A DIVISION OF CIRRUS LOGIC

AN138

Application Note

INTERFACING THE CS4954/5 TO THE SX28AC MICROCONTROLLER.

INTRODUCTION

This application note details the interface of the Crystal® CS4954/5 TV encoder to a Scenix SX28AC microcontroller. This note takes the reader through a simple example describing how to communicate with the encoder. All routines discussed are included in the Appendix at the end of this note. The SX28AC is compatible with the MicroChip PIC16C5X family. For more details on microcontrollers see the respective manufacturer's web site.

ENCODER DIGITAL INTERFACE

The CS4954/5 interfaces to the SX28AC either through a parallel (13 wires) or a serial (2 wires) interface. Figure 1 depicts the interface between the two devices.

The Encoder parallel interface consist of 13 lines, eight of which are data lines and the three control lines. The serial interface consist of one clock and one data line.

PDAT[7:0]	Host parallel data port	I/0
$\overline{WR}$	Write strobe, active low	I
$\overline{RD}$	Read strove, active low	I
ADDR	Address enable, active low	I

Table 1. Parallel Interface Description

SDA	Serial data port	Open collector
SCL	Serial input clock	Open collector

Table 2. Serial Interface Description

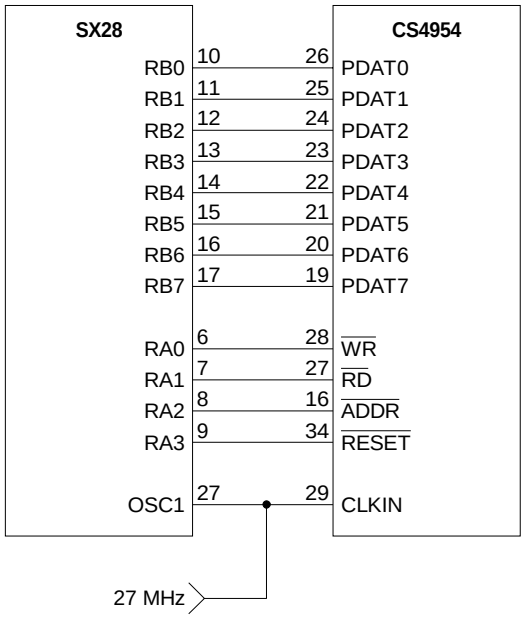

Figure 1. Parallel interface to a microcontroller.

The 8-bit parallel interface works as follows. When $\overline{ADDR}$ strobe is low, the address register is enabled for writing or reading (timing details not shown here, see the CS4954 datasheet for timing information). As illustrated in Figures 2 and 3 the rising edge of the $\overline{WR}$ strobe active low with $\overline{ADDR}$ strobe still active will enable selecting the register address you want to access. The following active low $\overline{WR}$ or $\overline{RD}$ strobe, while $\overline{ADDR}$ strobe is disabled (HIGH) will give access to the register content for read or write. There is no automatic register increment/decrement function. To access a different register, you must select it first and then read or write to it. (See Appendix B for assembly code example.)

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

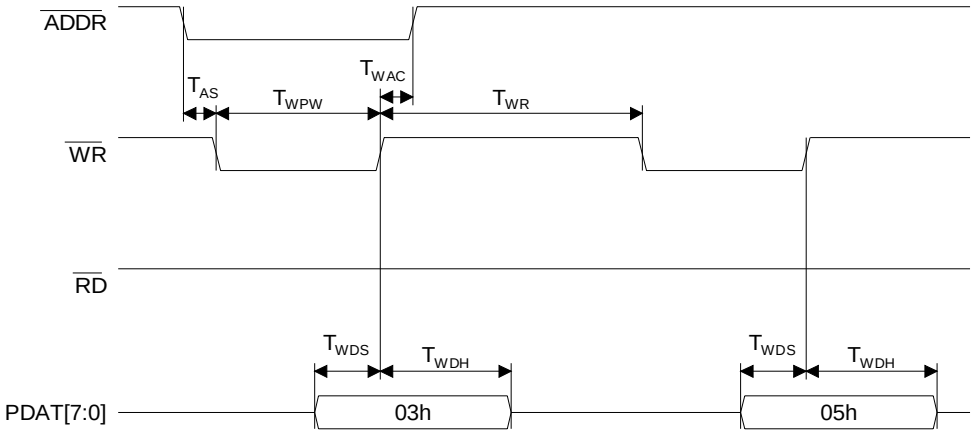

Figure 2. 8-bit parallel host port timing : write 05h in register 03h.

