APPLICATION NOTE

AN168
The I^{2}C serial bus: theory and practical consideration using Philips low-voltage PCF84Cxx and PCD33xx μC families

Author: Carl Fenger

1988 Dec

Philips
Semiconductors

PHILIPS

The I^2C serial bus: theory and practical consideration using Philips low-voltage PCF84Cxx and PCD33xx µC families

Author: Carl Fenger

INTRODUCTION

The I^2C (Inter-IC) bus has become a popular serial bus architecture which needs to be understood for proper implementation. On the hardware level, I^2C is a collection of microcomputers with integrated I^2C port (Philips PCD33xx, PCF84Cxxx, and many of their 80(C)51 family derivatives, plus µCs from several other manufacturers), and a peripheral set (LCD/LED drivers, RAM, ROM, E^2PROM, Clock/Calendars, I/O, A/D, D/A, IR transcoders, frequency synthesizers, audio processors, telephony ICs and various tuning ICs for TV/radio). These devices all communicate serially over a two-wire bus, serial data (SDA) and serial clock (SCL). The I^2C structure is optimized for hardwire simplicity. Parallel address and data buses inherent in conventional systems are replaced by a serial protocol that transmits both address and bidirectional data over a 2-wire bus. This means that interconnecting wires are reduced to a minimum; only V_{DD}, ground, and the two-wire bus are required to link the controller(s) with the peripherals or other controllers. This results in reduced IC size, reduced pin count, and simpler interconnections. An I^2C system is therefore smaller, simpler, and cheaper to implement than its parallel counterpart.

The data rate of the I^2C bus (100K bits/sec, with 400K bit/sec devices in development) makes it suited for systems that do not require high speed. The I^2C architecture is thus well-suited for use in applications such as handheld products (telephone handsets, cordless phones), television and other consumer electronics, appliances, medical instruments, general instrumentation panels, and any application involving human interface. Typically an I^2C system would be used in a control function where digitally controllable elements are adjusted and monitored by a human user via a central processor.

The I^2C bus is an innovative hardware interface which provides the software designer the flexibility to create a truly multi-master environment. Built into the serial interface of the controllers are status registers which monitor all possible bus conditions: bus free/busy, bus contention, slave acknowledgement, and bus interference. Thus an I^2C system might include several controllers on the same bus each with the ability to asynchronously communicate with peripherals or each other. This provision also provides expandability for future add-on controllers. (The I^2C system is also ideal for use in environments where the bus is subject to noise. Distorted transmissions are immediately detected by the hardware and the information presented to the software.) A slave acknowledgement on every byte also facilitates data integrity.

An I^2C system can be as simple or sophisticated as the operating environment demands. Whether in a single master or multi-master system, noisy or ‘safe’, correct system operation can be insured under software control.

CONTROLLERS

The Philips family of I^2C microcontrollers and microprocessors has grown to encompass mainly devices based on the Intel 8048, 8051, and Motorola 68000 cores. These devices have various degrees of I^2C port implementaion which dictates to which degree the I^2C protocol generation and data transmission/reception is executed in hardware vs software. Indeed, any standard microcontroller is capable of implementing I^2C on a normal open-drain port, in which case the protocol is 100% software generated (‘bit banging’). The I^2C port itself, even when fully hardware implemented, requires only a handful of instructions to control and monitor the I^2C bus. Hence, the I^2C port can be considered a core-independent interface.

Two families of Philips microcontrollers which include members fully implementing the I^2C interface on-chip are the PCF84Cxxx and PCD33xx families of 8-bit low-voltage microcontrollers. These micros are optimized for low-power, low-voltage (V_{DD} min. = 1.8V) and are hence ideal for battery powered, cordless products. These families implement the 8048 instruction set, with a few instructions deleted and replaced by I^2C-port

Table 1. PCF84Cxxx Family Instructions not in the Instruction Set

Serial I/O	Register	Control	Conditional Branch
MOVA,Sn	DEC@Rr	SEL MB2	JNTF addr
MOV Sn,A	DJNZ@Rr,addr	SEL MB3	
MOV Sn,#data			
EN SI			
DIS SI			

Table 2.PCF84Cxxx Instructions not in the 8400 Family Instruction Set

Data Moves	Flags	Branch	Control
MOVXA,@R	CLRF0	*JNI addr	ENTOCLK
MOVXA,@R,A	CPLF0	JF0 addr	
MOVP3A,@A	CLRF1	JF1 addr	
MOVDA,P	CPLF1		
NALDP,A			
ORLDP,A			
*replaced by JT0, JNT0			

specific instructions. The I^2C instructions involve moving data to and from the S0, S1, and S2 serial I/O control registers. The block diagram of the I^2C interface is shown in Figure 1.

SERIAL I/O INTERFACE

A block diagram of the Serial Input/Output (SIO) of the PCF84Cxxx family is shown in Figure 1. The clock line of the serial bus (SCL) has exclusive use of Pin 3, while the Serial Data (SDA) line shares Pin 2 with parallel I/O signal P2.3 of port 2. Consequently, only three I/O lines are available for port 2 when the I^2C interface is enabled.

