In-System Programming from an Embedded Processor

Overview

This section describes how to program Lattice ISP™ devices using an embedded processor. The first section shows the use of a microprocessor to control ISP, including the construction of a simple ISP port. The second section shows an 8051 microcontroller used as an ISP controller and covers the procedures and assembly code required for processor-based ISP programming. The 8051 assembly code is written in a modular format. A higher level of routines provides the user with device-level functions such as Read ID, Bulk Erase, Program, and Verify. In an attempt to provide routines that can be used across all ISP device families, specific routines are written for the ispGDS®22 devices. Only slight modifications to the basic functions used across these devices are required to adapt the routines to program any other ISP device.

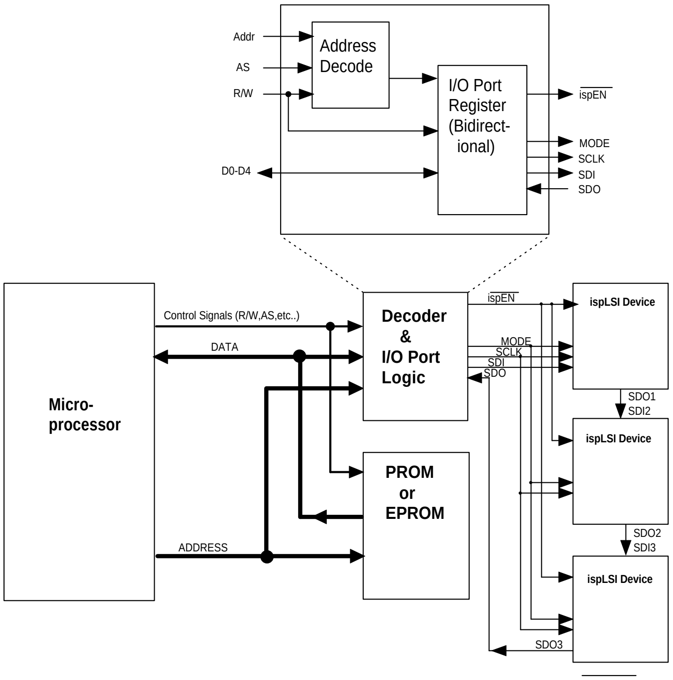
Programming with a Microprocessor

There are several ways to define the ISP programming hardware for microprocessor-based in-system programming, depending on the type of storage device used and how the ISP devices are to be programmed. Since an additional step is necessary to convert a JEDEC file into an ispSTREAM®, the fuse map information can be stored in a JEDEC standard fuse map file, simplifying design changes. Or, since a JEDEC standard fuse map file takes an order of magnitude more memory space to store the fuse map information than an ispSTREAM, the ISP programming routines can be stored in an ispSTREAM. This section presents a hardware configuration in which the fuse map information is stored in a JEDEC standard fuse map file.

The hardware configuration shown in Figure 1 uses an 8-bit wide EPROM to store the JEDEC fuse map file and object code, which is created from ispCODE® C++ source code (see the *Software Basics* document and ispCODE Software data sheet for a complete description of ispCODE). The patterns are then read from the EPROM by the microprocessor and converted into serial stream format. The ISP signals are driven from the decoder and I/O port which decodes the proper ISP read/write address space (similar to the I/O port definition of the previous setup). Similarly, fuse map memory addresses must be defined to be properly read from the EPROM.

The I/O port can be implemented using a dedicated port chip or a PLD. In the case of a PLD implementation, the device must have five pins for the ISP port, five pins for the data port, one pin each for AS and R/W, and enough additional pins to provide address inputs to the decoder. If a partial rather than full decode can be used, the number of address pins can be reduced. For example, for

Figure 1. Microprocessor-Based ISP



a processor with a 16-bit I/O space, if a block of 256 locations can be allocated to the ISP port, only the upper eight addresses (A8-A15) need to be decoded and the function can fit into a 24-pin low-density GAL® device such as the GAL 20RA10. If a full decode is needed, a high-density PLD with ample pins such as Lattice's ispLSI® or pLSI® 2032 or 1016 should be used.

Most hardware timing requirements can be satisfied by the microprocessor software instruction execution time. Only the program, verify, and bulk erase times require the software to have wait cycles. Many microprocessor boards will not have a timer chip to time the wait states. However, the instruction execution times can typically be estimated accurately. Therefore, timing loops must be inserted into the instructions to control critical hardware timing.

Within ispCODE source code, before the object code is created, address spaces for the ISP read/write locations and the EPROM read locations must be defined. The storage space requirement for the object code must also be determined if the code is going to reside in the storage device. Based on the ispCODE functions, the object code which is capable of executing basic ISP functions typically does not exceed 8K bytes of memory. This memory requirement is directly proportional to the number of ISP and user interface functions.

