AN_651X_026 NOVEMBER 2010

Infrequent Temperature Measurements

71M651x Demo Boards demonstrate temperature measurement techniques using the Demo Code 3.03 and later revisions. Teridian Semiconductor has released several application notes related to temperature measurement and compensation. For example, Application Note 18 (Chop Enable) describes how to achieve accurate temperature measurement using the chopping amplifier for the reference voltage (VREF). However, the technique described in Application Note 18 is based on driving the multiplexer to alternative mode at fixed one-second intervals. It is entirely possible to read temperature much less frequently. By nature, temperature changes are usually very slow, especially in enclosed systems, requiring only infrequent updates from the on-chip temperature sensor.

This application note describes methods for temperature measurement when intervals greater than one second are used.

Data Collection at 1-Second Intervals

Figure 1 shows the *TEMP_RAW_X* CE register information captured over 50,000 samples of data or (50000/2520 = 19.8 seconds). The variation of the data is less than 500 counts, which results in a temperature variation of less than 0.1°C.
This follows from inserting the count difference for *TEMP_RAW_X* in the equation:

$$TEMP = -DEGSCALE * 2^{-22} (TEMP\_RAW\_X - TEMP\_NOM)$$

where the LSB for TEMP is 0.1°C, and *DEGSCALE* is 9585 for the 71M6511

The graph in Figure 1 and the graphs that follow were generated with the RTM (Real-Time Monitor) utility, i.e. using the RTM output of the chip.

