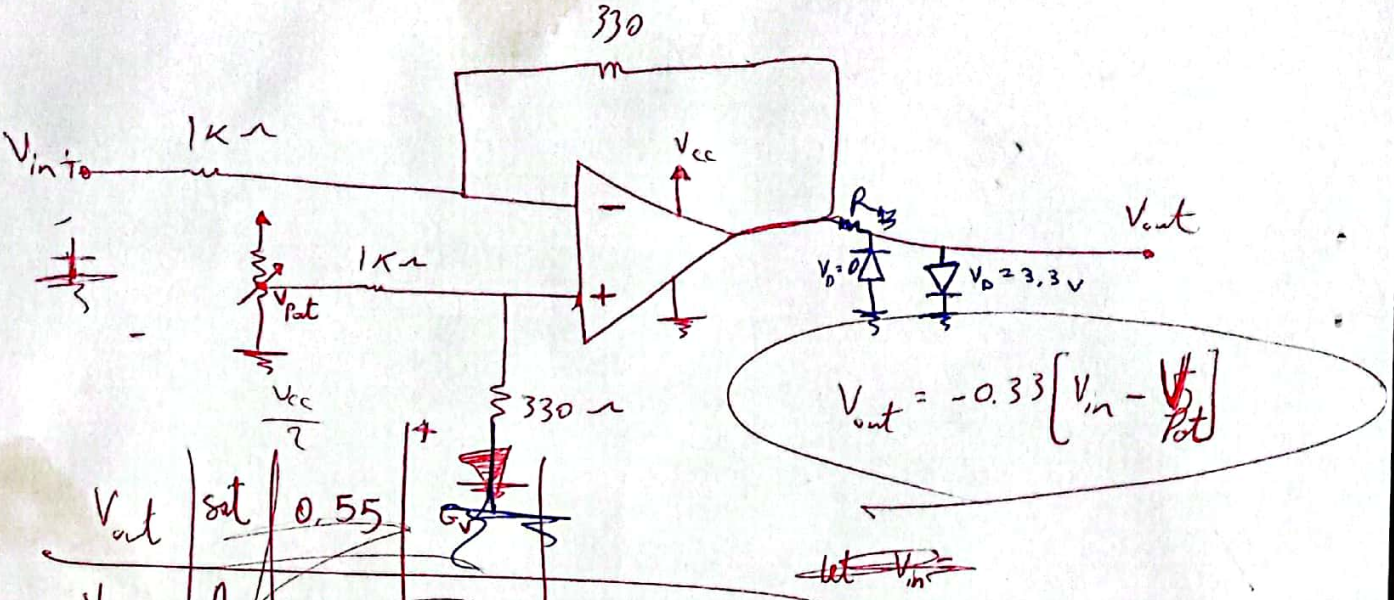


$$V_{out} = -0.33\left[V_{in} - V_{pot}\right]$$

V_{out}	sat	0,55	
V_{pot}	0	0	0,21
V_{in}	0	0,55	0,55

V_{out}	0	0,14
V_{pot}	0	0,28
V_{in}	0	0

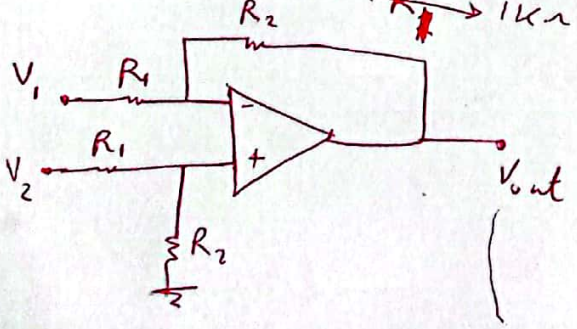
$$V_{out} = -A_V\left[V_1 - V_2\right]$$

