

## INTERFACING ICM7363/43/23 To MCS-51 MICROCONTROLLERS

Sairan Sakrani

### ICM7363/43/23 3-Wire Serial Interface

Commands and data are loaded to ICM7363/43/23 via 3-wire serial interface. SDI pin is used to assert the serial data input, while SCK pin is used to clock the data. The  $\overline{CS}$  pin is used to initiate and terminate the serial data transfer.

Figure 1.1 shows the proper signal sequences to correctly communicate and transfer serial data to the ICM7363/43/23 digital-to-analog converter. The timing specifications are as defined in the datasheet.

$\overline{CS}$  pin should initially be HIGH (note: SDI and SCK are ignored while  $\overline{CS}$  is HIGH), while SCK should initially be LOW. To start the serial data transfer,  $\overline{CS}$  is first brought LOW. This activates the ICM7363/43/23 serial interface. While  $\overline{CS}$  is LOW, data is serially transferred to ICM7363/43/23 internal serial buffer at

each positive edge of the SCK. Sixteen (16) SCK pulses are needed to transfer all the required 16-bit command and data bits. The first transferred bit is the MSB. Table 1.1 and Table 1.2 show the command format and description.

After all the 16 bits have been transferred,  $\overline{CS}$  needs to be brought back to HIGH level to end the communication. The  $\overline{CS}$  transition from LOW to HIGH causes ICM7363/43/23 to latch the data from internal serial buffer to another internal parallel buffer, and execute the loaded command.

Note that if more than 16 bits are transferred, only the last 16 bits will be asserted. If less than 16 bits are transferred just before the communication is terminated ( $\overline{CS}$  going from LOW to HIGH), the remaining bits are taken from the previous communication session. The result can be unpredictable.

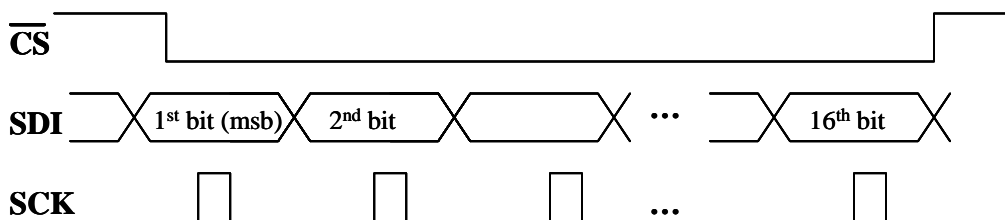


Figure 1.1: Serial Interface Signals

Devices	Command Bits				Data Bits											
	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ICM7363	C3	C2	C1	C0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
ICM7343	C3	C2	C1	C0	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X
ICM7323	C3	C2	C1	C0	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X

**Table 1.1:** 16-Bit Command and Data Format

Command Bits				Data Bits	Target DAC	Functions
C3	C2	C1	C0			
0	0	0	0	Data	A	Load data into input buffer
0	0	0	1	X	A	Update DAC (assert data on input buffer)
0	0	1	0	Data	A	Load data and update DAC simultaneously
0	0	1	1	Data	B	Load data into input buffer
0	1	0	0	X	B	Update DAC (assert data on input buffer)
0	1	0	1	Data	B	Load data and update DAC simultaneously
0	1	1	0	Data	C	Load data into input buffer
0	1	1	1	X	C	Update DAC (assert data on input buffer)
1	0	0	0	Data	C	Load data and update DAC simultaneously
1	0	0	1	Data	D	Load data into input buffer
1	0	1	0	X	D	Update DAC (assert data on input buffer)
1	0	1	1	Data	D	Load data and update DAC simultaneously
1	1	0	0	Data	A-D	Load data into input buffer
1	1	0	1	X	A-D	Update DAC (assert data on input buffer)
1	1	1	0	Data	A-D	Load data and update DAC simultaneously
1	1	1	1	X	X	No Operation

**Table 1.2:** ICM7363/43/23 Control Commands

## Interface Example using MCS-51 Microcontroller

Figure 2.1 shows example of interface block diagram for communicating with ICM7363/43/23 using 8051-based microcontroller.

