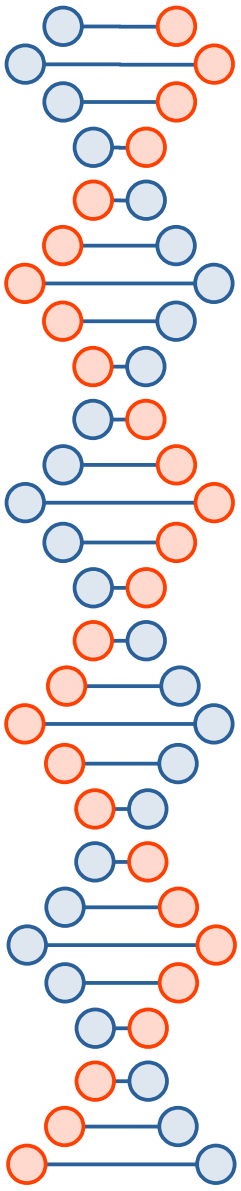


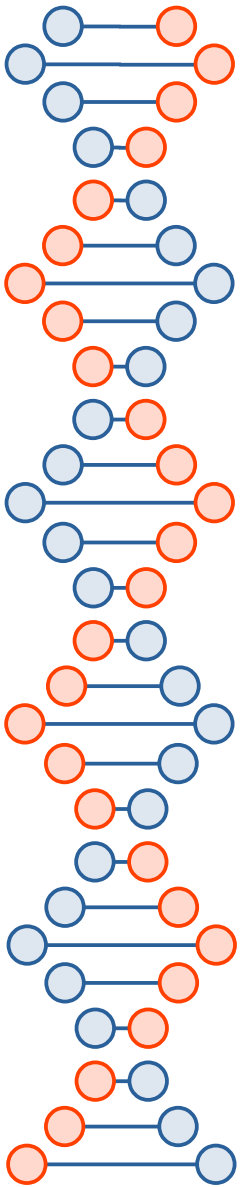
FTS

Fine Time Sync v0.1

Source code (GPLv3): https://github.com/abbbe/fts/

Author: Alexandre Bezroutchko

3/12/2025



My Use Case

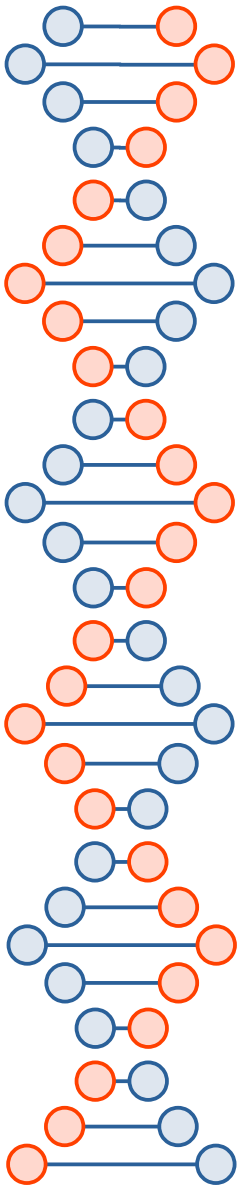
- Two synchronized wireless accelerometers
  - Sample sensors simultaneously
  - Timestamps from aligned clocks
  - Send data to a remote host for processing
- Looking for a library to
  - Sync local clock to a remote clock
  - Discipline the local timer to a remote timer

In the ideal world...

- Choose a role
- Set period
- Handle callbacks

No such thing, lets try to get close...

```c
#include "disciplined_timer.h"

#define MASTER 1 // set to 1 for master mode, 0 for slave mode

#define TIMER_PERIOD_US 500000 // 0.5 Hz

#define CYCLE_PULSE_TOGGLE_GPIO_NUM 7 // Hardware timer pulse
#define LED_GPIO GPIO_NUM_7 1         // LED pin to toggle

static int led_state;

void disciplined_timer_callback(int64_t cycle_count)
{
    // invoked every timer cycle, every TIMER_PERIOD_US microseconds
    // cycle_count: number of timer cycles since epoch

    // Toggle LED state
    led_state = !led_state;
    gpio_set_level(LED_GPIO, led_state);
}

void app_main(void)
{
    led_state = 1; // LED off (active low)
    pinMode(LED_GPIO, OUTPUT);

#if MASTER
    init_master_disciplined_timer(TIMER_PERIOD_US, disciplined_timer_callback,
                                  CYCLE_PULSE_TOGGLE_GPIO_NUM);
#else
    init_slave_disciplined_timer(TIMER_PERIOD_US, disciplined_timer_callback,
                                 CYCLE_PULSE_TOGGLE_GPIO_NUM);
#endif
}
```

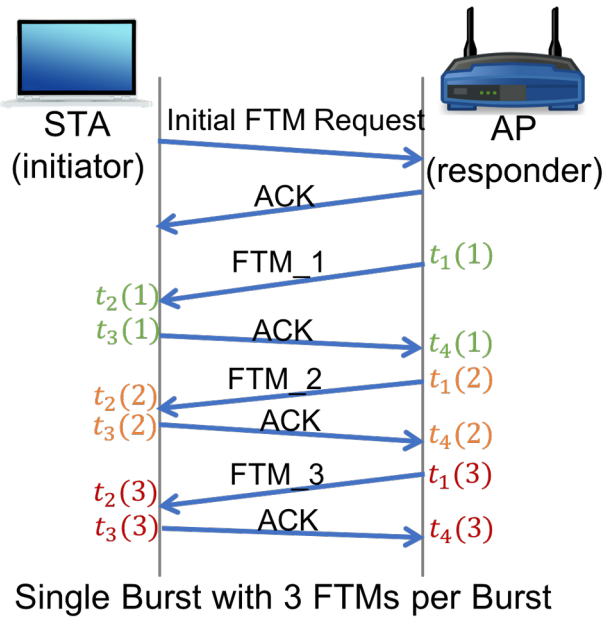


General Idea

- Mechanism to measure master/slave clocks offset
- Figure out a robust local clock (for master & slave)
- Build a Clock Relationship Model
  - Slave local clock <=> master local clock translation
- Discipline the local timer
  - Fine adjustments to timer phase and period
  - **TEZ to occur simultaneously on master and slave**