Programming with a Microcontroller

The advantage of using a microcontroller-based ISP interface is that the ports are integrated with the processor. As shown in Figure 2, the interface to the ISP devices is accomplished through I/O port 1. RAM and EPROM access is also shown in the block diagram through the use of I/O port 0 and I/O port 2. These specific connections may be changed according to the user's application. Direct connections are made from the port to the ISP pins of the device. The pinout used on I/O port 1 is listed below:

```
SDI     P1.0
SCLK    P1.1
MODE    P1.2
SDO     P1.3
```

The address and data to the RAM and EPROM are multiplexed through a 74LS373 latch and the control signals are routed through the 74LS138 decoder.

The timing requirements for ISP programming are with respect to the SCLK signal. To shift between states, the SDI and MODE control signals are set to the required values for a state transition and SCLK is applied. In this manner, the set up times are easily met and interrupts can occur at any time during the application of ISP signals.

Software Overview

The main function of the embedded processor assembly software routines is to drive the ISP programming state machine (Figure 3), while ensuring that all programming timing requirements are met. The programming pulse width is controlled by a counter delay. The resulting pulse width of the counter depends on the clock frequency of the microcontroller and the bit width of the counter. The values used here are based on a clock rate of 12 MHz and the use of the default state of the counter which is 13 bits.

Figure 2. Microcontroller Block Diagram

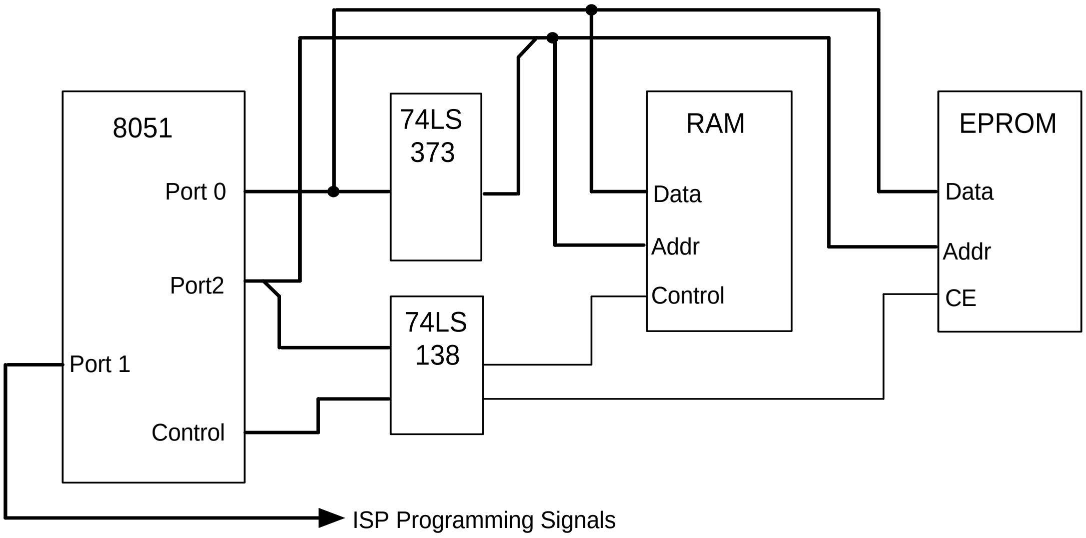
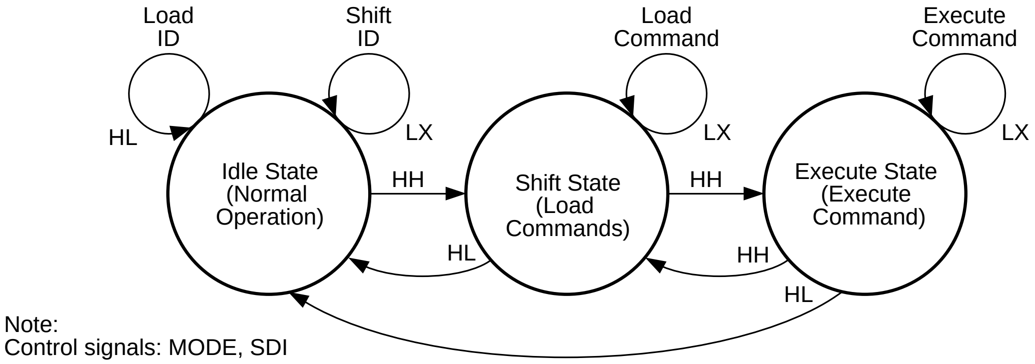


Figure 3. ISP Programming State Machine


Load ID
Shift ID
Load Command
Execute Command
HL
LX
LX
LX
Idle State (Normal Operation)
HH
Shift State (Load Commands)
HH
Execute State (Execute Command)
HL
HH
HL

Note:
Control signals: MODE, SDI

Software resources, such as internal registers, are self-contained within the routines. These resources are freed after the routines have been properly executed. The only resources that have to be dedicated to ISP in this example are the I/O port signals that are used to drive the ISP programming signals. The user has a choice to free the I/O port resources if the I/O port signals are multiplexed. The ORG statement indicates the beginning address of the subroutine and is used to place the assembly routines within the main code. If the routine codes are intended for a user application using the same internal register resources, the registers may be pushed onto a stack to preserve them. Upon completion of the subroutine, they are popped off the stack so that there is no interference with the calling assembly program.

