INTEGRATED CIRCUITS

APPLICATION NOTE

ABSTRACT
Presents short and simple I^{2}C software routines that support only slave (rather than master or master & slave) operation and an ASM demonstration program. The slave-only software in this app note complements the master mode software presented in AN464, *Using the 87LPC76X microcontroller as an I^{2}C bus master*.

AN463
I^{2}C slave routines for the 87LPC76X

Author: Bill Houghton

2000 Oct 10
Supersedes data of 2000 Jan 11

Philips Semiconductors

PHILIPS

I^{2}C slave routines for the 87LPC76X AN463

Author: Bill Houghton

The 87LPC76X Microcontroller combines in a small package the benefits of a high-performance microcontroller with on-board hardware supporting the Inter-Integrated Circuit (I^{2}C) bus interface.

The 87LPC76X can be programmed both as an I^{2}C bus master, a slave, or both. An overview of the I^{2}C bus and description of the bus support hardware in the 87LPC76X microcontrollers appears in application note AN464, *Using the 87LPC76X Microcontroller as an I^{2}C Bus Master*. That application note includes a programming example, demonstrating a bus-master code. Here we show an example of programming the microcontroller as an I^{2}C slave.

The code listing demonstrates communications routines for the 87LPC76X as a slave on the I^{2}C bus. It compliments the program in AN464 which demonstrates the 87LPC76X as an I^{2}C bus master. One may demonstrate two 87LPC76X devices communicating with each other on the I^{2}C bus, using the AN464 code in one, and the program presented here in the other. The examples presented here and in AN464 allow the 87LPC76X to be either a master or a slave, but not both. Switching between master and slave roles in a multimaster environment is described in application note AN435.

The software for a slave on the bus is relatively simple, as the processor plays a relatively passive role. It does not initiate bus transfers on its own, but responds to a master initiating the communications. This is true whether the slave receives or transmits data—transmission takes place only as a response to a bus master's request. The slave does not have to worry about arbitration or about devices which do not acknowledge their address. As the slave is not supposed to take control of the bus, we do not demand it to resolve bus exceptions or "hangups". If the bus becomes inactive the processor simply withdraws, not interfering with the master (or masters) on the bus which should (hopefully) try to resolve the situation.

The 87LPC76X has a single bit I^{2}C hardware interface where the registers may directly affect the levels on the bus, and the software interacting with the hardware registers takes part in the protocol implementation. The hardware and the low level routines dealing with the registers are tightly coupled. We repeat here the warning from the 87LPC76X bus-master application note: one should take extra care if trying to modify these lower level routines.

The service routine for the I^{2}C slave is interrupt driven per message. This allows for master communication requests which are not synchronized with the application program running on the slave. It is possible to write simple slave application programs which will not be interrupt driven, taking care not to lose master transmissions while doing something else, but the user should be discouraged from doing so. As the slave should respond to asynchronous requests of masters on the bus, an interrupt driven service routine makes sense—and, as the code demonstrates, is simple to implement.

DEMONSTRATION CODE

The main program operation, intended for demonstration only, is simple. There are two data buffers, one for data reception and one for data transmission. When new data has been received from the I^{2}C bus into the receive buffer, the program writes it into the transmit buffer. The first byte of received data is copied to Port 1. When a bus master requests to read data, Port 0 will be returned for the first byte of requested data, while the remaining bytes will come from the transmit buffer. This allows for simple testing of a master and slave system by having the master compare data received to data sent. This scheme also allows the 87LPC76X to be used as a one-byte I^{2}C I/O port.

The program begins at address 0, where the microprocessor begins execution after a hardware reset. This location contains a jump instruction to the main program, which starts at the label Reset (towards the end of the listing). Upon reset, the program initializes the stack pointer, the I^{2}C address of the slave processor (MyAddr) and clears the data buffers and software flags. In this program the receive and transmit buffers are each eight bytes long—the maximum number of bytes is defined by the label MaxBytes. One may easily change the program to handle longer messages by changing the value of MaxBytes and allocating more data memory to the buffers.

The I^{2}C interface is configured to operate as a slave by setting the msb of register I^{2}CFG. This is done simultaneously with loading the appropriate value of CTVAL—bits CT0 and CT1, which are determined by the frequency of the microprocessor's crystal. The interface hardware is explicitly instructed to get into the slave idle mode by setting the appropriate bit in the I2CON register. Timer I, which operates as a "watchdog" timer detecting bus hangups, is activated and its interrupts are enabled.

