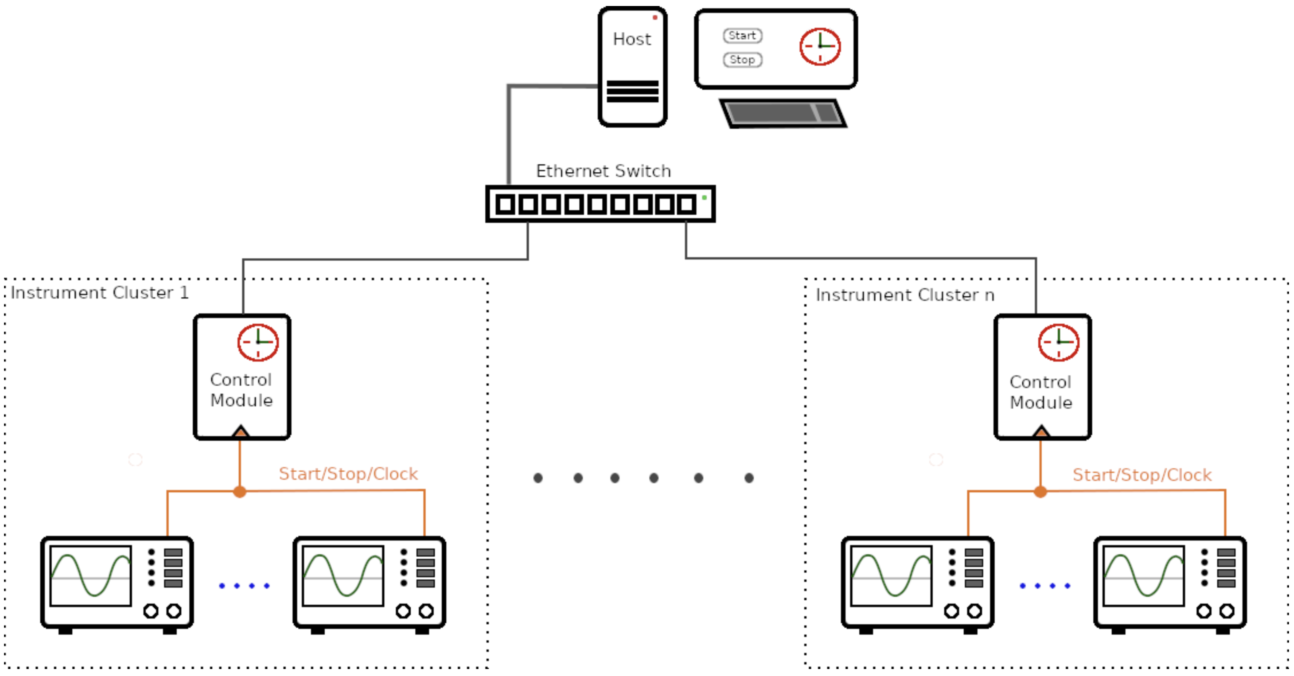
Precise Control of Instrumentation using PTP

A prototype system that precisely controls data collection instrumentation.
The start time, stop time and data transfer rate are synchronised across multiple instrument clusters (nodes), to within 100 nanoseconds.

Each node may contain multiple data collection instruments, connected to a single control module. The control modules are connected to the same LAN and controlled by a host server.



Host Controlling Instrument Clusters (Nodes)

Required Prototype Features

- Control modules connected to the same LAN using 100Base-T Ethernet
- Host server application configures and monitors control modules using TCP
- No grand master clock and no PTP ‘management’ messages
- Only PTP ‘event’ messages sent between control modules using UDP
- Designated control module acts as master clock
- Control module’s PTP clocks synchronised to within 100ns of each other
- Start and stop signals generated from control module’s PTP timer
- External clock generated from control module’s PTP timer

Key Components

- Host computer running Linux
- Host application written in C++ using the Qt toolkit
- Ethernet Switch to connect control modules to host computer
- Control modules consisting of Ethernet interface, IMXRT1062 MCU (with PTP timer) and frequency multiplier
- Control module program written in C, using the LwIP stack

Note: the prototype does not cover the interface(s) with the instrumentation, only the control signals.

What is PTP?

PTP stands for Precision Time Protocol and is defined by the IEEE 1588v2 standard. For the purposes of the prototype system, there is a single master clock and multiple ordinary clocks. PTP is used to synchronise the ordinary clocks to the master clock.

Why is PTP timestamping so accurate?

It uses hardware timestamping instead of software. By having specialized hardware fetch timestamps from the local clock, the master and ordinary clocks can avoid extra latency introduced by their local programs.