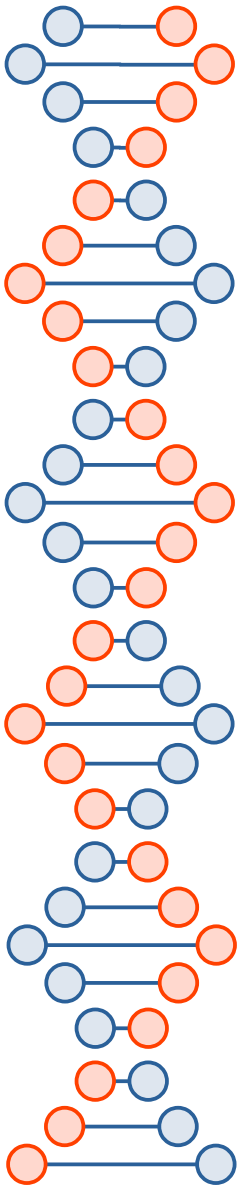
Fine Time Measurements

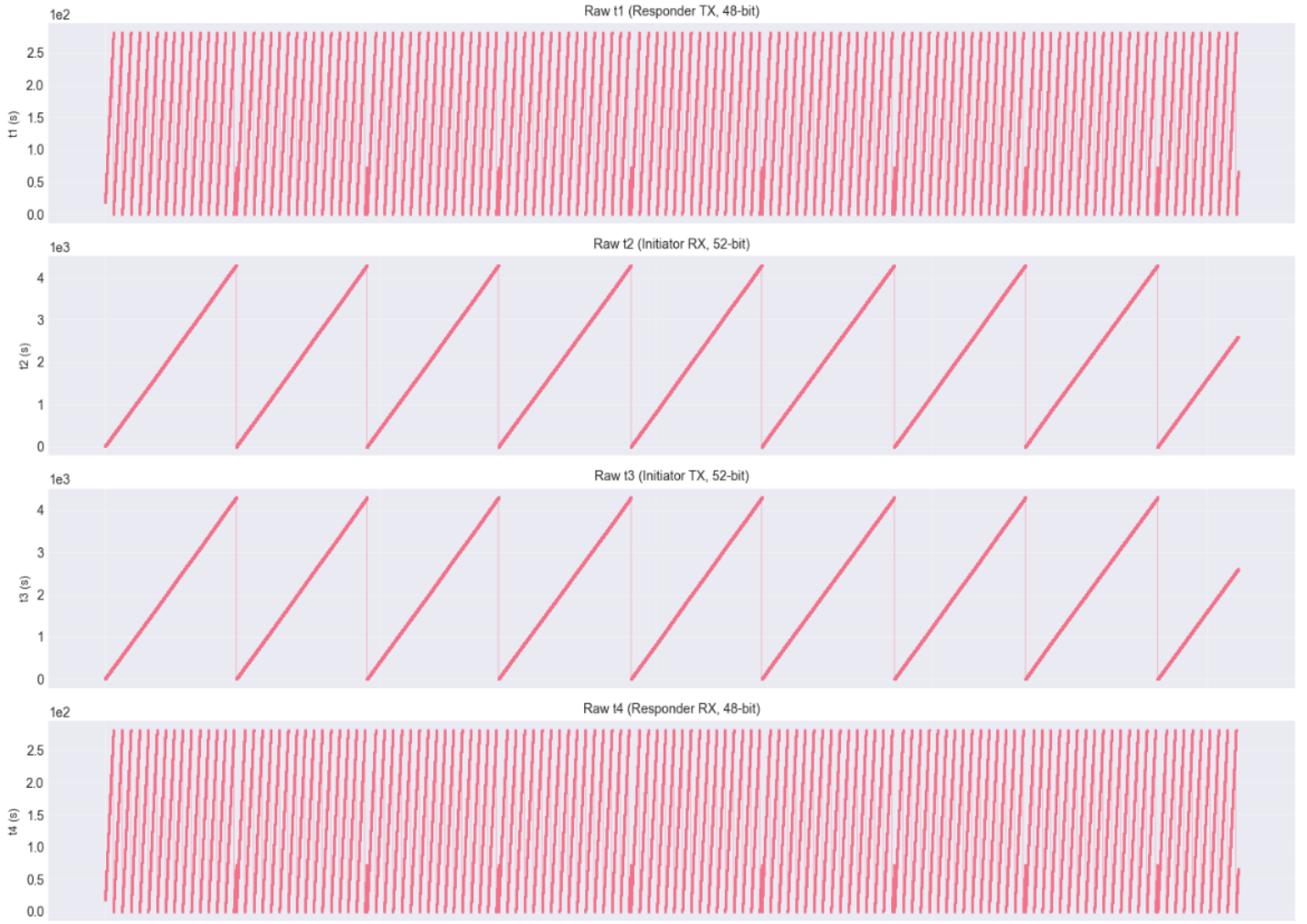
- Designed for ranging
- Initiator gets a bunch of reports:
  - T1 – responder sent the request
  - T2 – initiator got the request
  - T3 – initiator sent the response
  - T4 – responder got the response
- T1/T4 – responder's MAC clock
- T2/T3 – initiator's MAC clock

