Scenix™
SX Cross Assembler
User's Manual

Lit. No.: SXL-UM02-03

 ©2000 Scenix Semiconductor, Inc. All rights reserved.

Revision History

REVISION	RELEASE DATE	SUMMARY OF CHANGES
1.0	July 14, 1999	Initial Release
1.1	May 15, 2000	Updated to reflect latest SX devices
1.2	August 30, 2000	Updated to support SASM vl. 45.5 and higher revisions

©2000 Scenix Semiconductor, Inc. All rights reserved. No warranty is provided and no liability is assumed by Scenix Semiconductor with respect to the accuracy of this documentation or the merchantability or fitness of the product for a particular application. No license of any kind is conveyed by Scenix Semiconductor with respect to its intellectual property or that of others. All information in this document is subject to change without notice.

Scenix Semiconductor products are not authorized for use in life support systems or under conditions where failure of the product would endanger the life or safety of the user, except when prior written approval is obtained from Scenix Semiconductor.

Scenix™ and the Scenix logo are trademarks of Scenix Semiconductor, Inc.
All other trademarks mentioned in this document are property of their respective companies.

Scenix, Inc., 1330 Charleston Rroad, Mountain View, CA 94043 USA
Telephone: +1 650 210 1500, Web site: hhtp://www.scenix.com

Contents

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

List of Tables

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Chapter 1

Overview

1.1 Introduction

This User's Manual describes the SASM Cross Assembler for the SX communications controllers from Scenix.

The manual explains how to invoke and use SASM. Topics include program structure, directives, macros and file outputs. A summary on the SX basic instruction set is also given.

SASM Cross Assembler is a software development tool that accepts the SX symbolic assembly language as input and translates it into object codes under the MS-DOS operating system on the IBM PC or compatible systems.

1.2 Main Features

- Translates programs (source code) written in SX Assembly language to machine executable code (object code) on IBM PC or compatibles running MS-DOS version 3.0 or higher
- Generates object code for SX communications controllers including the SX18/20/28AC, and SX48/52BD devices using four different formats: three Intel hex formats (INHX8M, INHX16, INHX8S) binary format, and IEEE695 format.
- Provides MACRO and conditional assembly capabilities
- Supports Hex, Decimal (default) and Octal source and listing formats

1.3 Invoking SASM

Use an editor of your choice to create an ASM source file. Assemble this source file by typing the following at the command prompt of the directory where SASM.EXE resides:

SASM [options] file[.asm] [Enter]

where file = source file name

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Tables1-1 shows the summary of options specified at the command prompt.

Table1-1 Options Summary

Opt	Arguments	Description	Default
/C	SX|PARALLAX|ALL|	Compiler Mode	PARALLAX
/E		Extensions for various tool environments	NONE
/F	[INHX8M|INHX8S|INHX16|INHX32|BIN16| IEEE695]	Output Format	INHX8M
/H or /?		Display Help Message	
/I	Turn on case sensitivity	Symbols	Off
/L		Listing File	NOPAGE
/P	[SX18|SX18AC|PINS18|SX20|SX20AC| PINS20|SX28|SX28AC|PINS28|SX48| SX48AC|PINS48|SX52|SX52AC|PINS52|]	Processor Type	SX18AC
/R	[HEX|BIN|DEC|OCT|D|B|O|H]	Radix	DEC
/T	[TABWIDTH]	Tab Width	8
/W	[0|1|2]	Warning Level	1
/Z		Disable path	

NOTES: 1. To eliminate comments (e.g. crossing page boundary) from the list files, set warning to a higher level. For example, set /W to 2.

- /W 0 will include all comments, warning errors and severe errors.
- /W 1 will include warning errors and severe erros.
- /W 2 will include severe errors only.

2. It is recommended to set the processor type inside the main program rather than have it defined on-line during compilation. That is, include the following line in the .ASM file:

```
DEVICE      SX18AC
OR
DEVICE      PINS18
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

1.3.1 Compiler Mode

Command: /C

Arguments: SX|PARALLAX|ALL|

Description The assembler can handle three sets of mnemonics. This option chooses the specific collection of mnemonics to be recognized. It may take any of the values `SX', `PARALLAX',and `ALL'.

Default: ‘/C PARALLAX’

1.3.2 Extensions for Various Tool Environments

Command: /E

Arguments:

Description `/E NOHAU' can be used to cause the format of the logged error messages to use `#' characters to delimit the fields, and to write the error log to a file name `cmperr.log' in the current directory regardless of the name of the source file.

Default: ‘/E NONE

1.3.3 Output Format

Command: /F

Arguments: [INHX8M|INHX8S|INHX16|INHX32|BIN16|IEEE695]

Description The assembler can generate a binary file, several formats of hex files, or an IEEE-695 format object file. This option chooses the output format. It may take any of the values `BIN16', `INHX16', `INHX8M', `INHX8S', `INHX32', or `IEEE695'.

Default: ‘/F INHX8M’

1.3.4 Display Help Message

Command: /H or /?

Arguments:

Description Display the help screen and exit.

Default:

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

1.3.5 Case Independant Symbols

Command: /I

Arguments: Turn on case sensitivity

Description This option is “on” by default and there is no documented option to turn it “off”.

Default: Off

1.3.6 Listing File

Command: /L

Arguments: Use `/L NONE' to disable the listing.

Description This option takes a keyword indicating whether a listing file is desired, and whether it has page headers and form feeds. Use `/L PAGE' to produce a listing with page headers and form feeds. By default there are 51 source lines per page, which can be modified with the LIST directive. Use `/L NOPAGE' to produce a listing a listing with no page headers or form feeds.

Default: `/L NOPAGE'

1.3.7 Target Processor

Command: /P

Arguments: [SX18|SX18AC|PINS18|SX20|SX20AC|PINS20|SX28|SX28AC|PINS28|SX48|SX48BD|PINS48|SX52|SX52BD|PINS52|]

Description This option selects the default target processor, which may be overridden by the DEVICE directive. Choose one of `SX18', `SX18AC', `PINS18', `SX20', `SX20AC', `PINS20', `SX28', `SX28AC', `PINS28', `SX48', `SX48AC', `PINS48', `SX52', `SX52AC', or `PINS52'.

Default: ‘/P SX18AC’

1.3.8 Radix

Command: /R

Arguments: [HEX|BIN|DEC|OCT|D|B|O|H]

Description This option selects the default radix used to interpret numeric constants which do not specify a radix. Choose one of `DEC', `BIN', `OCT', `HEX', `D', `B', `O', or `H'.

Default: `/R DEC'

© 2000 Scenix Semiconductor, Inc. All rights reserved.

1.3.9 Default Tab Width

Command: /T

Arguments: [TABWIDTH]

Description This option sets the assumed width of a tab character, and may be set to any positive integer less than 20.

Default: `/T 8'

1.3.10 Error Level

Command: /W

Arguments: [0|1|2]

Description This option controls the number of comments, warnings, and error messages which appear. Set it to 0 for lots of output, 1 for warnings and errors only, or 2 for errors only.

Default: `/W 1'

1.3.11 Disable Full Pathnames in .MAP File

Command: /Z

Arguments:

Description Version 1.45.5 or higher of SASM defaults to `/I /CPARALLAX /FINHX8M /PSX18 / RDEC /T8 /W1 /LNOPAGE'. The `/F', `/L', `/P', and `/R' options may also be specified in the source file with the `LIST' directive.

Default:

1.4 Source Files

The source file is the file to be assembled. SASM assumes all source files to have .ASM extensions. If not, the entire filename, including extension, has to be provided at the command line.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

1.5 Output Files

SASM Assembler outputs different files with the following extensions::

HEX	-	Intel 8-bit merged Hex file (*Default file format)
OBJ	-	Binary object file if /F BIN is used
HXH/HXL	-	Address/Data pairs for high-order and low-order 8 bits (only when INHX8S format is selected as output)
LST	-	Program listing file
SYM	-	Symbol file used for defining watch variables and setting break point at label address. Used for symbolic or source-level debugging.
MAP	-	Map file used for source-level debugging
ERR	-	Error message file
SXE	-	IEEE695 output file format if /F IEEE695 option is used

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Chapter 2

Program Structure

2.1 Source Program

The structure of a source program consists of one or more statements and comments. Each statement can be a combination of mnemonics, directives, macros, symbols, expressions and/or constants.

Example of an assembly program:

```
;*****************************************************************************
; Copyright © [11/21/1999] Scenix Semiconductor, Inc. All rights reserved.
;
; Scenix Semiconductor, Inc. assumes no responsibility or liability for
; the use of this [product, application, software, any of these products].
; Scenix Semiconductor conveys no license, implicitly or otherwise, under
; any intellectual property rights.
; Information contained in this publication regarding (e.g.: application,
; implementation) and the like is intended through suggestion only and may
; be superseded by updates. Scenix Semiconductor makes no representation
; or warranties with respect to the accuracy or use of these information,
; or infringement of patents arising from such use or otherwise.
;*****************************************************************************
;
; Filename:   vpg_UART_1_04.src
;
; Authors:    Chris Fogelklou
;             Applications Engineer
;             Scenix , Inc.
;
; Program Description:
;
;                   Virtual Peripherals Guidelines:
;             Example source code, running at 50MHz, with just a transmit
;             and receive UART.  The code implements UART in software for baud rates of
;             1200,2400,4800,9600,19200,57600 bps depending on the rate selected,it can
;             be selected to work at interrupt rate of 4.32us.
;
; Interface Pins:
;
;             rs232RxPin    equ    ra.2          ;UART receive input
;             rs232TxPin    equ    ra.3          ;UART transmit output
;             rts_pin       equ    ra.0          ;UART 1 RTS input
;             cts_pin       equ    ra.1          ;UART 1 CTS outpu
;
;*****************************************************************************
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
;*****************************************************************************************
; Target SX
; Uncomment one of the following lines to choose the SX18AC,SX20AC,SX28AC,SX48BD, SX52BD.
;*****************************************************************************************

;SX18_20
;SX28AC
SX48_52

;*****************************************************************************************
; Assembler Used
; Uncomment the following line if using the Parallax SX-Key assembler. SASM assembler
; enabled by default.
;*****************************************************************************************
;SX_Key

;*****************************************************************************************
; Uncomment one of the following to run the uart vp at the required baud rate
;*****************************************************************************************
;baud1200           ;baud rate of 1.2 Kbps
;baud2400
;baud4800           ;baud rate of 4.8 Kbps
baud9600            ;baud rate of 9.6 kbps
;baud1920           ;baud rate of 19.2kbps
;baud5760           ;baud rate of 57.6kbps


;*****************************************************************************************
;                   Assembler directives
;
;      High speed external osc, turbo mode, 8-level stack, and extended option reg.
;      SX18/20/28 - 4 pages of program memory and 8 banks of RAM enabled by default.
;      SX48/52 - 8 pages of program memory and 16 banks of RAM enabled by default.
;*****************************************************************************************

IFDEF SX_Key                            ;SX-Key Directives

  IFDEF SX18_20                         ;SX18AC or SX20AC device directives for SX-Key
       device       SX18L,oschs2,turbo,stackx_optionx
  ENDIF

  IFDEF SX28AC                          ;SX28AC device directives for SX-Key
       device       SX28L,oschs2,turbo,stackx_optionx
  ENDIF


  IFDEF SX48_52                         ;SX48/52/BD device directives for SX-Key
       device       oschs2
  ENDIF
       freq   50_000_000

ELSE                                    ;SASM Directives

  IFDEF SX18_20                         ;SX18AC or SX20AC device directives for SASM
       device       SX18,oschs2,turbo,stackx,optionx
```

```
  ENDIF

  IFDEF SX28AC                          ;SX28AC device directives for SASM
       device       SX28,oschs2,turbo,stackx,optionx
  ENDIF

  IFDEF SX48_52                         ;SX48BD or SX52BD device directives for SASM
       device SX52,oschs2
  ENDIF

ENDIF
                    id     '1UART_VP'    ;
                    reset  resetEntry    ; set reset vector


       ;*****************************************************************************
       ;-------------------------------------Macro's-----------------------------------
       ; Macro: _bank
       ; Sets the bank appropriately for all revisions of SX.
       ;
       ; This is required since the bank instruction has only a 3-bit operand, it cannot
       ; be used to access all 16 banks of the SX48/52.  FSR.7 (SX48/52bd production
       ; release) needs to be set appropriately, depending on the bank address being
       ; accessed. This macro fixes this.
       ;
       ; So, instead of using the bank instruction to switch between banks, use _bank
       ; instead.
       ;*****************************************************************************

_bank  macro  1
  noexpand
       bank   \1
       IFDEF SX48_52
          IF \1 & %10000000       ;SX48BD and SX52BD (production release) bank instruction
  expand
       setb   fsr.7               ;modifies FSR bits 4,5 and 6. FSR.7 needs to be set by
                                  ; software.
  noexpand
          ELSE
  expand
       clrb   fsr.7
  noexpand
           ENDIF
       ENDIF
 endm

