MicroConverter® Technical Note uC001
MicroConverter® I²C® Compatible Interface

1.0 INTRODUCTION

This Technical Note describes the hardware slave and software master implementation of an I²C (inter integrated circuit) compatible interface using the MicroConverter. This technical Note also contains example code showing how a master and a slave can communicate with each other using the MicroConverter I²C compatible interface.

The main features of the I²C bus are:

- Only two bus lines are required; a serial data line (SDATA) and a serial clock line (SCLOCK). Both of these lines are bi-directional meaning that both the master and the slave can operate as transmitters or as receivers.
- An I²C master can communicate with multiple slave devices. Because each slave device has a unique 7-bit address then single master/slave relationships can exist at all times even in a multi slave environment.
- Software master can output data at approx 140 kbit/s using a 12MHz clock. Hardware slave can receive clocks (and data) at above 400 kbit/s.
- On-Chip filtering rejects <50ns spikes on the SDATA and the SCLOCK lines to preserve data integrity.

A typical block diagram of an I²C interface is shown in figure 1 below.

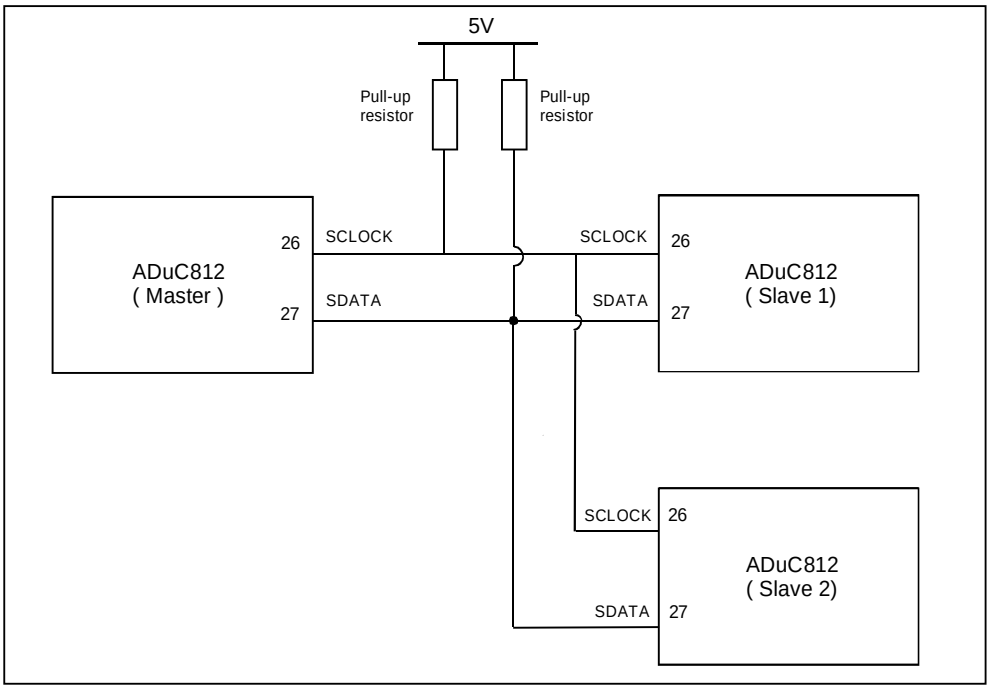

Figure 1. Single Master Multi Slave I²C Block Diagram

1.1 I²C INTERFACE OVERVIEW

I²C is a two-wire serial communication system developed by Philips, which allows multiple masters and multiple slaves to be connected via two wires (SCLOCK and SDATA). In an I²C interface there must be at least a single master and a single slave.

The SCLOCK signal controls the data transfer between master and slave. The SCLOCK signal is always transmitted from the Master to the Slave. (The slave however does have the ability to pull this line low if it is not ready for the next transmission to begin. This is called 'Clock Stretching' (see section 1.7)). One clock pulse must be generated for each data bit transferred.

The SDATA signal is used to transmit or receive data. The SDATA input must be stable during the HIGH period of SCLOCK. A transition of the SDATA line while SCLOCK is high will be seen as a START or STOP condition (fig 2a & 2b).

Note: For the ADuC812 pull-up resistors are needed on both the SCLOCK and the SDATA lines. However for the ADuC814/ADuC816/ADuC824/ADuC83x weak pull-ups are implemented in hardware.

1.2 I²C COMPLETE DATA TRANSFER SEQUENCE

START CONDITION

A typical data transfer sequence for an I²C interface starts with the START condition. The START condition is simply a HIGH to LOW transition in the SDATA line while the SCLOCK line is pulled HIGH (figure 2a). The master is always responsible for generating the START condition.

Note: The START (and STOP) conditions are the only time that the SDATA line should change during a high period of the SCLOCK line. During normal data transfer (including Slave addressing) the data on the SDATA line must be stable during the HIGH period of the SCLOCK line.

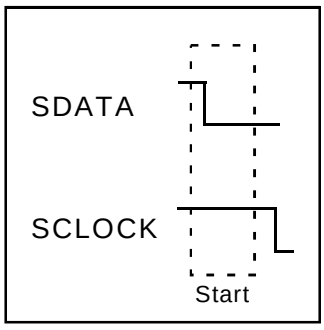

Figure 2a: Start Condition for I²C

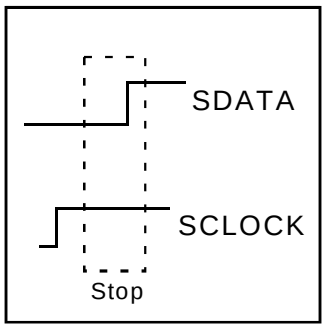

Figure 2b: Stop condition for I²C

SLAVE ADDRESS

After the start condition the master sends a byte (MSB first) on the SDATA line (along with eight SCLOCK pulses). The first seven bits of this byte is the 7-bit Slave Address. The slave will only respond to the master if this 7-bit address matches the unique address of the slave device (see section 1.3). The eighth bit (LSB) is the R/$\overline{W}$ status bit. The R/$\overline{W}$ status bit determines the direction of the message. If this bit is clear then the master will write data to a selected slave. If this bit is set then the master expects to receive data from the slave.

If the slave receives the correct address then the slave returns a valid ACK (more below), pulls the SCLOCK line low and sets the I2CI interrupt bit (generating an interrupt if configured correctly). The SCLOCK line is pulled low so that the master knows that the slave is not ready to receive the next byte yet. Clearing the I2CI bit will release the SCLOCK line.

ACKNOWLEDGE (ACK) / NO ACKNOWLEDGE (NACK)

If the slave address matches the address sent by the master then the slave will automatically send an Acknowledge (ACK), otherwise it will send a No Acknowledge (NACK). An **ACK** is seen as a LOW level on the SDATA line on the 9th clock pulse. An **NACK** is seen as a HIGH level on the SDATA line on the 9th clock pulse (see figure 3).

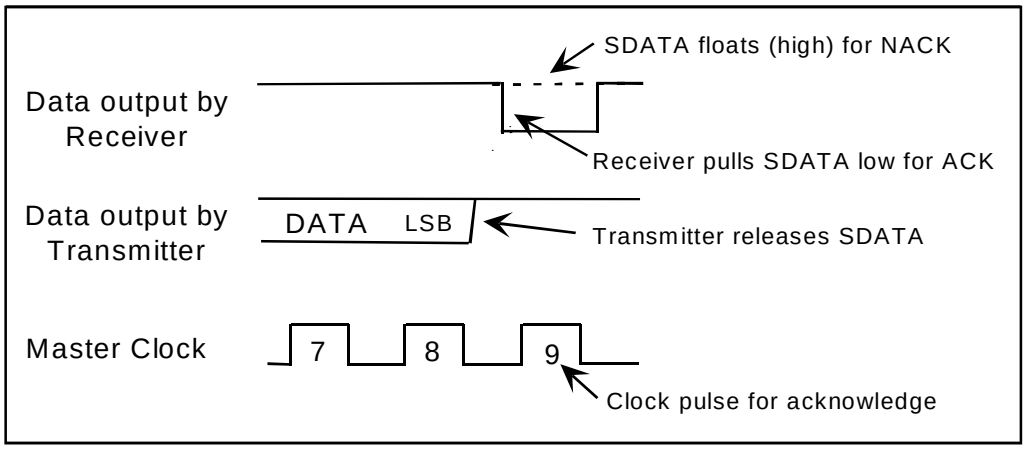

Figure 3. Acknowledge (ACK) and a no Acknowledge (NACK) on the I²C bus

During data transfer the ACK or the NACK is always generated by the **receiver**. However the clock pulse required for the ACK is always generated by the Master. The transmitter must release the SDATA line (high) during the ACK clock pulse. For an ACK the receiver must pull the SDATA line low for a valid ACK.

