AB_651X_001

APPLICATION BRIEF

APRIL 2006

Application Brief

This document describes several issues with Demo Code revision 3.04 (and earlier) that could affect meter operation if the original 3.04 sources are used to generate operational meter firmware. **These issues should be resolved by upgrading the critical portions of the code to revision 3.05, before releasing meter firmware**.

A complete list of 3.05 issues is published in the 651X Software User's Guide (SUG). This Application Brief covers only the critical issues that could affect meter operation.

More Application Briefs will be published by TERIDIAN Semiconductor if other critical issues should be discovered.

In addition to firmware issues, a hardware note is discussed in this document.

Firmware Issues

Interrupt Sharing between RTC and XFER_BUSY

The RTC_1SEC interrupt and the XFER_BUSY interrupt from the CE are combined using a logic OR function and the resulting signal is routed into interrupt 6 of the MPU. Therefore, both flags must be cleared during initialization, and both flags must always be cleared before exiting the interrupt service routine (ISR) for interrupt 6.

If both flags are not cleared, no edge can occur to trigger interrupt 6 at a later time, which results in the ISR for the XFER_BUSY interrupt ceasing to run. This condition will basically block any metering from occurring.

Similarly, clearing both flags requires some care: Either flag could be set by the hardware while the ISR is running on behalf of the other interrupt. In this situation, the unprocessed interrupt can create a lockout situation similar to the one mentioned above. To prevent this lockout, both interrupt flags must always be processed in the same ISR.

The code fragment below shows proper implementation of the ISR for the XFER_BUSY and RTC interrupts.

```
void xfer_rtc_int (void) small reentrant interrupt XFER_RTC_IV
{
    do
    {
        if (IE_XFER)    // check if XFER_BUSY interrupt
        {
            CLR_IE_XFER();    // clear flag

            EA = 0;  // disable interrupts for the following critical section
                     // code dealing with chop bits of ADC should be inserted here
            EA = 1;  // enable other interrupts, the critical section is ended

            if (0 == ce_first_pass)
            {
             // code copying CE data to RAM should be inserted here.
            }
            else
               ce_first_pass--; // wait for the CE's digital filters to settle
        }
        if (IE_RTC)     // check if RTC interrupt
        {
            CLR_IE_RTC();     // clear flag
                      // the once-per-second RTC code should be inserted here
        }
    }
    while (IE_RTC || IE_XFER); // stay in this loop as long as there are
                               // flags to process.
}
```

Interrupt Starving

The use of distributed flags to lock interrupts is discouraged. It is recommended to use one central flag, such as **EA (SFR bit)** used in the code example above.

If this rule is ignored, a low-priority interrupt can lock out a high-priority interrupt. A typical issue caused by this lock out mechanism is when a low-priority interrupt, such as the RTC, blocks a high-priority interrupt, such as the serial port. It will result in loss of bytes received via the serial port.

Display Rollover Issue

The rollover of energy displays is not handled properly by Demo Code revision 3.04 or earlier versions (shown below). The core of the problem is the `divide_1()` routine contained in the file `classics.c`. The condition `*u > v` fails to check equality between dividend and divisor. When *u equals v, the two instructions following the condition must also be executed. Failure to fix this issue will occasionally result in numbers not adding up properly from values below 1000.

```
void divide_1 (U08x *u, U08 v, U08 n)
{                                       // (n-byte) u /= (n-byte) v;
    U08_16 t;

    if (v > 1)
    {                                   // Proper divisor.
       t.c[ HI ] = *u;                  // Get current MSB of dividend.

       if (*u > v)                      // ignores the case where *u == v
       {                                // MSB of quotient not zero (0).
          *u /= v;                      // Compute partial quotient.
          t.c[ HI ] -= *u * v;          // Subtract partial product from dividend.
       }
         else
              *u = 0;                     // MSB of quotient is zero (0).

       if (n > 1)
       {                                // Dividend at least two bytes wide.
          n--;
          do
          {                             // Now *u < v always.
             t.c[ LO ] = *++u;          // t = u-1:u.
             *u = t.i / v;              // u = t / v; Compute partial quotient.
             t.i -= *u * v;             // Subtract partial product from dividend.
                   t.c[ HI ] = t.c[ LO ];       //
          } while (--n);
       }
    }
}
```

The code shown below is the source code used in Demo Code revision 3.05. The discussed condition has been fixed to check for equality, also (`if (*u >= v)`).

```
void divide_1 (U08x *u, U08 v, U08 n)
{                                       // (n-byte) u /= (n-byte) v;
    U08_16 t;

    if (v > 1)
    {                                   // Proper divisor.
       t.c[ HI ] = *u;                  // Get current MSB of dividend.

       if (*u >= v)
       {                                // MSB of quotient not zero (0).
          *u /= v;                      // Compute partial quotient.
          t.c[ HI ] -= *u * v;          // Subtract partial product from dividend.
       }
         else
              *u = 0;                     // MSB of quotient is zero (0).

       if (n > 1)
       {                                // Dividend at least two bytes wide.
          n--;
          do
          {                             // Now *u < v always.
             t.c[ LO ] = *++u;          // t = u-1:u.
             *u = t.i / v;              // u = t / v; Compute partial quotient.
             t.i -= *u * v;             // Subtract partial product from dividend.
                   t.c[ HI ] = t.c[ LO ];       //
          } while (--n);
       }
    }
}
```

Hardware Note

Connection of the V3 Pin (71M6513)

The V3 pin should be left unconnected or it should be connected to the VREF pin. Other connection schemes may result in loss of accuracy of the temperature measurements.

TERIDIAN Semiconductor Corp., 6440 Oak Canyon Rd., Irvine, CA 92618

TEL (714) 508-8800, FAX (714) 508-8877, http://www.teridian.com