$A_V = 0,33$; $V_1 = [-5:5]$; $V_2 = 5V$

$[-10:0]$

$[0:10]$

$$A_V = \frac{R_2}{R_1} \quad R_2 \to 330\Omega \quad R_1 \to 1k\Omega$$

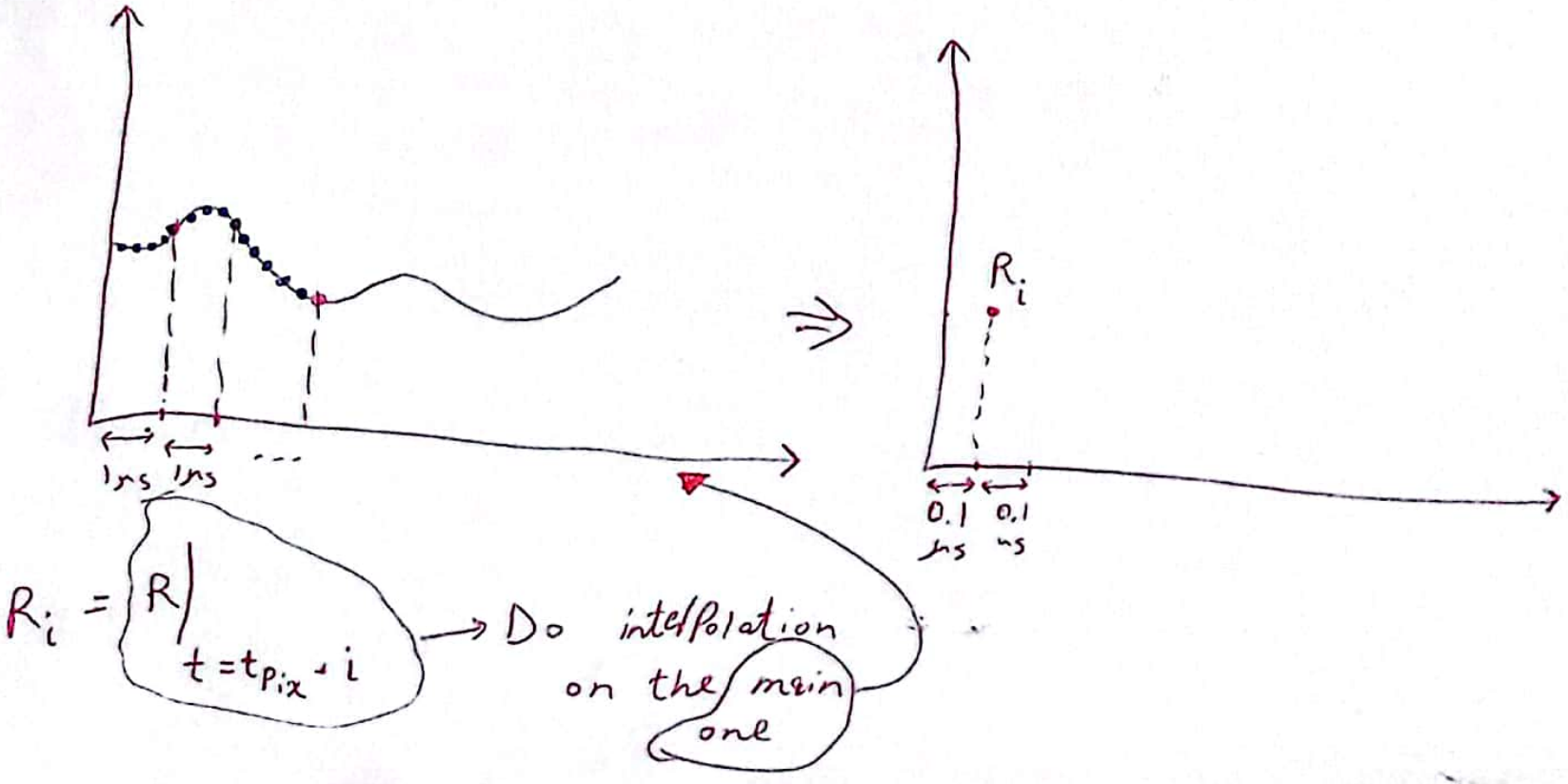


Sample Prediction/Interpolation

~~if N_Samples * t_{Pix}~~

* if $t_{Pix} < 1ns$ (min sample + Conv. time)