If the MicroConverter is configured in slave mode then, both the ACK and the NACK are automatically generated in hardware, at the end of each byte in the reception. If the MicroConverter is configured in Master mode then, the users software must implement the ACK at the end of each byte in the reception.

If a master receives a NACK from a slave-receiver (either the slave did not respond to the slave address or the data transmitted) then the master should generate the STOP condition (more below) to abort the transfer.

A master-receiver must signal the end of a data sequence to the slave-transmitter by generating an no acknowledge (NACK) after the last byte that was sent by the slave. Once the slave receives the NACK it releases the SDATA line to allow the master to generate the STOP condition.

DATA TRANSFER

In the I²C ISR (or in a polled implementation) the slave will decide whether or not to transmit or receive depending on status of the R/$\overline{W}$ bit sent by the master. The slave will then either transmit or receive a bit on each clock sent by the master. It is up to the master to provide the 9 clocks (8 for the data and 1 for the ACK) for the slave to transmit/receive data to/from the master. The I2CI bit will be set every time a valid data byte has been transmitted/received by the slave.

Note again that in a slave-transmitter, master-receiver system the master must signal the end of a data sequence to the slave by sending a NACK after the last byte transmitted by the slave. Once the slave receives the NACK it releases the SDATA line to allow the master to generate the STOP condition.

STOP CONDITION
The data transfer sequence is terminated by the STOP condition. A STOP condition is defined by a LOW to HIGH transition on the SDATA line while SCLOCK is HIGH (figure 2b).

The STOP condition is always generated by the master. The master will send the STOP condition once the master is satisfied that the data sequence is over or if it receives a NACK from the slave device. The reception of the STOP condition resets the slave device into waiting for the slave address again.

A typical transfer sequence is shown in figure 4.

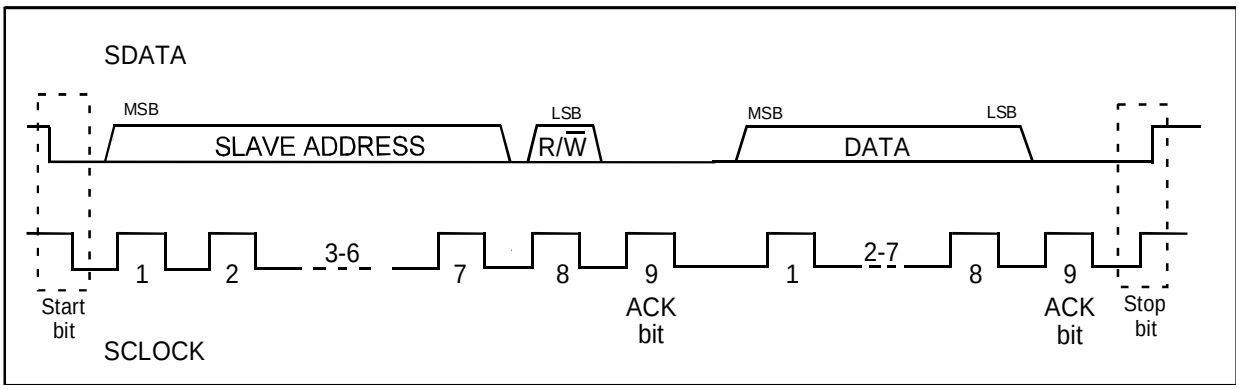

Figure 4. Typical I²C transfer sequence

1.3 SLAVE 7-BIT ADDRESSING

The master sends the 7-bit slave address as the 7 MSBs of the first transmitted data byte after a START condition. (The eighth bit is the R/$\overline{W}$ status bit). The slave compares only the 7 upper bits to its own address. To make a complete byte, the slave adds a zero to the MSB position. The result of this completion is compared to the slave address register I2CADD.

While the slave does all the manipulation of the I²C slave addressing as described above automatically in hardware it is up to the master to output the slave address appropriately.

e.g. To select a slave device with slave address 44h (I2CADD for slave device = 44h) the master must transmit the byte 88h (master-transmitter, slave-receiver) or 89h (master-receiver, slave-transmitter) after the START condition.

This is because the 7-bit address is mapped as the seven MSBs for the master. The LSB is either high or low depending on whether or not the master expects to transmit or receive data after communication with the slave device has been established. Once the slave manipulates this data the 7 MSBs of the master appear as the 7 LSBs for the slave address with a zero put in the MSB position. This is shown in figure 5 below.

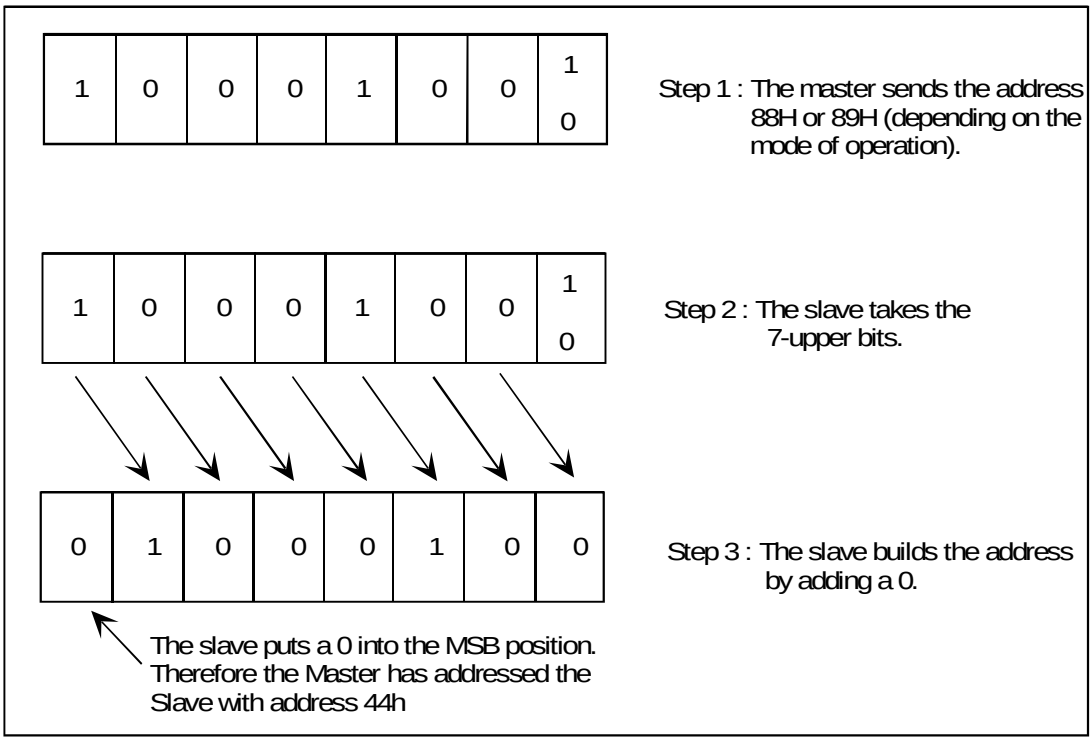

Figure 5: 7-bit Slave Address Procedure

1.4 I²C IMPLEMENTATION ON THE MICROCONVERTER

This section describes the I²C implementation on the MicroConverter. The MicroConverter provides both hardware slave and software master operating modes. Three SFRs are used to control the I²C interface :

I2CADD : Holds the 7-bit address of the MicroConverter device (default value = 55H).
Note: This SFR is only used in slave mode. See section 1.3 as regards addressing the slave.