       ;*****************************************************************************
       ; Macro: _mode
       ; Sets the MODE register appropriately for all revisions of SX.
       ;
       ; This is required since the MODE (or MOV M,#) instruction has only a 4-bit operand.
       ; The SX18/20/28AC use only 4 bits of the MODE register, however the SX48/52BD have
       ; the added ability of reading or writing some of the MODE registers, and therefore
       ; use 5-bits of the MODE register. The  MOV M,W instruction modifies all 8-bits of
       ; the MODE register, so this instruction must be used on the SX48/52BD to make sure
       ; the MODE register is written with the correct value.  ; This macro fixes this.
       ;
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
       ; So, instead of using the MODE or MOV M,# instructions to load the M register, use
       ; _mode instead.
       ;*****************************************************************************

_mode  macro  1

  noexpand
       IFDEF SX48_52
  expand
             mov    w,#\1        ;loads the M register correctly for the SX48BD and SX52BD
             mov    m,w
  noexpand
       ELSE
  expand
             mov    m,#\1        ;loads the M register correctly for the SX18AC, SX20AC
                                 ;and SX28AC
  noexpand
       ENDIF

endm

       ;*****************************************************************************
        ; INCP/DECP macros for incrementing/decrementing pointers to RAM
        ; used to compensate for incompatibilities between SX28AC and SX52BD
       ;*****************************************************************************


 INCP macro 1                    ; Increments a pointer to RAM
             inc    \1
       IFNDEF SX48_52
             setb  \1.4          ; If SX18 or SX28AC,keep bit 4 of the pointer = 1
       ENDIF                     ; to jump from $1f to $30,etc
endm

DECP macro 1                     ; Decrements a pointer to RAM
       IFDEF  SX48_52
             dec    \1
       ELSE
             clrb   \1.4         ; If SX18 or SX28AC, forces rollover to next bank
             dec    \1           ; if it rolls over. (skips banks with bit 4 = 0)
             setb   \1.4         ; Eg: $30 ---> $20 ---> $1f ---> $1f
       ENDIF                     ; AND: $31 ---> $21 ---> $20 ---> $30
endm

;*****************************************************************************
       ; Error generating macros
       ; Used to generate an error message if the label is intentionally moved into the
       ; second page.
       ; Use for lookup tables.
;*****************************************************************************

tableStart   macro  0            ; Generates an error message if code that MUST be in
                                 ; the first half of a page is moved into the second half
       if $ & $100
             ERROR      'Must be located in the first half of a page.'
       endif
```

```
endm

tableEnd     macro  0            ; Generates an error message if code that MUST be in
                                  ; the first half of a page is moved into the second half

       if $ & $100
              ERROR      'Must be located in the first half of a page.'
       endif
endm

;*****************************************************************************************
;-----------------------------------Memory Organization---------------------------------
;*****************************************************************************************


;*****************************************************************************************
;---------------------------Data Memory address definitions-----------------------------
; These definitions ensure the proper address is used for banks 0 - 7 for 2K SX devices
; (SX18/20/28) and 4K SX devices (SX48/52).
;*****************************************************************************************
****

IFDEF SX48_52

global_org    =      $0A
bank0_org     =      $00
bank1_org     =      $10
bank2_org     =      $20
bank3_org     =      $30
bank4_org     =      $40
bank5_org     =      $50
bank6_org     =      $60
bank7_org     =      $70

ELSE

global_org    =      $08
bank0_org     =      $10
bank1_org     =      $30
bank2_org     =      $50
bank3_org     =      $70
bank4_org     =      $90
bank5_org     =      $B0
bank6_org     =      $D0
bank7_org     =      $F0

ENDIF

;*****************************************************************************************
;------------------------------- Global Register definitions---------------------------
; NOTE: Global data memory starts at $0A on SX48/52 and $08 on SX18/20/28.
;*****************************************************************************************

       org           global_org

flags0               equ    global_org + 0      ; stores bit-wise operators like flags
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
                                            ; and function-enabling bits (semaphores)
;-----------------------------VP: RS232 Receive-----------------------------------------
       rs232RxFlag   equ    flags0.0;indicates the reception of a bit from the UART

isrTemp0            equ    global_org + 1      ; Interrupt Service Routine's temp register.
                                               ; Don't use this register in the mainline.
localTemp0          equ    global_org + 2      ; temporary storage register
                                               ; Used by first level of nesting
                                               ; Never guaranteed to maintain data
localTemp1          equ    global_org + 3      ; temporary storage register
                                               ; Used by second level of nesting
                                               ; or when a routine needs more than one
                                               ; temporary global register.
localTemp2          equ    global_org + 4      ; temporary storage register
                                               ; Used by third level of nesting or by
                                               ; main loop routines that need a loop
                                               ; counter, etc.

;*******************************************************************************
;--------------------------- RAM Bank Register definitions--------------------------------
;*******************************************************************************

       ;*****************************************************************************
       ; Bank 0
       ;*****************************************************************************

              org     bank0_org

bank0         =      $

       ;*****************************************************************************
       ; Bank 1
       ;*****************************************************************************

              org     bank1_org


bank1                =      $
rs232TxBank          =      $      ;UART bank
rs232Txhigh          ds     1      ;hi byte to transmit
rs232Txlow           ds     1      ;low byte to transmit
rs232Txcount         ds     1      ;number of bits sent
rs232Txdivide        ds     1      ;xmit timing (/16) counter
rs232Txflag          ds     1

rs232RxBank          =      $
rs232Rxcount         ds     1      ;number of bits received
rs232Rxdivide        ds     1      ;receive timing counter
rs232Rxbyte          ds     1      ;buffer for incoming byte
string               ds     1      ;used by send_string to store the address in memory
rs232byte            ds     1      ;used by serial routines
hex                  ds     1