PTP Event Messages and Associated Timestamps

Every PTP sequence involves a series of four event messages between the master and ordinary clocks:

- The initial sync message from the master clock to the ordinary clock
- A follow up sync message from the master clock to the ordinary clock
- A delay request message from the ordinary clock to the master clock
- A final delay response message from the master clock to the ordinary clock

This sequence produces four different timestamps:

- t1 when the master clock sends the initial sync message
- t2 when the ordinary clock receives the initial sync message
- t3 when the ordinary clock sends the delay request
- t4 when the master clock receives the delay request

PTP Calculations using the Timestamps

The four timestamps t1..t4 are stored on each ordinary clock and used to calculate two variables:

offset	The time difference between the master and ordinary clocks
delay	The delay between the transmission and receipt of each event message (assumed to be symmetrical)

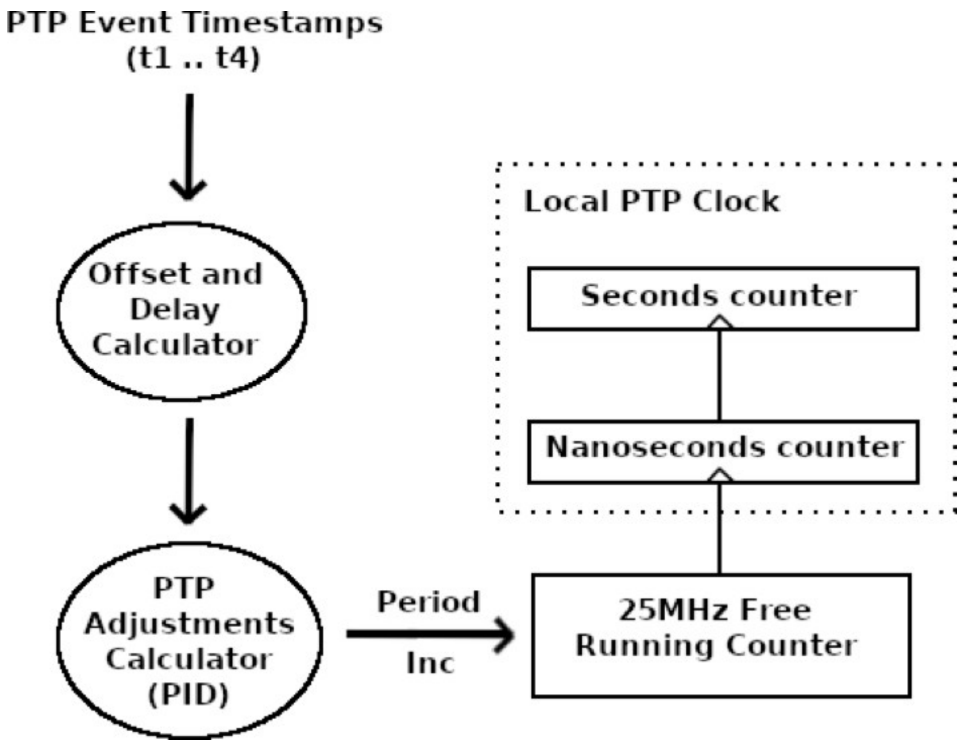
Let A = t2 – t1 = offset + delay
Let B = t4 – t3 = delay – offset

Rearranging and substituting:

offset = (A – B)/2
delay = (A + B)/2

Control Module’s PTP Clock

The PTP clock may operate as a master or an ordinary clock. The master clock is a free running 25MHz counter that increments a nanoseconds counter by 40 for each clock cycle. (The period of the 25Mhz clock is 40 nanoseconds). The nanoseconds counter overflow event increments the seconds counter.
When operating as an ordinary clock, the free running 25MHz counter is adjusted in proportion to the time offset between it and the master clock.


PTP Event Timestamps (t1 .. t4)
Offset and Delay Calculator
PTP Adjustments Calculator (PID)
Period
Inc
Local PTP Clock
Seconds counter
Nanoseconds counter
25MHz Free Running Counter

Adjusting the PTP Clock

Adjusting the Free Running Counter

The free running counter is the timebase for the PTP clock. The inputs to the counter are:

- 25MHz Clock (40ns period)
- Adjustment period
- Adjustment increment

To adjust the frequency of the free running counter, an adjustment increment is substituted for the ‘normal’ increment (of 40ns). This substitution takes place every adjustment period clock cycles. The adjustment increment is usually 40ns +/- 1ns i.e. 41ns or 39ns. The larger increment increases the frequency (the clock runs faster), whilst the smaller increment decreases the frequency (the clock runs slower). The adjustment increment is determined by the sign of the time offset between the master and the ordinary clock e.g. when the ordinary clock is behind the master clock and must speed up, the increment is 41ns.
The adjustment period determines how often the adjustment period is applied and therefore controls the rate of frequency change. It is calculated by a PID algorithm that uses the time offset as it’s input.

