Using PICO PIO to process Manchester and PWM encoded signals

Table of Contents

1 Summary

Uses two PIO state machines to handle the raw incoming bits being received by one 868 MHz RF OOK receiver from a WH1080 weather station (PWM encoded) and the incoming bits being received by one 433 MHz RF OOK receiver from five F007T remote temperature sensors (Manchester encoded). Theses are processed on one CPU core and added to message queues. The other CPU core reads the queues, formats the messages and transmits them over a UART.

Main files

- PICO_PIO_F007T_WH1080.c
- PICO_PIO_WH1080.c
- PICO_PIO_F007T.c
- queues_for_msgs_and_bits.c
- uart_IO.c
- output_format,c

PICO_PIO_F007T_WH1080 has the main() entry point on CPU core 0. It calls initialisation functions that includes WH1080_init (in PICO_PIO_WH1080.c) and F007T_init (in PICO_PIO_F007T.c). It starts core 1 and then core 0 goes into a while doing nothing loop.

On core 0 the WH1080_init and F007T_init start the PIO state machines and the repeating timers that trigger callbacks. For each RF receiver input, one callback polls the PIO's FIFO and puts the message bits into a bit queue and another callback "parses" that bit queue looking for the signature of valid messages and adds them to a message queue which is being polled from core 1.

Core 1 initialises the 2nd UART and periodically polls the message queues of the WH1080 and F007T running on core 0, formats them and sends them out through the 2nd UART. It also polls this UART for any messages received (currently not used).

Spin locks protect the message queue access between core 0 and core 1. The bit queues assume access is from the same CPU core and at the same interrupt priority level.

StdIO is on the default UART and is used for debug/statistics.

2 Remote Sensors

2.1 F007T

The particular F007T temperature sensors being used transmit 433 MHz OOK Manchester encoded messages. Example sensors are Oregon Scientific WMR86, Ambient Weather F007 and Froggit FT007. No doubt all are manufactured/cloned in China and just re-badged. Be aware that the encoding method may change between different versions of sensors.

For this F007T, the RF receiver can be any one that does 433 MHz OOK e.g. RF Solutions AM-RX9-433P or RadioControlli RCRX-434-L The message protocol is a bit limiting for the sophisticated receivers like HopeRF RFM65. There are also decodes for SDR. e.g. ambient_weather.c at https://github.com/merbanan/rtl_433

Its been done on Arduino's - Rob Ward (and the people he credits), https://github.com/robwlakes/ArduinoWeatherOS using a decoding delay algorithm.

Ron Lewis, https://eclecticmusingsofachaoticmind.wordpress.com/2015/01/21/home-automation-temperature-sensors/ for the checksum algorithm?

