Application Note

Interfacing the CS5525/6/9 to the 80C51

By Keith Coffey

INTRODUCTION

This application note details the interface of Crystal Semiconductor's CS5525/6/9 Analog-to-Digital Converter (ADC) to an 80C51 microcontroller. This note takes the reader through a simple example describing how to communicate with the ADC. All algorithms discussed are included in the **Appendix** at the end of this note.

ADC DIGITAL INTERFACE

The CS5525/6/9 interfaces to the 80C51 through either a three-wire or a four-wire interface. Figure 1 depicts the interface between the two devices. Though this software was written to interface to Port 1 (P1) on the 80C51 with a four-wire interface, the algorithms can be easily modified to work with the three-wire format.

The ADC's serial port consists of four control lines: $\overline{CS}$, SCLK, SDI, and SDO.

$\overline{CS}$, Chip Select, is the control line which enables access to the serial port.

SCLK, Serial Clock, is the bit-clock which controls the shifting of data to or from the ADC's serial port.

SDI, Serial Data In, is the data signal used to transfer data from the 80C51 to the ADC.

SDO, Serial Data Out, is the data signal used to transfer output data from the ADC to the 80C51.

SOFTWARE DESCRIPTION

This note presents algorithms to initialize the 80C51 and the CS5525/6/9, perform self-offset calibration, modify the CS5525/6/9's gain register, and acquire a conversion. Figure 2 depicts a block

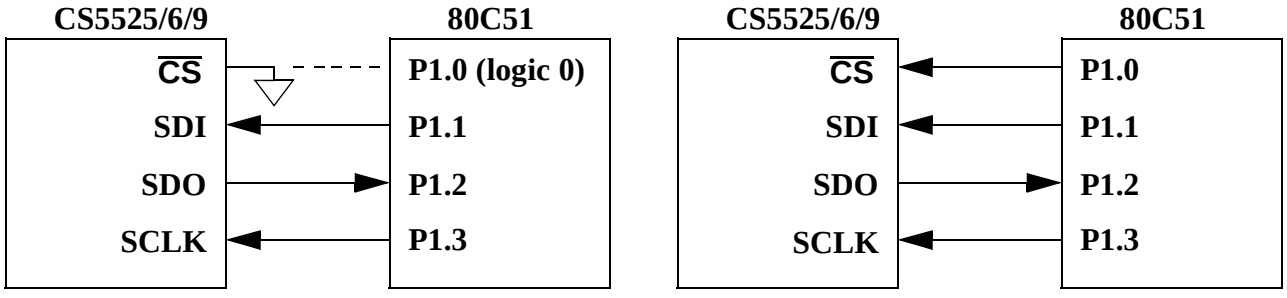


Figure 1. 3-Wire and 4-Wire Interfaces

Cirrus Logic, Inc.
Crystal Semiconductor Products Division
P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.crystal.com

diagram. While reading this application note, please refer to the **Appendix** for the code listing.

Initialize

Initialize is a subroutine that configures P1 (Port 1) on the 80C51 and places the CS5525/6/9 into the command-state. First, P1's data direction is configured as depicted in Figure 1 (for more information on configuring ports refer to 80C51 Data Sheet). After configuring the port, the controller enters a delay state to allow time for the CS5525/6/9's power-on-reset and oscillator to start-up (oscillator start-up time is typically 500ms). The last step is to reinitialize the serial port on the ADC (reinitializing the serial port is unnecessary here, the code was added for demonstration purposes only). This is implemented by sending the converter sixteen bytes of logic 1's followed by one final byte, with its LSB at logic 0. Once sent, the sequence places the serial port of the ADC into the command-state, where it awaits a valid command.

After retuning to *main,* the software demonstrates how to calibrate the converter's offset.