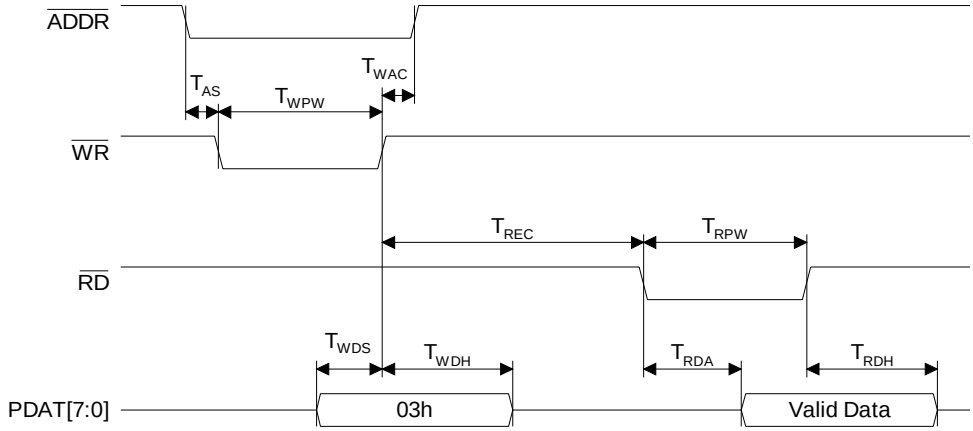

Figure 3. 8-bit parallel host port timing : read data in register 03h.

Parameter	Symbol	Min	Max	Unit
Parallel Port Timing				
Write recovery time	T_{WR}	60	-	ns
Address from write hold time	T_{WAC}	0	-	ns
Write pulse width	T_{WPW}	40	-	ns
Address setup time	T_{AS}	3	-	ns
Write data setup time	T_{WDS}	8	-	ns
Write data hold time	T_{WDH}	3	-	ns
Write-Read/Read-Write recovery time	T_{REC}	50	-	ns
Read pulse width	T_{RPW}	30	-	ns
Read data access time	T_{RDA}	-	40	ns
Read data hold time	T_{RDH}	10	50	ns

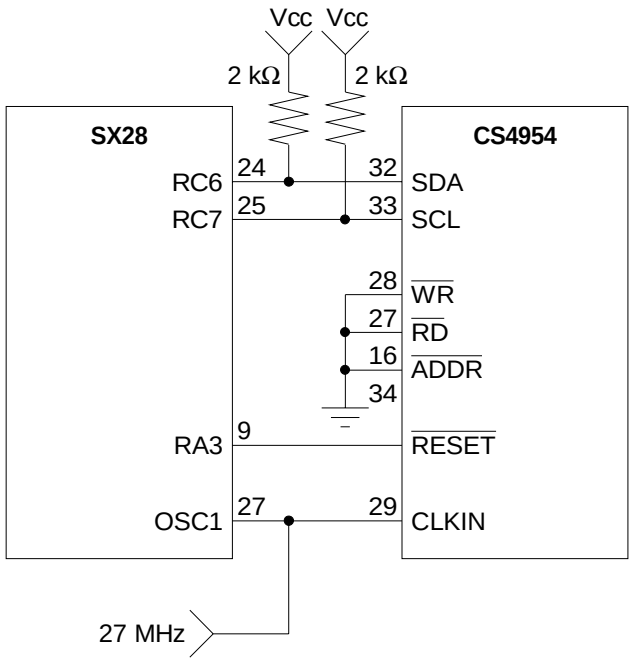

Figure 4. Serial interface to a microcontroller.

The serial interface is a 2-wire, I^2C compatible interface. The SDA and SCL pins are open drains, therefore pull-ups are required for proper operation. When there is no activity on the bus, the lines are HIGH. This is the idle state. The CS4954/5 can act as an I^2C slave device only. It cannot be a bus master. In the case where there is more than one device on the I^2C bus that has the same default Station Address as the CS4954/5 (default is 00), you need to sequence the reset of all the devices that are in conflict and change the Station Address of the CS4954/5 to avoid addressing problems. (See Appendix A for assembly code example.)