I2CDAT : In slave-receiver mode the received data from the SDATA line is latched into this SFR. Hence after a successful reception the received data can be read from this SFR. E.g.

```
MOV   A, I2CDAT
```

reads the received data into the accumulator.

In slave transmitter mode a write to this SFR will make the data available for transmission on the SDATA line under control of the master. E.g.

```
MOV I2CDAT, #60h
```

writes 60h out the SDATA line when clocked by the master.

Note: For the ADuC814/ADuC816/ADuC824/ADuC83x a write or a read of the I2CDAT SFR automatically clears the I2CI interrupt flag. Clearing this flag for a second time will cause the I²C controller to get 'lost'. For the ADuC812 the I2CI interrupt flag must be cleared in software.

I2CCON : Holds configuration/ control bits for master/slave mode of operation as defined in table 1.

Bit Mneumonic	Description
MDO	**Software Master Data Out Bit (MASTER ONLY)** This bit is used to implement a master I²C interface transmitter in software. Data written to this bit will be outputted on the SDATA pin if the data output enable (MDE) is set.
MDE	**Software Master Data Out Enable Bit (MASTER ONLY)** This bit is used to implement a master I²C interface in software. Setting this bit enables the SDATA pin as an output (TX). Clearing the bit enables SDATA as an input (RX).
MCO	**Software Master Clock Out Bit (MASTER ONLY)** This clock out bit is used to implement a master I²C interface in software. Data written to MCO will be outputted on the SCLOCK pin.
MDI	**Software Master Data In Bit (MASTER ONLY)** This bit is used to implement a master I²C receiver interface in software. The data on the SDATA pin is latched in here on SCLOCK if data output enable (MDE) is clear.
I2CM	**I²C Mode Bit** Setting this bit enables software master mode, clearing this bit enables hadware slave mode.
I2CRS	**I²C Serial Port Reset (SLAVE ONLY)** Setting this bit will cause a reset of the I²C interface.
I2CTX	**I²C Transmission Direction Status (SLAVE ONLY)** This bit indicates the direction of transfer. The bit is set if the master is reading from the slave. The bit is cleared if the master is writing data to the slave. This bit is automatically loaded with the R/$\overline{W}$ bit after the slave address and start condition.
I2CI	**I²C Interrupt Flag (SLAVE ONLY)** This is the interrupt flag for the I²C serial port. This bit is set after a byte has been transmitted or received. It must be cleared in software. See section 1.8 to see how this is done.

Table 1: Bit Definition of I2CCON

1.5 SOFTWARE MASTER MODE IMPLEMENTATION

The MicroConverter acts an I²C software master. The user must program the MicroConverter to 'bit bang' the SDATA and SCLOCK lines. Master mode is selected by setting the I2CM bit in the I2CCON register.

To transmit data on the SDATA line, the MDE bit must first be set to enable the output driver on the SDATA pin. The MDO bit in the I2CCON register is the Data Out bit. The output driver on the SDATA pin will either pull the SDATA line high or low depending on whether the MDO bit is set or cleared.

The MCO bit in the I2CCON register is the Clock output bit. The output driver on the SCLOCK pin is always enabled in master mode and will either pull the SCLOCK line high or low depending on the whether the bit MCO is set or cleared.

Note: On the ADuC812 there is no pull-up on the SDATA/SCLOCK output driver hence the user must implement external pull-ups to pull this line high. For the ADuC812S/ ADuC816/ ADuC824 weak pull-ups exist so the user does not have to pull these lines high.

To receive data, the MDE bit must be cleared to disable the output driver on SDATA. Software is used to toggle the MCO bit (send out a clock pulse) and read the status of the SDATA line via the MDI bit. The MDI bit is set if SDATA is high and cleared if SDATA is low (provided MDE is cleared).

Software must control the MDO, MCO and MDE bits appropriately to generate the START condition, slave address, acknowledge bits, data bytes and STOP conditions appropriately. The data is latched into MDI on a rising edge SCLOCK only if MDE is cleared. Use the functions in the master code example (I2Cmstr.asm) for the various functions to generate the various transmit and receive signals.

Note:

- It is not possible to read the status of the SDATA line (MDI bit) unless MDE is low (unless it is tied to another port pin).
- It is not possible to read back the status of the SCLOCK line (unless it is tied to another port pin).
- Since the Master is a software master on the MicroConverter I²C interrupts will not take place.

1.6 HARDWARE SLAVE MODE IMPLEMENTATION

The MicroConverter implements a hardware slave. The part defaults into I²C slave mode. I²C mode (as apposed to SPI mode) is activated by clearing the SPE bit in the SPICON SFR (SPE=0 by default). The I²C address is stored in the I2CADD register. Data received or to be transmitted is stored in the I2CDAT register.

Slave mode is selected by clearing the I2CM bit in the I2CCON register. The default state for the I²C slave is waiting for a START condition. The MicroConverter will set the I2CI bit when it has detected a valid start condition followed by a valid address followed by the R/$\overline{W}$ bit (i.e. I2CI is set 8 bits after the start condition if the address is correct).

The I²C peripheral will not generate a core interrupt unless the user has pre-configured the I²C interrupt enable bit in the IE2/IEIP2 SFR as well as the global interrupt bit EA in the IE SFR. i.e.

```
; Enabling I2C Interrupts for theADuC812
MOV   IE2,#01h          ; enable I2C interrupt
SETB  EA

; Enabling I2C Interrupts for the ADuC812S/ADuC816/ADuC824
MOV   IEIP2,#01h        ; enable I2C interrupt
SETB  EA
```

Note: On the ADuC812 it is very important that the I2CI bit be cleared only once per interrupt. If for any reason the user tries to clear I2CI more than once then the I²C controller will get 'lost'.

```
CLR     I2CI            ; clear interrupt bit for ADuC812
```

On the ADuC812S/ADuC816/ADuC824 : An auto-clear of the I2CI bit is implemented so this bit is cleared automatically on a read or write access to the I2CDAT SFR.

```
MOV     I2CDAT, A       ; I2CI cleared by transmission (812S/816/824)
MOV     A, I2CDAT       ; I2CI cleared by reception (812S/816/824)
```

Again if for any reason the user tries to clear the interrupt more than once i.e. access the data SFR more than once per interrupt then the I²C controller will get lost.

The user can choose to poll I2CI bit or enable the interrupt. In the case of the interrupt the PC counter will vector to 003BH at the end of each complete byte. For the first byte when the user gets to the I2CI ISR the 7-bit address and the R/$\overline{W}$ bit will be in the I2CDAT SFR i.e. the byte just received.

The I2CTX bit contains the R/$\overline{W}$ bit sent from the master. If I2CTX is set, the slave will transmit data by writing to the I2CDAT register. If I2CTX is cleared, the slave will receive a serial byte into the I2CDAT register. Software can interrogate the state of I2CTX to determine whether is should write to or read from I2CDAT.

The slave will hold SCLOCK low until the I2CI bit is cleared by software. This is "clock stretching" as described in section 1.7. Stretching the SCLOCK basically ensures that the master does not transmit the next data until the slave is ready.

The I2CI interrupt bit will be set every time a complete data byte is received or transmitted provided it is followed by a valid ACK. If the byte is followed by a NACK an interrupt is NOT generated. The MicroConverter will continue to issue interrupts for each complete data byte transferred until a STOP condition is received or the interface is reset.

Note: There is no way to distinguish between an interrupt generated by a received START + valid address and an interrupt generated by a received data byte. To differentiate between the two interrupts the user will have to keep track of the communication sequence. (perhaps set/clear a flag as appropriate)

When a STOP condition is received, the interface will reset to a state where it is waiting to be addressed (idle). Similarly, if the interface receives a NACK at the end of a sequence it also returns to the default idle state. The I2CRS bit can be used to reset the I²C interface. This bit can be used to force the interface back to the default idle state.

1.7 I²C FEATURES NOT IMPLEMENTED ON THE MICROCONVERTER

SLAVE MODE:

- The MicroConverter (slave) will not respond to the General Call Address (0000 000).

