MicroConverter™ Technical Note - uC001
Using the ADuC812 I²C Interface

INTRODUCTION

This Application Note describes the hardware and software implementation of an I²C channel using the ADuC812 's on-chip I²C interface. The I²C channel described is a simple single master to single slave implementation as shown in figure 1 below.

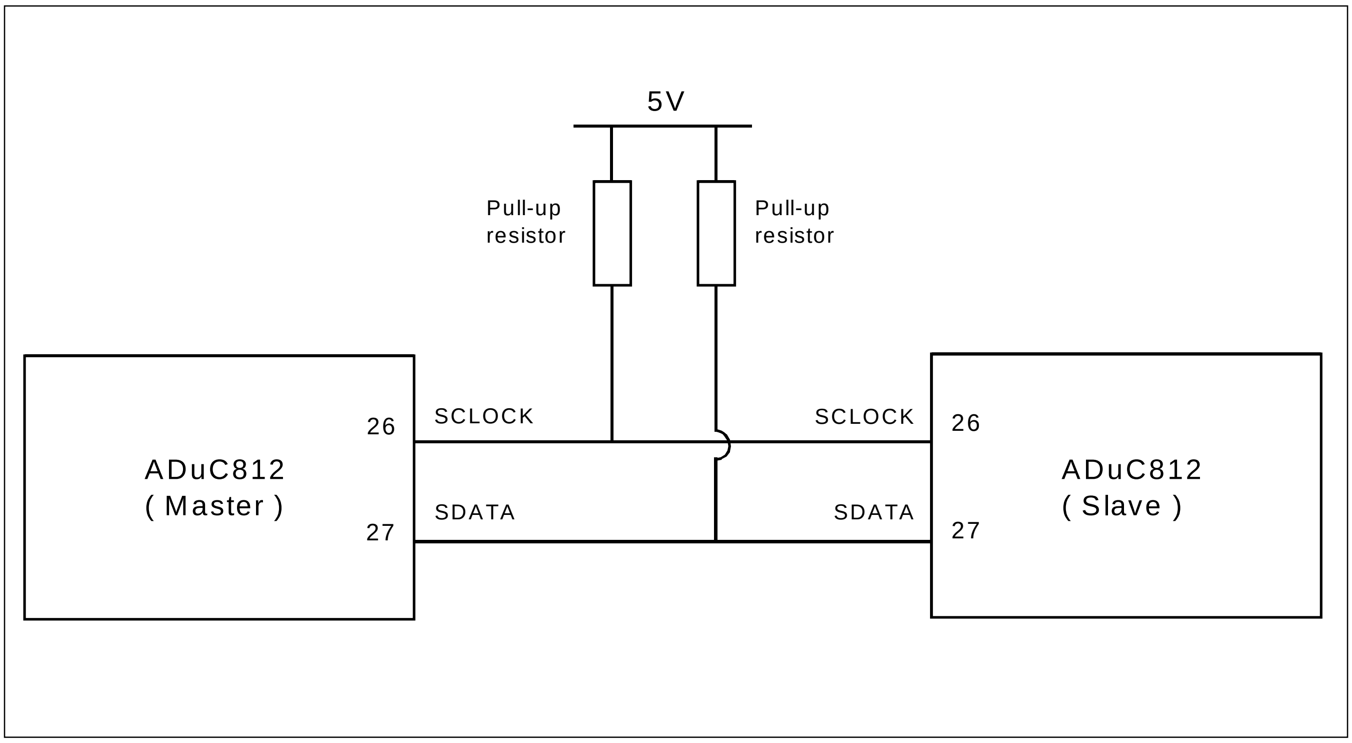


Figure 1. I²C Block Diagram

One Technology Way, P.O. Box 9106, Norwood. MA 02062-9106, U.S.A.
Tel: 617/329-4700 Fax: 617/326-8703

I2C INTERFACE OVERVIEW

The I^{2}C (Inter Integrated Circuit) is a two-wire serial communication system developed by Philips which allows multiple masters and multiple slaves connected via two wires (SCLOCK, SDATA).

The SCLOCK signal controls the data transfer between master and slave. The SDATA signal is used to transmit or receive data. Both lines are bidirectional. The bit-rate is controlled by the SCLOCK line.

In an I^{2}C interface there is at least a single master and a single slave although I^{2}C can also support multi-master multi-slave configurations. The master generates the clock whereas the slave is driven by the clock. A typical data transfer sequence is shown in figure 2.

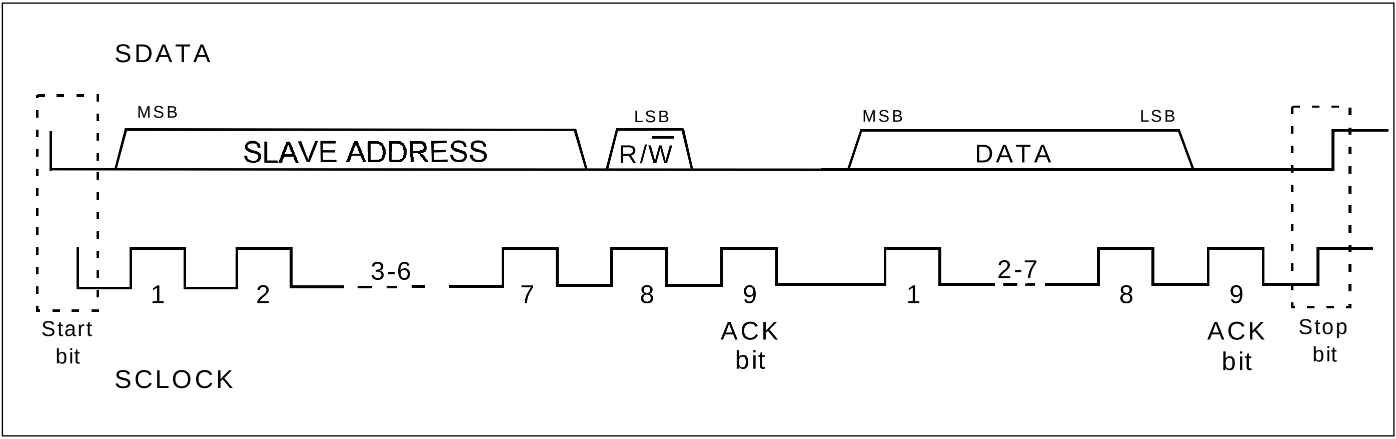


Figure 2. Typical I^{2}C transfer sequence

This sequence starts with a Start bit which is generated by the master. A start condition is indicated by a HIGH to LOW transition on the SDATA line while SCLOCK is HIGH as shown in figure 3.

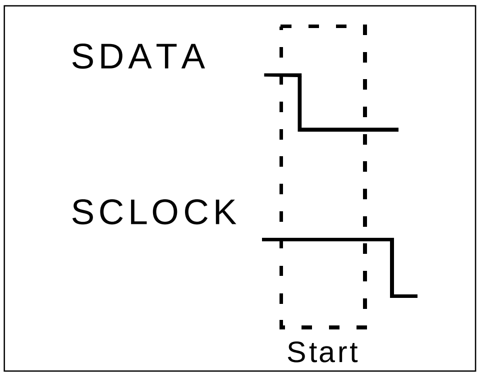


Figure 3. Start Condition on the I2C bus

After the start condition the master sends a byte (MSB first) on the SDATA line which contains the slave address and a R/$\overline{W}$ status bit. The first seven bits make up the slave address. The eight bit which is the LSB (least significant bit) determines the direction of the message (see figure 4). A '0' means that the master will write data to a selected slave. A '1' in this bit means that the master will read data from the slave. These operations will only occur once a valid acknowledge bit has been first received from the slave.