JEDEC Map Creation and Storage

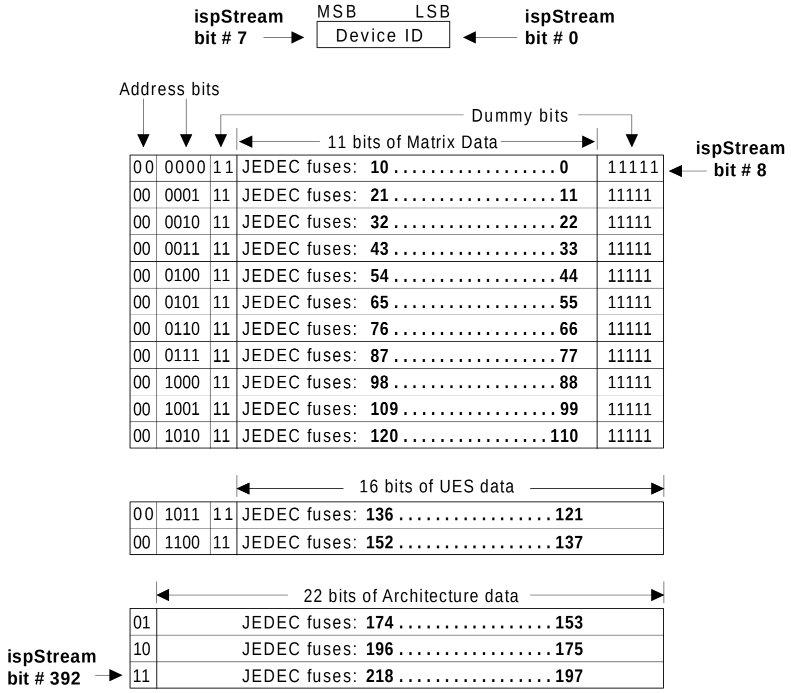
The JEDEC pattern is stored as part of the 8051 executable code. It is differentiated from the assembly listing by an assembler directive which flags the program space to be used as data bytes. The JEDEC map is placed at the bottom of the assembly file, as memory, and referenced with a label to mark its position. Assignments are automatically adjusted by the assembler. Placing the JEDEC map at the bottom also simplifies using sequential JEDEC files for multiple device programming. This is desirable for PC boards intended for multiple function programming during manufacturing or in the field.

The JEDEC Fuse Map Shift Procedures

In the source listing section of this document, the JEDEC fuse map for the ispGDS22 is listed as a sequential order of bytes. Each byte is broken into a bit sequence and written out serially to a port pin. Another port pin is then used as a clock driver for clocking in the serial data information. The JEDEC file is stored so that it is read out sequentially from top to bottom. In the case of ispGDS and ispGAL® devices, each line of the JEDEC file contains both the data and address information and can be programmed as a single stand-alone line. However, in the case of ispLSI devices, the address shift routine must be executed as a separate routine prior to shifting in the data bits.

When shifting the JEDEC data from program memory, a shift left operation is used on each byte to shift out the data to the I/O port. This means that a reverse bit order is stored in the bytes. If the order of the bits is to be maintained, then a shift right instruction may be used to shift the bits out of each of the bytes. The stored ispGDS JEDEC file is in the same format as Figure 4. Bits in the JEDEC file are shifted LSB first. This means that the five ‘1’s will be shifted out, starting at the right hand side of the table. Note that this method of shifting out the bits requires reverse ordered bytes of information.

Figure 4. ispGDS JEDEC Fuse Map



00	0000	11	JEDEC fuses: **10** **0**	11111
00	0001	11	JEDEC fuses: **21** **11**	11111
00	0010	11	JEDEC fuses: **32** **22**	11111
00	0011	11	JEDEC fuses: **43** **33**	11111
00	0100	11	JEDEC fuses: **54** **44**	11111
00	0101	11	JEDEC fuses: **65** **55**	11111
00	0110	11	JEDEC fuses: **76** **66**	11111
00	0111	11	JEDEC fuses: **87** **77**	11111
00	1000	11	JEDEC fuses: **98** **88**	11111
00	1001	11	JEDEC fuses: **109** **99**	11111
00	1010	11	JEDEC fuses: **120** **110**	11111

00	1011	11	JEDEC fuses: **136** **121**
00	1100	11	JEDEC fuses: **152** **137**

01	JEDEC fuses: **174** **153**
10	JEDEC fuses: **196** **175**
11	JEDEC fuses: **218** **197**

ISP Programming Routines

The programming routines were constructed in a modular format through the use of subroutines. Each subroutine is constructed so that it controls the appropriate state transitions as shown in Figure 3. The routines can be easily traced keeping in mind the ISP states and state transitions. The names of the subroutines follow the names of the state machine closely for ease of readability and comprehension.