Calculating the Adjustment Period

A Proportional Integral Derivative algorithm is used to calculate the adjustment period. The main formula has three terms, each multiplied by a constant:

output = (kp * error) + (ki * integral) + (kd * derivative)

Where:

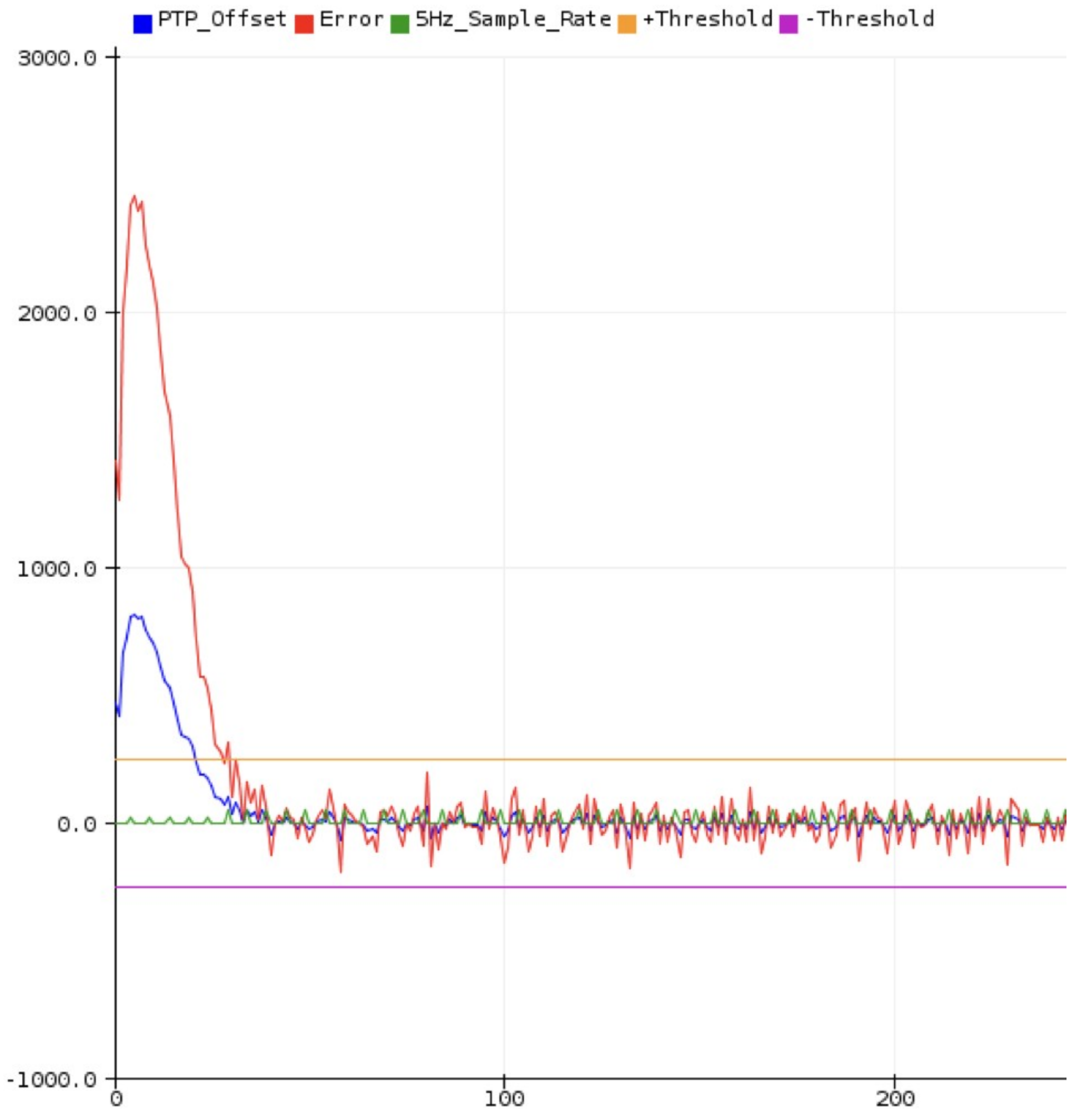
error = time offset/time between successive offsets
integral = previous integral + error
derivative = error – previous error

The adjustment period is obtained from the output:

adjustment period = 25000000.0/abs(output)

The graph below shows how the offset and error in nano seconds (Y axis) are adjusted over time in seconds (X axis). The rate of PTP message exchanges was 5Hz. Higher sample rates result in faster response times.

The goal is to keep the error within the +/- threshold limits.