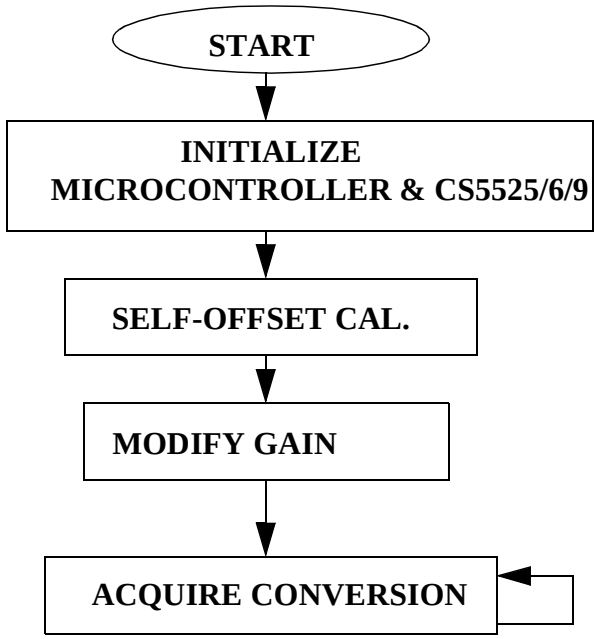


Figure 2. CS5525/6/9 Software Flowchart

Self-Offset Calibration

Calibrate is a subroutine that calibrates the converter's offset. *Calibrate* first sends 0x000001 (Hex) to the configuration register. This instructs the converter to perform a self-offset calibration. Then the Done Flag (DF) bit in the configuration register is polled until set. Once DF is set, it indicates that a valid calibration was performed. To minimize digital noise (while performing a calibration or a conversion), many system designers may find it advantageous to add a software delay equivalent to a conversion or calibration cycle before polling the DF bit.

Read/Write Gain Register

To modify the gain register the command-byte and data-byte variables are first initialized. Then the subroutine *write_to_register* uses these variables to set the contents of the gain register in the CS5525/6/9 to 0x800000 (HEX). To do this, *write_to_register* calls *transfer_byte* four times (once for the command byte and three additional times for the 24 bits of data). *Transfer_byte* is a subroutine used to 'bit-bang' a byte of information from the 80C51 to the CS5525/6/9. A byte is transferred one bit at a time, MSB (most significant bit) first, by placing a bit of information on P1.1 (SDI) and then pulsing P1.3 (SCLK). The byte is transferred by repeating this process eight times. Figure 3 depicts the timing diagram for the write-cycle in the CS5525/6/9's serial port. This algorithm demonstrates how to write to the gain register. It does not perform a gain calibration. To perform a gain calibration, follow the procedures outlined in the data sheet.

To verify that 0x800000(HEX) was written to the gain register, *read_register* is called. It duplicates the read-cycle timing diagram depicted in Figure 4. *Read_register* first asserts $\overline{CS}$. Then it calls *transfer_byte* once to transfer the command-byte to the CS5525/6/9. This places the converter into the

data-state where it waits until data is read from its serial port. *Read_register* then calls *receive_byte* three times and transfers three bytes of information from the CS5525/6/9 to the 80C51. Similar to *transfer_byte*, *receive_byte* acquires a byte one bit at a time MSB first. When the transfer is complete, the variables high_byte, mid_byte, and low_byte contain the CS5525/6/9's 24-bit gain register.

Acquire Conversion

To acquire a conversion the subroutine *acquire_conversion* is called. To prevent from corrupting the configuration register *acquire_conversion* first instructs the 80C51 to save the contents of configuration register. This information is stored in the variable high_byte, mid_byte and low_byte. Then, PF (Port Flag, the fifth bit in the configuration register which is now represented as bit five in the variable low-byte) is masked to logic 1. When PF is set to logic 1, SDO's function is modified to fall to logic 0 signaling when a conversion is complete and ready to acquire (refer to Figure 5). After the PF is set, *acquire_conversion* sends the command-byte 0xC0 to the converter instructing it to perform a single conversion. From there, *acquire_conversion* calls the subroutine *toggle_sdo*. *Toggle_sdo* is routine that polls P1.2 (SDO) until its logic level drops to logic 0. After SDO falls, *toggle_sdo* pulses P1.3 (SCLK) eight times to clear the SDO signal flag. After the SDO flag is cleared, the 80C51 reads the conversion data word. Figure 6 depicts how 16-bit and 20-bit conversion words are stored.