The following communication example is for 3-byte mode. To perform a write cycle, you must send the I^2C Write Station Address, the Register Address and the data. The read operation consist of sending the I^2C Write Station Address, the Register Address, and the I^2C Read Station Address. The data will then be placed on the I^2C bus. The I^2C protocol also includes start and stop condition as well as acknowledge messages. The CS4954/5 supports auto-increment of the register address. Here are some examples:

start / station address + write bit / ack / subaddress / ack / data / ack / data / ack / ... / stop

start / station address + write bit / ack / subaddress / ack / station address + read bit / ack / data / stop

Parameter	Symbol	Min	Typ	Max	Unit
I^2C Host Port Timing					
SCL Frequency	F_{clk}	100	-	1000	KHz
Clock Pulse High Time	T_{sph}	0.1	-	-	µs
Clock Pulse Low Time	T_{spl}	0.7	-	-	µs
Hold Time (Start Condition)	T_{sh}	100	-	-	ns
Setup Time (Start Condition)	T_{ssu}	100	-	-	ns
Data Setup Time	T_{sds}	50	-	-	ns
Rise Time	T_{sr}	-	-	1	µs
Fall Time	T_{sf}		-	0.3	µs
Setup Time (Stop Condition)	T_{ss}	100	-	-	ns
Bus Free Time	T_{buf}	100	-	-	ns
Data Hold Time	T_{dh}	0	-	-	ns
SCL Low to Data Out Valid	T_{vdo}	-	-	600	ns

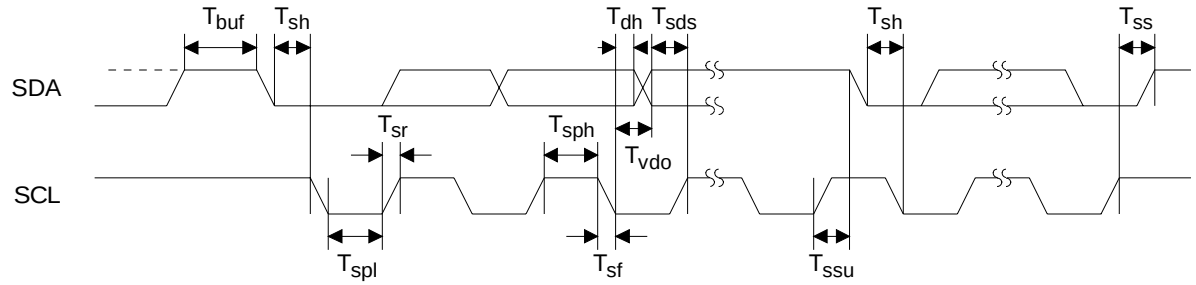

Figure 5. I²C Host Port Timing.

Understanding the station address of the CS4954/5 and how to change it:

The CS4954/5 address register provides 7 bits for addressing, therefore you are limited to 128 possible addresses. These 7 bits represents the MSBs of the Station Address. According to the I²C specification, the LSB is the READ (1) or WRITE (0) strobe.

For instance, 6Ah (in binary 0110 1010) is written in register 0x0Fh:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I²C add.	res	1	1	0	1	0	1	0

Remember, these bits are the MSBs of the Station Address:

	Base Address (MSBs)	Read/Write strobe	Base Address + Read/Write strobe
Read	1101 010	1	1101 0101 (D5)
Write	1101 010	0	1101 0100 (D4)

The value 6Ah written in the I²C address register will effectively change the Station Address to D5h for reads and D4h for writes. When reading or writing to register 0x0Fh, Bit 7 should be ignored.

SOFTWARE DESCRIPTION

Note : Refer to Appendices for complete code listing.

The following flow chart describes the initialization of the microcontroller, configuring the CS4954 for access in parallel mode, and performing a read and write on the CS4954 via the parallel interface. The same procedure is used with the serial mode interface. This particular example shows how to enable the color bar generator.

The reset pin does not need to be controlled by software, a simple RC network is sufficient. After a reset the CS4954/5 is accessible through the parallel port if both $\overline{RD}$ and $\overline{WR}$ pins are HIGH on the rising edge of the reset. If both pins are LOW the serial I²C mode is enabled and the parallel port becomes a general purpose I/O port. Also after a reset, the CS4954's DACs are disabled and powered down. Reset must be held low for at least 100 ns.

The register address must be selected to prior to any type of access, Read or Write. To access register 00h, set $\overline{ADDR}$ low, then set $\overline{WR}$ low, write the register address on the data bus (in this case 00h) then de-assert $\overline{WR}$ and $\overline{ADDR}$. Once this is, done all operations on the control port will reflect the content of this particular register.