Example shown in Figure 2.1 uses Port2.3 to control  $\overline{CS}$  pin, uses Port2.2 to connect to SDI pin for the serial data

and uses Port2.0 to connect to SCK pin for the serial data clock.

Figure 2.2 shows the corresponding sample of 8051 assembly codes that send and simultaneously update all four DACs inside ICM7363/43/23 with sequential set of data starting from code 0 to code 4095.

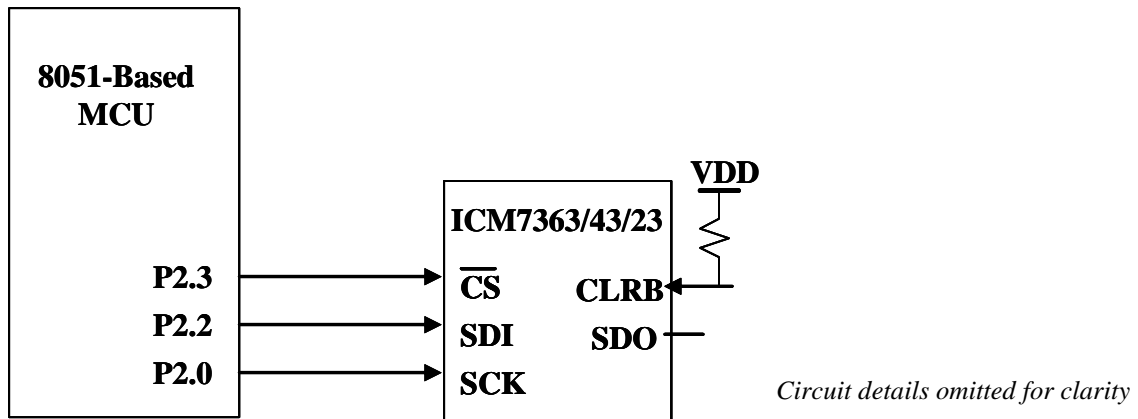


Figure 2.1: ICM7363/43/23 Communication Interface Example

Figure 2.2: Sample assembly code for hardware in Figure 2.1

```

;=====
; Description:
;   Sample codes for interfacing MCS-51 MCU to ICM7363/43/23 DAC.
;   This example uses Port 2 to communicate with the DAC.
;
;   MCS-51 pins      ICM7363/43/23 pins
;   P2.0             SCK
;   P2.1             CLRB  (optional)
;   P2.2             SDI
;   P2.3             CSB
;
;   P2.7 - used to generate pulse for oscilloscope triggering.
;
; Author:
;   Sairan Sakrani
;   21 Feb 2003
;=====

;Assembler-specific directives
$MOD51

;Hardware addresses
P_SCK EQU    P2.0           ;P2.0 - SCK
P_CLRB EQU   P2.1           ;P2.1 - CLRB
P_SDI EQU    P2.2           ;P2.2 - SDI
P_CSB EQU    P2.3           ;P2.3 - CSB

;Variables - DAC code
DACCL EQU    024H           ;DAC code LOW byte
DACCH EQU    025H           ;DAC code HIGH byte

;=====
;Interrupt Vectors
ORG        0000H
          LJMP   BEGIN      ;0000H - RESET vector
ORG        0003H
          LJMP   ISEXT0     ;0003H - External 0
ORG        000BH
          LJMP   ISTMR0     ;000BH - Timer/Counter 0
ORG        0013H
          LJMP   ISEXT1     ;0013H - External 1
ORG        001BH
          LJMP   ISTMR1     ;001BH - Timer/Counter 1
ORG        0023H
          LJMP   ISSER      ;0023H - Serial Port

ISEXT0:

```

```

ISTMR0:
ISEXT1:
ISTMR1:
ISSER: RETI                ;Empty interrupt services

;=====
;Beginning of Main Routine
;=====
ORG    0100H

BEGIN: MOV    SP,#007H      ;Init stack pointer
       MOV    IE,#000H     ;Disable interrupts

;----- Initialize interface to DACs
       CLR    P_CLRB       ;CLRB=LOW, reset DAC
       SETB   P_CSB        ;CSB(P2.3)=HIGH
       CLR    P_SCK        ;SCK(P2.0)=LOW
       CLR    P_SDI        ;SDI(P2.2)=LOW
       SETB   P_CLRB       ;CLRB=HIGH

;----- Send code to all DACs and update simultaneously
;----- Bits to sent: 1110 bbbb bbbb bbbb
       MOV    DACCL,#000H   ;Init DAC codes
       MOV    DACCH,#0E0H

ILOOP: LCALL  SNDDAC        ;Send the high and low bytes to DAC

;----- Increment DAC code
       MOV    A, DACCL      ;Increment low byte
       ADD    A, #001H
       MOV    DACCL, A      ;Save incremented low byte

       MOV    A, DACCH      ;Increment high byte if necessary
       ADDC   A, #000H

       ORL   A, #0E0H       ;Force Bit15,14,13 = 1
       ANL   A, #0EFH       ;Force Bit12=0
       MOV    DACCH, A      ;Save incremented high byte

;----- Create pulse#1 for oscilloscope triggering; when DAC data = 0000H
CHKTR2: CJNE  A,#0E0H,SKPTR1 ;[24cyc] Check if DAC data high byte = 000H
       MOV    A, DACCL      ;[12cyc]
       CJNE  A,#000H,SKPTR2 ;[24cyc] Check if DAC data low byte = 000H
       CLR    P2.7         ;[12cyc] Generate scope trigger pulse
       SETB   P2.7         ;[12cyc]
       LJMP  ILOOP         ;[24cyc] Repeat; send next DAC code

SKPTR1: NOP                ;[12cyc] NOPs for fix time interval
       NOP                ;[12cyc]
       NOP                ;[12cyc]
       NOP                ;[12cyc]
       NOP                ;[12cyc]
SKPTR2: NOP                ;[12cyc]
       NOP                ;[12cyc]
       LJMP  ILOOP         ;[24cyc] Repeat; send next DAC code

;=====
;End of Main Routine
;=====

;=====
;Subroutine:  SNDDAC
;Description: Send 16 bits of data to DAC.
;             First, CSB is set to LOW, then 16 bits are sent,
;             then, CSB is brought back HIGH.
;Arguments:  DACCH = Data high byte (Bit15 thru Bit8)
;            DACCL = Data low byte (Bit7 thru Bit0)
;Trashed:    Whatever trashed by subr SN8BIT
;*****
SNDDAC: SETB   P_CSB        ;CSB should already HIGH initially, do it anyhow
       CLR    P_CSB        ;CSB=LOW, initiate the data transfer
       MOV    A,DACCH
       LCALL  SN8BIT       ;Send high byte
       MOV    A,DACCL
       LCALL  SN8BIT       ;Send low byte

```

```

        SETB    P_CSB        ;CSB=HIGH, end the data transfer
        RET                ;Done
;=====

;=====
;Subroutine:   SN8BIT
;Description:  Send 8 bits of data on P2.2 (SDI)
;             Clock the P2.0 (SCK) as needed.
;Arguments:    A = data to be sent
;Trashed:     C,R0
;*****
SN8BIT: MOV     R0, #8        ;Init loop counter
SN8LUP: CLR     P_SCK        ;CLK=L
        RLC     A            ;Set C=bit to be sent
        MOV    P_SDI,C       ;Set SDI=bit to be sent
        SETB   P_SCK        ;CLK=H, ICM7363 latches data on SDI at +ve edge

        DJNZ   R0, SN8LUP    ;Decrement counter, loop if more bits to sent
        CLR    P_SCK        ;CLK=L
        RET                ;Done
;=====

END

```