Using the TNT4882 in an 8051-Family-Microcontroller-Based System

Andrew Thomson
Faisal Habib

Introduction

This application note is written for GPIB instrument designers who use 8051-family microcontrollers. It describes the hardware interface required to create a fully functional GPIB device using an 8051-family microcontroller and TNT4882 GPIB interface ASIC. For this example, a P87C52 is used, but other microcontrollers are similar.

In this document we show how to create an interface that will meet the following specifications:

- Complete IEEE 488.2 Talker/Listener functionality based on the TNT4882-AQ
- 8-bit data transfers
- Polled I/O and Interrupt Driven I/O

Note that the terms assert (assertion) and negate (negation) are used to avoid confusion when dealing with a mixture of active low and active high signals. The term assert (assertion) means that a signal is active or true, independent of the level represented by a high or low voltage. The term negate (negation) means that a signal is inactive or low, independent of the level represented by a high or low voltage.

Source files for the test program are included in the Appendix.

Hardware Description

The TNT4882 has two different pin configurations – ISA and Generic. This application uses the TNT4882 in Generic pin configuration. (Refer to Chapter 5 in Reference 1 for a Generic pin description). Figure 1 shows the P87C52 and TNT4882 hardware interface diagram. The interface logic can be implemented easily in a Programmable Array Logic (PAL) device such as a 16V8.
Memory Map of the System

The P87C52 has separate address space for Program and Data Memory. Program memory is read-only whereas Data memory is read-write.
Internal Program Memory

P87C52 EBF FA comes with 16 KB of internal Program Memory. User/System program may reside in Internal Program Memory.

Internal Data Memory

The lowest 128 bytes of internal data memory can be addressed using Direct and Indirect Addressing. To access user RAM above $7F, use Indirect Addressing. Direct Addressing above $7F access Special Function Registers (SFR).

External Data Memory

The TNT4882 is located in external data memory. It occupies 32 bytes of I/O memory.
Bus Operation

TNT4882 Address Lines

The address pins of the TNT4882 select one of the internal registers during I/O reads or writes. Port 0 of P87C52 is an 8-bit multiplexed address/data bus. Therefore, we latch the address bus during external data accesses.

TNT4882 Data Lines

Because the TNT4882 has built-in transceivers, the data lines connect directly to the CPU without requiring any external components. In general, 8-bit I/O accesses can use either data bus of the TNT4882. ABUSN and BBUSN control the selection of the data bus. We use the lower byte lane (D7-0) for external data accesses in this application.

TNT4882 ABUSN, BBUSN Signals

The TNT4882 can use either of its data busses for 8-bit I/O accesses. ABUSN and BBUSN signals can select either Bus A (D15-8) or Bus B (D7-0). You may leave one of these signals unconnected or tied to Vcc. We connected ABUSN to Vcc and BBUSN to ground to use the lower byte lane for data accesses.

TNT4882 RDN, WRN Signals

During write accesses, the TNT4882 latches data on the rising edge of WRN. The TNT4882 drives its data buses when RDN asserts during read accesses. The processor generates RDN and WRN signals as needed during external data memory accesses. TNT4882 RDN and WRN connect directly to the CPU RDN and WRN signals.

Figures 3 and 4 show the CPU read and write timing diagrams, respectively.
Interrupt Acknowledge Bus Operation

The TNT4882 can interrupt the processor by asserting its interrupt signal. For the above example, interrupt level zero was selected and programmed to be edge-triggered. The processor will acknowledge the interrupt whenever a high-to-low transition occurs on the interrupt pin of the processor. To disable TNT4882 interrupts, clear the IMR3 register.
Other CPU Interface Pins (see Reference 1 for more detail)

**Chip Select (CSN)**

CSN must be asserted during I/O accesses. Depending upon where the TNT4882 is placed in memory, the address lines can be decoded accordingly to generate the CSN signal. In our system, the TNT4882 occupies the lowest 32 bytes of external data memory.

