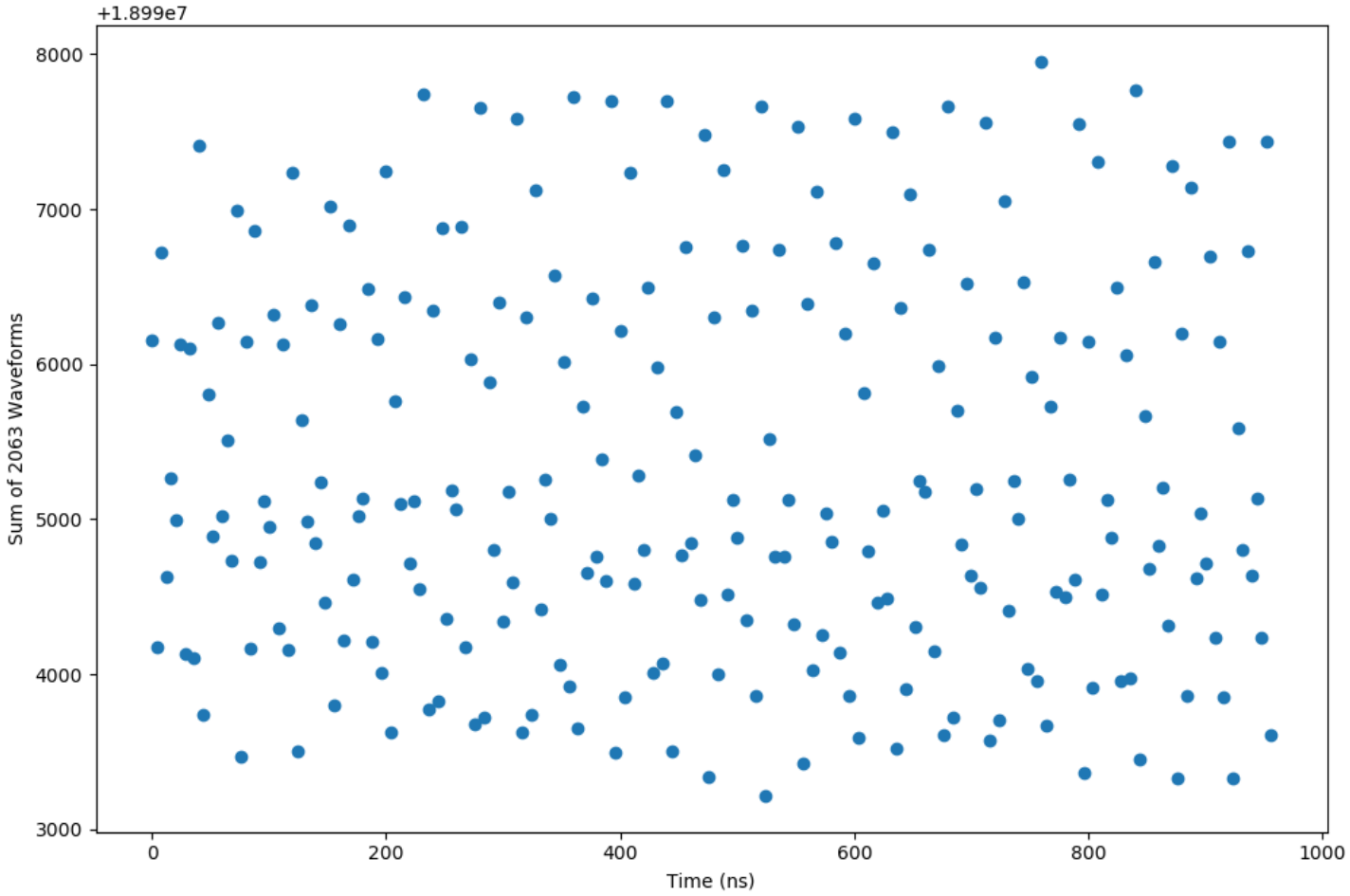
- Programmed the firmware to take a snapshot of the digitizer during when the com_dac fires



The plot here shows the digitizer saturates for 500 ns because I actually connected the MSB of the com_dac[11:0] line to the analog front end – to test my code: that it is taking the snapshot at the right time.

