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

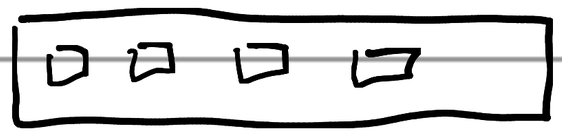
ETC_0 callback → collect A & B voltages & record time

ETC_1 callback → collect C & UN voltages & record time

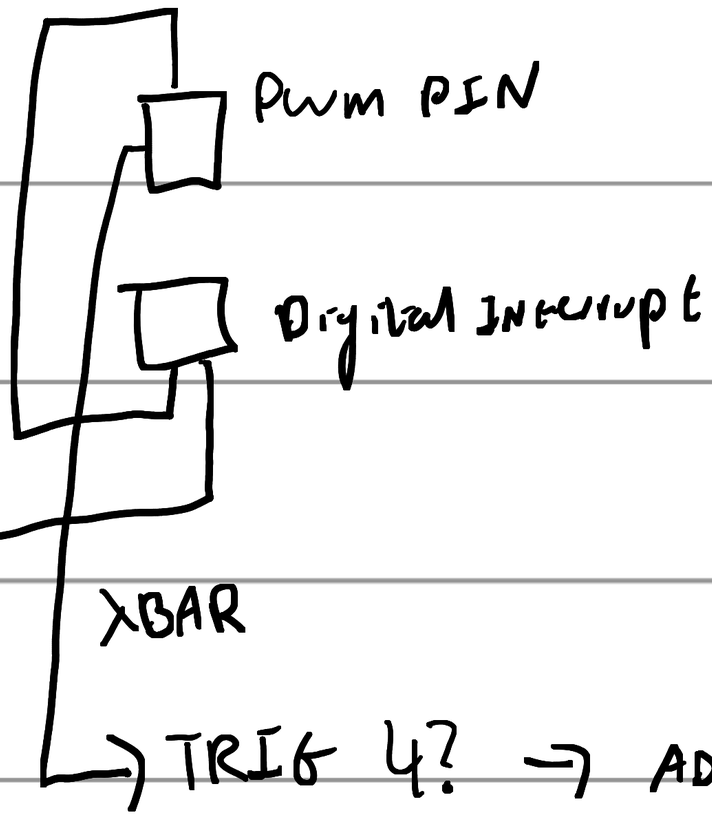
SPI read in loop (perhaps avoid timeouts & only print to serial here)

Output
Voltages/time^avg
angle/time

ADC



A B C VN



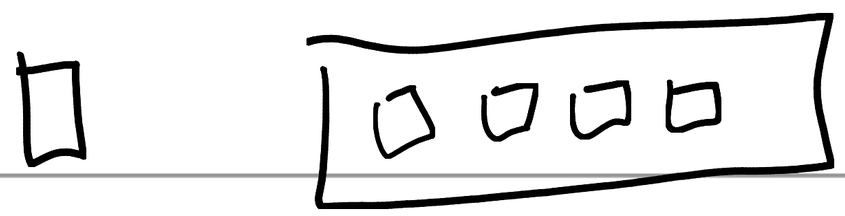
SPI

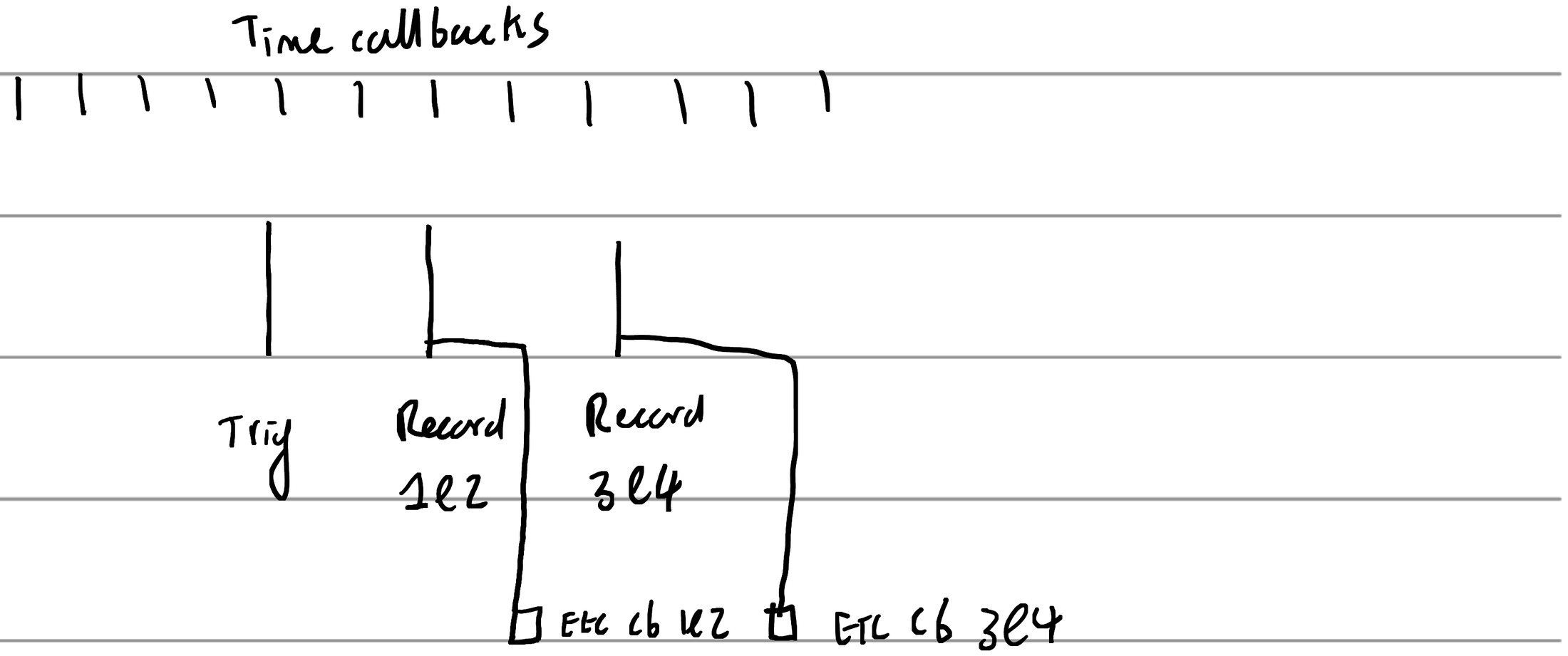
→ TRIG 4? → ADC ETC 0/1

→ 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

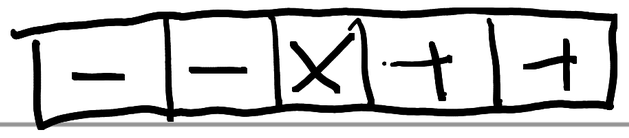
Created assembly for a power drill to directly drive a blde motor. So i can take temp measurements & determine zero crossing points from a motor & create a calibration map

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.

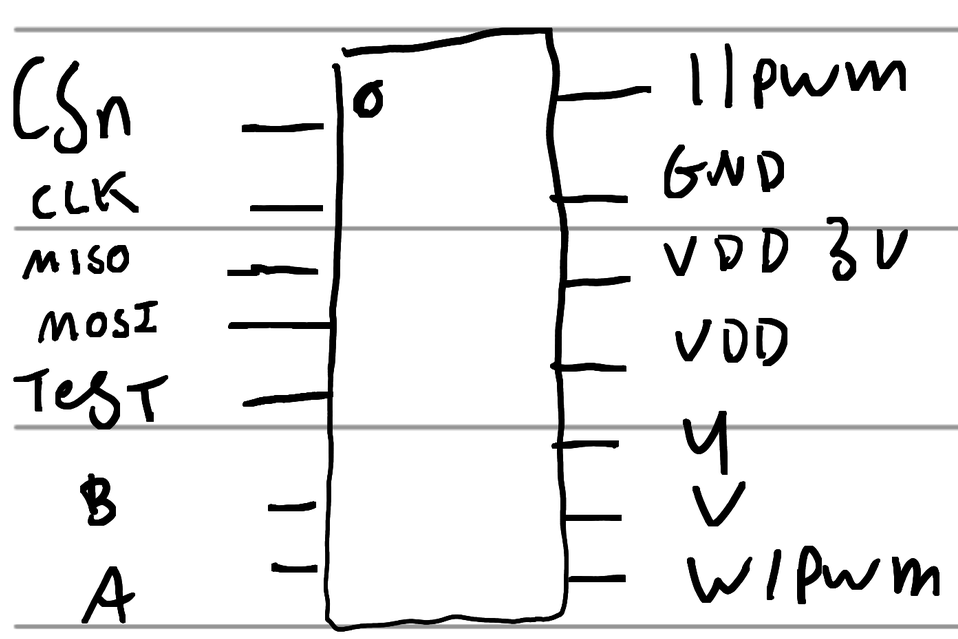
Determine sequence has UN consistently ? 0

→ 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.

