Application Note

A-Format Absolute-Encoder Primary Interface for F280039C

Hang Yang *Industrial ASM*

ABSTRACT

This application note provides guidance on leveraging the configurable logic block (CLB) of the real-time C2000 MCU to implement A-Format encoder interface. In the proposed implementation, the CLB modules are used to manage the SPI transmit/receive sequence, generate SPI clock signals, and calculate the receive (RX) cyclic redundancy check (CRC) on the fly. The SPI is used for data transfer and receive.

This application note includes details of SPI and CLB implementation and the results of validation tests.

Table of Contents

Trademarks

1 Introduction

In motor control applications, absolute encoder is commonly used in the scenario where high precision motor position sensing is required. A-Format is the interface protocol for Nikon absolute-encoder.

One of the focuses of TI's C2000 microcontrollers is motor control. The C2000 has unique configurable logic block (CLB) modules to support encoder interfaces for motor control applications. The CLB include various logic blocks that can be customized using the CLB tool chain. TI provides several reference designs that leverage the CLB to implement encoder interfaces to facilitate customer applications. One of them is the TIDM-1011 Tamagawa T-Format interface reference design (see References [1]). The T-Format interface is similar to the A-Format interface. The A-Format interface can be implemented by modifying the T-Format interface's software configurations (including SPI parameters, CLB logic, and frame formatting) without hardware changes.

The aim of this application note is to provide guidance on how to implement the A-Format interface by modification from the T-Format interface.

2 System Description

The A-Format interface implementation is based on the TIDM-1011 design. The implementation was validated using the same hardware platform: LAUNCHXL-F280039C (C2000 LaunchPad), BOOSTXL-POSMGR (position manager booster pack), and a Nikon A-Format encoder (model: Nikon-MX50AHN00, replacing the Tamagawa encoder in TIDM-1011). When adapting T-Format interface to the A-Format, only software modifications are required, including: 1) SPI peripheral configuration (FIFO width/depth, clock polarity/phase); 2) CLB logic customization (CRC calculation, transmit/receive sequence control); 3) A-Format frame formatting (18-bit field splitting, command/response parsing).

The A-Format interface uses SPI for data transfer and receive. CLB is used to manage the transfer/receive sequence of SPI as well as calculate the RX CRC on the fly. A simplified block diagram of the interface is shown in the following figure.

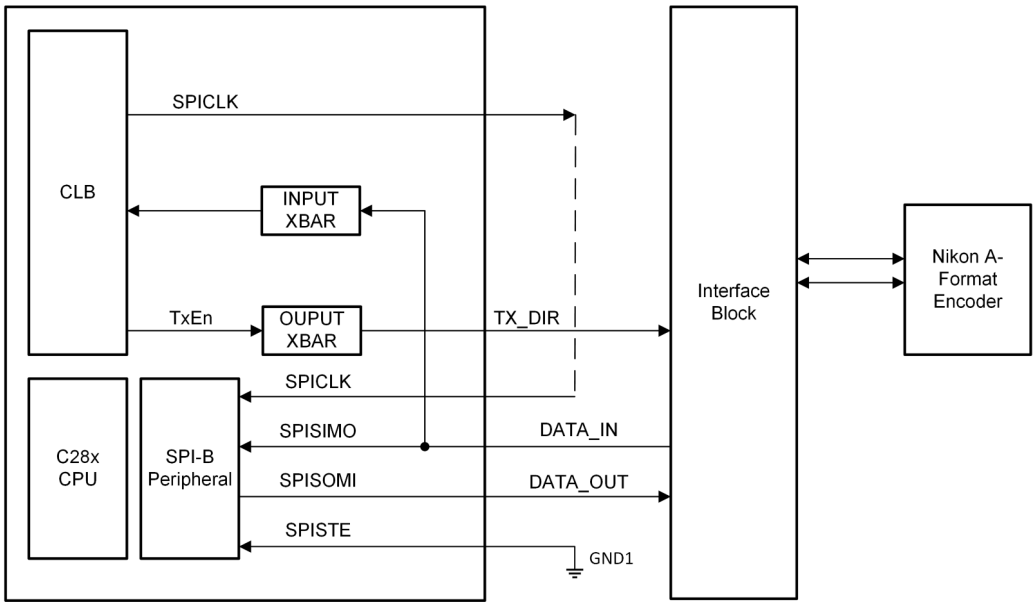

Figure 2-1. System Diagram

2.1 SPI Implementation

The A-Format frames are transferred via the SPI peripheral. For each communication cycle, the command fields are first filled into the TX FIFO, then the encoder response is received in the RX FIFO.