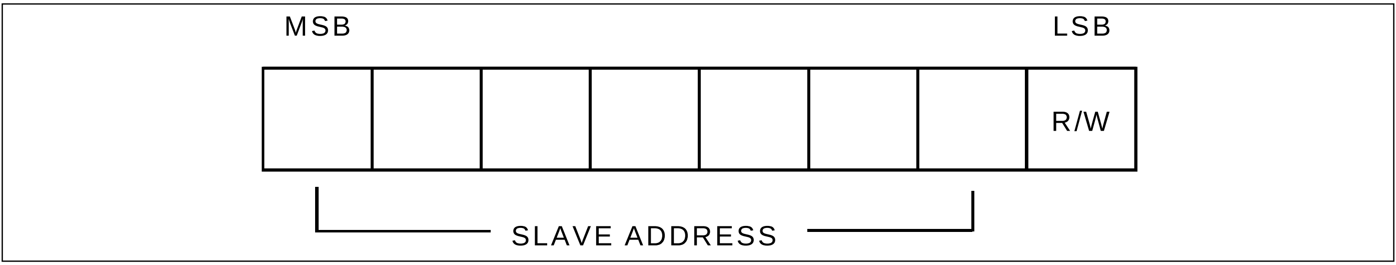


Figure 4. First byte after the START condition

When the master sends the address, each slave device in the system compares the first seven bits after the start condition with its own address. If they match, the slave device considers itself addressed by the master and replies by sending an acknowledge (see figure 5). An acknowledge is seen as a LOW level on the SDATA line at the 9th clock period and should be generated by the slave at the end of each byte in the transmission.

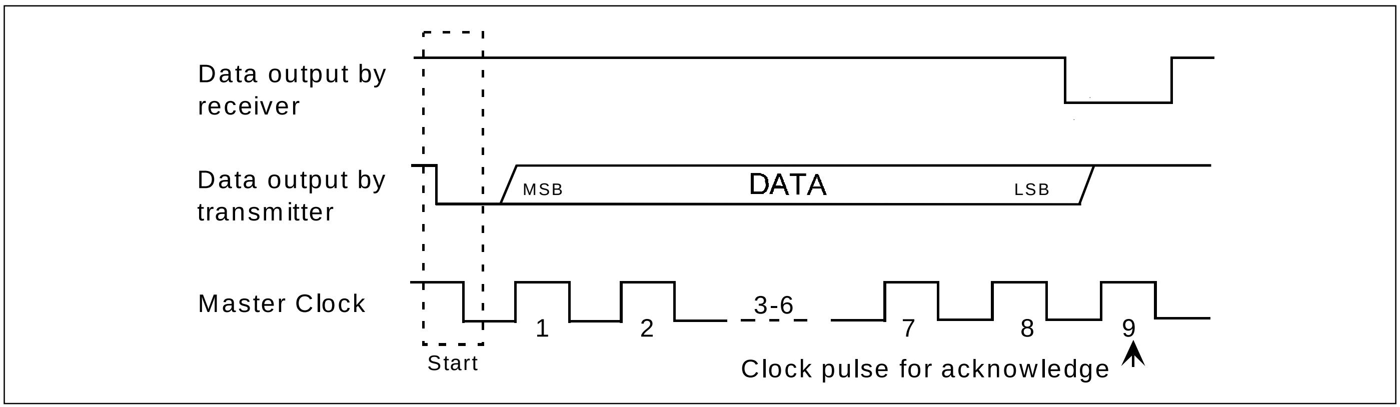


Figure 5. Acknowledge on the I^{2}C bus

If there is no acknowledge or if the transfer is completed, the master generates a STOP condition defined by a LOW to HIGH transition on the SDATA line while SCLOCK is HIGH (see figure 6).

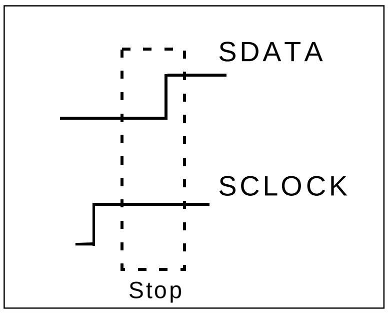


Figure 6. Stop Condition on the I^{2}C bus

I2C IMPLEMENTATION ON THE ADuC812

This section describes the I^{2}C implementation on the ADuC812 MicroConverter. The ADuC812 I2C interface provides both hardware slave and software master operating modes. The SCLOCK and SDATA are pins 26 and 27 respectively. On power on or reset, the I^{2}C interface defaults to slave operation.

Three Special Function Registers (SFRs) are used to control this interface :

I2CADD : Holds the 7-bit address of the ADuC812 device on the bus (default value = 55H).

7-bit address procedure :
If the slave holds in its I2CADD register the value 44H, the master must send the value 88H to open communication with the slave. Because of the 7-bit address, the slave knows automatically that the LSB (least significant bit) is the Read / Write status bit. Therfore, the slave compares only the 7 upper bits to his own address. To make a complete word (8 bits), the slave adds a zero for the MSB (most significant bit). The result of this completion is compared to his own address (see figure 7 for an illustration).