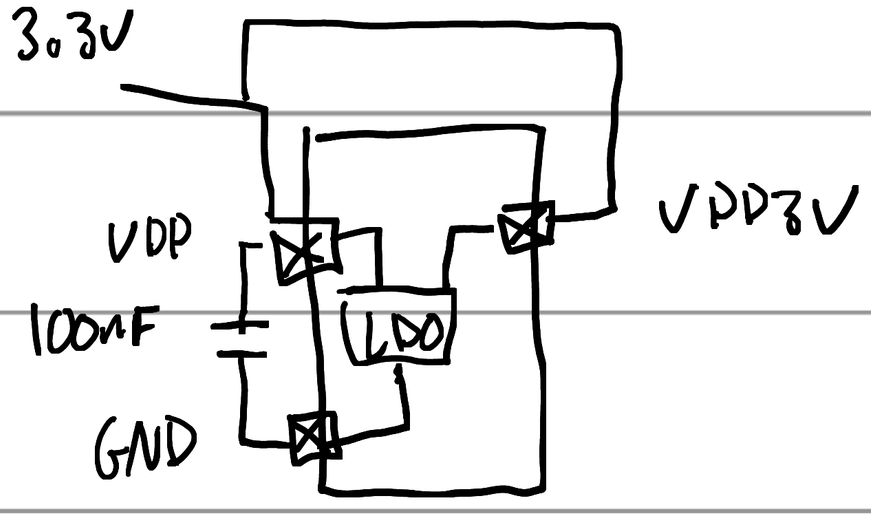
Teensy

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

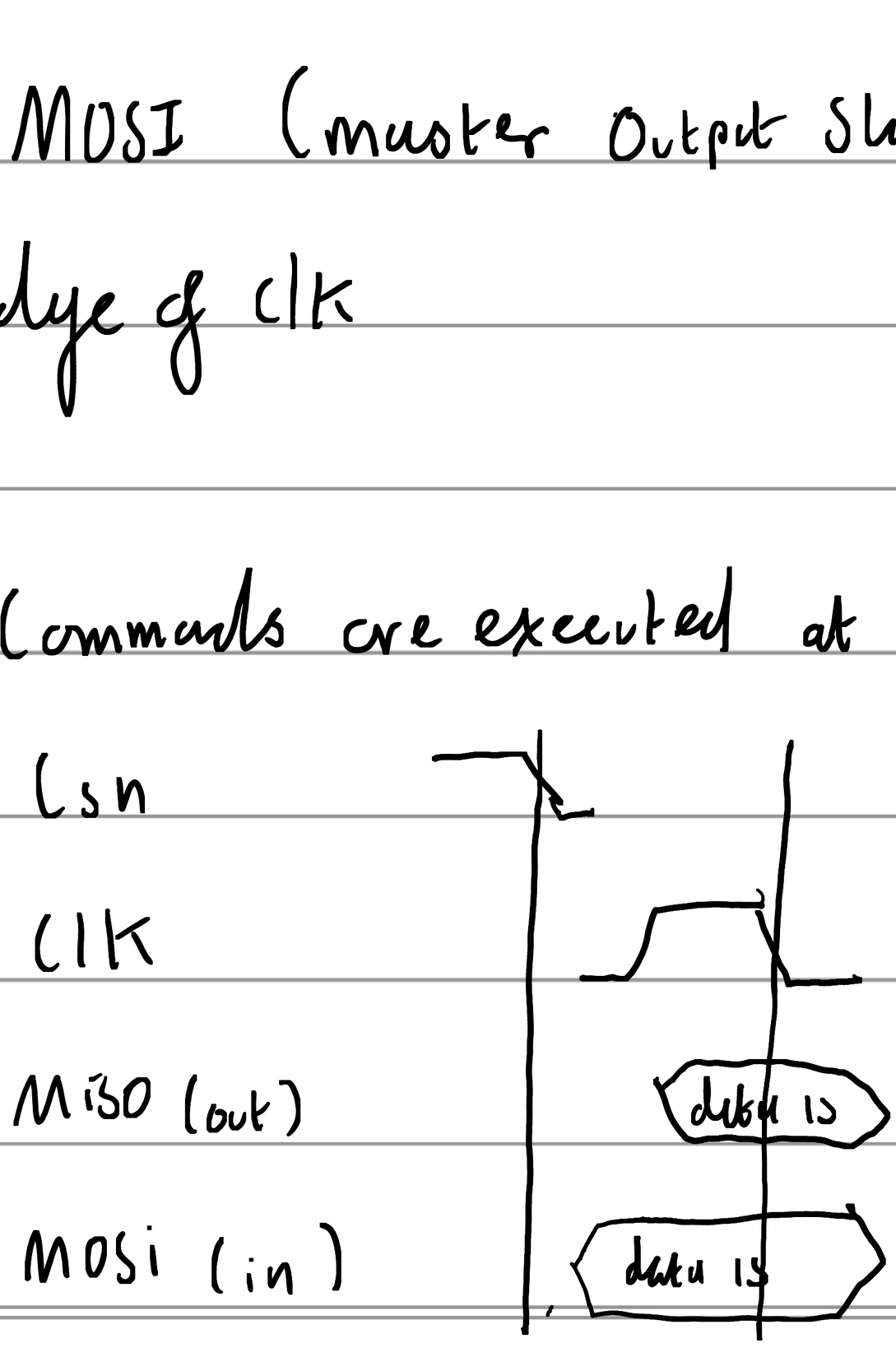
VDD	3V3	~~NC~~ NC	Csn	CLK	MOSI	MISO	GND

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

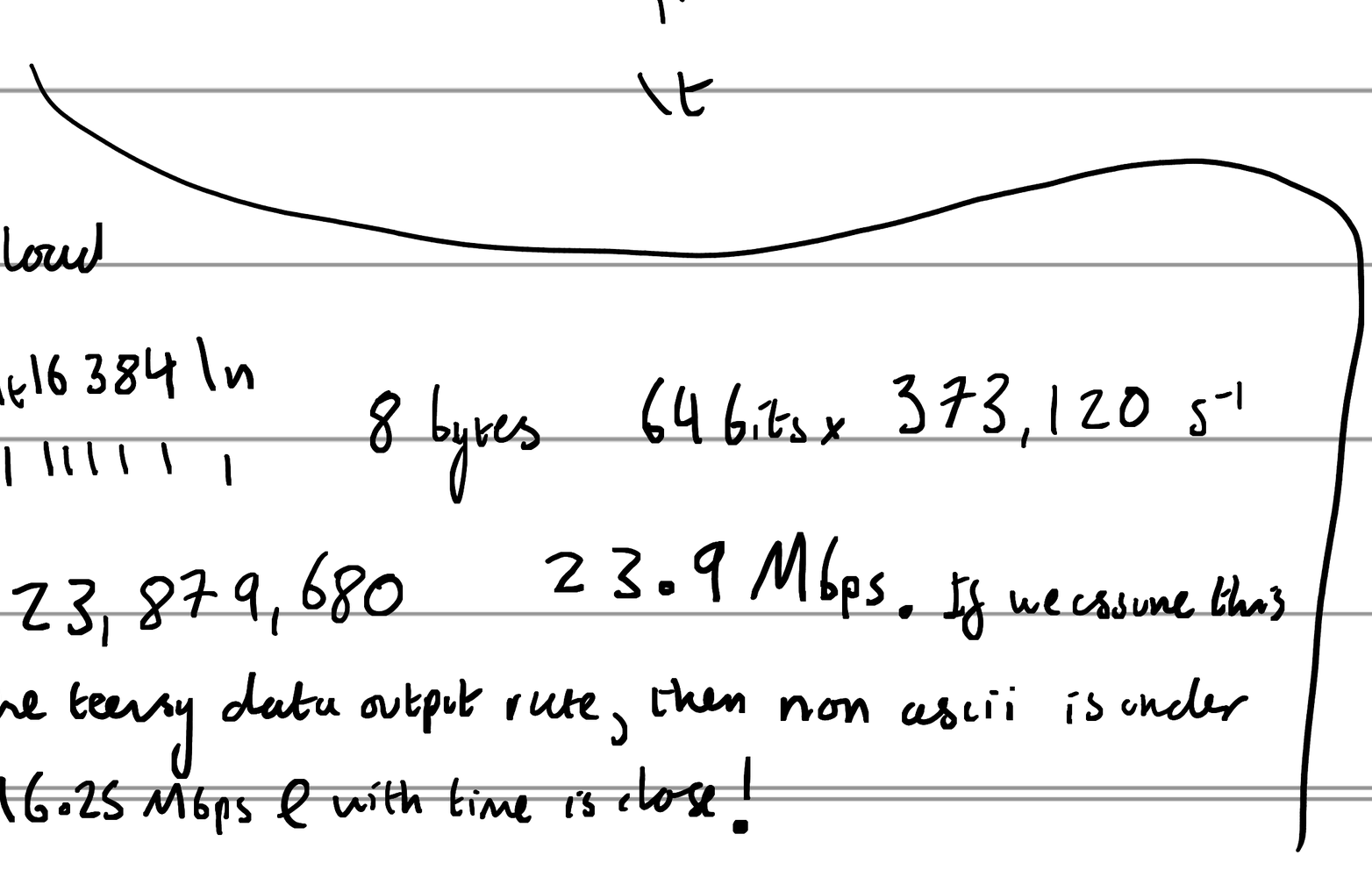
Finished 1, 2, 3 & 4. Next a little refactor so I can do (6), need a single configurable sample rate. Read cycle in ADC interrupt?

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

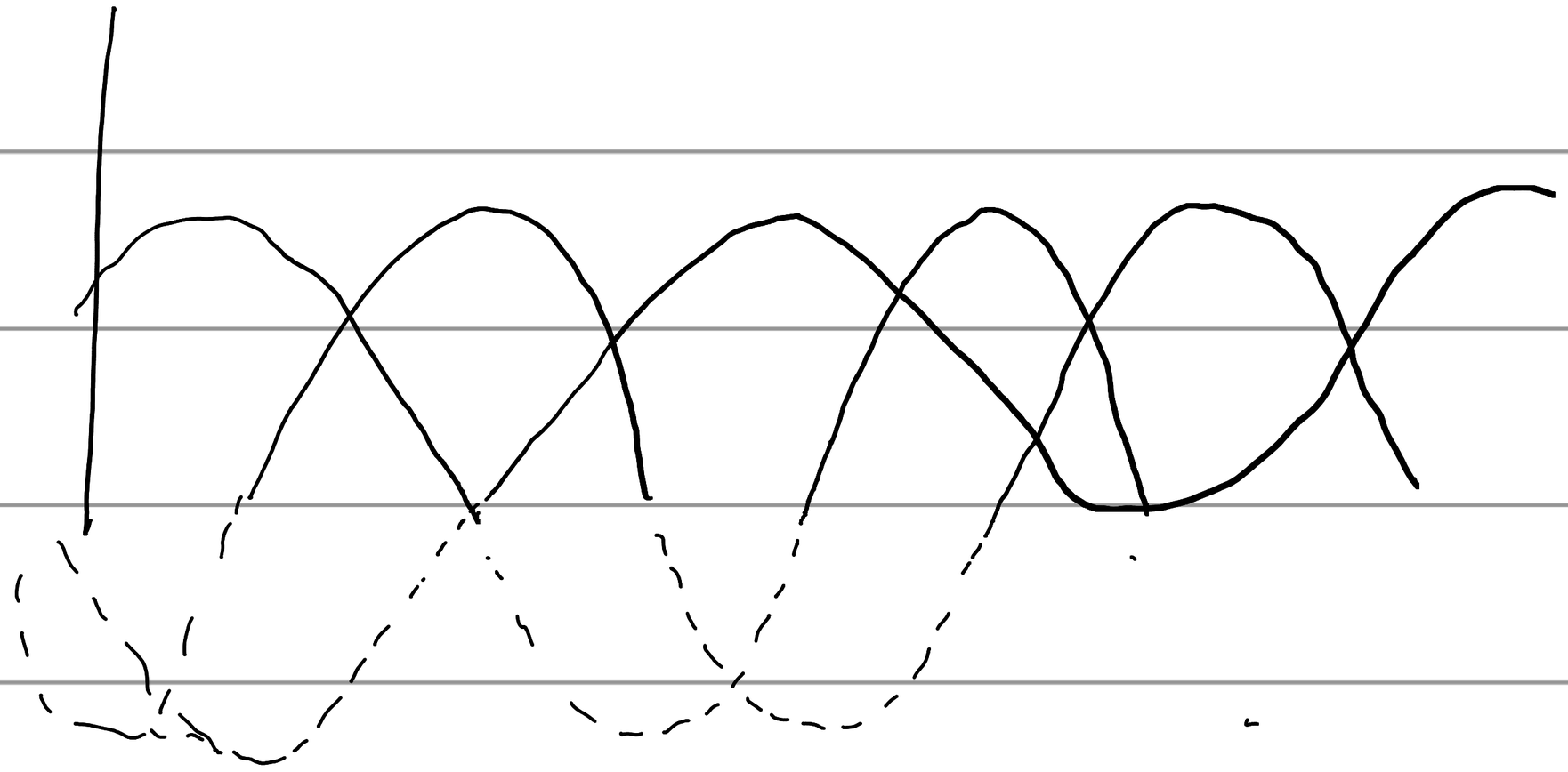
Implemented (6) & registered scripts thinking about (7) need a new python library to look at zero crossings.

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.



12/6/22

Coded up graph software to plot data gathered by code + encoder library.

One channel C is lower than A & B, virtual neutral is almost non existant. Encoder is generating noise on ground.

Trying to estimate vn as $\frac{A+B+C}{3}$, attempting to normalise A, B & C by dividing by their means.

Doing a Kalman filter for normalised signals, & non-normalised signals minus $\frac{A+B+C}{3}$.

Signal is pretty crap, removing the encoder does not stop channel C being lower ... buy?