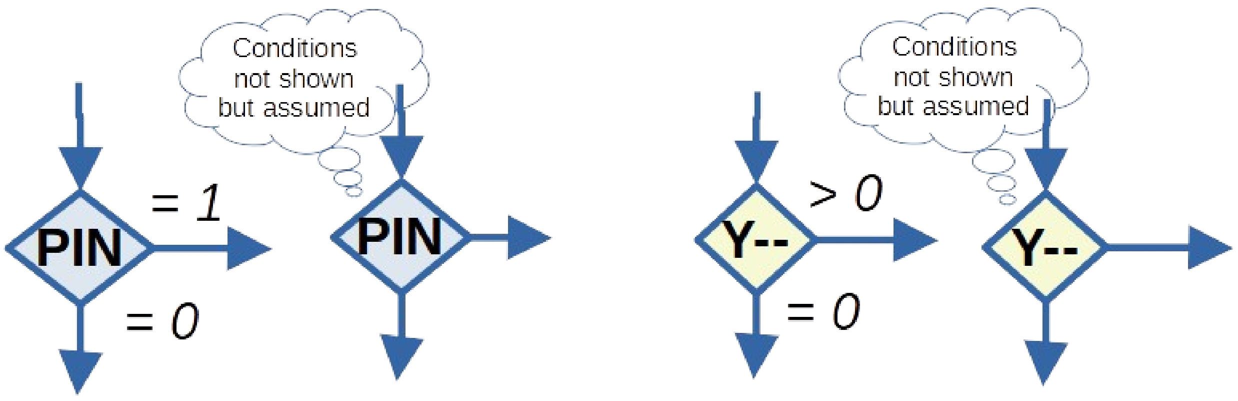
2.2 WH1080

The WH1080 weather station being used transmits 868 MHz OOK PWM encoded messages. Again probably re-badged and the encoding method may change between different versions of sensors. Simple RF OOK receivers that are low voltage are harder to come by - RadioControlli do the RCRX-868-L. Again there are SDR. decodes given at https://github.com/merbanan/rtl_433 by Benjamin Larsson et al.

The sensors on this weather station have outdoor temperature and humidity, rain and wind speed and direction. It also reports datetime information a couple of times over a day - received from Mainflingen near Frankfurt in Germany.

3 Flowchart Notes

To reduce clutter, some of arguments have been ignored on PIO instructions in the somewhat old-school flowcharts shown later. For the Manchester program, a flowchart delay symbol is shown following the 'WAIT' instruction. The delays for the PWM program have been specified on some conditional 'JMP' instructions so the flowchart delay symbol applies to both out branches.