**Reset (RESETN)**

The RESETN pin of the TNT4882 is active low, so the RESET signal from the processor is inverted and connected to the RESETN pin of the TNT4882. Asserting the RESETN signal will reset the TNT4882.

**Interrupt Signal (INTR)**

P87C52 IRQ lines are active low, so the INTR signal from the TNT4882 is inverted and connected to one of the available interrupt lines.

**PAGED Pin**

When the PAGED pin on the TNT4882 asserts, the TNT4882 enters the Paged-In state. If Page-In state is true, several registers are mapped to different offsets. In all new applications, PAGED may be connected to GND.

**MODE Pin**

The MODE pin determines whether the TNT4882 enters Turbo+7210 mode or Turbo+9914 mode after a hardware reset. For the above interface, MODE is left unconnected so that the TNT4882 enters the Turbo+7210 mode. (See MODE and SWAPN Pin Recommendations in Chapter 5 of Reference 1.)

**SWAPN Pin**

The TNT4882 samples the SWAPN pin during a hardware reset. If SWAPN asserts during a hardware reset, the SWAP bit is set. For the above interface, SWAPN pin is left unconnected. (See MODE and SWAPN Pin Recommendations in Chapter 5 of Reference 1.)
FIFO_RDY Pin

The FIFO_RDY output indicates that the FIFOs are ready for at least 8 word (or byte) transfers. Since we did not have any use for FIFO_RDY, it is left unconnected.

ABUS_OEN and BBUS_OEN

The ABUS_OEN output asserts when the TNT4882 drives Data Bus A during a read access. The BBUS_OEN output asserts when the TNT4882 drives Data Bus B during a read access. Since we did not have any use for the above pins, they are left unconnected.

BURST_RDN

When BURST_RDN asserts, the TNT4882C drives Data Bus A and Data Bus B with the next word to be read from the FIFOs. BURST_RDN does not remove data from the FIFOs. For the above interface, BURST_RDN is left unconnected.

Key Pins (KEYRSTN, KEYDQ, KEYCLKN)

The key pins are designed to be connected to a Dallas Semiconductor DS1204U Electronic Key. Applications that do not use the key can leave the key pins unconnected.

GPIB Device Status Pins

The TNT4882 has five device status pins: TADCS, LADCS, TRIG, DCAS, and REM. All device status pins are output only so we leave them unconnected. For further description of the status pins, refer to Chapter 5 of Reference 1.

GPIB Signal Pins

Connect the GPIB signal pins directly to a GPIB connector.

Oscillator Pins (XTALI, XTALO)

A 40-MHz oscillator is required to drive the clock signal. Connect the oscillator output to the XTALI pin of the TNT4882; leave the XTALO pin unconnected.

Vcc and GND Pins

Supply power to all the Vcc pins and connect the ground signal to all the GND pins.
Software Considerations

Once the hardware interface is constructed, a few initialization routines need to be performed. Once these initialization sequences are complete, the CPU can be programmed to implement any GPIB device.

System Configuration

System configuration must be performed before the TNT4882 can be used. Configuration of the system requires you to initialize and configure reset vectors and interrupt registers. Once you have configured these registers, the P87C52-TNT4882 interface is fully functional.

Device Level Programs

We recommend using the TNT4882 in one-chip mode. The TNT4882 by default powers up in two-chip mode to maintain backwards compatibility. Therefore, it is important to initialize the TNT4882 so that it is in one-chip mode.

To simplify your control program, configure the TNT4882 to only use FIFO B during GPIB data transfers. To do this clear the A/BN and 16/8N bits in the CFG register.

Follow the steps listed in Reference 2. It also includes sample routines that can help you write programs for the TNT4882.

Hardware Interface Test

A hardware interface test is included in the Appendix. The hardware interface test program tests the interface between P87C52 and TNT4882. It performs two different types of I/O:

- Programmed I/O
- Interrupt Driven I/O

To perform the hardware test, program the EPROM using the TEST.MDS file.

