4/6/22

Config map

ADC library → check old lib with Serial plot

Next:

1) Add hardware timer

2) Add try 0 interrupt (TDC prior)

3) Attempt to hook up cmp as well

4) For each direction do a manual spin look for 20 Virtual readout

5) Binning software for calculating Zero Crossing angles.

6) Software to find transitions from ADC with kernel

5/6/22

ADC -

→ Remove block commutation related code, except single PWM

ADC	ACMP	phase
14	21	A
15	22	B
16	23	C
17	18	VN

It has 3 identical chains

Reverse seems to work.

Does not seem to work with ADC & ACMP both at the same time

→ investigate why

Trigger chain 4 readings via etc 0, 1, two readings each.

3 trigger chains for 3 phases.

Work todo:

1 GPT timer @ n x fadc (n atleast 2)

Only need 1 pwm @ frequency fadc

Only need 1 trigger chain with etc 0 & 1

SPI read angle

GPT callback → increment timer

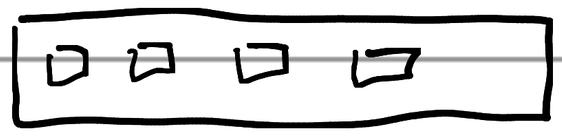
ETC_0 callback → collect A & B voltages & record time

ETC_1 callback → collect C & UN voltages & record time

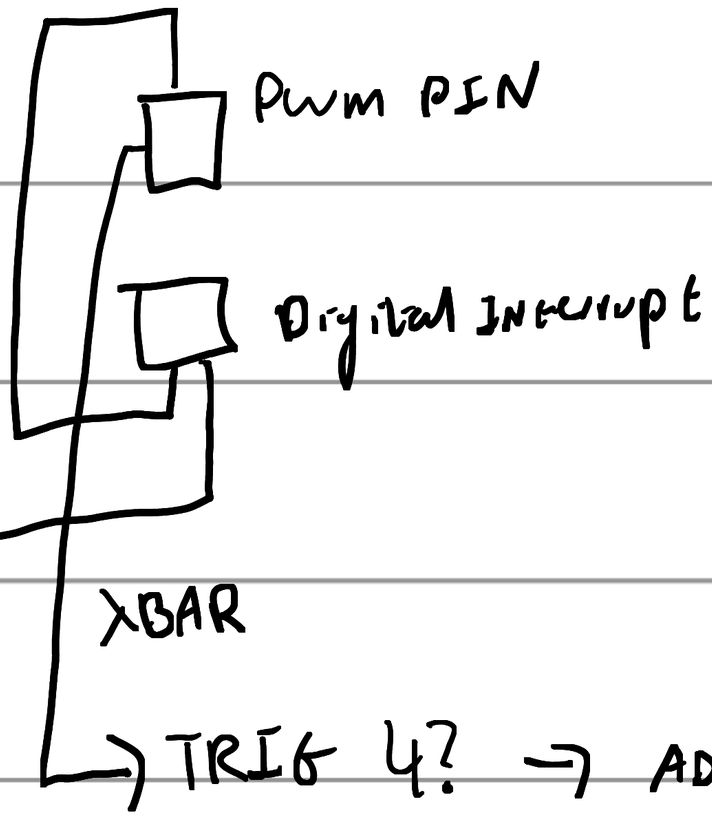
SPI read in loop (perhaps avoid timeouts & only print to serial here)