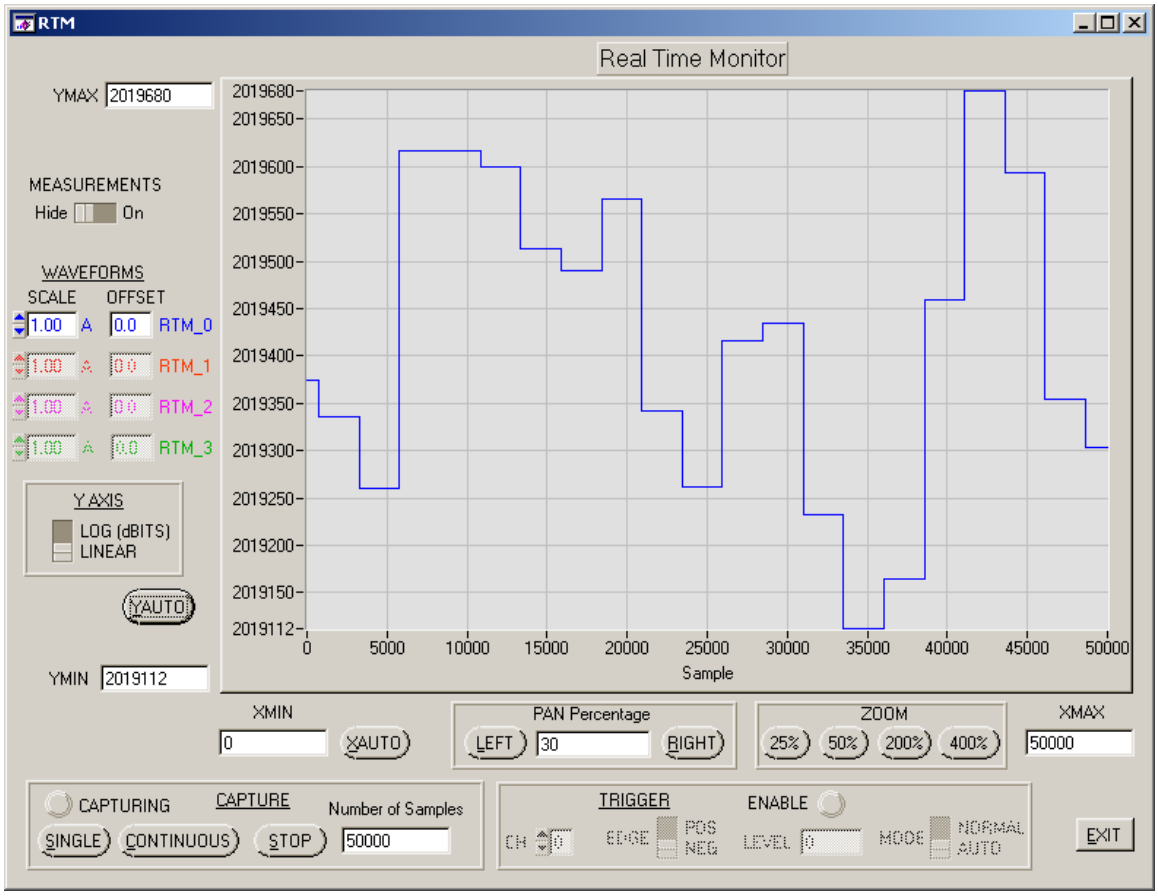

Figure 1: 1-Second Data Collected for 50,000 Samples

Data Collection at Longer Intervals

Figure 2 shows the *TEMP_RAW_X* register information captured over 50,000 samples of data or (50000/2520 = 19.8 seconds) with the interval for temperature measurement extended to four seconds. The variation of the data is now less than 19500, which can result in a temperature variation of more