Programmed I/O Test

Programmed I/O conducts two different tests that make accesses to FIFOs. These tests are:

- 8-Bit Accesses to FIFO A
- 8-Bit Accesses to FIFO B

Programmed I/O tests 1 and 2 write all possible values (0-255) to both the FIFOs using the upper byte lane and the lower byte lane which test the address and data lines (stuck-0, stuck-1, stuck-together). These tests make sure that the FIFOs are accessed properly. They also test the hardware interface such as the address lines, data lines, and bus operation signals.
Interrupt Driven I/O Test

By performing the Interrupt Driven I/O we are testing to make sure that the TNT4882 can assert its interrupt line and that it is properly acknowledged. Once the handler is loaded properly, we enable the NFF bit of IMR3. If the FIFOs have room for a byte, an interrupt will occur. After writing to the FIFOs we enable the NEF bit of IMR3. An interrupt will occur if FIFOs contain bytes that need to be transferred to memory.

If the interrupts do not get acknowledged, chances are that the interrupt line is not connected properly. If an unexpected interrupt occurs, then your system may have deleted the address of the new handler from the exception vector table.

References

Appendix

;***********************************************************************;
;* File Name  : TEST.TXT                                                 *
;* File Type  : Assembly Listing                                        *
;* Created On : 06/14/95                                                *
;* Modified On: 07/28/95                                                *
;* Created By : Faisal Habib                                            *
;* Description: This program will test the hardware interface for the   *
;*              TNT4882 and up8052. In particular it will test         *
;*              Programmed I/O and Interrupt Driven I/O.                *
;***********************************************************************;

;***********************************************************************;
;* NOTE : This program was hand-decoded and programmed on the on-chip   *
;*        EPROM of P87C52EBF FA.                                        *
;***********************************************************************;

;***********************************************************************;
;* Signature Bytes were programmed as follows:                          *
;*   ($30) = $15  (indicates manufactured by Philips)                   *
;*   ($31) = $97  (indicates 87C52)                                     *
;***********************************************************************;

;***********************************************************************;
;* Encryption Array ($2000 - $201F) must contain $FF                   *
;* Refer to Philips 80C51 Data Book                                    *
;***********************************************************************;

;***********************************************************************;
;* TNT4882 Register Map: One Chip Mode and Turbo+7210 Mode             *
;***********************************************************************;

DIR  EQU   $00           ;Data In Register
CDOR EQU   $00           ;Command/Data Out Register
ISR1  EQU   $02           ;Interrupt Status Register 1
IMR1  EQU   $02           ;Interrupt Mask Register 1
ISR2  EQU   $04           ;Interrupt Status Register 2
IMR2  EQU   $04           ;Interrupt Mask Register 2
ACCWR EQU   $05           ;Accessory Write Register (ISA)
SPSR  EQU   $06           ;Serial Poll Status Register
SPMR  EQU   $06           ;Serial Poll Mode Register
INTR  EQU   $07           ;Board Interrupt Register (ISA)
ADSR  EQU   $08           ;Address Status Register
ADMRI EQU   $08           ;Address Mode Register
CNT2  EQU   $09           ;Count 2 Register
CPTR  EQU   $0A           ;Command Pass Through Register
AUXMR EQU   $0A           ;Auxiliary Register
CNT3  EQU   $0B           ;Count 3 Register
ADR0  EQU   $0C           ;Address Register 0
ADR  EQU   $0C           ;Address Register
HSSEL EQU   $0D           ;Handshake Select Register
ADR1  EQU   $0E           ;Address Register 1
EOSR  EQU   $0E           ;End-of-String Register
STS1  EQU   $10           ;Status 1 Register
CFG   EQU   $10           ;Configuration Register
DSR   EQU   $11           ;DIO Status Register
SH_CNT EQU   $11           ;SH_CNT Register
IMR3  EQU   $12           ;Interrupt Mask Register 3
HIER  EQU   $13           ;High-Speed Enable Register
CNT0  EQU   $14           ;Count 0 Register
MISC  EQU   $15           ;Miscellaneous Register
CNT1  EQU   $16           ;Count 1 Register
CSR   EQU   $17           ;Chip Signature Register
KEYREG EQU   $17           ;Key Control Register
FIFOB EQU   $18           ;FIFO B
FIFOA EQU   $19           ;FIFO A
ISR3  EQU   $1A           ;Interrupt Status Register 3
CCR   EQU   $1A           ;Carry Cycle Register
SASR  EQU   $1B           ;Source/Acceptor Status Register
DCR EQU $1B ;DIO Control Register
STSR EQU $1C ;Status 2 Register
CMCR EQU $1C ;Command Register
ISRO EQU $1D ;Interrupt Status Register 0
IMRO EQU $1D ;Interrupt Mask Register 0
TIMR EQU $1E ;Timer Register
BSR EQU $1F ;Bus Status Register
BCR EQU $1F ;Bus Control Register