After the initialization, the program gets to the label MainLoop. Most of the time the program will "hang" in a wait loop at this label, simply waiting for an I^{2}C interrupt to occur. When there is an I^{2}C bus request there will be an interrupt, the service routine will be executed and we shall return to the MainLoop label. If the service routine receives new data, it sets a flag, DatFlag, signalling that data has been updated. This flag will allow us to leave the MainLoop label, and execute a short routine copying the updated input buffer to the output (transmit) buffer.

If a new bus interrupt comes before overwriting of the old read buffer data is completed, and an undesirable "mix" of old and new data might occur. This type of situation is avoided by disabling the I^{2}C interrupts (clearing the IE2 bit in the Interrupt Enable 1 Register) just before copying the data to the transmit buffer, and re-enabling the interrupts when the copy operation is completed.

When the copy routine is completed the DatFlag is cleared and we jump back to MainLoop, waiting for the next interrupt to occur. If the interrupt is for data transmission the service routine will not set DatFlag, and upon return we shall remain at the MainLoop label.

THE INTERRUPT SERVICE ROUTINE

The service routine is interrupt driven with respect to the start of each I^{2}C frame, but within each frame the interaction with the hardware is based on polling. An occurrence of a Start on the bus will cause an interrupt that will initiate the service routine which starts at address 33H. After saving registers, all interrupts except the I^{2}C interrupt itself are enabled, as we want to allow response to other interrupts during the routine. The philosophy behind this is that the I^{2}C may be a lower priority than some other operations in the system. Since the I^{2}C hardware will stretch the clock until the program responds, an interrupt of reasonable duration will not have a harmful effect on the data transfer.

Since we intend to react to the I^{2}C hardware by polling the ATN flag in wait loops, we do not want the expected changes on the bus to take us again to the beginning of the routine. Therefore, the EI2 flag is cleared, masking further I^{2}C interrupts even when interrupts are re-enabled (by the ACALL to a RETI instruction).

At the label Slave, the routine starts receiving the address on the bus. Each new address bit is read after a software wait loop detects that the ATN flag is set by the hardware. Note that with the single bit implementation of the I^{2}C port, the software must closely support the hardware: for example, we need to explicitly clear the Start status before we enter a wait loop for the next bit. If the software does not clear the Start flag, the hardware will stretch the low period of the clock (SCL line) on the bus—and the first address bit will simply not occur. (Such a state will not go on forever—eventually the processor will release the bus as a result of a Timer I timeout.)

Reception of the eight bits of Address + R/W is completed using part of the receive byte subroutines. The address received is compared to MyAddr, the address of this specific slave. If the address is different the processor goes idle and leaves the service routine. If the message is intended for this processor (received address matches MyAddr) the Read/Write bit is tested, and the program jumps to the appropriate labels. When the R/W bit is low the master requests a Write—and this slave should receive the data written into it. When the R/W bit is high the master is requesting a Read and this slave should transmit the data (at code label Read).

For “Master Write” we send an acknowledge for the address byte and proceed with receiving the data bytes, responding with an acknowledge for each and transferring them into the receive buffer. For long messages, when the buffer is full (we have received MaxByte bytes) we read from the bus one additional byte and then send a negative acknowledge, letting the master know it should stop sending us data. Then we set DatFlag to signal the mainline program that new data has been received, and jump to MsgEnd. At the MsgEnd label we wait for the next Stop or Repeated Start. On a Stop we resume the idle mode (GoIdle) and return from the service routine. On a Restart the slave process starts again with reception of the new address at the label Slave.

If the message is short enough so that the receive buffer is not filled up, the RcvByte subroutine (called after WrtLoop) will return due to the Stop condition, DRDY will not be set, and we shall exit the loop via label WLEx—setting the DatFlag and proceeding to MsgEnd.

For “Master Read” the transmit buffer is sent on the bus byte by byte in the RdLoop, using the XmitByte subroutine. We exit the loop when all the buffer is transmitted, or the Master does not respond with an acknowledge. Note that lack of acknowledgement for slave transmission does not necessarily indicate a problem or that the receiving master is busy. This could very well be a normal operation of the protocol, which defines that a receiving master signals the transmitting slave to end its message by explicitly transmitting a negative acknowledge as a response to the last byte the master is interested in. The protocol does not include inherent means for specifying in advance the length of a requested message.

