A Report on Rev1 Board Communication Firmware Development

Bunheng Ty

WIPAC

08-03-2017

Overview

- I’m trying to get the Rev1 board to talk with the DOMHub.

- So far
  - Got the Rev1 board to listen to DOM-DOMHub communication
  - Wrote a Verilog bit decoder module to decode the bits

- Still need to
  - Test and improve the bit decoder module
  - Set up a FIFO to stream the decoded bits to Nios II
  - Write the software to process the messages and generate the proper responses.

Settings on the Rev1 board

- P22 : change jumper to power the Rev1 board off the DOMHub
- P7 left open : connect Rev1 board as an Unterminated DOM
- P5 and P6 selected to bypass the differential amplifier (U8)
- COM ADC driven by 20 MHz clock from the FPGA
- Scavenged DOM penetrator cable (short) for the connection:
  - Pink (+) to WT2
  - Gray (-) to WT6
  - Black (GND) to WT8

(Rev1 COMs schematic on last slide)

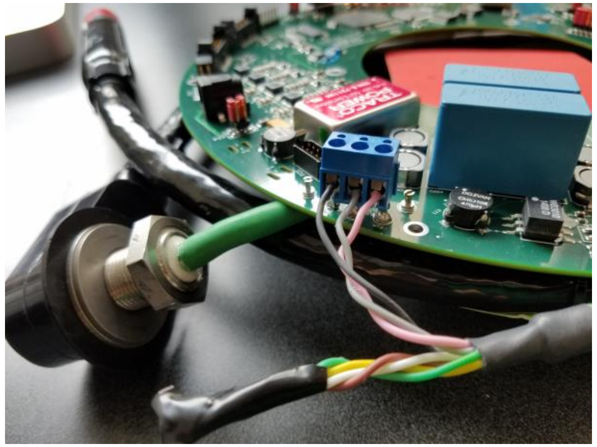
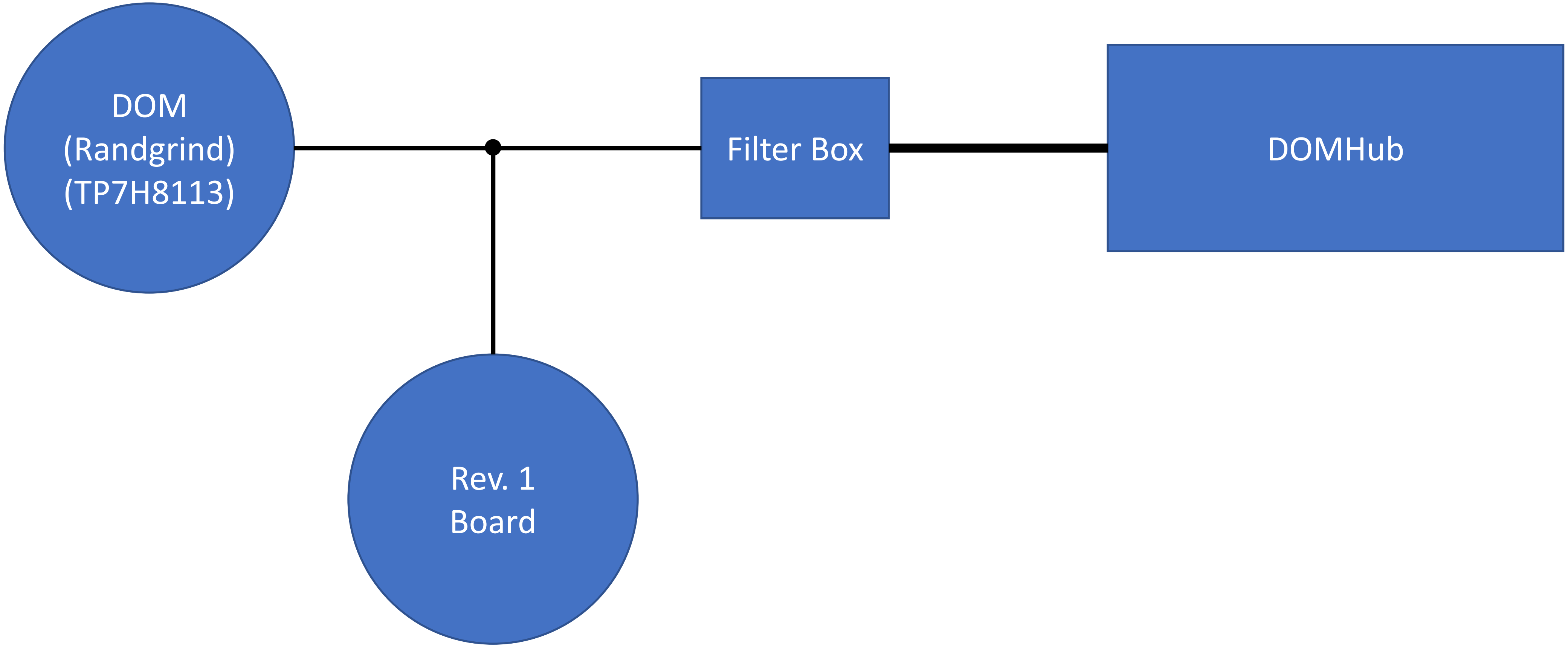
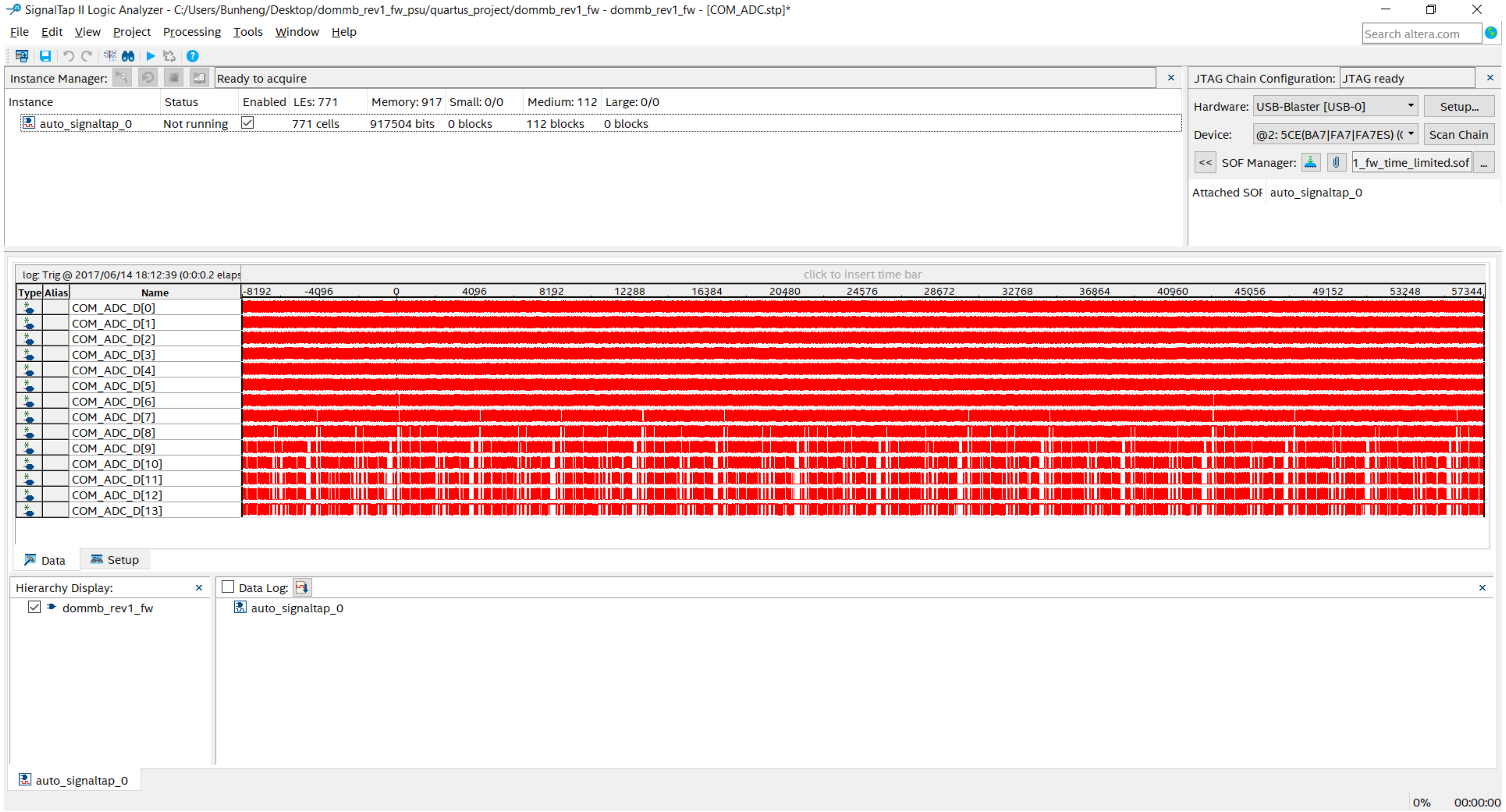