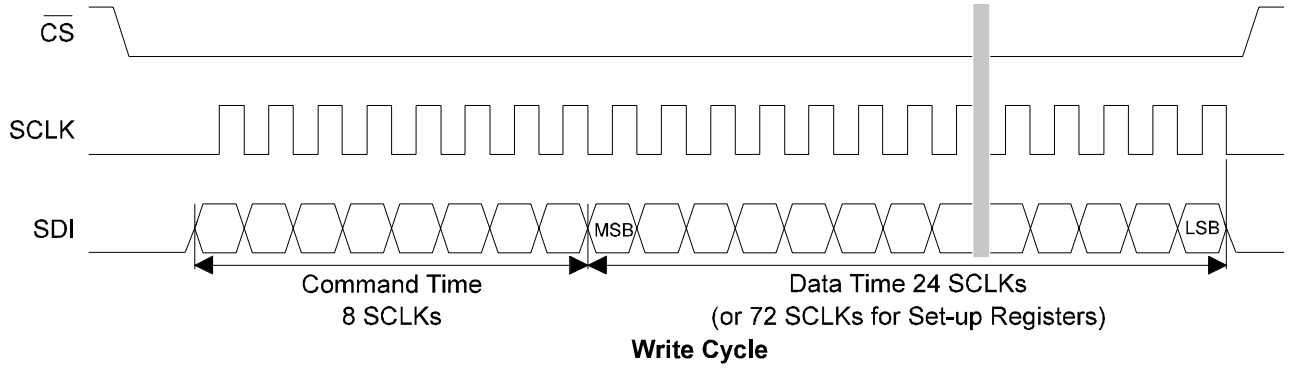


Figure 3. Write-Cycle Timing

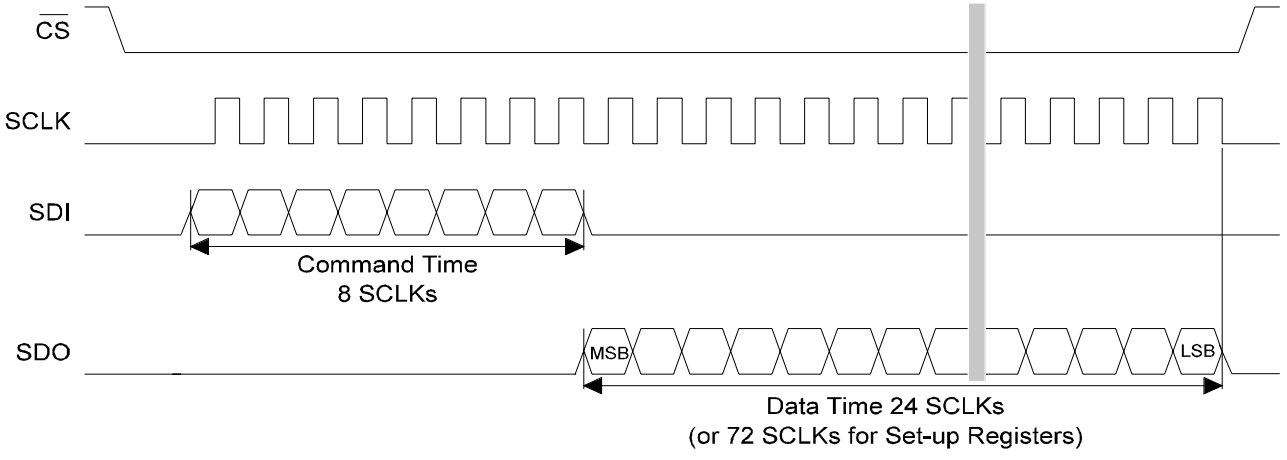
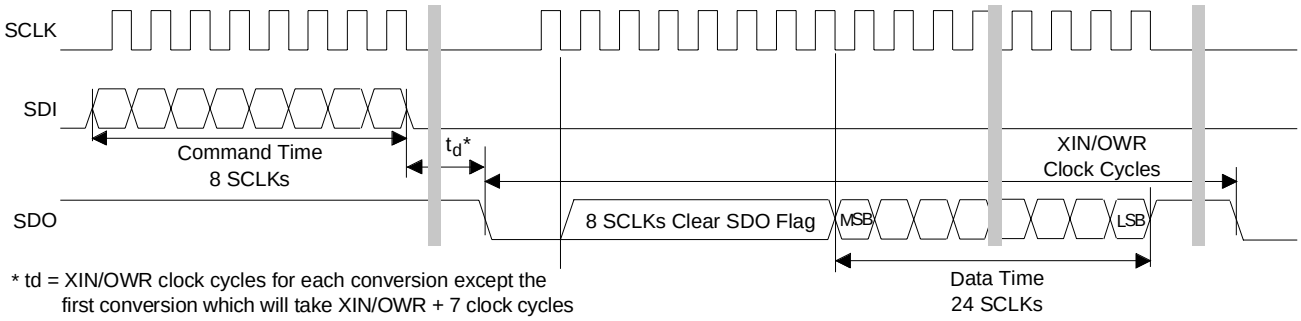