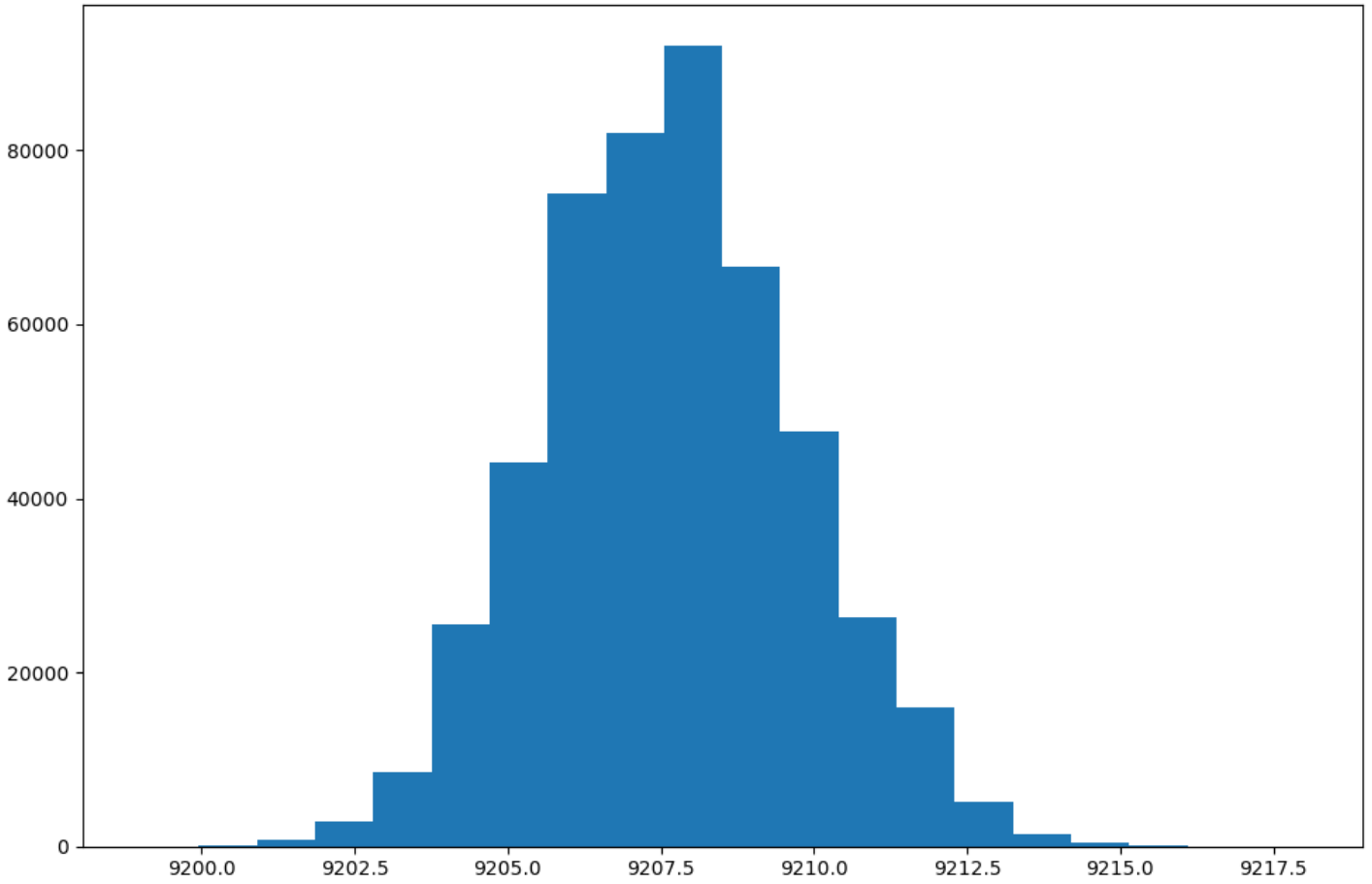
A Simple Digitizer Interference Test

Took 2063 waveforms, ~1 s apart. Added them up and result shown in plot below.



Digitizer Noise

Is this comparable to the results from the ddc2?



To Do Next

- Interference test.
  - Record waveforms from the digitizer when the com_dac fires.

- ~~Figure out why can not send longer packets.~~

New Communications Protocol

In early 2004, a new protocol was implemented in both the DOR driver and the DOM software which included the detection and correction of communications errors. The scheme was based loosely on certain features of TCP/IP and was designed primarily by Arthur Jones, who implemented the code on the DOM side while Jacobsen implemented it in the DOR driver.

