Summary

The following explains what an I2C bus protocol is. Included is the software program showing how the I2C bus communicates with a Signetics PCR8573 clock/calendar chip.

Description

The I2C bus protocol is a two-way, two-wire communication format used by the 8088 microcontroller family and many peripheral devices. One of the two lines (the SDA line) is a bi-directional serial data line and the other line (the SCL line) is a master controlled clock line. It is usually necessary to pull both lines high with resistors. A start signal (a falling SDA edge while SCL is high) initiates the serial data transmission, and after each transmission of eight bits the receiver sends an acknowledge bit to the transmitter. The stop signal is a rising edge of SDA while SCL is high. In general the first byte after the start signal is the slave address of the device being accessed by the master. The second word is either a mode word that comprises an instruction, or a data byte from a slave device. Subsequent words are data bytes.

The following application uses pins PA6 and PA7 of Port A on the QED board as I2C bus masters to communicate with a Signetics PCF8573 clock/calendar chip. Several important operations include reading and writing the clock/calendar's time and alarm registers, and setting and resetting various flags. High level routines such as RESET.CLOCK which sets the clock's time, and MINUTE.ALARM which sets the clock/calendar's COMP flag after a minute has elapsed, demonstrate useful functions that make use of lower level driver routines that send and receive bytes and acknowledge signals. Many of the driver routines, such as GET.WORD, SEND.WORD, and INIT.COM can be used in communicating with any device that employs the I2C protocol.

```
\ I2C DRIVER CODE AND CLOCK/CALENDAR APPLICATION    --     MOSAIC INDUSTRIES
7/5/94

HEX
0000 4 DP X!                            \ Set 32K definitions area on page 4
0000 5 NP X!                            \ Set 32K name area on page 5.
3000 F VP X!                            \ 4K variable area on page 15.
4000 F 7FFF F IS.HEAP                   \ 16K heap on page 15.

ANEW I2CDRIVER
6 WIDTH !
0 TRACE !
HEX
0080  CONSTANT SDA.MASK     \ P7 will be the data line.  SET.BITS and
                            \ CLEAR.BITS only use lower byte, but routines
                            \ like GET.ACKNOWLEDGE do an AND followed
                            \ by a BOOLEAN, so the high byte has to be 00.
0040  CONSTANT SCL.MASK     \ P6 will be the clock line.
PORTA    2CONSTANT   I2CPORT           \ PORTA will be the I2C port.
PORTA.DIRECTION 2CONSTANT I2CPORT.DIRECTION \ PORTA.DIRECTION will be dir. addr.
0  CONSTANT NOERROR              \ No error flag signal.
-1 CONSTANT ISERROR              \ Error flag signal.
```

```
: MASTER.QUIET.CONFIG      ( -- )\ Configures SDA and SCL as high master outputs.
   SDA.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SDA as output.
   SCL.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SCL as output.
   SDA.MASK I2CPORT  SET.BITS                \ Set data line high.
   SCL.MASK I2CPORT  SET.BITS                \ Set clock line high.
;

: SEND.BIT  ( bit -- )           \ First, bit is converted to a flag, and then
                                 \ a 0 or 1 is sent out the SDA line.
                                 \ *Assumes SDA is config'd as output.*
                                 \ SDA is output ("master xmitr") at finish.
   BOOLEAN                       \ Convert bit to flag.
   IF
      SDA.MASK I2CPORT     SET.BITS    \ Set SDA high if bit was high.
   ELSE
      SDA.MASK I2CPORT     CLEAR.BITS  \ Else set SDA low.
   ENDIF
   SCL.MASK I2CPORT  SET.BITS          \ Pulse clock on.
                                 \ Reminder: SCL must be high for at least 4us.
   SCL.MASK I2CPORT CLEAR.BITS         \ Turn clock off.
   ;

1 CONSTANT LSB.MASK
: READ.BIT  ( -- [0 OR 1]) \ Master, in receiver mode, reads a bit from
                           \ the bus.  Assumes SDA config. as input.
   SCL.MASK I2CPORT  SET.BITS          \ Pulse clock on.
   I2CPORT  C@                         \ Fetch the byte on the port.
   SCL.MASK I2CPORT  CLEAR.BITS        \ Turn off clock.
   SDA.MASK AND                        \ Isolate the bit of interest (the SDA bit).
   BOOLEAN LSB.MASK AND    \ Shift bit back down to proper LSB loc.
;

: SEND.ACKNOWLEDGE   ( -- )      \ Causes master to send out acknowlege pulse.
                                 \ SDA remains an input (for master) at finish
                                 \ and SCL is low.