than 2°C. This means that the data collected by the MPU will not be accurate enough for most applications requiring temperature measurement.

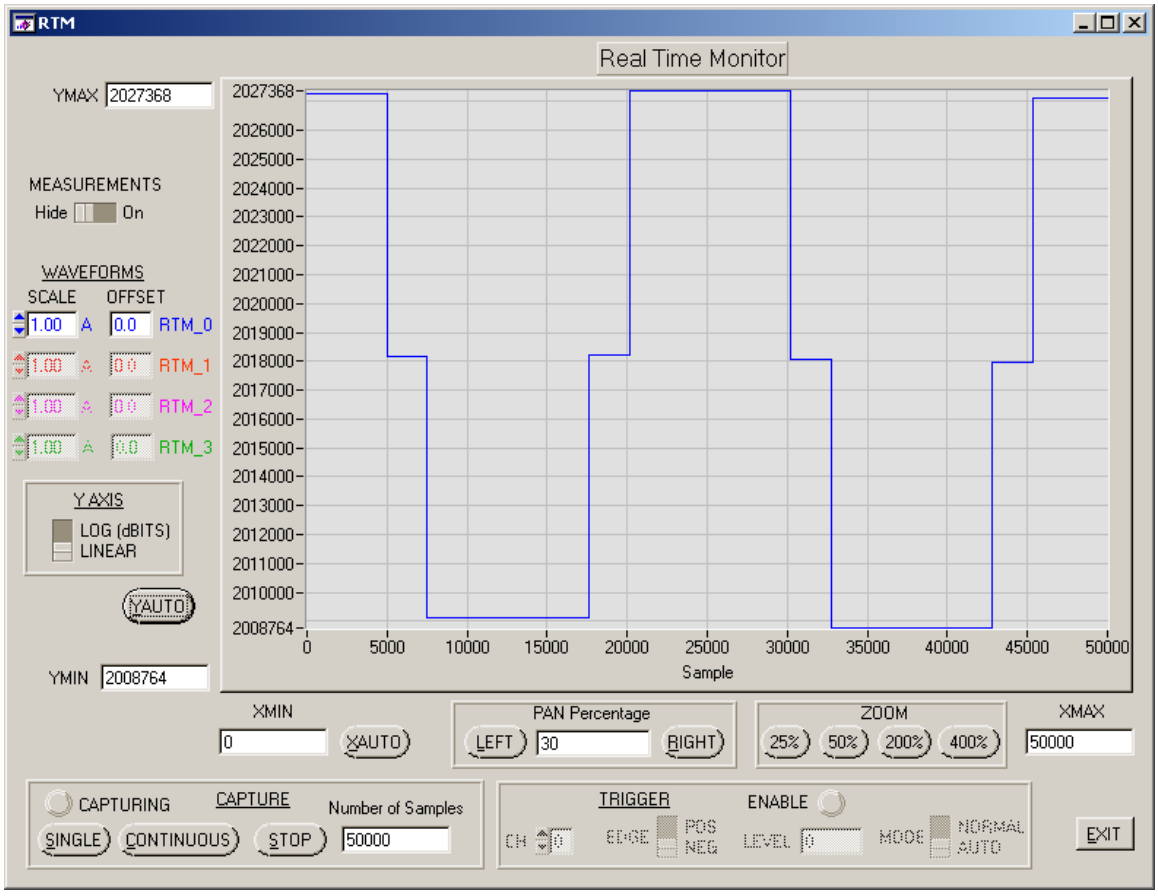

Figure 2: 4-Second Interval for Temperature Measurement

Figure 3 shows the temperature derived from the *TEMP_RAW_X* register, this time with an 8-second interval used for temperature measurement. The inaccuracy can result in variations of 2°C.