The fields in A-Format are 18-bit. To fill the 18-bit fields into the SPI FIFO, each field is split into two 9-bit half-fields. Accordingly, the width of the FIFO is set to 9 bits. The following figure illustrates the splitting of 18-bit A-Format fields into 9-bit SPI FIFO entries. Taking the CDF0 command as an example: the controller sends one 18-bit field to the encoder, and the encoder responds with three 18-bit fields. Therefore, the depth of the TX FIFO is set as 2 and the depth of the RX FIFO depth is set to 6 entries

The number of fields to transfer and receive may vary depending on the command. The user should change the depth of the TX and RX FIFO accordingly. When adjusting the FIFO depth based on different commands, the SPI peripheral configuration (FIFO depth register) and CLB logic (transmit/receive sequence control) must be modified consistently to ensure synchronization.

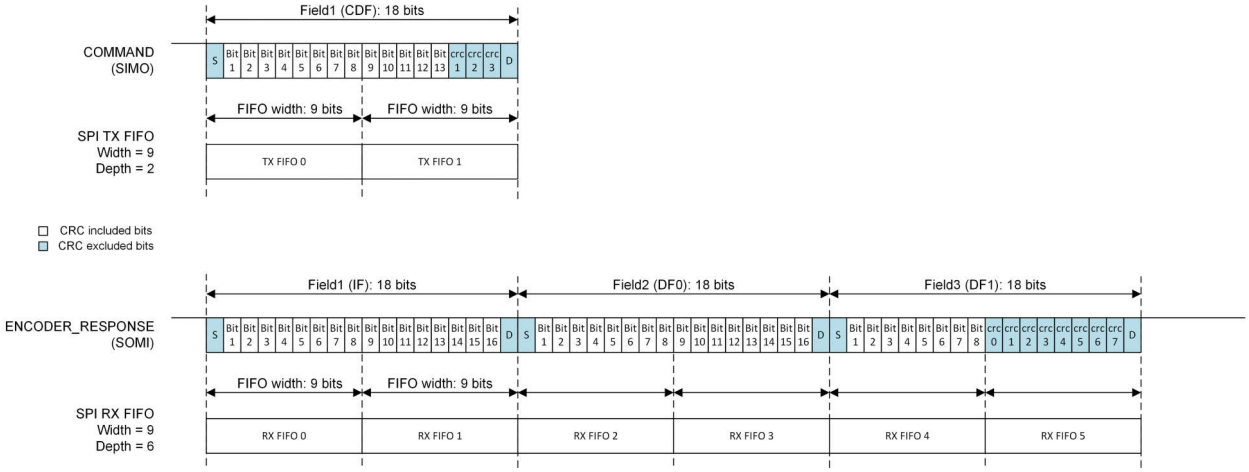

Figure 2-2. Splitting A-Format Fields to SPI FIFO

2.2 CLB Implementation

There are two CLB tiles in this solution, one controlling the SPI timing, the other calculates the RX CRC on the fly. The SPI timing CLB tile would be compatible with the one in the T-Format solution, and therefore no change is required. However, the RX CRC CLB tile would require changes since the field format and the CRC polynomial is different. The following describes the CLB implementation for CRC calculation. For the rest of the CLB functions, one may refer to the T-Format design guide.

