Using quad and octal ADCs in SPI mode

By Tom Hendrick
Data Acquisition Applications—Dallas

Introduction

This article describes the steps required to interface a microprocessor-based system using a serial peripheral interface (SPI) port of the quad and octal family of serial analog-to-digital converters (ADCs) listed in Table 1.

Table 1. TI ADC features

DEVICE	CHANNELS	SUPPLY VOLTAGE (V)	RESOLUTION (bits)	INTERNAL CONVERSION CLOCK
TLV1504	4	2.7 to 5.5	10	Yes
TLV1508	4	2.7 to 5.5	10	Yes
TLV1544	4	2.7 to 5.5	10	Yes
TLV1548	8	2.7 to 5.5	10	Yes
TLV2544	4	2.7 to 5.5	12	Yes
TLV2548	8	2.7 to 5.5	12	Yes
TLC1514	4	5.0	10	No
TLC1518	8	5.0	10	No
TLC2554	4	5.0	12	No
TLC2558	8	5.0	12	No

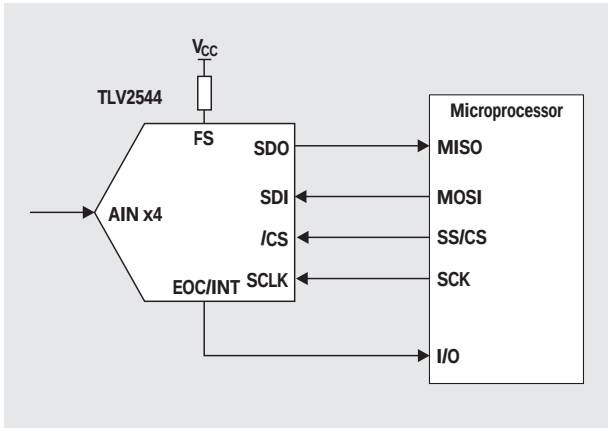
Serial ADC interface

Serial communication with this family of devices is accomplished through three serial inputs and a tri-state serial output: chip select (/CS), serial input clock (SCLK), serial data input (SDI), and serial data output (SDO). These 4 pins provide a direct 4-wire interface to most microprocessors. Figure 1 shows a typical ADC-to-microprocessor interface.

Frame sync and EOC/INT

A frame sync (FS) pin is provided for a typical DSP interface and is normally tied high when used in microprocessor applications. This family of devices also features a programmable end-of-conversion/interrupt (EOC/INT) pin. When programmed as EOC, the output goes from a high to low state at the falling edge of the 16th clock, indicating that the sampling process has been completed. EOC returns to a high state upon the completion of the conversion process. When programmed as /INT, it can act as an interrupt to the host processor. /INT goes from a high to low state at the end of the conversion process and is cleared automatically on the following /CS falling edge.

Figure 1. Four-wire microprocessor interface



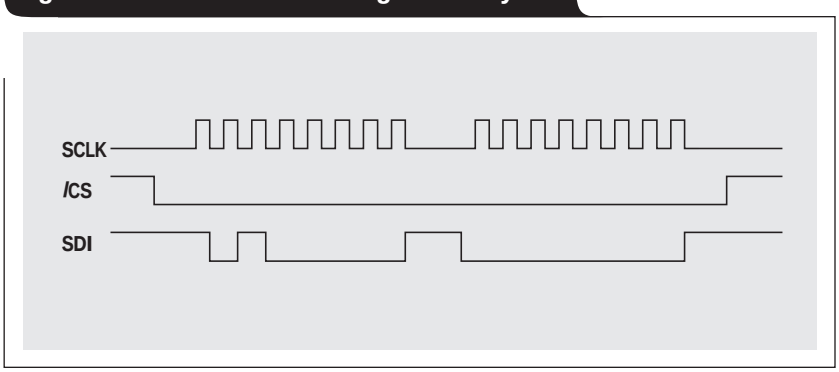
ADC read and write cycles

Many microprocessors offer an SPI port that transmits and receives data in 8-bit packets. The ADCs, however, use a 16-bit data string for configuration and provide serial data output in a 16-bit format.