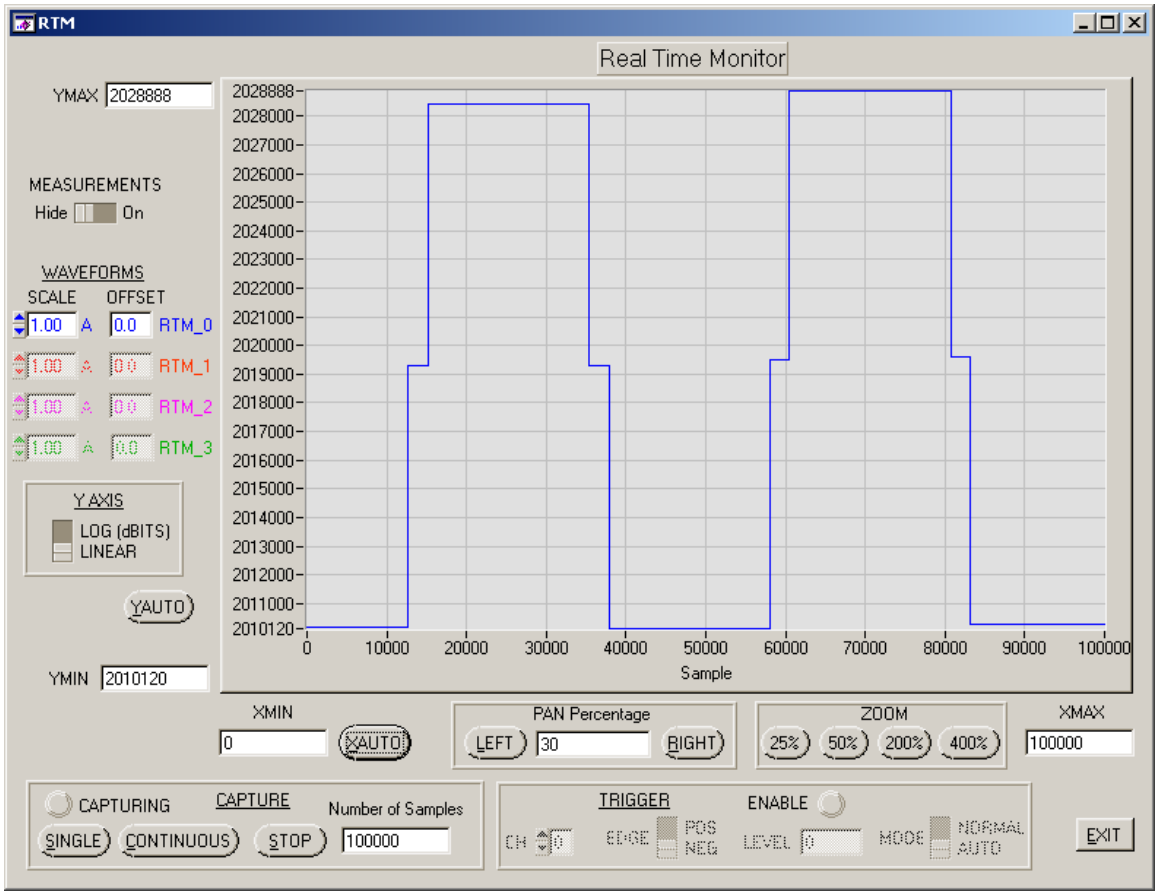

Figure 3: 8-Second Interval for Temperature Measurement

.