By writing 10h in register 00h the CS4954 will be in master mode and will not need timing cues to output the color bar. In the previous step, we have selected register 00h. To write data in the register, set $\overline{WR}$ low, write data to the data bus and de-assert the $\overline{WR}$ pin. If you use a fast micro-controller like the SX28AC make sure you allow for the 60 ns delay between writes.

Follow the same steps, select the register 03h and then write the data to the register. This will enable the color bar generator.

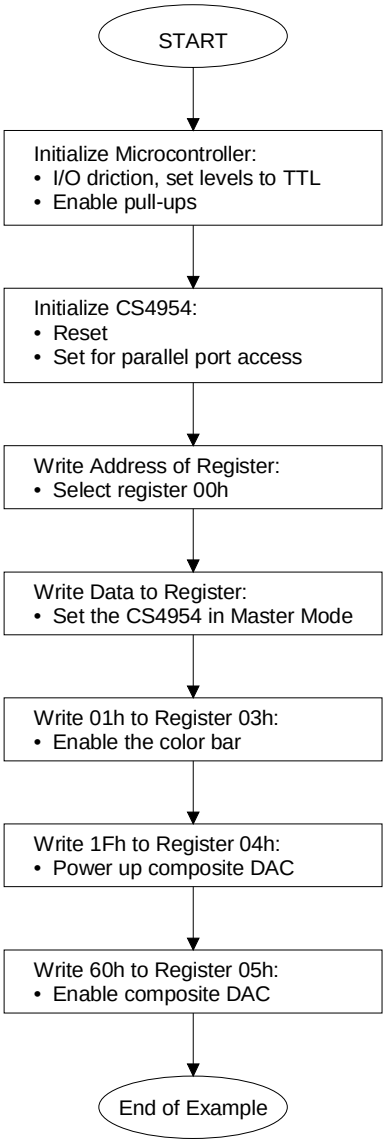

Figure 6.

This register allows the DACs to be powered up. Remember not to power up all six DACs in low impedance mode; this will result in damaging the part. (See datasheet for details.)

The composite DAC is now enabled. An NTSC 100% amplitude, 100% saturation signal is available on the composite DAC. By default, at power up the CS4954 will be configured for NTSC.

MAXIMUM CLOCK RATE

The CS4954/5 requires a 27 MHz clock (twice the pixel rate). Since the SX28AC can accept a clock rate up to 50 MHz, the same clock was used by the micro-controller. If you want to use the code listed in the appendixes at a higher speed, ensure you observe the CS4954/5 timing requirements.