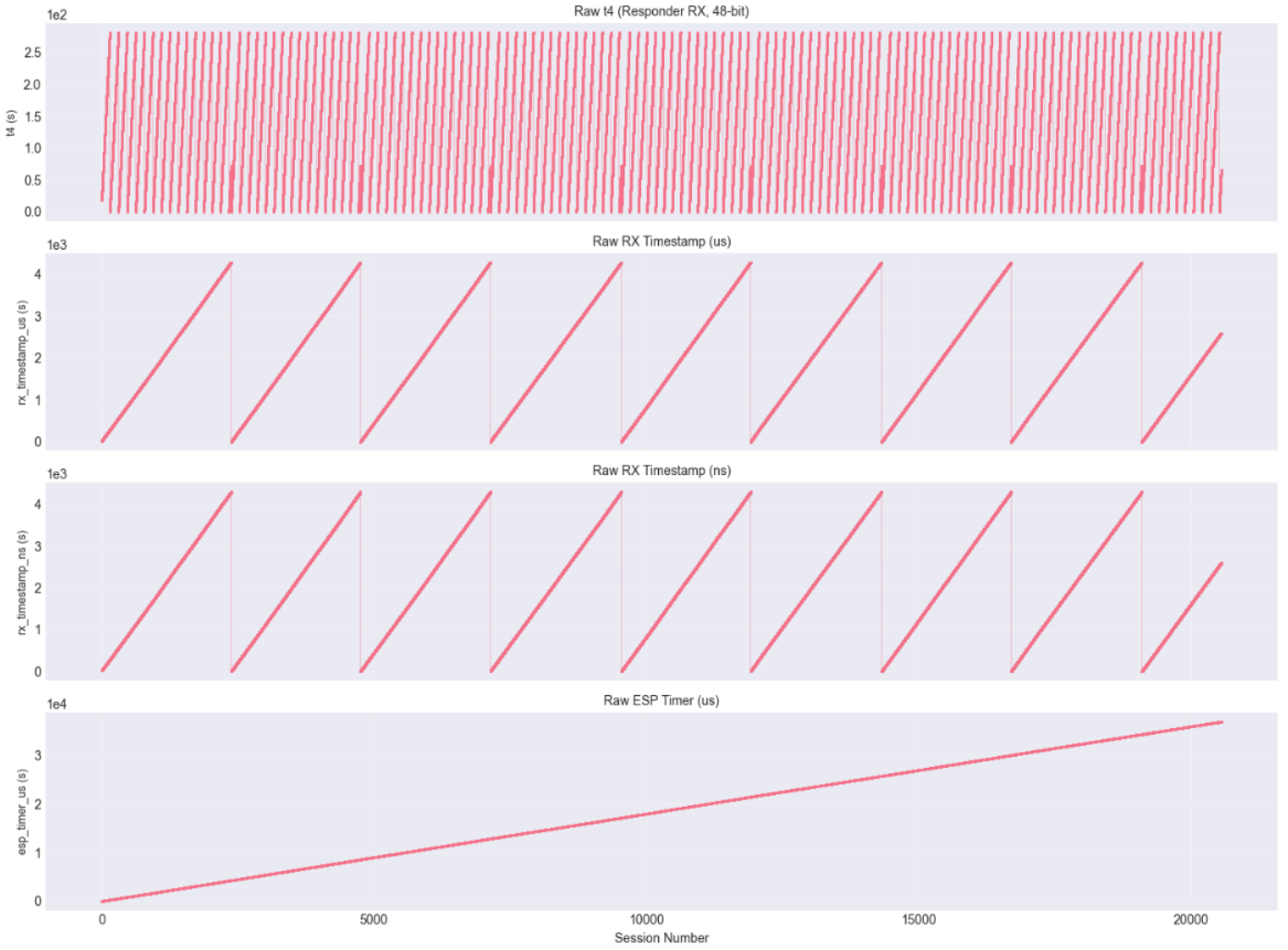


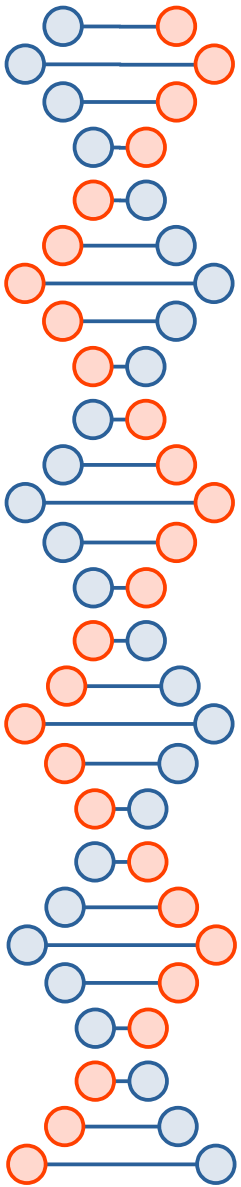
Single Burst with 3 FTMs per Burst

The diagram is from https://www.winlab.rutgers.edu/~gruteser/projects/ftm/index.htm



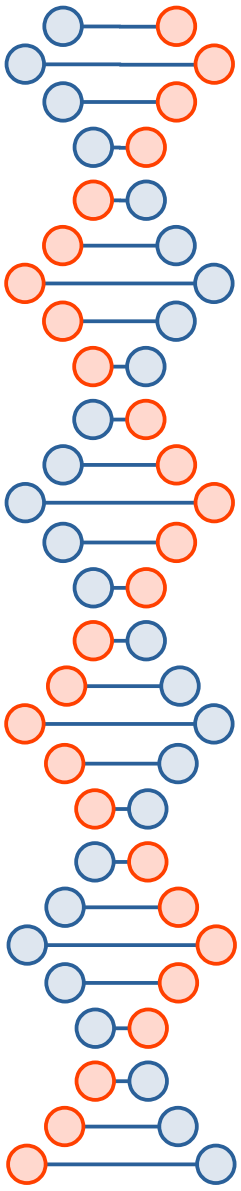

Raw t1 (Responder TX, 48-bit)
1e2
t1 (s)
2.5
2.0
1.5
1.0
0.5
0.0
Raw t2 (Initiator RX, 52-bit)
1e3
t2 (s)
4
3
2
1
0
Raw t3 (Initiator TX, 52-bit)
1e3
t3 (s)
4
3
2
1
0
Raw t4 (Responder RX, 48-bit)
1e2
t4 (s)
2.5
2.0
1.5
1.0
0.5
0.0


Raw t4 (Responder RX, 48-bit)
1e2
t4 (s)
Raw RX Timestamp (us)
1e3
rx_timestamp_us (s)
Raw RX Timestamp (ns)
1e3
rx_timestamp_ns (s)
Raw ESP Timer (us)
1e4
esp_timer_us (s)
Session Number



Disciplined Timer

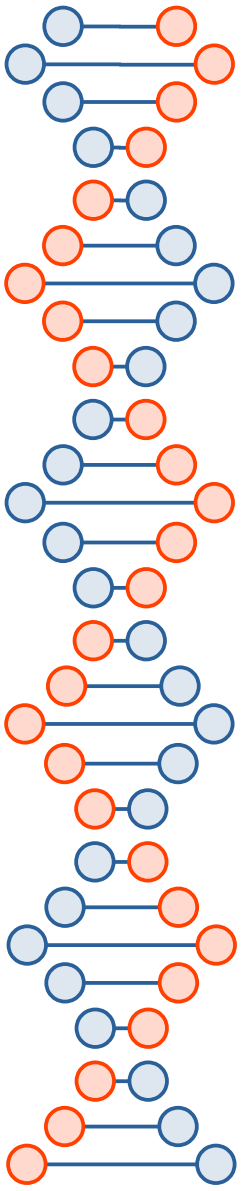
- We need a timer to
  - Poll sensors (@2 kHz for my use case)
  - Send pulses (for easy quality control with an oscilloscope)
- GPTimer:
  - 64bits, up to 40MHz, 25ns period, wide compatibility, no GPIO output
- MCPWM:
  - 16 bits, up to 160MHz, 8.5ns period, limited hw support (not on C3)
- Go for MCPWM @40MHz:
  - 25ns resolution, 2kHz TEZ frequency with 16000 period
  - GPIO output, but easy to port to GPTimer if pulses are not needed