Diagram of the Setup



- Rev. 1 Board listening to Randgrind-DOMHub communications

SignalTap II



SignalTap II Logic Analyzer - C:/Users/Bunheng/Desktop/dommb_rev1_fw_psu/quartus_project/dommb_rev1_fw - dommb_rev1_fw - [COM_ADC.stp]*

Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 771	Memory: 917	Small: 0/0	Medium: 112	Large: 0/0
auto_signaltap_0	Not running	☑	771 cells	917504 bits	0 blocks	112 blocks	0 blocks

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @2: 5CE(BA7|FA7|FA7ES) (

SOF Manager: 1_fw_time_limited.sof

Attached SOF auto_signaltap_0

log: Trig @ 2017/06/14 18:12:39 (0:0:0.2 elaps

Type	Alias	Name
		COM_ADC_D[0]
		COM_ADC_D[1]
		COM_ADC_D[2]
		COM_ADC_D[3]
		COM_ADC_D[4]
		COM_ADC_D[5]
		COM_ADC_D[6]
		COM_ADC_D[7]
		COM_ADC_D[8]
		COM_ADC_D[9]
		COM_ADC_D[10]
		COM_ADC_D[11]
		COM_ADC_D[12]
		COM_ADC_D[13]

Hierarchy Display:

☑ dommb_rev1_fw

□ Data Log:

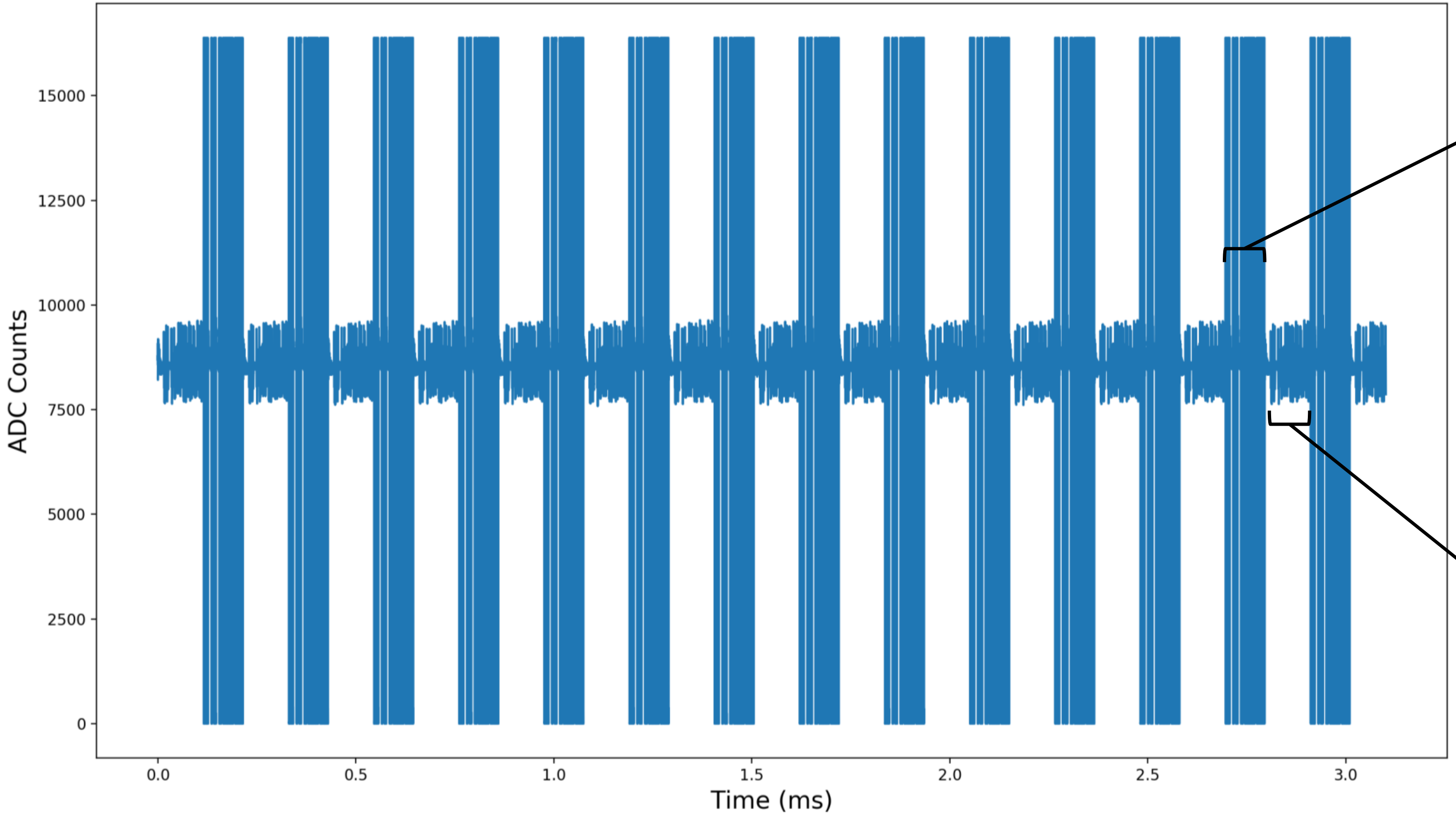
auto_signaltap_0

auto_signaltap_0

0% 00:00:00

- SignalTap II allows signal extraction from the FPGA on the Rev1 board
- All 14 channels of the COMM ADC, 64K samples each @ 20MHz -> 3.2 ms of COM data