Figure 4. Read-Cycle Timing



Data SDO Continuous Conversion Read (PF bit = 1)

Figure 5. Conversion/Acquisition Cycle with PF Bit Asserted

An alternative method can be used to acquire a conversion. By clearing the Port Flag bit, the serial port's function isn't modified. The Done Flag bit (bit three in the configuration register) can be polled as it indicates when a conversion is compete and ready to acquire. The conversion is acquired by reading the conversion data register.

MAXIMUM SCLK RATE

A machine cycle in the 80C51 consists 12 oscillator periods or 1µs if the microcontroller's oscillator frequency is 12 MHz. Since the CS5525/6/9's maximum SCLK rate is 2MHz, additional no operation (NOP) delays may be necessary to reduce the transfer rate if the microcontroller system requires higher rate oscillators.

MSB **High-Byte**

D19	D18	D17	D16	D15	D14	D13	D12

Mid-Byte

D11	D10	D9	D8	D7	D6	D5	D4

Low-Byte

D3	D2	D1	D0	0	0	OD	OF

A) 20-Bit Conversion Data Word

MSB **High-Byte**

D15	D14	D13	D12	D11	D10	D9	D8

Mid-Byte

D7	D6	D5	D4	D3	D2	D1	D0

Low-Byte

1	1	1	1	0	0	OD	OF

B) 16-Bit Conversion Data Word

0- always zero, 1- always one,

OD - Oscillation Detect, OF - Overflow

Figure 6. Bit Representation/Storage in PIC16F84

DEVELOPMENT TOOL DESCRIPTION

The code in the application note was developed using a software development package from Franklin Software, Inc. The code consists of intermixed C and assembler algorithms which are subsets of the algorithms used by the CDB5525/6/9, a customer evaluation board from Crystal Semiconductor. Moreover, Franklin's A51 Assembler, C51 Compiler, and L51 Linker development software were used to generate the run-time software for the microcontroller on the CDB5526.

CONCLUSION

This application note presents an example of how to interface the CS5525/6/9 to the 80C51. It is divided into two main sections: hardware and software. The hardware interface illustrates both a three-wire and a four-wire interface. The three-wire is *SPI™* and *MICROWIRE™* compatible. The software, developed with development tools from Franklin Software, Inc., illustrates how to write to the ADC's internal register, read from the ADC's internal registers, and acquire a conversion. The software is modularized and illustrates the important subroutines, e.g. *write_byte*, *read_byte*, and *toggle_sdo*, each of which were written in assembly language. This allows both assembly and C programmers access to these modules.

The software described in the note is included in the **Appendix** at the end of this document.

SPI™ is a trademark of Motorola.

MICROWIRE™ is a trademark of National Semiconductor.

