Interfacing the X9241 XDCPs to 8051 Microcontrollers

by Applications Staff, June 2000

The X9241 has a variety of different instructions that provide flexibility to the designer. Additionally, the nonvolatile nature of the device allows for stored wiper positions that can be retrieved after power cycles. The following code implements all of the available X9241 instructions using a standard bi-directional bus protocol. Although the routines occupy less than 300 bytes of program memory, designers who won't need to implement all of the X9241 instructions can shorten the code by removing any unnecessary routines. However, this will necessitate the reassembly of the code.

For those instructions which program the nonvolatile data registers (XFR_WCR, GXFR_WCR, and WRITE_DR), acknowledge polling has been implemented to determine an early completion of the internal write cycle. Although this is automatically handled by the routines, a word or two regarding the procedure is in order. After issuing a start condition, the master sends a slave address and receives an acknowledge. It then issues an instruction byte to the X9241 and again receives an acknowledge. If necessary, it now transmits the data byte and receives a final acknowledge. The master must then initiate a stop condition which will cause the X9241 to begin an internal write cycle. The X9241 pins go high impedance until this internal cycle is complete. The master can now begin acknowledge polling by successively sending start conditions followed by "dummy" instructions. When the X9241 finally answers with an acknowledge, the internal write cycle has been completed and the master must initiate a stop condition. After the next start condition, the X9241 is ready to receive further instructions.

In the code listing, an assumption was made that the code would execute upon a reset of the microcontroller. The code was also loaded into low memory, however this can be changed with an ORG assembler directive. A simple MAIN program to exercise these routines is included on the next page. In this listing, the commands cause an X9241 (at A3A2A1A0 = 0000) to be accessed and the WCR of XDCP #2 to be rewritten with the value 43 (for wiper tap position #43). Then a 15 pulse decrement of the wiper tap is initiated, causing the selected WCR to be reduced to the value 28 (for wiper tap position #28). The issuing of other commands follows the same general procedure.

In Figure 1, a representative hardware connection between the X9241 and an 8051 family microcontroller is shown. The pull-up resistors on the SDA and SCL lines are determined by the total capacitance of all of the devices connected to the bus, which is about 18pF in this case, however these may not be necessary since I/O port pins on 8051 family devices have internal pull-ups.

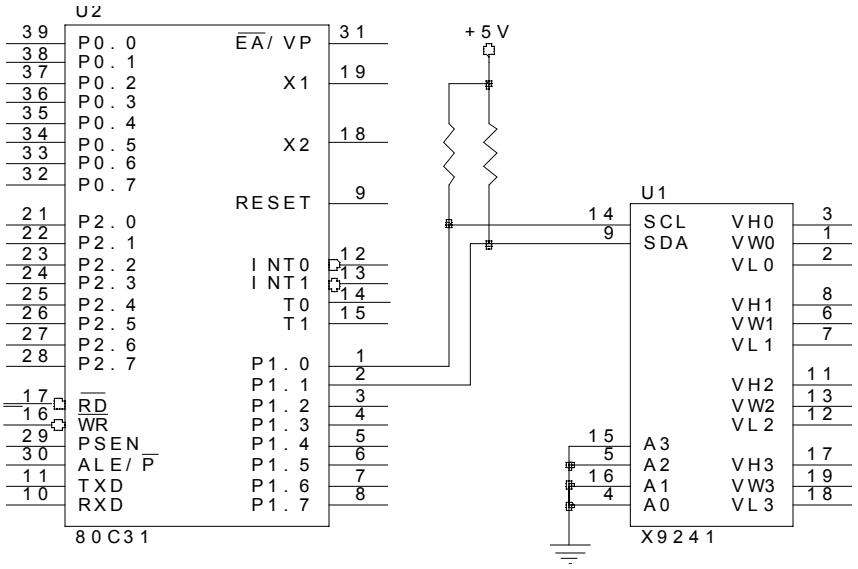

Figure 1. Typical connection between an 80C31 and an X9241 (with A3A2A1A0 = 0000)

Code Listing 1: Sample MAIN Code Listing for Using the Following Interface Routines

```
MAIN:     mov  ADDR_BYTE,#01010000b       ;* LOAD SLAVE ADDRESS BYTE
          mov ID,#00001000b               ;* LOAD ID BYTE (EEPOT #2)
          mov  COMMAND,#4                 ;* WRITE TO WCR
          mov  DATA_BYTE,#00101011b       ;* SET D5D4D3D2D1D0 = 101011
          call INTERPRET
          mov ID,#00001000b               ;* RELOAD ID BYTE (EEPOT #2)
          mov  PULSES,#00001111b          ;* DEC FOR 15 PULSES
          mov  COMMAND,#32                ;* INCREMENT/DECREMENT WIPER
          call INTERPRET
          •
          •
          •
```

Code Listing 2: 80C31 Microcontroller Routines for Manipulating AN X9241

```
                    1   ;*********************************************************
                    2   ;*
                    3   ;* 80C31 MICROCONTROLLER ROUTINES FOR MANIPULATING AN X9241