   SDA.MASK I2CPORT  CLEAR.BITS              \ Send out low acknowledge pulse.
   SDA.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SDA as output.
   SDA.MASK I2CPORT  CLEAR.BITS              \ Send out low acknowledge pulse.
   SCL.MASK I2CPORT  SET.BITS                \ Pulse clock on.
                                 \ Reminder: SCL must be high for at least 4us.
   SCL.MASK I2CPORT  CLEAR.BITS              \ Turn clock off.
   SDA.MASK I2CPORT  SET.BITS                \ Reset SDA to high.
   SDA.MASK I2CPORT.DIRECTION    CLEAR.BITS  \ Reconfig. SDA as input.
;
: GET.ACKNOWLEDGE    ( -- flag)  \ Retrieves ack. signal from slave receiver.
                                 \ SDA is high output at finish, SCL is low.
   SDA.MASK I2CPORT  SET.BITS    \ SDA goes high.
   SDA.MASK I2CPORT.DIRECTION CLEAR.BITS     \ Config. SDA as input.
                                 \ SDA should emerge high when reconfig'd
                                 \ as an output at finish.
   SCL.MASK I2CPORT  SET.BITS    \ Pulse clock on.
   I2CPORT  C@                   \ Fetch the byte on the port.
   SDA.MASK AND                  \ Isolate the bit of interest (the SDA bit).
   BOOLEAN                       \ convert value to flag and leave on stack.
   SCL.MASK I2CPORT  CLEAR.BITS  \ Turn clock off.
   SDA.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SDA as output.
;
```

```
1  CONSTANT SHIFT.RIGHT
7  CONSTANT BYTE.LENGTH

: SEND.WORD ( word -- error )     \ Configures SDA and SCL as outputs and sends
                                  \ an 8 bit word and gets and returns the ackn.
                                  \ signal. At finish, SCL is low and SDA is
                                  \ high output.
   LOCALS{ &word }
   SCL.MASK I2CPORT  CLEAR.BITS              \ Activate clock (set low).
   SDA.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SDA as output.
   BYTE.LENGTH
   FOR
      &word I  NEGATE      SCALE \ Shift right by I bits, so MSB goes out
                                 \ first.
      LSB.MASK AND               \ Get the bit to send.
      SEND.BIT
   NEXT
   GET.ACKNOWLEDGE          ( -- flag)
;

: GET.WORD  ( flag -- word | flag is order to send acknowledge)
                           \ Configures SDA as input and SCL as output
                           \ and pulls in a word from the slave xmtr.
                           \ Leaves master config'd as (i/p) receiver.
   LOCALS{ &ack.flag | &word }
   SCL.MASK I2CPORT  CLEAR.BITS              \ Activate clock (set low).
   SDA.MASK I2CPORT.DIRECTION    CLEAR.BITS  \ Configure SDA as input.
   0 TO &word                                \ Initialize &word to 0.
   BYTE.LENGTH
   FOR
      READ.BIT ( -- [0 or 1])
      I  SCALE                   \ Shift bit in starting with MSB.
      &word +                    \ Use &word as a sum register.
      TO &word
   NEXT
   &word                         \ Place &word on stack before exit.
   &ack.flag
   IF                            \ Acknowledge, if required.
      SEND.ACKNOWLEDGE
   ELSE                          \ Else no acknowledge: leave SDA config'd
                                 \ as input, and pull-up resistors will
                                 \ pull the bus high.
   ENDIF
;

: SEND.START   ( -- )                        \ Transmit start signal.
   SDA.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SDA as output.
   SCL.MASK I2CPORT.DIRECTION    SET.BITS    \ Config. SCL as output.
   SCL.MASK I2CPORT  SET.BITS                \ SCL must be high.
   SDA.MASK I2CPORT  CLEAR.BITS        \ Setting data line low is "start" sig.
;
```

```
: SEND.STOP ( -- )                \ Transmit stop signal.
   SCL.MASK I2CPORT  SET.BITS     \ Set clock to high.
   SDA.MASK I2CPORT  SET.BITS     \ Set data line high.
   SDA.MASK I2CPORT.DIRECTION     SET.BITS    \ Make sure SDA is an output.
;


00D0  CONSTANT DEV.1.ADDR         \ DEV.1.ADDR includes 0=r/w
1  CONSTANT DEV.1
: COMM.INIT ( device#\R/W -- error )   \ Addresses device, ret. 0 if no err.
                                       \ R/W can be 0 or 1, or a flag.
   LOCALS{ &r/w }
   &r/w  BOOLEAN     TO &r/w      \ Ensures &r/w is a flag
   \ CHECK.BUS                    \ Normally you would need to wait for the bus
                                  \ to clear before initiating communication.
   CASE                           \ Transform device# to device address.
      DEV.1 OF DEV.1.ADDR  NOERROR ENDOF
      CR ." Invalid device"
      DROP
      ISERROR
   ENDCASE