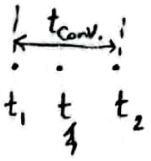
* Then : # number of samples to be Predicted = N_sample (Per frame)



Sample Prediction (For ~~signals~~ $t_{Pix} < t_{ConV}$) → 1 µs

* Capture samples at maximum rate (Continious mode)
* Get the interPolated "sampleBuffer[]" using the function:
  "interPolate(sampleBuffer, t_{Pix})"
* The function "interPolate()" works as follows:

```
u16 sampleBufferInterPolated[N_samples];
for (u8 i=0; i< N_samples ; i++)
{
  t = i * t_pix ;
  t1 = int(t/t_conV) * t_conV.
  t2 = t1 + t_conV.
  S1 = sampleBuffer[i];
  S2 = sampleBuffer[i+1];
```



$$\because \frac{S - S_1}{t - t_1} = \frac{+S_2 + S_1}{t_2 - t_1} \Rightarrow S = \underbrace{(S_2 - S_1)}_{\text{MUST Be "signed"}} \cdot \frac{t - t_1}{t_2 - t_1} + S_1$$

```
  sampleBufferInterPolated[i] = S ;
}
```

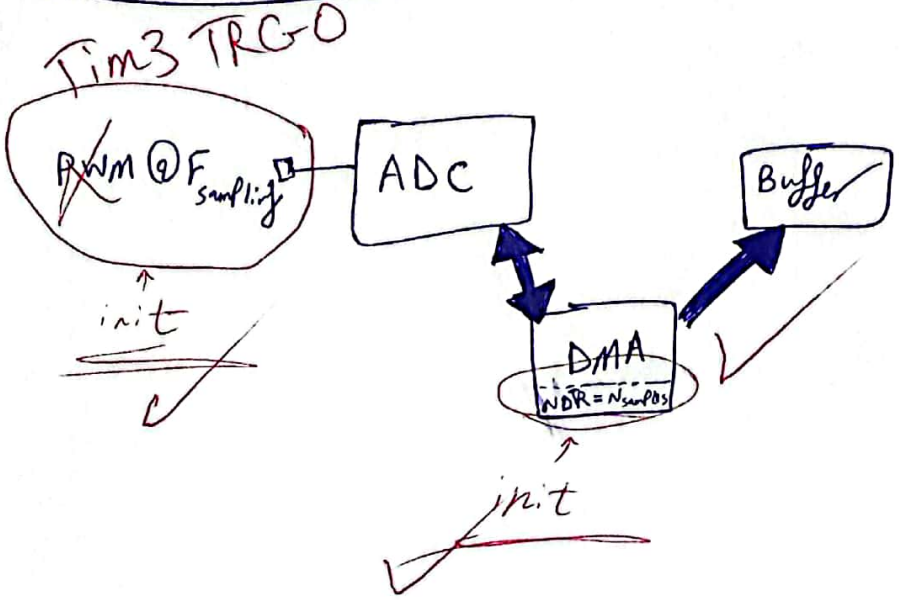
How to sync?

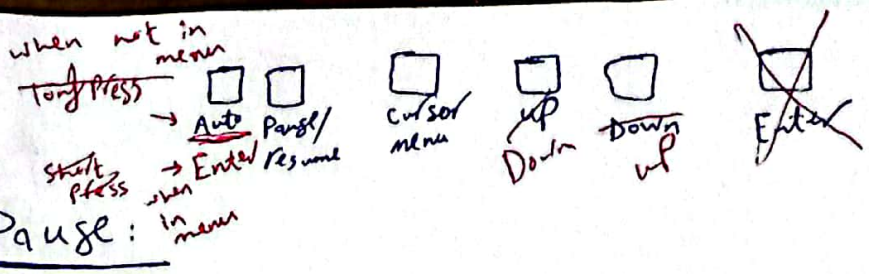
Wait for rising edge with a timeout();

Enable DMA to copy from ADC to buffer;

Wait for DMA TC;

Draw;





Pause:

On Press, EXTI ISR is executed.

