Application Report

Implementation of CRC for ADS7066

ABSTRACT

A cyclic redundancy check (CRC) is a common method to detect errors within digital communication. The use of CRC is found in applications where there are higher levels of safety requirements. Motor drivers and factory automation are examples of where CRC is implemented. This document explains what a CRC code is and how it is implemented by using the ADS7066 device as an example.

The ADS7066 device features a bidirectional CRC module. When CRC is enabled, the ADS7066 sends data with a CRC byte appened to the host to be evaluated by the host. For incoming data to the ADS7066, the host will send the data with a CRC byte appended to be evaluated by the ADS7066.

For each data frame, a CRC code is calculated based on the generator polynomial and the data payload in the frame. The ADS7066 implements the CRC-8-CCITT polynomial, shown in *ADS7066 Register Read Timing Diagram With CRC Enabled*, and is pre-set with 1 data values(1111 1111b). The host and ADC append a CRC code on transmitted data. Each device also computes that the CRC code received is correct based on the data received. This allows for instant detection of communication errors over the SPI bus.

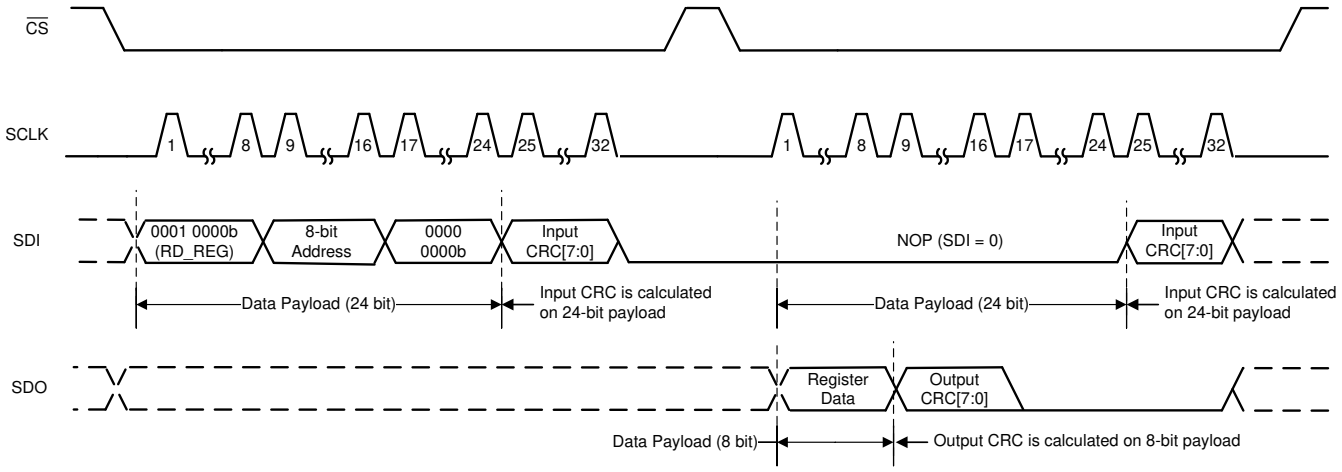
$$CRC\text{-}8\text{-}CCITT = X^8 + X^2 + X^1 + 1 \quad (1)$$

When the CRC module is enabled in the ADS7066 device, the communication bus will need to use 32-bit SPI frames, usually comprised of a 24-bit data payload and an 8-bit CRC byte; zeros should be used to fill the 32-bit data frame if needed. The timing diagram in Figure 1-1 illustrates a completed data transfer of the ADS7066 CRC of a typical register read with CRC enabled.

Due to the elongated communication frames, the throughput rate is decreased. Each frame will be eight clock pulses longer than expected with CRC enable. For example, without CRC enabled, the clock rate needed to achieve a sampling rate of 500 kSPS would be 8 Mhz. This is based on only outputing conversion data without any status flags included, which means one frame is completed with 16 SCLK pulses. When CRC is enabled, the SPI frame now needs to be 32 SCLK pulses. If the clock frequency is maintained at 8 Mhz, then the sampling rate has now decreased from 500kSPS to 250 kSPS. This is due to the longer SPI frames needed. To achieve the target sampling rate of 500 kSPS with CRC enabled the clock rate will need to be increased from 8 Mhz to 16 Mhz to compensate for the longer SPI frame.

After the data integrity check of recomputing the CRC calculation and there are no errors detected, then all commands continue as expected.

If a CRC error occurs, meaning an error in the expected CRC value does not match the CRC value transmitted, then the ADS7066 device will not execute the command where the error was detected. Thereafter, the device will no longer respond to any register writes, but will respond to read commands and provide ADC conversion measurements. During this time, correct CRC codes are still needed for read commands sent to the device. When the error is detected, the device will set a status flag that notifies the user a CRC error has occured. The device will return back to normal operation once the status flag is cleared. Until the CRC status error flag is cleared the device will not respond to any write commands other than a write command to clear the CRC error status flag.



ADS7066 Register Read Timing Diagram With CRC Enabled

For more details on this, see the CRCERR_IN register in the *ADS7066 Small, 8-Channel, 16-Bit SAR ADC With GPIOs Data Sheet*. To help calculate the CRC code for the ADS7066 device, TI provides a *ADS7066 CRC Calculator* available in the ADS7066 Product Folder. For further reading about CRC and other data inegrity methods, see the *Communication Methods for Data Integrity Using DeltaSigma Data Converters Application Report*.

Example Code

The following is a short software example of how to implement CRC for the ADS7066 device in the C language.

```
//*****************************************************************************
//! Calculates the 8-bit CRC for the selected CRC polynomial.
//! \fn uint8_t calculate CRC(const uint8_t dataBytes[], uint8_t numberBytes, uint8_t initial Value)
//! \param dataBytes[] pointer to first element in the data byte array
//! \param numberBytes number of bytes to be used in CRC calculation
//! \param initialValue the seed value (or partial crc calculation), use 0xFF when beginning a new
CRC
computation
//! NOTE: This calculation is shown as an example and is not optimized for speed.
//! \return 8-bit calculated CRC word
//*****************************************************************************
uint8_t calculateCRC(const uint8_t dataBytes[], uint8_t numberBytes, uint8_t initialValue)
{
// Check that "dataBytes" is not a null pointer
assert(dataBytes != 0x00);
int bitIndex, byteIndex;
bool dataMSb; /* Most significant bit of data byte */
bool crcMSb; /* Most significant bit of crc byte */
// Initial value of crc register
// NOTE: The ADS7066 defaults to 0xFF,
// but can be set at function call to continue an on-going calculation
uint8_t crc = initialValue;
// ANSI CRC polynomial = x^8 + x^2 + x^1 + 1
const uint8_t poly = 0x07;
/* CRC algorithm */
// Loop through all bytes in the dataBytes[] array
for (byteIndex = 0; byteIndex < numberBytes; byteIndex++)
{
// Point to MSb in byte
bitIndex = 0x80u;
// Loop through all bits in the current byte
while (bitIndex > 0)
{
// Check MSB's of data and crc
dataMSb = (bool) (dataBytes[byteIndex] & bitIndex);
crcMSb = (bool) (crc & 0x80u);
www.ti.com
2 Implementation of CRC for ADS7066 SBAA456 – JULY 2020
Submit Document Feedback
```

```
Copyright © 2020 Texas Instruments Incorporated
crc <<= 1;
// Check if XOR operation of MSBs results in additional XOR operations
if (dataMSb ^ crcMSb)
{
crc ^= poly;
}
// Shift MSb pointer to the next data bit
bitIndex >>= 1;
}
}
return crc;
}
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265