Plot of COM Data Extracted using SignalTap II

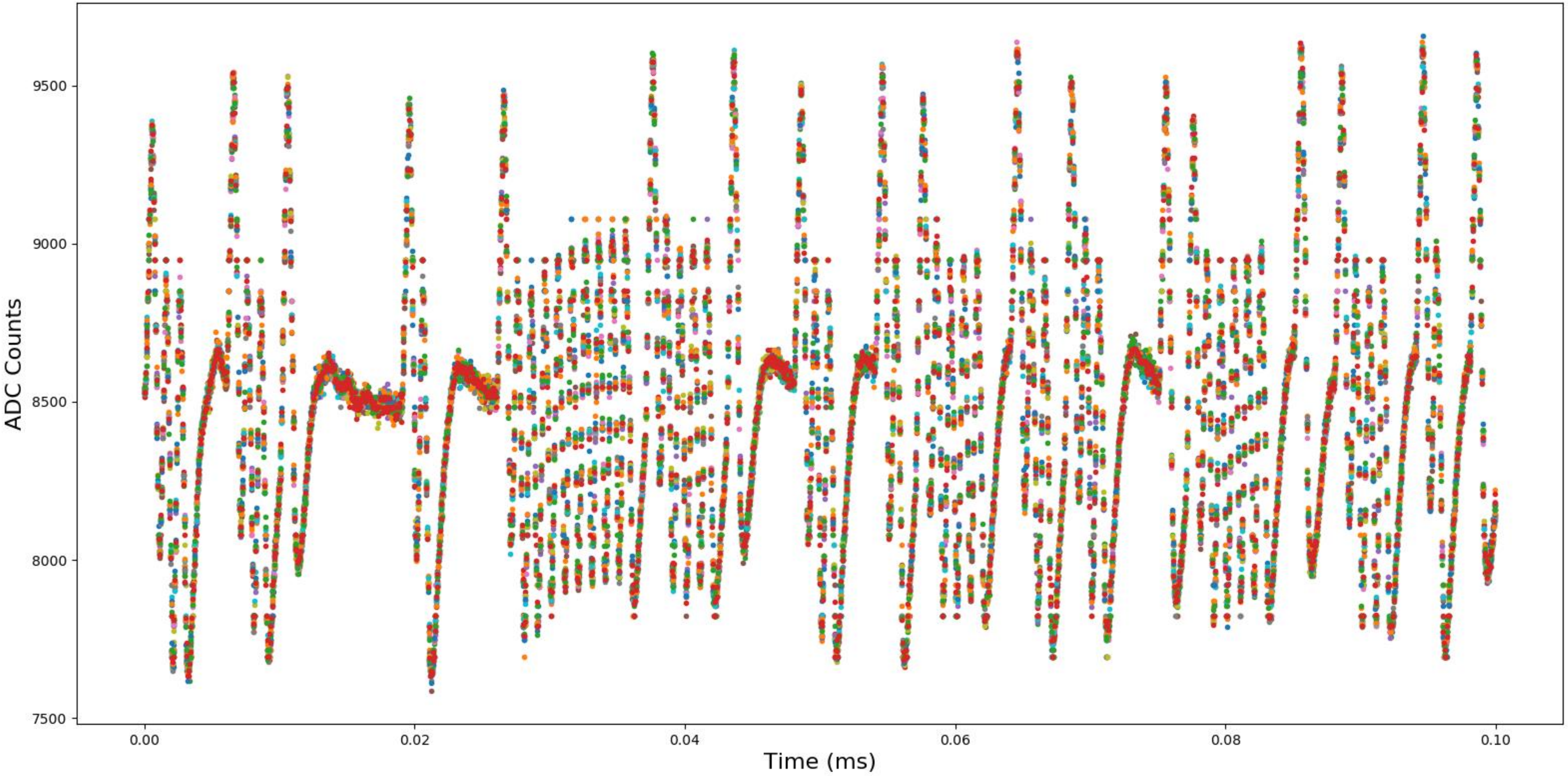


100 bits packet from Randgrind to DOMHub

100 bits packet from DOMHub to Randgrind

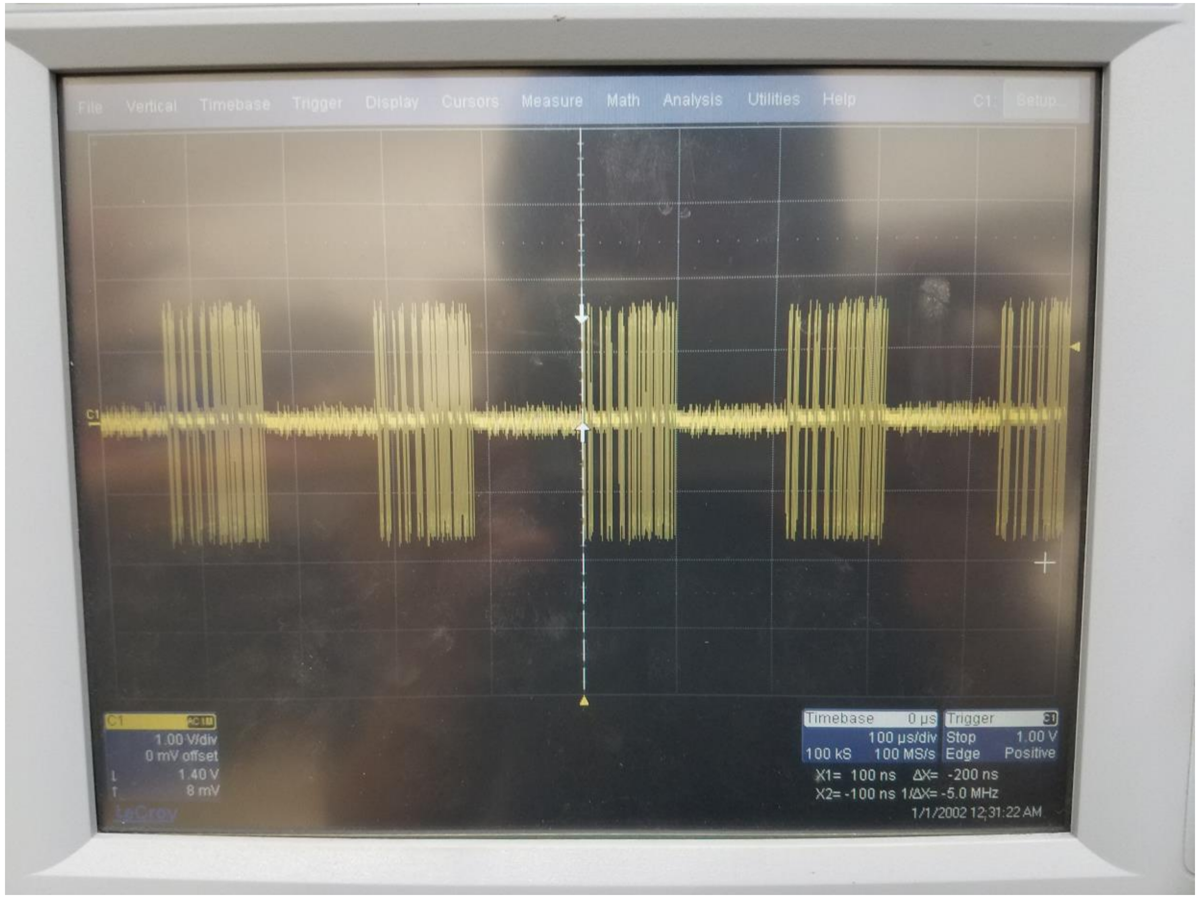
- The DOM is a slave—it can only response. The DOMHUB is the master.