Communication between the CPU and I^2C interface takes place via the internal bus of the microcomputer and the Serial I/O Interrupt Request line (or via polling of status bits). Four registers are used to store data and information controlling the operation of the interface:

- data shift register S0
- address register S0’
- status register S1
- clock control register S2

THE I^2C BUS INTERFACE SERIAL CONTROL REGISTERS S0, S1

All serial I^2C transfers occur between the accumulator and register S0. The I^2C hardware takes care of clocking out/in the data, and receiving/generating an acknowledge. In addition, the state of the I^2C bus is controlled and monitored via the bus control register S1. A definition of the registers is as follows:

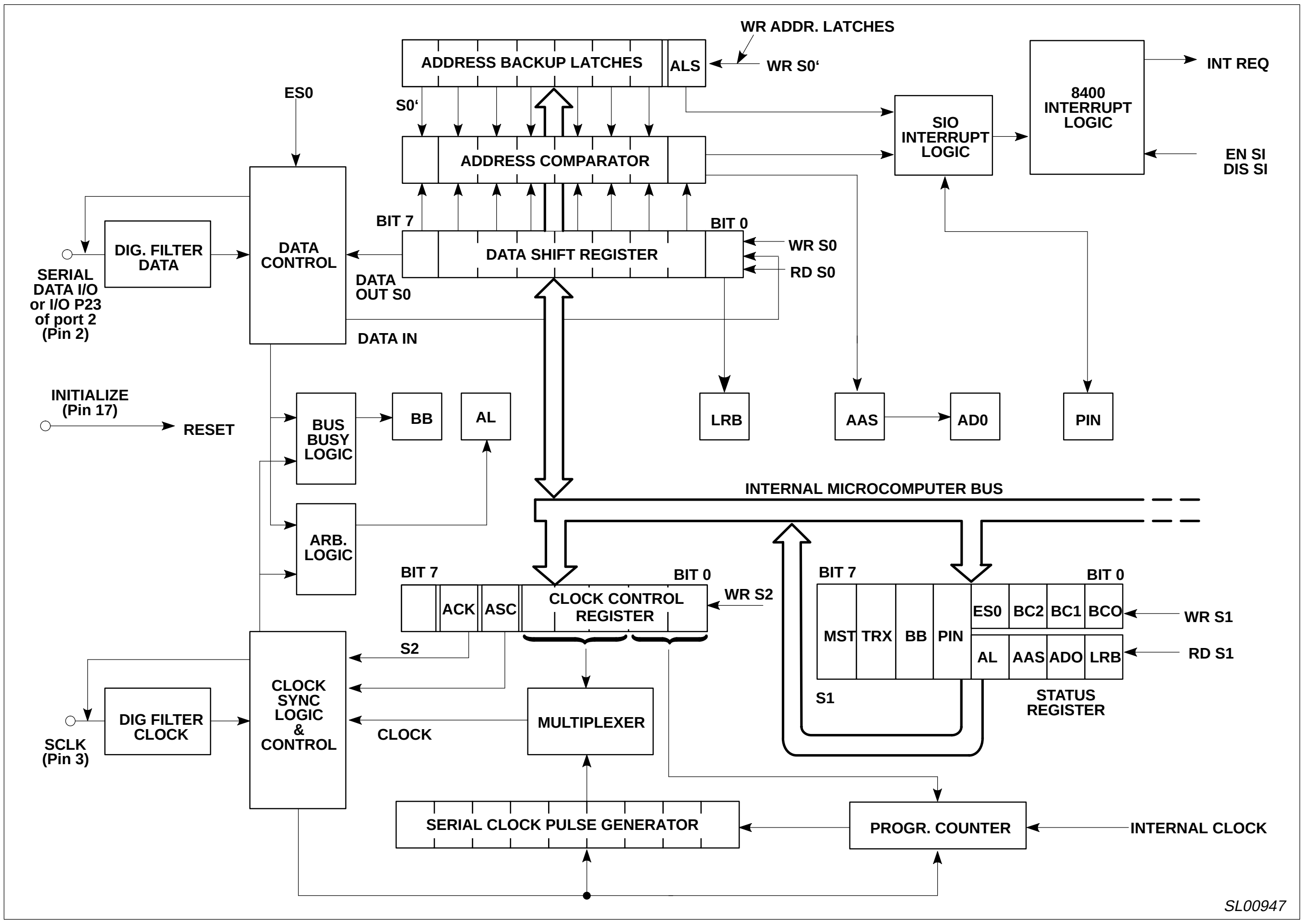


Figure 1. Block Diagram of the PCF84Cxxx I²C Interface

Data Shift Register S0 – S0 is the data shift register used to perform the conversion between serial and parallel data format. All transmissions or receptions take place through register S0 MSB first. All I²C bus receptions or transmissions involve moving data to/from the accumulator from/to S0.

Address Register S0' – In multi-master systems, this register is loaded with a controller's slave address. When activated, (ALS = 0), the hardware will recognize when it is being addressed by setting the AAS (Addressed As Slave) flag. This provision allows a master to be treated as a slave by other masters on the bus.
Status Register S1 – S1 is the bus status register. To control the SIO interface, information is written to the register. The lower 4 bits in S1 serve dual purposes; when written to, the control bits ES0, BC2, BC1, BC0 are programmed (Enable Serial Output and a 3-bit counter which indicates the current number of bits left in a serial transfer). When reading the lower four bits, we obtain the status information AL, AAS, AD0, LRB (Arbitration Lost, Addressed As Slave, Address Zero (the general call has been received), the Last Received Bit (usually the acknowledge bit)). The upper 4 bits are the MST, TRX, BB and PIN control bits (Master, Transmitter, Bus Busy, and Pending Interrupt Not). These bits define what role the controller has at any particular time. The values of the master and transmitter bits define the controller as either a master or slave (a master initiates a transfer and generates the serial clock; a slave does not), and as a transmitter or receiver. Bus Busy keeps track of whether the bus is free or not, and is set and reset by the 'Start' and 'Stop' conditions which will be defined. Pending Interrupt Not is reset after the completion of a byte transfer + acknowledge, and can be polled to indicate when a serial transfer has been completed. An alternative to polling the PIN bit is to enable the serial interrupt; upon completion of a byte transfer, an interrupt will vector program control to location 07H.

