

# Scaling a Microsecond Counter

*Method to correct inaccuracy*

Tue, Jul 20, 1999

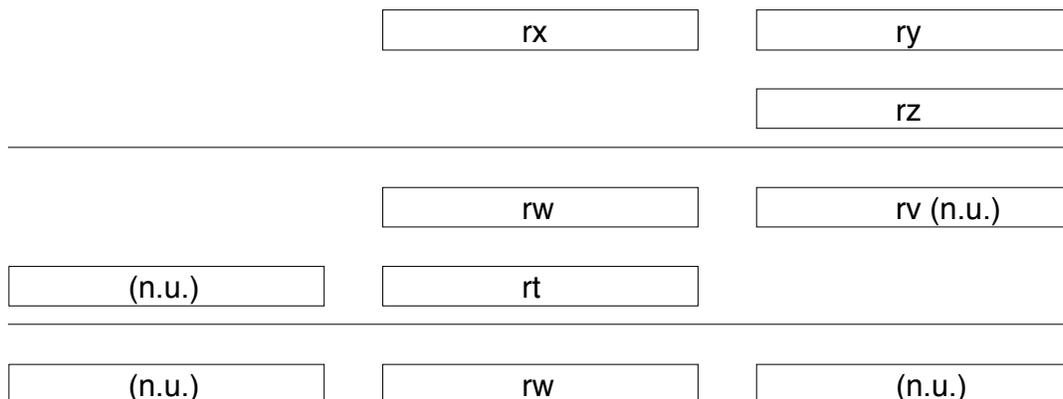
The PowerPC board includes the Hawk multifunction ASIC that includes a 32-bit counter that is meant to count in units of microseconds. But the counter uses another register as a prescaler, which for the case of a 66.67 MHz bus clock, is loaded with an 8-bit value of 66. This causes the counter to increment about 1% faster than it would if it were a true microsecond counter. But we need a more accurate microsecond counter to time-stamp plotting data for the Acnet plotting protocol FTPMAN. The front end captures the time of Tevatron clock events as well as the times of A/D digitizations, so that we can accurately time-stamp all data points in the 64K-byte circular memory buffer on the Analog IP board. The current software assumes the existence of a crystal-accurate megahertz clock, so we need to provide a function that can return times in such units.

Assuming that the above frequencies are the ones we will use, we need to multiply the 32-bit counter value we get by a fraction like 0.99. But simply doing that and taking the low 32 bits of the product is not enough, because there would be a 40-second gap as the 32-bit counter of approximate microseconds rolls over. (This is because the period of a 32-bit microsecond counter is about 4000 seconds.) In order to eliminate the gap, we need to manage more than 32 bits of precision in the counter.

We can use an overflow counter that is incremented whenever the 32-bit raw value rolls over. The logic that performs this "carry" operation should be done at a task level at the lowest priority of any task that needs this microsecond service. It merely checks when a reading of the counter register is less—in an unsigned sense—than the previous reading. This might be done at 15 Hz, or whatever is convenient.

The function that returns the scaled microseconds units first builds a 64-bit counter value by concatenating the overflow counter and the live counter register. It needs to also have access to the previous value of the counter register maintained by the usual carry logic. The logic proceeds as follows: Read the overflow counter as the high-order part of the 64-bit counter value. Read the CTR32 register as the low-order half of the 64-bit counter value. If this reading is less (unsigned) than the previous value, increment the high-order half of the 64-bit result by one.

Apply the fractional correction to the 64-bit value using the appropriate multiply and add instructions needed. Return the 32-bit microsecond result. The following diagram depicts the steps for scaling that are required:



Here, `rz` holds the constant of  $(66/66.67)$ , approximately 0.99, scaled at  $2^{32}$ . The most significant half of the 64-bit counter value is in `rx`; the least significant half is in `ry`. Multiply `ry` and `rz` and store the most significant half of the result in `rw`. Multiply `rz` by `rx` and store the least significant half in `rt`. Return  $(rw + rt)$  as the result. Note that the product of `ry` and `rz` must be the unsigned instruction, as both `ry` and `rz` are unsigned values. The PowerPC instructions are as follows:

```
mulhwu rw,ry,rz #compute upper half of product
mullw  rt,rz,rx #compute lower half of product (rx is likely < rz)
add    rw,rw,rt #sum upper half of 1st product with lower half of 2nd
```

The 32-bit result is in `rw` after the `add` instruction. The result should not exhibit the 40-second gap every hour or so as described earlier. The roll-over problem is handled by the routine when it applies, but the overflow counter must not be modified by the function, even if the occurrence of a rollover is detected, since it may be invoked from a high priority task. Only the low priority task should modify the overflow counter.