;************************************************************************;
;************************************************************************;
;* Reset and Interrupt Jump Table                                       *
;************************************************************************;

ORG $0000 ;Reset Vector Location
0000 02 00 7D LJMP MAIN ;Jump to MAIN after RESET

ORG $0003 ;Interrupt 0 Vector Location
0003 02 01 53 LJMP HANDLER ;Load interrupt handler address

ORG $000B ;TIMER 0 Vector Location
000B 02 01 A0 LJMP TIMER0 ;Jump to TIMER0 after time-out

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

; The main routine conducts two different types of tests and displays *
; the result on the LEDs. PP is displayed with decimal points flashing;
; if the tests were successful; else incorrect values are displayed. *
; Correct value is displayed on Port1, Incorrect values on Port2.

ORG $007D
MAIN:

007D 75 81 80 MOV SP,#$80 ;Initialize stack pointer
0080 11 B9 ACALL CFG_uP ;Configure microprocessor
0082 31 3C ACALL SIGNAL ;Signal successful end of routine
0084 11 CC ACALL CFG_TNT ;Configure TNT4882
0086 31 3C ACALL SIGNAL ;Signal successful end of routine
0088 78 19 MOV R0,$00 ;Load address of FIFO A
008A 11 EE ACALL PGM_TST ;Programmed I/O Test (FIFO A)
008C 31 12 CJNE R6,#$00,FINISH ;Stop if error detected
008E 31 3C ACALL SIGNAL ;Signal successful end of routine
0091 78 18 MOV R0,FIFOB ;Load address of FIFO B
0093 11 EE ACALL PGM_TST ;Programmed I/O Test (FIFO B)
0095 31 FF CJNE R6,#$00,FINISH ;Stop if error detected
0097 31 3C ACALL SIGNAL ;Signal successful end of routine
0099 31 29 ACALL INT_TST ;Interrupt Test
009B 31 02 CJNE R6,#$00,FINISH ;Stop if error detected
009D 31 3C ACALL SIGNAL ;Signal successful end of routine

FINISH:
009F BE FF 02 CJNE R6,#$FF,iEND ;Check for Programmed I/O error
00A1 BE EE 02 CJNE R6,#$EE,nEND ;Check for interrupt error
00A4 FF 80 pEND: SJMP pEND ;Loop endlessly
00A6 75 90 08 MOV P1,#$06 ;Display 'I'
00A8 75 A0 06 MOV P2,#$06 ;Display 'I'
00AA 80 F3 SJMP pEND
00A2 75 90 73 nEND: MOV P1,#$73 ;Display 'P'
00A4 75 90 73 MOV P2,#$73 ;Display 'P'
00A6 80 EB SJMP pEND