Programming Sequence

The code is broken into three major blocks. Each is executed in sequence. The first block is that of device identification. In this section, note that the identification of the device is hard coded. The next major block performs a bulk erase on the device. The last block loads the JEDEC map into the device.

Prior to these blocks is the configuration area of the 8051 microcontroller. This is standard configuration information that is generally present at the top of any 8051 microcontroller assembler file. Typical information contained in the configuration area includes port addresses and interrupt type of addresses.

The identification of the device is marked in the assembler code by the label “READID.” The completion of this section is marked by the label “END_READID:.”

The first step in reading the ID is to establish the starting state from which the action is to take place. Normally this is done by moving to the first state of the state machine, which is the idle state (Listing 1).

Listing 1. Embedded Processor Programming Algorithm

Follow the steps below to read the device identification:

- Move to the IDLESTATE.
- In the IDLESTATE execute the LOAD ID Command.
- Enable the shifting of the IDENTIFICATION of the device.
- Set up of the registers for a loop counter and for storing the device ID.
- Clock each bit of the device ID with an SCLK signal and do a shift store of the information.
- Compare the ID that was shifted in with the ID of an ispGDS22 and continue if the identification matches. Otherwise, enter into an endless loop with no other actions.

Follow the steps below to bulk erase the part prior to programming:

- From the IDLESTATE, the subroutine BULK_ERASE is called. It in turn performs the following actions:
  - It places the state machine into the IDLESTATE
  - Changes state to the SHIFTSTATE , getting the ISP state machine ready for shifting in a command
  - Shifts in the Bulk erase command
  - Changes state to the EXECUTE STATE
  - Calls EXECUTE COMMAND once and waits 200 milliseconds
  - Returns to the point from where the subroutine call was made (Note that at this point Bulk erasing of the part is recommended so the device will be programmed from a known starting point)

Follow the steps below to program the device:

- Place the device in the IDLESTATE
- Change state to the SHIFTSTATE
- Shift in the SHIFTDATA command
- Move to the EXECUTE_STATE
- Execute the command

The state machine is now ready for shifting in each line of the JEDEC file:

- Prepare the registers for the looping that is required for the shifting in of the bytes and bits within each byte of information
- Load each of the bytes in its turn in a shift register and shift out.

Once a complete line has been shifted out, call the subroutine PROGRAM which does the following:

- Calls the subroutine SHIFTSTATE
- Moves from the EXECUTE_STATE to the SHIFT STATE
- Calls the subroutine PROGRAM_CMD which shifts in the program command and returns to the point where it was called
- Moves to the EXECUTE_STATE
- EXECUTEs the command
- Delays the movement out of the execute above to allow for the programming of the line of the JEDEC file
- Shifts to the SHIFTSTATE state
- Calls the SHIFTSTATE subroutine which performs the functions as noted above
- Moves to the EXECUTE_STATE

```
 - EXECUTES the command
 - Returns to the point from where the PROGRAM subroutine was called
```

After completion of the PROGRAM subroutine call, go back to the top of the loop that loads in another row of the JEDEC file and repeat the procedure until all of the rows and bits have been loaded in. On completion of programming the device, continue with execution of the user code or go into a loop which tells the user that the programming of the device has been completed (such as the indefinite subroutine call to PROG_COMP which flashes an LED).

Listing 2 is the assembler listing that follows the algorithm in Listing 1.