- The repeated START allows the master to address another slave (or change the direction of data transfer) without issuing the STOP condition. The data transfer continues as if the repeated START is a new START condition. The MicroConverter does accept a repeated START condition. However the only way for the user to distinguish the repeated start condition from data that has been transmitted is by monitoring the R/$\overline{W}$ bit. This bit will only change if the repeated START condition changes the direction of data transfer. Thus a user can only distinguish a repeated START if it changes the direction of data transfer.

  In general on the MicroConverter the STOP condition must be sent to the SLAVE to end the communication with a single slave. After the STOP condition has been sent a new START condition can be sent to initiate the new communication.

- The MicroConverter cannot support the 'Start Byte' in slave mode. This can be implemented in software in master mode however.

- The MicroConverter does not support 10-bit addressing.

MASTER MODE:

CLOCK STRETCHING

- Clock stretching is used when interfacing a fast master to a slow slave. When the slave receives a valid address or valid data byte it automatically pulls the SCLOCK line low. Once the I2CI bit has been cleared in software the slave releases the SCLOCK line. To implement clock stretching correctly the master must be able to read the status of the SCLOCK line and delay sending out new data until the slave has released SCLOCK.
  On the MicroConverter the master cannot read back the SCLOCK line via the I²C interface. To read back the SCLOCK line the SCLOCK line must be connected to a digital input (any port pin) and read back the state of the SCLOCK line to see if the slave has released this line or not.

ARBITRATION

- In multi-master applications it is necessary to be careful that two masters do not try and write data on the bus at the same time. For this reason Arbitration is used to monitor the SDATA line. Arbitration is said to take place on the SDATA line while the SCLOCK line is high, in such a way that the master which transmits a high level, while another master transmits a low will switch off its data output stage and wait until the bus is free.
  For arbitration, it is necessary to read SDATA while driving it in order to compare bits sent by master with the actual state of the SDATA line. On the MicroConverter the master cannot read back the SDATA line via the I²C interface. To read back the SDATA line the SDATA line must be connected to a digital input (any port pin) on the master.

1.8 DIFFERENCES BETWEEN MICROCONVERTER I²C IMPLEMENTATIONS

ADUC812

- No pull-ups on the SDATA/SCLOCK output drivers.
- I2CI must be cleared in software using the `CLR I2CI` instruction (only clear once)
- The I²C interrupt enable bit (ESI) and I²C interrupt priority bit (PSI) are in the IE2 SFR (SFR address A9h) and the IP SFR (SFR address B8h) respectively.

ADUC814/ADUC816/ADUC824/ADUC83X

- Weak pull-ups (approx 20μA) on the SDATA/SCLOCK output drivers. These pull-ups are only switched in when I²C mode is selected. For SPI mode these pull-ups are switched off. For a system with many slaves external pull-ups might be necessary.
- The I2CI interrupt bit is automatically cleared once a read or a write of the I2CDAT SFR is executed in software. Clearing this bit again will cause the I2C controller to 'get lost' hence it should be ensured that this bit is only ever cleared once per interrupt.
- The I²C interrupt enable bit (ESI) and I²C interrupt priority bit (PSI) are both in the IEIP2 SFR (SFR address A9h) at IEIP2.0 and IEIP2.4 respectively

2.0 I²C MASTER OPERATION CODE EXAMPLE

This note describes the master program I²Cmstr.asm. This program should be used in conjunction with the slave program I2Cslave.asm on two separate evaluation boards to show the I²C communication between the master and the slave on the MicroConverter. This example code (I2Cmstr.asm) demonstrates how the I²C interface on the MicroConverter can be programmed in master mode.

The master program sends the START condition, the slave address, and the R/$\overline{W}$ bit (set to initiate a master-reception) to the slave. It then receives a single byte from the slave and sends back a NACK and the STOP condition. The NACK indicates to the slave that the master has received the last byte to be transmitted by the slave. It then sends the START condition, slave address and R/$\overline{W}$ bit (cleared to indicate a transmission) to the slave again. This time the master transmits a byte to the slave. After the transmission of the byte it examines the ACK and sends the STOP condition. The master then transmits the received byte up the UART to the PC where it can be viewed using hyperterminal. The master program jumps back to the start and receives a character from the slave again.

If a character is received from the UART (by pressing the keyboard), then the master sends out the ASCII value associated with that character to the slave device. By pressing the INT0 button on the evaluation board the data being outputted by the master to the slave will increment.

Flow charts are shown for the overall master program (figure 6). The subroutines RCVDATA and SENDDATA are shown in figures 7a and 7b respectively.

PROGRAM VARIABLES

SLAVEADD: Slaveadd holds the address of the slave that you wish to address. In this case we want to address the slave with the address 44h, this means that the master must send 88h (or 89h) as explained in section 1.3.

OUTPUT: Output holds the value to be transmitted to the slave, this is initialized to 0h.

INPUT: Input is the value received from the slave. This value is sent out the UART.

NOACK: This is set if a NACK is received when an ACK is expected.

ERR: The ERR flag is set if the NOACK is set anywhere in the program and allows the NOACK flag to be cleared in the software. If the ERR flag is set at the end of the program an error message is sent out the UART.

DESCRIPTION OF CODE

The following description of the code I2Cmstr.asm should be read in conjunction with the main flowchart in figure 6 and also the two separate flowcharts for RCVDATA and SENDDATA in figure 7a and figure 7b.

Configuration: The UART and the External Interrupt 0 (INT0) are configured.
UART: UART is configured for 9600 baud.
External Interrupt 0: Set EX0/IT0 to enable INT0 interrupt as edge triggered
Set EA to allow interrupts

Initialization: The I²C registers and flags are initialized.
I2CCON = A8h => Master Mode

Disables the output driver on SDATA
SCLOCK float high

OUTPUT = 0 Initial byte to be transmitted is '0'.

Note: Since the I²C interface is in master mode there is no need to enable the I²C interrupt, or put a value into I2CADD.

Reception: Reception of a byte is done as follows (see RCVDATA figure 7a)

1) Send the START BIT
2) Send the Slave address (manipulated with R/$\overline{W}$ bit set for reception)
3) Check the ACK.
4) If an NACK is received send a STOP bit and set the ERR flag
5) If an ACK is received then send 8 clocks to the slave device reading the MDI bit after each clock is transmitted. After the 8 clocks the received byte is saved in INPUT
6) Send a NACK to indicate that this is the last byte to be received
7) Send STOP BIT.
8) If a NACK is received then set the ERR flag.

Transmission: Transmission of a byte is done as follows (see SENDDATA figure 7b)

1) Send the START BIT
2) Send the Slave address (manipulated with R/$\overline{W}$ bit clear for transmission)
3) Check the ACK
4) If an NACK is received send a STOP bit and set the ERR flag
5) If an ACK is received then send 8 clocks to the slave device. With each clock the MDO bit in I2CCON should be loaded with the appropriate value from the data byte OUTPUT.
6) Check the ACK
7) If a NACK is received then set the ERR flag.
8) Send STOP BIT

Check ERR: Check the ERR flag to see if an error occurred. If an error occurred send an error message up the UART to the PC.

Transmit i/p: Transmit the inputted byte up the UART to the PC.

Delay: This large delay (approx 1s) is only used to slow the program down for visual purposes so that the LED can be seen to toggle.

Toggle LED: Each toggle of the LED represents that a byte has been received from the slave and that the master has transmitted a byte back to the slave.

Check RI: If a byte is received by the master from the UART then the byte is loaded into the OUTPUT byte and future transmissions will transmit this value as the new OUTPUT.
A byte can be sent to the master by connecting the evaluation board to the PC via the serial port and then pressing any key on the keyboard. In this case the ASCII representation of the character will be loaded into OUTPUT.
E.g. if a '0' is pressed then 30h is loaded into OUTPUT. If 'a' is pressed then 61h is loaded into OUTPUT.

The program now goes back and waits to receive the next data byte.

INTO ISR: Pressing the INT0 button on the evaluation board (causing an INT0 interrupt) will cause the output byte to be incremented.