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

; Module      : CFG-uP                                                 *
; Parameters  : NONE                                                   *
; Return Value: NONE                                                   *
; Created On : 06/14/95                                               *
; Modified On : 06/14/95                                               *
; Description : Configures the CPU. Enables the necessary interrupts, *
; sets the interrupt priority, and initializes the ports. *;
CFG_uP:
00B9 75 A8 83           MOV     IE,#$83                 ;Enable Interrupts (INT0,TIMER0)
00BC 75 B8 01           MOV     IP,#$01                 ;INT0 has highest priority
00BF 75 80 00           MOV     P0,#$00                 ;Initialize ports (0-3)
00C2 75 90 00           MOV     P1,#$00
00C5 75 A0 00           MOV     P2,#$00
00C8 00 00 00           NOP
00CB 22                 RET

CFG_TNT:
00CC 78 1C              MOV     R0,CMDR
00CE 74 22              MOV     A,#$22
00D0 F2                 MOVX    @R0,A                   ;Issue a soft reset command
00D1 78 06              MOV     R0,SPMR
00D3 79 0A              MOV     R1,AUXMR
00D5 74 80              MOV     A,#$80
00D7 F2                 MOVX    @R0,A                   ;Write $80 to SPMR
00D8 F3                 MOVX    @R1,A                   ;Write $80 to AUXMR
00D9 74 99              MOV     A,#$99
00DB F3                 MOVX    @R1,A                   ;Write $99 to AUXMR
00DC 78 17              MOV     R0,KEYREG
00DE E4                 CLR     A
00DF F2                 MOVX    @R0,A                   ;Write $00 to KEYREG
00E0 78 0D              MOV     R0,HSSEL
00E2 74 01              MOV     A,#$01
00E4 F2                 MOVX    @R0,A                   ;Write $01 to HSSEL
00E5 74 02              MOV     A,#$02
00E7 F3                 MOVX    @R1,A                   ;Write $02 to AUXMR
00E8 74 51              MOV     A,#$51
00EA F3                 MOVX    @R1,A                   ;Write $51 to AUXMR
00EB E4                 CLR     A
00EC F3                 MOVX    @R1,A                   ;Write $00 to AUXMR
00ED 22                 RET

PGM_TST:
00CC 78 1C              MOV     R0,CMDR
00CE 74 22              MOV     A,#$22
00D0 F2                 MOVX    @R0,A                   ;Issue a soft reset command
00D1 78 06              MOV     R0,SPMR
00D3 79 0A              MOV     R1,AUXMR
00D5 74 80              MOV     A,#$80
00D7 F2                 MOVX    @R0,A                   ;Write $80 to SPMR
00D8 F3                 MOVX    @R1,A                   ;Write $80 to AUXMR
00D9 74 99              MOV     A,#$99
00DB F3                 MOVX    @R1,A                   ;Write $99 to AUXMR
00DC 78 17              MOV     R0,KEYREG
00DE E4                 CLR     A
00DF F2                 MOVX    @R0,A                   ;Write $00 to KEYREG
00E0 78 0D              MOV     R0,HSSEL
00E2 74 01              MOV     A,#$01
00E4 F2                 MOVX    @R0,A                   ;Write $01 to HSSEL
00E5 74 02              MOV     A,#$02
00E7 F3                 MOVX    @R1,A                   ;Write $02 to AUXMR
00E8 74 51              MOV     A,#$51
00EA F3                 MOVX    @R1,A                   ;Write $51 to AUXMR
00EB E4                 CLR     A
00EC F3                 MOVX    @R1,A                   ;Write $00 to AUXMR
00ED 22                 RET
;*******************************************************************************;
;*******************************************************************************;
PGM_TST:

00EE 7C 00      MOV R4,#$00      ;Clear R4 (Long Loop)
00F0 7D 00      LOOP: MOV R5,#$00  ;Clear R5 (Short Loop)

00F2 ED         WRITE: MOV A,R5
00F3 2C                 ADD A,R4        ;Add R4 and R5
00F4 F2                 MOVX @R0,A      ;Write value to FIFO
00F5 0D                 INC R5          ;Increment R5
00F6 BD 10 F9       CJNE R5,#$10,WRITE ;Loop while R5 < 16