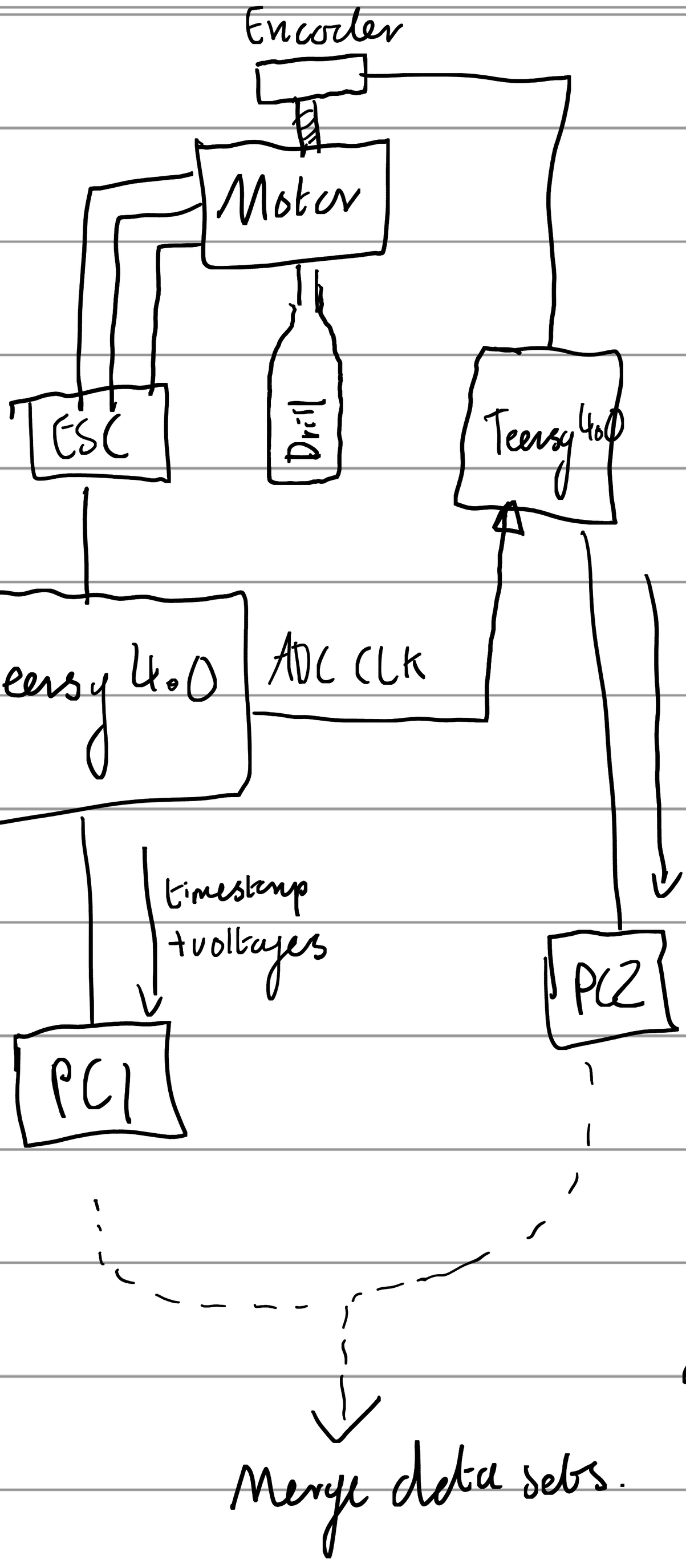
B/6/22

Trying old ADC code... it works! channels are balanced, VN is valid and without noise if the encoder is absent.

After many hours debugging... Channels are unbalanced when serial logging the parity bit! wierd! if only printing angle, A, B, C & VN it works, angle is blanks with encoder driver code disabled it has good signal.

Created a new grapher to show phase X; & VN vs time, phase X; - VN;, VN; & kalmen of phase X; - VN;, good results! Just needs the angle data & can have a go at the 1D kernetic

Conclusion... need a second Teensy 4.0 to read the encoder & ensure grounds are not shared. Use one Teensy to trigger the other. Merge the dataset with software.



Need to order an additional Teensy 4.0

14/6/22

Thinking about processing dataset, presume
angle, d, b, c all run is the incoming data
Kalmen filter all channels.

Next kernel $[-, -1, +]$ rising kernel

$[+, -1, -]$ falling kernel

len kernel = 5

	0	1	2	3	4		101	102	103
						...	x	x	x
more complex	−	−	0	+	+		−	0	+

104 len data

Need atleast 5 data points so first attempt at idx = 2 & final attempt at idx

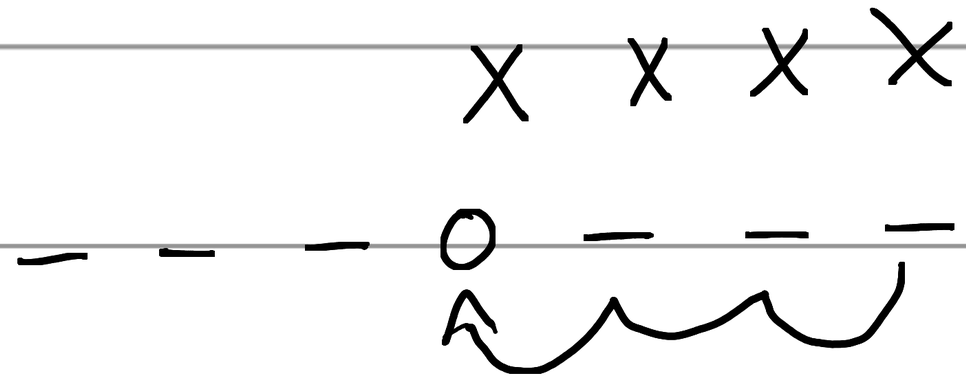
idx > 1

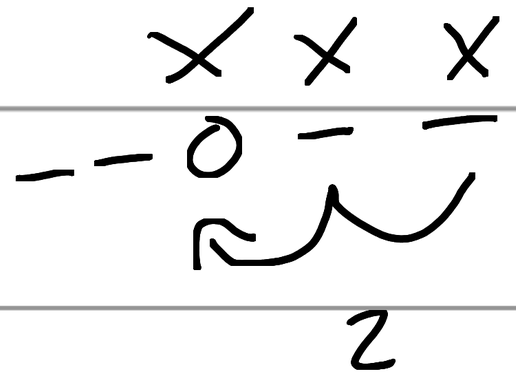
idx > 101

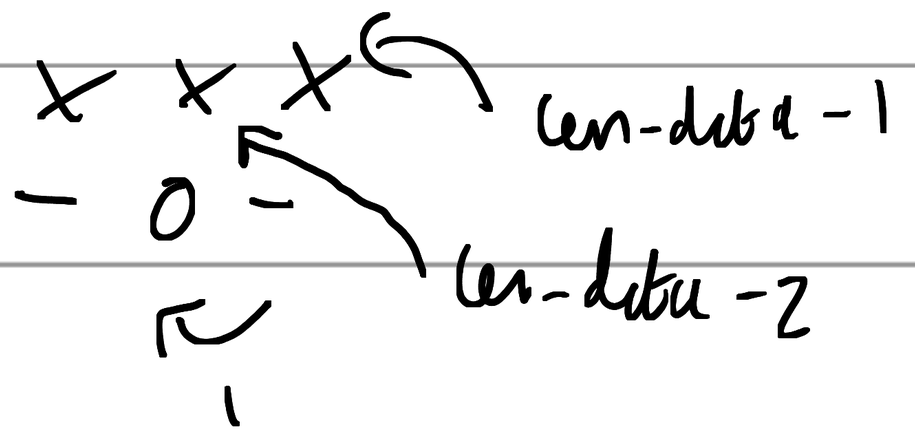
idx > len_data − 2

3	idx ≥ 0	idx ≤ (len-data − 2
5	idx ≥ 1	idx ≤ (len-data − 3)
7	idx ≥ 2	idx ≤ (len-data − 4)

len-data −4







3	1	$= \frac{\text{len\_kernel} - 1}{2}$
5	2	
7	3	

len_kernel	$\frac{\text{len\_kernel} - 1}{2}$ Ⓐ	start	end
3	1	idx ≥ Ⓐ − 1	idx ≤ len_data − (Ⓐ + 1)
5	2	idx ≥ Ⓐ − 1	idx ≤ len_data − (Ⓐ + 1)
7	3	idx ≥ Ⓐ − 1	idx ≤ len_data − (Ⓐ + 1)

0 1 2 3 4 5 6 7 8

x x − − +/− + + x x . . . .