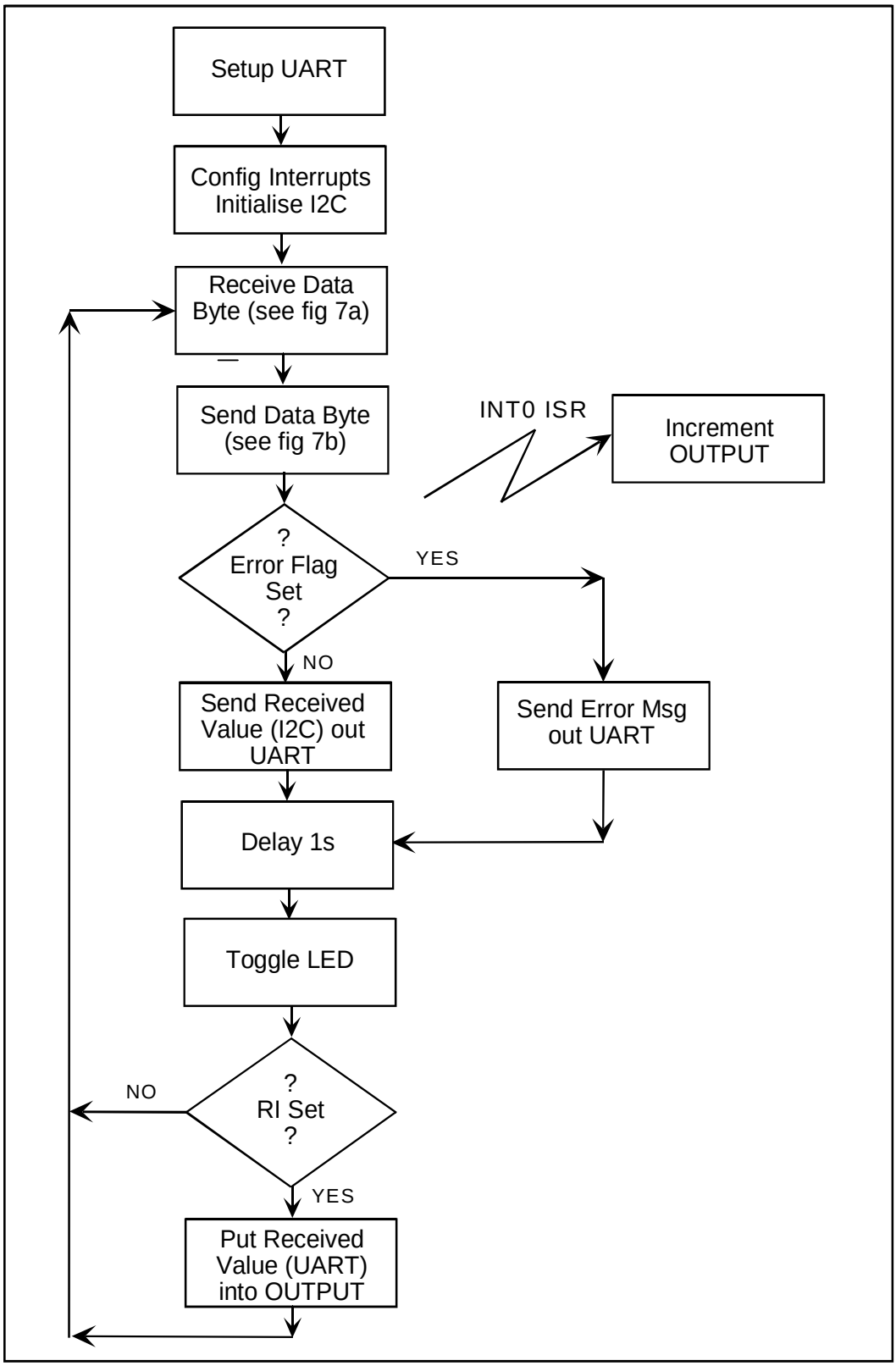

Figure 6: Flow Chart for master example code (I2Cmstr.asm)

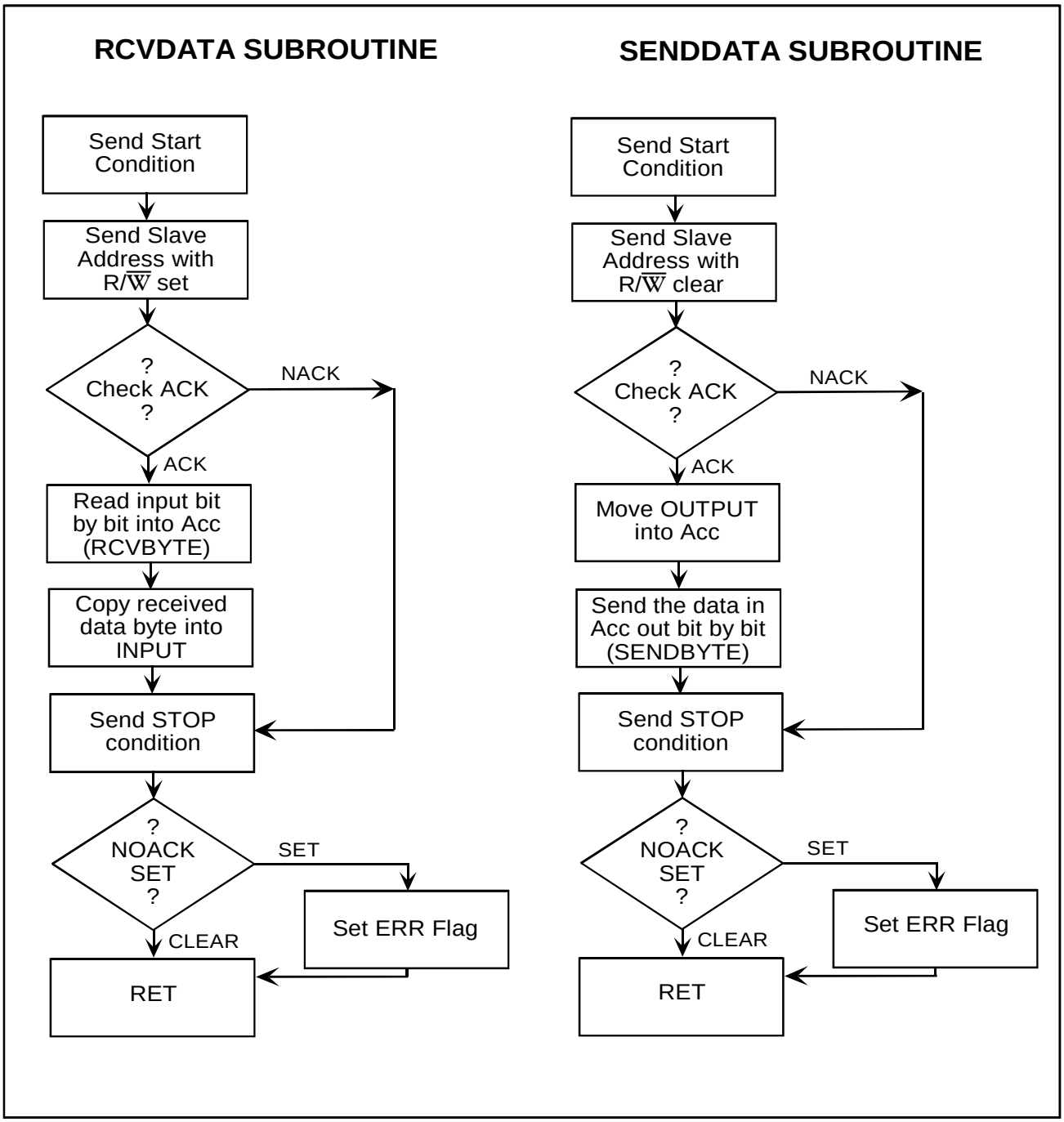

Figure 7a and 7b: Flowchart for the SENDDATA and RCVDATA subroutines.