APPENDIX A

```
;-------------------------------------------------------------------------
;  Copyright CIRRUS LOGIC, INC. 1998
;-------------------------------------------------------------------------
;
;
;  54DEMO_I.SRC: Example code illustrating the initialization and configuration
;           of the CS4954/55 Digital Video Encoder via the I2C interface.
;
;           This simple code assumes the microcontroller host is connected
;           directly to the CS4954/55, with the micro acting as the I2C
;           bus master, the CS4954 acting as a slave device, and no other
;           devices on the I2C bus.
;
;  PROCESSOR:    Scenix SX-28 R.S.
;
;
;
;-------------------------------------------------------------------------




;-------------------------------------------------------------------------
; DEVICE CONFIGURATION
; SX-28 (28 pin)
; 4 pages of EEPROM (4 X 512 bytes)
; 8 banks of RAM (128 bytes)
; Extended stack (8 words deep)
; On reset, execution starts at label "main"
;-------------------------------------------------------------------------

            device      pins28,pages4,banks8,oschs
            device      turbo,stackx,optionx
            id          '54 Init'
            reset       main


;-------------------------------------------------------------------------
; EQUATES
;-------------------------------------------------------------------------
```

```
; Control signals connected to CS4954 inputs.  Refer to CS4954 datasheet.
wr_pin      =           ra.0  ;Port A pin connected to CS4954's WR*
rd_pin      =           ra.1  ;Port A pin connected to CS4954's RD*
padr_pin    =           ra.2  ;Port A pin connected to CS4954's PADDR
rst_pin     =           ra.3  ;Port A pin connected to CS4954's RESET

; I2C interface signals
sda_pin     =           rc.6  ;Port C pin - I2C data line
scl_pin     =           rc.7  ;Port C pin - I2C clock line

; Miscellaneous constants
bdelay      =           40h   ;delay loop counter
CS4954_W    =           00h   ;CS4954 I2C "WRITE" addr (address + Write = 0)
CS4954_R    =           01h   ;CS4954 I2C "READ" addr (address + Write = 1)



;-------------------------------------------------------------------------
; VARIABLES - All reside in bank 0
;-------------------------------------------------------------------------
            org         $10

VARS        EQU         $
data_in     ds          1     ;storage for data read from CS4954
data_out    ds          1     ;storage for data to be written to CS4954
reg_addr    ds          1     ;holds address of register to be read/written
i2c_data    ds          1     ;temp storage for byte shifted in/out on I2C bus
temp        ds          1
bcount      DS          1     ;bit counter for I2C read/writes



;-------------------------------------------------------------------------
; MAIN PROGRAM (Memory Page 0)
;-------------------------------------------------------------------------

            org         $00
main
```

```
; Initialize the uController's ports

;set IO direction
mov         m,#$F             ;set mode for io direction
mov          ra,#%1111        ;init ra - all ones
mov         !ra,#%0000        ;ra3-0 outputs
mov          rb,#%11111111          ;init rb
mov         !rb,#%00000000          ;start out as outputs
mov          rc,#%11111111          ;init rc - all ones
mov         !rc,#%00000000          ;start out as outputs

;turn on pullups
mov         m,#$E             ;set mode for pullups
mov         !ra,#%0000
mov         !rb,#%00000000
mov         !rc,#%00000000

;set input levels to TTL
mov         m,#$D             ;set mode for io level
mov         !ra,#%0000
mov         !rb,#%00000000
mov         !rc,#%00000000
mov         m,#$F

; all variables in bank 0
bank vars


; Initialize the CS4954 for i2c access.

page        cfg_i2c
call        cfg_i2c

; As an example, generate a color bar

;register 0 = 10h
mov         w,#$0
mov         reg_addr,w        ;write to register 0
mov         w,#$10
```

```
mov         data_out,w        ;write data byte of $10
page        write_reg
call        write_reg

;register 3 = 01h
mov         w,#$3
mov         reg_addr,w        ;write to register 3
mov         w,#$1
mov         data_out,w        ;write data byte of $01
page        write_reg
call        write_reg

;register 4 = 1Fh
mov         w,#$4
mov         reg_addr,w        ;write to register 4
mov         w,#$1F
mov         data_out,w        ;write data byte of $1F
page        write_reg
call        write_reg

;register 5 = 60h
mov         w,#$5
mov         reg_addr,w        ;write to register 5
mov         w,#$60
mov         data_out,w        ;write data byte of $60
page        write_reg