                    4   ;*                          QUAD XDCP
                    5   ;*
                    6   ;*                   (C) XICOR INC. 1993
                    7   ;*                                                   GHC IV
                    8   ;*********************************************************

 0090               9        SCL       bit   p1.0   ;* 80C31 PIN USED AS SCL
 0091               10       SDA       bit   p1.1   ;* 80C31 PIN USED AS SDA
 REG                11       TEMP      equ   r1     ;* SCRATCH REGISTER
 REG                12       COUNT     equ   r2     ;* LOOP COUNTING REGISTER
```

```
REG                 13        PULSES    equ   r3      ;* BITS -> DIR X###### (# = 1 or 0)
REG                 14        COMMAND   equ   r4      ;* INSTRUCTION (I.E. 0,4,8,12,16,...)
REG                 15        ID        equ   r5      ;* BITS -> 0 0 0 0 P1 P0 R1 R0
REG                 16        ADDR_BYTE equ   r6      ;* BITS -> 0 1 0 1 A3 A2 A1 A0
REG                 17        DATA_BYTE equ   r7      ;* BITS -> CM DW D5 D4 D3 D2 D1 D0

                    18     ;*******************************************************
                    19     ;*
                    20     ;* INSERT A "JUMP TO MAIN" INSTRUCTION INTO 80C31 RESET
                    21     ;* VECTOR POSITION
                    22     ;*
                    23     ;*******************************************************

0000                24           org    0000h   ;* RESET VECTOR HANDLER
                    25                          ;* AT THIS ADDRESS
0000 02 01 2C       26           jmp    MAIN

                    27     ;*******************************************************
                    28     ;*
                    29     ;* NAME: INTERPRET
                    30     ;* FUNCTION: DETERMINES WHICH X9241 INSTRUCTION IS ISSUED,
                    31     ;* THEN EXECUTES
                    32     ;* INPUTS: COMMAND
                    33     ;* OUTPUTS: NONE
                    34     ;* CALLS: READ WCR, READ_DR, WRITE_WCR, WRITE_DR, XFR_DR,
                    35     ;* XFR_WCR, GXFR_DR, GXFR_WCR,INC_WIPER
                    36     ;* AFFECTED: DPTR,A
                    37     ;*
                    38     ;*******************************************************

0003 90 00 08       39   INTERPRET:     mov  dptr,#FIRST   ;* JMP BASE ADDRESS
0006 EC             40                  mov  a,COMMAND     ;* JMP OFFSET
0007 73             41                  jmp  @a+dptr       ;* JUMP TO INSTRUCTION
                    42                                     ;* HANDLER
0008 12 00 2C       43   FIRST:         call  READ_WCR     ;* COMMAND #0
000B 22             44                  ret
000C 12 00 37       45                  call  WRITE_WCR    ;* COMMAND #4
000F 22             46                  ret
0010 12 00 42       47                  call  READ_DR      ;* COMMAND #8
0013 22             48                  ret
0014 12 00 4D       49                  call  WRITE_DR     ;* COMMAND #12
0017 22             50                  ret
0018 12 00 58       51                  call  XFR_DR       ;* COMMAND #16
001B 22             52                  ret
001C 12 00 63       53                  call  XFR_WCR      ;* COMMAND #20
001F 22             54                  ret
0020 12 00 6E       55                  call  GXFR_DR      ;* COMMAND #24
0023 22             56                  ret
0024 12 00 79       57                  call  GXFR_WCR     ;* COMMAND #28
0027 22             58                  ret
0028 12 00 84       59                  call  INC_WIPER    ;* COMMAND #32
002B 22             60                  ret

                    61    ;*******************************************************
                    62    ;*
```

```
                        63   ;* THE FOLLOWING ROUTINES HANDLE EACH X9241 INSTRUCTIONS.