2.1 I²C SLAVE OPERATION CODE EXAMPLE

This note describes the slave program I2Cslave.asm. This program should be used in conjunction with the master program I2Cmstr.asm on two separate evaluation boards to show the I²C communication between the master and the slave on the MicroConverter. This example (I2Cslave.asm) code demonstrates how the I²C interface on the MicroConverter can be programmed in slave mode.

The slave program waits for an I²C interrupt. Once an interrupt occurs it checks to see if the master requires it to transmit or receive. If it is required to transmit then the I2CDAT register is written to and the slave transmits. If it is required to receive it waits for the next I2CI interrupt before reading in the data. After a reception has occurred the slave program then transmits the received byte up the UART to the PC where it can be viewed using hyperterminal. The program then jumps back to the start waiting for an I²C interrupt.

If a character is received from the UART (by pressing the keyboard), then the slave saves this new byte as its output byte which is transmitted to the master device. By pressing the INT0 button on the evaluation board the data being outputted by the slave to the master will increment.

Flow charts are shown for the slave program (figure 8). Below there is a description of the code and details of the registers, variables etc. used, is also given.

PROGRAM VARIABLES

OUTPUT: OUTPUT holds the value to be transmitted to the slave, this is initialized to 0.

INPUT: INPUT holds the value received from the slave and is sent out the UART to the PC.

GO: The flag GO is used so that the program will wait for an I²C interrupt.

FIRST: The flag FIRST is used in the receive mode, it is set for the first interrupt so the program only reads in the data byte and not the address.

DESCRIPTION OF CODE

The following description of the code I2Cslave.asm should be read in conjunction with the flowchart in figure 8. This program communicates with I2Cmstr.asm which was described in section 2.0.

Configuration: The UART and the External Interrupt 0 (INT0) are configured.

UART:	UART is configured for 9600 baud.
External Interrupt 0:	Set EX0/IT0 to enable INT0 interrupt as edge triggered Set EA to allow interrupts

Initialization: The I²C registers and flags are initialized.

I2CCON = 00h	=>	Slave Mode
I2CCON = 44h		Slave address = 44h
Initialise OUTPUT to 30h		

Wait for I2CI: Wait for an I²C interrupt.

Check I2CTX: Once an I²C interrupt occurs check the I2CTX bit. This tells the slave whether the master wants the slave to transmit or to receive.
If transmitting move 'OUTPUT' into I2CDAT and wait for another interrupt to signify that the byte has been sent.
If receiving wait for a second interrupt (after the first interrupt the address is in I2CDAT). Once the second interrupt occurs then move the contents of I2CDAT into 'INPUT'.

Toggle LED: Each toggle of the LED represents that a byte has been transmitted and received to/from the master.

Transmit i/p: Transmit the inputted byte up the UART to the PC.

Check RI: Move the ASCII representation of any inputted bytes from the keyboard into 'OUTPUT'

The program now goes back and waits for an I²C interrupt.

INTO ISR: Pressing the INT0 button on the evaluation board (causing an INT0 interrupt) will cause the output byte to be incremented.

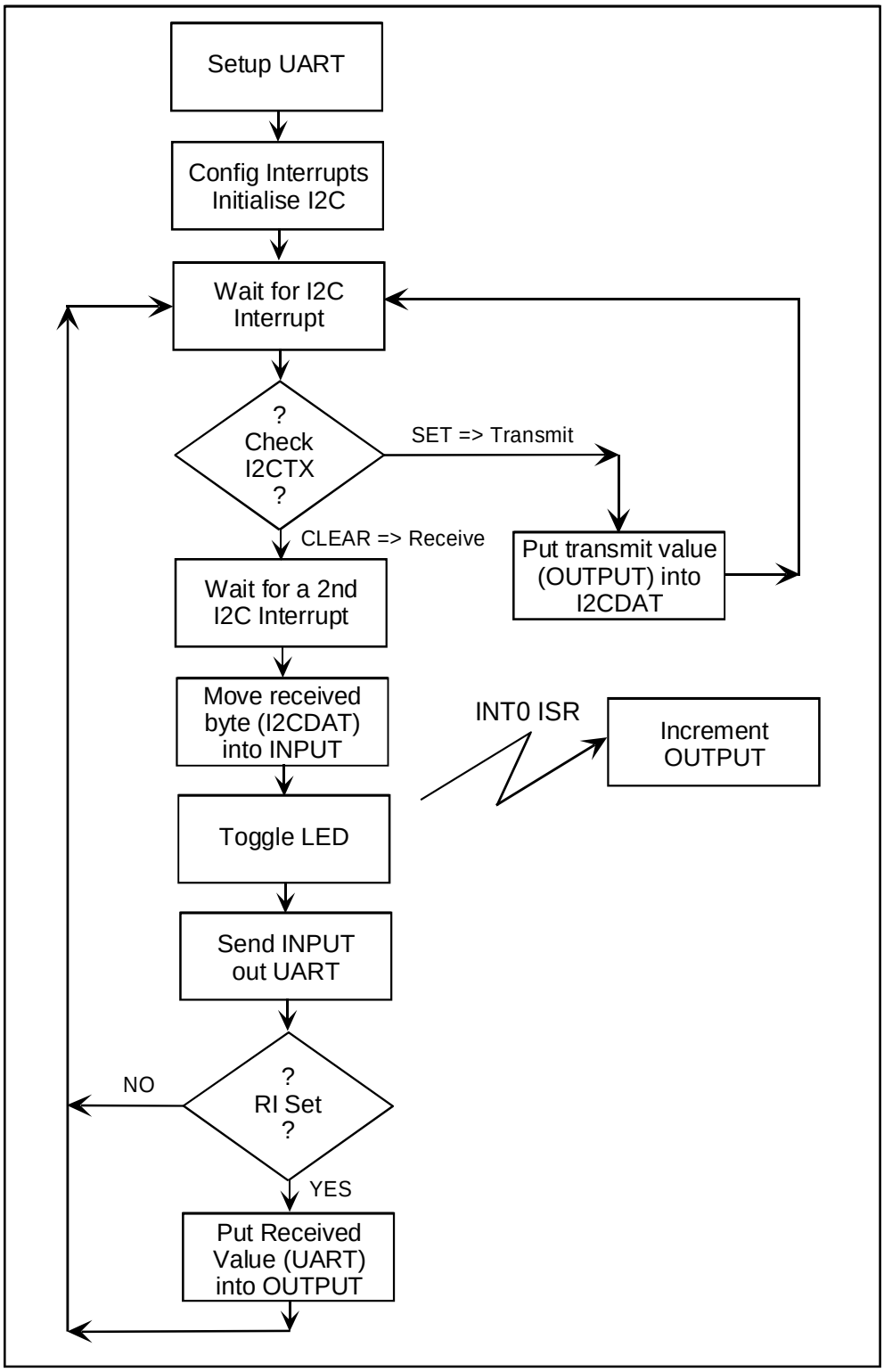

Figure 8: Flow Chart for a typical I²C Slave transmit and receive (I2Cslave.asm)