Output
Voltages/time^avg
angle/time

ADC



A B C VN



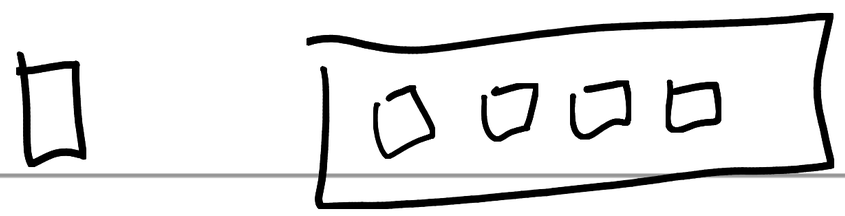
SPI

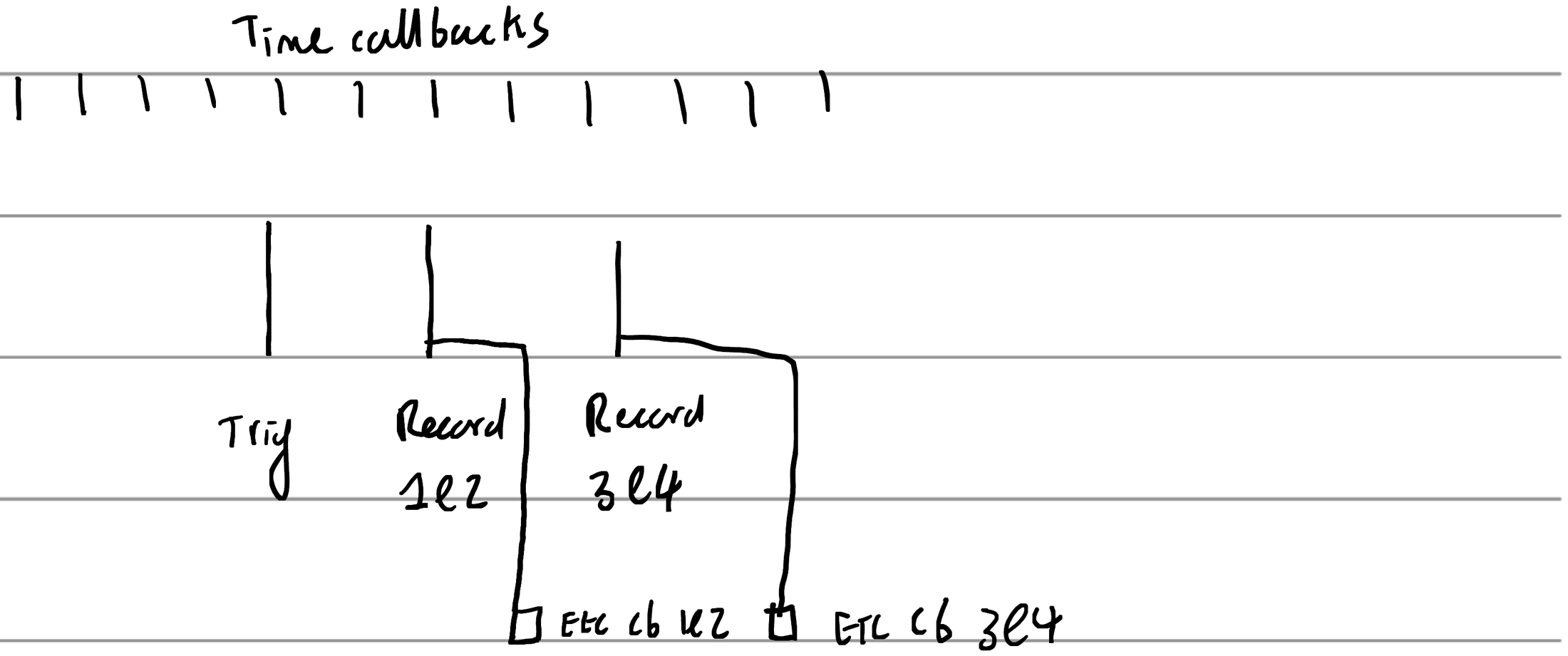
→ TRIG 4? → ADC ETC 0/1

→ TRIG 3? equivalent

↳ Record time before

Pwm





Record time

send time & voltages

(maybe on send in the loop to avoid halting interrupts)

loop

→ Send read command

SPI

→ record time just before clock pulse (closest temporally)

&

voltages from latest ADC + its time.

→ print values to serial out

6/6/22

ADC: Simplest setup trigger ADC via PWM trigger, have timeCtr increment on the 2nd ADC_ETC callback.

Implemented code to take ADC measurements ✓

Add PS4 input code (controller)

Find Encoder max sample rate

Freq to high overloads serial plot & monitor ... write custom code

Find highest serial output data rate

Refactored repository README.md in each directory.

7/6/22

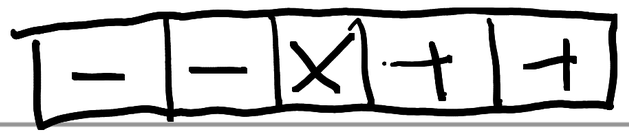
Created assembly for a power drill to directly drive a blde motor. So i can take temp measurements & determine zero crossing points from a motor & create a calibration map

Next → Test ADC at hyper frequency (custom logger?) driven by a power drill.... hopefully the voltages are strong enough.