CLOCK CONTROL REGISTER (S2)

The clock control register of the PCF84Cxxx family defines the frequency of f_{SCLK} as the microcontroller clock frequency divided by an integer (Table 2). It also defines ASC (asymmetrical clock) and ACK (acknowledge).

If ASC=1, the generated SCLK has a duty cycle of about 75%. The asymmetrical clock limites I²C bus transmission rate to below 55kHz. Divisors 39, 45 and 51 are not allowed if ASC=1. However, an SCLK duty cycle of about 50% results if ASC=0. This permits I²C bus transmission rates of up to 100kHz. All divisors of NO TAG are available. Therefore, it is recommended to select ASC=0.

For the normal I²C bus protocol, ACK must be set. After each byte transfer, an extra SCLK pulse is generated during which the receiver may acknowledge reception. If ACK

3. f_{SCLK} as defined by clock control register S2 of the PCF84Cxxx μC family

CC4 to CC0	Divisor of f_{XTAL} (DF)	f_{SCLK} (kHz) at f_{XTAL} = 3.58MHz	f_{SCLK} (kHz) at f_{XTAL} = 10MHz	f_{SCLK} (kHz) at f_{XTAL} = 16MHz
H'00'	forbidden	—	—	—
H'01'	39	91.8	*256.4	*410.3
H'02'	45	79.5	*222.2	*355.6
H'03'	51	70.2	*196.1	*313.7
H'04'	63	56.8	*158.7	*254.0
H'05'	75	47.7	*133.3	*213.3
H'06'	87	41.1	*114.9	*183.9
H'07'	99	36.2	*101.0	*161.6
H'08'	123	29.1	81.3	*130.1
H'09'	147	24.4	68.0	*108.8
H'0A'	171	20.9	58.5	93.6
H'0B'	195	18.4	51.3	82.1
H'0C'	243	14.7	41.2	65.8
H'0D'	291	12.3	34.4	55.0
H'0E'	339	10.6	29.5	47.2
H'0F'	387	9.2	25.8	41.3
H'10'	483	7.4	20.7	33.1
H'11'	579	6.2	17.3	27.6
H'12'	675	5.3	14.8	23.7
H'13'	771	4.6	13.0	20.8
H'14'	963	3.7	10.4	16.6
H'15'	1155	3.1	8.7	13.9
H'16'	1347	2.7	7.4	11.9
H'17'	1539	2.3	6.5	10.4
H'18'	1923	1.9	5.2	8.3
H'19'	2307	1.6	4.3	6.9
H'1A'	2691	1.3	3.7	5.9
H'1B'	3075	1.2	3.3	5.2
H'1C'	3843	0.9	2.6	4.2
H'1D'	4611	0.8	2.2	3.5
H'1E'	5379	0.7	1.9	3.0
H'1F'	6147	0.6	1.6	2.6
*Not permitted in non-FAST I^2C systems; maximum specified f_{SCLK} = 100kHz.				

is zero, no acknowledge is generated. This mode is temporarily used when a master/receiver refuses the acknowledgement in order to signal an end of transmission to the slave transmitter (see the section on the bit counter bits BC0 to BC2 in the status register S1). The clock control register S2 is write-only. It can be written by MOV S2,A and MOV S2,#data.

These speeds represent the frequency of the serial clock bursts and do not reflect the speed of the processor's main clock (i.e., it controls the bus speed and has no effect on the CPU's execution speed).

BUS ARBITRATION

Due to the wire-AND configuration of the I^2C bus, and the self-synchronizing clock circuitry of I^2C masters, controllers with varying clock speeds can access the bus without clock contention. During arbitration, the resultant clock on the bus will have a low period equal to the longest of the low periods; the high period will equal the shortest of the high periods. Similarly, when two masters attempt to drive the data line simultaneously, the data is 'ANDed', the master generating a low while the other is driving a high will win arbitration. The resultant bus level will be low, and the loser will withdraw from the bus and set its 'Arbitration Lost' flag (S1 bit 3).

The losing Master is now configured as a slave which could be addressed during this very same cycle. These provisions allow for a number of microcomputers to exist on the same bus. With properly written subroutines, software for any one of the controllers may regard other masters as transparent.