The RX CRC is calculated using the counter block in Linear Feedback Shift Register (LFSR) mode. The RX bits are shifted in the LFSR upon each SPI clock, and the CRC result can be read from the LFSR once the last bit is shifted in. Figure 2-3 marks the bits that should be shifted into the LFSR. The CLB logic implements a bit mask filter to select the specific bits that need to be shifted into the LFSR for CRC calculation, the mask is generated based on the below two rules:

- The middle 16 bits of every 18 bits
- The first n bits

where n is the total frame length excluding the CRC fields and the delimiter at the end. The signal used to mask the CRC bits are named as CRC_MASK and DATA_VALID.

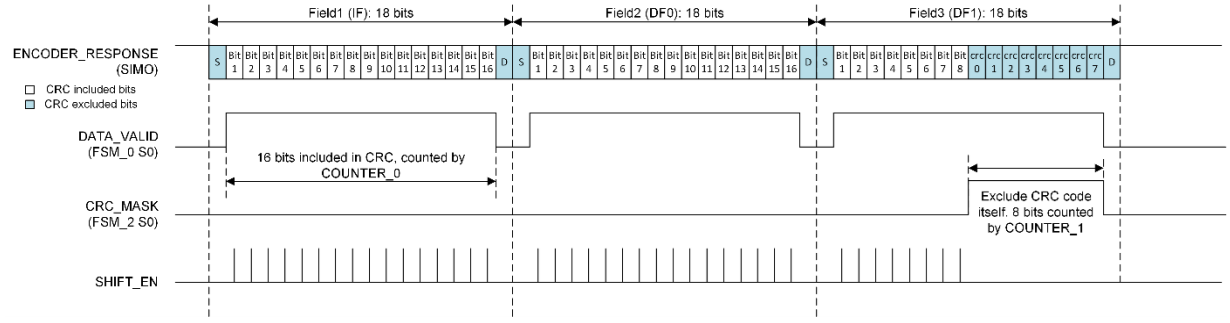

Figure 2-3. Generating Mask for CRC Calculation in CLB

Since the mask rules are different to T-Fomat, the CLB logic for SHIFT_EN signal generation is changed as well. Two counters and one FSM are used to generate the SHIFT_EN signal. Figure 2-4 shows the block diagram of the CLB tile for CRC calculations. The block diagrams are mostly identical to the T-Format implementation, except for SHIFT_EN signal generation logic, which includes COUNTER0, COUNTER1 and FSM2 and LUT1.