Configuring the ADC requires the microprocessor to issue back-to-back data transfers while holding the SS/CS line low. The processor's transmit buffer is loaded first with the upper byte of configuration data, sending the SS/CS line low and starting the SCLK, which transmits the serial data to the ADC. The processor's SPI interrupt flag is then cleared, the transmit buffer is loaded with the lower byte of configuration data, and the transfer process begins again. Figure 2 shows the host mode configuration cycle (write 0xA000 to ADC).

The same sequence could be used to read the contents of the configuration register or the contents of the FIFO. The serial data output would simply be configured to perform the requested read operation (write 0x9000 or 0xE000).

Figure 2. The host mode configuration cycle



Continued on next page

Continued from previous page

ADC sample and convert cycles

Each sample/convert cycle requires the user to issue a valid channel select command in the upper nibble of the SDI string. The valid channel selection codes consist of the hex values 0x0000 through 0x9000. FIFO Read (0xE000) will also allow a conversion cycle to take place.

Hex commands 0xA000 through 0xD000 and 0xF000 are reserved for access to the configuration register and internal test modes. More information about these commands can be found in the respective device data sheets listed under References at the end of this article.

Another factor to consider when using these devices in the microprocessor mode is the sample and conversion time. When the TLV2544 is operated in short sampling with the internal conversion clock, for instance, sampling is not complete until the 16th falling edge of SCLK (see Figure 3). Another 3.5 µs are required for the conversion to take place. If chip select is asserted low before the conversion cycle is complete (EOC going Hi), the current conversion data will be lost.

If the microprocessor is set to clock polarity = 1 and clock phase = 1 as shown in Figure 4, the ADC requires the user to issue a third, or "null data," transfer to the SPI port. Since the ADC samples, converts, and shifts data out on the rising edge of the clock, the first falling edge is ignored. As shown in Figure 4, the EOC signal appears on the 16th falling clock edge after the *first* rising. The length of conversion time and number of dummy write transfers the processor has to make is dependent upon the mode in which the ADC is being used, and the clock phase and polarity settings of the SPI port.

The TLC series of devices does not contain an internal conversion clock and depends on SCLK throughout the entire sample/conversion process. This requires that additional "null data" SPI transfers be conducted. The total number of clock cycles is dependent upon the operating mode of the ADC being used. The TLC2544 (in single-shot mode), for example, would require 16 clocks for channel selection and sampling, plus an additional 16 clocks for conversion. A minimum of four 8-bit SPI blocks would be needed.

Figure 3. Sample/convert cycle

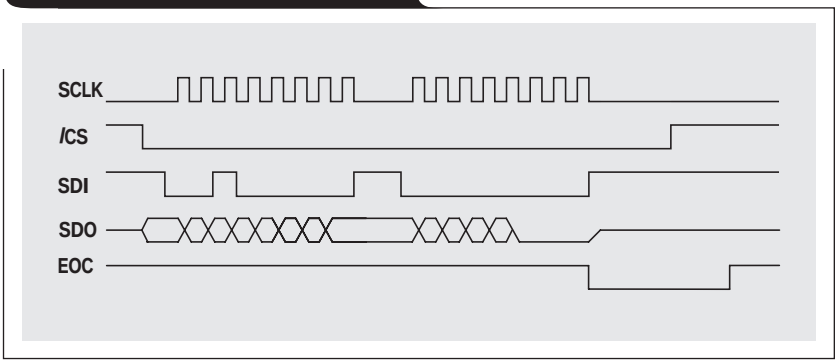
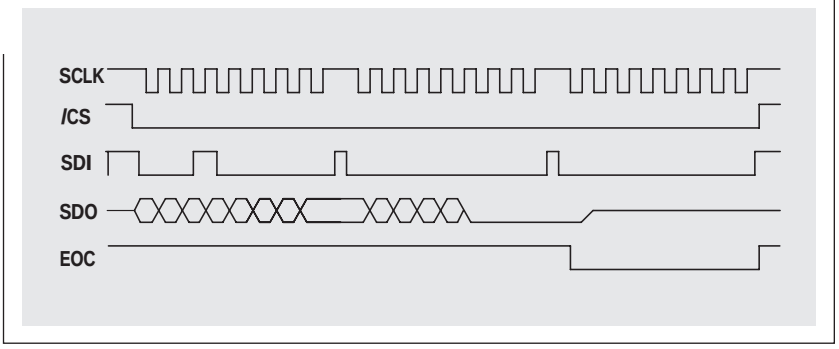