Analysis of Temperature Data

Close analysis of the temperature measurement data for 4-second and 8-second intervals leads to the conclusion that the actual issue lies in the exact time when the *TEMP_RAW_X* register is accessed. That is, the Compute Engine (CE) averages the current *TEMP_RAW_X* with the previous sample on every XFER_BUSY cycle irrespective of the MPU driving the multiplexer into alternate multiplexer mode for temperature measurement. The CE simply cannot be synchronized with the status of the *ALT_MUX* bit since it is not aware of when an alternative multiplexer conversion occurs.
In other words, random reads of the *TEMP_RAW_X* register will yield inaccurate data, since the CE updates this register all the time, while correct data is only present when the MPU controls the multiplexer and chopper circuitry correctly.

The temperature values shown in the two graphs (Figure 2 and Figure 3) for the one-second interval immediately after each 4 or 8-seconds interval are actually close to the data that can be obtained with the 1-second interval for temperature measurement.

Suggested Solutions

Two possible firmware solutions for the accurate temperature measurement are presented below.

Firmware Solution 1

This solution involves reading the *TEMP_RAW_X* register at exactly the desired interval and calculating the average of this reading with the previous reading. This method will provide accurate temperature measurement similar to the one-second accumulation interval data.

Firmware Solution 2:

This solution involves reading the *TEMP_RAW_X* register at exactly one XFER_BUSY cycle after driving the alternate multiplexer cycle for temperature measurement. To show the proof of concept, one can read the data one XFER_BUSY cycle after driving the alternate multiplexer cycle and store it to CE data RAM (CE DRAM), where it can be captured using the RTM utility.
This can easily be implemented in the firmware as shown in the source code example below. Code and comments added for temperature measurement are shown in **bold** letters.

Please refer to the CE.C file in the Demo Board firmware (revision 3.04 or later).

```
#pragma save
#pragma REGISTERBANK (XFER_BANK)
void xfer_busy_isr (void) small reentrant
{
    EA = 0; // disable other interrupts for the following critical section

    // alternate polarity to chop temperature, which is read once per sumcycle
    CE2 = (CE2 & ~CHOP_EN) | chop; // start fixed chop on next cycle
    chop ^= CHOP_EN;               // Toggle chop between 1 (01b) and 2 (10b)

    // enable next CE busy to restore hardware chopping
    IEX_CE_BUSY = 0;
    EX_CE_BUSY = 1;

    if(++Alt_Cnt == 8)  // Alt_Cnt is the number Xfer_cycles to drive the mux to
                        // Alt_mux mode
       {
           if (meter_alt)
                    mux_alt ();   // Do Alternate MUX cycle.

       }
    EA = 1;  // enable other interrupts, the critical section is ended

    // on the first two passes, the CE's calculations are incorrect
    // so the results are not worth recording.
    if (0 == ce_first_pass)
    {
       U08x *pDst;
       U08x *pSrc;
       U08  len;

       // set the pulse source values, and check for errors
       if (PulseWSource > INTERNAL)
       {                              // Is APULSEW ready before next XFER cycle?
          if (0 == apulsew_written)
          {
             Status |= PULSEW_ERR;   // No, it's too late!
          }
          else if (2 == apulsew_written)
          {                           // Update CE copy w/ buffered one.
             pDst = (U08x *) &apulsew; // CE copy.
             pSrc = (U08x *) &ApulseW; // Buffered copy.
             MEMCPY_CEX;               // Copy single 32-bit word from CE to XRAM.
             apulsew_written = 0;
          }
          else
             apulsew_written--;
       }

       // set the pulse source values, and check for errors
       if (PulseRSource > INTERNAL)
       {                              // Is APULSER ready before next XFER cycle?
          if (0 == apulser_written)
             Status |= PULSER_ERR;   // No, it's too late!
          else if (2 == apulser_written)
          {                           // Update CE copy w/ buffered one.
             pDst = (U08x *) &apulsew; // CE copy.
             pSrc = (U08x *) &ApulseW; // Buffered copy.
             MEMCPY_CEX;            // Copy single 32-bit word from CE to XRAM.
             apulser_written = 0;
          }
          else
             apulser_written--;
```

```
    }
```

```
       // copy CE values to the MPU
       pDst = (U08x *) &Outputs;
       pSrc = (U08x *) &CE.Outputs;
       len  = sizeof (CE.Outputs) / sizeof (S32);

       do
       {
          MEMCPY_XCE;             // Copy single 32-bit word from CE to XRAM.
          NEXT_XX;                // Point to start of next Dst & Src.
       } while (--len);

      // The data is transferred to MPU is complete.

      if (Alt_Cnt == 9)  // Since Alt_Cnt is the number of XFER_BUSY cycles to drive the
                         // MUX to alternate mode for temperature, the raw data will
                         // be available after Alt_Cnt + 1 XFER_BUSY cycles.
            {
                 memcpy_cex_1 (&phadj_2,&temp_raw);
                              // Raw temperature data read at the alternative mux
                              // cycle is at phadj_2
                Alt_Cnt =0; // reset counter
            }

       outputs_ready = TRUE;
       xfer_update = TRUE;        // inform the main loop to process the data

       Apply_Creep_Threshold ();

#ifndef M6511
       if (xfer_busy_beat)
          DIO_1 ^= 1;          // Toggle DIO_1.
#endif
    }
    else
       ce_first_pass--;        // count CE passes after startup
}
```