~~if Previously Paused,~~ Toggle "Pause" flag.

In main loop drawing, let the code segment of taking new array of samples be conditioned with that flag. Drawing is still on, but for the same values.

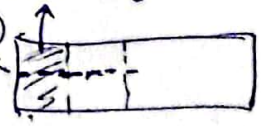
Cursors

* When "Cursor" button is Pressed:
  open Cursor menu, and select between → Add → V[2]' (if not yet) (~), t[2]' (~)
  → remove → V[2]' (if added) (~), t[2]' (~)
  → move → V[2]' (~), t[2]' (~)
  → select → t[2]' (~)

* Each cursor of the 4 has 2 values:
  [u8 Pos;
    b8 enabled;
  ]

* In main loop drawing, check "enabled" of each of the cursors and draw them (if enabled) according to their "Pos"

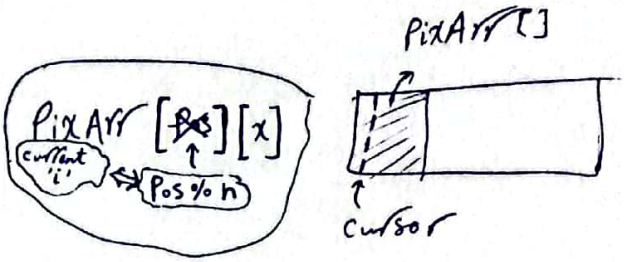
* Drawing Voltage Cursor: After done drawing "PixArr_i[]"
  - Draw a dashed line ②: PixArr[y][Pos] with color of "Cursor1_Color"



  - i.e.: ② PixArr[y][Pos_i] ; y ∈ {[0:dash_length], [2*dash_length:3*dash_length], ...}
  - Notice: -(n / dash_length) must be an integer.
    - last/max 'y' must be n - dash_length.

this will slow line drawing

* Drawing time cursor:



- Draw a dashed line @ only if (Pos == reqCount)
- $x \in \{[0:cursor\_length], [2*cursor\_length : 3*cursor\_length], \dots\}$
- Notice :- (128 / cursor_length) must be an integer.