MultiplexBank        =      $
isrMultiplex         ds     1
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
        ;*****************************************************************************
        ; Bank 2
        ;*****************************************************************************

                org     bank2_org

bank2           =       $

        ;*****************************************************************************
        ; Bank 3
        ;*****************************************************************************

                org     bank3_org

bank3           =       $

        ;*****************************************************************************
        ; Bank 4
        ;*****************************************************************************

                org     bank4_org

bank4           =       $

        ;*****************************************************************************
        ; Bank 5
        ;*****************************************************************************

                org     bank5_org

bank5           =       $

        ;*****************************************************************************
        ; Bank 6
        ;*****************************************************************************

                org     bank6_org

bank6           =       $

        ;*****************************************************************************
        ; Bank 7
        ;*****************************************************************************

                org     bank7_org

bank7           =       $


IFDEF SX48_52

        ;*****************************************************************************
        ; Bank 8
        ;*****************************************************************************
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
              org    $80           ;bank 8 address on SX52

bank8         =      $

        ;*****************************************************************************
        ; Bank 9
        ;*****************************************************************************

              org    $90           ;bank 9 address on SX52

bank9         =      $

        ;*****************************************************************************
        ; Bank A
        ;*****************************************************************************

              org    $A0           ;bank A address on SX52

bankA         =      $

        ;*****************************************************************************
        ; Bank B
        ;*****************************************************************************

              org    $B0           ;bank B address on SX52

bankB         =      $

        ;*****************************************************************************
        ; Bank C
        ;*****************************************************************************

              org    $C0           ;bank C address on SX52

bankC         =      $

        ;*****************************************************************************
        ; Bank D
        ;*****************************************************************************

              org    $D0           ;bank D address on SX52

bankD         =      $


        ;*****************************************************************************
        ; Bank E
        ;*****************************************************************************

              org    $E0           ;bank E address on SX52

bankE         =      $

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

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
        ; Bank F
        ;*****************************************************************************

                org     $F0             ;bank F address on SX52

bankF           =       $

ENDIF

;*****************************************************************************
;---------------------------------- Port Assignment----------------------------------
;*****************************************************************************

RA_latch        equ     %00001000       ;SX18/20/28/48/52 port A latch init
RA_DDIR         equ     %11110111       ;SX18/20/28/48/52 port A DDIR value
RA_LVL          equ     %00000000       ;SX18/20/28/48/52 port A LVL value
RA_PLP          equ     %00001100       ;SX18/20/28/48/52 port A PLP value

RB_latch        equ     %00000000       ;SX18/20/28/48/52 port B latch init;intial value affter
                                        ;reset
RB_DDIR         equ     %11111111       ;SX18/20/28/48/52 port B DDIR value;0=Output,1=Input
RB_ST           equ     %11111111       ;SX18/20/28/48/52 port B ST value;0=Enable,1=Disable
RB_LVL          equ     %00000000       ;SX18/20/28/48/52 port B LVL value;0=CMOS,1=TTL
RB_PLP          equ     %00000000       ;SX18/20/28/48/52 port B PLP value;0=Enable,1=Disable

RC_latch        equ     %00000000       ;SX18/20/28/48/52 port C latch init;intial value affter
                                        ;reset
RC_DDIR         equ     %11111111       ;SX18/20/28/48/52 port C DDIR value;0=Output,1=Input
RC_ST           equ     %11111111       ;SX18/20/28/48/52 port C ST value;0=Enable,1=Disable
RC_LVL          equ     %00000000       ;SX18/20/28/48/52 port C LVL value;0=CMOS,1=TTL
RC_PLP          equ     %00000000       ;SX18/20/28/48/52 port C PLP value;0=Enable,1=Disable

IFDEF SX48_52

RD_latch        equ     %00000000       ;SX48/52 port D latch init;intial value affter reset
RD_DDIR         equ     %11111111       ;SX48/52 port D DDIR value;0=Output,1=Input
RD_ST           equ     %11111111       ;SX48/52 port D ST value;0=Enable,1=Disable
RD_LVL          equ     %00000000       ;SX48/52 port D LVL value;0=CMOS,1=TTL
RD_PLP          equ     %00000000       ;SX48/52 port D PLP value;0=Enable,1=Disable

RE_latch        equ     %00000000       ;SX48/52 port E latch init;intial value affter reset
RE_DDIR         equ     %11111111       ;SX48/52 port E DDIR value;0=Output,1=Input
RE_ST           equ     %11111111       ;SX48/52 port E ST value;0=Enable,1=Disable
RE_LVL          equ     %00000000       ;SX48/52 port E LVL value;0=CMOS,1=TTL
RE_PLP          equ     %00000000       ;SX48/52 port E PLP value;0=Enable,1=Disable

ENDIF

;*****************************************************************************
;---------------------------------- Pin Definitions----------------------------------
;*****************************************************************************

rs232RTSpin     equ     ra.0            ;UART RTS input
rs232CTSpin     equ     ra.1            ;UART CTS output
rs232Rxpin      equ     ra.2            ;UART receive input
rs232Txpin      equ     ra.3            ;UART transmit output
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
;*****************************************************************************
;-------------------------------- Program constants----------------------------------
;*****************************************************************************

_enter        equ    13           ; ASCII value for carridge return
_linefeed     equ    10           ; ASCII value for a line feed

;*****************************************************************************
;      UART Constants values
;*****************************************************************************

intPeriod           = 217

UARTfs              = 230400

Num                 = 4

IFDEF baud1200
       UARTBaud     = 1200
ENDIF

IFDEF baud2400
       UARTBaud     = 2400
ENDIF

IFDEF baud4800
       UARTBaud     = 4800
ENDIF

IFDEF baud9600
       UARTBaud     = 9600
ENDIF

IFDEF baud1920
       UARTBaud     = 19200
ENDIF

IFDEF baud5760
       UARTBaud     = 57600
ENDIF

UARTDivide          = (UARTfs/(UARTBaud*Num))
UARTStDelay         = UARTDivide +(UARTDivide/2)+1


IFDEF SX48_52

;*****************************************************************************
; SX48BD/52BD Mode addresses
; *On SX48BD/52BD, most registers addressed via mode are read and write, with the
; exception of CMP and WKPND which do an exchange with W.
;*****************************************************************************
;-------------------------------- Timer (read) addresses-----------------------------
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
TCPL_R       equ    $00    ;Read Timer Capture register low byte
TCPH_R       equ    $01    ;Read Timer Capture register high byte
TR2CML_R     equ    $02    ;Read Timer R2 low byte
TR2CMH_R     equ    $03    ;Read Timer R2 high byte
TR1CML_R     equ    $04    ;Read Timer R1 low byte
TR1CMH_R     equ    $05    ;Read Timer R1 high byte
TCNTB_R      equ    $06    ;Read Timer control register B
TCNTA_R      equ    $07    ;Read Timer control register A

;---------------------------------- Exchange addresses-----------------------------------

CMP          equ    $08    ;Exchange Comparator enable/status register with W
WKPND        equ    $09    ;Exchange MIWU/RB Interrupts pending with W

;----------------------------------port setup (read) addresses---------------------------