APPENDIX

80C51 Microcode to Interface to the CS5525/6/9

```
/*****************************************************************************
* File:           55268051.asm
* Date:           November 1, 1996
* Programmer: Keith Coffey
* Revision:       0
* Processor:      80C51
* Program entry point at routine "main".
*****************************************************************************
* This program is designed as an example of interfacing a 80C51 to a CS5525/6/9
* Analog-to-Digital Converter. The program interfaces via Port 1 which controls the
* serial communications, calibration, and conversion signals.
*****************************************************************************/
/***  Function Prototypes  ***/
void        initialize(void);
void        reset_converter(void);
void        toggle_sdo(void);
char        receive_byte(void);
void        transfer_byte(char);
void        write_to_register(char command,char low,char mid, char high);
void        read_register(char command);
void        acquire_conversion(char command);

/***  Byte Memory Map Equates  ***/
sfr         P1          =           0x90;       /*Port One*/
sfr         ACC         =           0xE0;       /*Accumulator Register Equate*/

/***  Bit Memory Map Equates  ***/
sbit        CS          =           0x90;       /* Chip Select, only used in four-wire mode*/
sbit        SDI         =           0x91;       /* Serial Data In*/
sbit        SDO         =           0x92;       /*Serial Data Out*/
sbit        SCLK        =           0x93;       /*Serial Clock*/

/***  Global Variable  ***/
char            command,                        /*Memory Storage Variable for Command Byte */
                high_byte,                      /*Memory Storage Variable for Most Significant Byte*/
                mid_byte,                       /* Memory Storage Variable for Most Significant Byte*/
                low_byte,                       /* Memory Storage Variable for Most Significant Byte*/
                temp,                           /*General Purpose Temporary Variable*/
                mode;                           /*Variable Stores Mode of Operation 0 = three wire, 1 = 4 wire*/
```

```
/*****************************************************************************************
*      Program Code
*****************************************************************************************
* Routine  - Main
* Input    - none
* Output   - none
* This is the entry point to the program
*****************************************************************************************/
main() {
        mode        = 1;                        /*Make Communication be Four-Wire Mode*/
        initialize();                           /*Call Routine to Initialize 80C51 and CS5525/6/9*/
        while(1){
          command           = 0x82;             /*Prepare to Write to Gain Register*/
          high_byte         = 0x80;             /*Make High_byte 80 (HEX)*/
          mid_byte          = 0x00;             /*Make Mid_byte all Zero's*/
          low_byte          = 0x00;             /*Make low_byte all Zero's*/
          write_to_register(command,low_byte,mid_byte,high_byte);/*Write to gain Register*/
          read_register(0x92);                  /*Read Contents of Gain Register*/
          while(1){
                acquire_conversion(0xC0);       /*Acquire a Single Conversion*/
          }/*End inner while loop*/
        }/*End While Loop*/
}/*end main*/

/*****************************Subroutines*****************************************************/
/*****************************************************************************************
* Routine  - initialize
* Input    - none
* Output   - none
* This subroutine initializes Port 1 for interfacing to the CS5525/6/9 ADC.
* It provides a time delay for oscillator start-up/wake-up period.
* A typical start-up time for a 32768 Hz crystal, due to high Q, is 500 ms.
* Also 1003 XIN clock cycles are allotted for the ADC's power on reset.
*****************************************************************************************/
void initialize()
/***  Local Variables  ***/
          data int     counter;
/***  Body of Subroutine  ***/
          /***  Initialize 80C51's Port 1 ***/
          P1        =         0xF4;         /*SCLK - Output */
                                            /*SDI    - Output */
                                            /*SDO    - Input */
                                            /*CS     - Output */
          /*Initialize CS5525/6/9*/
          /*Delay 2048 SCLK Cycles, to allow time for Oscillator start-up and power on reset*/
          for(counter=0;counter<2047;counter++){
                    SCLK      =         0x01;       /*Assert SCLK*/
                    SCLK      =         0x00;       /*Deassert*/
          }
```

```
        /*Reset Serial Port on CS5525/6/9*/
        SDI     =       0x01;           /*Assert SDI*/