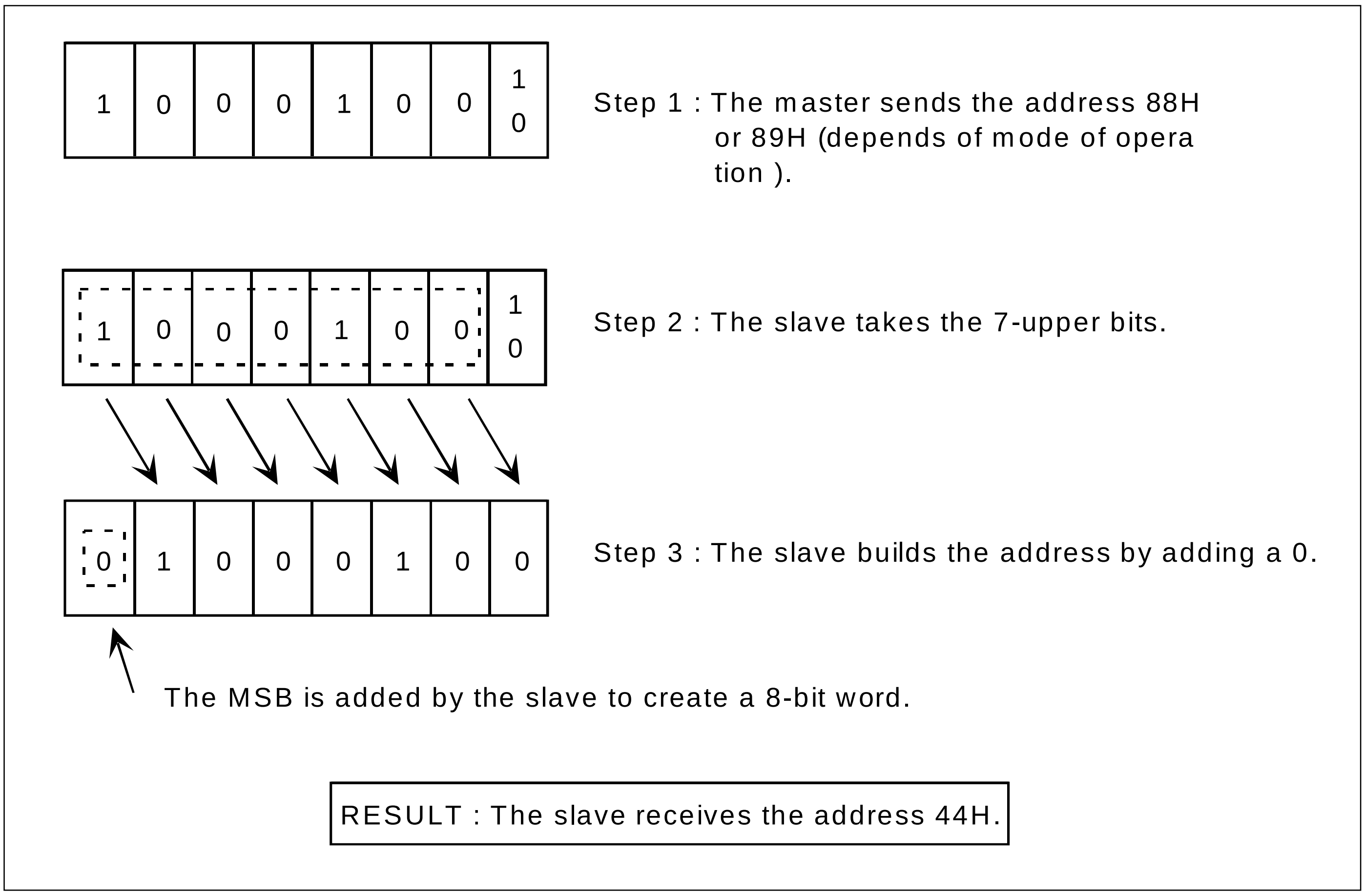

1 0 0 0 1 0 0 1/0
Step 1 : The master sends the address 88H or 89H (depends of mode of operation).
1 0 0 0 1 0 0 1/0
Step 2 : The slave takes the 7-upper bits.
0 1 0 0 0 1 0 0
Step 3 : The slave builds the address by adding a 0.
The MSB is added by the slave to create a 8-bit word.
RESULT : The slave receives the address 44H.

Figure 7. Slave address procedure

I2CDAT : Holds the 8 data bits to be received or to be transmitted.

I2CCON : Holds configuration/control bits for master/slave mode of operation (see figure 7).

The slave uses the three lower bits of the I2CCON register. As the slave implementation is hardware, the slave will automatically detect a START bit, an acknowledge, an interrupt or STOP bits.

The master uses the four upper bits of the I2CCON register to control the generation of the master signals on the SCLOCK and SDATA pins. Therefore, in a software master, the user must generate both SCLOCK and SDATA signals in software using these bit addressable locations.
For example, the bit that controls the SCLOCK signal (pin 26) is defined as a bit addressable location (MCO- bit address) in the I2CCON register.

An example of code used to generate a continuous HIGH-LOW-HIGH pulses on the SCLOCK pin is shown below :

```
AGAIN :    SETB MCO
           CLR  MCO
           JMP  AGAIN
```

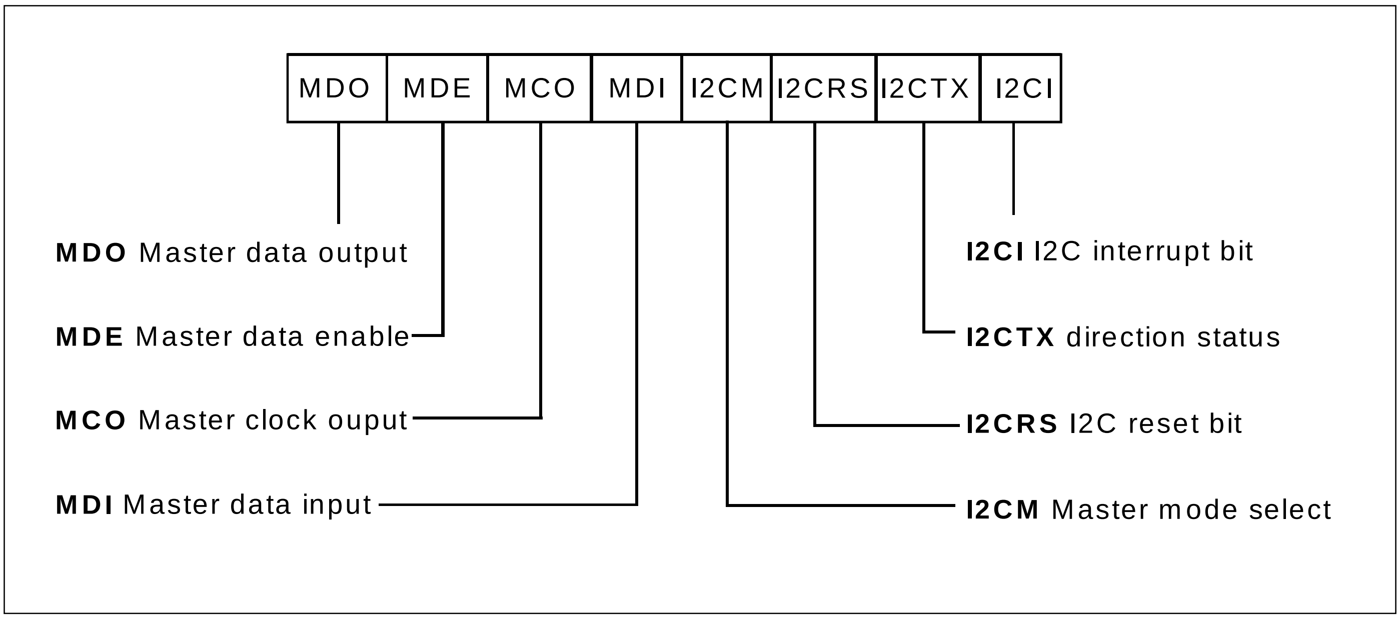
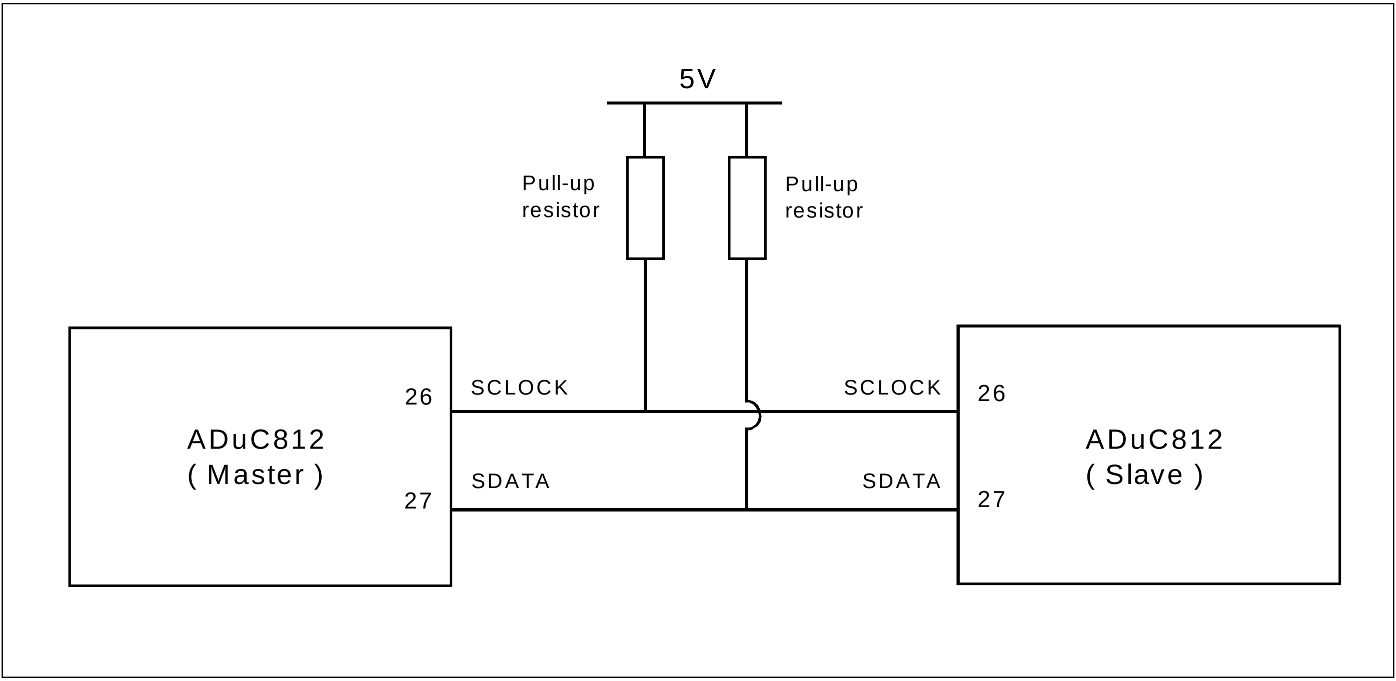


Figure 7. I2CCON bit designations

I²C COMMUNICATION (Master-transmitter to Slave-receiver)



In this mode, the master both transmits the slave address and on receipt of a valid acknowledge also transmits three bytes (the transmission/reception of three bytes is used for the code examples documented in this technical only) before terminating the communication by generating a stop bit.