WKED_R       equ    $0A    ;Read MIWU/RB Interrupt edge setup, 1 = falling, 0 = rising
WKEN_R       equ    $0B    ;Read MIWU/RB Interrupt edge setup, 0 = enabled, 1 = disabled
ST_R         equ    $0C    ;Read Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
LVL_R        equ    $0D    ;Read Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
PLP_R        equ    $0E    ;Read Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
DDIR_R       equ    $0F    ;Read Port Direction

;----------------------------------Timer (write) addresses-------------------------------

CLR_TMR      equ    $10    ;Resets 16-bit Timer
TR2CML_W     equ    $12    ;Write Timer R2 low byte
TR2CMH_W     equ    $13    ;Write Timer R2 high byte
TR1CML_W     equ    $14    ;Write Timer R1 low byte
TR1CMH_W     equ    $15    ;Write Timer R1 high byte
TCNTB_W      equ    $16    ;Write Timer control register B
TCNTA_W      equ    $17    ;Write Timer control register A

;-----------------------------------Port setup (write) addresses-------------------------

WKED_W       equ    $1A    ;Write MIWU/RB Interrupt edge setup, 1 = falling, 0 = rising
WKEN_W       equ    $1B    ;Write MIWU/RB Interrupt edge setup, 0 = enabled, 1 = disabled
ST_W         equ    $1C    ;Write Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
LVL_W        equ    $1D    ;Write Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
PLP_W        equ    $1E    ;Write Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
DDIR_W       equ    $1F    ;Write Port Direction

ELSE

;*******************************************************************************************
; SX18AC/20AC/28AC Mode addresses
; *On SX18/20/28, all registers addressed via mode are write only, with the exception of
; CMP and WKPND which do an exchange with W.
;*******************************************************************************************

;----------------------------------------Exchange addresses-----------------------------

CMP          equ    $08    ;Exchange Comparator enable/status register with W
WKPND        equ    $09    ;Exchange MIWU/RB Interrupts pending with W
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
;----------------------------------------Port setup (read) addresses---------------------

WKED_W       equ    $0A    ;Write MIWU/RB Interrupt edge setup, 1 = falling, 0 = rising
WKEN_W       equ    $0B    ;Write MIWU/RB Interrupt edge setup, 0 = enabled, 1 = disabled
ST_W         equ    $0C    ;Write Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
LVL_W        equ    $0D    ;Write Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
PLP_W        equ    $0E    ;Write Port Schmitt Trigger setup, 0 = enabled, 1 = disabled
DDIR_W       equ    $0F    ;Write Port Direction

ENDIF

;*******************************************************************************************
;-----------------------------------------------Program memory ORG defines-----------------
;*******************************************************************************************

INTERRUPT_ORG       equ    $0     ; Interrupt must always start at location zero
RESETENTRY_ORG      equ    $1FB   ; The program will jump here on reset
SUBROUTINES_ORG     equ    $200   ; The subroutines are in this location
STRINGS_ORG         equ    $300   ; The strings are in the location $300
PAGE3_ORG           equ    $400   ; Page 3 is empty
MAINPROGRAM_ORG     equ    $600   ; The main program is in the lastpage of program memory


;*******************************************************************************************
      org    INTERRUPT_ORG        ; First location in program memory.
;*******************************************************************************************

;*******************************************************************************************
;-----------------------------------------------Interrupt Service Routine--------------------
; Note 1: The interrupt code must always originate at address $0.
;        Interrupt Frequency = (Cycle Frequency / -(retiw value))
;        For example: With a retiw value of -217 and an oscillator frequency
;        of 50MHz, this code runs every 4.32us.
; Note 2: Mode Register 'M' is not saved in SX 28 but saved in SX 52 when an Interrupt
;        occurs. If the code is to run on a SX 28 and 'M' register is used in the ISR,
;        then the 'M' register has to be saved at the Start of ISR and restored at the
;        End of ISR.
;*******************************************************************************************

             org     $0
interrupt                        ;3

;*******************************************************************************************
; Interrupt
; Interrupt Frequency = (Cycle Frequency / -(retiw value))  For example:
; With a retiw value of -217 and an oscillator frequency of 50MHz, this code runs
; every 4.32us.
;*******************************************************************************************

;*******************************************************************************************
;-----------------------------------------------VP:VP Multitasker---------------------------
; Virtual Peripheral Multitasker : up to 16 individual threads, each running at the
; (interrupt rate/16). Change them below:
; Input variable(s): isrmultiplex: variable used to choose threads
; Output variable(s): None,executes the next thread
```

```
; Variable(s) affected: isrmultiplex
; Flag(s) affected: None
; Program Cycles: 9 cycles (turbo mode)
;*****************************************************************************

               _bank          Multiplexbank        ;
               inc            isrMultiplex         ; toggle interrupt rate
               mov            w,isrMultiplex       ;

;*****************************************************************************
; The code between the tableStart and tableEnd statements MUST be completely within the first
; half of a page. The routines it is jumping to must be in the same page as this table.
;*****************************************************************************

         tableStart                                ; Start all tables with this macro
               jmp            pc+w                 ;
               jmp            isrThread1           ;
               jmp            isrThread2           ;
               jmp            isrThread3           ;
               jmp            isrThread4           ;
               jmp            isrThread1           ;
               jmp            isrThread5           ;
               jmp            isrThread6           ;
               jmp            isrThread7           ;
               jmp            isrThread1           ;
               jmp            isrThread8           ;
               jmp            isrThread9           ;
               jmp            isrThread10          ;
               jmp            isrThread1           ;
               jmp            isrThread11          ;
               jmp            isrThread12          ;
               jmp            isrThread13          ;
         tableEnd                                  ; End all tables with this macro.

;*****************************************************************************
;VP: VP Multitasker
; ISR TASKS
;*****************************************************************************

isrThread1                                         ; Serviced at ISR rate/4

;-----------------------------------------VP: RS232 Transmit-------------------------