Clocks

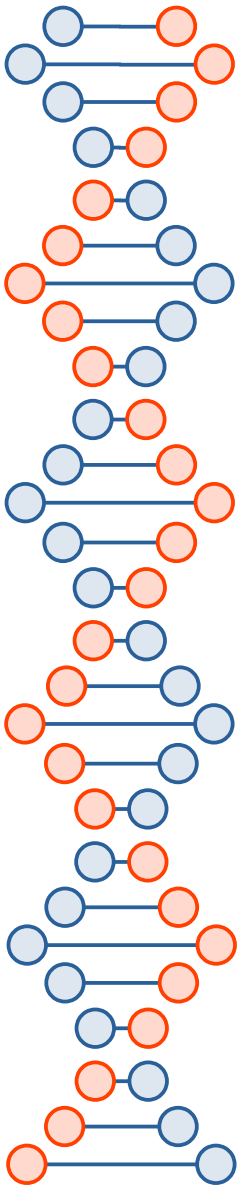
- MAC clock:
  - As of March 2025 exposed(*) by API as 32 bits @1us counter
  - Must be MSBs of some internal high-res MAC clock
  - Runs off same oscillator as MCPWM (different resolution, random offset)
- FTM timers: 1ps resolution
  - T2/T3 (initiator, local) – 52 bits, sampled off same high-res MAC clock
  - T1/T4 (responder) – 48 bits, sampled off remote high-res MAC clock
- MAC clock of the master = the master clock
- Cycle Counter = floor(master clock / TIMER_PERIOD)

(*) esp_wifi_internal_get_mac_clock_time(), see https://github.com/espressif/esp-idf/issues/15348. Also seen in beacons/TSF (1us resolution) and timestamps of sniffed packets: 1us (or 1ns with https://github.com/espressif/esp-idf/issues/9843 patch)



Robust Local Clock

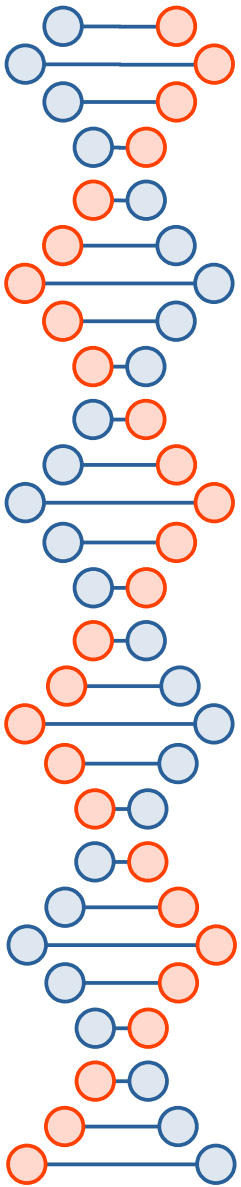
- Track overflows of MAC clock
  - 32 bits @1us (wrap period 2^32)
  - => **mac_clock_us**
- The disciplined timer (MCPWM, ticks at 25ns)
  - Clocked off same oscillator as MAC clock
  - Track all ticks and overflows, but never mess with tick counts
  - Ok to adjust period to alter phase and frequency of TEZ
- Sample MAC clock edges against the disciplined timer
  - => **mac_clock_at_timer_start_ns**
- The disciplined timer becomes local MAC clock with 25ns resolution!



Architecture

- FTM: Fine Time Measurement
  - Initialize Wifi
  - Initiate and collect FTM measurements
- CRM: Clock Relationship Model
  - Model master clock with linear regression, kind of:

    `master_time = slope * local_time + intercept`
- DTC: Disciplined Timer Controller
  - Handle CRM updates, produce timer alignment instructions
- DTR: Disciplined Timer Realtime
  - TEZ handler: consume alignment instructions from DTC, adjust period & phase
  - Pulse GPIO, invoke application callback (i.e. for trigger sensor sampling)



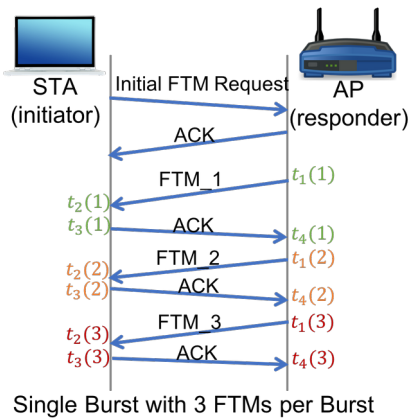
FTM → CRM → DTC ↔ DTR

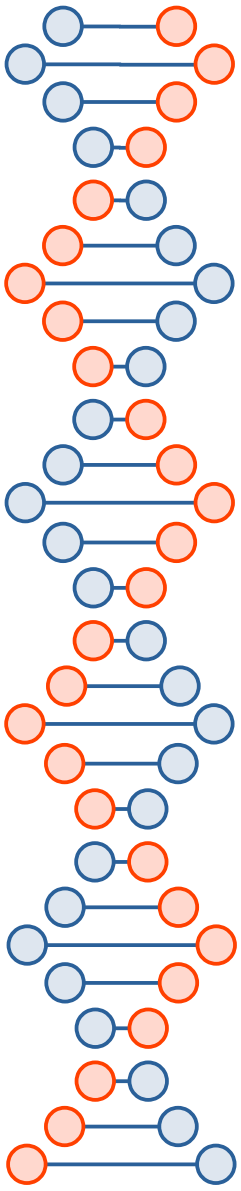
(Legend: **in**, **out**)