        for(counter=0;counter<255;counter++) {
                SCLK = 0x01;            /*Assert SCLK*/
                SCLK = 0x00;            /*Deassert SCLK*/
        }
        SDI     =       0x00;           /*Deassert SDI PIN*/
        SCLK    =       0x01;           /*Assert SCLK*/
        SCLK    =       0x00;           /* Deassert SCLK*/
}
/*****************************************************************************************
* Routine  - calibrate
* Input    - none
* Output   - none
* This subroutine instructs the CS5525/6/9 to perform self-offset calibration.
*****************************************************************************************/
void    calibrate()
{       write_to_register(0x84,0x01,0x00,0x00);     /*Assert RS bit*/

        /*Read Configuration Register Until DF Bit is Asserted*/
        do {
                read_register(0x94);            /*Read Configuration Register*/
                temp    = low_byte&0x08;        /*Mask DF bit to 1*/
        } while (temp != 0x08);
        read_register(0x92);   /*Deasserts DF Bit*/
}/*End calibrate */

/*****************************************************************************************
* Routine  - write_to_register
* Input    - command, lowbyte, midbyte, highbyte
* Output   - none
* This subroutine instructs the CS5525/6/9 to write to an internal register.
*****************************************************************************************/
void    write_to_register(char command,char low,char mid,char high){
        if(mode == 1)   P1   = 0xF4;            /*Assert CS if necessary*/
        transfer_byte(command);                 /*Transfer Command Byte to CS5525/6/9*/
        transfer_byte(high);                    /*Transfer High Byte to CS5525/6/9*/
        transfer_byte(mid);                     /*Transfer Middle Byte to CS5525/6/9*/
        transfer_byte(low);                     /*Transfer Low Byte to CS5525/6/9*/
        if(mode == 1)   P1   = 0xF5;            /*Deassert CS if necessary*/
}
```

```
/*****************************************************************************************
* Routine  - read_register
* Input    - command
* Output   - low_byte, mid_byte, high_byte
*
* This subroutine reads an internal register of the ADC
*****************************************************************************************/
void        read_register(char command){
            if(mode == 1) P1 = 0xF4;                    /*Assert CS if necessary */
            transfer_byte(command);                     /*Transfer Command Byte to CS5525/6/9*/
            high_byte  = receive_byte();                /*Receive Command Byte from CS5525/6/9*/
            mid_byte   = receive_byte();                /*Receive Command Byte from CS5525/6/9*/
            low_byte   = receive_byte();                /*Receive Command Byte from CS5525/6/9*/
            if(mode == 1)P1 = 0xF5;                     /*Deassert CS if necessary */
}

/*****************************************************************************************
* Routine  - acquire_conversion
* Input    - command
* Output   - Conversion results in memory locations HIGHBYTE, MIDBYTE and
*            LOWBYTE. This algorithm performs only single conversions. If
*            continuous conversions are needed the routine needs to be
*            modified. Port flag is zero.
*
*                       HIGHBYTE       MIDBYTE        LOWBYTE
*                       7 6 5 4 3 2 1 0  7 6 5 4 3 2 1 0  7 6 5 4 3 2 1 0
* 16-bit results MSB                                    LSB 1 1 1 1 0 0 OD OF
* 20-bit results MSB                                     LSB 0 0 OD OF
* This subroutine initiates a single conversion.
*****************************************************************************************/
void        acquire_conversion(char command){
            /***  Read Configuration Register to Prevent Previously Set Bits from being Altered  ***/
            read_register(0x94);                        /*Read Configuration Register*/
            low_byte = low_byte|0x20;                   /*Assert Port Flag Bit*/
            write_to_register(0x84,low_byte, mid_byte, high_byte);/*Actually Send Commands*/

            /*Acquire a Conversion*/
            if(mode == 1)P1       = 0xF4;               /*Assert CS if necessary*/
            transfer_byte(0xC0);                        /*Transfer Command to CS5525/6/9*/
            toggle_sdo();                               /*Clear SDO*/
            high_byte  = receive_byte();                /*Receive Command Byte from CS5525/6/9*/
            mid_byte   = receive_byte();                /*Receive Command Byte from CS5525/6/9*/
            low_byte   = receive_byte();                /*Receive Command Byte from CS5525/6/9*/
            if(mode == 1) P1      = 0xF5;               /*Deassert CS if necessary*/
}
```

```
;*****************************************************************
;* Routine - RECEIVE_BYTE