→ Write up SPI interface for encoder

→ Write code to detect & classify zero-crossings,

2D kernel?



Validate magnitude to avoid noise... low value filter?

Need to validate to 6 steps follow in a consistent pattern for forward or reverse directions.

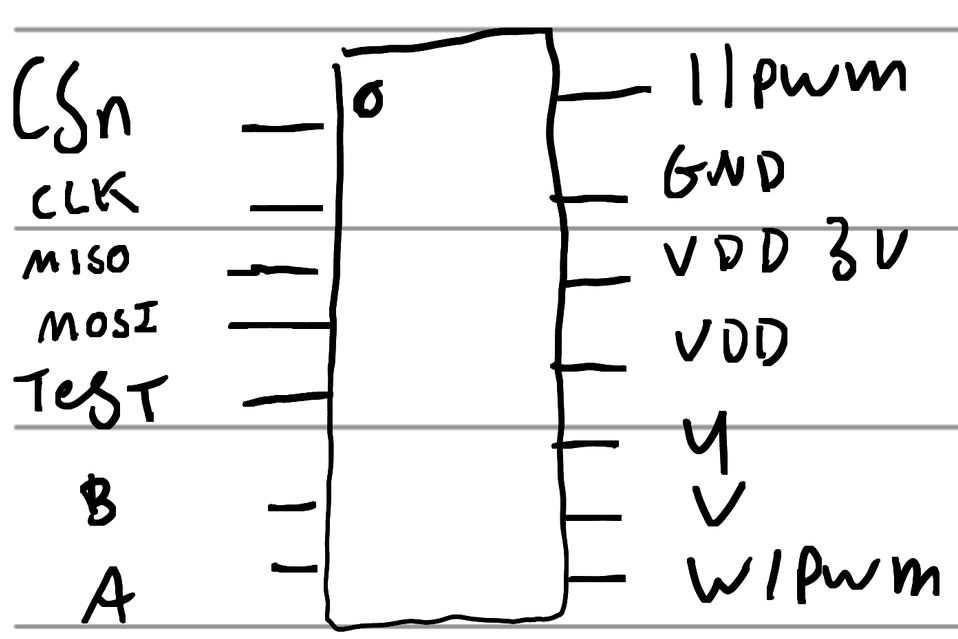
Determine sequence has UN consistently ? 0

→ Eventually would be nice to have an axle measurement at about 30khz if voltages permit, or about 2 samples unique angle ADC

Longer term when encoder is finished build a histogram of angle vs zero crossing points, variable sequence, approximate error. etc...

8/6/22

14 bit encoder AS 5147 P



Pin	Name	Type (I/O)	Description
✓ 1	CSn	I	SPI chip select
✓ 2	CLK	I	SPI clock
✓ 3	MISO	O	SPI master data input slave out
✓ 4	MOSI	I	SPI master data output slave in
✗ 5	Test	/	
✗ 6	A	O	Increment signal A
✗ 7	B	O	Increment signal B
✗ 8	W/PWM	O	Commutation W
✗ 9	V	O	commutation V
✗ 10	U	O	Commutation U
✓ 11	VDD	Power	5V power
✓ 12	VDD3V	Power	3.3V power
✓ 13	GND	Power	Ground
X 14	I	O	Increment signal I or PWM.