2.3 I²C MASTER EXAMPLE CODE

```
;=====================================================================
;
; Author        : ADI - Apps              www.analog.com/MicroConverter
;
; Date          : Oct 2000
;
; File          : i2Cmstr.asm
;
; Hardware      : ADuC812 (commented out = ADuC824/ADuC816/ADuC834/ADuC836)
;
; Description   : Code for a master in an I2C system. This code will
;                 continuously receive and transmit a byte over the I2C
;                 interface, then send the received byte out the UART,
;                 then check if a character had been entered in the UART,
;                 if so, it will send the ASCII value of the character
;                 entered to the slave, the next time it transmits a byte.
;
; Reference     : Tech Note, uC001: "MicroConverter I2C Compatible
;                 Interface" find it at www.analog.com/MicroConverter

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

$MOD812                         ; use ADuC812 & 8052 predefined symbols
;$MOD816
;$MOD824

;_____________________________________________________________________
                                  ; DEFINE VARIABLES IN INTERNAL RAM

BITCNT          DATA    30h     ; bit counter for I2C routines
SLAVEADD        DATA    31h     ; slave address for I2C routines
INPUT           DATA    32h     ; data recieved from the slave
OUTPUT          DATA    33h     ; data to be transmitted to slave

NOACK           BIT     00h     ; I2C no acknowledge flag
ERR             BIT     00h     ; I2C error flag

LED             EQU     P3.4



;_____________________________________________________________________
                                                  ; BEGINNING OF CODE
CSEG
ORG 0000h
        JMP MAIN

;_____________________________________________________________________
                                                          ; INT0 ISR
ORG 0003h
        INC     OUTPUT
        RETI
```

```
;_____________________________________________________________________
                                                        ; MAIN PROGRAM
ORG 0060h
MAIN:

; configure the UART ADuC812
        MOV     SCON,#52h       ; configure UART for 9600baud..
        MOV     TMOD,#20h       ; ..assuming 11.0592MHz crystal
        MOV     TH1,#-3
        SETB    TR1

; configure the UART ADuC824/ADuC816
;        MOV     RCAP2H,#0FFh   ; config UART for 9830baud
;        MOV     RCAP2L,#-5     ; (close enough to 9600baud)
;        MOV     TH2,#0FFh
;        MOV     TL2,#-5
;        MOV     SCON,#52h
;        MOV     T2CON,#34h

; configure & enable interrupts
        SETB    EX0             ; enable INT0
        SETB    IT0             ; INT0 edge triggered
        SETB    EA              ; allow all the interrupts

; initialise settings
        MOV     SLAVEADD,#88H   ; clear RW bit
        MOV     I2CCON,#0A8h    ; sets SDATA & SCLOCK, and
                                ; selects master mode
        MOV     OUTPUT,#0       ; TX 0 as default
        CLR     NOACK
        CLR     ERR

RXTXLOOP:
; code for a read mode ( master recieves one byte from slave )
        CALL    RCVDATA         ; sends start bit
                                ; sends address byte
                                ; checks acknowledge
                                ; receives byte into ACC
                                ; checks ACK
                                ; sends stop bit

; code for write mode ( master transmits one byte to slave )
        CALL    SENDDATA        ; sends start bit
                                ; sends address byte
                                ; checks acknowledge
                                ; transmits ACC
                                ; checks ACK
                                ; sends stop bit

; Check for Error message
      JB    ERR,SENDERR         ; if error, send error message

; Transmit received byte (INPUT) up UART to PC (hyperterminal)
      MOV     A,INPUT           ; put value received into ACC
      CALL    SENDVAL           ; send value received out the UART
```

```
      JMP     SKIP

SENDERR:
      CALL    ERROR             ; send error message out the UART
      CLR     ERR               ; clear error flag

SKIP:
      MOV     A,#10             ; send LF+CR
      CALL    SENDCHAR
      MOV     A,#13
      CALL    SENDCHAR

; Toggle LED (1s delay so that LED can be seen toggle)
      MOV     A, #10
      CALL    DELAY
      CPL     LED

; Check for new OUTPUT
      JNB     RI, RXTXLOOP      ; repeat (unless UART data received)

; If UART data received, then save to OUTPUT
      MOV     OUTPUT,SBUF       ; update OUTPUT byte to new value
      CLR     RI                ; must clear RI
      JMP     RXTXLOOP          ; back to main loop


;=====================================================================
;                            SUBROUTINES
;=====================================================================

;_____________________________________________________________________
                                                              ; SENDDATA
; Send all the sequence to the slave (slave address + data (OUTPUT))

SENDDATA:
        ; send start bit
        CALL    STARTBIT        ; acquire bus and send slave address

        ; send slave address
        MOV     A, SLAVEADD
        CALL    SENDBYTE        ; sets NOACK if NACK received

        JB      NOACK, STOPSEND ; if no acknowledge send stop

        ; send OUTPUT byte
        MOV     A, OUTPUT
        CALL    SENDBYTE        ; sets NOACK if NACK received

STOPSEND:
        CALL    STOPBIT         ; sends stop bit
        JNB     NOACK, SENDRET  ; if slave sends NACK send error
        SETB    ERR             ; sets the error flag
SENDRET:
        RET
```

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

RCVDATA:
        INC     SLAVEADD        ; Set RW for reception

        ; send start bit
        CALL    STARTBIT        ; acquire bus and send slave address

        ; send slave address
        MOV     A, SLAVEADD
        CALL    SENDBYTE        ; sets NOACK if NACK received

        DEC     SLAVEADD        ; returns SLAVEADD to 88h (after INC)

        JB      NOACK, STOPRCV  ; Check for slave not responding.
        CALL    RCVBYTE         ; Receive next data byte.
        MOV     INPUT,A         ; Save data byte in buffer.

STOPRCV:
        CALL    STOPBIT
        JNB     NOACK, RCVRET   ; if slave sends NACK send error
        SETB    ERR             ; sets the error flag
RCVRET:
        RET
;_____________________________________________________________________________
                                                                   ; STARTBIT
; Sends the start bit to initiate an I2C communication

STARTBIT:
        SETB    MDE             ; enable SDATA pin as an output
        CLR     NOACK
        CLR     MDO             ; low O/P on SDATA
        CLR     MCO             ; start bit
        RET
;_____________________________________________________________________________
                                                                    ; STOPBIT
; Sends the stop bit to end an I2C transmission

STOPBIT:
        SETB    MDE             ; to enable SDATA pin as an output
        CLR     MDO             ; get SDATA ready for stop
        SETB    MCO             ; set clock for stop
        SETB    MDO             ; this is the stop bit
        RET
;_____________________________________________________________________________
                                                                   ; SENDBYTE
; Send 8-bits in ACC to the slave
SENDBYTE:
        MOV     BITCNT,#8       ; 8 bits in a byte
        SETB    MDE             ; to enable SDATA pin as an output
        CLR     MCO             ; make sure that the clock line is low
SENDBIT:
        RLC     A               ; put data bit to be sent into carry
        MOV     MDO,C           ; put data bit on SDATA line
```

```
        SETB    MCO             ; clock to send bit
        CLR     MCO             ; clear clock
        DJNZ    BITCNT,SENDBIT  ; jump back and send all eight bits

        CLR     MDE             ; release data line for acknowledge
        SETB    MCO             ; send clock for acknowledge
        JNB     MDI,NEXT        ; this is a check for acknowledge
        SETB    NOACK           ; no acknowledge, set flag
NEXT:   CLR     MCO             ; clear clock
        RET
;_____________________________________________________________________________
                                                                    ; RCVBYTE
; receives one byte of data from an I2C slave device. Returns it in A

RCVBYTE:
        MOV     BITCNT,#8       ; Set bit count.
        CLR     MDE             ; to enable SDATA pin as an input
        CLR     MCO             ; make sure the clock line is low
RCVBIT:
        SETB    MCO             ; clock to recieve bit
        CLR     MCO             ; clear clock
        MOV     C,MDI           ; read data bit into carry.
        RLC     A               ; Rotate bit into result byte.

        DJNZ    BITCNT,RCVBIT   ; Repeat until all bits received.
                                ; recieved byte is in the accumulator

        SETB    MDE             ; Data pin =Output for NACK
        SETB    MDO             ; Send NACK (always send NACK for
                                ; last byte in transmission)
        SETB    MCO             ; Send NACK clock.
        CLR     MCO
        RET

;_____________________________________________________________________________
                                                                    ; DELAY
; DELAY ROUTINE FOR THE ADuC812/ADuC816/ADuC824
DELAY:                  ; Delays by 100ms * A

; ADuC812 100ms based on 11.0592MHz Core Clock
; ADuC824 100ms based on 1.573MHz Core Clock

       MOV     R2,A     ; Acc holds delay variable
DLY0:  MOV     R3,#200  ; Set up delay loop0
DLY1:  MOV     R4,#229  ; Set up delay loop1
;DLY0:  MOV     R3,#50    ; Set up delay loop0
;DLY1:  MOV     R4,#131   ; Set up delay loop1
       DJNZ   R4,$      ; Dec R4 & Jump here until R4 is 0
                        ; wait here for 131*15.3us=2ms
       DJNZ   R3,DLY1   ; Dec R3 & Jump DLY1 until R3 is 0
                        ; Wait for 50*2ms
       DJNZ   R2,DLY0   ; Dec R2 & Jump DLY0 until R2 is 0
                        ; wait for ACC*100ms
       RET              ; Return from subroutine
```

```
;_____________________________________________________________________________
                                                                      ; ERROR
; this subroutine is run if a NACK is received from the slave

ERROR:
      MOV     A,#45h
      CALL    SENDCHAR  ; send the letter E out the UART
      RET
;_____________________________________________________________________________
                                                                    ; SENDCHAR
; sends ASCII value contained in A to UART

SENDCHAR:
        JNB     TI,$            ; wait til present char gone
        CLR     TI              ; must clear TI
        MOV     SBUF,A
        RET
;_____________________________________________________________________________
                                                                  ; HEX2ASCII
; converts A into the hex character representing the value of A's
; least significant nibble

HEX2ASCII:
        ANL     A,#00Fh
        CJNE    A,#00Ah,$+3
        JC      IO0030
        ADD     A,#007h
IO0030: ADD     A,#'0'
        RET
;_____________________________________________________________________________
                                                                    ; SENDVAL
; converts the hex value of A into two ASCII chars, and then spits
; these two characters up the UART. does not change the value of A.