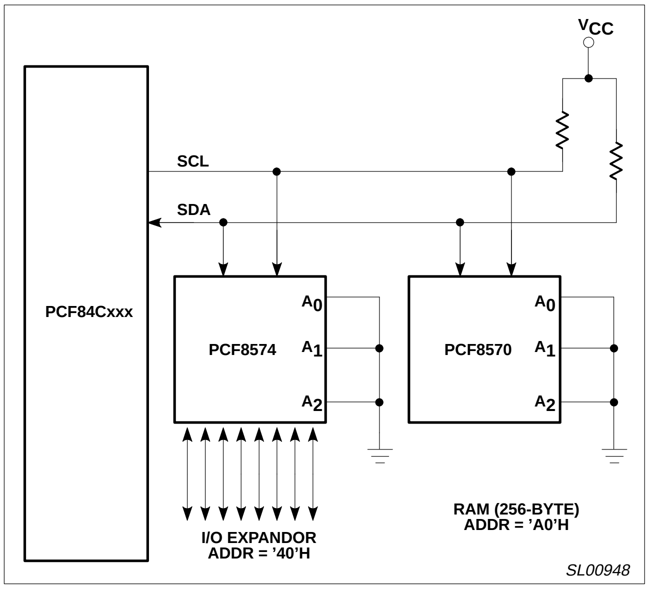


Figure 2. Schematic for Assembly Examples

I²C PROTOCOL AND ASSEMBLY LANGUAGE EXAMPLES

I²C data transfers follow a well-defined protocol. A transfer always takes place between a master and a slave. Currently a microcomputer can be master or slave, while the 'CLIPS' peripherals are always slaves. In a 'bus-free' condition, both SCL and SDA lines are kept logical high be external pull-up resistors. All bus transfers are bounded by a 'Start' and a 'Stop' condition. A 'Start' condition is defined as the SDA line making a high-to-low transition **while the SCL line is high**. At this point, the internal hardware on all slaves are activated and are prepared to clock-in the next 8 bits and interpret it as a 7-bit address and a R/$\overline{W}$ control bit (MSB first). All slaves have an internal address (most have 2–3 programmable address bits) which is then compared with the received address. The slave that recodnized its address will respond by pulling the data line low during a ninth clock generated by the master (all I²C byte transfers require the master to generate 8 clock pulses plus a ninth acknowledge-related clock pulse). The slave-acknowledge will be registered by the master as a '0' appearing in the LRB (Last Received Bit) position of the S1 serial I/O status register. If this bit is high after a transfer attempt, this indicates that a slave did not acknowledge and that the transfer should be repeated.

After the desired slave has acknowledged its address, it is ready to either send or receive data in response to the master's driving clock. All other slaves have withdrawn from the bus. In addition, for multi-master systems, the start condition has set the 'Bus Busy' bit of the serial I/O register S1 on all masters on the bus. This gives a software indication to other master that the bus is in use and to wait until the bus is free before attempting an access.

There are two types of I²C peripherals that now must be defined: there are those with only a chip address such as the I/O expandor, PCF8574, and those with a chip address plus an internal address such as the static RAM, PCF8570. Thus after sending a start condition, address, and R/$\overline{W}$ bit, we must take into account what type of slave is being addressed. In the case of a slave with only a chip address, we have already indicated its address and data direction (R/$\overline{W}$) and are therefore ready to send or receive data. This is performed by the master generating bursts of 9 clock pulses for each byte that is sent or received. The transaction for writing one byte to a slave with a chip address only is shown in Figure 3.

In this transfer, all bus activity is invoked by writing the appropriate control byte to the serial I/O control register S1, and by moving data to/from the serial bus buffer register S0. Coming from a known state (MOV S1, #18H-Slave, Receiver, Bus not Busy) we first load the serial I/O buffer S0 with the desired slave's address (MOV S0, #40H). To transmit this preceded by a start condition, we must first examine the control register S1, which, after initialization, looks like this:

MAS-TER	TRANS	BUS BUSY	PIN	ES0	BC2	BC1	BC0
0	0	0	1	1	0	0	0

To transmit to a slave, the Master, Transmitter, Bus Busy, PIN (Pending Interrupt Not), and ESO (Enable Serial Output) must be set to a 1. This results in an 'F8H' being written to S1. This word defines the controller as a Master Transmitter, invokes the transfer by setting the 'Bus Busy' bit, clears the Pending Interrupt Not (an active low flag indicating the completion of a compete byte transfer), and activates the serial output logic by setting the Enable Serial Output (ESO) bit.

BIT COUNTER S12, S11, S10

BC2, BC1 and BC0 comprise a bit-counter which indicates to the logic how long the word is to be clocked out over the serial data line. By setting this to a 000H, we are telling
it to produce 9 clocks (8 bits plus an acknowledge clock) for this transfer. The bit counter will then count off each bit as it is transmitted. The bit counter possibilities are shown in NO TAG.

Thus, the bit counter keeps track of the number of clock pulses remaining in a serial transfer. Additionally, there is a not-acknowledge mode (controlled through bit 6 of clock control register S2) which inhibits the acknowledge clock pulse, allowing the possibility of straight serial transfer. We may thus define the word size for a serial transfer (by pre-loading BC2, BC1, BC0 with the appropriate control number), with or without an acknowledge-related clock pulse being generated. This makes the controller able to transmit serial data to most any serial device regardless of its protocol (e.g., C-bus devices).