call        write_reg

; An example register read - not used for anything here.

; read regsiter 4
mov         w,#$4
mov         reg_addr,w
page        read_reg
call        read_reg
mov         w,data_in

;loop here forever
jmp         $
```

```
;-------------------------------------------------------------------------
; I2C INTERFACE ROUTINES (Memory page 1)
;-------------------------------------------------------------------------
                        ORG     200h


;-------------------------------------------------------------------------
; SUBROUTINE: CFG_I2C
; Configure the CS4954 for I2C interface mode by setting RD* and WR* low and
; then holding the CS4954 in reset for > 100ns.
;
; Entry: Nothing assumed.
;
; Exit:  CS4954/55 out of reset
;
; Calls: Nothing
;
;-------------------------------------------------------------------------

cfg_i2c     clrb        rd_pin              ;RD* and WR* held low
            clrb        wr_pin
            clrb        rst_pin                   ;pull CS4954's reset low
            nop
            nop                             ;delay for at least 100ns
            nop
            nop
            setb        rst_pin                   ;bring CS4954's reset high again

            retp                            ;now in I2C interface mode


;-------------------------------------------------------------------------
; SUBROUTINE: WRITE_REG
; Writes a byte to a CS4954 register using the I2C interface.
;
; Entry: REG_ADDR holds the CS4954 register address to be written to.
;        DATA_OUT holds the data byte to be written.
;
; Exit:
;
```

```
; Calls: SND_START
;        SND_BYTE
;        SND_STOP
;
;-------------------------------------------------------------------------

write_reg   ; send a start-of-frame delimeter
            call        snd_strt

            ; send the I2C addr for CS4954
            mov         w,#CS4954_W
            mov         i2c_data,w
            call        snd_byte

            ; send the CS4954 register address
            mov         w,reg_addr
            mov         i2c_data,w
            call        snd_byte

            ; send data to the addressed register
            mov         w,data_out
            mov         i2c_data,w
            call        snd_byte

            ; send the end-of-frame delimeter
            call        snd_stop

            retp


;-------------------------------------------------------------------------
; SUBROUTINE: READ_REG
; Read a byte from a CS4954 register using the I2C interface.
;
; Entry: REG_ADDR holds the CS4954 register address to be read.
;        DATA_IN holds the data byte to be written.
;
; Exit:  DATA_IN = byte read
;
; Calls: SND_START
```

```
;        SND_BYTE
;      GET_BYTE
;        SND_STOP
;
;-------------------------------------------------------------------------------

read_reg    ; send a start-of-frame delimeter
            call        snd_strt

            ; send the I2C WRITE addr for CS4954
            mov         w,#CS4954_W
            mov         i2c_data,w
            call        snd_byte

            ; send the CS4954 register address
            mov         w,reg_addr
            mov         i2c_data,w
            call        snd_byte

            ; send an end-of-frame delimeter
            call        snd_stop

            ; now do a I2C read cycle to get the register's contents

            ; send a start-of-frame delimeter
            call        snd_strt

            ; send the I2C READ addr for CS4954
            mov         w,#CS4954_R
            mov         i2c_data,w
            call        snd_byte

            ; read the register
            call        get_byte
            mov         w,i2c_data
            mov         data_in,w

            ; send the end-of-frame delimeter
            call        snd_stop
```

```
            retp


;-------------------------------------------------------------------------------
; SUBROUTINE: SND_BYTE
; Send a byte on the I2C bus.
;
; Entry: I2C_DATA holds the byte to be written.
;
; Exit:  I2C_DATA contents destroyed
;
; Calls: DELAY
;        I2C_ACK
;
;-------------------------------------------------------------------------------

snd_byte    mov         w,#08h            ;set a bit counter for 8 bits
            mov         bcount,w

            ; shift out 8 data bits
:next_bit   clrb        scl_pin           ;set clock low
            call        delay             ;make sure it is not a stop condition
            movb        sda_pin,i2c_data.7;put the bit on data line
            call        delay             ;make sure is it is not a start condition
            setb        scl_pin                 ;set clock high