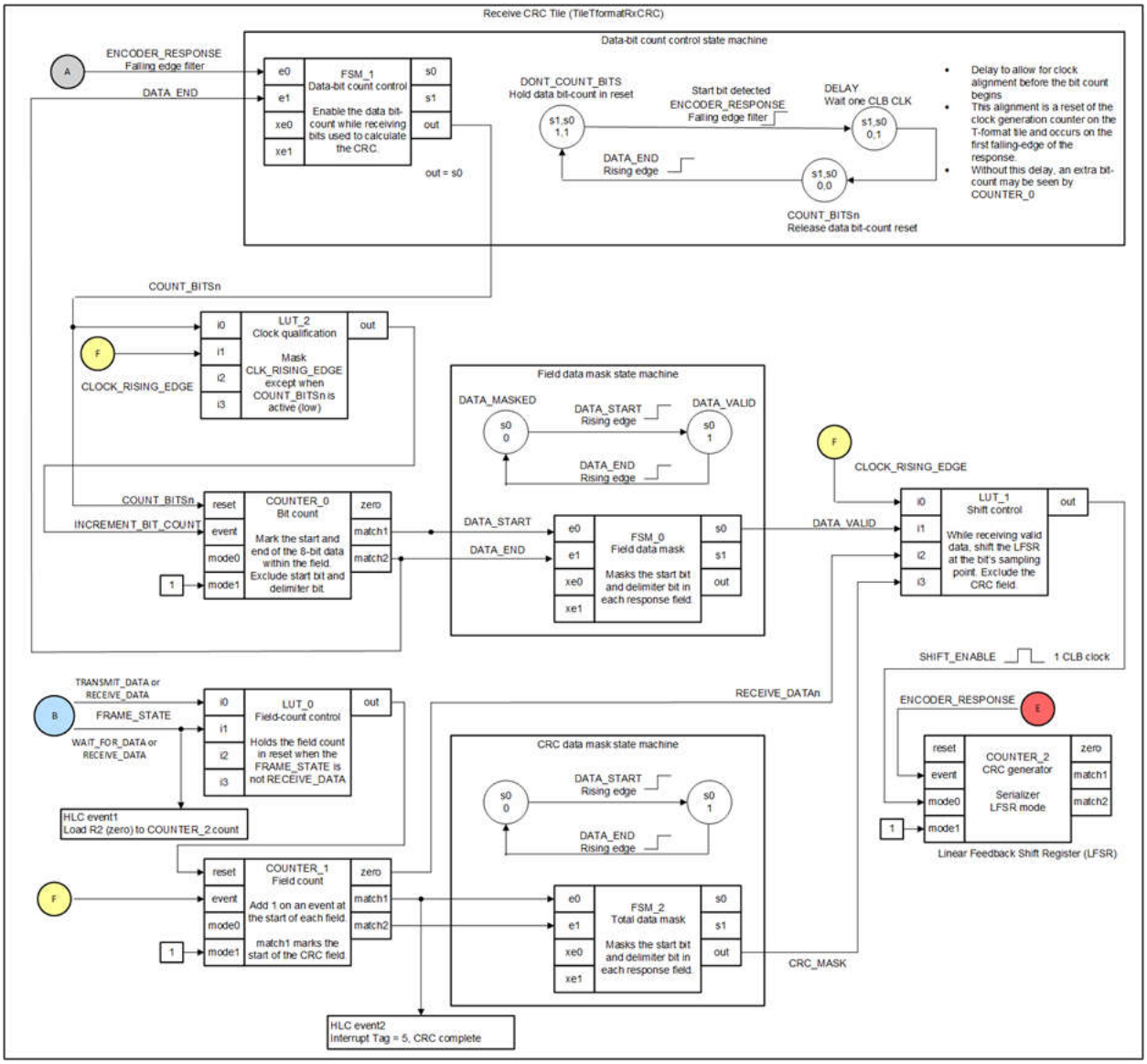

Figure 2-4. CLB Diagram for CRC Calculation

The above logics are implemented using the TI CLB Tool Chain (Rev. B, see References [2]). For the usage of the tool chain, one may refer to CLB Tool User's Guide (Rev. B). A quick way to verify the logic is through simulation, which is part of the tool chain. Figure 2-5 shows an example simulation of the CRC calculation. The response from encoder is input as “Boundary input 4” in the waveforms. As the response data comes in, the SHIFT_EN signal is generated as “Counter2_mode0”. This signal marks the bits included in the CRC calculation. The response data is shifted into the LFSR, which is marked as “counter_2”, upon every pulse of the SHIFT_EN signal. As the last SHIFT_EN pulse comes in, the LFSR holds a value of 0x1576608F, where the final 8 bits, 0x8F, represent the desired CRC result.

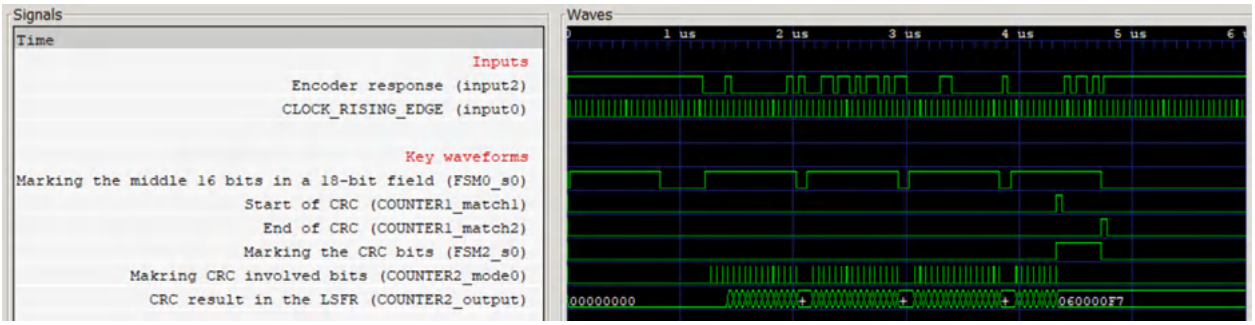

Figure 2-5. Simulation of CRC Calculation Logic in CLB

2.3 Software Changes

Most of the software in the T-Format solution can be reused in the A-Format interface. Based on the above SPI and CLB implementation, there are changes in the software.