00F9 7D 00      READ: MOV R5,#$00
00FB ED         MOV A,R5
00FC 2C                 ADD A,R4        ;Add R4 and R5
00FD FB                 MOV R3,A        ;Store in R3 (Correct Value)
00FE E2                 MOVX R2,A       ;Store in R2 (Received Value)
0100 6B                 XRL R3
0101 B4 00 0E       CJNE A,#$00,pERROR ;Generate error if mismatch
0104 0D                 INC R5          ;Else increment R5
0105 BD 10 F3       CJNE R5,#$10,READ  ;Loop while R5 < 16

0108 EC                 MOV A,R4
0109 24 0F       ADD A,#$0F          ;Add 15 to R4
010B FC                 MOV R4,A
010C BC FE E1       CJNE R4,#$FE,LOOP ;Loop while R4 < 255
010F 7E 00      MOV R6,#$800     ;Clear error flag
0111 22                 RET

0112 7F 06      pERROR: MOV R7,#$06 ;Flash 6 times
0114 75 89 01  MOV TMOD,#$01  ;16-bit timer
0117 75 8C 00  MOV TH0,#$00    ;Load Hi-byte
011A 75 8A 00  MOV TL0,#$00    ;Load Lo-byte
011D 8B 90                 MOV P1,R3    ;Load correct value
011F 8A A0                 MOV P2,R2    ;Load received value
0121 D2 8C                SETB TR0      ;Start Timer0
0123 BF 00 FD     FLASH: CJNE R7,#$00,FLASH ;Repeat until R7 = 0
0126 7E FF                 MOV R6,#$FF  ;Set error flag
0128 22                 RET

;*******************************************************************************;
;*******************************************************************************;
INT_TST:

0129 78 12                 MOV R0,IMR3
012B 74 08                 MOV A,#$08
012D F2                 MOVX @R0,A      ;Set NFF bit
012E E2                 WT_NFF: MOVX A,@R0
012F 54 08                 ANL A,#$08
0131 70 FB                JNZ WT_NFF      ;Wait for NFF interrupt
0133 74 04                 MOV A,#$04
0135 F2                 MOVX @R0,A      ;Set NEF bit
0136 E2                 WT_NEF: MOVX A,@R0
0137 54 04                 ANL A,#$04
0139 70 FB                JNZ WT_NEF      ;Wait for NEF interrupt
013B 22                 RET

;*******************************************************************************;
;************************************************************************;
;* Module      : SIGNAL                                                 *
;* Parameters  : NONE                                                   *
;* Return Value: NONE                                                   *
;* Created On  : 06/16/95                                               *
;* Modified On : 07/24/95                                               *
;* Description : Signals the successful end of a routine                *
;* Reference   : MCS51 & TNT4882 Application Notes.                     *
;************************************************************************;

SIGNAL:
013C 7F 09              MOV     R7,#$09                 ;Flash 9 times
013E 75 89 01           MOV     TMOD,#$01               ;16-bit timer
0141 75 8C 00           MOV     TH0,#$00                ;Load Hi-byte
0144 75 8A 00           MOV     TL0,#$00                ;Load Lo-byte
0147 75 90 80           MOV     P1,#$80                 ;Display decimal point
014A 75 A0 80           MOV     P2,#$80                 ;Display decimal point
014D D2 8C              SETB    TR0                     ;Start Timer0
014F BF 00 FD   CHECK:  CJNE    R7,#$00,CHECK           ;Repeat until R7 = 0
0152 22                 RET