            call        delay             ;let them settle
            rl          i2c_data          ;shift to next bit
            djnz        bcount,:next_bit  ;entire byte written yet?
            clrb        scl_pin                 ;bring clock low at end of last bit

            ; create and ACK and return
            call        i2c_ack
            retp


;-------------------------------------------------------------------------------
; SUBROUTINE: GET_BYTE
; Read a byte from the I2C bus.
;
; Entry: Nothing assumed.
```

```
;
; Exit:  I2C_DATA holds byte read.
;
; Calls: DELAY
;        I2C_NACK
;
;-------------------------------------------------------------------------

get_byte    mov         w,#08h            ;set a bit counter for 8 bits
            mov         bcount,w

            ; make sda_pin an input
            mov         !rc,#%01000000

            ; shift in 8 data bits
:next_bit2  setb        scl_pin           ;set clock high
            call        delay
            movb        i2c_data.7,sda_pin;sample the bit on data line
            call        delay
            clrb        scl_pin                ;set clock low
            call        delay
            rl          i2c_data          ;shift to next bit
            djnz        bcount,:next_bit2 ;entire byte read yet?
            rl          i2c_data          ;shift last bit into place

            ; make sda_pin an output again
            mov         !rc,#%00000000

            ; create and ACK and return
            call        i2c_nack
            retp

;-------------------------------------------------------------------------
; SUBROUTINE: I2C_ACK
; Generate an I2C ACK (acknowledge).
;
; Entry: Nothing assumed.
;
; Exit:
;
```

```
; Calls: DELAY
;
;-------------------------------------------------------------------------

i2c_ack                 call  delay

            ;rx pulls data low here (forced here - should poll for data = 0)
            clrb        sda_pin
            call        delay

            ;tx generates another clock during the ACK
            setb        scl_pin
            call        delay
            clrb        scl_pin
            call        delay

            retp


;-------------------------------------------------------------------------
; SUBROUTINE: I2C_NACK
; Generate an I2C ACK (NOT acknowledge).
;
; Entry: Nothing assumed.
;
; Exit:
;
; Calls: DELAY
;
;-------------------------------------------------------------------------

i2c_nack    call        delay

            ;tx forces data high
            setb        sda_pin
            call        delay

            ;tx generates another clock during the nACK
            setb        scl_pin
            call        delay
```

```
            clrb        scl_pin
            call        delay

            retp


;-------------------------------------------------------------------------------
; SUBROUTINE: SND_STRT
; Send I2C start-of-frame delimeter.
;
; Entry: Nothing assumed.
;
; Exit:
;
; Calls: DELAY
;
;-------------------------------------------------------------------------------

snd_strt    ;ensure both data and clock high
            setb        sda_pin
            setb        scl_pin
            call        delay

            ;bring data low during clock high
            clrb        sda_pin
            call        delay

            ;now bring clock low to finish start delimeter
            clrb        scl_pin
            call        delay
            retp


;-------------------------------------------------------------------------------
; SUBROUTINE: SND_STOP
; Send I2C end-of-frame delimeter.
;
; Entry: Nothing assumed.
;
; Exit:
```

```
;
; Calls: DELAY
;
;-------------------------------------------------------------------------

snd_stop    ;make sure clock and data low
            clrb        sda_pin
            call        delay
            clrb        scl_pin
            call        delay

            ;next bring clock high
            setb        scl_pin
            call        delay

            ;now bring data high while clock high
            setb        sda_pin
            call        delay       ;make a gap at end of message
            call        delay
            retp


;-------------------------------------------------------------------------
; SUBROUTINE: DELAY
; Create a delay processing a busy loop.
;
; Entry: BDELAY holds loop counter.
;
; Exit:
;
; Calls: Nothing
;
;-------------------------------------------------------------------------

delay       mov         w,#bdelay   ;i2c bit delay loop constant
            mov         temp,w
bitloop                 djnz  temp,bitloop;loop for bit time
            retp
```