;*****************************************************************************
; Virtual Peripheral: Universal Asynchronous Receiver Transmitter (UART)
; These routines send and receive RS232 serial data, and are currently
; configured (though modifications can be made) for the popular
; "No parity-checking, 8 data bit, 1 stop bit" (N,8,1) data format.
;
; RECEIVING: The rs232Rxflag is set high whenever a valid byte of data has been
; received and it is the calling routine's responsibility to reset this flag
; once the incoming data has been collected.
;
; TRANSMITTING: The transmit routine requires the data to be inverted
; and loaded (rs232Txhigh+rs232Txlow) register pair (with the inverted 8 data bits
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
; stored in rs232Txhigh and rs232Txlow bit 7 set high to act as a start bit). Then
; the number of bits ready for transmission (10=1 start + 8 data + 1 stop)
; must be loaded into the rs232Txcount register. As soon as this latter is done,
; the transmit routine immediately begins sending the data.
; This routine has a varying execution rate and therefore should always be
; placed after any timing-critical virtual peripherals such as timers,
; adcs, pwms, etc.
; Note: The transmit and receive routines are independent and either may be
;       removed, if not needed, to reduce execution time and memory usage,
;       as long as the initial "BANK serial" (common) instruction is kept.
;       Input variable(s) : rs232Txlow (only high bit used), rs232Txhigh, rs232Txcount
;       Output variable(s) : rs232Rxflag, rs232Rxbyte
;       Variable(s) affected : rs232Txdivide, rs232Rxdivide, rs232Rxcount
;       Flag(s) affected : rs232Rxflag
;       Variable(s) affected : Txdivide
;       Program cycles: 17 worst case
;       Variable Length?  Yes.
;*****************************************************************************************

rs232Transmit
             _bank        rs232TxBank         ;2 switch to serial register bank

             decsz        rs232Txdivide       ;1 only execute the transmit routine
             jmp          :rs232TxOut         ;1
             mov          w,#UARTDivide       ;1 load UART baud rate (50MHz)
             mov          rs232Txdivide,w     ;1
             test         rs232Txcount        ;1 are we sending?
             snz                              ;1
             jmp          :rs232TxOut         ;1

:txbit       clc                              ;1 yes, ready stop bit
             rr           rs232Txhigh         ;1  and shift to next bit
             rr           rs232Txlow          ;1
             dec          rs232Txcount        ;1 decrement bit counter
             snb          rs232Txlow.6        ;1 output next bit
             clrb         rs232TxPin          ;1
             sb           rs232Txlow.6        ;1
             setb         rs232TxPin          ;1,17
:rs232TxOut


;*****************************************************************************************
;----------------------------------------VP: RS232 Receive--------------------------------
; Virtual Peripheral: Universal Asynchronous Receiver Transmitter (UART)
; These routines send and receive RS232 serial data, and are currently
; configured (though modifications can be made) for the popular
; "No parity-checking, 8 data bit, 1 stop bit" (N,8,1) data format.

; RECEIVING: The rx_flag is set high whenever a valid byte of data has been
; received and it is the calling routine's responsibility to reset this flag
; once the incoming data has been collected.
;       Output variable(s) : rx_flag, rx_byte
;       Variable(s) affected : tx_divide, rx_divide, rx_count
;       Flag(s) affected : rx_flag
;       Program cycles: 23 worst case
```

```
;       Variable Length?  Yes.
;*******************************************************************************

rs232Receive
              _bank         rs232RxBank         ;2
               sb           rs232RxPin          ;1 get current rx bit
               clc                              ;1
               snb          rs232RxPin          ;1
               stc                              ;1
              test          rs232Rxcount        ;1 currently receiving byte?
              sz                                ;1
              jmp           :rxbit              ;1 if so, jump ahead
              mov           w,#9                ;1 in case start, ready 9 bits
              sc                                ;1 skip ahead if not start bit
              mov           rs232Rxcount,w      ;1 it is, so renew bit count
              mov           w,#UARTStDelay      ;1 ready 1.5 bit periods (50MHz)
              mov           rs232RxDivide,w     ;1
:rxbit        decsz         rs232Rxdivide       ;1 middle of next bit?
              jmp           :rs232RxOut         ;1
              mov           w,#UARTDivide       ;1 yes, ready 1 bit period (50MHz)
              mov           rs232Rxdivide,w     ;1
              dec           rs232Rxcount        ;1 last bit?
              sz                                ;1 if not
              rr            rs232Rxbyte         ;1  then save bit
              snz                               ;1 if so,
              setb          rs232RxFlag         ;1,23  then set flag
:rs232RxOut

;*******************************************************************************
;=================================== PUT YOUR OWN VPs HERE==============================
; Virtual Peripheral:
;
;      Input variable(s):
;      Output variable(s):
;      Variable(s) affected:
;      Flag(s) affected:
;*******************************************************************************
;-------------------------------------------------------------------------------
              jmp           isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread2                                  ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
              jmp           isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread3                                  ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
              jmp           isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread4                                  ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
              jmp           isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread5                                  ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
              jmp           isrOut        ; 7 cycles until mainline program resumes execution
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
;-------------------------------------------------------------------------------
isrThread6                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread7                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread8                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread9                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread10                             ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread11                             ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread12                             ; Serviced at ISR rate/16
            jmp         isrOut          ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------
isrThread13                             ; Serviced at ISR rate/16
                                        ; This thread must reload the isrMultiplex register
      _bank         Multiplexbank
      mov           isrMultiplex,#255   ; reload isrMultiplex so isrThread1 will be run on
                                        ; the next interrupt.
      jmp           isrOut              ; 7 cycles until mainline program resumes execution
                                        ; This thread must reload the isrMultiplex register
                                        ; since it is the last one to run in a rotation.
;-------------------------------------------------------------------------------
isrOut

;*******************************************************************************
; Set Interrupt Rate
;*******************************************************************************

isrend
      mov           w,#-intperiod       ;refresh RTCC on return
                                        ;(RTCC = 217-no of instructions executed in the ISR)
      retiw                             ;return from the interrupt

;*******************************************************************************
; End of the Interrupt Service Routine
;*******************************************************************************

;*******************************************************************************
; RESET VECTOR
;*******************************************************************************
```

```
;*******************************************************************************************
;---------------------------------------Reset Entry---------------------------------------
;*******************************************************************************************

       org    RESETENTRY_ORG

resetEntry                                      ; Program starts here on power-up
              page   _resetEntry
              jmp    _resetEntry

;*******************************************************************************************
;---------------------------------------UART Subroutines----------------------------------
;*******************************************************************************************

       org    SUBROUTINES_ORG

;*******************************************************************************************
;      Function      : getbyte
;      INPUTS        : NONE
;      OUTPUTS       : Received byte in rs232Rxbyte
;      Get byte via serial port and echo it back to the serial port
;*******************************************************************************************

getbyte       jnb     rs232RxFlag,$             ; wait till byte is received
              clrb    rs232RxFlag               ; reset the receive flag
              _bank  rs232RxBank                ; switch to rs232 bank

              mov     rs232byte,rs232Rxbyte     ; store byte (copy using W)

              retp

;*******************************************************************************************
;      Function             : sendbyte
;      INPUTS               : 'w' - the byte to be sent via RS-232
;      OUTPUTS              : Outputs The byte via RS-232
;      Send byte via serial port
;*******************************************************************************************

sendbyte      mov           localTemp0,w
              _bank         rs232TxBank

:wait         test          rs232Txcount        ; wait for not busy
              sz
              jmp           :wait               ;

              not           w                   ; ready bits (inverse logic)
              mov           rs232Txhigh,w       ; store data byte
              setb          rs232Txlow.7        ; set up start bit
              mov           w,#10               ; 1 start + 8 data + 1 stop bit
              mov           rs232Txcount,w
              retp                              ; leave and fix page bits