Again to reduce flowchart clutter (and thought) a simplistic convention has been used for conditional 'JMP' instructions. The branching decisions have
- flow out is always to the left or right if the test result is > 0
- and always out of the lower if it's 0.
So on a flowchart the following '= 1', '= 0' and '> 0' aren't shown and are, by this convention, to be assumed. e.g.


Conditions not shown but assumed
PIN
= 1
= 0
PIN
Conditions not shown but assumed
Y--
> 0
= 0
Y--

4 PIO Notes

4.1 There is some advantage to noise

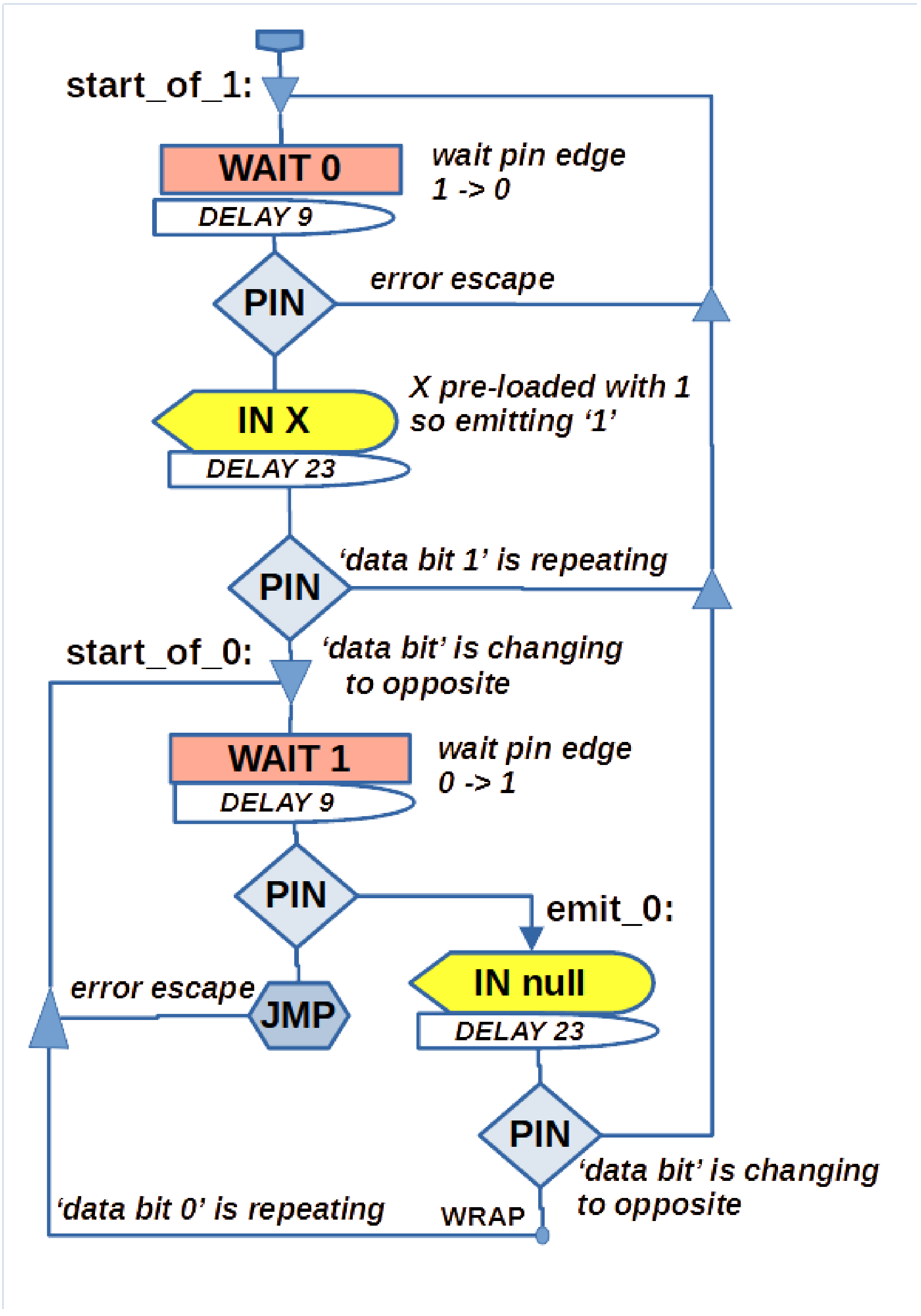
Both PIO programs push the data bits through the FIFOs 32 bits at a time. This could delay data bits for messages that aren't multiples of 4 bytes – one message might not be seen as complete until the next message pushes the remaining bits of the previous through the FIFO. The RF receivers, with active gain control, produce a lot of noise pulses. The "advantage" that this has is the FIFO 4 byte delay is not so noticeable.