we are here

idx = 4

Kernel size = 5

midpoint = 2

2 3 4 5 6 data space

[0 1 2 3 4] Kernel space

Kernel input idx=0

data_input idx = 4

idx (0, 4) + midpoint/2 ≟ 2

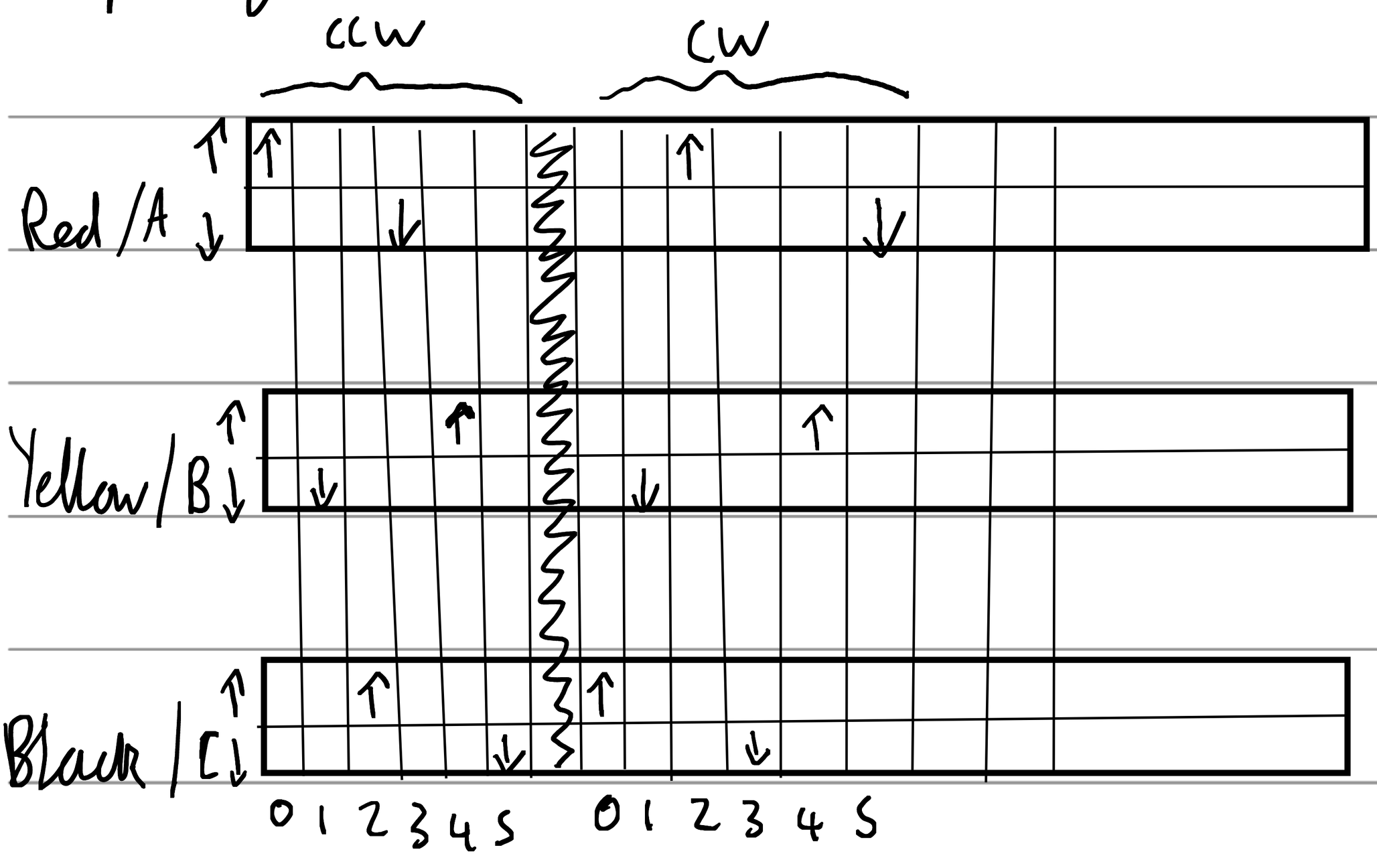
k idx	(k idx + D idx) − k_m
0	(0+4) − 2 = 2
1	(1+4) − 2 = 3
2	(2+4) − 2 = 4
3	

wrote draft of fc kernel code.

Updated scripts & improved graphing. Estimating the angle as proportional to the time + white noise, have to wait for additional micro controller to get real data.