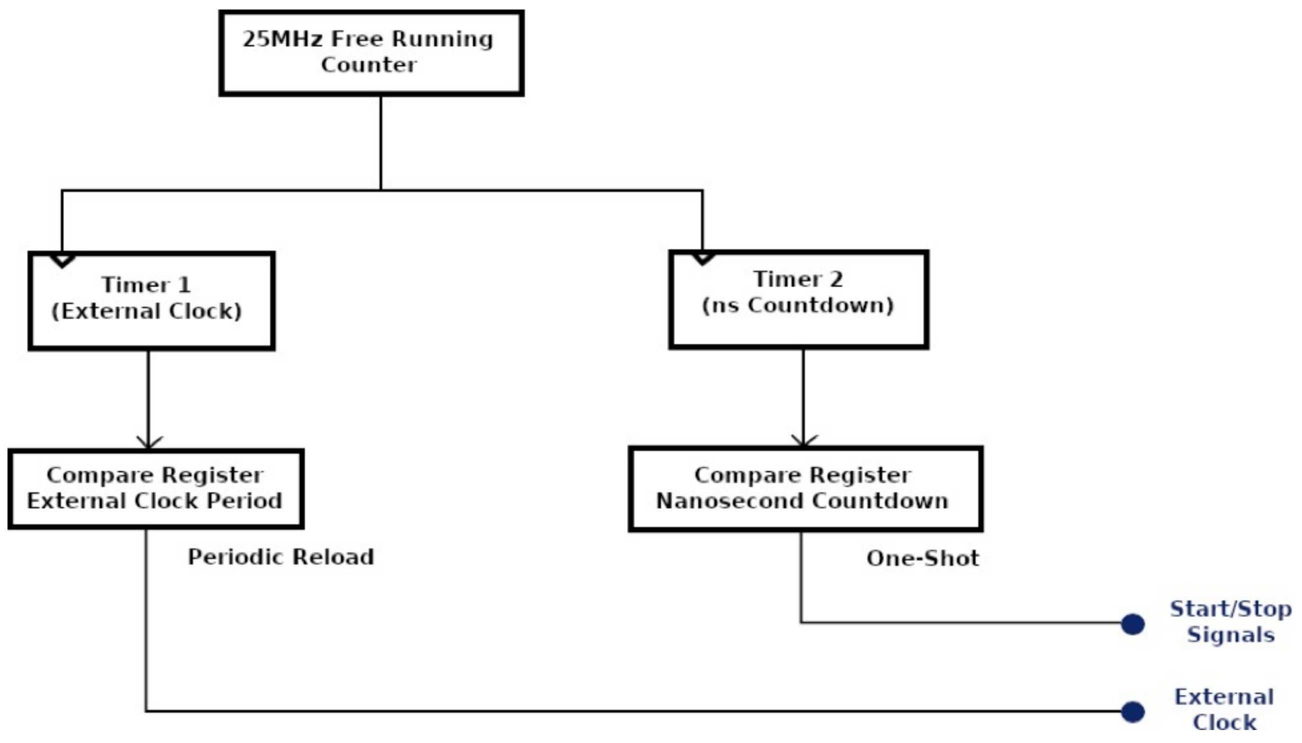
PTP Offset Correction Using PID Algorithm

The values of the constants kp, ki and ki were set by experiment, with different sets of constants depending on the sample rate.

Producing the Start, Stop and External Clock

The control module’s external signals are produced using timers configured in compare mode. Compare mode uses a register to hold a value that, when matched by the timer’s counter creates an event. The event generates an interrupt and is also routed to a GPIO pin using the MCU’s I/O multiplexer.

The clock input to each timer is the output from the free running counter.


25MHz Free Running Counter
Timer 1 (External Clock)
Timer 2 (ns Countdown)
Compare Register External Clock Period
Compare Register Nanosecond Countdown
Periodic Reload
One-Shot
Start/Stop Signals
External Clock

Compare Timers Producing External Clock and Start/Stop Signals

The external clock events continually repeat at a period set by the external clock period value which is in nanoseconds.

The start/stop signals are a one-shot (single) event. The nanosecond countdown is loaded when the control module receives a command from the host to start or stop at a specific time. The time is usually in seconds with a nanosecond component. When the seconds have counted down, Timer 2 is enabled, resulting in the Start/Stop event occurring with a high degree of accuracy.

The use of events connected to GPIO pins improved synchronisation of external signals between different control modules because the events are produced by the timers, which, being hardware-based are semi-autonomous i.e. independent of the CPU/Memory load.