As you have two ADuC812s, you will need two different programs, one for the master, one for the slave. On the next two pages, you will find the flowcharts corresponding to both master and slave programs in a Master-transmitter Slave-receiver mode of operation.

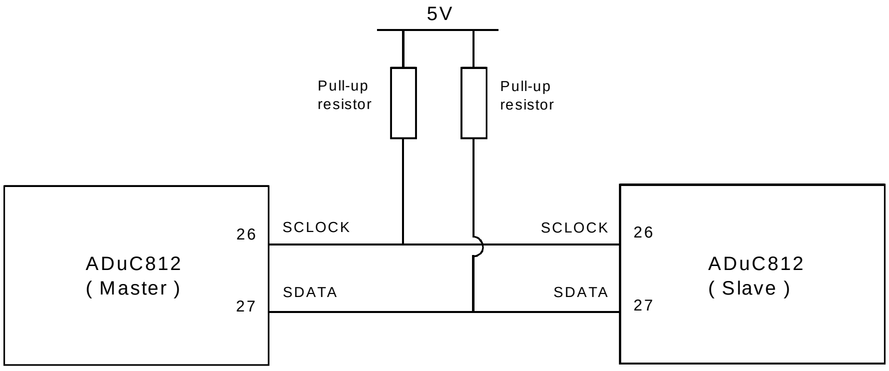
The master uses this sequence of operations :

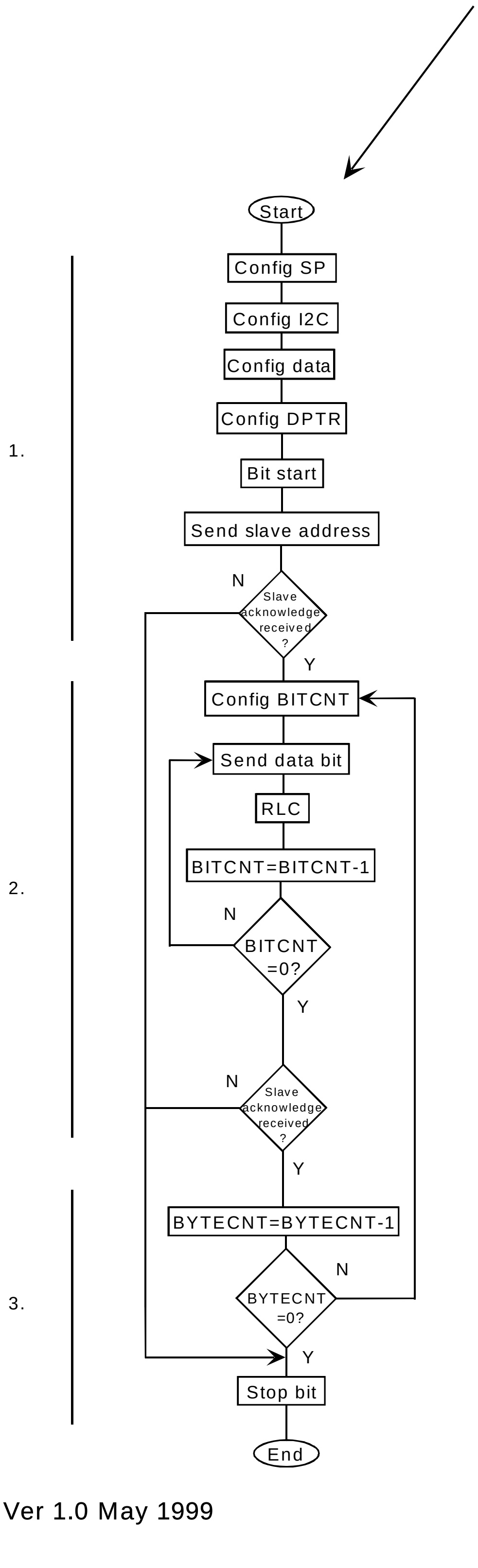
-Send a Start bit.
-Send the slave address.
-Wait for a valid acknowledge bit.
-Send the first data.
-Wait for a valid acknowledge bit.
-Send the second data.
-Wait for a valid acknowledge bit.
-Send the third data.
-Wait for a valid acknowledge bit
-Send a Stop bit.

The slave uses this sequence of operations :

-Receive address.
-Send the acknowledge.
-Receive the first data.
-Send the acknowledge.
-Receive the second data.
-Send the acknowledge.
-Receive the third data.
-Send the acknowledge.

FLOWCHARTS





Master software :

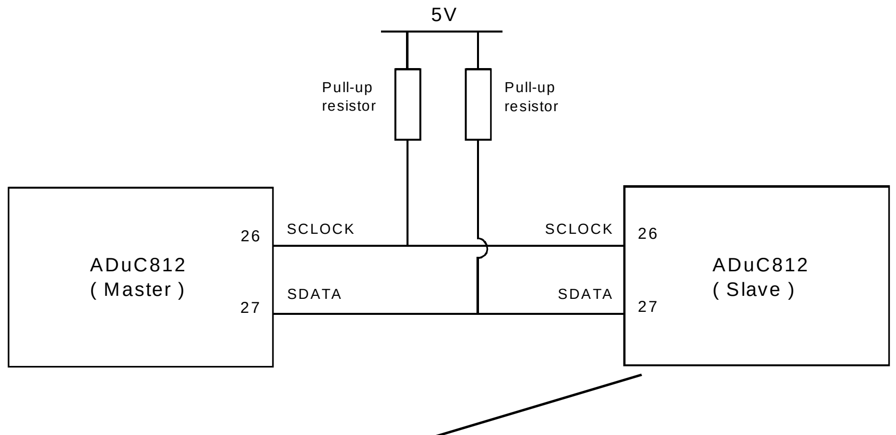
The flowchart on the left side describes all the operations that occur in this mode. The master, which is the transmitter, transmits data to the slave-receiver. In this mode the transfer direction is not changed (i.e. the master transmits the address and then follows by transmitting the data).

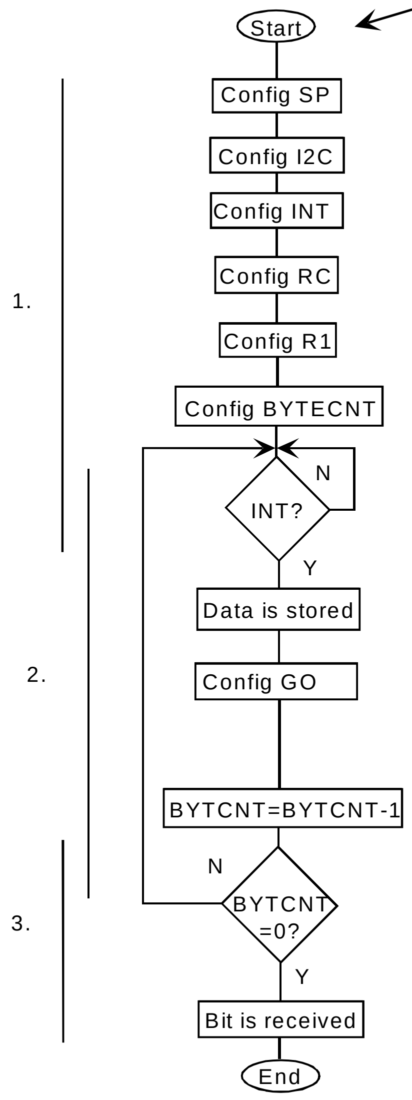
1. In the software, once the I^{2}C SFRs have been configured, the master sends a START bit and the slave address on the SDATA line. In this mode the R/$\overline{W}$ status bit is reset. If the master doesn't receive an acknowledge from the slave, it sends a STOP bit, an error bit is set and the transfer is finished.