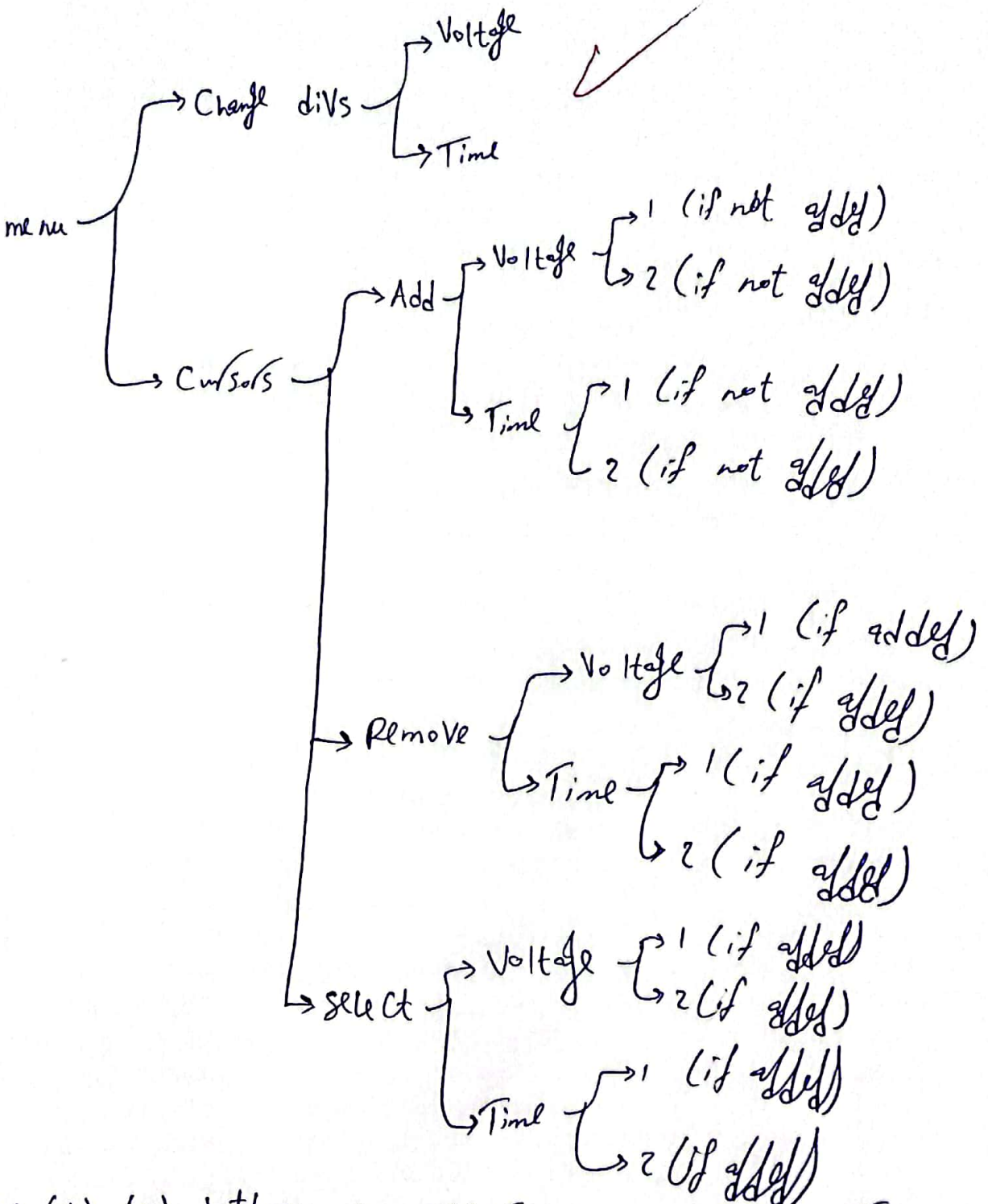
moving buttons

- on press, EXTI ISR is executed. increment or decrement "Pos" of the "selected" if "enabled"
- when in cursor menu, disable the interrupt of these buttons temprarily.

→ Notice that drawing a dashed time cursor will slow down line drawing, to avid this, make it a solid line from x=0 : x=127 by DMA