                        64   ;* THESE ARE CALLED BY THE INTERPRET ROUTINE AND ARE
                        65   ;* STRAIGHT FORWARD
                        66   ;*
                        67   ;* READ_WCR - READS A WCR AND RETURNS ITS' VALUE IN
                        68   ;* DATA_BYTE
                        69   ;* WRITE_WCR - WRITES THE VALUE IN DATA_BYTE TO A WCR
                        70   ;* READ_DR - READS A DATA REGISTER AND RETURNS ITS' VALUE
                        71   ;* IN DATA_BYTE
                        72   ;* WRITE_DR - WRITES THE VALUE IN DATA_BYTE TO A DATA
                        73   ;* REGISTER
                        74   ;* XFR_DR - TRANSFERS THE VALUE IN A DATA REGISTER TO ITS'
                        75   ;* WCR
                        76   ;* XFR_WCR - TRANSFERS THE VALUE IN A WCR TO ONE OF ITS'
                        77   ;* DATA REGISTERS
                        78   ;* GXFR_DR - GLOBAL TRANSFER OF LIKE DATA REGISTERS TO
                        79   ;* THEIR WCRS
                        80   ;* GXFR_WCR - GLOBAL TRANSFER OF WCRS TO THEIR LIKE DATA
                        81   ;* REGISTERS
                        82   ;* INC_WIPER - SINGLE STEP INCREMENT/DECREMENT OF WIPER
                        83   ;* POSITION FOR WCR
                        84   ;*
                        85   ;* FUNCTION: APPENDS BITS P1,P0,R1,R0 TO THE APPROPRIATE
                        86   ;* INSTRUCTION CODE & PASSES THE INSTRUCTION BYTE TO THE
                        87   ;* INSTRUCTION GENERATOR
                        88   ;* INPUTS: ID
                        89   ;* OUTPUTS: NONE
                        90   ;* CALLS: INSTR_GEN
                        91   ;* AFFECTED: ID,A,DPTR
                        92   ;*
                        93   ;*********************************************************

002C ED                 94   READ_WCR:   mov  a,ID          ;* GET BITS P1 P0 X X
002D 44 90              95               orl  a,#090h       ;* APPEND TO READ_WCR
                        96                                  ;* INSTRUCTION CODE
002F FD                 97               mov  ID,a          ;* SAVE THE RESULT
0030 90 00 B2           98               mov  dptr,#CASE1   ;* JMP BASE ADDRESS FOR THIS
                       100                                  ;* INSTRUCTION


0033 12 00 8F          101               call  INSTR_GEN
0036 22                102               ret
0037 ED                103   WRITE_WCR:  mov   a,ID         ;* GET BITS P1 P0 X X
0038 44 A0             104               orl   a,#0A0h      ;* APPEND TO WRITE_WCR
                       105                                  ;* INSTRUCTION CODE
003A FD                106               mov   ID,a         ;* SAVE THE RESULT
003B 90 00 AB          107               mov   dptr,#CASE2  ;* JMP BASE ADDRESS FOR THIS
                       108                                  ;* INSTRUCTION
003E 12 00 8F          109               call  INSTR_GEN
0041 22                110               ret
0042 ED                111   READ_DR:    mov   a,ID         ;* GET BITS P1 P0 R1 R0
0043 44 B0             112               orl   a,#0B0h      ;* APPEND TO READ_DR
                       113                                  ;* INSTRUCTION CODE
0045 FD                114               mov  ID,a          ;* SAVE THE RESULT
0046 90 00 B2          115               mov  dptr,#CASE1   ;* JMP BASE ADDRESS FOR THIS
```

```
                  116                                   ;* INSTRUCTION
0049 12 00 8F     117               call INSTR_GEN
004C 22           118               ret
004D  ED          119  WRITE_DR:    mov  a,ID            ;* GET BITS P1 P0 R1 R0
004E 44 C0        120               orl  a,#0C0h         ;* APPEND TO WRITE_DR
                  121                                    ;* INSTRUCTION CODE
0050 FD           122               mov  ID, a           ;* SAVE THE RESULT
0051 90 00 B8     123               mov  dptr,#CASE3     ;* JMP BASE ADDRESS FOR THIS
                  124                                    ;* INSTRUCTION
0054 12 00 8F     125               call INSTR_GEN
0057 22           126               ret
                  127
0058 ED           128  XFR_DR:      mov  a,ID            ;* GET BITS P1 P0 R1 R0
0059 44 D0        127               orl  a,#0D0h         ;* APPEND TO XFR_DR
                  128                                    ;* INSTRUCTION CODE