The first is the tformat_configureCLBLen() API. This function is used to configure the CLB counters with the proper number of transfer and receive clocks which may vary among different commands. In the A-Format CLB implementation (For CRC calculation), the CRC mask depends on the number of clocks of the encoder response, and hence two extra configurations are added to this API. The two lines configured COUNTER_1_MATCH_1 and COUNTER_1_MATCH_2 to trigger FSM_2 correctly and mark the CRC in the encoder response. Refer to Figure 2-5 for the effect of the configuration.

```
static inline void
tformat_configureCLBLen(uint16_t transmitClocks, uint16_t receiveClocks)
{
    CLB_writeInterface(PM_TFORMAT_CLB_BASE,
                       CLB_ADDR_COUNTER_1_MATCH1, transmitClocks);
    CLB_writeInterface(PM_TFORMAT_CLB_BASE,
                       CLB_ADDR_COUNTER_1_MATCH2, transmitClocks);
    CLB_writeInterface(PM_TFORMAT_CLB_BASE,
                       CLB_ADDR_HLC_R0, receiveClocks);


    // Below are the changes
    CLB_writeInterface(PM_TFORMAT_RX_CRC_BASE,
                      CLB_ADDR_COUNTER_1_MATCH1, receiveClocks - 9);
    CLB_writeInterface(PM_TFORMAT_RX_CRC_BASE,
                      CLB_ADDR_COUNTER_1_MATCH2, receiveClocks - 1);

    return;
}
```

The second change in the parameters that passed into the PM_tformat_setupCommand() function.

```
PM_tformat_setupCommandReadoutOrReset(uint16_t commandID0_1_2_3_7_8_C,
                                      uint16_t tformatRXClocks,
                                      uint16_t tformatRXFields,
                                      uint16_t tformatTXClocks,
                                      uint16_t tformatFIFOLevel)
```

The parameters include the frame data, the number of fields/clocks of transfer and receive data, and the FIFO LEVEL. These parameters are related to the interface protocol itself and affected by the FIFO splitting mentioned in the SPI implementation section. The below shows an example of how to calculate these parameters correctly. The example is based on the A-Format CDF0.

```
    #define AFORMAT_FRAME_LEN 9
    #define AFORMAT_TX_FRAMES 1
    #define AFORMAT_CFID0RES_RX_FIELDS 4

    uint16_t aformatTXClocks = (AFORMAT_TX_FRAMES*AFORMAT_FRAME_LEN);
    uint16_t aformatRXClocks = (2u*AFORMAT_CFID0RES_FIELDS*AFORMAT_FRAME_LEN);
    uint16_t aformatRXFields = (2u*AFORMAT_CFID0RES_FIELDS);
    uint16_t aformatFIFOLevel = (2u+2u*AFORMAT_CFID0RES_FIELDS);
```

Since the 18-bit field in A-Format is split into two 9-bit half-fields, when filling the FIFO, the most significant half is filled first, followed by the least significant half. Same to the T-Format solution, the rest of the TX FIFO are filled with 0xFFFF to keep the TX line high during receive. Below is an example of FIFO filling.

```
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = cdf_msh;  // Change
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = cdf_lsh;  // Change

    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;

    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
    HWREGH(PM_TFORMAT_SPI + SPI_O_TXBUF) = 0xFFFF;
```