Using a Frequency Multiplier on the External Clock

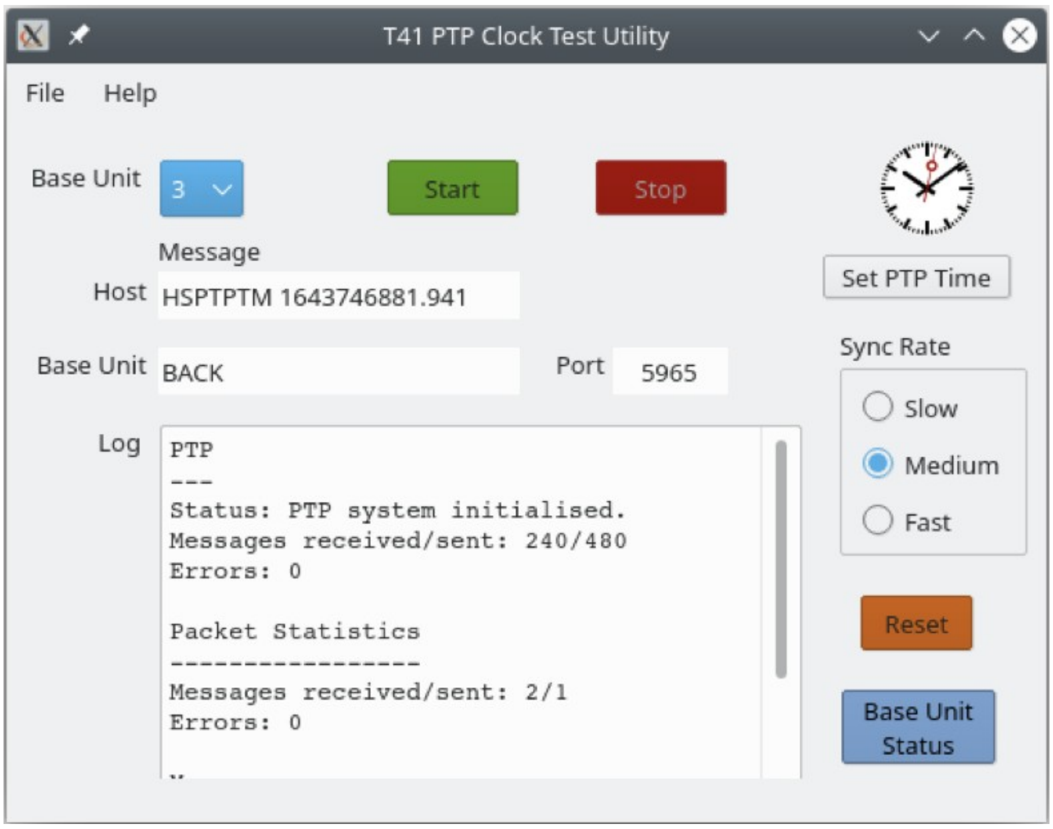
The prototype clock module included a 64x frequency multiplier that operated in the range of 1.5MHz to 5MHz. By setting the external clock period in Timer 1's compare register to 64 times the desired output period, the required frequency could be obtained.

For example, to obtain 4MHz (250ns) at the output of the frequency multiplier, an input frequency of 62.5KHz is needed. This is equivalent to an external clock period of 16000ns. (16000 = 250 x 64).

The benefits of using a frequency multiplier are reduced load on the control module's CPU/Memory due to the reduction (by a factor of 64) of the number of interrupts generated by Timer 1 and higher instrument data transfer clock speeds.

(For details of the frequency multiplier hardware, please see the article in the electronics section).

Host Application



Base Unit: 3

Message

Host: HSPTPTM 1643746881.941

Base Unit: BACK

Port: 5965

Sync Rate: ( ) Slow [✓ Medium] ( ) Fast

Log:

```
PTP
---
Status: PTP system initialised.
Messages received/sent: 240/480
Errors: 0

Packet Statistics
-----------------
Messages received/sent: 2/1
Errors: 0
```