005B FD           129               mov  ID, a           ;* SAVE THE RESULT
005C 90 00 A8     129               mov  dptr,#CASE4     ;* JMP BASE ADDRESS FOR THIS
                  130                                    ;* INSTRUCTION
005F 12 00 8F     131               call INSTR_GEN
0062 22           132               ret
0063 ED           133  XFR_WCR:     mov  a,ID            ;* GET BITS P1 P0 R1 R0
0064 44 E0        134               orl  a,#0E0h         ;* APPEND TO XFR_WCR
                  135                                    ;* INSTRUCTION CODE
0066 FD           136               mov  ID, a           ;* SAVE THE RESULT
0067 90 00 C5     137               mov  dptr,#CASE5     ;* JMP BASE ADDRESS FOR THIS
                  138                                    ;* INSTRUCTION
006A 12 00 8F     139               call INSTR_GEN
006D 22           140               ret
006E ED           141  GXFR_DR:     mov  a,ID            ;* GET BITS X X R1 R0
006F 44 10        142               orl  a,#010h         ;* APPEND TO GXFR_DR
                  143                                    ;* INSTRUCTION CODE
0071 FD           144               mov  ID, a           ;* SAVE THE RESULT
0072 90 00 A8     145               mov  dptr,#CASE4     ;* JMP BASE ADDRESS FOR THIS
                  146                                    ;* INSTRUCTION
0075 12 00 8F     147               call INSTR_GEN
0078 22           148               ret
0079 ED           149  GXFR_WCR:    mov  a,ID            ;* GET BITS X X R1 R0
007A 44 80        150               orl  a,#080h         ;* APPEND TO GXFR_WCR
                  151                                    ;* INSTRUCTION CODE
007C FD           152               mov  ID, a           ;* SAVE THE RESULT
007D 90 00 C5     153               mov  dptr,#CASE5     ;* JMP BASE ADDRESS FOR
                  154                                    ;* THIS INSTRUCTION
0080 12 00 8F     155               call INSTR_GEN
0083 22           156               ret
0084 ED           157  INC_WIPER:   mov  a,ID            ;* GET BITS P1 P0 X X
0085 44 20        158               orl  a,#020h         ;* APPEND TO INC_WIPER
                  159                                    ;* INSTRUCTION CODE
0087 FD           160               mov  ID,a            ;* SAVE THE RESULT
0088 90 00 9C     161               mov  dptr,#CASE6     ;* JMP BASE ADDRESS FOR
                  162                                    ;* THIS INSTRUCTION
008B 12 00 8F     163               call INSTR_GEN
008E 22           164               ret

                  165 ;*******************************************************
                  166 ;*
```

```
                    167 ;* NAME: INSTR_GEN (INSTRUCTION GENERATOR)
                    168 ;* FUNCTION: ISSUES APPROPRIATE I2C PROTOCOL FOR EACH X9241
                    169 ;* INSTRUCTION
                    170 ;* INPUTS: ADDR_BYTE,ID,PULSES,DPTR,DATA_BYTE
                    171 ;* OUTPUTS: DATA_BYTE
                    172 ;* CALLS: START_COND, STOP_COND, SEND_BYTE, SEND_BIT,
                    173 ;* GET_BYTE, POLLING
                    174 ;* AFFECTED: DATA_BYTE,A,COUNT
                    175 ;*
                    176 ;*********************************************************

008F 12 01 04       177  INSTR_GEN:  call START_COND    ;* ISSUE AN I2C START
                    178                                 ;* CONDITION
0092 EE             179              mov  a,ADDR_BYTE   ;* SEND X9241 ADDRESS BYTE
0093 12 00 CF       180              call SEND_BYTE
0096 ED             181              mov  a,ID          ;* SEND X9241 INSTRUCTION
                    182                                 ;* BYTE
0097 12 00 CF       183              call SEND_BYTE
009A E4             184              clr  a             ;* JMP OFFSET (DON'T NEED
                    185                                 ;* AN OFFSET)
009B 73             186              jmp  @ a +dptr     ;* JUMP TO VARIOUS
                    187                                 ;* INSTRUCTION CASES
009C EB             188  CASE6:      mov  a,PULSES      ;* A <- BITS DIR X D5 D4 D3
                    189                                 ;* D2 D1 D0
009D 54 3F          190              anl  a,#00111111b  ;* A <- BITS 0 0 D5 D4 D3
                    191                                 ;* D2 D1 D0
009F F9             192              mov  COUNT, a      ;* SAVE AS THE NUMBER OF