2. If the slave acknowledges, the master sends data that has been previously written in the external data memory on the master evaluation board. After each byte transmitted, the slave has to acknowledge. If it acknowledges, the next data is sent by the master. If at any moment the slave fails to acknowledge, the master sends a STOP bit, an error bit is set and the transfer is finished.

3. When BYTECNT is equal to '0' which means that the last byte of the transfer has been sent (there are 3 bytes in this example), the master sends a STOP bit indicating that the transfer is complete.

FLOWCHARTS





Slave software :

The flowchart on the left side describes all the operations that occured in this mode. The master, which is the transmitter, transmits data to the slave-receiver. In this mode the transfer direction is not changed.

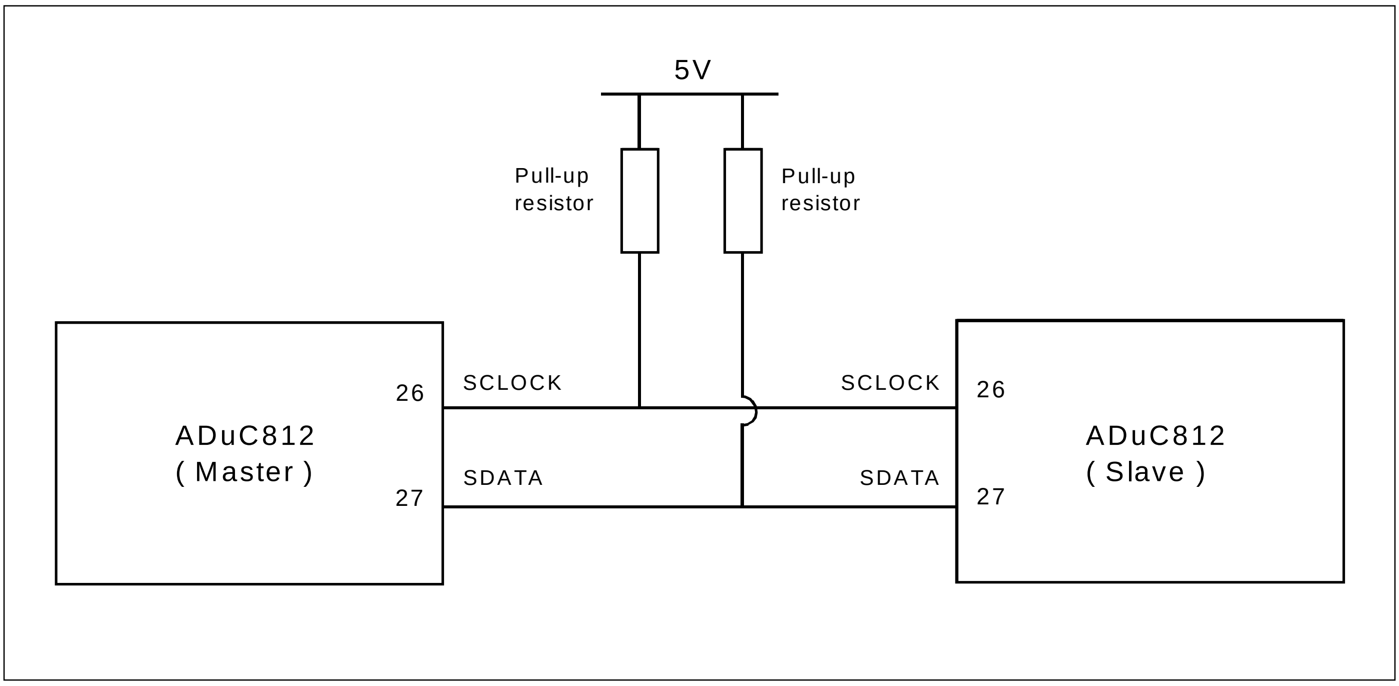
1. In software, once the I^{2}C SFRs have been configured, and after the START bit, which is sent by the master, the slave waits for the first data byte (the arrival of the data will generate an interrupt). Once it is received, the slave compares the data to its own address. If they match, the slave sends an acknowledge on the SDATA line and as the R/$\overline{W}$ status bit is set waits for the data (the arrival of the data will generate an interrupt).

2. If the master sends a data byte, the slave stores it in its internal memory, acknowledges and waits for the next data byte (the arrival of subsequent data bytes will again generate an interrupt).

3. When the slave receives the last byte of the transfer (there are three bytes in this example), BYTECNT is equal to '0'. At this time, the slave waits for the STOP bit. Once it is received, the I2C channel will immediately terminate communication.

Note :
When the interrupt occurs in software, the interrupt bit (I2CI) is automatically set but the user must clear it in the interrupt service routine (see line CLR I2CI in the code). If it is not cleared the slave will hold the clock line low thus disabling further I2C transmissions.

I²C COMMUNICATION (Slave-transmitter to Master-receiver)



In this mode, the master transmits the slave address and on receipt of a valid acknowledge waits for three data bytes to be transmitted by the slave, before terminating the communication by generating a stop bit.

As you have two ADuC812s you again need two different programs, one for the master, one for the slave. On the next two pages, you will find the flowcharts corresponding to both master and slave programs in a Slave-transmitter to Master-receiver mode of operation.

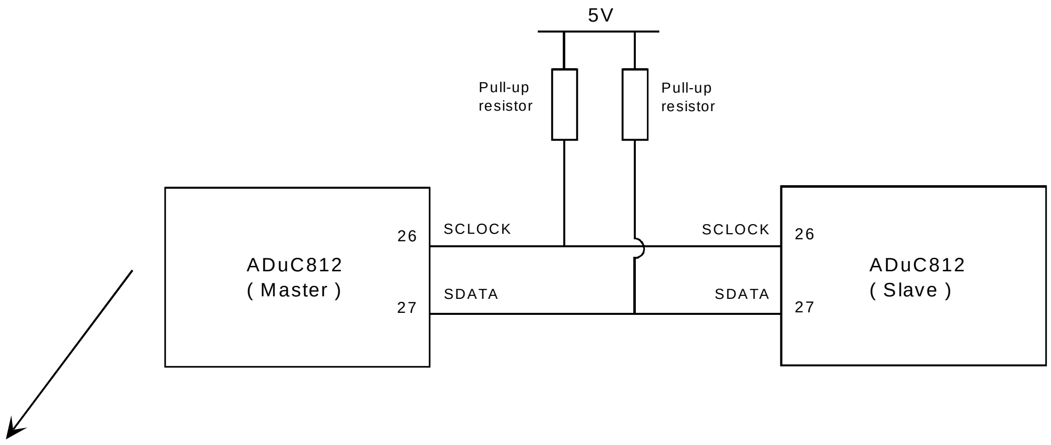
The master uses this sequence of operations :