SUBROUTINES

The lower level subroutines closely interact with the hardware and the activity on the bus. The XmitByte subroutine transmits one byte and receives the acknowledge bit that comes in response. The byte receive routine, which one may use from different entry points, receives a data or an address byte, and takes care of acknowledgements. When a Start or Stop is detected the subroutine returns immediately—the calling routine is expected to check the flags to determine whether a whole byte has been received (DRDY will be set), or a Start or a Stop condition has occurred.

Close inspection of RcvByte code shows that a total of nine bits are being read off the bus. The first bit does not belong to the received byte, but is the acknowledge this processor sent in response of the former byte or address. Reading the Ack bit from the I2DAT register clears the Transmit Active state and DRDY, thus releasing SCL and allowing the bus activity to proceed to the next data bit. Upon return the Ack bit is left in the Carry flag, and the actual data byte received is returned in the Acc register.

Upon Timer I interrupt code execution commences at address 73H, where there is a jump to the service routine TimerI. This interrupt is caused by the watchdog timer, as a result of an I^{2}C bus that is “hanging” without activity in the middle of a transmission for too long a period of time. The slave simply clears the bus interface, and starts all over again at the label Reset.

```
;*****************************************************************************

;                    I2C Slave Routines for the 87LPC764

; This program demonstrates I2C slave functions for the 87LCP764. It is a
; modified version of code published for the 8xC751/752 in AN430.

; The program uses separate transmit and receive data buffers that are each
; eight bytes deep. The sample main program copies received data to the
; transmit buffer such that transmitted data can be read back by a bus master.

; Buffer address 0 is mapped to port 1, such that an I2C write will affect
; the port outputs, except for the I2C pins P1.6 & P1.7. An I2C read will return
; port 0 pin data. The code will accept only eight data bytes in any one I2C
; transmission, additional bytes will not be acknowledged. Similarly, only eight
; data bytes may be read in any one I2C transmission. This program does not
; support subaddressing for buffer access.

;*****************************************************************************

; Notes on 87LPC764 I2C differences:
;     – I2C interrupt vector address.
;     – Timer I interrupt vector address.
;     – IEN0 SFR name (IE on 751) and addition of IEN1.
;     – I2C interrupt enable location (now in IEN1 and a different bit).
;     – I2C SFR addresses (altered by inclusion of the MOD764 file).


$mod764
$debug


;*****************************************************************************

; Value definitions.

CTVAL      equ     02h                  ; CT1, CT0 bit values for I2C.
MaxBytes   equ     8                    ; Max # of bytes to be sent or recvd.

SlvAdr     equ     7Eh                  ; 7Eh is the keypad on I2C demo bd.
;SlvAdr    equ     76h                  ; 76h is the LED display on I2C demo bd.

; Masks for I2CFG bits.

BTIR       equ     10h                  ; Mask for TIRUN bit.
BSLAV      equ     80h                  ; Mask for Slave Enable bit.

; Masks for I2CON bits.

BCXA       equ     80h                  ; Mask for CXA bit.
BIDLE      equ     40h                  ; Mask for IDLE bit.
BCDR       equ     20h                  ; Mask for CDR bit.
BCARL      equ     10h                  ; Mask for CARL bit.
BCSTR      equ     08h                  ; Mask for CSTR bit.
BCSTP      equ     04h                  ; Mask for CSTP bit.

; RAM locations used by I2C routines.

RcvDat     data    10h                  ; I2C receive data buffer (8 bytes).
                                        ;   addresses 10h through 17h.

XmtDat     data    18h                  ; I2C transmit data buffer (8 bytes).
                                        ;   addresses 18h through 1Fh.

Flags      data    20h                  ; I2C software status flags.
NoAck      bit     Flags.7              ; Holds negative acknowledge flag.
DatFlag    bit     Flags.6              ; Tells whether an I2C write operation
                                        ;   has occurred.
```

```
BitCnt     data    21h                 ; I2C bit counter.
ByteCnt    data    22h                 ; Send/receive byte counter.
MyAddr     data    24h                 ; Holds address of THIS slave.