                    193                                 ;* PULSES
00A0 EB             194              mov  a,PULSES
00A1 54 80          195              anl  a,#10000000b  ;* A <- BITS DIR 0 0 0 0 0
                    196                                 ;* 0 0
00A3 12 00 E1       197  WIPER_LOOP: call SEND_BIT      ;* SEND THE BIT (A SINGLE
                    198                                    ;* PULSE)
00A6 D9 FB [00A3]   199              djnz COUNT,WIPER_LOOP  ;* CONTINUE UNTIL ALL
                    200                                    ;* PULSES ARE SENT
00A8 02 00 CB       201  CASE4:      jmp  STOP_GEN      ;* IF PROGRAM GETS HERE,
                    202                                 ;*  THEN IT'S DONE
00AB EF             203  CASE2:      mov  a,DATA_BYTE   ;* SEND X9241 DATA BYTE
00AC 12 00 CF       204              call SEND_BYTE
00AF 02 00 CB       205              jmp  STOP_GEN
00B2 12 00 F6       206  CASE1:      call GET_BYTE      ;* RECEIVE X9241 DATA BYTE
00B5 02 00 CB       207              jmp  STOP_GEN
00B8 EF             208  CASE3:      mov  a,DATA_BYTE   ;* SEND X9241 DATA BYTE
00B9 12 00 CF       209              call SEND_BYTE
00BC 12 01 15       210              call STOP_COND     ;* ISSUE A STOP CONDITION
00BF 12 01 24       211              call POLLING       ;* BEGIN ACKNOWLEDGE POLLING
00C2 02 00 CB       212              jmp  STOP_GEN
00C5 12 01 15       213  CASE5:      call STOP_COND     ;* ISSUE A STOP CONDITION
00C8 12 01 24       214              call POLLING       ;* BEGIN ACKNOWLEDGE POLLING
00CB 12 01 15       215  STOP_GEN:   call STOP_COND     ;* I2C TRANSMISSION OVER!
00CE 22             216           ret

                    217   ;*********************************************************
                    218   ;*
                    219   ;* NAME: SEND_BYTE
```

```
                    220   ;* FUNCTION: SENDS 8 BITS (FROM MSB TO LSB) TO SDA AND
                    221   ;* READS 1 BIT FROM SDA
                    222   ;* INPUTS: A
                    223   ;* OUTPUTS: NONE
                    224   ;* CALLS: SEND_BIT,GET_BIT
                    225   ;* AFFECTED: COUNT,TEMP,A
                    226   ;*
                    227   ;*********************************************************

00CF 79 08          228  SEND_BYTE:  mov  COUNT,#8      ;* SET LOOP FOR 8
                    229                                 ;* REPETITIONS
00D1 F8             230               mov  TEMP,a       ;* STORE AS SHIFTED BYTE (NO
                    231                                 ;* SHIFT)
00D2 E8             232  BIT_LOOP:   mov  a,TEMP        ;* RETRIEVE LAST SAVED
                    233                                 ;* SHIFTED BYTE
00D3 54 80          234               anl  a,#10000000b ;* MASK FOR MSB (MOST
                    235                                 ;* SIGNIFICANT BIT)
00D5 12 00 E1       236               call SEND_BIT     ;* PLACE THIS BIT ON SDA
00D8 E8             237  NEXT_BIT:   mov  a,TEMP        ;* RETRIEVE LAST SAVED
                    238                                 ;* SHIFTED BYTE
00D9 23             239               rl   a            ;* ROTATE ALL BITS 1
                    240                                 ;* POSITION LEFT
00DA F8             241               mov  TEMP,a       ;* STORE THIS UPDATED
                    242                                  ;* SHIFTED BYTE
00DB D9 F5 [00D2]   243               djnz COUNT,BIT_LOOP
00DD 12 00 EB       244               call CLOCK         ;* WHEN DONE ALL 8 BITS,
                    245                                  ;* READ SDA LINE
00E0 22             246               ret

                    247   ;*********************************************************
                    248   ;*
                    249   ;* NAME: SEND_BIT
                    250   ;* FUNCTION: PLACES A BIT ON SDA AND INITIATES A CLOCK
                    251   ;* PULSE ON SCL
                    252   ;* INPUTS: A
                    253   ;* OUTPUTS: NONE
                    254   ;* CALLS: CLOCK
                    255   ;* AFFECTED: SDA
                    256   ;*
                    257   ;*********************************************************

00E1 C2 91          258  SEND_BIT:  clr  SDA          ;* PULL SDA LOW
00E3 60 02 [00E7]   259             jz   SENT_ZERO    ;* SHOULD SDA REALLY BE LOW?