++
--

MENU



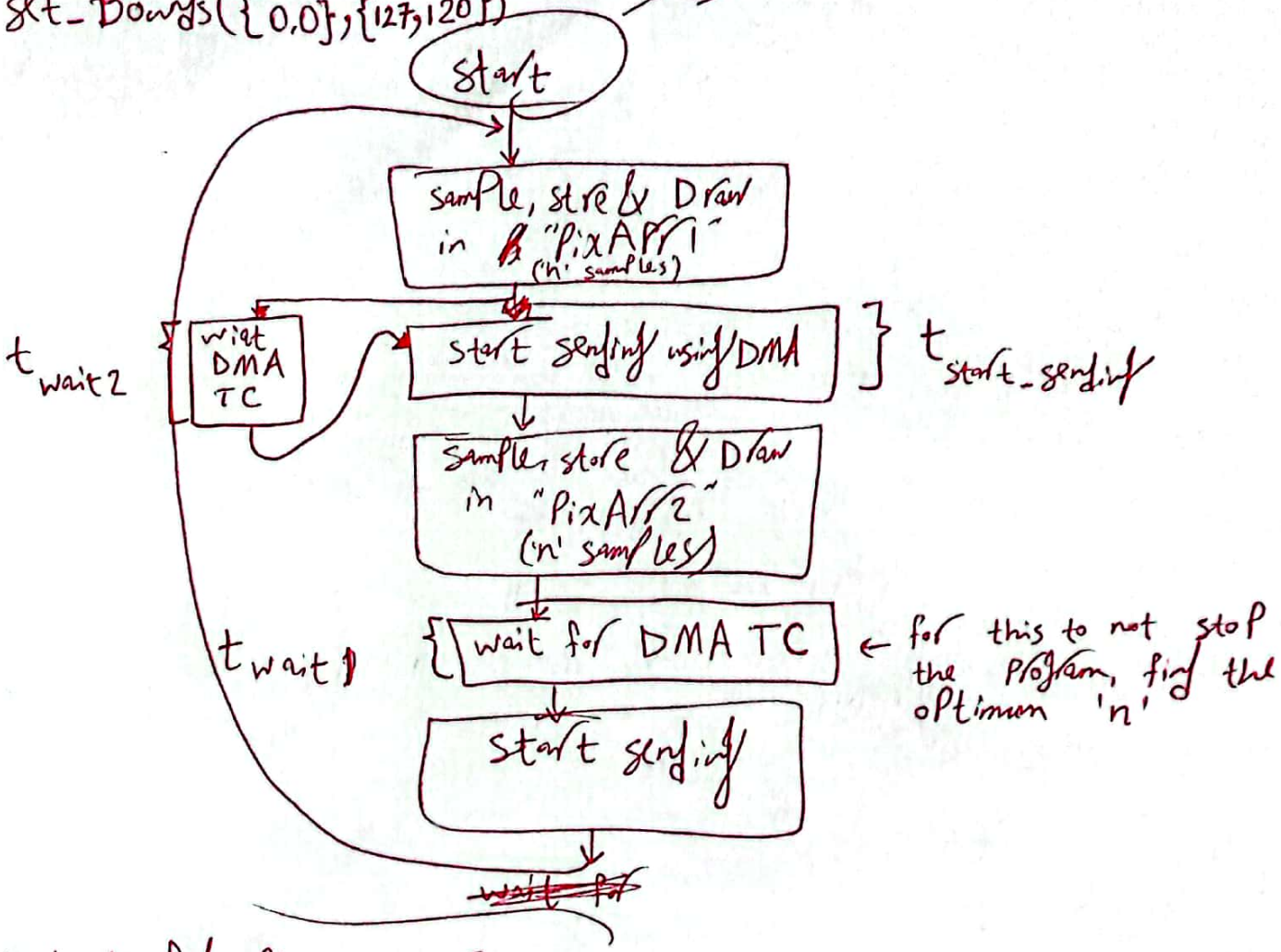
*The (+), (-) buttons can either move a selected cursor, or change a selected div, based on the last selected one of the two.

* Changing Voltage div may change input channel.

Pause
Cursor
-
+

use 3×DMA channels to draw single sample on PixArr

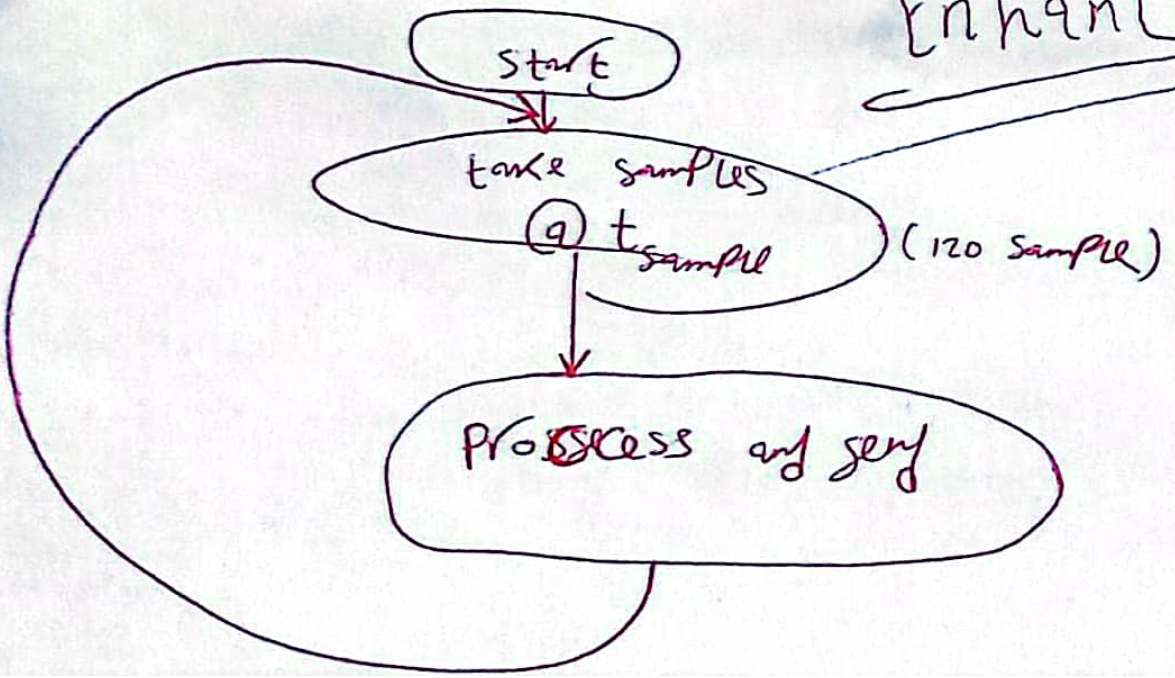
PixArr1[n][128];
PixArr2[n][128];
set_Bounds({0,0},{127,120})