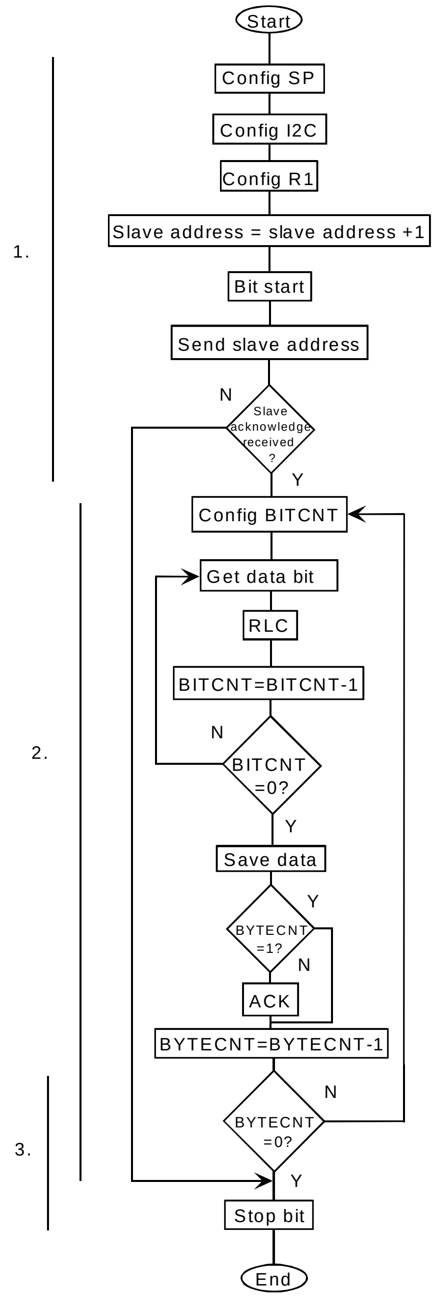
-Send a Start bit.
-Send the slave address.
-Wait for a valid acknowledge bit.
-Receive the first data.
-Send acknowledge.
-Receive the second data.
-Send acknowledge.
-Receive the third data.
-Send acknowledge.
-Send a Stop bit.

The slave uses this sequence of operations :

-Receive address.
-Send the acknowledge.
-Send the first data.
-Expect acknowledge.
-Send the second data.
-Expect acknowledge.
-Send the third data.
-Expect acknowledge.

FLOWCHARTS





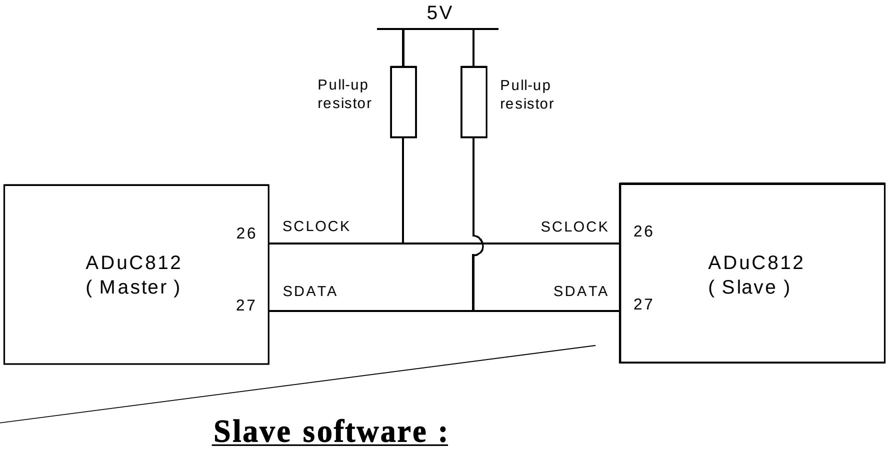
Master software :
The flowchart on the left side describes all the operations that occur in this mode. The master reads slave immediately after the first byte. In this mode, the transfer direction is changed i.e the master first sends the slave address and then receives three subsequent data bytes.

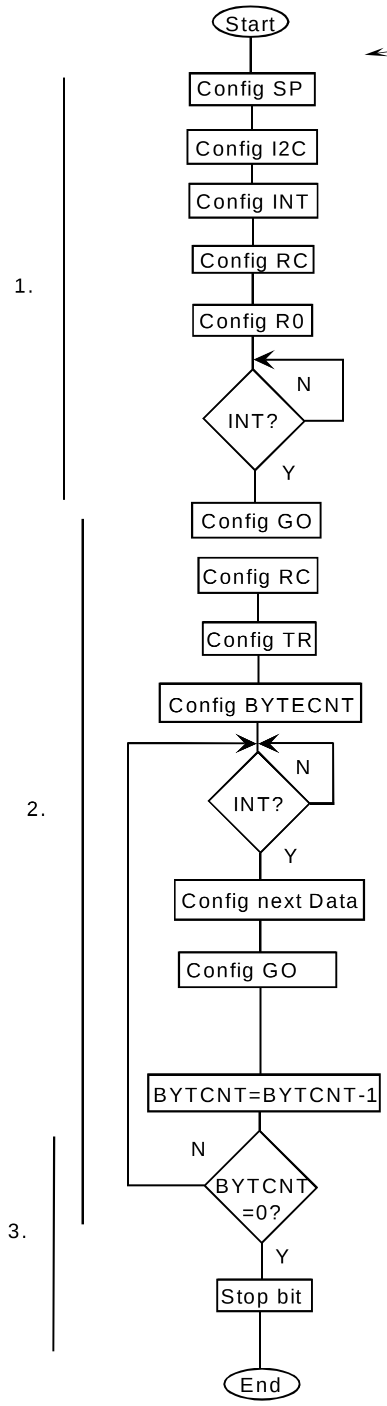
1. Once the I^{2}C SFRs have been configured in software, the master sends a START bit. Then the master sends the slave address on the SDATA line. In this mode the R/$\overline{\text{W}}$ status bit is set. If the master doesn't receive an acknowledge from the slave, it sends a STOP bit, an error bit is set and the transfer is terminated.

2. If the slave acknowledges, the master waits for the first data byte. Once it is received, the master stores it in internal memory, sends an acknowledge to the slave and waits for the next data byte.

3. When BYTECNT is equal to '0' which means that the last byte of the transfer has been sent (there are 3 bytes in this example), the master terminates the transmission by sending a STOP bit.

FLOWCHARTS





Slave software :
The flowchart on the left side describes the operations that occur in this mode. The master reads the slave immediately after the first byte. In this mode the transfer direction is changed i.e the master first sends the slave address and then receives three subsequent data.

1. In the software, once the I^{2}C SFRs have been configured and after the START bit, which is sent by the master, the slave waits for the first data byte(the arrival of the data will generate an interrupt). Once it is received, the slave compares the data to its own address. If they match, the slave sends an acknowledge bit on the SDATA line.

2. As the R/$\overline{\text{W}}$ status bit is set, the slave sends data that has been written previously in its internal memory.
After it has sent the data, the slave waits for an acknowledge from the master. After each valid acknowledge from the master, the slave sends another data byte and waits for the acknowledge.

3. When BYTECNT is equal to '0' which means that the last byte of the transfer has been sent (there are 3 bytes in this example), the master doesn't send an acknowledge but a STOP bit, and the transfer is terminated.

Note:
When an interrupt occurs, the interrupt bit (I2CI) is automatically set but the user must clear it in the interrupt routine (see line CLR I2CI in the code). If it's not cleared the slave will hold the clock line low thus disabling further I2C transmissions.

CONCLUSIONS

The ADuC812 incorporates a software master, hardware slave I^{2}C peripheral on-chip. While master mode I^{2}C operation will require additional software overhead as described in this Application Note, software master mode uses the same SFR and identical I^{2}C external pins to implement the I^{2}C protocol and is capable of sustaining bit-rates in excess of 100Kbs.

Even though a simple master/slave is described here, this can be easily extended to support multi-slave configurations as shown in figure 8 below.