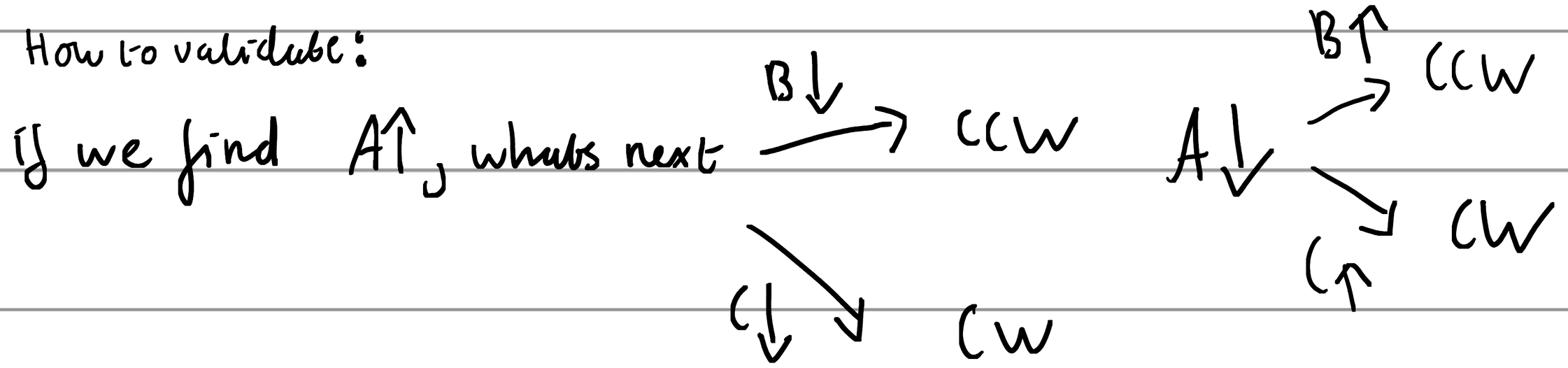
Thinking about zero cross detection

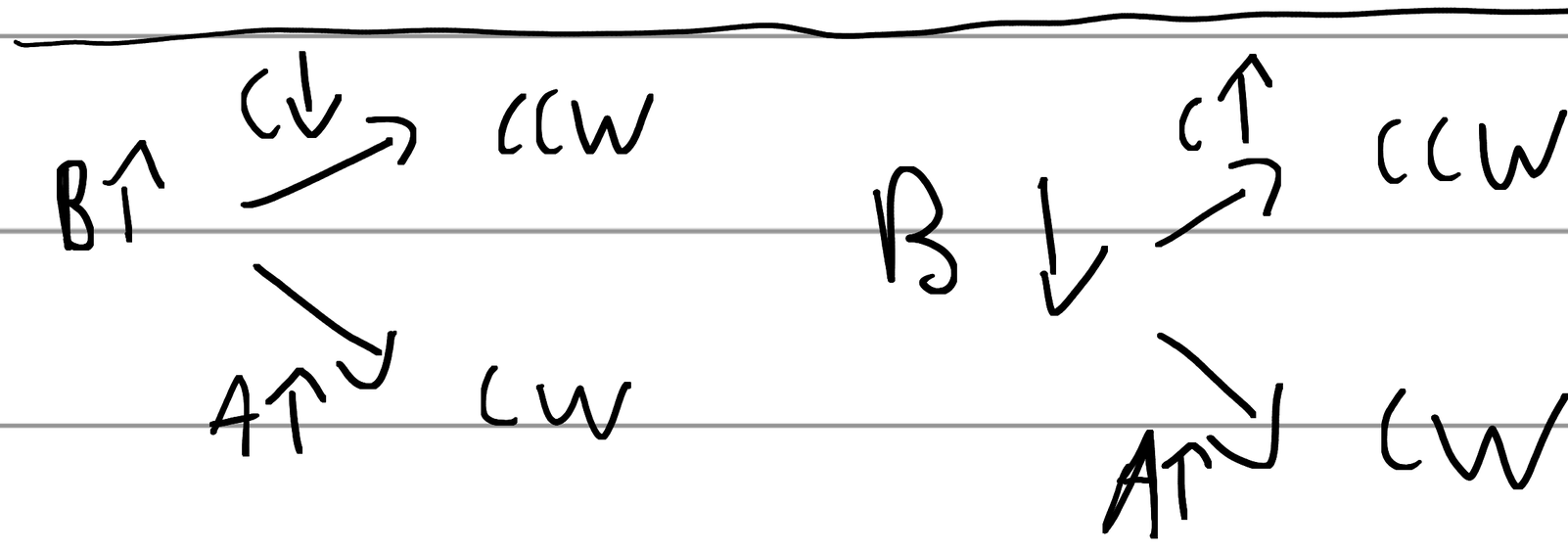
Complete sequence

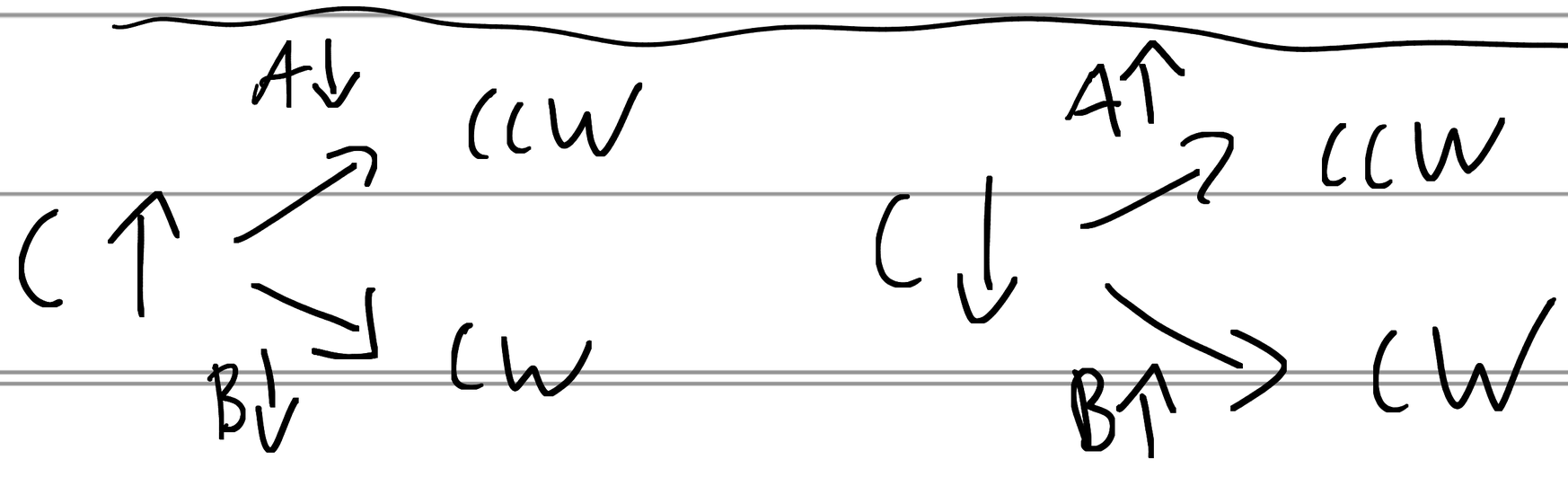


How to validate:

If we find $A\uparrow$, what's next







So we have 6 crossings per electrical cycle & $\frac{\text{Poles}}{2}$

electrical cycles per mechanical cycle. $6 \times \frac{12}{2} = 36$ total crossings.

$36 / 3 = 12$ crossings per phase

We have 3 phases A', B', C' & two channels

+ve cross, -ve cross

$12 / 2 = 6$ crossings per phase crossing channel

$$\text{channel\_ZC\_cluster\_number} = \frac{\text{Poles}}{2}$$

15/6/22

Refactored ZC detection code, re-written in pyspark in case data is very large.

Cleaned up script / file names.

Clustering code is next... chunk this data? Sanity check that ZC is in good place vs raw $X_i - V_N$.

After clustering finalise the ZC angles, validate the sequency cw or ccw. Chunk & display results.

16/6/22

Just cleaning up code today... chores. Had a look at spark on windows, might need to install spark within Linux VM. WSL pyspark cannot use my existing network cluster as the VM has an IP seperate from the host & this network node cannot reply. Bridge networking? & add WSL to bring up.

17/6/22

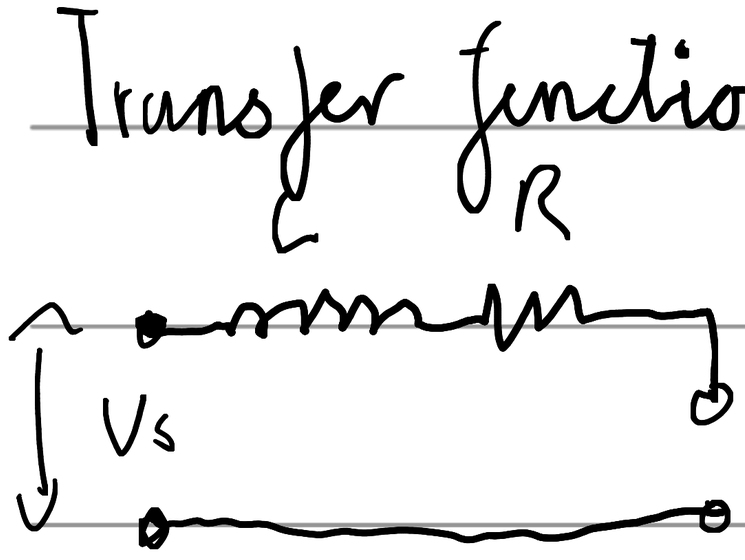
PID control parameters

Parameter	Rise time	Overshoot	setting time	steady state error
K_P	↓	↑	small change	↓
K_I	↓	↑	↑	eliminate
K_D	small change	↓	↓	small change

The goal is to obtain a just rise, with minimal overshoot & almost no steady state error.

1) obtain open loop response
2) Add K_p to improve rise time.
3) Add Kd to improve overshoot percentage
4) Add K_I to reduce steady state error.

Transfer function



back emf $e = k_e w$

$$e = -Ri - L\frac{di}{dt} + Vs \quad *'$$

rotor
Inertia load (J) Angular velocity w (Ti torques)

TL mechanical torque

Te electrical torque

$$J\frac{dw}{dt} = \sum_{i}^{N} T_i \quad *'$$

friction constant

$$Te = k_f w + J\frac{dw}{dt} + T_L$$

$*'$ what is $\frac{di}{dt}$ $L\frac{dI}{dt} = -e - Ri + Vs$

$$\frac{dI}{dt} = \frac{1}{L}\left(-e - Ri + Vs\right)$$

$$= \frac{1}{L}\left(-k_e w - Ri + Vs\right)$$

$$\frac{di}{dt} = -\frac{iR}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s$$

$$\frac{\partial w}{\partial t} = \frac{1}{J}\sum T_i \qquad T_e = K_t w$$

$$\sum T_i = T_e + T_L$$

$$T_e = K_f w + J\frac{dw}{dt} + T_L \qquad \frac{1}{J}(T_e - K_f w - T_L) = \frac{dw}{dt}$$

$$+J\frac{dw}{dt} = -K_f w - T_L + T_e$$

$$\frac{dw}{dt} = \frac{1}{J}(-K_f w - T_L + T_e) \qquad T_e = K_t w$$

$$= \frac{1}{J}(-K_f w - T_L + K_t w)$$

$$K_t w = 2T_L - K_{tw} + ik$$

?

Don't follow derivation but the result is

$$\frac{d\omega_m}{dt} = i\frac{k_t}{J} - \frac{k_f \omega_m}{J} + \frac{1}{J} T_L$$

$$\frac{di}{dt} = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L} V_s$$

$\int u\,dv = uv - \int v\,du$

Laplace transformations — by parts identity

$$\int \frac{df(t)}{dt} = \int_0^\infty \frac{df(t)}{dt} \underbrace{e^{-st}}_{u} dt$$

$u' \therefore v = f(t)$

$\frac{du}{dt} = -se^{-st}$

$du = -se^{-st}\,dt$

$dv = \frac{df(t)}{dt}\,dt$

$$\int_0^\infty e^{-st} \frac{df(t)}{dt}\,dt = \left[e^{-st} f(t)\right]_0^\infty - \int_0^\infty f(t)(-s)e^{-st}\,dt$$

$$= \left[e^{-st} \cdot f(\infty) - e^{0t} f(0)\right] + s\int_0^\infty f(t)e^{-st}\,dt$$

Laplace again (1st derivative)

$$\mathcal{L}\left\{\frac{df(t)}{dt}\right\} = \int_{0^-}^{\infty} \underbrace{e^{-st}}_{u} \frac{df(t)}{dt}\, dt = F(s)$$