- DOMHub’s message is so much quieter due to having gone through the filter box.

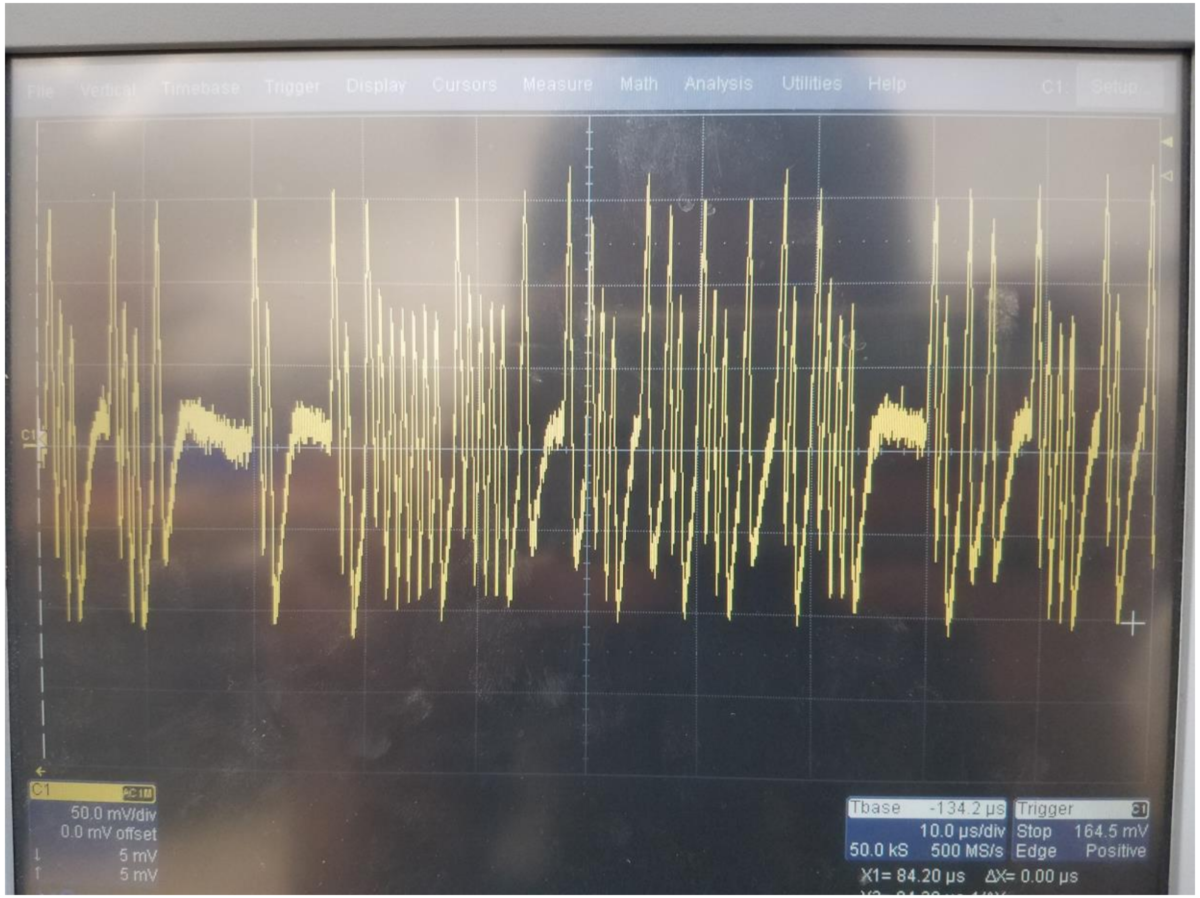
Data Packet from DOMHub to Randgrind



- 14 data packets from previous slide superimposed on top of each other
- Each bit is 1 us long. A “1” is a ‘bisymmetric pulse’. A “0” is a quiet line.