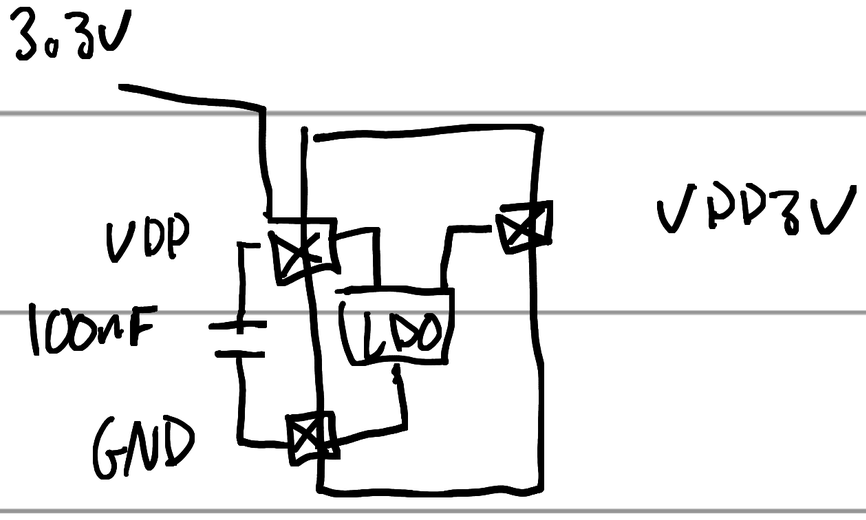
Teensy

5V is VDD as labeled on encoder ← Breadboard

Reserved	Available
PWM	
A_{IN} 2	CS 10 Green
A_{SD} 1	SCK 13 Blue
B_{IN} 9	MISO 12 gray
B_{SD} 0	MOSI 11 Purple
C_{IN} 8	
C_{SD} 7	
ADC	
A 14	
B 15	
C 17	
VN 16	
GND	

Replace 13?
↳ too noisy!

3.3 V operation



(5V)

100nF between VDD & GND

VDD & VDD3V bridged.

Breakout pin layout

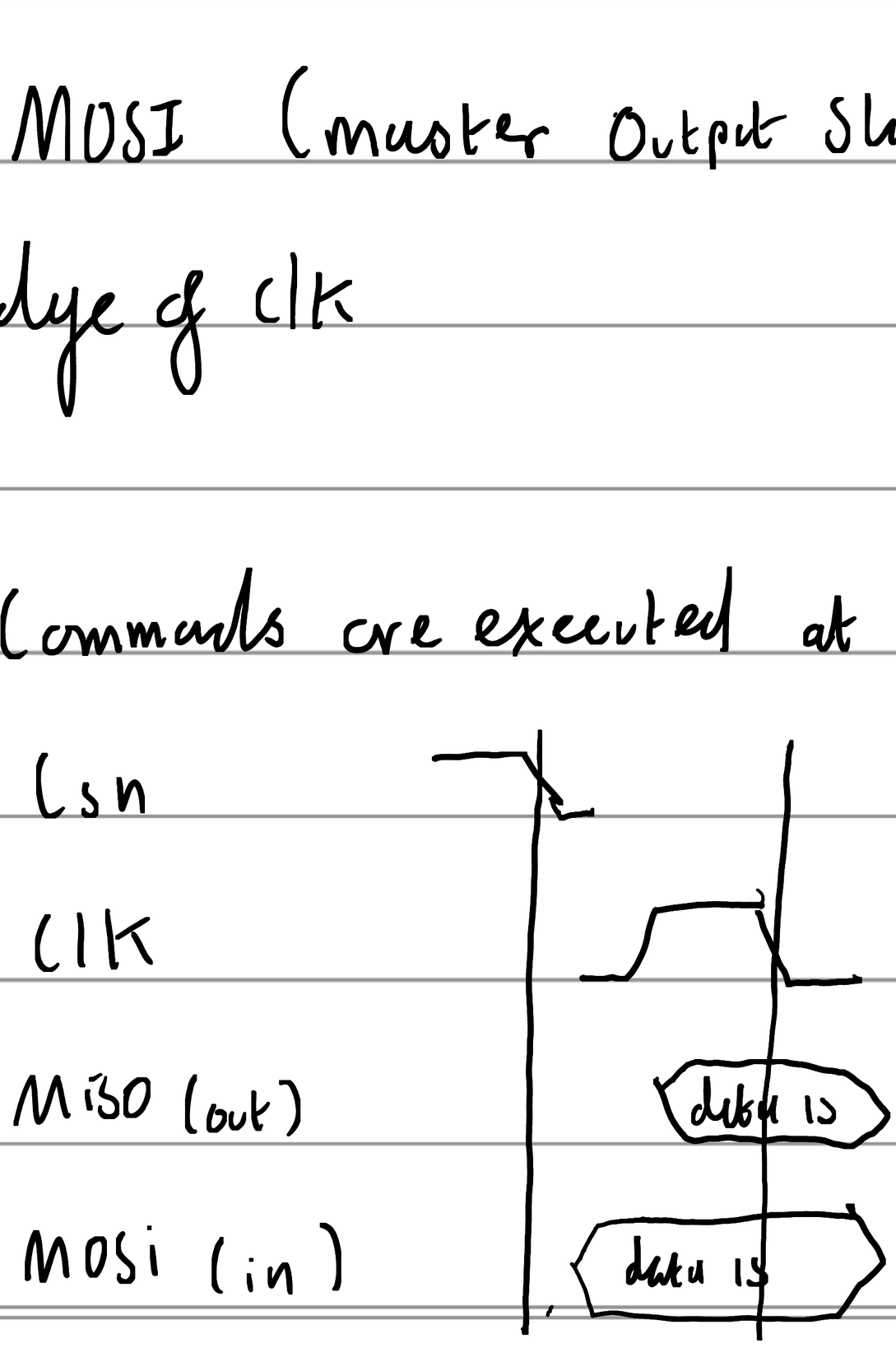
VDD	3V3	~~NC~~ NC	Csn	CLK	MOSI	MISO	GND

Data transfer:

Data transfer starts with falling edge of Csn (Clk low)

MOSI (master Output Slave Input) sampled on falling edge of clk

Commands are executed at the end of the frame (Csn rising)



etc...

Procedure

Each frame CSn high → low

Each bit of command

1) Set CLK High → wait → low

2) Set MOSI high (keep as command is just FFF...)

3) Read MISO bit

After all at end of frame CSn low → high.

Don't forget the value is only 14 bits!

Building little assembly to hold the magnet above the sensor, with screw nuts to adjust the standoff distance from the encoder. I can rotate the magnet by twisting the screw.

Sensor header pins soldered to the encoder PCB.

Sample rate

The drill drives the bldc motor at about ~ 8 hz

at 2^{14} 16,384 steps per cycle it would be nice to have an ADC sample at each step, or more if we also record the time to differentiate the order. So say we sample twice we would get 16 Hz * 16,384 we need a pwm triggering the ADC at 262,144 Hz.

What are we transmitting

time	phase A	phase B	phase C	phase VN	rotor step
32 bits	12 bits	12 bits	12 bits	12 bits	14 bits

94 bits x 262,144 Hz = 24,641,536 bits / second

24.64 Mbps! (50% greater)

Without the time 62 bits x .... = 16,252,928 bps = 16.25 Mbps

In ASCII for convience (without time) 4096, 4046, 4096, 4096, 16384/n

(with time)

32bits 4,294,967,296 10 chars 20+6 26 bytes

36 bytes 288bits [75.4 Mbps] YIKES 208bits

54.82MBps (maybe!)

Will depend on if the host can keep up otherwise it will move like

Teensy 4.0 wait. Will it buffer? Try & redirect tty to ram file, which

is GBps region. (stty raw; cat.) < /dev/ttyAXXX something like that.

9/6/22

Finished assembly setup today, soldiered together the cable which

connects the drill driven test blde motor & finished the encoder

container which allows for a height adjustable screw which suspends

a magnet above the encoder.

Next...

1) Check wiring (Encoder, SMT → Motor cables, SMT → Teensy) ✓

2) check ADC code & hardware works. ✓

3) write encoder SPI code ✓

4) Test encoder works (debug etc) ✓

5) Find the limit of usb serial transfer (can we do it with ASCII at 7200 kHz?) ✓

6) Combine the ADC code with Encoder code ✓

7) Write some code to find the 6 states of the zero crossing & their repetition around entire rotation of the motor (couple), this code must validate, that the sequence is continuous, VN never drops too low, noise is ignored & the direction the motor was spinning during one run.

8) Write code which takes two valid datasets of a given motor, in opposite directions. Rotates each map from the zero crossing point to the ideal switching point (~30° electrical) given the spinning direction. Superimpose these two maps & plot on a chart to check for symmetry.