AdrRcvd    data    25h                 ; Holds received slave address + R/W.
RWFlag     bit     AdrRcvd.0           ; Slave read/write flag.

;*****************************************************************************
;                                Begin Code
;*****************************************************************************

; Reset and interrupt vectors.

           ajmp    Reset               ; Reset vector at address 0.

; I2C interrupt is used to detect a start while the slave is idle.

           org     33h                 ; I2C interrupt.
           push    psw                 ; Save status.
           push    acc                 ; Save accumulator.
           clr     ei2                 ; Disable I2C interrupt.
           acall   ClrInt              ; Re–enable other interrupts.
           ajmp    Slave

; Timer I timeout interrupt service routine.

           org     0073h               ; Timer I interrupt address.
TimerI:    setb    CLRTI               ; Clear timer I interrupt.
           mov     I2CFC,#0            ; Turn off I2C.
           mov     I2CON,#BCXA+BCDR+BCARL+BCSTR+BCSTP   ; Reset I2C flags.
           clr     TIRUN
           acall   ClrInt              ; Clear interrupt pending.
           ajmp    Reset               ; Re–start.
ClrInt:    reti
           org     0100h

;*****************************************************************************
;                    Main Transmit and Receive Routines
;*****************************************************************************

Slave:     mov     I2CON,#BCARL+BCSTP+BCSTR+BCXA ; Clear start status.
           jnb     ATN,$               ; Wait for next data bit.
           mov     BitCnt,#7           ; Set bit count.

           acall   RcvB2               ; Get remainder of slave address.
           mov     AdrRcvd,A           ; Save received address + R/W bit.
           clr     acc.0
           cjne    A,MyAddr,GoIdle     ; Enter idle mode if not our address.

           jb      RWFlag,Read         ; Read or Write?
           mov     R0,#RcvDat          ; Set up receive buffer pointer.
           mov     ByteCnt,#MaxBytes   ; Max 4 bytes can be received.

WrtLoop:
           acall   SendAck             ; Send acknowledge.
           acall   RcvByte             ; Get data byte from master.
           jnb     DRDY,WLEx           ; Must be end of frame?
           mov     @R0,A               ; Save data.
           inc     R0                  ; Advance buffer pointer.
           djnz    ByteCnt,WrtLoop     ; Back to receive if buffer not full.
           acall   SendAck             ; Send acknowledge.
           acall   RcvByte             ; Get, but do not store add’l data.
           mov     I2DAT,#80h          ; Send negative acknowledge.
           jnb     ATN,$               ; Wait for acknowledge sent.
WLEx:      setb    DatFlag             ; Flag main that data has been received.
           sjmp    MsgEnd              ; Buffer full, enter idle mode.
```

```
Read:      mov     R0,#XmtDat          ; Set up transmit buffer pointer.
           mov     ByteCnt,#MaxBytes   ; Max bytes to be sent.
           acall   SendAck             ; Send address acknowledge.

RdLoop:    mov     A,@R0               ; Get data byte from buffer.
           cjne    R0,#XmtDat,RdL1     ; Return port 1 value instead of buffer
           mov     A,P0                ;   data if this is buffer address 0.
           MOV     A,#05h              ; Debug send fixed data.

RdL1:      inc     R0,                 ; Advance buffer pointer.
           acall   XmitByte            ; Send data byte.
           jb      NoAck,RLEx          ; Exit if NAK.
           djnz    ByteCnt,RdLoop      ; Back if more data requested & avail.

RLEx:      sjmp    MsgEnd              ; Done, enter idle mode.

MsgEnd:    jnb     ATN,$               ; Wait for stop or repeated start.
           JB      STR,Slave           ; If repeated start, go to slave mode,
                                       ;   else enter idle mode.

GoIdle:    mov     I2CON,#BCSTP+BCXA+BCDR+BCARL+BIDLE ; Enter slave idle mode.
           pop     ACC                 ; Restore accumulator.
           pop     PSW                 ; Restore status.
           setb    EI2                 ; Re–enable I2C interrupts.
           ret

;*****************************************************************************
;                                Subroutines
;*****************************************************************************

; Byte transmit routine.
;   Enter with data in ACC.