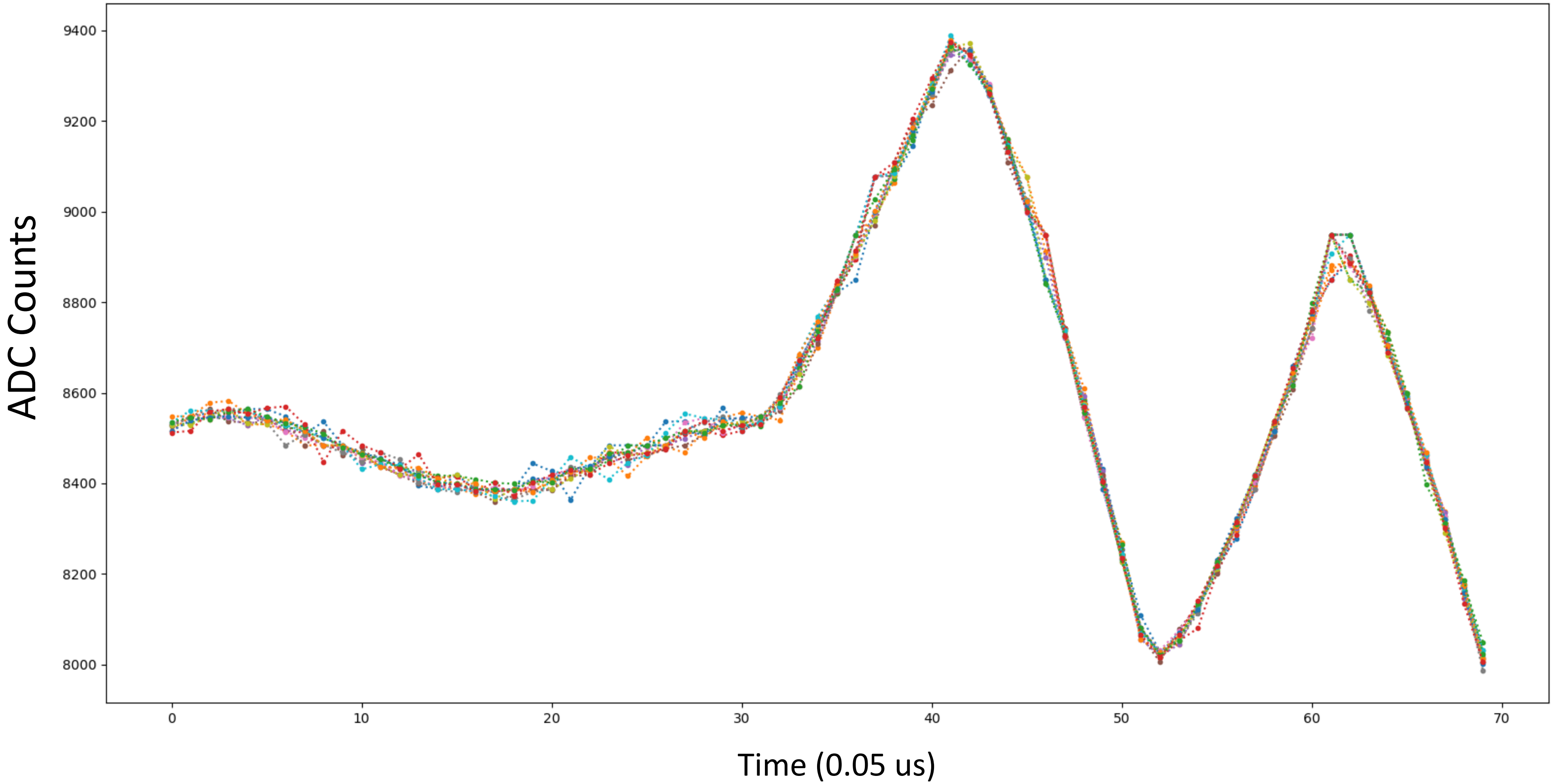
DOMHub – Oden Communications on an Oscilloscope





- Note: A different DOM was used

The “Bi-symmetric Pulse”