(F(s): complex)

$$u = e^{-st} \qquad \frac{du}{dt} = -se^{-st} \qquad du = -se^{-st}\,dt$$

$$\frac{dv}{dt} = \frac{df(t)}{dt} \qquad v = f(t)$$

$$\int uv' = uv - \int u'v$$

$$dv = df(t)$$

$$\int_0^{\infty} e^{-st} \frac{df(t)}{dt}\, dt = f(t)\, e^{-st}\Big|_0^{\infty} - \int_0^{\infty} f(t)(-s)e^{-st}\, dt$$

$$= \left[f(\infty)\underbrace{e^{-\infty}}_{0} - f(0^-)\underbrace{e^{0}}_{1}\right] + s\underbrace{\int_0^{\infty} f(t)e^{-st}\, dt}$$

$$F(s) = \mathcal{L}\{f(t)\}$$

$$\mathcal{L}\{f'(t)\} = sF(s) - f(0) \quad \text{NEAT!}$$

$$\int_0^{\infty} e^{-st} \frac{dw}{dt} dt \qquad \int u\,dv = uv - \int v\,du$$

$$\frac{dw_m}{dt} = i\frac{k_t}{J} - \frac{k_f\,\omega_m}{J} + \frac{1}{J} T_L$$

$$u' = \frac{dw}{dt} \qquad dv = \frac{df(t)}{dt}\,dt \qquad v = f(t)$$

$$u = w\,?$$

$$\int_0^{\infty} e^{-st} \frac{dw}{dt} dt = \left[e^{-st} w(t) \right]_0^{\infty}$$

$$- \int_0^{\infty} w(t)\,(-s)\,e^{-st}\,dt$$

$$w(0) + s \int_0^{\infty} w(t)\, e^{-st}\, dt = \mathcal{L}\left\{ \frac{dw}{dt} \right\}$$

$$= w(0) + s \int_0^{\infty} \frac{dw}{dt} e^{-st} dt = w(0)$$

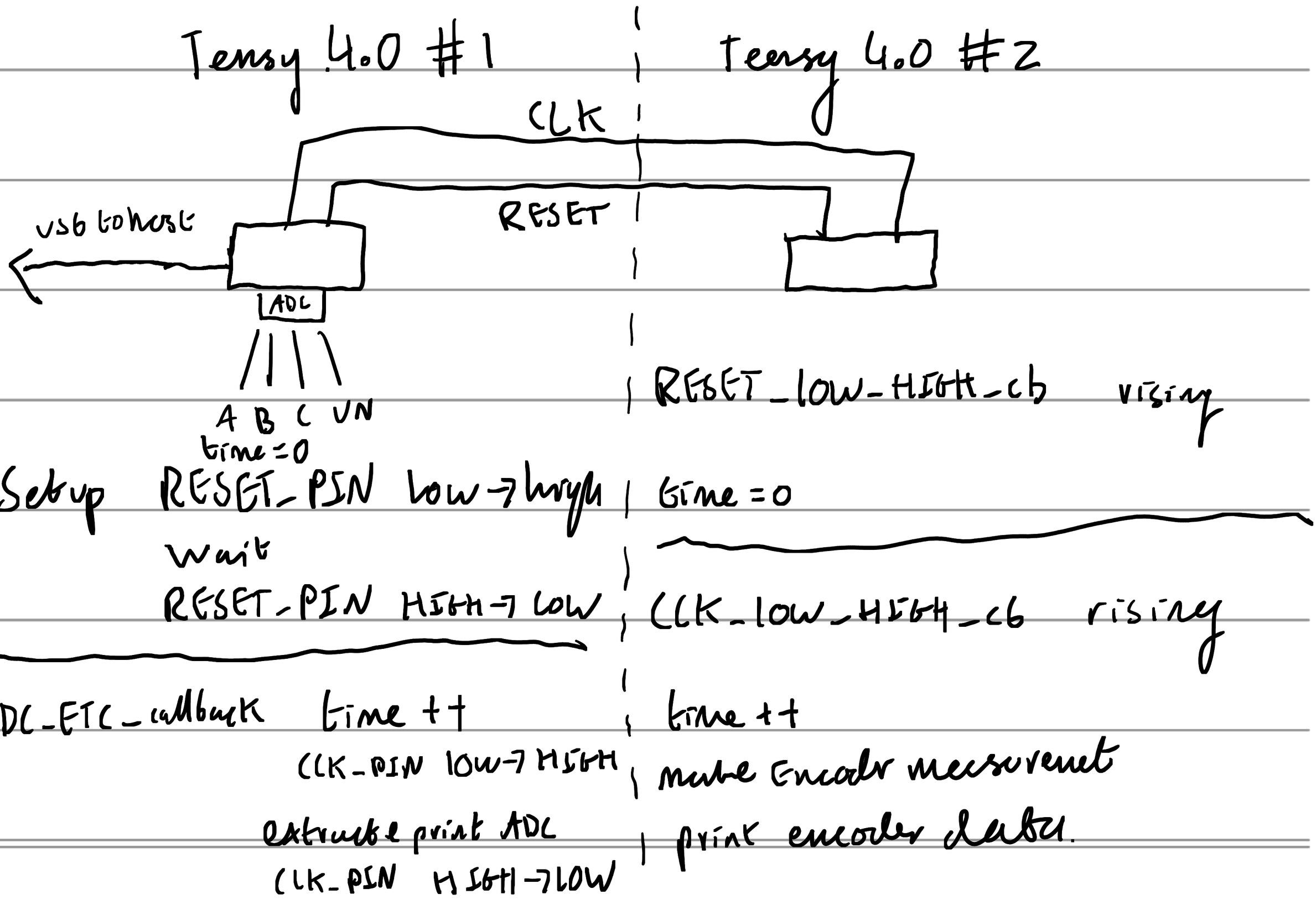
Inverse laplace

$$f(x) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^{st} F(s)\, ds$$

reminds me of the cauchy integral.

2016/22

New teensy is here. Need to remove encoder code from ADC! ADC by itself but in callback send a pulse to 2nd teensy, need to have an integer representing the tick of time. We want a 2nd channel to send a pulse so when the first teensys powers on send a pulse to reset the timer counter.



Drafts code done. Both teensy's flashed with correct hex.

Encoder is on /dev/tty ACM0
ADC on /dev/tty ACM1.