10/6/22

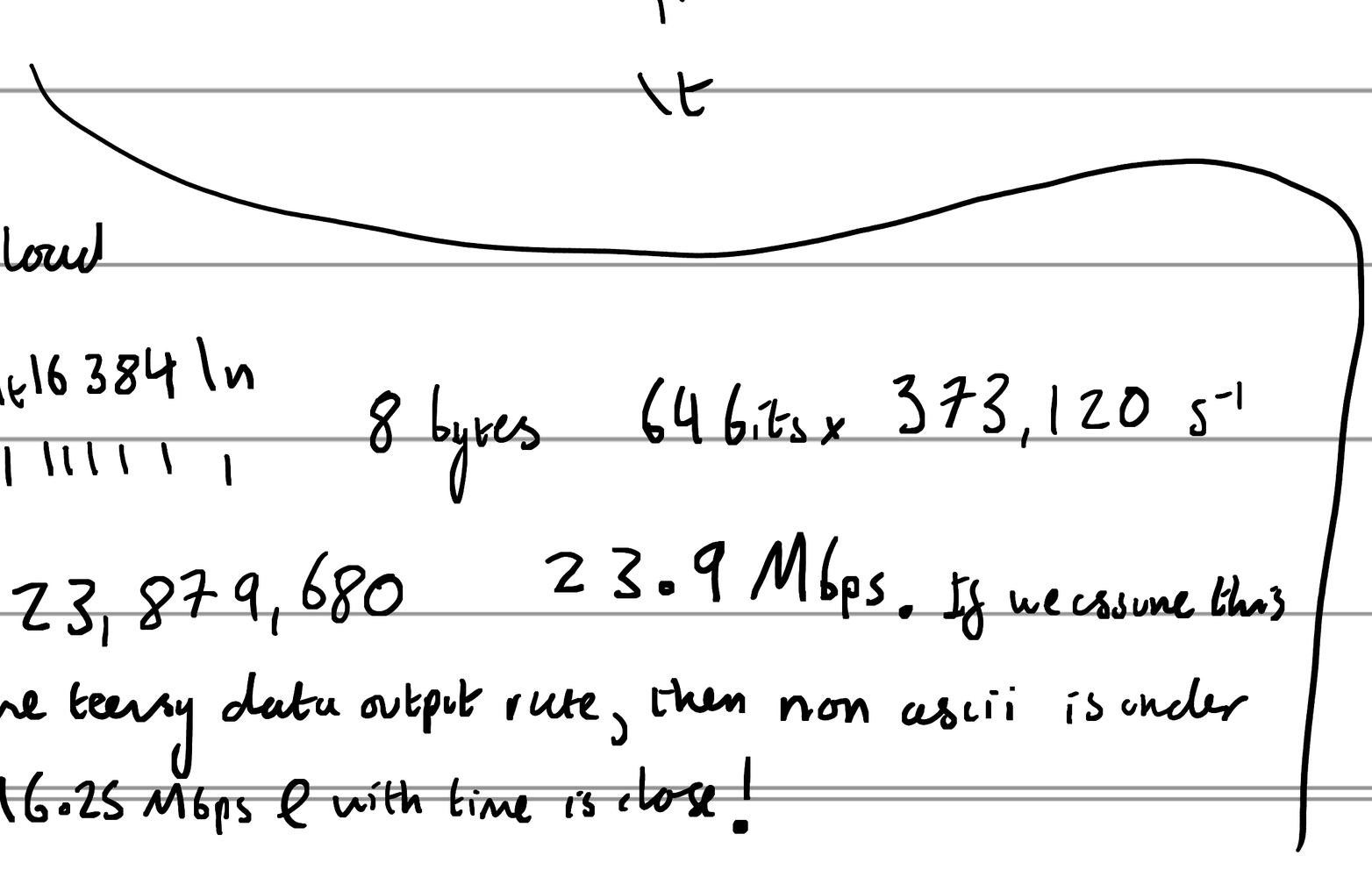
Finished 1, 2, 3 & 4. Next a little refactor so I can do (6), need a single configurable sample rate. Read cycle in ADC interrupt?

Perhaps even printing to serial, need to fix caching / sampling ADC to ensure sync is not lost between readings {A, B} & {C, V} which occur seperately

Perhaps in ADC code need to subtract a zero point offset? check data quality near jest before drill pulse.

(5) got encoder delay down to 3ns per bit to read otherwise parity fails.

serial lines of valve, parity \n in ~1 second of 373,120



Payload

Can just write a buffer using Serial.write(buffer, len-buffer)

14 bit angle UINT (probably 16 bit for convience)

1 bit encoder parity

(4) x 12 bit ADC UINT 48 bits

63 bits or 65 bits with 16bit angle

Say we log at 300 kHz then 19.5 Mbps, perpare for lots of noise!

11/6/22

ADC + Encoder combined

Implemented (6) & registered scripts thinking about (7) need a new python library to look at zero crossings.

ADC VN channel just looked like noise... try swapping VN to another channel

So v_N is very noisy depending on the change of position of the encoder. Guess is the data transfer via SPI is creating noise in ground & thus v_N.

One could perhaps approximate v_N as $v_N = \frac{V_a + V_b + V_c}{3}$ so long as encoder noise is not affecting signal of phase A, B or C but i think v_N is most sensitive to noise in ground.