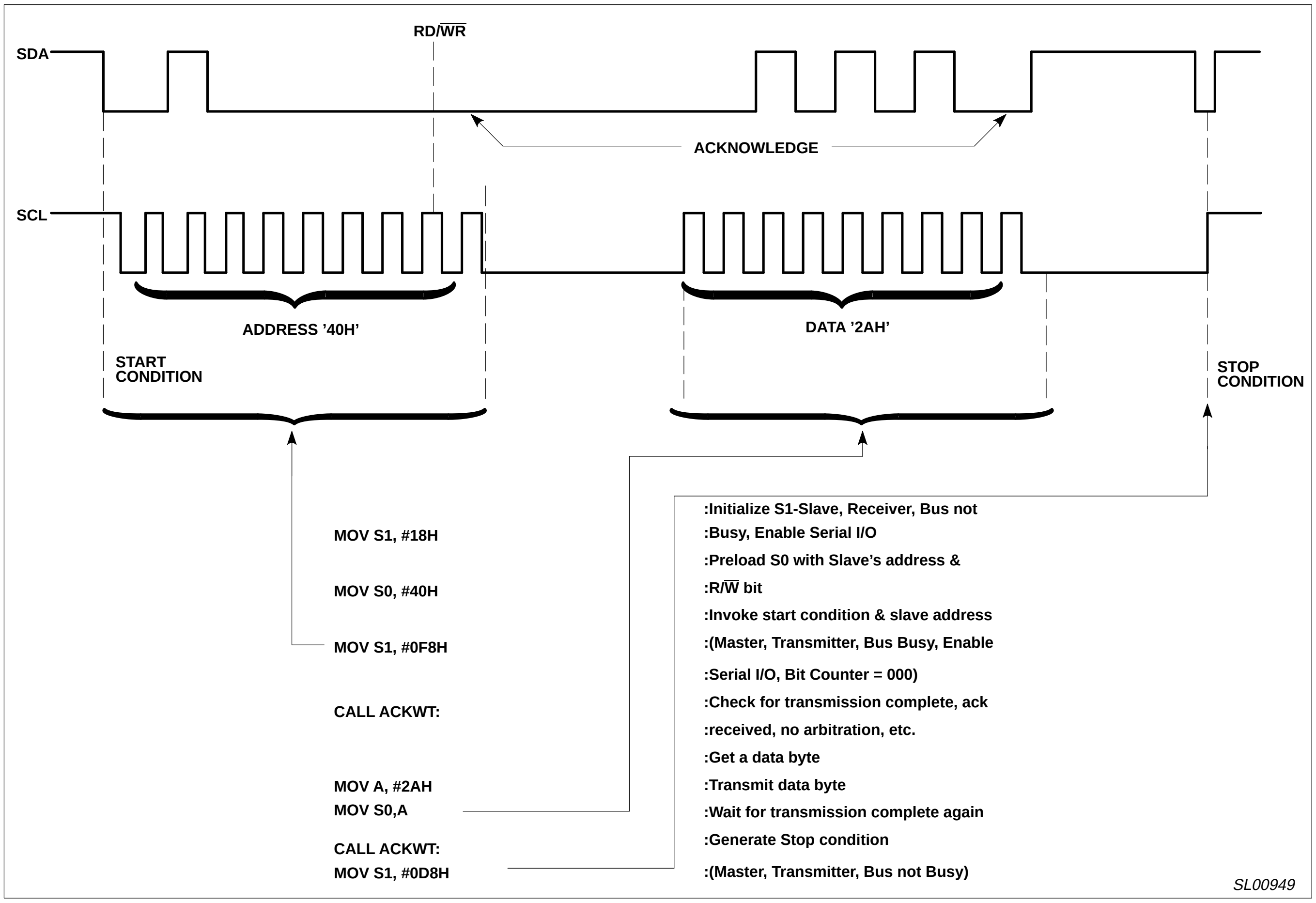

RD/$\overline{WR}$
SDA
ACKNOWLEDGE
SCL
ADDRESS '40H'
DATA '2AH'
START CONDITION
STOP CONDITION
MOV S1, #18H
MOV S0, #40H
MOV S1, #0F8H
CALL ACKWT:
MOV A, #2AH
MOV S0,A
CALL ACKWT:
MOV S1, #0D8H
:Initialize S1-Slave, Receiver, Bus not
:Busy, Enable Serial I/O
:Preload S0 with Slave's address &
:R/$\overline{W}$ bit
:Invoke start condition & slave address
:(Master, Transmitter, Bus Busy, Enable
:Serial I/O, Bit Counter = 000)
:Check for transmission complete, ack
:received, no arbitration, etc.
:Get a data byte
:Transmit data byte
:Wait for transmission complete again
:Generate Stop condition
:(Master, Transmitter, Bus not Busy)
SL00949

Figure 3.

Table 4. Binary Numbers in Bit-Count Locations BC2, BC1 and BC0

BC2	BC1	BC0	Bits/Byte without ACK	Bits/Byte with ACK
0	0	1	1	2
0	1	0	2	3
0	1	1	3	4
1	0	0	4	5
1	0	1	5	6
1	1	0	6	7
1	1	1	7	8
0	0	0	8	9

CHECKING FOR SLAVE ACKNOWLEDGE

After a 'Start' condition and address have been issued, the selected slave will have recognized and acknowledged its address by pulling the data line low during the ninth clock pulse. During this period, the software (which runs on the processor's main clock) will have been either waiting for the transfer to be completed by polling the PIN bit in S1 which goes low on completion of a transfer/reception (whose length is defined by the pre-loaded Bit-counter value), or by the hardware in Serial Interrupt mode. The serial interrupt (vectored to 07H) is enabled via the EN SI (enable serial interrupt) instruction.

At the point when PIN goes low (or the serial interrupt is received) the 9-bit transfer has been completed. The acknowledgement bit will now be in the LRB position of register S1, and may be checked in the routine 'ACKWT' (Wait for Acknowledge) as shown in Figure 4.

This routine must go one step further in multi-master systems; the possibility of an Arbitration Lost situation may occur if other masters are present on the bus. This condition may be detected by checking the 'AL' bit (bit 3). If arbitration has been lost, provisions for re-attempting the transmission should be taken. If arbitration is lost, there is the possibility that the controller is being addressed as a Slave. If this condition is to be recognized, we must test on the 'AAS' bit (bit 2). A 'General Call' address (00H) has also been defined as an 'all-call' address for all slaves; bit 1, AD0, must be tested if this feature is to be recognized by a Master.

After a successful address transfer/acknowledge, the slave is ready to be sent its data. The instruction MOV S0,A will now automatically send the contents of the accumulator out on the bus. After calling the ACKWT routine once more, we are ready to terminate the transfer. The Stop condition is created by the instruction 'MOV S1, #0D8H'. This re-sets the bus-busy bit, which tells the hardware to generate a Stop – the data line makes a low-to-high transition while the clock remains high. All bus-busy flags on other masters on the bus are reset by this signal.

The transfer is now complete – PCF8574 I/O Expandor will transfer the serial data stream to its 8 output pins and latch them until further update.

```
ACKWT:   MOV A,S1       ;Get bus status word
                        ;from S1.
         JB4 ACKWT      ;Poll the PIN bit
                        ;until it goes low
                        ;indicating transfer
                        ;completed
         JBO BUSERR     ;Jump to BUSERR
                        ;routine if acknowledge
                        ;not received.
         RET            ;transfer complete,
                        ;acknowledge received – return.
                                                SL00950
```

Figure 4.