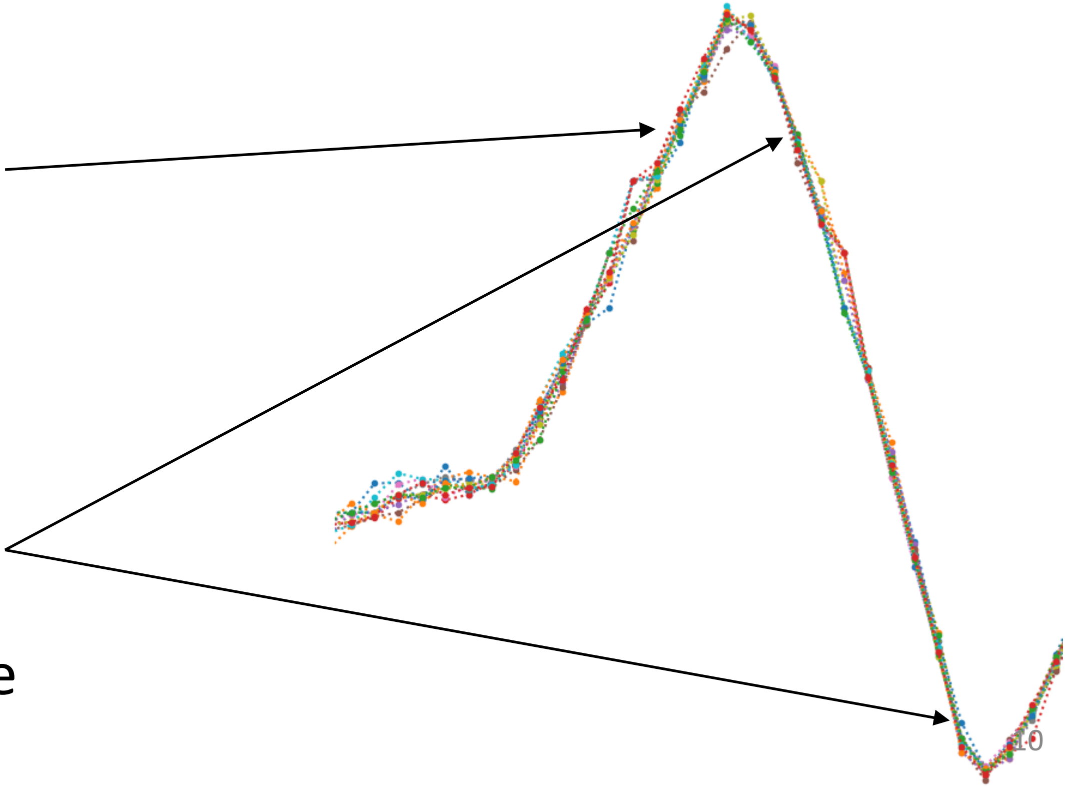


- Stable signal -> easier time implementing the bit decoder module

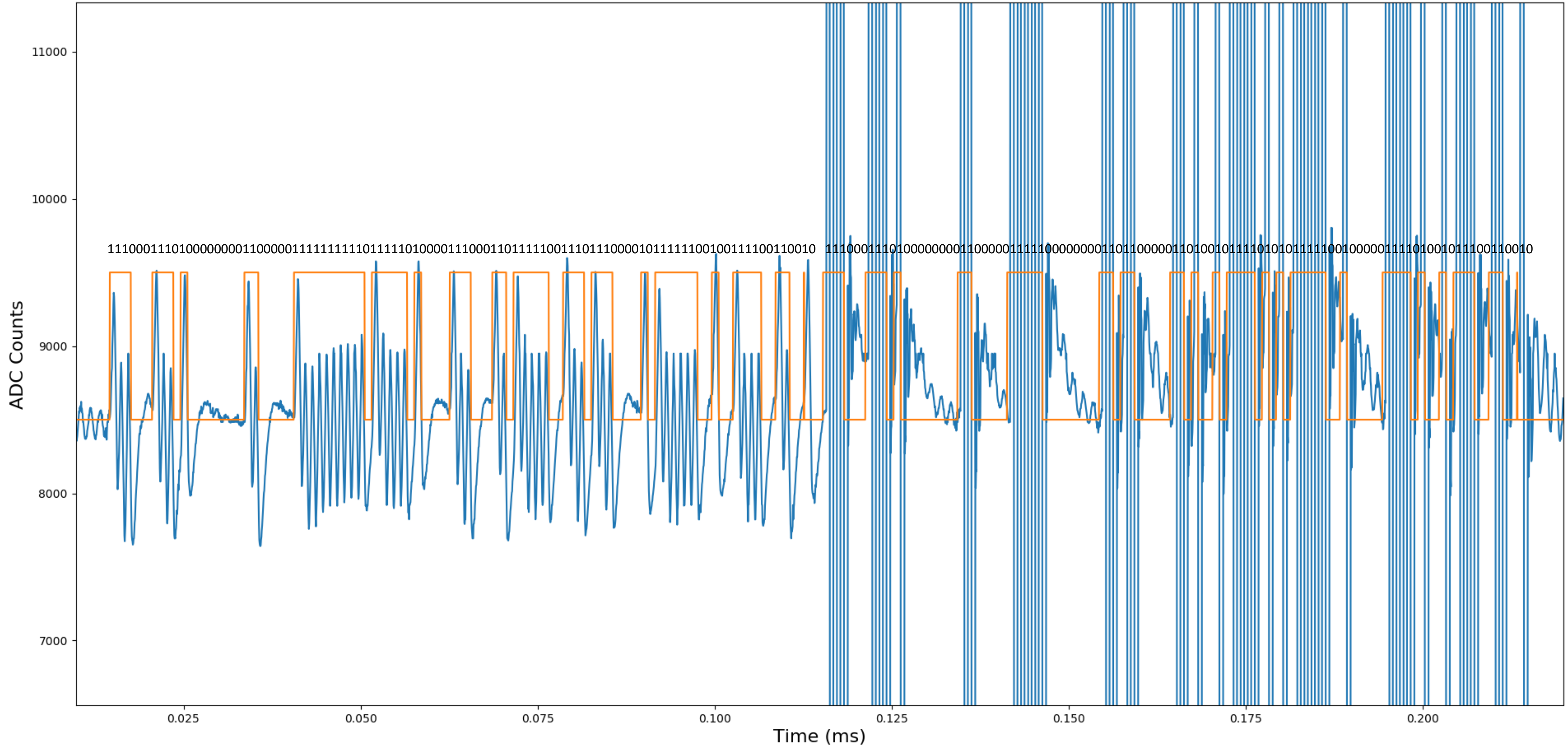
The Bit Decoder module

- A fairly simple state machine, ~50 lines of Verilog codes. That was enough to decode both the filtered and unfiltered signals at the same time.
- Will need to rewrite it to be able to recognize structures in the data packets and reliably interface with the Nios II.

1. Monitor the COM_ADC[13:0] line, trigger when it goes above threshold for a certain number of clock cycles.
2. When triggered, go into decoding mode for 100 us, because expecting to decode 100 bits.
3. For each bit, record the value of COM_ADC[13:0], then for several clock cycles later, record again. Subtract the two values. If larger than a certain threshold, the bit is a “1”, otherwise a “0”.



Bit Decoder Outputs in ModelSim



- Simulation only, the module has not been tested in actual hardware

Structure of the COM Data Packets

- The DOMHub-DOM communications protocol is quite sophisticated. “TCP lite”
- Check this document “DOR – API Description” for more information.

The DOMHub to Randgrind Packet

1110001110	:	“0xE3” in little endian. Start-of-frame byte. Always the same for every packet.
1000000001 1000001111 1111110111 1101000011	:	No data. This packet is the “DOR control” type. It says “data read request”
1000110111 1100111011 1000010111 1110010011	:	For Error checking, CRC-32
1100110010	:	“0x99” , End-of-frame byte. Always the same for every packet.

