EvtHRMOffset Computation

Analysis

Fri, Jan 17, 2003

The logic used to compute the average offset between the clock event time stamps and the HRM Slow Data time stamps as done in the `EvtHRMOffset` routine is a bit obscure. This note analyzes the results that are observed.

The objective behind the logic is to provide an evolving average value for each occurrence of a power-of-2 in the accumulation count, until the first complete period is reached. To be precise, the accumulation period in use is 1024 cycles, in which a new offset value is extrapolated for each successive 15 Hz cycle in between average computations. Once the first 1024-cycle period has passed, after having computed 11 short-term averages, a new average is calculated every 1024 cycles thereafter, which is about every 68 seconds. Before the first 1024-cycle period is complete, a new average is computed for every power-of-2 value in the accumulation counter. The first average is computed for the count value of 1, which is 2^0. The second is computed for a count of 2, which is 2^1, the third for a count of 4, which is 2^2, etc. The tenth average is thus computed for a count of 512, which is 2^9.

What does the 10th average represent? As we evaluate successive averages and compute their differences to gauge the drift between the two independent time stamp crystal-driven counters, consider what we have at this point. The average value over the first 512 offsets gives us a measure of what the drift was halfway to this point; i.e., it measures the drift at 256 cycles. As we move on to the 11th average at the 1024-cycle point, its value represents the drift at the 512-cycle midway point. If we ask what the difference in these two averages represents, it amounts to a measure of the drift over 512 – 256 = 256 cycles.

After the 11th average calculation, the algorithm shifts into its final mode, and new averages are computed every 1024 cycles. The difference between the 12th average value and the 11th will show what amounts to measuring the drift at 1536 cycles and subtracting the drift at 512 cycles, for a difference of 1024 cycles. Everafter, this situation will repeat.

This analysis is exactly what we see. The 11th average shows a difference compared with the 10th average of `0xB8`, say, and the 12th and subsequent averages computed show values about 4 times higher, as `0x2E2`. (The 10th average showed a difference compared with the 9th average, `0x2F`, that is about half the difference stored with the 11th average.)

These differences are used to support an extrapolation of an average offset value that can be used by the `RFTData` routine in time-stamping data points relative to a selected clock event. During the 512 cycles between the 10th and 11th averages, for example, the base average value computed at the 512 cycle point must be augmented by using a fraction in the range 0–1 during the 512 cycles between 512 and 1023. During this time, the shift variable is 10. To get a suitable fraction, we need to use 0–511 in the numerator and 512 in the denominator. We can get the power-of-2 denominator by using the shift value less 1. The numerator comes from ANDing the accumulation counter with a 9-bit mask. But what do we do about the difference that effectively measured the drift over 128 cycles? It needs to be multiplied by 4, so that it represents a 512-cycle drift.

What about the previous power-of-2, after the 256-cycle average was computed? The difference between that average and the previous one measures the drift over 64 cycles. Here, while the shift value is 9, we need a denominator that is 256 and a numerator that ranges from 0–255. We get this by ANDing the accumulation counter with an 8-bit mask and again multiplying the difference by 4.

Finally, let's go the other direction. What about the period of time between the 1024th and 2047th cycles, after the 11th average was stored, when a completely new average is accumulated

without making any further short-term calculations of the average? The shift value is still 10, and it will ever remain so. The accumulation counter ranges from 0–1023. As noted above, the difference associated with the most recent two averages effectively reflects the drift over a period of 256 cycles. The suitable fraction to use can simply be the accumulation counter divided by the value of 1024. But we must still multiply the difference value by 4. Beyond the 2048-cycle point, however, we have the final situation. The fraction can continue to be the same as before, but we should no longer modify the difference value, as starting with the 12th average the diff always represents the drift over a 1024-cycle period.