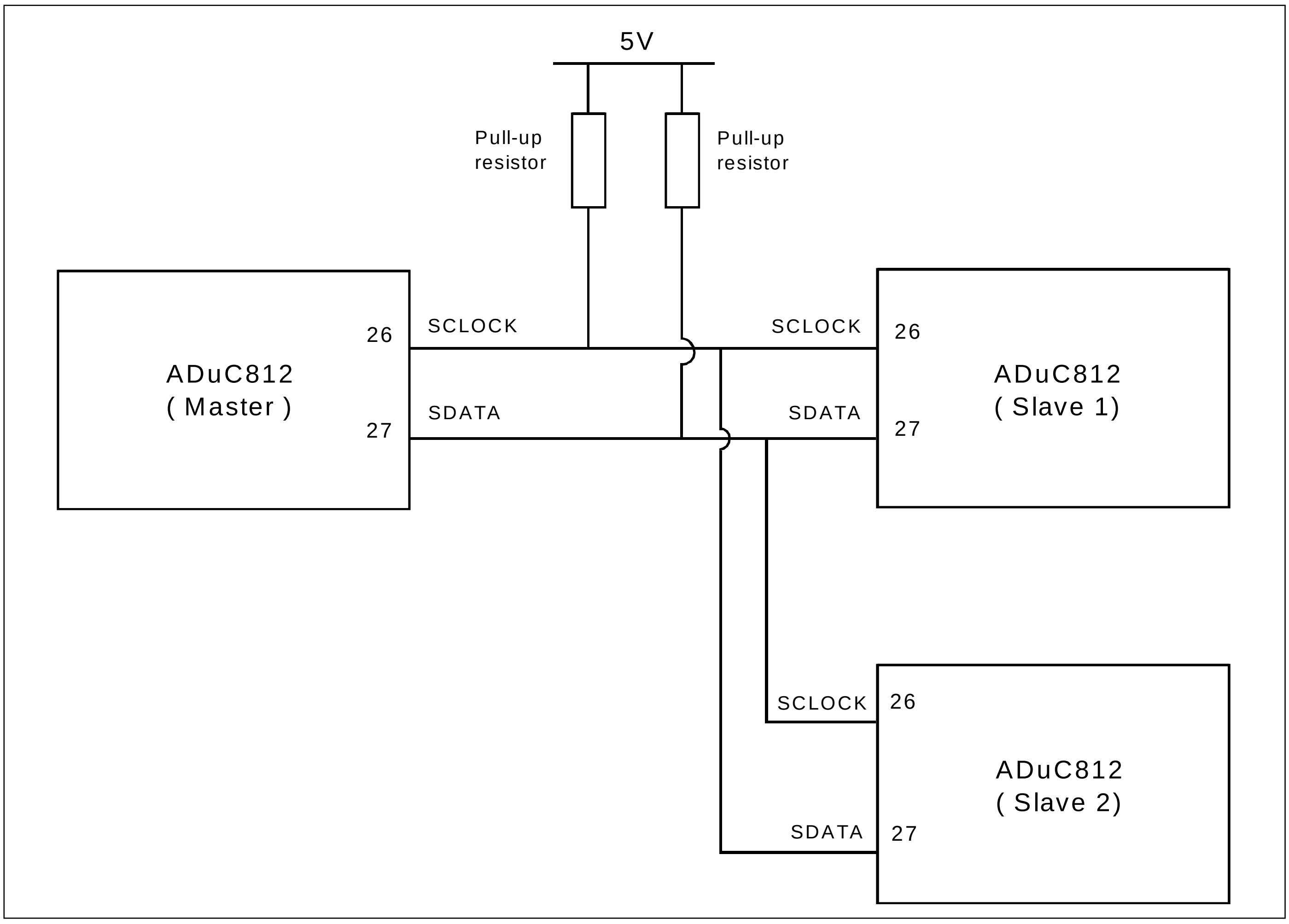


Figure 8. 1-Master, 2-Slaves

Programs

The software described in this Application Note can be found at Analog Devices external web site :

http://www.analog.com/microconverter

FREQUENTLY ASKED QUESTIONS

Question :

When I run the code, both SDATA and SCLOCK lines stay low?

Answer :

Make sure that the external pull-up resistors (3.9k) is present and connected to +5V on each line.

Question :

The master sends an address but there is no response from the slave?

Answer :

Be sure that the slave's I2CADD register is configured correctly .

Question :

The slave holds the value 44H in its I2CADD register, the master sends 44H but the slave doesn't acknowledge?

Answer :

Because of the 7-bit address, the master has to send 88H or 89H (depends of the $R/\overline{W}$ status bit). For more details see page 4 of this application note.

Question :

The master sends the slave address and receives an acknowledge from the slave ; however the slave holds subsequently the SCLOCK line low?

Answer :

Each time the slave receives or sends a data, the interrupt bit (I2CI) is set and the slave runs an interrupt routine. In this interrupt routine the interrupt bit (I2CI) must be cleared otherwise the SCLOCK line will stay low.

Question :

The master sends the slave address, but it seems that the slave never reaches the interrupt?

Answer :

Make sure that both interrupt registers (IE & IE2) are correctly configured.

IE = 80H enable all interrupts.
IE2 = 01H enable I2C interrupt.