How to find optimum 'n'?

Run the program from 'n' = 1 to 'n' = 20, each 'n' runs for 'm' iterations, store total time spent in starting sending and waiting DMA TC: $\sum^{m}(t_{start\_sending} + t_{wait1} + t_{wait2})$

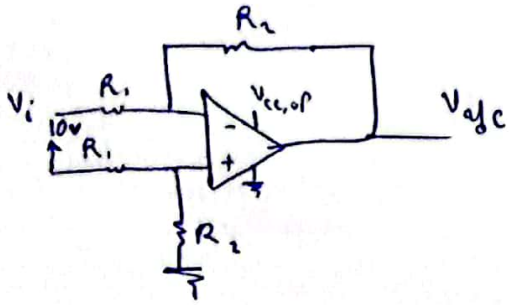
the 'n' with the lowest $\sum^{m}$ is the optimum to use.



Enhancement

Also: sync start of sampling with the rising edge of the signal to have better display

For $V_{in} \in (-10 : +10)$



$$V_{adc} = -0.165\,[V_i - 10]$$

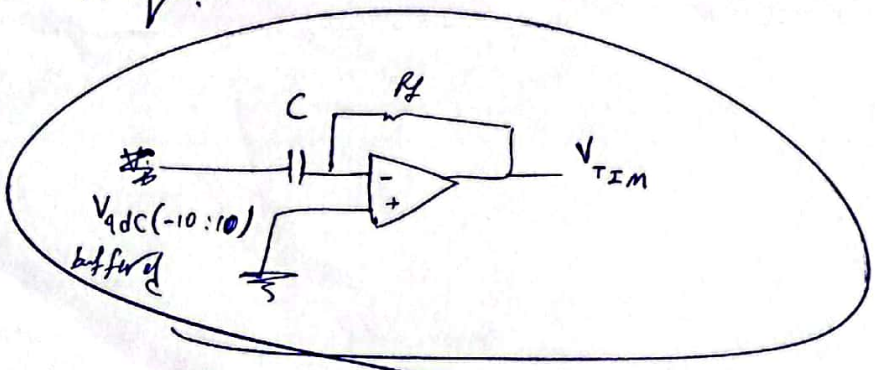
$$V_{out} = -\frac{R_2}{R_1}\left[V_i - 10\right]$$