Here is a listing the first entries of the log of computed averages of the offsets between the clock event time stamps and the HRM Slow Data time stamps:

```
FILE<0590>             01/14/03 1110
 0590:0097A980   0400 0000 0000 0007 0030 A9BA 01A6 000A
     :0097A990   0000 0000 503B 24D2 5050 0000 02E0 0007
     :0097A9A0   0010 0000 0000 0011 0030 AC70 0000 0012
     :0097A9B0   0030 9718 0000 0000 0301 1411 0139 0108   1.
     :0097A9C0   0030 971F 0000 0007 0301 1411 0140 0007   2.
     :0097A9D0   0030 970F FFFF FFF0 0301 1411 0140 0208   3.
     :0097A9E0   0030 970A FFFF FFFB 0301 1411 0140 0607   4.
     :0097A9F0   0030 9705 FFFF FFFB 0301 1411 0140 1408   5.
     :0097AA00   0030 9703 FFFF FFFE 0301 1411 0142 0507   6.
     :0097AA10   0030 9716 0000 0013 0301 1411 0145 0007   7.
     :0097AA20   0030 972C 0000 0016 0301 1411 0149 0607   8. 128-pt avg
     :0097AA30   0030 975B 0000 002F 0301 1411 0157 1407   9. 256-pt avg
     :0097AA40   0030 97B8 0000 005D 0301 1411 0215 0007  10. 512-pt avg
     :0097AA50   0030 9870 0000 00B8 0301 1411 0332 0807  11. 1024-pt avg
     :0097AA60   0030 9B52 0000 02E2 0301 1411 0440 1207  12. 1024-pt avg
     :0097AA70   0030 9E35 0000 02E3 0301 1411 0549 0207  13. 1024-pt avg
     :0097AA80   0030 A115 0000 02E0 0301 1411 0657 0607  14. etc.
     :0097AA90   0030 A3F8 0000 02E3 0301 1411 0805 1107  15.
     :0097AAA0   0030 A6DA 0000 02E2 0301 1411 0914 0007  15.
     :0097AAB0   0030 A9BA 0000 02E0 0301 1411 1022 0508  16.
```

Each 16-byte entry shows the average offset, the difference from the previous average logged, and the time-of-day the entry was logged. Note that the difference logged increases by a factor of 4 between the 11th and 12th entries as the logic switches into its final mode.

So what is the final logic that implements all this? In summary, after the 12th average, that stored after 2048 cycles, the logic is simple. Use the full difference and multiply by the fraction given by the accumulation count divided by 1024. Before that, after the 11th average, that stored after 1024 cycles, we can compute the fraction in the same way, since the shift value has reached 10, but we must multiply the difference by 4. For each time before that, a common formula can be used. The accumulation counter must be ANDed with a mask formed from the shift value minus 1, which is 8 or less, to form the numerator, and the denominator is also obtained from the shift value minus 1. The diff is multiplied by 4.

```
numer = aop->avgCnt;
if (totalLogged <= (HRM_OFFSET_SHIFT_MAX+1))
      {
      diff *= 4;
      if (totalLogged < (HRM_OFFSET_SHIFT_MAX+1))
            numer &= ((1 << (--shift)) - 1);
      }
sdrp->avgOffset = aop->avgOffset + ((numer * diff) >> shift);
```

The above code may be considered obscure, but so is the logic it implements. That is why it

takes so many words to explain how and why it works.

Further refinement

The above extrapolation logic does not go quite far enough. In the 11th average, for example, the average offset computed reflects the influence of the drift as of the mid-point of the range of the average, in this case the first 1024 points. But it should ideally represent the offset that should apply as of the end of the interval, allowing the above extrapolation to take it from there. This means we should add another term to what is stored in `sdrp->avgOffset`. That term should be half the `diff` value used in the last formula above.

Let us test the veracity of the last assertion. For the use of the formula following the 11th average, it is surely correct. We want to adjust by half the full 1024-point difference. Since the diff has already been multiplied by 4 to get that 1024-point value, it will be ok.

What about after the 10th average? Here, the diff reflects the drift over 128 points, and we need to adjust by the drift of 256 points. The above logic in this case, multiplies the given diff by 4, so it now represents a 512-point drift. Again, taking half this amount gets us our required 256-point adjustment. We can assume the same holds for earlier averages.

What about after the 12th average. Now the diff is the full adjustment of 1024 points. But we need to make a 512-point adjustment to bring it to the end of the interval. Again, taking half the diff is the right amount. The final formula, then, is:

```
sdrp->avgOffset = aop->avgOffset + (diff >> 1) + ((numer * diff) >> shift);
```