;*******************************************************************************************
;      Function             : sendstring
;                            Care should be taken that the srings are located within program
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
;                          memory locations $300-$3ff as the area
;      INPUTS              : 'w' -the address of a null-terminated string in program memory
;      OUTPUTS             : Outputs the string via RS-232
;      Send string pointed to by address in W register
;*****************************************************************************

sendstring    _bank        rs232TxBank
              mov          localTemp1,w              ; store string address
:loop
              mov          w,#STRINGS_ORG>>8         ; with indirect addressing
              mov          m,w
              mov          w,localTemp1              ; read next string character
              iread                                  ; using the mode register
              test         w                         ; are we at the last char?
              snz                                    ; if not=0, skip ahead
              jmp          :out                      ; yes, leave & fix page bits
              call         sendbyte                  ; not 0, so send character
              _bank        rs232TxBank
              inc          localTemp1                ; point to next character
              jmp          :loop                     ; loop until done

:out          mov          w,#$1F                    ; reset the mode register
              mov          m,w
              retp

;*****************************************************************************
;      Function            : uppercase
;      INPUTS              : byte - the byte to be converted
;      OUTPUTS             : byte - converted byte
;      Convert byte to uppercase.
;*****************************************************************************

uppercase     mov                w,#'a'       ;if byte is lowercase, then skip ahead
              mov          w,rs232byte-w
              sc
              retp
              mov          w,#'a'-'A'          ;change byte to uppercase
              sub          rs232byte,w
              retp                             ;leave and fix page bits

;*****************************************************************************
;      Function            : sendhex
;      INPUTS              : 'w' - the byte to be output
;      OUTPUTS             : Outputs the hex byte via RS-232
;      Output a hex number
;*****************************************************************************

sendhex       mov          localTemp1,w
              swap         wreg
              and          w,#$0f
              call         hextable
              call         sendbyte
              mov          w,localTemp1
              and          w,#$0f
              call         hextable
```

```
             call        sendbyte
             retp

;*****************************************************************************************
;      Function            : gethex
;      Inputs              : None
;      OUTPUTS             : Received HEX value is in 'hex' register.
;      This routine returns with an 8-bit value in the W and in the hex
;      register.  It accepts a hex number from the terminal screen and
;      returns. Remember to write a prompt to the screen before calling get_hex
;*****************************************************************************************

gethex       _bank       rs232RxBank              ;2
             mov         w,#_enterhex
             call        @sendstring
             call        :getvalidhex
             mov         w,rs232byte              ; send the received (good) byte
             call        sendbyte
             swap        localTemp2               ; put the nibble in the upper nibble
             mov         w,localTemp2
             mov         hex,w                    ; of hex register

             call        :getvalidhex
             mov         w,rs232byte              ; send the second received byte
             call        sendbyte
             mov         w,localTemp2
             and         w,#$0f
             or          w,hex
             mov         hex,w
             retp
:getvalidhex

:gh1         clr         localTemp2
             jnb         rs232Rxflag,$            ; get a byte from the terminal
             clrb        rs232Rxflag
             mov         rs232byte,rs232Rxbyte
             call        uppercase                ; uppercase it.

:loop        mov         w,localTemp2             ; get the value at temp (index)
             call        hextable
             xor         w,rs232byte
             snz                                  ; compare it to the received byte
             ret
             inc         localTemp2               ; if they are equal, we have the
             jb          localTemp2.4,:gh1        ; upper nybble.  Continue if not.
             jmp         :loop
             ret

hextable     add         pc,w
             retw        '0'
             retw        '1'
             retw        '2'
             retw        '3'
             retw        '4'
             retw        '5'
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
            retw        '6'
            retw        '7'
            retw        '8'
            retw        '9'
            retw        'A'
            retw        'B'
            retw        'C'
            retw        'D'
            retw        'E'
            retw        'F'

;*****************************************************************************************
org    STRINGS_ORG          ; This label defines where strings are kept in program space.
                            ; all the following strings must be within the same half page of
                            ; theprogram memory for sendstring to work,and they must be
                            ; preceeded by this label.
;*****************************************************************************************

;*****************************************************************************************
;--------------------------------------String Data--------------------------------------
;*****************************************************************************************

;VP: RS232 Transmit

_hello      dw          13,10,'Yup, The UART works!!!',0
_hitSpace   dw          13,10,'Hit Space...',0
_enterhex   dw          13,10,'Enter Hex Value',0

      org   PAGE3_ORG
            jmp         $

;*****************************************************************************************
;---------------------------------------- Main Program ----------------------------------
;       Program execution begins here on power-up or after a reset
;*****************************************************************************************

      org               MAINPROGRAM_ORG

_resetEntry

;*****************************************************************************************
;---------------------------- Initialise all port configuration -------------------------
;*****************************************************************************************