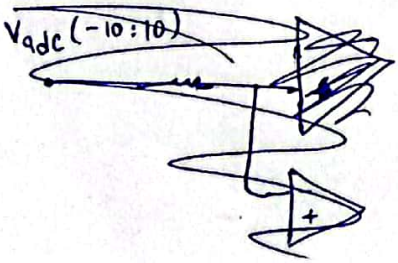
$$\frac{R_2}{R_1} = 0.165$$

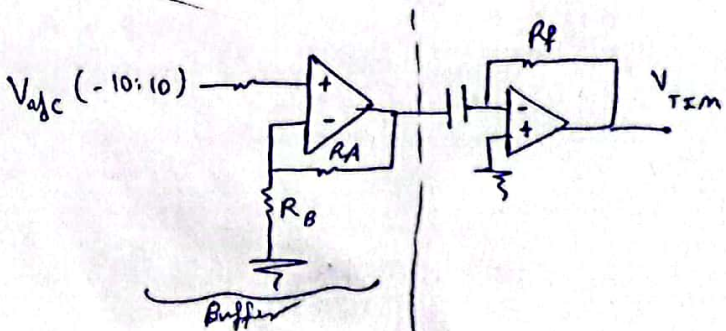
Notice: $V_{cc,op}$ is the V_{cc} value that make the op amp output 3,3 v @ saturation

(tested ~~to~~ and found: v)

For freq:







$$\frac{R_A}{R_B} \approx 0$$

$$V_{TIM} = -R_f C \frac{d\,V_{adc}(-10:10)}{dt}$$

* ACtion	time
Reading ADC_DR	2 ~ 3 ms
Reading register	
scroll (using your func.	5 ms
GPIO set Pin macro	less than 1µs
SPI set frame format macro	7 ms
SPI transmit using 'for loop' for 128* 16-bit	128 ms
SPI transmit using 'for loop' for 2*128* 8-bit	147 ms ⇒ ≃ 0,574 ms for 8-bits (±1)
set bords	2~3 ms
wrt Cmd	2~3 ms

write on ~~main block~~ Display main

Exellent work! ★

* Now devide drawing process to 3 Parts:

① scroll, readADC, set smallest & largest, set bound.s, draw from 0 to 'smallest'

② Draw from smallest to largest.

③ Draw from 'largest+1' to 127.

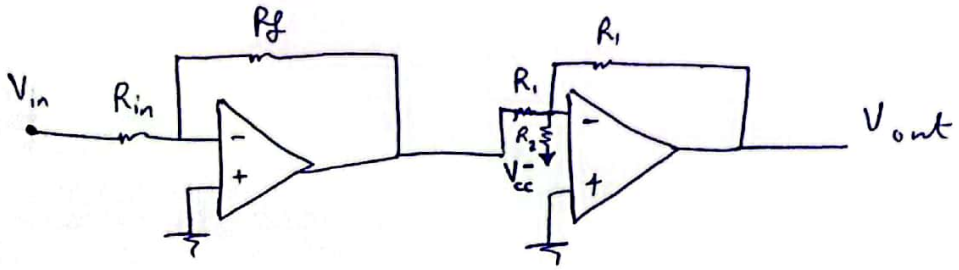
* As noticed, one line draw time is not deterministic. Hence, make the whole operation of drawing a line (the previous 3 Parts) timer triggered, to obtain Constant timing of them. Here's a Q.: why isn't it deter—. ??!

min: 3μs, avg: 50~60μs, max: 90μs

+ Note: Hal to change max adcReal to 125, and add 2 Pixels to any DMA transfer, so that the smallest possible number of data to be transferred (0) never happens, as it does not trigger TC interrupt, which causes flow fault logic.

* Note: To trigger start of operation, use basic timer, more eco.

Gain & offset



$$V_{out} = \frac{R_f}{R_{in}} V_{in} + \frac{R_1}{R_2} |V_{cc}^-|$$ → must be $(-V_c)$

When $V_{cc}^- = -12\ V$ & $R_1 = 3{,}3 // 3{,}3\ K\Omega \Rightarrow R_2 = 12K$

in order to obtain a $V_{offset} = +1{,}65$

~~* Let the maximum acc. be : 0.5 mV~~

~~$\Rightarrow \frac{1}{2^{12}-1}$~~

using: * $50\,mV_{PP}$, $10V\text{-}PP$, $R_{in} = 1K\Omega$

$$A_{V_1} = \frac{3{,}3}{50m} = 66 \qquad A_{V_2} = \frac{3{,}3}{10} = 0.33$$

$R_{f_1} = 66K\Omega$

$= 33K + 33K$

$R_{f_2} = 330\Omega$

"R_f" is changable via a 5-Pin relay.

$V_{out} = +(-1{,}65)$

$= +1{,}65$