FTM

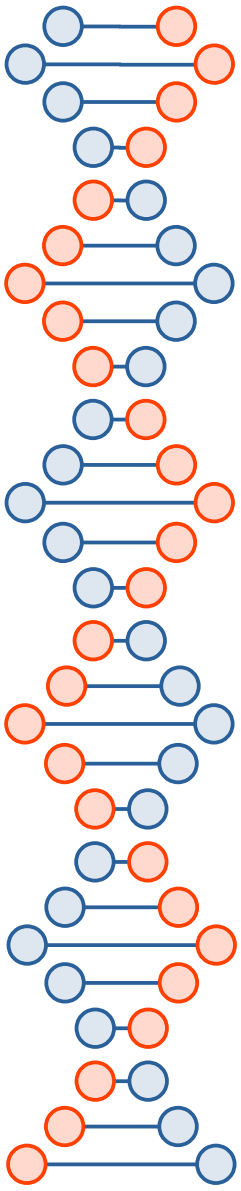
- Initiates FTM sessions every 1s
- FTM session = 32 reports, each report = 4 timers: **T1, T2, T3, T4**
- Tracks wraps and unwraps (a separate story): 4x 64bit @1ps T-counters
- RTT = (t4 - t1) - (t3 – t2)
- Local ref time:
  - **t2_unwrapped_ps**
- Corresponding remote time (assuming symmetric link):

  t1_unwrapped_ps + rtt/2
- Feed the list of (**local_ps, remote_ps**) to CRM module



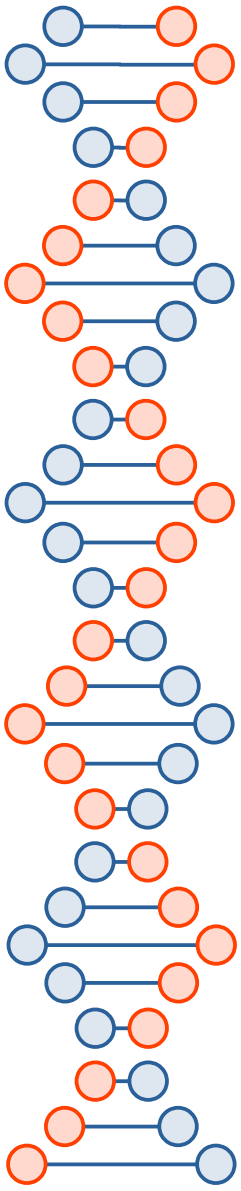


FTM → **CRM** → DTC ↔ DTR

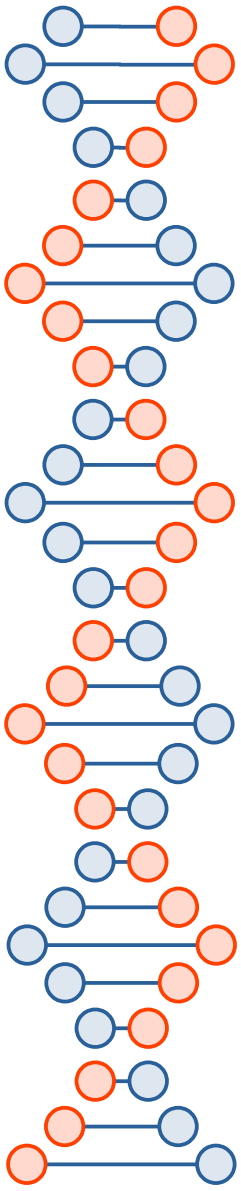


CRM: Clock Relationship Model

- Maintains a ring buffer of 100x recent FTM reports (**local_ps, remote_ps** pairs)
- Normal “y = a * x + b” linear regression is not enough
  - The intercept (b) is projected very far into the past (zero), poor accuracy
  - The slope (a) is very close to zero, 1ppm <=> 1.000001 (not much left for mantissa)
  - Inverse operation will require floating point division which is slow
- To improve accuracy and numeric stability:
  - Use centroids (mean_x, mean_y) instead of zero-intercept
  - Keep both forward and inverse slope to avoid any divisions later
  - Keep slope values minus 1 instead of actual slope values
- Feed the CRM model to DTC:
  - **crm_ref_local_ticks, crm_ref_remote_ticks, crm_slope_lr_m1, crm_slope_rl_m1**

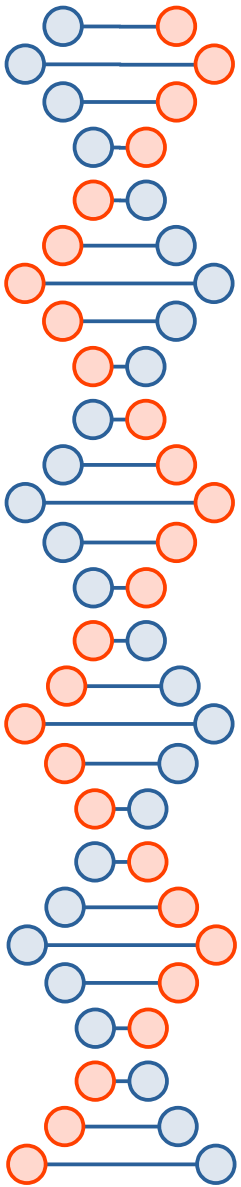


FTM → CRM → **DTC** ↔ DTR



DTC

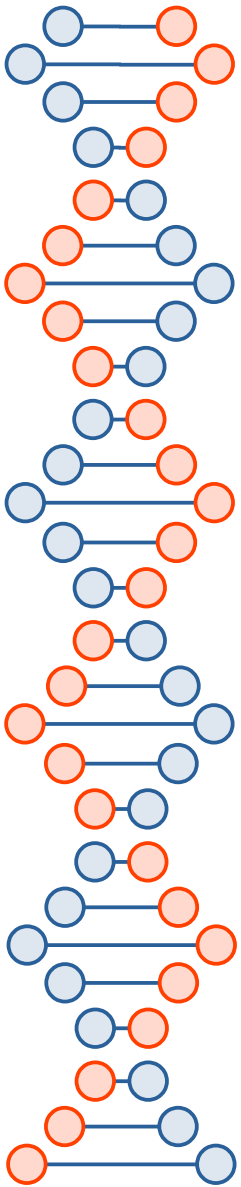
- Gets from CRM
  - **crm_ref_local_ticks, crm_ref_remote_ticks**
  - **crm_slope_lr_m1, crm_slope_rl_m1**
- Waits for next TEZ to get timer_base_ticks
  - get **timer_base_ticks** (= the time of last TEZ, in local timer ticks)
- Can calculate
  - Slave / master timer ticks @ last TEZ
  - Master / **slave timer ticks @ next TEZ**
- The latter can be directly used to adjust slave's TEZ phase
- To adjust TEZ period – just multiply by slope