Finally, the decoding function PM_tformat_receiveDataID0_1_7_8_C() must be changed. This function reads the SPI buffer and decodes the raw data into a comprehensive data structure according to the interface protocol. An example of decoding response of CDF0 in the A-Format is given below. Note that the RX FIFO is also split.

```
void aformat_decode(){
    // for now, limited to the response of CF0
    uint16_t fullData[4] = {0};

    // full data: get rid of start bit 0 and end bit 1
    fullData[0] = (tformatRxData[3]<<7)|(tformatRxData[4]>>2);
    fullData[1] = (tformatRxData[5]<<7)|(tformatRxData[6]>>2);
    fullData[2] = (tformatRxData[7]<<7)|(tformatRxData[8]>>2);
    fullData[3] = (tformatRxData[9]<<7)|(tformatRxData[10]>>2);
    aformatData.fullData = ((uint64_t)fullData[0]<<48)|
                           ((uint64_t)fullData[1]<<32)|
                           ((uint64_t)fullData[2]<<16)|
                           ((uint64_t)fullData[3]);

    // crc data
    aformatData.crcIncludedData = aformatData.fullData>>8; //for cmd ID 0 only!!

    // crc recived from encoder
    aformatData.crcRecv = (aformatData.fullData & 0x00000000000000ff);

    // crc by CLB
    aformatData.crcByCLB = aformat_getRxCRCbyCLB();
    aformatData.crcBothRecvAndCLB = ((aformatData.crcRecv&0xff)<<8) | (aformatData.crcRecv&0xff);

    // single turn data fullData[16:34] = ST[0:18] (19bit)
    aformatData.stData = (aformatData.fullData & 0x0000ffffd0000000)>>29;
    aformatData.stData = (__flip32(aformatData.stData)>>13)&0x7FFFF;

    // multi turn data fullData[35:51] (17bit)
    aformatData.mtData = (aformatData.fullData & 0x000000001ffff000)>>12;
    aformatData.mtData = (__flip32(aformatData.mtData)>>15)&0x1FFFF;

    // err code
    aformatData.errCode = (aformatData.fullData & 0x000f000000000000)>>48;

    // command ID
    aformatData.cmdID =  (aformatData.fullData & 0x03e0000000000000)>>53;

    // encoder address
    aformatData.encoderAddress = (aformatData.fullData & 0x1c00000000000000)>>58;

    encoderStatus.position = (float)aformatData.stData/524288.0*360;
    encoderStatus.turns = aformatData.mtData;
}
```

3 Validation

This interface solution is tested on a Nikon A-Format encoder. The test setup includes the hardware below:

- LAUNCHXL-F280039C
- BOOSTXL-POSMGR
- Format Encoder Nikon-MX50AHN00

The physical photo of the test setup are shown in Figure 3-1.