MASTER READS ONE BYTE FROM SLAVE

A read operation is a similar process; the address, however, will be 41H, the LSB indicating to the I/O device that a read is to be performed. During the data portion of a read, the I/O port 8574 will transmit the contents of its latches in response to the clock generated by the master. The Master/Receiver in this case generates a low-level acknowledge on reception of each byte (a 'positive' acknowledge). Upon completion of a read, the master must generate a 'negative' acknowledge during the ninth clock to indicate to the slaves that the read operation is finished. This is necessary because an arbitrary number of bytes may be read within the same transfer. A negative acknowledge consists of a high signal on the data line during the ninth clock of the last byte to be read. To accomplish this, the master must leave the acknowledge mode just before the final byte, read the final byte (producing only 8 clock pulses), program the bit-counter with 001 (preparing for a one-bit negative acknowledge pulse), and simply move the contents of S0 to the accumulator. This final instruction accomplishes two things simultaneously: it transfers the final byte to the accumulator and produces one clock pulse on the SCL line. The structure of the serial I/O register S0 is such that a read from it causes a double-buffered transfer from the I²C bus to S0, while the original contents of S0 are transferred to the accumulator. Because the number of clocks produced on the bus is determined by the control number in the Bit Counter, by presetting it to 001, only one clock is generated. At this point in time the slave is still waiting for an acknowledge; the bus is high due to the pull-up, as single clock pulse in this condition is interpreted as a 'negative' acknowledge. The slave has now been informed that reading is completed, a Stop condition is now generated as before. The read process (one byte from a slave with only a chip address) is shown in Figure 5.

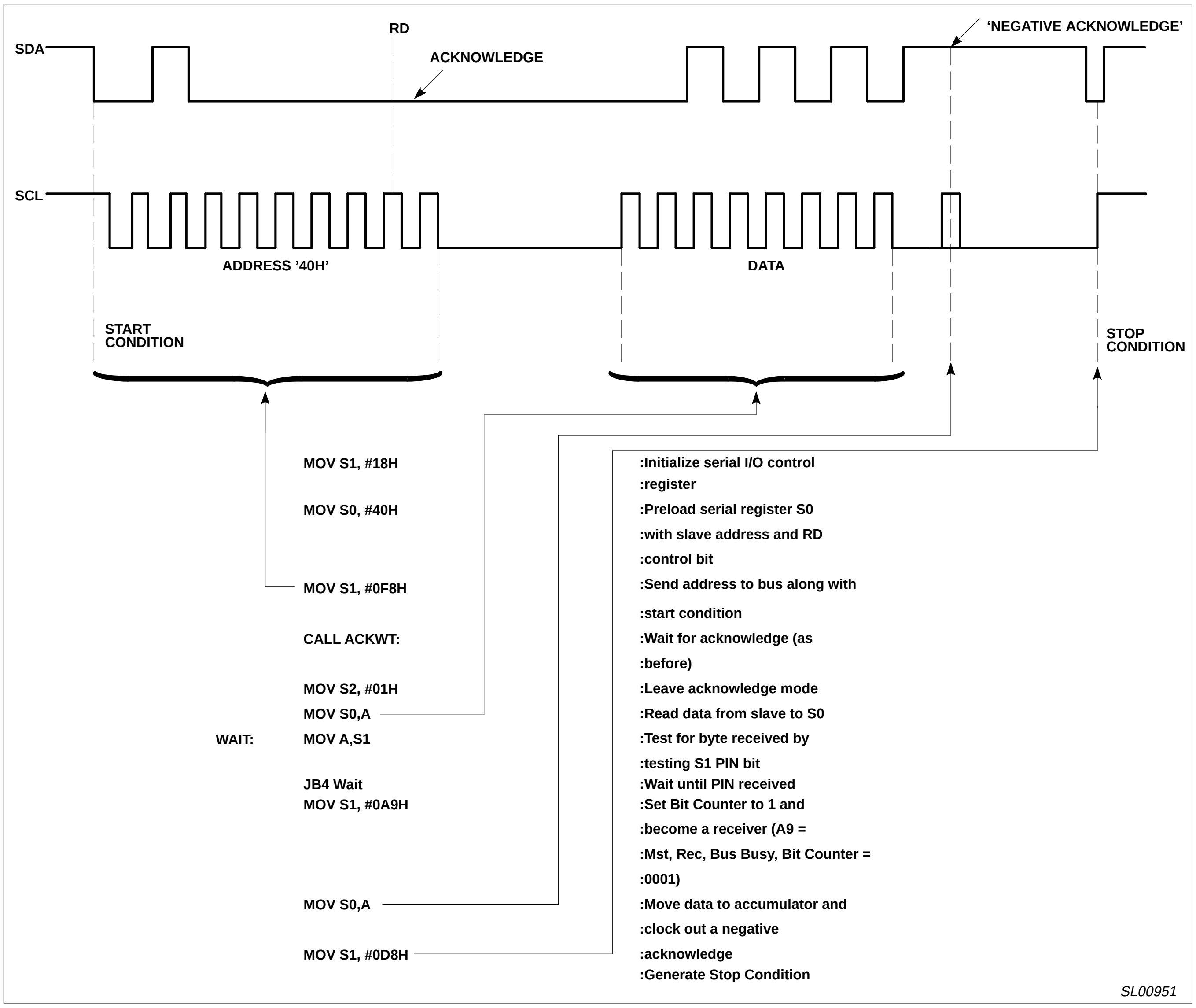
These examples apply to a slave with a chip address – more than one byte can be written/read within the same transfer; however, this option is more applicable to I²C devices with sub-addresses such as the static RAMs or Clock/Calendar. In the case of these types of devices, a slightly different protocol is used. The RAM, for example, requires a chip address and an internal memory location before it can deliver or accept a byte of information. During a write operation, this is done by simply writing the secondary address right after the chip address – the peripheral is designed to interpret the second byte as an internal address. In the case of a Read operation, the slave peripheral must send data back to the Master after it has been addressed and sub-addressed. To accomplish this, first the Start, Address, and Sub-address is transmitted. Then we have repeated start condition to reverse the direction of the data transfer, followed by the chip address and RD, than a data string (w/acknowledges). This repeated Start does not affect other peripherals – they have been deactivated and will not reactivate until a Stop condition is detected. I²C peripherals are equipped with auto-incrementing logic which will automatically transmit or receive data in consecutive (increasing) locations. For example, to read 3 consecutive bytes to PCF8571 RAM locations 00, 01 and 02, we use the following format as shown in Figure 7.