APPENDIX B :

```
;-------------------------------------------------------------------------
;  Copyright CIRRUS LOGIC, INC. 1998
;-------------------------------------------------------------------------
;
;
;  54DEMO_P.SRC: Example code illustrating the initialization and configuration
;            of the CS4954/55 Digital Video Encoder via the parallel host
;                interface.
;
;
;  PROCESSOR:    Scenix SX-28, Microchip PIC 16C5X
;
;
;-------------------------------------------------------------------------


;-------------------------------------------------------------------------
; DEVICE CONFIGURATION
; SX-28 (28 pin)
; 4 pages of EEPROM (4 X 512 bytes)
; 8 banks of RAM (128 bytes)
; Extended stack (8 words deep)
; On reset, execution statrs at label "main"
;-------------------------------------------------------------------------

            device      pins28,pages4,banks8,oschs
            device      turbo,stackx,optionx
            id          '54 Init'
            reset       main


;-------------------------------------------------------------------------
; EQUATES
;-------------------------------------------------------------------------

; Control signals connected to CS4954 inputs.  Refer to CS4954 datasheet.
wr_pin      =           ra.0  ;Port A pin connected to CS4954's WR*
rd_pin      =           ra.1  ;Port A pin connected to CS4954's RD*
```

```
padr_pin    =           ra.2  ;Port A pin connected to CS4954's PADDR
rst_pin     =           ra.3  ;Port A pin connected to CS4954's RESET


;-------------------------------------------------------------------------
; VARIABLES - All reside in bank 0
;-------------------------------------------------------------------------
            org         $10

VARS        EQU         $
data_in     ds          1     ;storage for data read from CS4954
data_out    ds          1     ;storage for data to be written to CS4954
reg_addr    ds          1     ;holds address of register to be read/written


;-------------------------------------------------------------------------
; MAIN PROGRAM (Memory Page 0)
;-------------------------------------------------------------------------

            org         $00
main

            ; Initialize the uController's ports

            ;set IO direction
            mov         m,#$F             ;set mode for io direction
            mov          ra,#%1111        ;init ra - all ones
            mov         !ra,#%0000        ;ra3-0 outputs
            mov          rb,#%11111111          ;init rb
            mov         !rb,#%00000000          ;start out as outputs
            mov          rc,#%11111111          ;init rc - all ones
            mov         !rc,#%00000000          ;start out as outputs

            ;turn on pullups
            mov         m,#$E             ;set mode for pullups
            mov         !ra,#%0000
            mov         !rb,#%00000000
            mov         !rc,#%00000000

            ;set input levels to TTL
```

```
mov         m,#$D             ;set mode for io level
mov         !ra,#%0000
mov         !rb,#%00000000
mov         !rc,#%00000000
mov         m,#$F

; all variables in bank 0
bank vars

; Initialize the CS4954 for parallel access

page        cfg_para
call        cfg_para

; For this example, enable color bar generation.

;register 0 = 10h
mov         w,#$0
mov         reg_addr,w        ;write to register 0
mov         w,#$10
mov         data_out,w        ;write data byte of $10
page        write_reg
call        write_reg

;register 3 = 01h
mov         w,#$3
mov         reg_addr,w        ;write to register 3
mov         w,#$1
mov         data_out,w        ;write data byte of $01
page        write_reg
call        write_reg

;register 4 = 1Fh
mov         w,#$4
mov         reg_addr,w        ;write to register 4
mov         w,#$1F
mov         data_out,w        ;write data byte of $1F
page        write_reg
call        write_reg
```

```
            ;register 5 = 60h
            mov         w,#$5
            mov         reg_addr,w        ;write to register 5
            mov         w,#$60
            mov         data_out,w        ;write data byte of $60
            page        write_reg
            call        write_reg

            ; Read register 5 back to demonstrate a register read.

            mov         w,#$5
            mov         reg_addr,w        ;register address = 5
            page        read_reg
            call        read_reg          ;read the register
            mov         w,data_in

            ;loop here forever
            jmp         $

;-------------------------------------------------------------------------
; I2C INTERFACE ROUTINES (Memory page 1)
;-------------------------------------------------------------------------
                        ORG     200h


;-------------------------------------------------------------------------
; SUBROUTINE: CFG_PARA
; Configure the CS4954 for parallel interface mode by setting RD* and WR* high and
; then holding the CS4954 in reset for > 100ns.

; Entry: Nothing assumed.
;
; Exit:  CS4954/55 out of reset
;
; Calls: Nothing
;
;-------------------------------------------------------------------------

cfg_para    setb        rd_pin            ;RD* and WR* held high
            setb        wr_pin
```

```
            clrb        rst_pin                 ;pull CS4954's reset low
            nop
            nop                         ;delay for at least 100ns
            nop
            nop
            setb        rst_pin                 ;bring CS4954's reset high again

            retp                        ;now in parallel interface mode


;-----------------------------------------------------------------------------
; SUBROUTINE: READ_REG
; Read a byte from a CS4954 register using the parallel host interface.
;
; Entry: REG_ADDR holds the CS4954 register address to be read.
;        DATA_IN holds the data byte to be written.
;
; Exit:  DATA_IN = byte read
;
; Calls: Nothing
;
;-----------------------------------------------------------------------------

read_reg    ; config port b for output
            mov         w,#$F
            mov         m,w
            mov         !rb,#$0

            ; send register address
            clrb        padr_pin            ;set PADR* active low
            clrb        wr_pin              ;set WR* active low
            mov         w,reg_addr
            mov         rb,w                ;write register address
            setb        wr_pin              ;de-assert WR* and PADR*
            setb        padr_pin            ;latch register address

            ; config port b for input
            mov         w,#$F
            mov         m,w
            mov         !rb,#$FF
```

```
            ; read register
            clrb        rd_pin          ;assert RD* active low
            nop                         ;delay > 40ns
            nop
            mov         w, rb
            mov         data_in, w      ;read register data
            setb        rd_pin          ;de-assert RD*

            retp

;-------------------------------------------------------------------------
; SUBROUTINE: WRITE_REG
; Writes a byte to a CS4954 register using the parallel host interface.
;
; Entry: REG_ADDR holds the CS4954 register address to be written to.
;        DATA_OUT holds the data byte to be written.
;
; Exit:
;
; Calls: Nothing
;
;-------------------------------------------------------------------------

;
; Write to a CS4954 register via parallel interface
;
write_reg   ; config port b for output
            mov         w,#$F
            mov         m,w
            mov         !rb,#$0

            ; send register address
            clrb        padr_pin        ;set PADR* active low
            clrb        wr_pin          ;set WR* active low
            mov         w,reg_addr
            mov         rb,w            ;write register address
            setb        wr_pin          ;de-assert WR* and PADR*
            setb        padr_pin        ;latch register address
```

```
; write data to register
nop
nop                         ;delay > 60ns between writes
clrb        wr_pin          ;assert WR* active low
mov         w, data_out
mov         rb, w           ;write data
setb        wr_pin          ;de-assert WR*

retp
```

• Notes •


SMART
Analog
™