DTC: Timer ticks @ last TEZ

- Regression: slave => master
  - delta_local_ticks =

    timer_base_ticks – crm_ref_local_ticks
  - delta_remote_ticks =
    - delta_local_ticks + delta_local_ticks * crm_slope_rl_m1
  - **remote_ticks** =
    - crm_ref_local_ticks + delta_remote_ticks



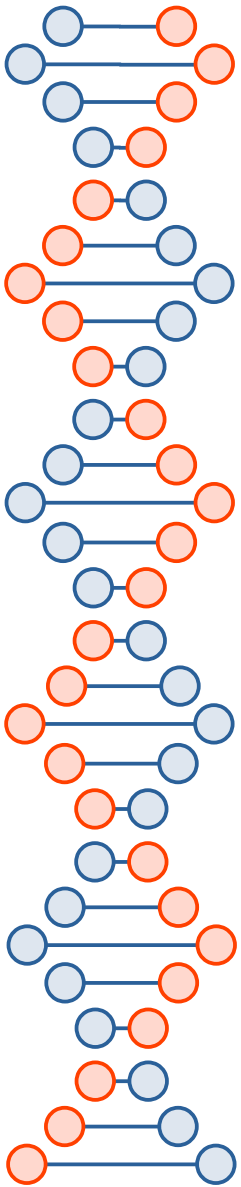
DTC: Align to Master Cycle

- Figure out the master cycle we are aligning to
  - **aligned_cycle_count** =

    (**remote_ticks** + TIMER_PERIOD_TICKS/2)

    / TIMER_PERIOD_TICKS + 2
  - **aligned_remote_ticks** =

    aligned_cycle_count * TIMER_PERIOD_TICKS
- +2 because:
  - remote_ticks - corresponds to the cycle in progress now
  - DTR runs alignment a cycle after, to controls the next cycle after next



DTC: Timer ticks @ next TEZ

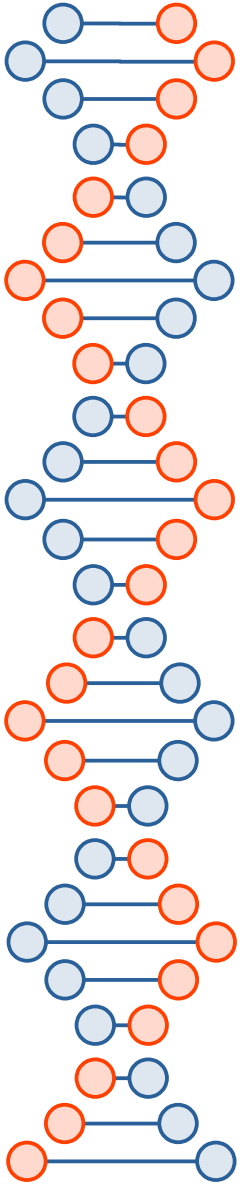
- Inverse Regression: master => slave
  - aligned_delta_remote_ticks =

    aligned_remote_ticks – crm_ref_remote_ticks
  - aligned_delta_local_ticks =

    aligned_delta_remote_ticks + aligned_delta_remote_ticks * crm_slope_lr_m1
  - **aligned_local_ticks** =

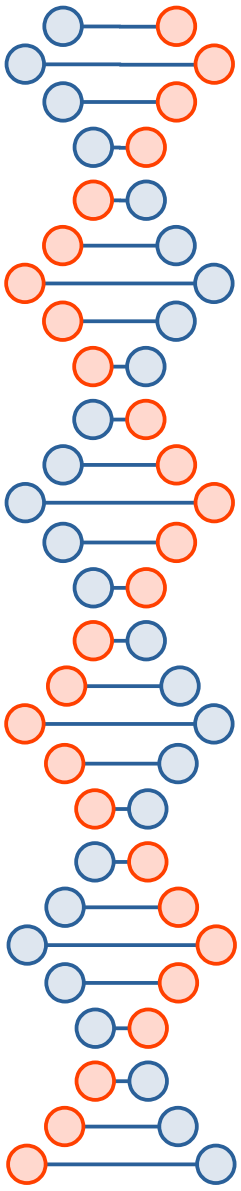
    crm_ref_local_ticks + aligned_delta_local_ticks
- If math ok:

  aligned_local_ticks ~ timer_base_ticks + 2 * TIMER_PERIOD_TICKS

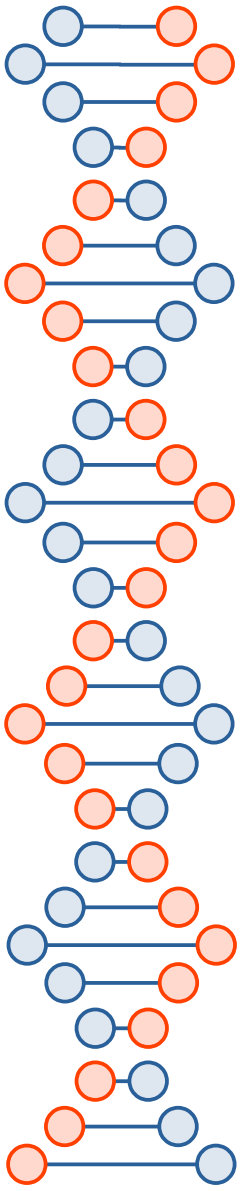


DTC: Alignment Instructions

- **aligned_cycle_counter** - cycle number for that TEZ
- **aligned_local_ticks** – the next aligned TEZ
- **aligned_base_period_fp16 - n**ew timer period, in FP16
  - master_base_period_fp16 = (TIMER_PERIOD_TICKS * 65536.0)
  - aligned_base_period_fp16 = master_base_period_fp16 + master_base_period_fp16 * **crm_slope_lr_m1**