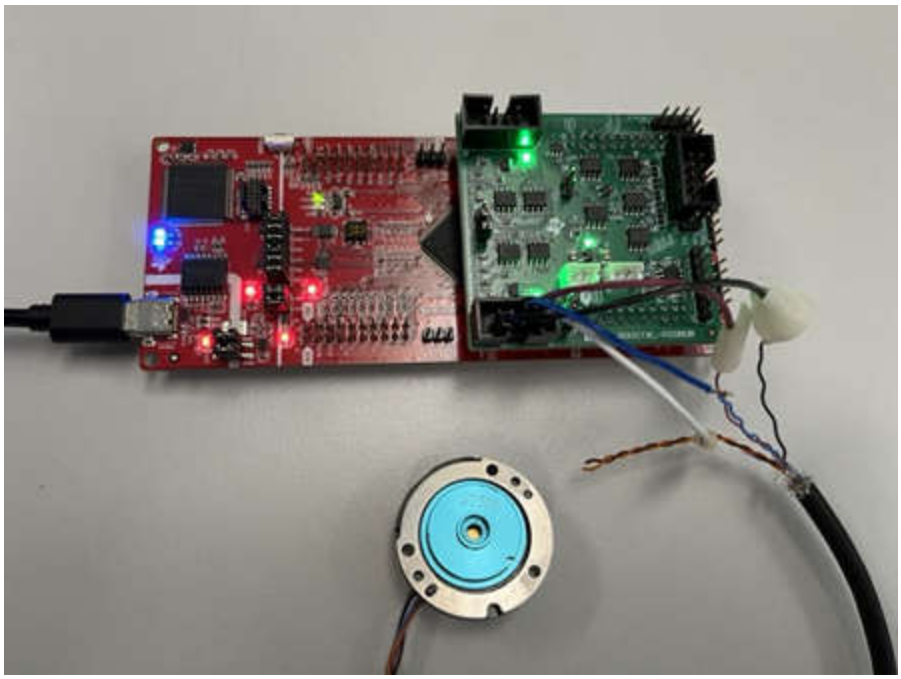

Figure 3-1. Validation Setup

The objective of this test setup is to verify that the A-Format frames can be correctly sent and received by the C2000 controllers. The goal of the test is to acquire an encoder response that can pass the CRC check. Note that the encoder's angular precision (mechanical performance) is beyond the scope of this verification, and therefore the encoder is not installed on an actual motor.

To achieve the desired test goal, the command frame CDF0 is chosen for the test. This command frames requests the encoder to return full 40-bit of the position data. The test program would first encode the CDF0 command and fill the SPI buffer, issue a CLB start signal to send the command, and verify the CRC of the response. The result is displayed as variables using the CCS expression window.

First, the SPI receive buffer data (raw 9-bit entries) is compared with the oscilloscope waveform (captured on the POCI pin), shown in Figure 3-2. The SPI buffer starts with three words of dummy data, each word is 9-bit, as described above. The dummy data is followed by the encoder response, which includes 8 words, each word is 9-bit, with valid data stored in the least significant bits (LSB) of each 9-bit entry. The buffer is aligned with the waveform on the right. This comparison shows that the SPI and CLB had been setup correctly to transfer and receive A-Format frames.

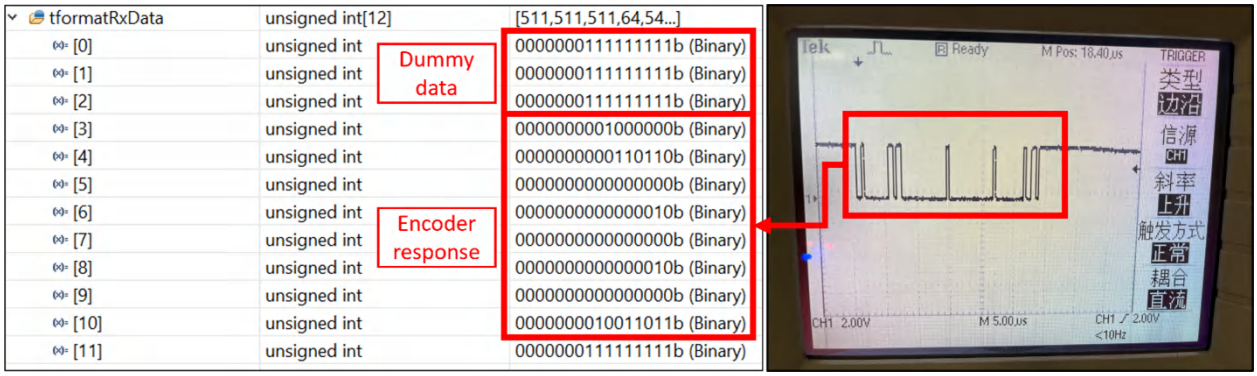

Figure 3-2. RX FIFO Data and Waveform

Next, the decoded data is further checked in Figure 3-2.

For demonstration purposes, the CRC is verified by combining the received CRC and calculated CRC into one 32-bit variable and displaying the variable in the CCS's expression window. This will ensure the CCS sample the CRC variable in one communication cycle. As the result, Figure 3-3 shows a screen shot of CSS expression window, in which one can see the raw data of the received frame, the decoded frame, and that the received CRC is aligned with the calculated one

In the meantime, the behavior of the angle and multi-turn data decoded from the response is verified by spinning the encoder manually and observing the changes in the decoded angular position and multi-turn data (consistent with manual encoder rotation).