4.2 Too much noise is a disadvantage

Because the RF receivers are so noisy and because the PIO algorithms are not very defensive there has to be message integrity checking. The WH1080 and F007T both have checksums but these aren't strong enough for all types of noise. Both WH1080 and F007T repeat the message more than once in some transmissions and so if the checksum passes then the message can be compared to the previous one. If two consecutive messages pass the checksum and are identical then the message could be accepted as good. However there is still a possibility that corruption can occur that passes the checksum and also the consecutive identical test. Be aware that the PWM OOK with 10 byte message and 8 bit CRC appears to be susceptible with a lot of '1's in the noise.
The Manchester PIO algorithm is not good for an ideal signal with no noise. If the input is idle low then, at program start up, a '1' data bit is involuntarily emitted. When the header sequence of '1's arrives, the algorithm emits '0's instead of '1's. This happens until the first bit reversal occurs. After that, the emitted data bits are correct. Fortunately the F007T repeats the message 3 times in a transmission and so the following two messages are reported correctly. Rearranging the PIO algorithm to avoid this has defeated me so far - partly because, for a signal with noise, the algorithm appears to work sufficiently well.

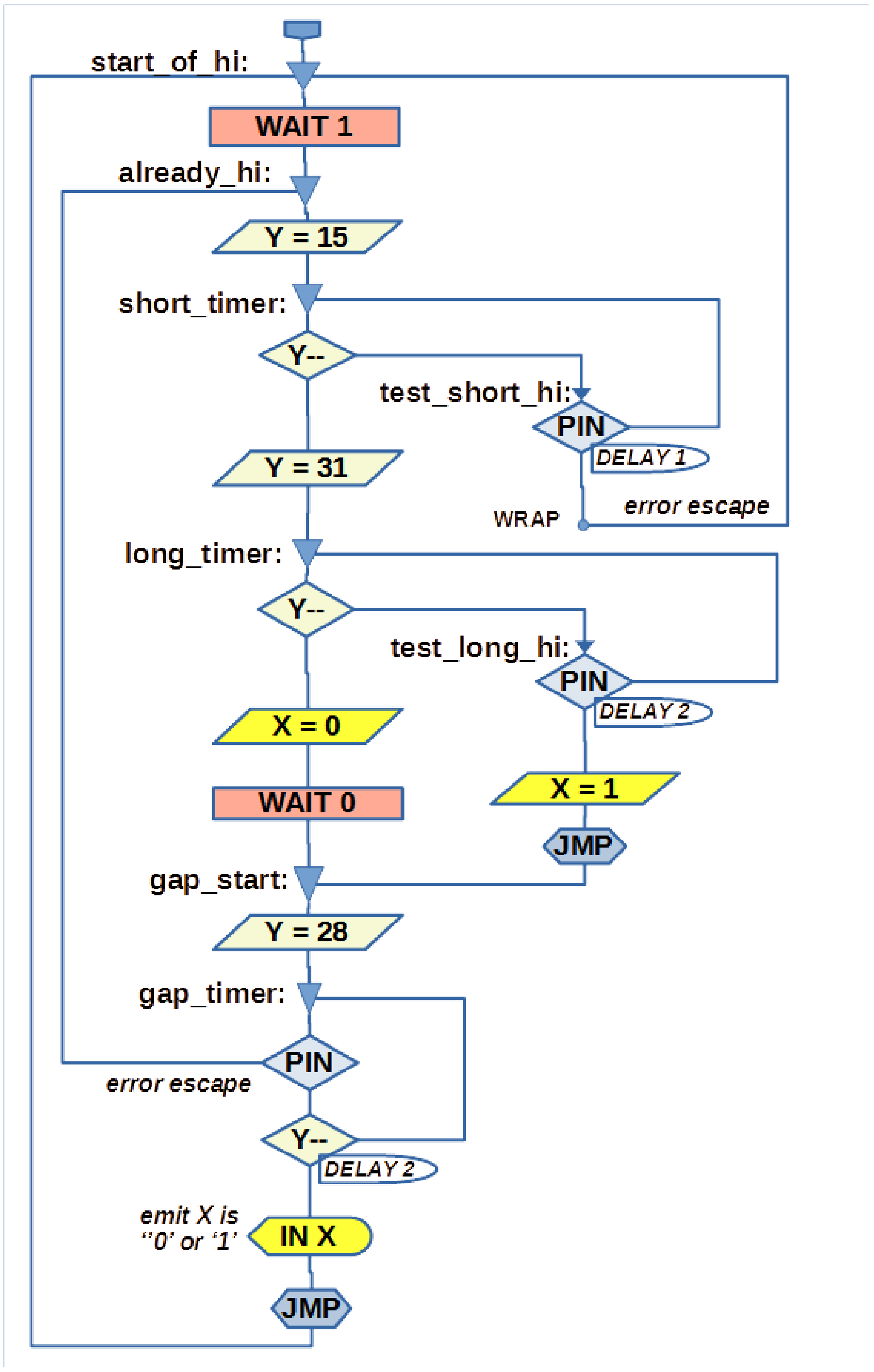
5 PIO manchWithDelay (9 instructions)

125 MHz Clock with divider 2543.0F => 20.34 us clock tick
One information ‘data bit’ @ 1024 kHz => 976.56 us per ‘data bit’ (the Manchester frequency is up to twice the ‘data bit’ frequency) giving 48 cycles per ‘data bit’ if it’s the same ‘data bit’ value repeating (the Manchester frequency is the same as the ‘data bit’ frequency if the ‘data bit’ values are alternating).
Method:- wait for an edge then, after a short wait, test again for the changed-to-value. If still the same then emit the appropriate ‘data bit’ value and test again after a long wait. If it’s changed from the changed-to-value then the same ‘data bit’ value will be repeating, otherwise the next ‘data bit’ value will be the opposite.
short wait 9+1 = 10 => 203.4 us, long wait 23+1 = 24 => 488.16 us