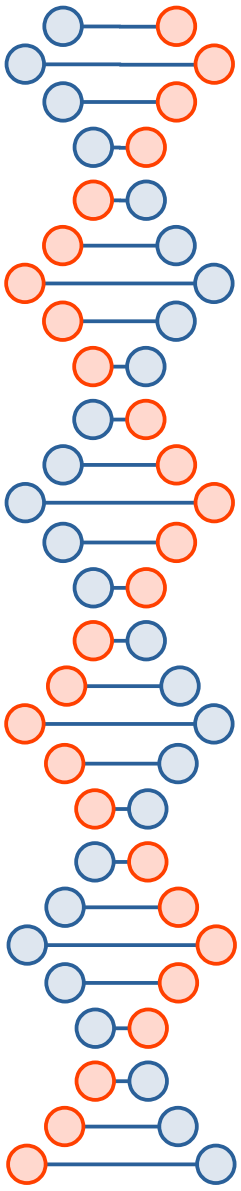
$$\text{FTM} \rightarrow \text{CRM} \rightarrow \text{DTC} \leftrightarrow \mathbf{DTR}$$



DTR: Disciplined Timer

- MCPWM Timer
  - clocked 40MHz @25ns
  - period dithering
- DTR State
  - cycle_counter
  - timer_base_ticks
  - actual_period_ticks
  - shadow_period_ticks
  - base_period_fp16
  - period_ticks_frac_acc

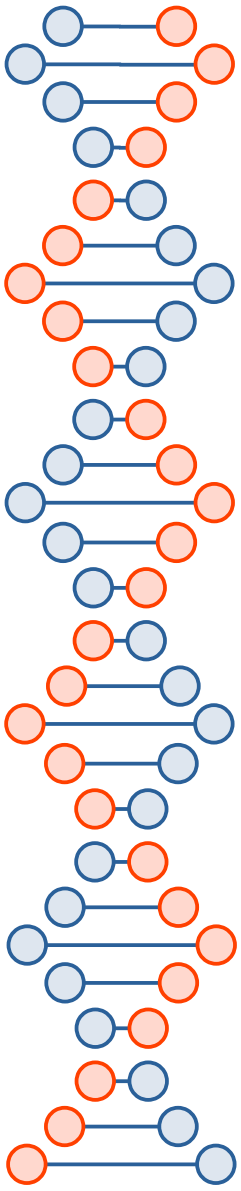
- Shadow register
  - Configure period any time
  - Have it set on next TEZ
  - Avoids race condition
- Period dithering
  - Update period_ticks from base_period_fp16 & period_ticks_frac_acc



DTR ISR

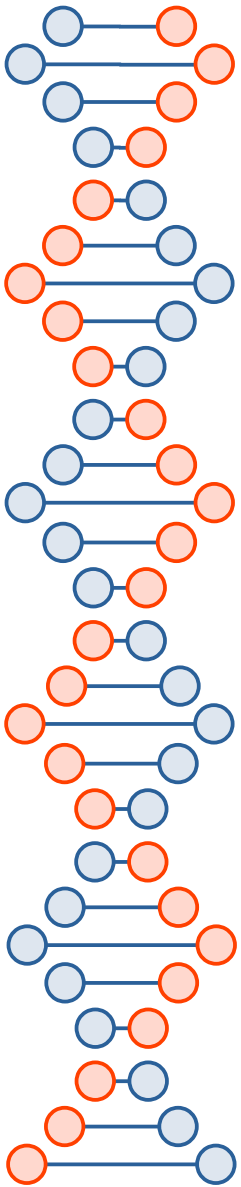
- Initial values (MCPWM)
  - cycle_counter = -1
  - active_period_ticks = 0
  - period_ticks = shadow_period_ticks = TIMER_PERIOD_TICKS
- ISR state at middle: @ TEZn+1
- Same ISR beginning and ending
- Impact of shadow reg
  - Set now at TEZn+1
  - Affects phase of TEZn+3

- Beginning
  - cycle_counter++
  - timer_base_ticks += active_period_ticks
  - active_period_ticks = shadow_period_ticks
- Middle: adjusts phase of TEZn+3 (!)
  - Dithering or alignment => **period_ticks**
- Ending
  - shadow_period_ticks = **period_ticks**
  - shadow_period_ticks => MCPWM HW



DTR: Alignment

- DTC tells DTR to align, targeting next TEZ
- Next TEZ should happen on **aligned_local_ticks**
  - period_ticks = aligned_period_ticks – timer_base_ticks
  - cycle_counter = aligned_cycle_counter + 1
  - If math ok:
    - period_ticks ~ TIMER_PERIOD_TICKS
    - cycle_counter = old_cycle_counter
- If period_ticks is too short, have to roll forward (only happens on initial align)
- Use adjusted FP16 base period for future cycles
  - base_period_fp16 = aligned_base_period_fp16

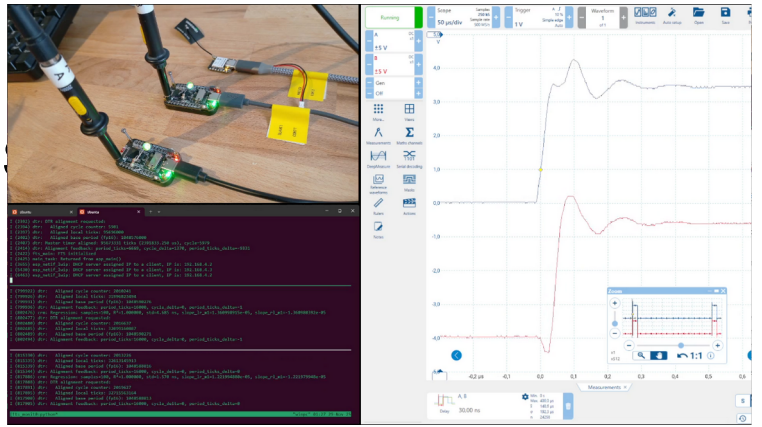


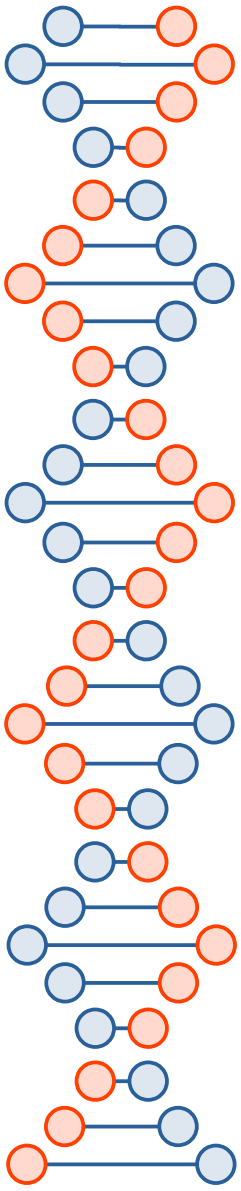
DTR: Feedback to DTC

- Signal TZE => DTC sets alignment close to TZE
- Results of last alignment handling
  - cycle_counter (=0)
  - cycle_counter – old_cycle_counter
  - period_ticks
  - period_ticks – old_period_ticks (~0)