Listing 2. Assembler Listing

```
;$MOD51
$TITLE(ISP PROGRAMMER FOR 8051)
$PAGEWIDTH(132)
$DEBUG
$OBJECT
$NOPAGING

;  Variable declarations
;
      P0              DATA      080H      ;PORT 0
      P1              DATA      090H      ;PORT 1
      P2              DATA      0A0H      ;PORT 2
      P3              DATA      0B0H      ;PORT 3

      SP              DATA      081H      ;STACK POINTER
      DPL             DATA      082H      ;DATA POINTER - LOW BYTE
      DPH             DATA      083H      ;DATA POINTER - HIGH BYTE
      PCON            DATA      087H      ;POWER CONTROL
      TCON            DATA      088H      ;TIMER CONTROL
      TMOD            DATA      089H      ;TIMER MODE
      TL0             DATA      08AH      ;TIMER 0 - LOW BYTE
      TL1             DATA      08BH      ;TIMER 1 - LOW BYTE
      TH0             DATA      08CH      ;TIMER 0 - HIGH BYTE
      TH1             DATA      08DH      ;TIMER 1 - HIGH BYTE

      IE              DATA      0A8H      ;INTERRUPT ENABLE
      IP              DATA      0B8H      ;INTERRUPT PRIORITY
      ACC             DATA      0E0H      ;ACCUMULATOR
      B               DATA      0F0H      ;MULTIPLICATION REGISTER

      IT0             BIT       088H      ;TCON.0 - EXT. INTERRUPT 0 TYPE
      IE0             BIT       089H      ;TCON.1 - EXT. INTERRUPT 0 EDGE FLAG
      IT1             BIT       08AH      ;TCON.2 - EXT. INTERRUPT 1 TYPE
      IE1             BIT       08BH      ;TCON.3 - EXT. INTERRUPT 1 EDGE FLAG
      TR0             BIT       08CH      ;TCON.4 - TIMER 0 ON/OFF CONTROL
      TF0             BIT       08DH      ;TCON.5 - TIMER 0 OVERFLOW FLAG
      TR1             BIT       08EH      ;TCON.6 - TIMER 1 ON/OFF CONTROL
      TF1             BIT       08FH      ;TCON.7 - TIMER 1 OVERFLOW FLAG
      RI              BIT       098H      ;SCON.0 - RECEIVE INTERRUPT FLAG
      TI              BIT       099H      ;SCON.1 - TRANSMIT INTERRUPT FLAG
      RB8             BIT       09AH      ;SCON.2 - RECEIVE BIT 8
      TB8             BIT       09BH      ;SCON.3 - TRANSMIT BIT 8
      REN             BIT       09CH      ;SCON.4 - RECEIVE ENABLE
      SM2             BIT       09DH      ;SCON.5 - SERIAL MODE CONTROL BIT 2
      SM1             BIT       09EH      ;SCON.6 - SERIAL MODE CONTROL BIT 1
      SM0             BIT       09FH      ;SCON.7 - SERIAL MODE CONTROL BIT 0

      EX0             BIT       0A8H      ;IE.0 - EXTERNAL INTERRUPT 0 ENABLE
      ET0             BIT       0A9H      ;IE.1 - TIMER 0 INTERRUPT ENABLE
      EX1             BIT       0AAH      ;IE.2 - EXTERNAL INTERRUPT 1 ENABLE
      ET1             BIT       0ABH      ;IE.3 - TIMER 1 INTERRUPT ENABLE
      ES              BIT       0ACH      ;IE.4 - SERIAL PORT INTERRUPT ENABLE

      SCON            DATA      098H      ;SERIAL PORT CONTROL
      SBUF            DATA      099H      ;SERIAL PORT BUFFER

      EA              BIT       0AFH      ;IE.7 - GLOBAL INTERRUPT ENABLE
      RXD             BIT       0B0H      ;P3.0 - SERIAL PORT RECEIVE INPUT
      TXD             BIT       0B1H      ;P3.1 - SERIAL PORT TRANSMIT OUTPUT
      INT0            BIT       0B2H      ;P3.2 - EXTERNAL INTERRUPT 0 INPUT
      INT1            BIT       0B3H      ;P3.3 - EXTERNAL INTERRUPT 1 INPUT
      T0              BIT       0B4H      ;P3.4 - TIMER 0 COUNT INPUT
      T1              BIT       0B5H      ;P3.5 - TIMER 1 COUNT INPUT
      WR              BIT       0B6H      ;P3.6 - WRITE CONTROL FOR EXT. MEMORY
      RD              BIT       0B7H      ;P3.7 - READ CONTROL FOR EXT. MEMORY
      PX0             BIT       0B8H      ;IP.0 - EXTERNAL INTERRUPT 0 PRIORITY
      PT0             BIT       0B9H      ;IP.1 - TIMER 0 PRIORITY
      PX1             BIT       0BAH      ;IP.2 - EXTERNAL INTERRUPT 1 PRIORITY
      PT1             BIT       0BBH      ;IP.3 - TIMER 1 PRIORITY
      PS              BIT       0BCH      ;IP.4 - SERIAL PORT PRIORITY

      RS1             BIT       0D4H      ;PSW.4 - REGISTER BANK SELECT 1