   IF                \ If error, set error flag before ending.
      ISERROR
   ELSE              \ Else go ahead and send the word.
      &r/w  -        ( device.address -- device.word )
                     \ The byte sent to the slave needs to have the last
                     \ bit signal read or write.  Since &r/w is a flag
                     \ (0 or -1) it is subtracted so that 1 is *added*
                     \ if the flag is set.
      SEND.START
      SEND.WORD      \ Sends device word, which is on the top of the stack.
      IF             \ If error ocurred in SEND.WORD,
         ISERROR     \ Put the error flag on the stack before ending
      ELSE
         NOERROR     \ Clear error flag if no error.
      ENDIF
   ENDIF
;

1  CONSTANT RD.MODE  \ RD.MODE is not a flag (i.e. it should not be -1)
0  CONSTANT WR.MODE
00 CONSTANT ACC.TIME.MODE.WORD   \ Mode word used to access time register.
04 CONSTANT ACC.ALRM.MODE.WORD   \ Mode word used to access alarm register.
: SET.CLK.REGISTER   ( months\days\minutes\hours\mode.word\device# -- error )
                     \ Writes to either the alarm register or the time
                     \ register, depending on the mode.word.
   WR.MODE           \ Set write mode.
   COMM.INIT
```

```
      IF                    \ If there is an error,
      ISERROR               \ Put error flag on the stack before finish.
   ELSE                     \ Else continue.
      SEND.WORD             \ The mode word is the next word on the stack.
      IF                    \ If error,
         ISERROR            \ Set error flag
      ELSE
         SEND.STOP
         &device.num
         RD.MODE
         COMM.INIT
         IF
            ISERROR
         ELSE
            ACK.SET.FLG          \ Flag tells GET.WORD to send ack. to slave xmtr.
            GET.WORD
            ACK.SET.FLG
            GET.WORD
            ACK.SET.FLG
            GET.WORD
            ACK.CLR.FLG
            GET.WORD
            NOERROR
         ENDIF
      ENDIF
   ENDIF
   SEND.STOP
;

20 CONSTANT RST.PRES.MODE.WORD
30 CONSTANT TIM.ADJS.MODE.WORD
40 CONSTANT RST.NODA.MODE.WORD
50 CONSTANT SET.NODA.MODE.WORD
60 CONSTANT CLR.COMP.MODE.WORD
: ONE.WORD.INSTR ( modeword\device# -- err)  \ Many different commands can
                    \ be issued with one mode word. ONE.WORD.INSTR
                    \ is a general Forth word that executes such
                    \ commands, such as clearing the COMP flag,
                    \ for example.
   WR.MODE          \ Set write mode.
   COMM.INIT
   IF               \ If there is an error,
      ISERROR       \ Put error flag on the stack before finish.
   ELSE             \ Else continue.
      SEND.WORD
      IF
         ISERROR
      ELSE
         NOERROR
      ENDIF
   ENDIF
   SEND.STOP
;
```

```
: SHOW.TIME ( hours\minutes\days\months\error -- ) \ Calendar outputs
                          \ are in binary coded decimal, so the desired
                          \ decimal time appears when they are displayed
                          \ in hex.
IF
   CR ." Error found."
ELSE
   CR ." Time is: "
   . ." months: "
   . ." days: "
   SWAP
   . ." hours: "
   . ." minutes"
ENDIF
;


MASTER.QUIET.CONFIG             \ Set up a quiet (non-busy) bus.

7  CONSTANT MONTHS
5  CONSTANT DAYS
31 CONSTANT MINUTES
11 CONSTANT HOURS
: RESET.CLOCK  ( -- error) \ Loads in MONTHS:DAYS:MINUTES:HOURS as the time.
MONTHS
DAYS
MINUTES
HOURS
ACC.TIME.MODE.WORD
DEV.1
SET.CLK.REGISTER
;

ANEW TST.ROUTINES

: REVERSE.STACK.TIME ( r1\r2\r3\r4 -- r4\r3\r2\r1 )
   SWAP
   ROT
   3 ROLL
;
```

```
1  CONSTANT MINUTES
: MINUTE.ALARM ( -- error)        \ Proper carries for incrementing
                                  \ time not implemented.
   CLR.COMP.MODE.WORD
   DEV.1
   ONE.WORD.INSTR                 \ Clear the COMP flag.
   IF
      ISERROR
   ELSE
      RST.PRES.MODE.WORD
      DEV.1
      ONE.WORD.INSTR              \ Reset the seconds.
      IF
         ISERROR
      ELSE
         ACC.TIME.MODE.WORD
         DEV.1
         GET.CLK.REGISTER         \ Get the time.
         IF
            ISERROR
            CR ." Cannot get time."
         ELSE
            ROT                   \
            MINUTES  +            \ Increment minutes counter
            -ROT                  \
            REVERSE.STACK.TIME
            ACC.ALRM.MODE.WORD
            DEV.1
            SET.CLK.REGISTER      \ Set the alarm.
         ENDIF
      ENDIF
   ENDIF
;
```


Mosaic Industries

5437 Central Ave Suite 1, Newark, CA 94560 Telephone: (510) 790-8222 Fax: (510) 790-0925