XmitByte:  mov     BitCnt,#8           ; Set 8 bits of data count.
XmBit:     mov     I2DAT,A             ; Send this bit.
           rl      A                   ; Get next bit.
           jnb     ATN,$               ; Wait for bit sent.
           djnz    BitCnt,XmBit        ; Repeat until all bits sent.
           mov     I2CON,#BCDR+BCXA    ; Switch to receive mode.
           jnb     ATN,$               ; Wait for acknowledge bit.
           mov     Flags,I2DAT         ; Save acknowledge bit.
           ret

; Byte receive routines.
;   SendAck : sends an I2C acknowledge.
;   RcvByte : receives a byte of data.
;   RcvB2   : receives a partial byte of I2C data, used to allow reception of
;                7 bits of slave address information.
;   Data is returned in the ACC.

SendAck:   mov     I2DAT,#0            ; Send receive acknowledge.
           jnb     ATN,$               ; Wait for acknowledge sent.
           ret

RcvByte:   mov     BitCnt,#8           ; Set bit count.
RcvB2:     clr     A                   ; Init received byte to 0.
RBit:      orl     A,I2DAT             ; Get bit, clear ATN.
           rl      A                   ; Shift data.
           jnb     ATN,$               ; Wait for next bit.
           jnb     DRDY,RBEx           ; Exit if not a data bit.
           djnz    BitCnt,RBit         ; Repeat until 7 bits are in.
           mov     C,RDAT              ; Get last bit, don’t clear ATN.
           rlc     A                   ; Form full data byte.
RBEx:      ret
```

```
;*****************************************************************************
;                                Main Program
;*****************************************************************************

Reset:     mov     SP,#2Fh              ; Set stack location.
           mov     R0,#RcvDat           ; Set up pointer to data area.
           mov     R1,#2*MaxBytes       ; Set up buffer length counter.
RLoop:     mov     @R0,#0               ; Clear buffer memory.
           inc     R0                   ; Advance to next buffer position.
           djnz    R1,RLoop             ; Repeat until done.

           mov     AdrRcvd,#0
           mov     MyAddr,#SlvAdr       ; Set our slave address.
           mov     Flags,#0             ; Clear system flags.
           setb    EI2                  ; Enable I2C interrupt.
           setb    ETI                  ; Enable Timer I interrupt.
           setb    EA                   ; Enable interrupt system.
           mov     I2CFG,#BSLAV+CTVAL   ; Enable slave functions.
           mov     I2CON,#BCSTR+BCSTP+BXCA+BCDR+BCARL_BIDLE ; Put slave into idle mode.
           setb    TIRUN                ; Turn on timer I.

; This sample mainline program copies the first received bytes to Port 0
;   whenever there is an I2C write operation. It Also copies the rest of
;   the input buffer to the output buffer at the same time, acting like a
;   small memory device.

MainLoop:  jnb     DatFlag,$            ; Wait for data sent from I2C.

; ***      mov     pcon,#01h            ; Enter Idle Mode.

           clr     EA                   ; Turn off interrupts during data move.

           mov     A,RcvDat+1           ; Get first data byte (second buffer location).
           orl     a,#0Ch               ; Mask off I2C pins to prevent disaster.
           mov     P1,A                 ; Store data to port 1.

           mov     R0,#RcvDat           ; Set input buffer start pointer.
           mov     R1,#XmtDat           ; Set output buffer start pointer.
           mov     R2,#MaxBytes         ; Set buffer length counter.
ML2:       mov     A,@R0                ; Get data from input buffer.
           mov     @R1,A                ; Store data in output buffer.
           inc     R1                   ; Increment input buffer pointer.
           inc     R0                   ; Increment output buffer pointer.
           djnz    R2,ML2               ; Repeat until entire buffer is updated.
           clr     DatFlag              ; Clear I2C transmission flag.

           setb    EA                   ; Data move done, re–enable interrupts.
           sjmp    MainLoop             ; Wait for next I2C transmission.

           org     0fd00h               ; EPROM Configuration Byte (UCFG1)
           db      038h                 ; WDT off, RST enabled, port RST high,
                                        ;   BO=2.5V, CLK / 1, osc = high freq.

           end
```

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

Date of release: 10-00

Document order number: 9397 750 07607

Let's make things better.

Philips Semiconductors

PHILIPS