- The greyed out bits are the start/stop bits

Structure of the COM Data Packets

- The DOMHub-DOM communications is quite sophisticated. “TCP lite”
- Check this document “DOR – API Description” for more information.

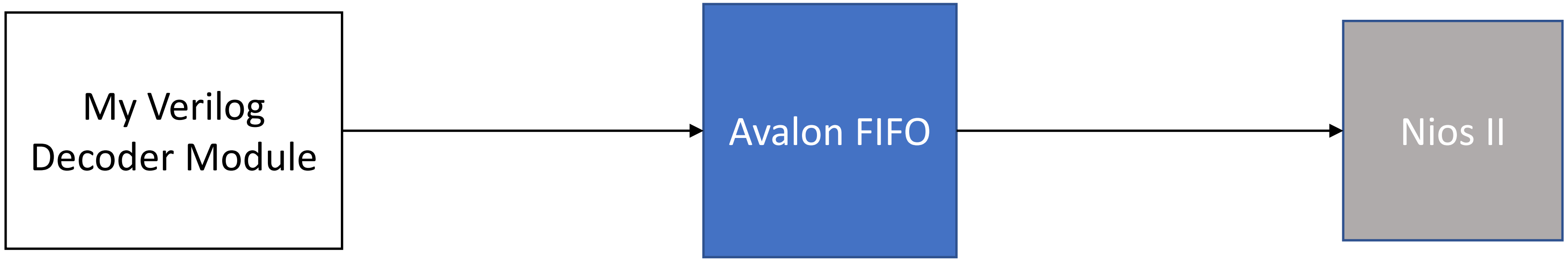
The Randgrind to DOMHub Packet (responding to the packet from previous slide)

1110001110	:	“0xE3” in little endian. Start-of-frame byte. Always the same for every packet.
1000000001 1000001111 1000000001 1011000001	:	The DOM responses to the data read request with “no data”
1010010111 1010101111 1001000001 1110100101	:	For Error checking, CRC-32
1100110010	:	“0x99” , End-of-frame byte. Always the same for every packet.

- Initially I thought these were “Idle” packets. But they are not, as shown here.
- Packet length can vary. The max length is 41040 bits.

To Do Next

- Need to set up an Avalon FIFO to pass the bits from the decoder to Nios II. (Still haven't figured this out exactly)



Should I use Avalon-MM or Avalon-ST for the FIFO interface?

- Then I can tell the Nios II to record the exact messages between Randgrind and DOMHub during initial boot up, going into IceBoot, data transfer, or anything else.

Conclusions

- The first step towards Rev1 Board – DOMHub communications has been taken. But there’s still a lot of work to do.
- The hardest part, I think, will be writing the software for the Nios II to response correctly to all types of behaviors from the DOMHub.

- Comments and suggestions, I would appreciate very much.

Thank You!