This routine reads the contents of location 00, 01 and 02 of the PCF8570 256-byte RAM and puts them in registers R0, R1, and R2. The auto-incrementing feature allows the programmer to indicate only a starting location, then read an arbitrary block of consecutive memory addresses. The WAIT 1 loop is required to poll for the completion of the final byte because the ACKWT routine will not recognize the negative acknowledge as a valid condition.

BUS ERROR CONDITIONS: ACKNOWLEDGE NOT RECEIVED

In the above routines, should a slave fail to acknowledge, the condition is detected during the 'ACKWT' routine. The occurrence may indicate one of two conditions: the slave has failed to operate, or a bus disturbance has occurred. The software response to either event is dependent on the system application. In either case, the 'BusErr' routine should reinitialize the bus by issuing a 'Stop' condition. Provision may then be taken to repeat the transfer an arbitrary number of times. Should the symptom persist, either an error condition will be entered, or a backup device can be activated.

These sample routines represent single-master systems. A more detailed analysis of multi-master/noisy environment systems are treated in other application notes. For more complex systems, the 80C51 derivative microcontrollers with I²C interface are recommended. Philips 80C51 micros implement a slightly modified I²C interface and conventions, but operate in a similar fashion to the micros described here. (See Philips Semiconductors AN430, "The 83/87C51/752 in a multi-master I²C environment".)



```
         MOV S1, #18H      :Initialize serial I/O control
                           :register
         MOV S0, #40H      :Preload serial register S0
                           :with slave address and RD
                           :control bit
         MOV S1, #0F8H     :Send address to bus along with
                           :start condition
         CALL ACKWT:       :Wait for acknowledge (as
                           :before)
         MOV S2, #01H      :Leave acknowledge mode
         MOV S0,A          :Read data from slave to S0
WAIT:    MOV A,S1          :Test for byte received by
                           :testing S1 PIN bit
         JB4 Wait          :Wait until PIN received
         MOV S1, #0A9H     :Set Bit Counter to 1 and
                           :become a receiver (A9 =
                           :Mst, Rec, Bus Busy, Bit Counter =
                           :0001)
         MOV S0,A          :Move data to accumulator and
                           :clock out a negative
                           :acknowledge
         MOV S1, #0D8H     :Generate Stop Condition
```

Figure 5.