      F0              BIT       0D5H      ;PSW.5 - FLAG 0
      AC              BIT       0D6H      ;PSW.6 - AUXILIARY CARRY FLAG
      CY              BIT       0D7H      ;PSW.7 - CARRY FLAG
      SDI             EQU       P1.0
      SCLK            EQU       P1.1
      MODE            EQU       P1.2
      SDO             EQU       P1.3
      ESC             EQU       1BH        ;escape character


;NOTE THAT THE JEDEC_TABLE IS LOCATED AT THE BOTTOM OF THE FILE.


;————END OF VARIABLES——————————————————————

            ORG 2100H       ;START OF PROGRAM
            AJMP BEGIN
            ORG 2103H       ;EXTERNAL INTERUPT
            SETB B.0        ;SET FLAG TO PROGRAM
            RETI
```

```
;============== INTERRUPT SERVICE ROUTINE FOR TIMER0 INTERRUPT ==========

            ORG 210BH
            SETB TFO          ; TIMER HAS OVERFLOW
            RETI

BEGIN:          SETB EA       ;ENABLE ALL INTERRUPTS, GLOBAL
            SETB EX0          ;ENABLE INTERRUPT 0
            MOV B,00H
            AJMP BLINK        ;OUT OF RESET JUMP TO
                              ;THE TABLE START


BLINK:          SETB P1.7
            JB B.0,START
            ACALL DELAY200
            ACALL DELAY200
            CLR P1.7
            ACALL DELAY200
            AJMP BLINK


START:

;############### READ ID OF THE DEVICE #########################

READID:     ACALL   IDLESTATE       ;THESE 2 FUNCTION CALLS
            ACALL   LOAD_ID         ;ACCOMPLISH THE SAME THING
            ACALL   SHIFT_EN        ;CLEARING THE MODE BIT SO THAT
                                    ;SHIFTING CAN TAKE PLACE

            MOV     R0,#07H         ; LOOP COUNTER SET UP FOR 7 COUNT
            MOV     R1,#00H         ; TEMP REG FOR ID BYTE
            SETB    P1.5            ; LED INDICATOR FOR ID OFF

LABEL:      MOV     A,R1
            JB      P1.3,HI_BIT     ; JUMP IF ITS A HI ON SDIN (P1.3)
            CLR     ACC.7           ; IF NOT ITS A LOW, PUT A LOW AT MSB
            AJMP    OVER1
HI_BIT:     SETB    ACC.7           ; IT IS A HIGH, PUT A HIGH AT MSB
OVER1:
            RR      A               ; SHIFT RIGHT
            MOV     R1,ACC          ; MOVE IT TO TEMP REG R1
            ACALL   SCLOCK          ; CLOCK AND GET READY FOR NEXT BIT
            DJNZ    R0,LABEL        ; GET NEXT VALUE, DO THIS 7 TIMES

            CJNE    A,#072H,NO_ID   ; 72H IS THE DEVICE ID FOR ispGDS22
            CLR     P1.5            ; SET P1.5 IF THE CORRECT ID
            AJMP    CONT1
NO_ID:      SETB    P1.5            ; LED LIGHTS ON THE BOARD
            AJMP    NO_ID           ; SO THAT SIGNAL INTEGRITY CHECKS
                                    ; CAN BE MADE.

CONT1:          NOP
```

```
; END_READ ID
; TO THIS POINT THE DEVICE ID HAS BEEN READ
; AND GENERAL BULK ERASE AND PROGRAMMING IS
; TO TAKE PLACE


            ACALL   BULK_ERASE

; AT THIS POINT HAVE READ ID AND BULK ERASED THE PART


; SET UP FOR JEDEC PROGRAMMING SEQUENCE OF THE DEVICE
            ACALL   IDLESTATE

            ACALL   SHIFTSTATE      ;MOVED FROM EXECUTE STATE TO
                                    ;LOAD COMMAND STATE


            ACALL   SHFTDATA_CMD    ; SETUP FOR PLACING DATA
                                    ; INTO THE SHIFT COMMAND

            ACALL   EXECUTE_STATE
            ACALL   EXECUTE         ; EXECUTE SHIFT CMD



;NOW DEVICE IS READY FOR THE LOADING OF THE JEDEC PATTERN
;LOADING OF THE JEDEC PATTERN IS DONE IN THE EXECUTE STATE AFTER THE
;SHIFTDATA COMMAND.

;LOOP COUNTERS ARE SET UP SO THAT LOADING OF THE JEDEC FILE CAN TAKE PLACE.

;=============== PROGRAM THE DEVICE =======================

;THE NEXT 4 LINES INITIALIZE COUNTERS AND DATA POINTER

;PROGRAMMING OF THE DEVICE IS TO TAKE PLACE IN THE
;EXECUTE STATE OF THE DEVICE.

            MOV     R3,#16D          ; ROW COUNTER FOR ispGDS
            MOV     R2,#03D          ; COLUMN COUNTER 3 BYTES/ROW
            MOV     R1,#08D          ; SHIFT COUNTER WITH THE BYTE

            MOV    DPTR,#JEDEC_TABLE ; ADDR OF JEDEC FILE

;=========================================================================