00E5 D2 91          260             setb SDA          ;* IF NOT, PULL SDA HIGH
00E7 12 00 EB       261  SENT_ZERO: call CLOCK        ;* INITIATE A CLOCK PULSE
00EA 22             262             ret

                    263   ;*********************************************************
                    264   ;*
                    265   ;* NAME: CLOCK
                    266   ;* FUNCTION: ISSUES A LOW-HIGH-LOW CLOCK PULSE OF
                    267   ;* SUFFICIENT DURATION & READS SDA DURING THE HIGH PHASE,
                    268   ;* JUST IN CASE IT'S NEEDED
                    269   ;* INPUTS: NONE
                    270   ;* OUTPUTS: C
```

```
                    271   ;* CALLS: NONE
                    272   ;* AFFECTED: SCL,C
                    273   ;*
                    274   ;*********************************************************

00EB 00             275  CLOCK:     nop              ;* LET SDA SET-UP
00EC D2 90          276             setb SCL         ;* PULL SCL HIGH AND HOLD
00EE 00             277             nop
00EF     00         278             nop
00F0     00         279             nop
00F1     A2 91      280             mov  c,SDA       ;* MOVE SDA BIT INTO CARRY FLAG
00F3     C2 90      281             clr  SCL         ;* PULL SCL LOW
00F5     22         282             ret

                    283   ;*********************************************************
                    284   ;*
                    285   ;* NAME: GET_BYTE
                    286   ;* FUNCTION: RECEIVES 8 BITS FROM SDA (MSB TO LSB) AND
                    287   ;* SENDS 1 BIT TO SDA
                    288   ;* INPUTS: NONE
                    289   ;* OUTPUTS: DATA_BYTE
                    290   ;* CALLS: CLOCK,SEND_BIT
                    291   ;* AFFECTED: COUNT,SDA,A,DATA_BYTE
                    292   ;*
                    293   ;*********************************************************

00F6 D2 91          294  GET_BYTE:  setb SDA        ;* RECEIVER SHOULDN'T DRIVE SDA
                    295                             ;* LOW
00F8  79 08         296             mov  COUNT,#8   ;* SET LOOP COUNTER TO 8
                    297                             ;* REPETITIONS
00FA 11 EB [00EB]   298  GET_LOOP:  call CLOCK      ;* CLOCK IN THE CURRENT BIT
00FC 33             299            rlc   a          ;* RECONSTRUCT BYTE USING LEFT
                    300                             ;* SHIFTS
00FD D9 FB [00FA]   301             djnz COUNT,GET_LOOP
00FF FF             302             mov  DATA_BYTE,a  ;* STORE RETRIEVED BYTE
                    303                               ;* FOR USER
0100 E4             304             clr  a            ;* A <- LOW (SENDING A 0)
0101 11 E1 [00E1]   305             call END_BIT      ;* SEND AN ACKNOWLEDGE
0103 22             306             ret

                    307   ;*********************************************************
                    308   ;*
                    309   ;* NAME: START_COND (START CONDITION)
                    310   ;* FUNCTION: ISSUES AN I2C BUS START CONDITION
                    311   ;* INPUTS: NONE
                    312   ;* OUTPUTS: NONE
                    313   ;* CALLS: NONE
                    314   ;* AFFECTED: SDA,SCL
                    315   ;*
                    316   ;*********************************************************

0104 D2 91          317  START_COND: setb SDA       ;* PULL SDA HIGH AND ALLOW SET-UP
0106 D2 90          318              setb SCL       ;* PULL SCL HIGH AND HOLD
0108 00             319              nop
0109 00             320              nop
```

```
010A 00           321             nop
010B 00           322             nop