SENDVAL:
        PUSH    ACC
        SWAP    A
        CALL    HEX2ASCII
        CALL    SENDCHAR        ; send high nibble
        POP     ACC
        PUSH    ACC
        CALL    HEX2ASCII
        CALL    SENDCHAR        ; send low nibble
        POP     ACC
        RET
;_____________________________________________________________________________

END
```

2.4 I²C SLAVE EXAMPLE CODE

```
;=====================================================================
;
; Author        : ADI - Apps              www.analog.com/MicroConverter
;
; Date          : Oct 2000
;
; File          : i2cslave.asm
;
; Hardware      : ADuC812 (commented out = ADuC816/ADuC824/ADuC834/ADuC86)
;
; Description   : Code for a slave in an I2C system. This code will
;                 continuously receive and transmit a byte over the I2C
;                 interface, then send the received byte out the UART,
;                 then check if a character had been entered in the UART.
;                 If so, it will send the ASCII value of the character
;                 entered to the slave, the next time it transmits a byte.
;
; Reference     : Tech Note, uC001: "MicroConverter I2C Compatible
;                 Interface" find it at www.analog.com/MicroConverter
;
;=====================================================================

$MOD812                         ; use ADuC812 & 8052 predefined symbols
;$MOD816
;$MOD824

;_____________________________________________________________________
                                  ; DEFINE VARIABLES IN INTERNAL RAM

BYTECNT         DATA    30h       ; byte counter for I2C routines
INPUT           DATA    31h       ; data recieved from master
OUTPUT          DATA    32h       ; data to be transmitted to master

GO              BIT     00h       ; flag to wait for interrupts
FIRST           BIT     01h       ; flag to indicate first receive Int

LED             EQU     P3.4      ; P3.4 drives the LED on eval board

;_____________________________________________________________________
                                                  ; BEGINNING OF CODE
CSEG
ORG 0000h
        JMP MAIN
;_____________________________________________________________________
                                                          ; INT0 ISR
ORG 0003h
        INC     OUTPUT
        RETI
;_____________________________________________________________________
                                                          ; I2C ISR
ORG 003Bh

      JB      I2CTX, SLAVE_TRANSMITTER
```

```
SLAVE_RECEIVER:
      JB      FIRST, ENDINT1  ; if first INT then wait for next int
      SETB    GO              ; reception complete
      MOV     INPUT, I2CDAT   ; store data received in INPUT
      JMP     ENDINT1

SLAVE_TRANSMITTER:
      SETB    GO              ; transmission complete
      MOV     I2CDAT, OUTPUT  ; move data to be transmitted into I2CDAT
;      JMP     ENDINT2         ; Note: On the ADuC824/816 the read or
                               ;       write of I2CDAT register
                               ;       automatically clears i2ci. If
                               ;       I2CI is cleared twice then the
                               ;       MicroConverter will hang.)

ENDINT1:
      CLR     I2CI            ; clear I2C interrupt bit (812 only)
ENDINT2:
      CLR     FIRST           ; address has already been received
      RETI

;_____________________________________________________________________________
                                                          ; MAIN PROGRAM
ORG 0060h
MAIN:

; configure the UART ADuC812
      MOV     SCON,#52h        ; configure UART for 9600baud..
      MOV     TMOD,#20h        ; ..assuming 11.0592MHz crystal
      MOV     TH1,#-3
      SETB    TR1

; configure the UART ADuC824/ADuC816
;      MOV     RCAP2H,#0FFh    ; config UART for 9830baud
;      MOV     RCAP2L,#-5      ; (close enough to 9600baud)
;      MOV     TH2,#0FFh
;      MOV     TL2,#-5
;      MOV     SCON,#52h
;      MOV     T2CON,#34h

;configure and enable interrupts
      MOV     IE2,#01h        ; enable I2C interrupt
;      MOV     IEIP2,#01h      ; enable I2C interrupt
      SETB    EX0             ; enable INT0
      SETB    IT0             ; INT0 edge triggered
      SETB    EA              ; allow all the interrupts

;initialize settings
      MOV     I2CADD,#044h    ; slave address is 44h
      MOV     I2CCON,#00h     ; slave mode (default=>not necessary)
      CLR     GO              ; clear flag to wait for interrupt
                              ; GO is set once data is TX'd or RX'd
      SETB    FIRST           ; FIRST is cleared after receiving the
                              ; first SLAVE receiver interrupt
```

```
        MOV     OUTPUT,#0       ; first byte to be transmitted is 40h
        CLR     LED

WAITFORDATA:
        JNB     GO,$            ; ----- wait for i2c interrupt ------
                                ; If it is in receive mode, it will
                                ; wait here for a second interrupt (as
                                ; the first interrupt only contains the
                                ; slave address in I2CDAT).
                                ; In transmit mode the tranmission will
                                ; occur after the first interrupt.
        SETB    FIRST           ; re-initialise flags
        CLR     GO
        JB      I2CTX,WAITFORDATA
                                ; if the slave has just transmitted then
                                ; wait to receive a byte
                                ; if the slave has just received then
                                ; send input up the UART

SENDUART:
        CPL     LED             ; LED changes each time one byte has been
                                ; received and another transmitted
        MOV     A,INPUT         ; send value received out the UART
        CALL    SENDVAL
        MOV     A,#10
        CALL    SENDCHAR        ; send LF + CR
        MOV     A,#13
        CALL    SENDCHAR

        JNB     RI, WAITFORDATA ; repeat (unless UART data received)

; WHEN UART DATA RECEIVED, MOVE DATA TO I2C OUTPUT...

        MOV     OUTPUT, SBUF    ; update OUTPUT byte to new value
        CLR     RI              ; must clear RI
        JMP     WAITFORDATA     ; back to main loop


;=====================================================================
;                             SUBROUTINES
;=====================================================================

;_____________________________________________________________________
                                                              ; SENDCHAR
; sends ASCII value contained in A to UART

SENDCHAR:
          JNB     TI,$            ; wait 'til present char gone
          CLR     TI              ; must clear TI
          MOV     SBUF,A
          RET
```

```
;_____________________________________________________________________________
                                                                    ; HEX2ASCII
; converts A into the hex character representing the value of A's
; least significant nibble

HEX2ASCII:
        ANL     A,#00Fh
        CJNE    A,#00Ah,$+3
        JC      IO0030
        ADD     A,#007h
IO0030: ADD     A,#'0'
        RET
;_____________________________________________________________________________
                                                                    ; SENDVAL
; converts the hex value of A into two ASCII chars, and then spits
; these two characters up the UART. does not change the value of A.

SENDVAL:
        PUSH    ACC
        SWAP    A
        CALL    HEX2ASCII
        CALL    SENDCHAR        ; send high nibble
        POP     ACC
        PUSH    ACC
        CALL    HEX2ASCII
        CALL    SENDCHAR        ; send low nibble
        POP     ACC
        RET
;_____________________________________________________________________________

END
```