ROWS:
LOADBYTE:   CLR     A               ; CLEAR ACC
            MOVC    A,@A+DPTR       ;LOAD FIRST BYTE OF JEDEC IN A

;MOVE CODE BYTE RELATIVE TO DPTR TO Acc
;METHOD OF BRINGING IN STORED DATA IN THE CODE SEGMENT OF MEMORY.
```

```
LOOP8:      JB  ACC.7,ITS_1         ;JUMP IF MSB IS 1

ITS_0:      CLR SDI                 ;NO! MSB WAS 0, SO LOAD A 0
            CLR MODE
            ACALL SCLOCK
            JMP OVER                ;JUMP OVER TEST FOR BIT=1, IT WAS 0

ITS_1:      SETB SDI                ;MSB IS 1, SO LOAD A 1
            CLR MODE
            ACALL SCLOCK
OVER:

            RL A                    ; GET NEXT BIT IN ACC INTO MSB
                              ; POSITION, ROTATE LEFT
            DJNZ R1,LOOP8
            INC DPTR                ; MOVE DATA POINTER TO THE NEXT BYTE
            MOV R1,#08D             ; RESET BIT POSITION COUNTER FOR ACC


            DJNZ R2,LOADBYTE
            MOV R2,#03D             ; RESET BYTE COUNTER 3 BYTES PER ROW



            ACALL PROGRAM           ;PROGRAM A SINGLE ROW, PROGRAM A ROW
                              ;AND RETURN

            DJNZ R3,ROWS            ;READY TO LOAD NEXT ROW WITH NEW
                                    ;DATA



            ACALL IDLESTATE         ;PLACE THE DEVICE IN A KNOWN STATE
                                    ;READY FOR NORMAL OPERATION.
PROG_COMP:  SETB P1.5
            ACALL DELAY200          ;ON COMPLETION OF THE JEDEC PROGRAMMING
            ACALL DELAY200          ;THE DEVICE IS PLACED INTO AN ENDLESS
            CLR P1.5                ;LOOP OF FLASHING AN LED
            ACALL DELAY200
            AJMP PROG_COMP

;##############################################################
;                 SUBROUTINES
;##############################################################

;=============== BULK ERASE THE DEVICE ======================
BULK_ERASE: CALL    IDLESTATE         ; GO TO IDLESTATE
            ACALL   SHIFTSTATE        ; GO TO SHIFT STATE
            ACALL   BERASE_CMD      ; LOAD BULK ERASE CMD
            ACALL   EXECUTE_STATE   ; GOTO EXECUTE STATE
            ACALL   EXECUTE         ; EXECUTE BULK ERASE CMD
            ACALL   DELAY200        ; CALL DELAY ROUTINE
            RET
```

```
;——————ID SHIFT ENABLE—LX——————————————
SHIFT_EN:   CLR MODE          ;JUST SET THE REQUIRED BITS
            RET               ;WITHOUT PULSING THE CLOCK

;———————SUBROUTINE FOR THE PROGRAMMING OF THE SHIFTED ROWS

PROGRAM:    ACALL   SHIFTSTATE      ; GOTO SHIFT STATE (MOVE FROM THE
                                    ; EXECUTE STATE TO THE SHIFT STATE)
            ACALL   EXECUTE
            ACALL   PROGRAM_CMD     ; LOAD PROGRAM COMMAND
            ACALL   EXECUTE_STATE   ; GOTO EXECUTE STATE
            ACALL   EXECUTE         ; EXECUTE PROGRAM CMD
            ACALL   DELAY50
            ACALL   DELAY50

            ACALL   SHIFTSTATE      ; PROGRAMMING OF ONE ROW IS DONE
            ACALL   EXECUTE         ; GET READY FOR NEXT ROW

;THE RETURN FROM THIS COMMAND IS IN THE SHIFT STATE (LOAD COMMANDS ;STATE)

             ACALL   SHFTDATA_CMD     ;GET READY TO SHIFT IN NEXT ROW
             ACALL   EXECUTE_STATE
             ACALL   EXECUTE
             RET

;———————IDLESTATE ——HL——————————————

IDLESTATE:        SETB MODE         ;ALSO LOADS THE DEVICE ID
            CLR SDI                 ;SO THAT IT CAN BE SHIFTED OUT
            ACALL SCLOCK
            RET
;———————SHIFTSTATE ———HH————————————————

; GENERIC SUBROUTINE FOR THE CHANGING OF STATE
SHIFTSTATE:     SETB MODE
            SETB SDI
            ACALL SCLOCK
            RET

;———————EXECUTESTATE ———HH——————————————

EXECUTE_STATE:    SETB SDI
            SETB MODE
            ACALL SCLOCK
            RET
;———————LOAD ID STATE————HL—————————————

LOAD_ID:        SETB MODE
            CLR SDI
            ACALL SCLOCK
            RET
```

```
;——————TOGGLE SCLK——LHL————————————————
SCLOCK:           CLR SCLK
            SETB SCLK
            CLR SCLK
            RET

;—————DELAY 50MS———————————————————
