Piccolo MCU CAN Module Operation Using the On-Chip Zero-Pin Oscillator

Mark Labbato

ABSTRACT

CAN protocol imposes rather stringent requirements on its timing source, which are generally met by the provision of an external quartz crystal/oscillator. The TMS320C2803x/TMS320F2806x series of microcontrollers have an on-chip zero-pin oscillator that needs no external components. This application report describes how to use the CAN module with this oscillator to operate at the maximum bit rate and bus length without the added cost of an external clock source. The experimental setup was tested over the entire operating temperature range of the device.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/sprabi7.

Contents

1 Introduction

The CAN module on the Piccolo™ series of devices can operate at up to 1 Mbps. Such high bit-rates across the maximum demand a precision clock source. Accuracy needs to be maintained regardless of the thermal conditions in order to comply with timing requirements imposed by the CAN protocol. Without compensation, the internal oscillator's drift across temperature could negatively impact CAN communication. One solution is to use an external clock source, which adds to overall system cost and complexity. This document explains how to implement CAN protocol using the internal oscillator with compensation for the frequency drift due to temperature. The oscillator compensation method is provided in the header files and a description of the function can be found in the *Oscillator Compensation Guide* (SPRAB84).

2 Experiment

Using the Oscillator Compensation function provided in the header files, the CAN module was able to transmit across a 40 meter cable at full 1 Mbps speed, across the entire temperature range of the device. This was done by connecting two TMS320F2803x devices with a 40 meter cable. The receiving node waits in a loop that uses the compensation function and acknowledges any valid transmission across the bus. The transmitting node sends a message at 1 Mbps. The transmitter toggles GPIO0 with every successful transmission. The transmitter also polls the temperature sensor and then uses the Oscillator Compensation function with every transmission.

Once the system is running, the transmitter is raised to 125°C while the receiver is lowered to -40°C. The transmitter continues to transmit successfully, even while the transmitter and the receiver are on opposite ends of the operating temperatures outlined in the spec. While still running, the transmitter is dropped to -40°C while the receiver is raised to room temperature. Proper communication is observed across the operating temperature range of the device, if the oscillator compensation function is used.

Piccolo is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

3 Implementation

The ADC temperature sensor should be sampled at a rate that fulfills the needs of the application in which it is used. If the temperature of the system changes slowly, the compensation function can be called once every few minutes. However, if the ambient temperature fluctuates rapidly, the oscillator may have to be calibrated more frequently. This can be done by having a CPU timer generate an interrupt that does the calibration, or by just placing the necessary code into the main application loop.

The following is the main loop of the provided CAN_Osc_Comp.c file:

```
while(1)
{
       /************** Application code goes here*************/


       // Sample temperature sensor. Note: the end application
       // will be responsible for deciding how and when to sample
       // the temperature sensor so that the value can be
       // passed to the compensation function

       //Force start of conversion on SOC0 and SOC1
       AdcRegs.ADCSOCFRC1.all = 0x03;

       //Wait for end of conversion.
       while(AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){}  //Wait for ADCINT1
       AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;        //Clear ADCINT1

       //Get temp sensor sample result from SOC1
       temp = AdcResult.ADCRESULT1;

       //Use temp sensor measurement to perform oscillator
       // compensation even as temperature changes.
       Osc1Comp(temp);

}
```

Based on the conditions of this experiment, it was necessary to sample the temperature sensor as often as possible. In order to do so, the ADC must be commanded to begin a conversion.

```
AdcRegs.ADCSOCFRC1.all = 0x03;
```

This is done on both SOC0 and SOC1 in order to get an accurate reading.

After the conversion is triggered, the device waits until an end of conversion interrupt is generated, signaling that there is a valid sample in the ADC result registers. The interrupt bit was also reset so that it could be used again in the next iteration of the loop.

```
while(AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){}

AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      //Clear ADCINT1
```

Since the result of the temperature sensor sampling is stored in the ADC result register, the value can be saved as the current temperature or *temp*.

```
temp = AdcResult.ADCRESULT1;
```

Once the temp is saved, it can be passed to the oscillator compensation function.

```
 Osc1Comp(temp);
```

This function uses the new temperature reading to adjust the fine trim value of the oscillator. Because of the application, this function was called in each iteration of the loop in order to keep the temperature as accurate as possible.

4 References

- *Oscillator Compensation Guide* (SPRAB84)
- *Controller Area Network* by Konrad Etschberger. IXXAT Press, 2001.
- *A Comprehensible Guide To Controller Area Network* by Wilfried Voss. Copperhill Technologies, 2005.

IMPORTANT NOTICE