6 PIO PWMpulseBits (16 instructions)

Roughly: a data ‘1’ is 0.5 ms pulse, ‘0’ is 1.5 ms and the gap between pulses is 1.0 ms.
Method:- count duration of a pulse, if long enough for a ‘1’ but not a ‘0’ or if long enough for a ‘0’.
After ‘1’ or ‘0’ then count the gap duration, if gap long enough emit X which will be ‘1’ or ‘0’.
Divider 625.0F => sm clock period 5 us
=> short_timer: one loop time = 5 us * (2+1) = 15 us; loop time * 15 = 225 us
=> long_timer: one loop time = 5 us * (1+3) = 20 us; loop time * 31 = 620 us
=> gap_timer: one loop time = 5 us *(1+3) = 20 us; loop time * 28 = 560 us



7 Performance Measurement

7.1 What to measure

- Rate of bits per second through the PIO state machine and the proportion of bits that have the value ‘1’. This is a rough proportion as it will vary with message content.
- Rate of message headers per minute with proportion of headers with valid checksum.
- Rate of messages with valid checksum and that have had two identical consecutive messages and the proportion of these against all those messages with valid checksums.
- The delta time between valid messages. Its easy to measure and should, on average, be the equivalent to the rate.
- Queue “high water marks”
    - max number of unread 32 bit words found in a FIFO
    - max number of unread 32 bit words found in a bit queue
    - max number of unread messages found in a message queue

The algorithm that looks for identical consecutive messages only allows for a maximum of two and the F007T transmission which has message in triplicate only counts as one message duplicated and not two.

7.2 “Ideal” (no noise) Rates and Proportions

7.2.1 F007T 433 OOK Manchester

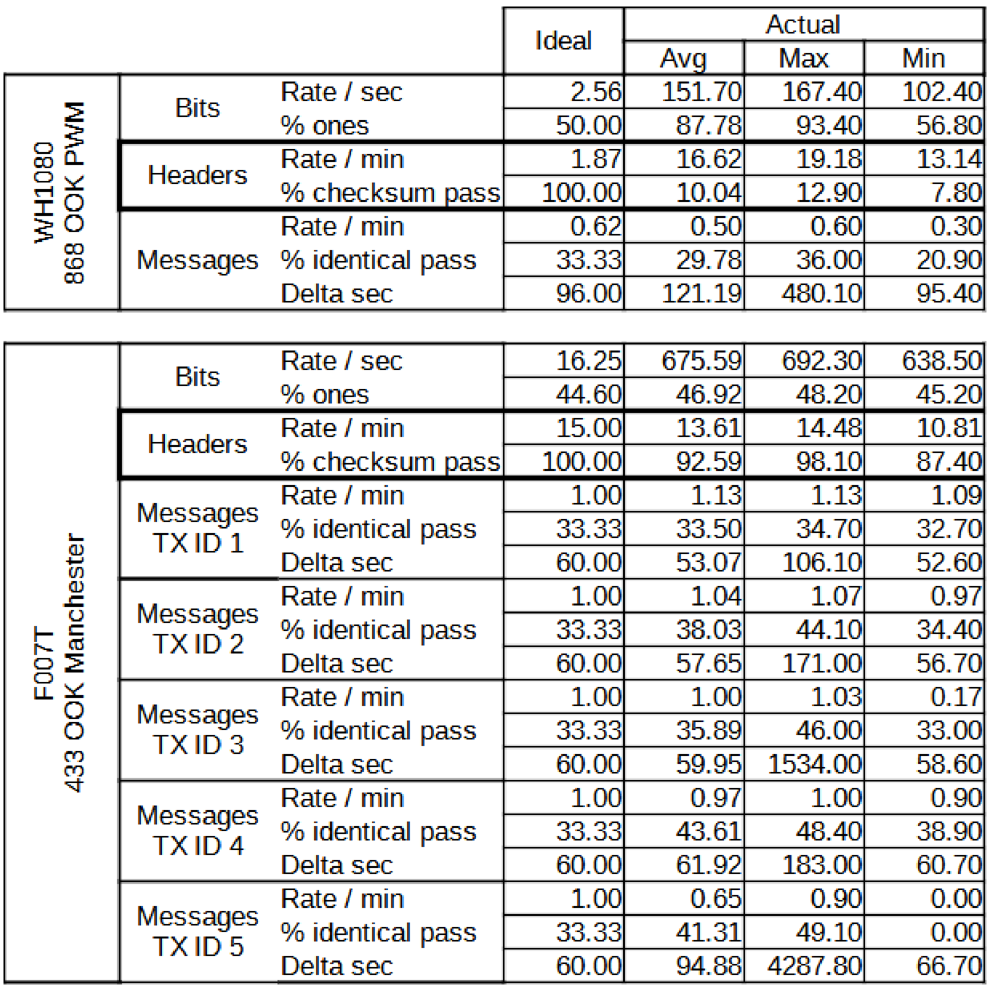
5 transmitters, each transmitting every 60s, each transmission has the message in triplicate, a triplicated message is 195 bits (87 one's)
=> 5 * 195 = 975 bits per min (435 1's) => 16.25 bits per second (44.6% one's)
=> 5 * 3 headers per 60 seconds => 15 headers per minute with 100% checksum passed
=> verified messages rate for each transmitter is 1 per minute (33.3% consecutive identical seen)