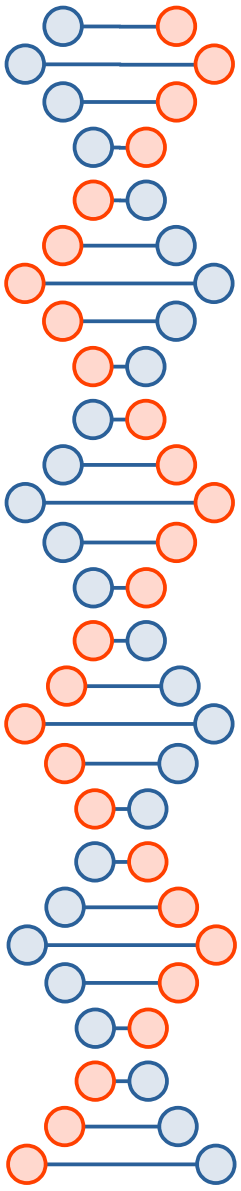
DEMO

- Master
  - Seeed Studio XIAO ESP32 S3
  - Built in Yellow LED @ GPIO21
- Slaves
  - 2x Waveshare ESP32-S3-LCD-1.47
  - Green LED tied to VDD @ GPIO1
  - Picoscope 5204 @ GPIO7
  -



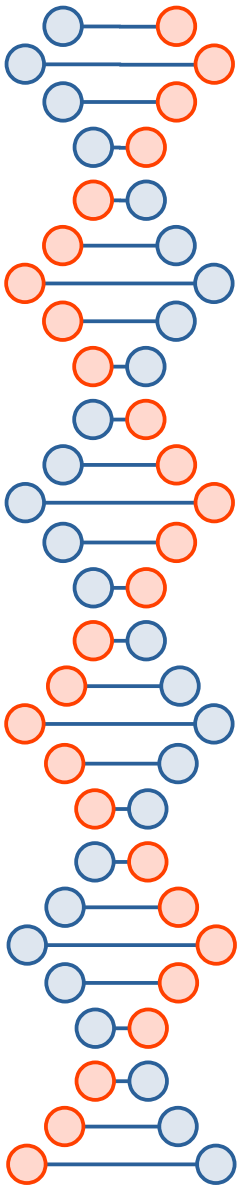


THE END



FIXME / TODO

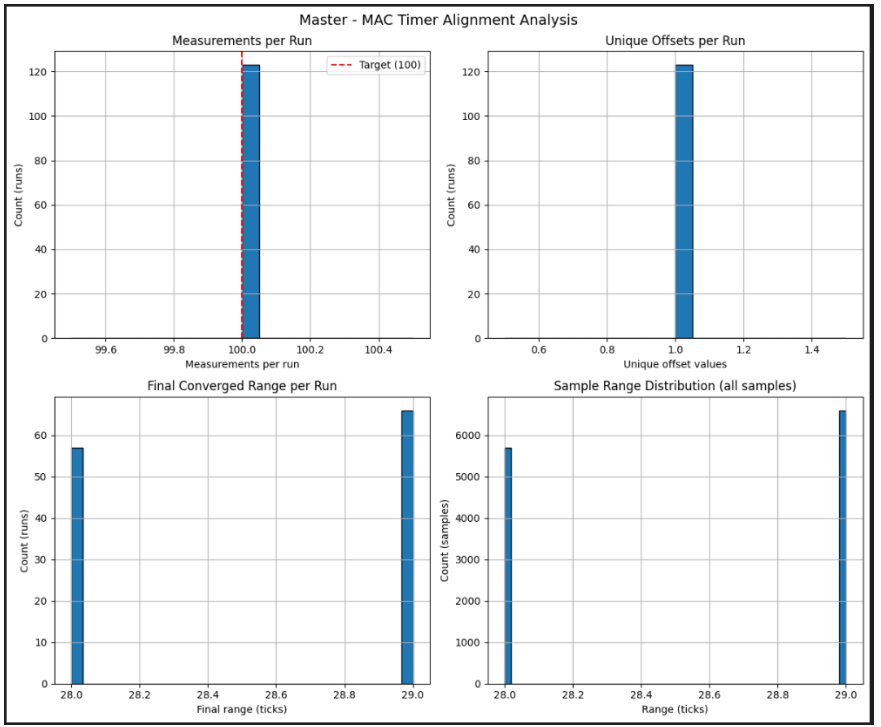
- add screenshots from notebooks to show MAC/FTM clock unwrapping
- broadcast master clock to allow arbitrary restarts (now must restart all)
- check for math problems (we mix signed and unsigned variables), reconsider units (ticks, ps, ns) for all vars
- consistent error handling strategy (check current error logs & aborts)



FUTURE

- implement as easy to use component / Arduino IDE library
- assess the impact of higher RTT / lower RSSI
- run MCPWM at higher clock rate (but seems to impact FTM quality)
- more FTM sessions, drop outliers & asymmetric entries
- use floats (by FPU) instead of doubles (soft emulation)
- finer CRM, slope velocity factor
- port to GPTimer to support C3
- power saving
- merge FTM/CRM/DTC: use period-aligned remote_ps as ref

MAC Clock Timer Start Offset Measurements Quality Control



- Set CONFIG_SYNC_MAC_TIMER_ALIGNMENT_TEST_CYCLES=100
- 120 runs, each run: do 100 measurements, reboot
- Gathered data from 3 systems (1x master, 2x slaves)

- All runs finish with 100 measurements - nothing breaks
- Within each run the algorithm always settles on the same min/max values, 1 unique offset per run
- The min / max bounds are 28-29 ticks apart, with roughly same probability << ? try to measure few times and take the narrowest

See docs/mac_timer_align_analysis/