010C C2 91        323             clr  SDA       ;* PULL SDA LOW (SCL=HIGH) AND HOLD
010E 00           324             nop
010F 00           325             nop
0110 00           326             nop
0111 00           327             nop
0112 C2 90        328             clr  SCL       ;* COMPLETE CLOCK PULSE
0114 22           329             ret

                  330   ;*********************************************************
                  331   ;*
                  332   ;* NAME: STOP_COND (STOP CONDITION)
                  333   ;* FUNCTION: ISSUES AN I2C BUS STOP CONDITION
                  334   ;* INPUTS: NONE
                  335   ;* OUTPUTS: NONE
                  336   ;* CALLS: NONE
                  337   ;* AFFECTED: SDA,SCL
                  338   ;*
                  339   ;*********************************************************

0115 C2 91        340  STOP_COND:   clr  SDA      ;* PULL SDA LOW AND HOLD
0117 D2 90        341               setb SCL      ;* PULL SCL HIGH AND HOLD
0119 00           342               nop

011A 00           343               nop
011B 00           344               nop
011C 00           345               nop
011D D2 91        346               setb SDA      ;* PULL SDA HIGH (SCL=HIGH)
011F 22           347               ret

                  348   ;*********************************************************
                  349   ;*
                  350   ;* NAME: ACK_SEND (SEND ACKNOWLEDGE)
                  351   ;* FUNCTION: SENDS AN ACKNOWLEDGE BIT TO COMPLETE SDA LINE
                  352   ;* DATA READS
                  353   ;* INPUTS: NONE
                  354   ;* OUTPUTS: NONE
                  355   ;* CALLS: SEND_BIT
                  356   ;* AFFECTED: A
                  357   ;*
                  358   ;*********************************************************

0120 E4           359  ACK_SEND:    clr  a            ;* A <- LOW (SENDING A 0)
0121 11 E1 [00E1] 360               call SEND_BIT     ;* SEND THE BIT!
0123 22           361               ret

                  362   ;*********************************************************
                  363   ;*
                  364   ;* NAME: POLLING (ACKNOWLEDGE POLLING FOR XFR_WCR,
                  365   ;* WRITE_DR, GXFR_WCR)
                  366   ;* FUNCTION: SENDS DUMMY COMMANDS TO X9241 DURING AN
                  367   ;* INTERNAL WRITE CYCLE SO THAT THE END OF THE CYCLE IS
                  368   ;* MARKED BY AN ACKNOWLEDGE
                  369   ;* INPUTS: ADDR_BYTE
```

```
                    370   ;* OUTPUTS: NONE
                    371   ;* CALLS: START_COND,SEND_BYTE
                    372   ;* AFFECTED: C
                    373   ;*
                    374   ;*********************************************************

 0124 31 04 [0104]  375  POLLING:     call START_COND       ;* REESTABLISH I2C PROTOCOL
 0126 EE            376                mov  a,ADDR_BYTE      ;* ATTEMPT TO SEND A DUMMY
                    377                                      ;* COMMAND
 0127 11 CF [00CF]  378  AGAIN:       call SEND_BYTE
 0129 40 F9 [0124]  379                jc    POLLING         ;* IF C=1, THEN THERE WAS NO
                    380                                      ;* ACKNOWLEDGE
 012B 22            381                ret

                    382   ;*********************************************************
                    383   ;*
                    384   ;* PUT MAIN PROGRAM HERE...
                    385   ;*
                    386   ;*********************************************************

 012C               387  MAIN:

ASSEMBLY END, ERRORS:0, LAST CODE ADDRESS:012BH, TOTAL BYTES:299
```