APPENDIX A : MASTER.ASM

```
;========================================================================
;
; Author        : ADI - Apps
;
; Date          : 7/24/98
;
; File          : master.asm
;
; Description   : Code for a master in an I2C system
;
;========================================================================
;
$MOD812                           ;Use 8052 predefined Symbols

;                    Definitions

BITCNT               DATA 8h            ; bit counter for I2C routines
BYTECNT              DATA 030h          ; byte counter for I2C routines
SLAVEADD             DATA 032h          ; slave address for I2C routines

FLAGS                DATA 28h
NOACK                BIT    FLAGS.0             ; I2C no acknowledge flag
BUSY                 BIT    FLAGS.1             ; I2C busy flag
ERROR                BIT    FLAGS.2             ; I2C error flag
MISTAKE              BIT    P3.4

       ORG 00H
;
              JMP START
;===========================================================================

              ORG 07BH           ; Subroutines

;-----------------------------------------------------------------------
; DELAY:Create a delay for the main signals ( SCLOCK , SDATA )
;-----------------------------------------------------------------------

DELAY:
              RET
```

```
;------------------------------------------------------------------------
; SENDSTOP:Send the bit stop of the transmission
;------------------------------------------------------------------------

SENDSTOP: SETB      MDE         ; to enable SDATA pin as an output
          CLR       MDO         ; get SDATA ready for stop
          SETB      MCO         ; set clock for stop
          ACALL     DELAY
          SETB      MDO         ; this is the stop bit
          CLR       BUSY        ; bus should be released
          RET

;------------------------------------------------------------------------
; SENDBYTE:Send a 8-bits word to the slave
;------------------------------------------------------------------------

SENDBYTE:



          MOV       BITCNT,#8   ; 8 bits in a byte

          SETB      MDE         ; to enable SDATA pin as an output
          CLR       MDO
          CLR       MCO
     LOOP:RLC       A           ; send one bit
          MOV       MDO,C       ; put data bit on pin
          SETB      MCO         ; send clock
          CLR       MCO         ; clock is off
          DJNZ      BITCNT,LOOP

          CLR       MDE         ; release data line for acknowledge
          SETB      MCO         ; send clock for acknowledge
          JNB       MDI,NEXT    ; this is a check

          SETB      NOACK       ; no acknowledge

     NEXT:CLR       MCO         ; clock for acknowledge

          RET
```

```
;-----------------------------------------------------------------------
; BITSTART:Send the bit start of the transmission and the slave address to the slave
;-----------------------------------------------------------------------

BITSTART:   SETB        BUSY                ; I2C is in progress
            SETB        MDE                 ; to enable SDATA pin as an output

            CLR         NOACK

            CLR         ERROR
            JNB         MCO,FAULT
            JNB         MDO,FAULT
            CLR         MDO                 ; this is
            ACALL       DELAY               ; the
            CLR         MCO                 ; start bit
      FAULT:
            CLR         MISTAKE             ; set error flag
            MOV         A,SLAVEADD          ; Get slave address
            ACALL       SENDBYTE            ; call the routine to send the slave address byte
            RET

;---------------------------------------------------------------------
; SENDATA:Send all the sequence to the slave ( slave address + data )
;---------------------------------------------------------------------

SENDATA:    ACALL       BITSTART
            JB          MDI,NEXT1
            MOV         A,#00
      SLOOP:
            MOVX        A,@DPTR
            ACALL       SENDBYTE
            INC         DPTR
            JB          NOACK,NEXT1
            DJNZ        BYTECNT,SLOOP

      NEXT1:
            ACALL       SENDSTOP
            MOV         A,FLAGS
            ANL         A,#07h
            JZ          RETOUR
            SETB        P3.4
            CLR         RST
      RETOUR:
            RET
```

```
;---------------------------------------------------------------------
; RCVBYTE:receives one byte of data from an I2C slave device.
; --------------------------------------------------------------------

RCVBYTE:  MOV     BITCNT,#8       ;Set bit count.

          CLR     MDE             ;Data pin of the master is now an input
          CLR     MCO
LOOP2:    SETB    MCO
          CLR     MCO
          MOV     C,MDI           ;Get data bit from pin.
          RLC     A               ;Rotate bit into result byte.


          DJNZ    BITCNT,LOOP2    ;Repeat until all bits received.

                                  ;result byte is in the accumulator

          PUSH    ACC             ;Save result byte in the stack

          SETB    MDE             ;Data pin of the master must be an output for
                                  ;the acknowledge
          MOV     A,BYTECNT
          CJNE    A,#1,SACK       ;Check for last byte of frame.
          SETB    MDO             ;Send no acknowledge on last byte.
          SJMP    NACK

     SACK:
          CLR     MDO             ;Send acknowledge bit.

     NACK:
          SETB    MCO             ;Send acknowledge clock.
          POP     ACC             ;Restore accumulator
          ACALL   DELAY
          CLR     MCO
          SETB    MDO             ;Clear acknowledge bit.
          ACALL   DELAY
          CLR     MDE

          RET
```

```
;-------------------------------------------------------------------------
;RCVDATA:receives one or more bytes of data from an I2C slave device.
;-------------------------------------------------------------------------

RCVDATA:
            INC         SLAVEADD          ;Set for READ of slave.
            ACALL       BITSTART          ;Acquire bus and send slave address.
            JB          NoAck,RDEX        ;Check for slave not responding.

RDLoop:     ACALL       RCVBYTE           ;Receive next data byte.
            MOV         @R1,A             ;Save data byte in buffer.
            INC         R1                ;Advance buffer pointer.
            DJNZ        BYTECNT,RDLoop    ;Repeat untill all bytes received.

RDEX:       ACALL       SENDSTOP          ;Done, send an I2C stop.
            RET

;---------------------------------------------------------------------
; Main program
;---------------------------------------------------------------------

START:                                 ; Main program
            MOV         SP,#040h
            CLR         NOACK
            MOV         SLAVEADD,#088H
            MOV         BYTECNT,#3
            MOV         I2CCON,#0A8h

            ; code for a write mode ( master-transmitter transmits to slave-receiver )

      ;     MOV         DPTR,#080H        ; master transmits to slave
      ;     MOV         A,#055H           ; datas which are located in
      ;     MOVX        @DPTR,A           ; the external memory
      ;     MOV         DPTR,#081H
      ;     MOV         A,#044H
      ;     MOVX        @DPTR,A
      ;     MOV         DPTR,#082H
      ;     MOV         A,#033H
      ;     MOVX        @DPTR,A

;           MOV         DPTR,#080h
;           ACALL       SENDATA

            ; code for a read mode ( master reads immediately after first byte )

            MOV         R1,#035h
            ACALL       RCVDATA
END
```

APPENDIX B : SLAVE.ASM

```
;====================================================================
;
; Author        : ADI - Apps
;
; Date         : 7/24/98
;
; File        : slave.asm
;
; Description   : Code for a slave in an I2C system
;
;====================================================================
;
$MOD812                         ;Use 8052 predefined Symbols

;                       Definitions

BYTECNT                 DATA 030h           ; byte counter for I2C routines

FLAGS                   DATA 28h
GO              BIT     FLAGS.0             ; flag for all the interrupts
RC              BIT     FLAGS.1             ; flag for Write mode interrupt
TR              BIT     FLAGS.2             ; flag for Read mode interrupt


        ORG 00H
;
                JMP     START               ; jump to label 'start'

;=======================================================================

                ORG     03Bh                ; I2C slave interrupt

                JB      RC,RECEIVE          ; depending on flags there
                JB      TR,TRANSMIT         ; are two different interrupts

;=======================================================================
                ORG         07BH          ; Subroutines

;------------------------------------------------------------------------
;RECEIVE: receive interrupt routine
;------------------------------------------------------------------------

RECEIVE:
                SETB        GO
                MOV         @R1,I2CDAT; move data on internal RAM
                CLR         ISI           ; clear interrupt bit
                RETI

;------------------------------------------------------------------------
;TRANSMIT: transmit interrupt routine
;------------------------------------------------------------------------

TRANSMIT:
                SETB        GO
                MOV         I2CDAT,R0
                CLR         ISI           ; clear interrupt bit
                RETI

;------------------------------------------------------------------------
;RCVBYTE2:receive byte routine for read mode
;------------------------------------------------------------------------

RCVBYTE2:

                NOP
                RET

;------------------------------------------------------------------------
;RCVBYTE:receive byte routine
;------------------------------------------------------------------------

RCVBYTE:

        WAIT1:
                JNB         GO,WAIT1      ; wait for the interrupt
                INC         R1            ; next storage will be on 41h then 42h ...
                CLR         GO            ; flag is cleared for the next interrupt
                RET
```

```
;------------------------------------------------------------------------
;RCVDATA:receive bytes routine
;------------------------------------------------------------------------

RCVDATA:

          MOV       BYTECNT,#4        ; 4 bytes : address + 3 datas
LOOP2:    ACALL     RCVBYTE
          DJNZ      BYTECNT,LOOP2
          RET

;------------------------------------------------------------------------
;SENDBYTE:byte transmit routine
;------------------------------------------------------------------------

SENDBYTE:

     WAIT2:JNB      GO,WAIT2          ; wait for the interrupt
          INC       R0                ; second data is 34h and third data is 35h
          CLR       GO
          RET

;------------------------------------------------------------------------
;SENDATA:bytes transmit routine
;------------------------------------------------------------------------

SENDATA:
          MOV       BYTECNT,#3        ; 3 data will be send by the slave
LOOP:     ACALL     SENDBYTE
          DJNZ      BYTECNT,LOOP
          RET
```

```
;-----------------------------------------------------------------------
;Main program
;-----------------------------------------------------------------------
START:

                CLR       GO                  ; clear flag used in the interrupt
                MOV       I2CADD,#044h        ; slave address
                MOV       SP,#020h
                MOV       IE,#80h             ; enable all the interrupts
                MOV       IE2,#01h            ; enable I2C interrupt
                MOV       I2CCON,#000h        ; slave mode


                ; code for a write mode ( master-transmitter transmits to slave-receiver )

;               SETB      RC                  ; specific flag for the interrupt routine
;               MOV       R1,#040h            ; first data will be store in internal RAM at 40h
;               ACALL     RCVDATA             ; slave receives his address + 3 datas

                ; code for a read mode ( master reads slave immediately after first byte )


                SETB      RC                  ; specific flag for the interrupt routine
                MOV       R0,#033h            ; first data send is 33h
                ACALL     RCVBYTE2            ; slave receives the address send by the master
                CLR       RC
                SETB      TR
                ACALL     SENDATA             ; slave sends 3 datas

                CLR    P3.4                   ; led is off, everything is OK

END
```