Host Application

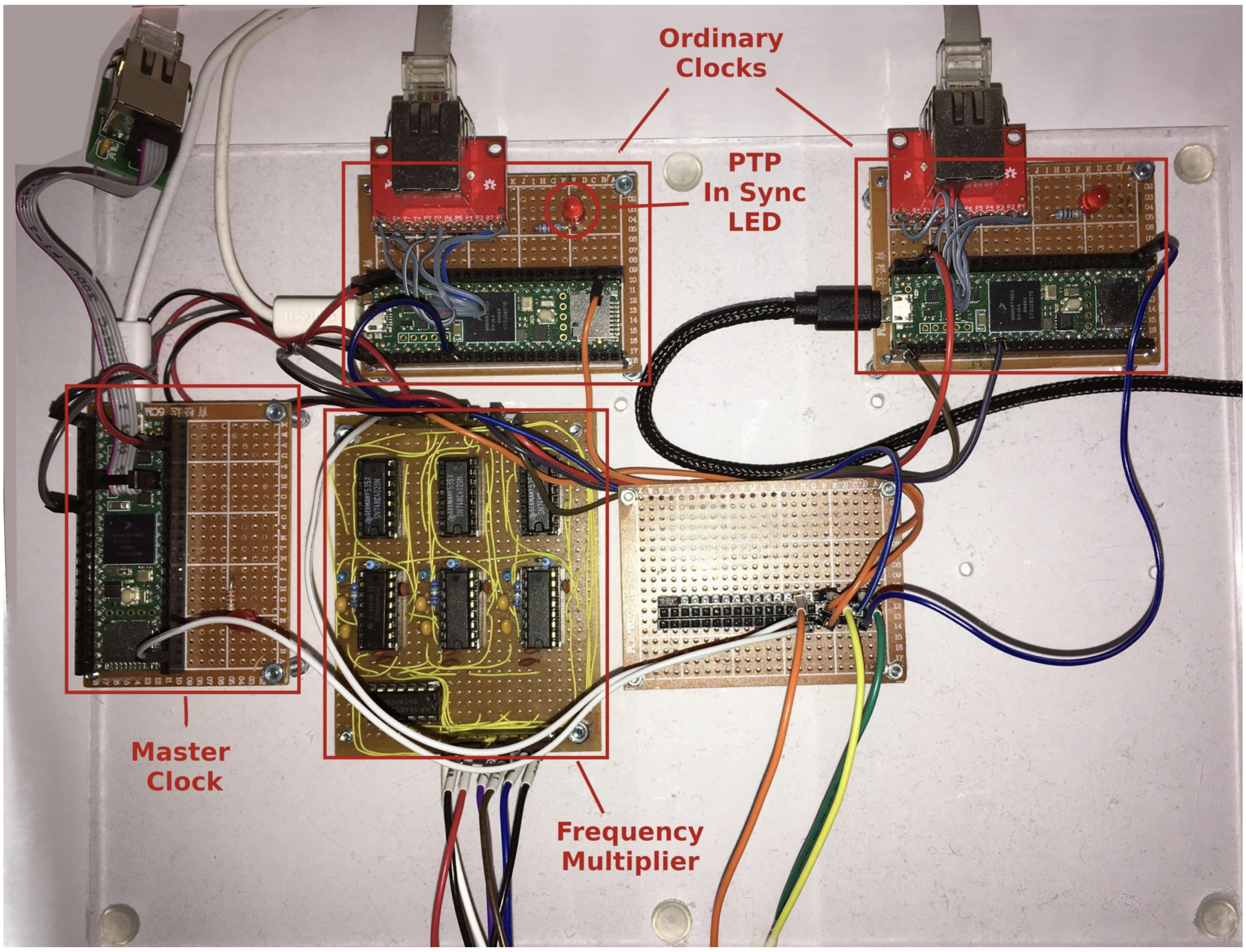
The host application is used to configure the control modules, designating the master and ordinary clocks and setting the PTP synchronisation message rate.
Once configured, the control modules PTP time is then set to the host server's time.

When the ordinary clocks are synchronised to the master clock, instrument data collection can be started by pressing the start button.

Test Setup

The prototype was developed and tested using three Teensy 4.1 development boards with Ethernet adapters. A board containing three 64x frequency multipliers was constructed along with test points for the Saleae Logic 16 logic analyser. The Ethernet switch was a TP-Link TL-SG1005D Gigabit Switch.

The Qt host application ran on an HP8300 Elite USDT under Debian 11.



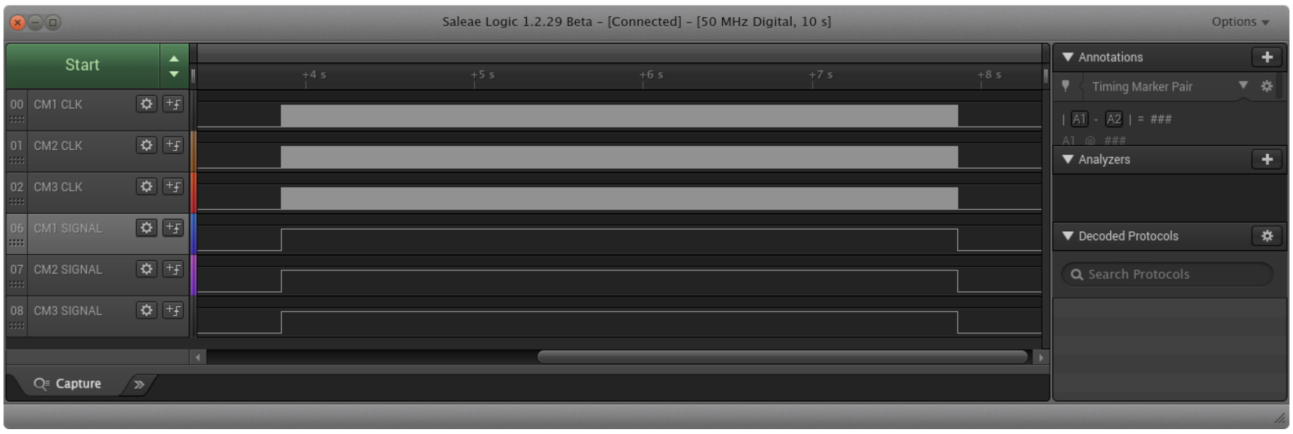
Results

The following logic analyser screenshots show traces for three control modules (one master and two ordinary clocks).
The left hand column of each trace shows the name of each channel:

Channel	Description
0	Control Module 1 Frequency Multiplier O/P
1	Control Module 2 Frequency Multiplier O/P
2	Control Module 3 Frequency Multiplier O/P
3	Control Module 1 MCU Clock O/P
4	Control Module 2 MCU Clock O/P
5	Control Module 3 MCU Clock O/P
6	Control Module 1 Start/Stop Signal
7	Control Module 2 Start/Stop Signal
8	Control Module 3 Start/Stop Signal

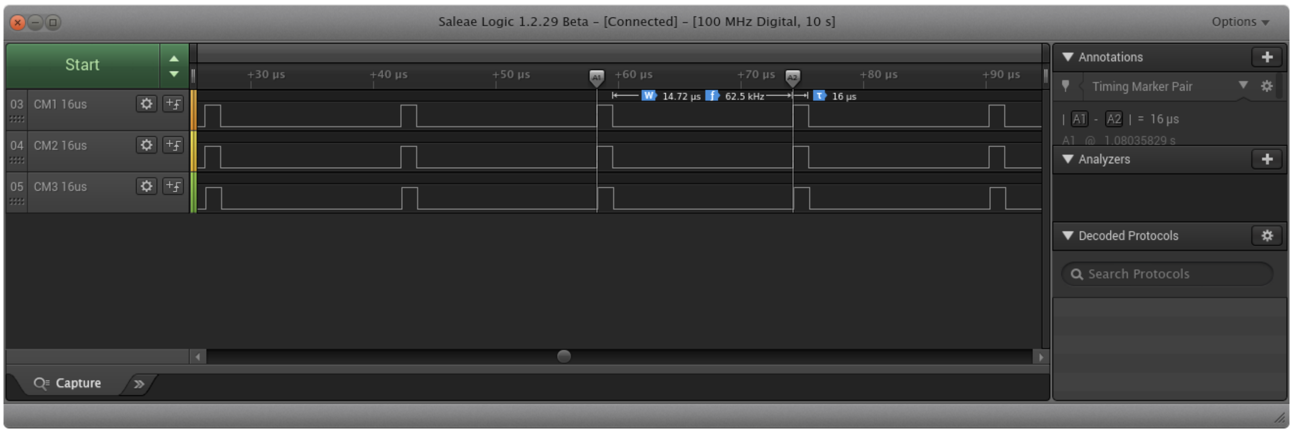
The right hand column “Timing Marker Pair” section shows the time difference between the two markers A1 and A2.

Synchronisation of the three control modules in starting and stopping instrument data collection:



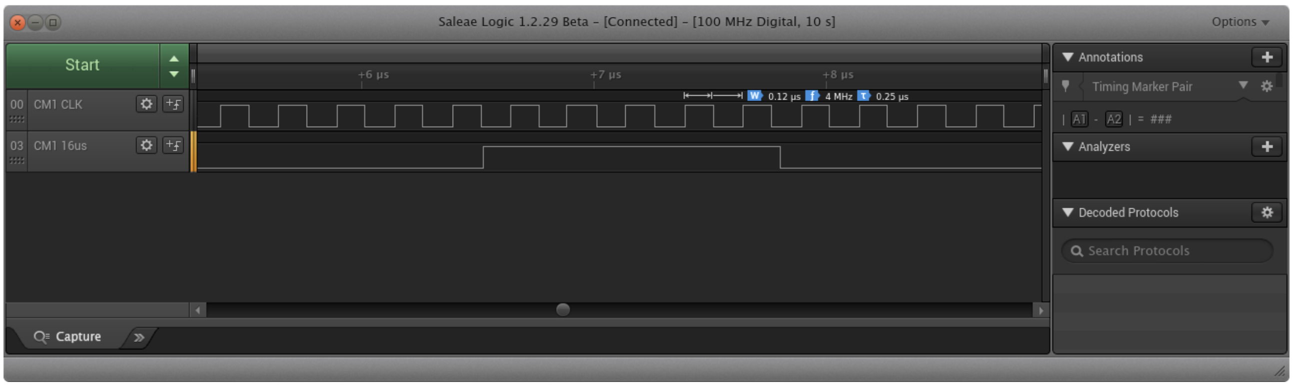
Start, Instrument Data Transfer and Stop

Synchronised 62.5KHz external clock output of each control modules’ MCU:



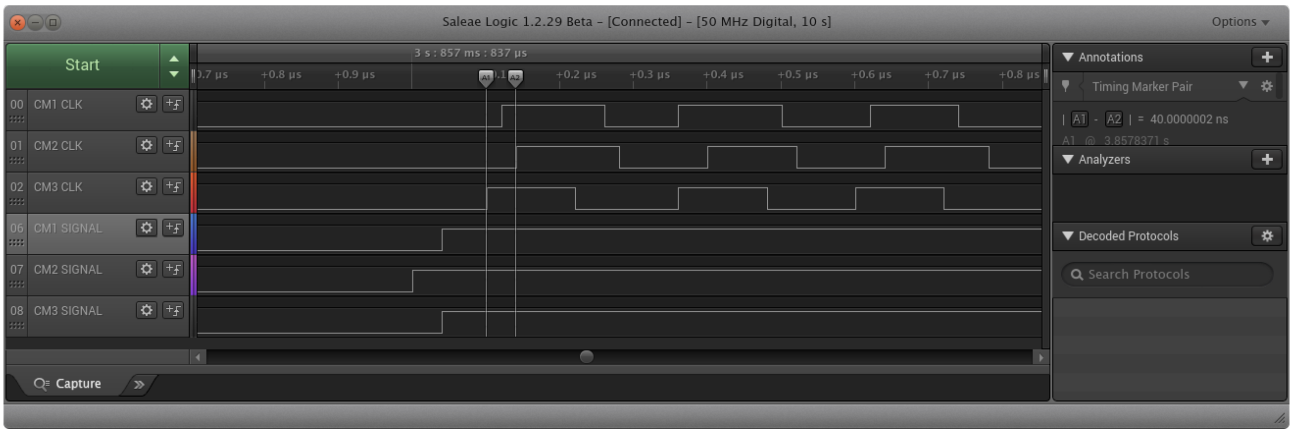
MCU External Clock (16us Period)

MCU external clock and the corresponding frequency multiplier output (4MHz).
Of note is the 50% duty cycle of the 4MHz clock even though the input to the frequency multiplier has a low mark to space ratio:



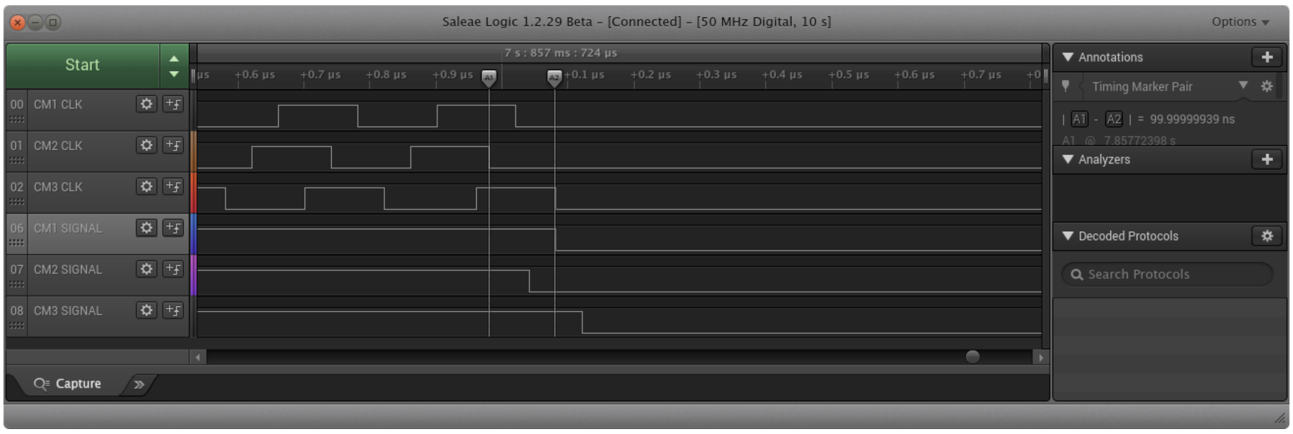
Frequency Multiplier 4MHz Clock

40ns time interval between beginning data transfer clocks (frequency multiplier output):



Start Signals with Time Gaps

100ns time interval between end of data transfer clocks (frequency multiplier output):



Stop Signals with Time Gaps

Conclusions

The results demonstrate that an interval of 100ns or less between control module signals is possible.

Potential Improvements

The system has scope for improvement if an interval of less than 100ns is desired and the data collected can be accurately referenced to ‘actual’ time.

Increasing the Frequency of the Free Running Clock

The interval could be reduced by increasing the source clock of the free running counter from 25MHz to 100MHz, resulting in 10ns increments instead of 40ns increments.

Using a Stratum 1 Clock as the Master Clock

The use of a delegated clock module as the master clock leads to the ordinary clocks matching the master clock as it drifts in frequency over time. Therefore, although the data collection instruments will be synchronised, the data samples will need to be referenced to the master PTP clock and not an external time reference. Adding a stratum 1 clock to the network along with the PTP management messages would ensure that data collected could be referenced to ‘actual’ time.