Results from measurements using the above source code are pictured on the following pages.

The graph in Figure 4 was obtained using 8-second intervals with the firmware modification presented above. Even though the difference between minimum and maximum readings is 18,288 counts, the data obtained every 8th XFER_BUSY interval (red line) has much less variation and results in a temperature error of less than 0.1°C.

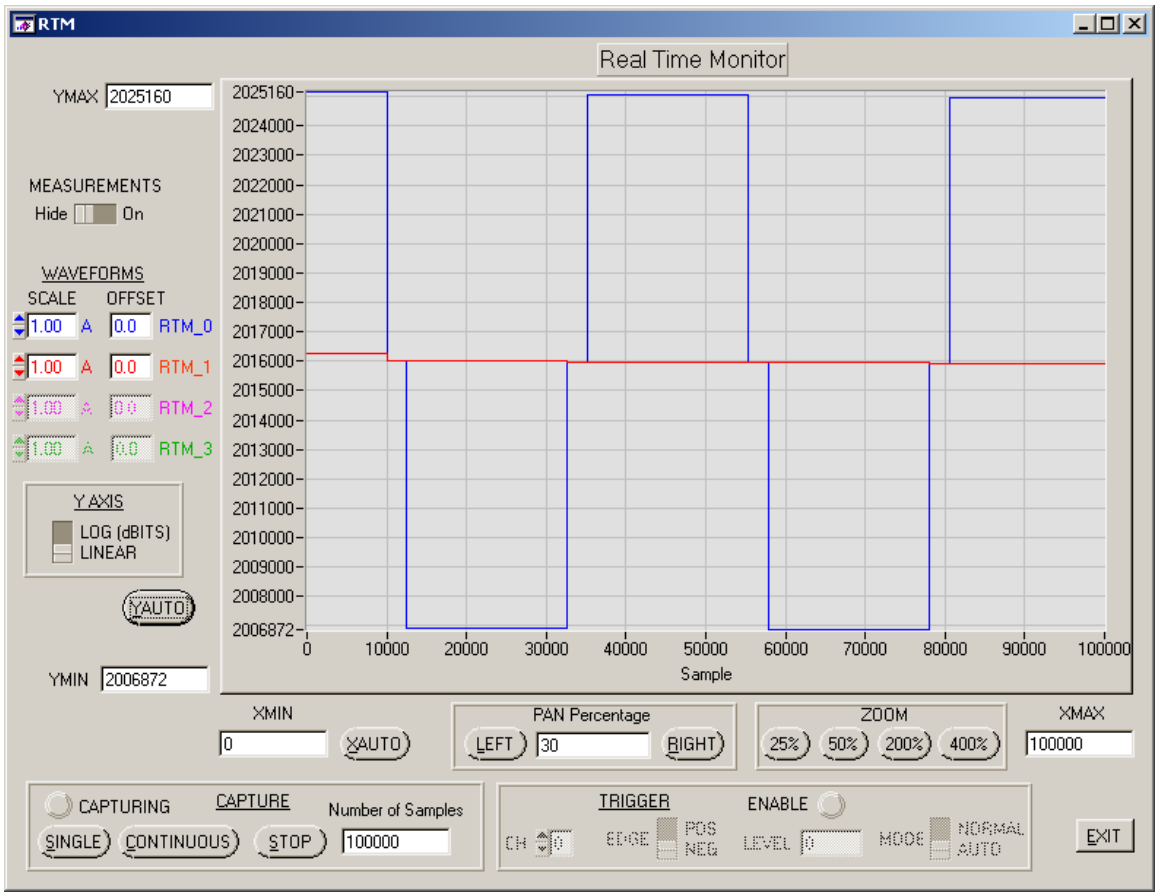

Figure 4: 8-Second Interval for Temperature Measurement

Figure 5 shows a closer look of the *TEMP_RAW* register value using the modified firmware.

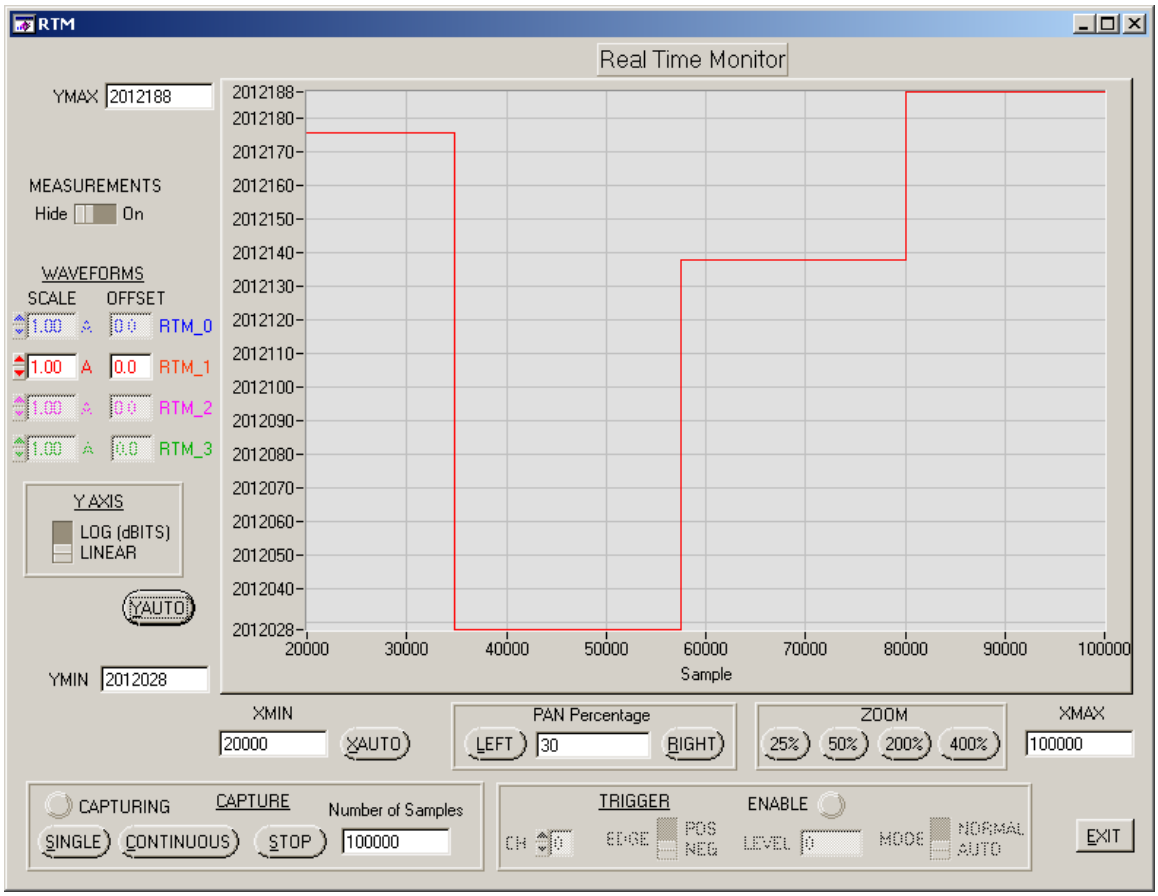

Figure 5: Temperature Measurement with Modified Firmware (Detail)

Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600