;FOR 12 MHZ     13 BIT UP COUNTER OVERFLOW GENERATES AN INTERUPT
;   65535 - (50MS/12US)=61368——> EFB8  HEX

DELAY50:        MOV TH0, #0FH
            MOV TL0, #0BAH
            SETB TR0                ;START TIMER0 SET TCON.4

COUNTING:     JB TF0, TIMEOUT
            SJMP COUNTING           ; COUNTING IN LOOP

TIMEOUT:      CLR TF0
            CLR TR0                 ; CLEAR TCON.4
            RET

;—————DELAY 200MS———————————————————
DELAY200:     MOV R0, #04H
LOOP4:        ACALL DELAY50
            DJNZ R0, LOOP4
            RET

;———————BULK ERASE COMMAND—————————————————
; BULK ERASE COMMAND: 00011
BERASE_CMD:   CLR MODE                 ; MODE=0
            SETB SDI                 ; SDI=1
            ACALL SCLOCK             ; SCLK LHL CLOCK IN 2 ONES
            ACALL SCLOCK             ; SCLK LHL
            CLR  SDI                 ; SDI=0
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            RET

;———EXECUTE COMMAND——LL————————————————————
;THIS COMMAND CAN BE USED TO EXECUTE LOAD COMMAND AS WELL AS EXECUTE
;INSTRUCTIONS

EXECUTE:      CLR SDI
              CLR MODE
            ACALL SCLOCK
            RET
;———————————————————————————————————

;———————PROGRAM COMMAND—————————————————

;THIS SUB ROUTINE DOES NOT HAVE THE SET UP REQUIRED PRIOR TO ENTERING
;INTO IT.
```

```
; PROGRAM COMMAND: 00111
PROGRAM_CMD:    CLR MODE                 ; MODE=0
            SETB SDI                 ; SDI=1
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            CLR  SDI                 ; SDI=0
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            RET

;———————SHIFT DATA COMMAND—————————————————

; SHIFT DATA COMMAND: 00010
; SET UP OF THE COMMAND TO ACCEPT PROGRAMMING INFORMATION

SHFTDATA_CMD:   CLR MODE                 ; MODE=0
            CLR  SDI                 ; SDI=0
            ACALL SCLOCK             ; SCLK LHL
            SETB SDI                 ; SDI=1
            ACALL SCLOCK             ; SCLK LHL
            CLR  SDI                 ; SDI=0
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            ACALL SCLOCK             ; SCLK LHL
            RET


;———— JEDEC DATA CONTAINS DUMMY BITS AND ADDRESSES ————————
;NOTE THAT IN THE DATA BLOCK BELOW THAT THE COMPILER REQUIRES THAT
;NUMBERS DO NOT HAVE A LEADING A LETTER AND AS SUCH A LEADING 0 MUST
;BE INCLUDED.
;BIT MAPPING AND TRANSLATION OF THE JEDEC PATTERN THAT IS TO BE
;PROGRAMMED INTO THE PART. NOTE THE BYTES AND BITS HAVE BEEN REVERSED
;IN THE ORDER THAT IS SHOWN IN THE MANUAL.

;BIT SHIFTING IS DONE LEFT TO RIGHT.

; F    A    A    A    C    0
;1111 1010 1010 1010 1100 0000
; F    A    A    A    E    0
;1111 1010 1010 1010 1110 0000
; F    A    A    A    D    0
;1111 1010 1010 1010 1101 0000
; F    A    A    A    F    0
;1111 1010 1010 1010 1111 0000
; F    A    A    A    C    8
;1111 1010 1010 1010 1100 1000
; F    A    A    A    E    8
;1111 1010 1010 1010 1110 1000
; F    A    A    A    D    8
;1111 1010 1010 1010 1101 1000
; F    A    A    A    F    8
;1111 1010 1010 1010 1111 1000
```

```
; F    A    A    A    C    4
;1111 1010 1010 1010 1100 0100
; F    A    A    A    E    4
;1111 1010 1010 1010 1110 0100
; F    A    A    A    D    4
;1111 1010 1010 1010 1101 0100
; A    A    A    A    F    4
;1010 1010 1010 1010 1111 0100
; A    A    A    A    C    C
;1010 1010 1010 1010 1100 1100
; A    A    A    A    A    A
;1010 1010 1010 1010 1010 1010
; A    A    A    A    A    9
;1010 1010 1010 1010 1010 1001
; A    A    A    A    A    B
;1010 1010 1010 1010 1010 1011


JEDEC_TABLE:

DB  0FAH,0AAH,0C0H

DB  0FAH,0AAH,0E0H

DB  0FAH,0AAH,0D0H

DB  0FAH,0AAH,0F0H

DB  0FAH,0AAH,0C8H

DB  0FAH,0AAH,0E8H

DB  0FAH,0AAH,0D8H

DB  0FAH,0AAH,0F8H

DB  0FAH,0AAH,0C4H

DB  0FAH,0AAH,0E4H

DB  0FAH,0AAH,0D4H

DB  0AAH,0AAH,0F4H

DB  0AAH,0AAH,0CCH

DB  0AAH,0AAH,0AAH

DB  0AAH,0AAH,0A9H

DB  0AAH,0AAH,0ABH


END
; END OF THE CODE SEGMENT FOR THE ispGDS22.
; USER CODE CAN BE APPENDED FROM THIS POINT FORWARD.
```

LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com

November 1996