Procedure

→ Connect Encoder teensy

→ Start 2 instances of serial Plot

→ Bind first serial plot to /dev/tty ACM0

→ Disconnect ADC teensy

→ Connect ADC teensy

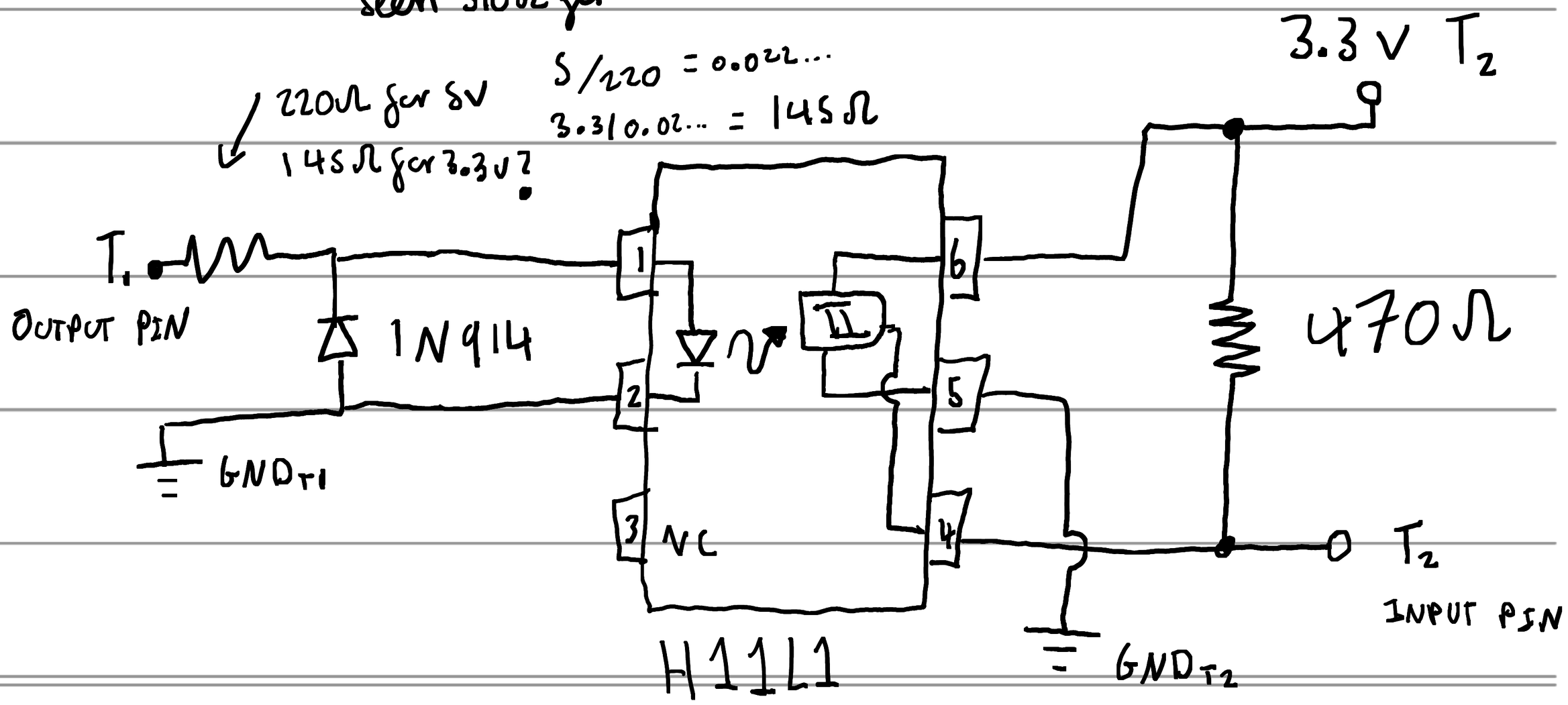
→ Bind 2nd serial plot to /dev/tty ACM0

Results & thoughts

Noise from Encoder still present in TDC measurements if both teensys are plugged into same laptop usb & grounds interact.

Noise is gone if using second laptop but digitally interrupting the slave teensy does not work. Need to trigger slave teensy without sharing grounds, galvanic isolation. Best option is probably opto-electrically.

Isolator circuit using H11L1



21/6/22

Shopping list

H11L1 opto-isolator x2

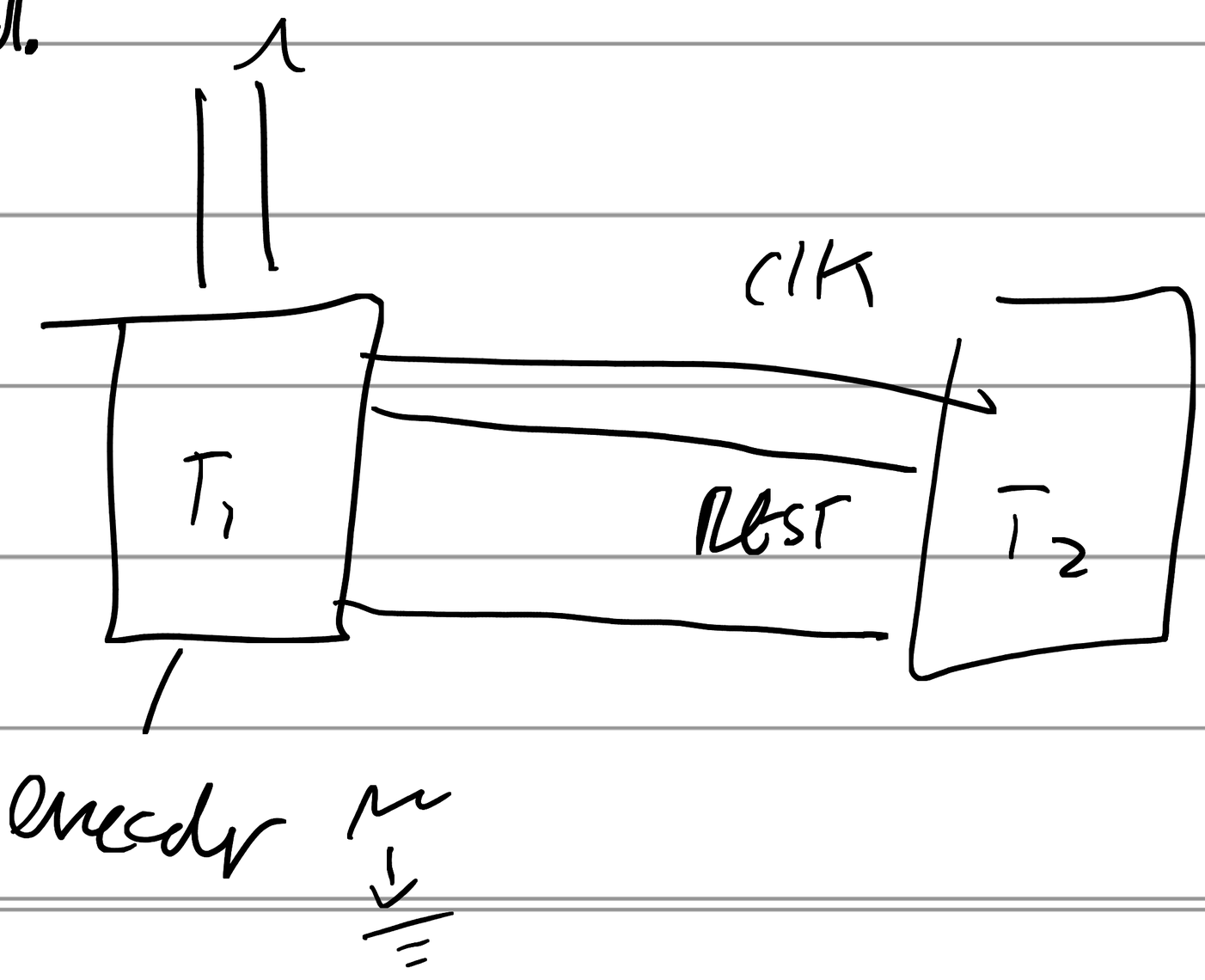
1N914 diode x2

470Ω resistor x2

220Ω (5v) or 145Ω (3.3v) x2

Attempted to fix some obvious bugs, ctr for time was ticking twice for each high signal to slave.

Shopping list ordered.



29/6/22

Optoelectronics have arrived. Have multiple resistors to try should try highest Ω resistors first & only if they don't work try lower resistances.

Until resistances are known then hardware is at risk. To avoid destroying microcontrollers the plan would be to get one teensy 4.0 communicating with itself. This could be a good way to pre-validate that the clk pulse is not firing erroneously via noise & introducing error & the time getting out of sync.

Timer 1 GPT ~ 60,000 Hz

↳ SET CLK Low (cpto inverts input)

wait ~ 4ns

SET CLK High

tick T_1

ISR Digital Interrupt

CLK_IN Rising?

debounce?

tick T_2

comp T_1 & T_2

3, 4

Reset CLK

seen 310Ω for 3.3V

TX

220Ω for 5V

145Ω for 3.3V?

5/220 = 0.022...

3.3/0.02... = 145Ω

3.3 V

CLK_OUT

(8)

OUTPUT PIN

1N914

GND_T1

1

2

3 NC

6

5

4

H11L1

GND_T2

is this too low!?

1K

2.2K?

1K?

3, 4

CLK_IN

RX (7)

Rx2 7 pin Tx2 8 pin

V=iR

i = V/R

470Ω @ 3V

0.007A

↓

@5V 712Ω

17 TX4

16 Rx4

14 TX3

15 TX3

use with ADC

3.3 V

~7mA

Rx1 Tx1 0 1 used for pwm

Rx/TX5 21 20 } clear i think

RxTx6 25 24

Perhaps use Rx2 & Tx2 for CLK as maybe higher switching rate as used for serial port.

try 1k first

1K @ 3.3	0.0033
470 @ 3.3	0.0070

anode cathode

\+ −

\+ −

cathode

order to buy:

	220Ω	1kΩ ↓ danger teensy
danger opto ↓	220Ω	470Ω
	150Ω	470Ω

5/7/22

Tested single Teensy 4.0 opt-coupling to itself ok.
Pulse rate @ 120 kHz, pulse duration 3,000 ns / 3 µs.
Fairly stable seems to miss between 0 → 3 ticks, but remains stable, no determined the reason for missed clocks perhaps some kind of initialization issue.

I picked 220Ω & 1kΩ resistors.

Now attempt to mount the teensy on a breadboard.

Moving on to multiple Teensy setup.

Final setup (calibration)