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

HANDLER:
0153 75 A8 83           MOV     IE,#$83                 ;Enable Interrupts
0156 7E 00              MOV     R6,#$00                 ;Clear error flag
0158 78 12              IT_NFF: MOV     R0,IMR3
015A E2                MOVX    A,@R0
015B FA                MOV     R2,A                   ;Load IMR3 in R2
015C 78 1A              MOV     R0,ISR3
015E E2                MOVX    A,@R0
015F FB                MOV     R3,A                   ;Load ISR3 in R3
0160 5A                ANL     A,R2
0161 54 00              ANL    A,#$08
0163 B4 08 1A           CJNE    A,#$08,IT_NEF
0166 78 12              MOV     R0,IMR3
0168 EA                MOV     A,R2
0169 54 F7              ANL    A,#$F7
016B F2                MOVX    @R0,A                   ;Store value in IMR3
016C 79 1C              MOV     R1,CMDR
016E 74 10              MOV     A,#$10
0170 F3                MOVX    @R1,A                   ;Reset FIFOs
0171 79 18              MOV     R1,FIFOB
0173 74 55              MOV     A,#$55
0175 F3                MOVX    @R1,A                   ;Write $55 to FIFOB
0176 74 AA              MOV     A,#$AA
0178 F3                MOVX    @R1,A                   ;Write $AA to FIFOB
0179 32                 RETI
017A 00                NOP                             ;Empty Space
017B 00                NOP
017C 00                NOP
017D 00                NOP
017E 00                NOP
017F 00                NOP

14
0180 EA   IT_NEF: MOV     A,R2
0181 5B       ANL     A,R3
0182 54 04  ANL     A,#$04
0184 B4 04 11 CJNE    A,#$04,iERROR
0187 78 12  MOV     R0,IMR3
0189 EA       MOV     A,R2
018A 54 FB   ANL     A,#$FB
018C F2       MOVX    @R0,A
018D 79 18  MOV     R1,FIFOB
0187 B4 04 11 CJNE    A,#$04,iERROR
0189 EA       MOV     A,R2
018A 54 FB   ANL     A,#$FB
018C F2       MOVX    @R0,A
018D 79 18  MOV     R1,FIFOB
0190 B4 55 05 CJNE    A,#$55,iERROR
0193 E3       MOVC    @R1
0194 B4 AA 01 CJNE    A,#$AA,iERROR
0197 32       RETI

0198 7E EE  iERROR: MOV     R6,#$EE                 ;Set error flag
019A 32       RETI
019B 00       NOP                             ;Empty Space
019C 00       NOP
019D 00       NOP
019E 00       NOP
019F 00       NOP

************************************************************************
************************************************************************

;************************************************************************;
; Module      : TIMER0                                                 *
; Parameters  : Number of flashes passed in R7                         *
; Return Value: NONE                                                   *
; Created On  : 06/16/95                                               *
; Modified On : 07/24/95                                               *
; Description : Timer0 Handler. This routine is entered when the timer*;
;                overflows. R7 contains the number of flashes. Values*;
;                at port 1 and 2 are turned ON and OFF after 50ms.    *
;************************************************************************;

TIMER0:
01A0 C2 8C  CLR     TR0                     ;Stop Timer0
01A2 1F       DEC     R7                      ;Decrement number of flashes
01A3 E5 90  MOV     A,P1                    ;Read value at P1
01A5 60 19  JZ      ON                      ;Jump to ON if value is 0 (OFF)
01A7 FB       MOV     R3,A                    ;Store value in R3
01AB 75 90 00  MOV     P1,#$00                ;Write the new value
01AB AA A0  MOV     R2,P2                    ;Read value at P2
01AD 75 A0 00  MOV     P2,#$00                ;Write the new value
01B0 BF 00 01  CJNE    R7,#$00,START         ;Is R7 0 ?
01B3 32       RETI                             ;Return if R7 is 0
01B4 75 89 01  START: MOV     TMOD,#$01        ;16-bit timer
01B7 75 8C 00  MOV     TH0,#$00                ;Load Hi-byte
01BA 75 8A 00  MOV     TL0,#$00                ;Load Lo-byte
01BD D2 8C  SETB    TR0                    ;Start Timer0
01BF 32       RETI
01C0 BB 90  ON:  MOV     P1,R3                 ;Turn LEDs ON
01C2 8A A0  MOV     P2,R2                   ;Turn LEDs ON
01C4 80 EA  SJMP    CHKR7                    ;************************************************************************;