7.2.2 WH1080 868 OOK PWM

1 transmitter with transmission rate of 48s with every other transmission having the message duplicated.
3 messages headers per 96s; one transmission of 88 bits (44 one's) and one of 176 bits (88 one's)
=> 264 bits per 96 seconds (with 132 one's) => 2.5625 bits per second (50% one's)
=> 3 headers per 96 seconds => 3*60/96 = 1.875 headers per minute with 100% checksum passed
=> verified messages rate 1/3 the header rate => 0.625 per minute (33.3% consecutive identical seen)

7.3 Results

The code was instrumented to log the header and message statistics over 30 minute sample periods and the bit rates were recorded every minute. The delta times between valid messages from each individual transmitter were logged as they occurred. These were recorded over multiple 24 hour periods. The high water marks are not reported in the following table as the repeating timer callbacks are currently set to quite a high frequency. The maximum number of unread words found unread in FIFO was 2, the bit queue word max was 4 and the message queue was 2.



			Ideal	Actual		
				Avg	Max	Min
WH1080 868 OOK PWM	Bits	Rate / sec	2.56	151.70	167.40	102.40
		% ones	50.00	87.78	93.40	56.80
	Headers	Rate / min	1.87	16.62	19.18	13.14
		% checksum pass	100.00	10.04	12.90	7.80
	Messages	Rate / min	0.62	0.50	0.60	0.30
		% identical pass	33.33	29.78	36.00	20.90
		Delta sec	96.00	121.19	480.10	95.40

			Ideal	Avg	Max	Min
F007T 433 OOK Manchester	Bits	Rate / sec	16.25	675.59	692.30	638.50
		% ones	44.60	46.92	48.20	45.20
	Headers	Rate / min	15.00	13.61	14.48	10.81
		% checksum pass	100.00	92.59	98.10	87.40
	Messages TX ID 1	Rate / min	1.00	1.13	1.13	1.09
		% identical pass	33.33	33.50	34.70	32.70
		Delta sec	60.00	53.07	106.10	52.60
	Messages TX ID 2	Rate / min	1.00	1.04	1.07	0.97
		% identical pass	33.33	38.03	44.10	34.40
		Delta sec	60.00	57.65	171.00	56.70
	Messages TX ID 3	Rate / min	1.00	1.00	1.03	0.17
		% identical pass	33.33	35.89	46.00	33.00
		Delta sec	60.00	59.95	1534.00	58.60
	Messages TX ID 4	Rate / min	1.00	0.97	1.00	0.90
		% identical pass	33.33	43.61	48.40	38.90
		Delta sec	60.00	61.92	183.00	60.70
	Messages TX ID 5	Rate / min	1.00	0.65	0.90	0.00
		% identical pass	33.33	41.31	49.10	0.00
		Delta sec	60.00	94.88	4287.80	66.70

7.4 Notes on Results

The amount of ones for the WH1080 is very high and will inevitably fool the checksum.

Some of the F007T transmitters (TX ID's 3, 4 and 5) are further away and some are also reporting low battery.