;* Input   - none
;* Output  - Byte received is placed in R7
;* Description -  This routine moves 1 byte from the CS5525/6/9 to the 80C51.
;                 It transfers the byte by acquiring the logic level on PORT1 BIT 2
;                 It then pulses SCLK high and then back low again
;                 to advance the A/D's serial output shift register to the next bit.
;                 It does this eight times to acquire one complete byte.
;                 This function's prototype in C is: char receive_byte(void);
;Note:      This routine can be used three time consecutively to transfer all 24 bits
;           from the internal registers of the CS5525/6/9.
;*****************************************************************
$DEBUG
USING   0                           ; Use register bank 0
TCOD    SEGMENT CODE                ; Define ROUT as a segment of code
PUBLIC  RECEIVE_BYTE                ; Make subroutine global
RSEG    TCOD                        ; Make code relocatable
RECEIVE_BYTE:
        MOV     R1,#08              ; Set count to 8 to receive byte

LOOP:                               ; Receive the byte
        MOV      C,P1.2             ; Move bit to carry
        RLC     A                   ; Rotate A in preparation for next bit
        SETB    P1.3                ; Set SCLK
        CLR     P1.3                ; Clear SCLK
        DJNZ    R1,LOOP             ; Decrement byte, repeat loop if not zero
        MOV     R7,A                ; Byte to be return is placed in R7
        RET                         ; Exit subroutine
END
```

```
;*****************************************************************
;* Routine  - TRANSFER_BYTE
;* Input    - byte to be transferred
;* Output   - None
;* Description -   This subroutine transfers 1 byte to the CS5525/6/9
;                  It transfers the byte by first placing a bit in PORT1 BIT 1.
;                  It then pulses the SCLK to advance the A/D's serial
;                  output shift register to the next bit.
;                  It does this eight times to transmit one complete byte.
;                  The function prototype is:   void TRANSFER_BYTE(char);
;Note:      This routine can be used three time consecutively to transfer all 24 bits
;            from the 80C51 to the internal registers of the CS5525/6/9.
;*****************************************************************
$DEBUG
USING   0                               ; Use register bank 0
TCOD    SEGMENT CODE                    ; Make TCOD a segment of code
TDAT    SEGMENT DATA                    ; Make TDAT a segment of data
PUBLIC  TRANSFER_BYTE                   ; Make subroutine global
PUBLIC  ?TRANSFER_BYTE?BYTE             ; Make subroutine global
RSEG    TDAT                            ; Make code relocatable
?TRANSFER_BYTE?BYTE:
VAR:    DS      1                       ; Define a storage location
RSEG    TCOD                            ; Make code relocatable
TRANSFER_BYTE:
            MOV     A,VAR               ; Move byte to be transmitted to ACC
            MOV     R1,#08              ; Set count to 8 to transmit byte
            CLR     P1.3                ; Clear SCLK

loop:   ; Send Byte
            RLC     A                   ; Rotate Accumulator, send MSB 1st
            MOV     P1.1,C              ; Transmit MSB first through C bit
            SETB    P1.3                ; Set SCLK
            CLR     P1.3                ; Clear SCLK
            DJNZ    R1,loop             ; Decrement byte, repeat loop if not zero
            CLR     P1.1                ; Reset SDI to zero when not transmitting
            RET                         ; Exit subroutine
END
```

```
;*****************************************************************
;* Routine - TOGGLE_SDO
;* Input   - none
;* Output  - none
;* Description -  This routine reset the DRDY pin by toggling
;*                SCLK 8 times after SDO falls.
;                 This routine polls SDO, waits for it to be asserted, then clears SDO
;                 for next conversion by pulsing SCLK eight times after SDO falls
;                 This functions prototype in C is: void toggle_sdo(void);
;*****************************************************************
$DEBUG
USING   0                               ; Use register bank 0
TCOD    SEGMENT CODE                    ; Define Rout as a segment of code
PUBLIC TOGGLE_SDO                       ; Make subroutine public
RSEG    TCOD                            ; Make code relocatable
TOGGLE_SDO:
        MOV     R1,#08                  ; Setup counter
        CLR     P1.1                    ; Clear SDI
        JB      P1.2,$                  ; Poll SDO

loop:
        SETB    P1.3                    ; Set SCLK
        CLR     P1.3                    ; Clear SCLK
        DJNZ    R1,loop                 ; Decrement byte, repeat loop if not zero
        RET                             ; Exit Subroutine
END
```

• Notes •


SMART
Analog
™