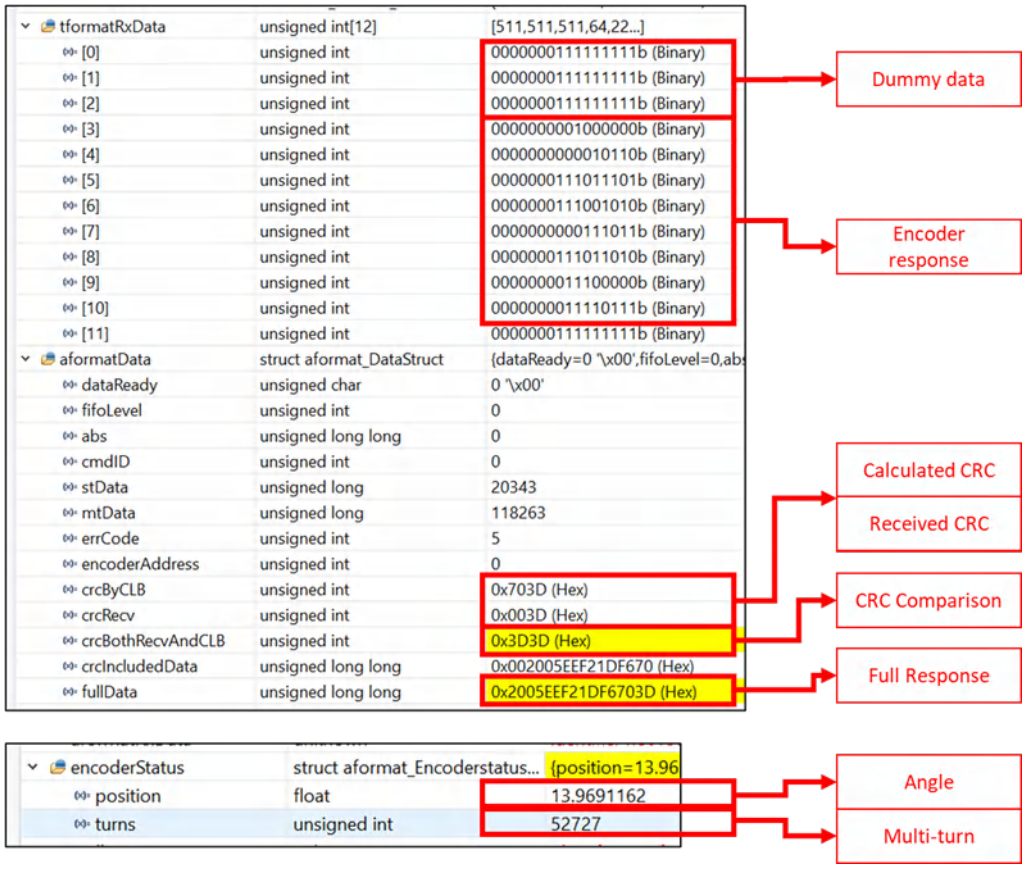

Figure 3-3. Full Decode Encoder Response Data

4 Summary

This application note provides guidance on leveraging the configurable logic block (CLB) of the C2000 F280039C MCU to implement the Nikon A-Format absolute encoder interface. Key implementations include: 1) SPI peripheral configuration for 18-bit field splitting (9-bit FIFO width); 2) CLB logic customization for SPI timing control and A-Format-specific CRC calculation 3) software frame parsing for command/response handling. Validation results confirm that the interface correctly transmits/receives A-Format frames, passes CRC checks, and decodes position data accurately. This solution requires only software modifications based on the TIDM-1011 T-Format reference design, reducing development effort for developers.

5 References

1. TIDM-1011 Tamagawa T-Format Encoder Interface Reference Design. Texas Instruments, 2022.
2. CLB Tool User's Guide (Texas Instruments)
3. TMS320F280039C Microcontroller Data Sheet (Texas Instruments)

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, regulatory 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025