Figure 4. Alternate sample/convert cycle



Reading valid data

The serial data out from these devices is pre-released by 1/2 clock cycle, plus a delay. What this means is that while valid output data is available on the rising edge of SCLK, the rising edge also triggers the shifting out of the next data bit. With low clock speeds, it may appear as if data is changing when it should be valid. A processor's setup and hold time requirements may affect its ability to read correctly the data presented by the converter. Figures 5 and 6 show the relationship between SCLK, SDO, and SDI with a 4-MHz and 20-MHz clock, respectively.

The code example at the bottom of the next page was written for the Motorola 68HC912 and was assembled using ASM12.EXE. The code samples and stores data from channel 0 of a TLV2544.

References

For more information related to this article, you can download an Acrobat Reader file at www-s.ti.com/sc/techlit/*litnumber* and replace "*litnumber*" with the **TI Lit. #** for the materials listed below.

Document Title	TI Lit. #
1. TLV1504/1508/1544/2544/2548, TLC1514/1518/2554/2558 10-Bit and 12-Bit ADC EVM User's Guide	slau029
2. TLV1504,TLV1508 Data Sheet	slas251
3. TLV1544,TLV1548 Data Sheet	slas139
4. TLV2544, TLV2548 Data Sheet	slas198
5. TLC1514,TLC1518 Data Sheet	slas252
6. TLC2554,TLC2558 Data Sheet	slas220

Related Web sites

www.dataconverter.com

www.ti.com/sc/docs/products/analog/*device***.html**
Replace *device* with tlc1514, tlc1518, tlc2554, tlc2558, tlv1504, tlv1508, tlv1544, tlv2544, or tlv2548

Figure 5. SCLK = 4 MHz

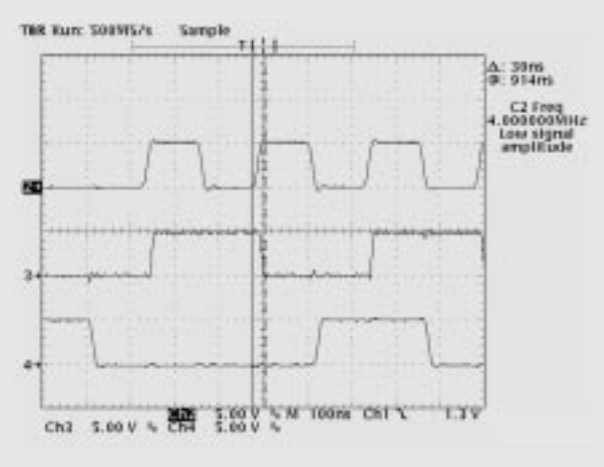
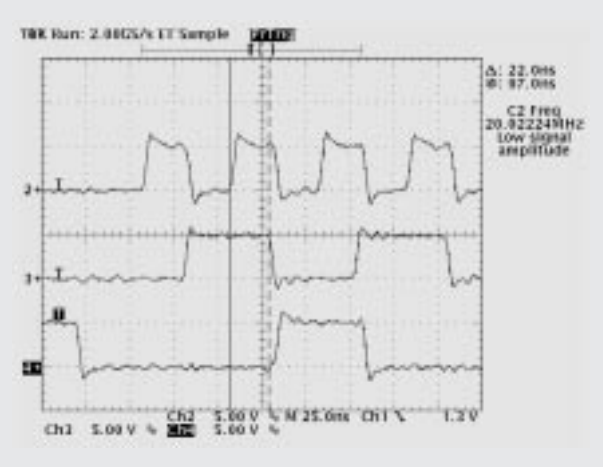