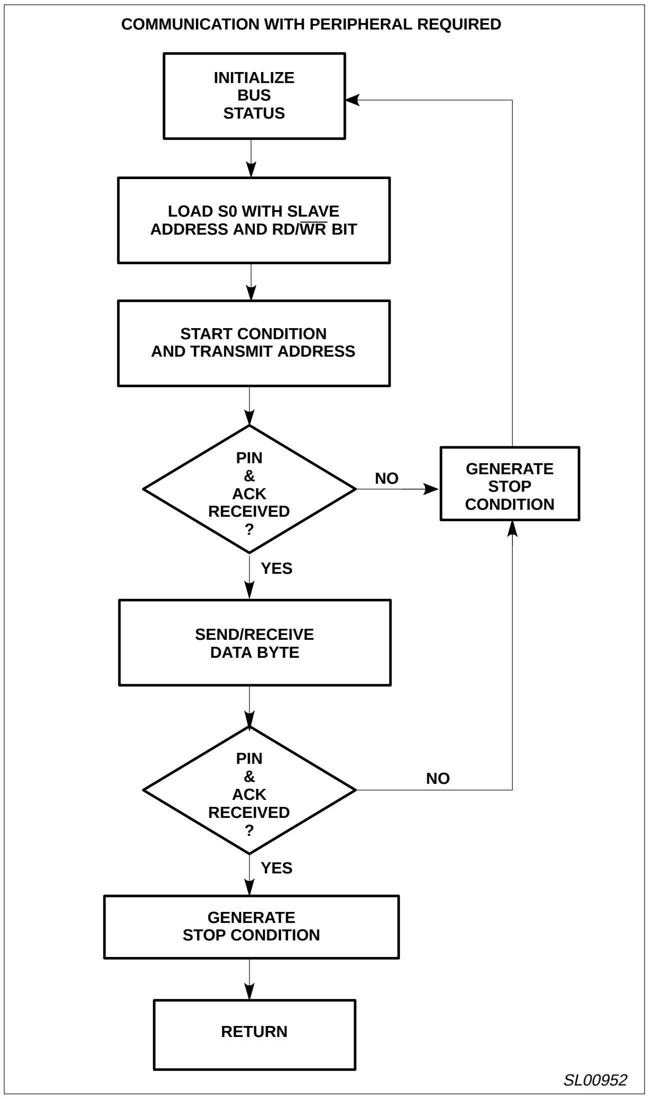


Figure 6. Flowchart for Reading/Writing One Byte to an I^{2}C Peripheral; Single-Master, Single-Address Slave

```
            MOV S1, #18H        :Initialize bus-status register
                                :Master, Transmitter,
                                :Bus-not-Busy, Enable SIO,
            MOV S0, #0A0H       :Load S0 with RAM's chip
                                :address
            MOV S1, #0F8H       :Start cond. and transmit
                                :address
            CALL ACKWT          :Wait until address received
            MOV A, #00H         :Set up for transmitting RAM
                                :location address
            MOV S0,A            :Transmit first RAM address
            CALL ACKWT          :Wait
            MOV S1, #18H        :Set up for a repeated Start
                                :condition
            MOV A, #0A1H        :Get RAM chip address & RD bit
            MOV SO, A           :Send out to bus
            MOV S1, #0F8H       :preceded by repeated Start

            CALL ACKWT          :Wait
            MOV A,S0            :First data byte to S0
            CALL ACKWT          :Wait
            MOV A,S0            :Second data byte to S0
                                :And First data byte to Acc.
            CALL ACKWT          :Wait
            MOV R0,A            :Save first byte in R0
            MOV A,S0            :Third data byte to S0
                                :and second data byte to Acc.
            CALL ACKWT          :Wait
            MOV R1,A            :Save second data byte
                                :in R1
            MOV S2, #01H        :Leave ack. mode
                                :Bit Counter=001 for neg ack.
            MOV A,S0            :Third data byte to acc
                                :negative ack. generated
            MOV R2,A            :Save third data byte in R2
WAIT1:      MOV A,S1            :Get bus status
            JB4 WAIT1           :Wait until transfer complete
            MOV S1, #0D8H       :Stop condition
            MOV S2, #41H        :Restore acknowledge mode
```

SL00953

Figure 7.

ASSIGNED I^{2}C BUS ADDRESSES

PART NUMBER	FUNCTION	I^{2}C ADDRESS						
		A6	A5	A4	A3	A2	A1	A0
—	General call address	0	0	0	0	0	0	0
—	Reserved addresses	0	0	0	0	X	X	X
PCD3311/12	Tone generator DTMF/modem/musical	0	1	0	0	1	0	A
PCF8200	Voice synthesizer (male or female)	0	0	1	0	0	0	0
PCF8566	96-segment LCD driver 1:1–1:4 Mux	0	1	1	1	1	1	A
PCF8568	LCD row driver for dot matrix displays	0	1	1	1	1	0	A
PCF8569	Column driver for dot matrix displays	0	1	1	1	1	0	A
PCF8570/71	256 × 8, 128 × 8 static RAM	1	0	1	0	A	A	A
PCF8570C	256 × 8 static RAM	1	0	1	1	A	A	A
PCF8573	Clock/calendar	1	1	0	1	0	A	A
PCF8574	I^{2}C bus to 8-bit bus converter	0	1	0	0	A	A	A
PCF8574A	I^{2}C bus to 8-bit bus converter	0	1	1	1	A	A	A
PCF8576	160-segment LCD driver 1:1–1:4 Mux	0	1	1	1	0	0	A
PCF8577	64-segment LCD driver 1:1–1:2 Mux	0	1	1	1	0	1	0
PCF8577A	64-segment LCD driver 1:1–1:2 Mux	0	1	1	1	0	1	1
PCF8578	Row/column LCD dot-matrix driver	0	1	1	1	1	0	A
PCF8579	Row/column LCD dot-matrix driver	0	1	1	1	1	0	A
PCF8581	128-byte EEPROM	1	0	1	0	A	A	A
PCF8582	256 × 8 EEPROM	1	0	1	0	A	A	A
PCF8583	256 × 8 RAM with clock/calendar	1	0	1	0	0	0	A
PCF8591	4-channel, 8-bit A/D plus 8-bit D/A	1	0	0	1	A	A	A
PCF8594	512-byte EEPROM	1	0	1	0	A	A	A
SAA1064	4-digit LED driver	0	1	1	1	0	A	A
SAA1136	PCM-Audio indent-word interface	0	0	1	1	1	1	0
SAA1300	5-bit high current driver	0	1	0	0	0	A	A
SAA5243/45	Enhanced teletext circuit	0	0	1	0	0	0	1
SAA7191	S-VHS digital multistandard decoder “square pixel”	1	0	0	0	1	A	1
SAA7192	Digital color space converter	1	1	1	0	0	0	A
SAA7199	Digital encoder	1	0	1	1	0	0	0
SAA9020	Field memory controller	0	0	1	0	1	A	A
SAA9051	Digital multi-standard TV decoder	1	0	0	0	1	0	1
SAA9068	(PIPCO) Picture-in-picture controller	0	0	1	0	0	1	A
SAB3035/36/37	(CITAC) CPU interface for tuning and control	1	1	0	0	0	A	A
SAF1135	Data line decoder	0	0	1	0	0	A	A
TDA4670	Picture signal improvement circuit	1	0	0	0	1	0	0
TDA4680	Video processor	1	0	0	0	1	0	0
TDA8421	Hi-fi stereo audio processor	1	0	0	0	0	0	A
TDA8425	Audio processor w/loudspeaker channel	1	0	0	0	0	0	1
TDA8440	Switch for CTV receivers	1	0	0	1	A	A	A
TDA8442	Interface for color decoders	1	0	0	0	1	0	0
TDA8443	YUV/RGB interface circuit	1	1	0	1	A	A	A
TDA8444	Octuple 6-bit DAC	0	1	0	0	A	A	A
TDA8461	PAL/NTSC color decoder	1	0	0	0	1	0	A
TEA6100	FM/IF and tuning interface	1	1	0	0	0	0	1
TEA6300/6310T	Sound fader control circuit	1	0	0	0	0	0	0
TSA5511/12/14	PLL frequency synthesizer for TV	1	1	0	0	0	A	A
TSA6057	Radio tuning PLL frequency synthesizer	1	1	0	0	0	1	A
UMF1009	Frequency synthesizer	1	1	0	0	0	A	A
—	Reserved addresses	1	1	1	1	X	X	X

X = Don’t care.
A = Can be connected high or low by the user.