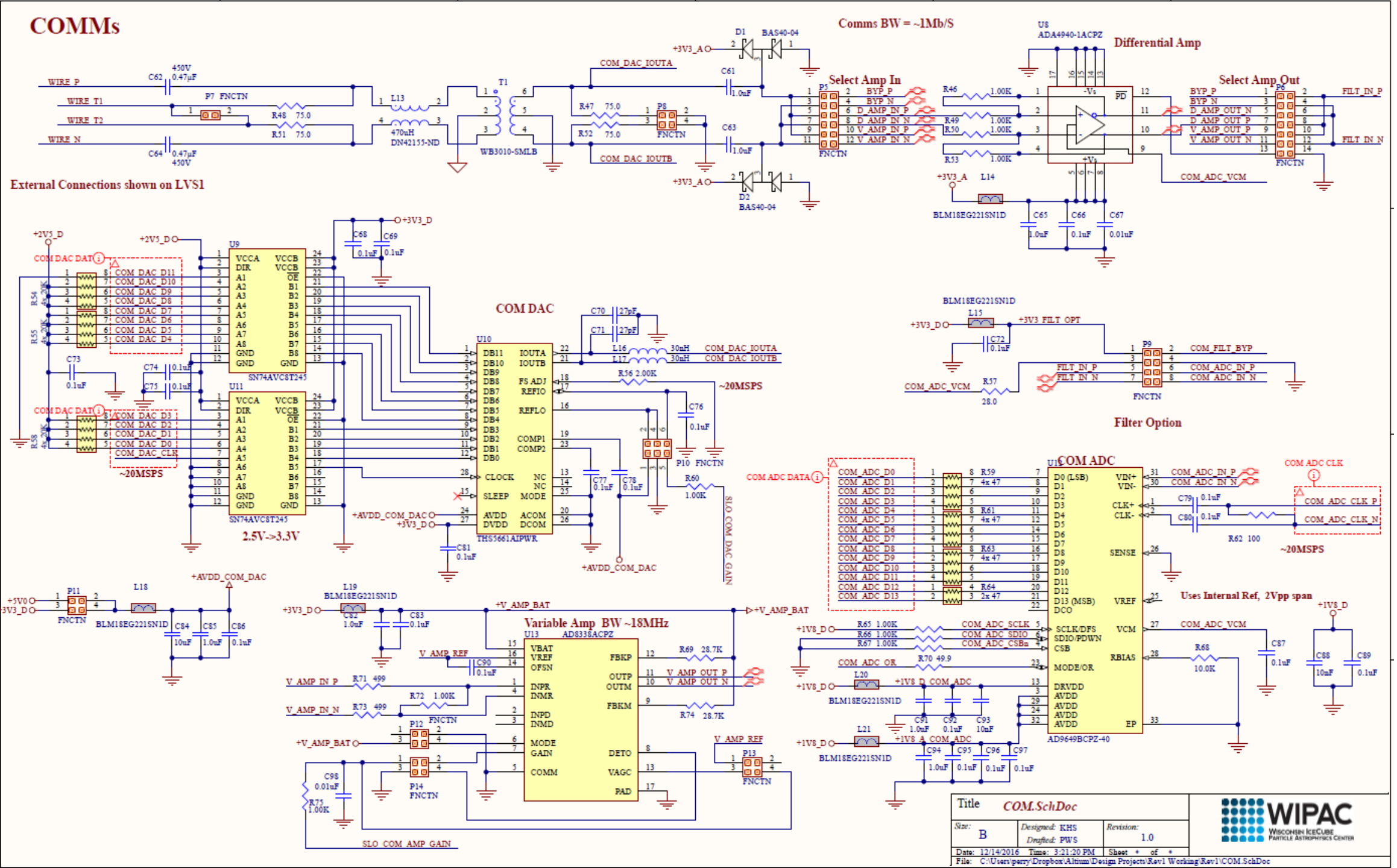
Whereas before there was a one-to-one correspondence between messages written to / read from the DOR driver, and packets sent on the wire by the DOR firmware, the new protocol is somewhat more complicated. Messages larger than 488 bytes is broken up into multiple "hardware packets" and reassembled on the other end. A CRC is calculated for each hardware packet which allows one to detect the occasional bit error introduced by noise in the hardware. Furthermore, upon transmission, each packet is assigned a sequence number. The sequence number and packet type are encoded in an 8 byte packet header, and the 32-bit CRC is appended at the end of the packet. Each hardware packet is acknowleged by the recipient, unless it has a bad CRC or is out of sequence. Packets not receiving ACKs are retransmitted after a small timeout interval. In order to synchronize packet sequence numbers, a synchronization protocol is used whenever a new connection is established (DOM device file opened on the DOM Hub side, or DOM change of state, e.g. from Iceboot to Domapp).

DOR Driver Function and Design, version 2.25
John Jacobsen

Discussions

- Being able to send data to the DOMHub is a big step up. We can now do the interference tests.
- There is still the problem with long data packets (500 bytes+). They definitely get sent (because they get picked up by the Rev1's own comm_receiver module). But somehow, the DOMHub does not receive them.
- These Verilog modules are in no way complete and stable. There are many missing features and functions (known and unknown) compared to the original firmware. Reliability testing missing. Expect many bugs. (Codes can be found here)

Thank you for listening.


COMMs
Comms BW = ~1Mb/S
Differential Amp
Select Amp In
Select Amp Out
External Connections shown on LVS1
COM DAC
~20MSPS
2.5V->3.3V
Filter Option
COM ADC
Variable Amp BW ~18MHz
Uses Internal Ref, 2Vpp span
Title COM.SchDoc
Size: B
Designed: KHS
Drafted: PWS
Revision: 1.0
Date: 12/14/2016 Time: 3:21:20 PM
File: C:\Users\perry\Dropbox\Altium\Design Projects\Rev1 Working\Rev1\COM.SchDoc