Figure 6. SCLK = 20 MHz



Code example

```
; SPI program using the 68HC912 uP and a TLV2544 ADC with Frame Sync hi via SP3
;* ------------------------------------------------;
;*             Equates and Variables
;* ------------------------------------------------

SP0CR1:     equ $D0        ;SPI 0 Control Register 1
SP0CR2:     equ $D1        ;SPI 0 Control Register 2
SP0BR:      equ $D2        ;SPI 0 Baud Rate Register
SP0SR:      equ $D3        ;SPI 0 Status Register
SP0DR:      equ $D5        ;SPI 0 Data Register
PORTS:      equ $D6        ;Port S Data Register
DDRS:       equ $D7      ;Port S Data Direction Register

; User Variables

Upper_Byte:  EQU $0B00
Lower_Byte:  EQU $0B01
Upper_CFIG:  EQU $0B02
Lower_CFIG:  EQU $0B03
;* ------------------------------------
;*             MAIN PROGRAM
;* ------------------------------------
        ORG      $0800     ; User code data area,
                           ; start main program at $0800
DATA    FCB      00,01     ; Set up 16 bit Output DATA
MAIN:
        BSR      INIT      ; Subroutine to initialize SPI registers
        JSR      SAMPLE    ; Subroutine to start transmission
```

Continued on next page

Continued from previous page

Code example (Continued)

```
;* ____________________________________
;*             Initialization Subroutine
;* ____________________________________

INIT:
       BSET DDRS, #%11101100      ; Configure PORT S input/ouput:
                                  ; SS/CS, SCK, MOSI, MISO, PS3, PS2, TXD, RXD
       BSET SP0BR, #$00           ; Set Baud Rate
       MOVB #$50, SP0CR1          ; Configure SPI(SP0CR1)
                                  ; SPIE, SPE, SWOM, MSTR, CPOL, CPHA, SSOE, LSBF
       MOVB #$00, SP0CR2          ; Configure SPI(SP0CR2):
                                  ; -,-,-,-,-,-,SSWAI, SPCO
       MOVB #$88, PORTS           ; Sets ADC CS Hi, FS Hi
       BCLR PORTS, #%10000000     ; Select ADC
                                  ; Configure ADC
       MOVB #$A0, Upper_CFIG      ; Write 0xA000 to set up Host Communication.
       MOVB Upper_CFIG, SP0DR     ; Put data in XMIT Buffer
       JSR FLAG                   ; Clear SPI Flag
       MOVB #$04, Lower_CFIG      ; Select EOC Mode
       MOVB Lower_CFIG, SP0DR     ;
       JSR FLAG                   ; Clear SPI Flag
       BSET PORTS, #$80           ; Sets ADC CS Hi
       MOVB #$00, UPPER_BYTE      ; Set initial values
       MOVB #$00, LOWER_BYTE      ; Set initial values
       RTS
;* ____________________-
;*      Sample / Convert
;* ____________________-
SAMPLE:
       MOVB #$08, PORTS           ; Sets ADC CS Low
       MOVB #$20, SP0DR           ; Tell ADC what channel to read and generate SCLK for ADC
       JSR FLAG                   ; Clear SPI Flag
                                  ; Store received data
       LDAA SP0DR                 ; Load first ADC Sample
       MOVB #$00, SP0DR           ; Write zero value to data register to generate SCLK for ADC
       JSR FLAG                   ; Clear SPI Flag
       LDAB SP0DR                 ; Load second ADC Sample
       STD DATA                   ; Store ACCA and ACCB in Data

       nop                        ; Extra time for conversion nop
       MOVB #$88, PORTS           ; Sets ADC CS Lo
       JMP SAMPLE                 ; Go back and do it again

;* ____________________-
;*         Clear SPI Flag Subroutine
;* ____________________-

FLAG:  BRCLR   SP0SR,#$80,FLAG    ; Wait for flag.
       RTS
.end
```