            _mode  ST_W             ;point MODE to write ST register
            mov    w,#RB_ST         ;Setup RB Schmitt Trigger, 0 = enabled, 1 = disabled
            mov    !rb,w
            mov    w,#RC_ST         ;Setup RC Schmitt Trigger, 0 = enabled, 1 = disabled
            mov    !rc,w
 IFDEF SX48_52
            mov    w,#RD_ST         ;Setup RD Schmitt Trigger, 0 = enabled, 1 = disabled
            mov    !rd,w
            mov    w,#RE_ST         ;Setup RE Schmitt Trigger, 0 = enabled, 1 = disabled
            mov    !re,w
 ENDIF
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
            _mode  LVL_W             ;point MODE to write LVL register
            mov    w,#RA_LVL         ;Setup RA CMOS or TTL levels, 1 = TTL, 0 = CMOS
            mov    !ra,w
            mov    w,#RB_LVL         ;Setup RB CMOS or TTL levels, 1 = TTL, 0 = CMOS,0,1
                                     ;= TTL, 2..7 = CMOS
            mov    !rb,w
            mov    w,#RC_LVL         ;Setup RC CMOS or TTL levels, 1 = TTL, 0 = CMOS
            mov    !rc,w
 IFDEF SX48_52
            mov    w,#RD_LVL         ;Setup RD CMOS or TTL levels, 1 = TTL, 0 = CMOS
            mov    !rd,w
            mov     w,#RE_LVL        ;Setup RE CMOS or TTL levels, 1 = TTL, 0 = CMOS
            mov    !re,w
 ENDIF
            _mode  PLP_W             ;point MODE to write PLP register
            mov    w,#RA_PLP         ;Setup RA Weak Pull-up, 0 = enabled, 1 = disabled
            mov    !ra,w
            mov    w,#RB_PLP         ;Setup RB Weak Pull-up, 0 = enabled, 1 = disabled
            mov    !rb,w
            mov    w,#RC_PLP         ;Setup RC Weak Pull-up, 0 = enabled, 1 = disabled
            mov    !rc,w
IFDEF SX48_52
            mov    w,#RD_PLP         ;Setup RD Weak Pull-up, 0 = enabled, 1 = disabled
            mov    !rd,w
            mov    w,#RE_PLP         ;Setup RE Weak Pull-up, 0 = enabled, 1 = disabled
            mov    !re,w
ENDIF
            _mode  DDIR_W            ;point MODE to write DDIR register
            mov    w,#RA_DDIR        ;Setup RA Direction register, 0 = output, 1 = input
            mov    !ra,w
            mov    w,#RB_DDIR        ;Setup RB Direction register, 0 = output, 1 = input
            mov    !rb,w
            mov    w,#RC_DDIR        ;Setup RC Direction register, 0 = output, 1 = input
            mov    !rc,w
IFDEF SX48_52
            mov    w,#RD_DDIR        ;Setup RD Direction register, 0 = output, 1 = input
            mov    !rd,w
            mov    w,#RE_DDIR        ;Setup RE Direction register, 0 = output, 1 = input
            mov    !re,w
ENDIF
            mov    w,#RA_latch       ;Initialize RA data latch
            mov    ra,w
            mov    w,#RB_latch       ;Initialize RB data latch
            mov    rb,w
            mov    w,#RC_latch       ;Initialize RC data latch
            mov    rc,w
IFDEF SX48_52
            mov     w,#RD_latch       ;Initialize RD data latch
            mov     rd,w
            mov     w,#RE_latch       ;Initialize RE data latch
            mov     re,w
ENDIF

;*****************************************************************************
;------------------------------ Clear all Data RAM locations ----------------------------
;*****************************************************************************
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
IFDEF SX48_52                           ;SX48/52 RAM clear routine
             mov    w,#$0a              ;reset all ram starting at $0A
             mov    fsr,w
:zeroRamclr  ind                        ;clear using indirect addressing
             incsz  fsr                 ;repeat until done
             jmp    :zeroRam

             _bank  bank0               ;clear bank 0 registers
             clr    $10
             clr    $11
             clr    $12
             clr    $13
             clr    $14
             clr    $15
             clr    $16
             clr    $17
             clr    $18
             clr    $19
             clr    $1a
             clr    $1b
             clr    $1c
             clr    $1d
             clr    $1e
             clr    $1f

ELSE                                    ;SX18/20/28 RAM clear routine
             clr    fsr                 ;reset all ram banks
:zeroRamsb   fsr.4                      ;are we on low half of bank?
                                        ;If so, don't touch regs 0-7
             setb   fsr.3               ; To clear from 08 - Global Registers
             clr    ind                 ;clear using indirect addressing
             incsz  fsr                 ;repeat until done
             jmp    :zeroRam
ENDIF

;*****************************************************************************
; Initialize program/VP registers
;*****************************************************************************

             _bank  rs232TxBank         ;select rs232 bank
             mov    w,#UARTDivide       ;load Txdivide with UART baud rate
             mov    rs232TXdivide,w

;*****************************************************************************
; Setup and enable RTCC interrupt, WREG register, RTCC/WDT prescaler
;*****************************************************************************

RTCC_ON      =      %10000000           ;Enables RTCC at address $01 (RTW hi)
                                        ;*WREG at address $01 (RTW lo) by default
RTCC_ID      =      %01000000           ;Disables RTCC edge interrupt (RTE_IE hi)
                                        ;*RTCC edge interrupt (RTE_IE lo) enabled by
                                        ;default
RTCC_INC_EXT =      %00100000           ;Sets RTCC increment on RTCC pin transition (RTS hi)
                                        ;*RTCC increment on internal instruction (RTS lo)
is default
```

```
RTCC_FE       =      %00010000           ;Sets RTCC to increment on falling edge (RTE_ES hi)
                                         ;*RTCC to increment on rising edge (RTE_ES lo) is
                                         ;default
RTCC_PS_ON    =      %00000000           ;Assigns prescaler to RTCC (PSA lo)
RTCC_PS_OFF   =      %00001000           ;Assigns prescaler to WDT (PSA hi)
PS_000        =      %00000000           ;RTCC = 1:2, WDT = 1:1
PS_001        =      %00000001           ;RTCC = 1:4, WDT = 1:2
PS_010        =      %00000010           ;RTCC = 1:8, WDT = 1:4
PS_011        =      %00000011           ;RTCC = 1:16, WDT = 1:8
PS_100        =      %00000100           ;RTCC = 1:32, WDT = 1:16
PS_101        =      %00000101           ;RTCC = 1:64, WDT = 1:32
PS_110        =      %00000110           ;RTCC = 1:128, WDT = 1:64
PS_111        =      %00000111           ;RTCC = 1:256, WDT = 1:128

OPTIONSETUPequRTCC_PS_OFF                ;the default option setup for this program.
              mov    w,#OPTIONSETUP      ;setup option register for RTCC interupts enabled
              mov    !option,w           ;and no prescaler.
              jmp    @mainLoop

;*****************************************************************************************
;------------------------------------ MAIN PROGRAM CODE --------------------------------
;*****************************************************************************************

mainLoop
              mov    w,#_hitSpace        ; Send prompt to terminal at UART rate
              call   @sendstring

:loop
              call   @getbyte
              cjne   rs232Rxbyte,#' ',:loop     ; just keep looping until user
                                                ; hits the space bar
              mov    w,#_hello                  ; When space bar hit, send out string.
              call   @sendstring
              jmp    :loop

;*****************************************************************************************
END           ;End of program code
;*****************************************************************************************
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

2.2 Assembler Source Line Format

The general format for a program source line is as followed:

[<Lable1>] <Mnemonic> [<Operand>] [<Comment>]

2.2.1 Label

The optional label field, if present, begins at column one of the source line, and is terminated by the first white space (a space, tab, or end-of-line character). A label may be the only field in a statement. Labels are generally used as a symbolic reference to program memory locations in the source code.

A label consists of 1 to 32 characters. It must begin with a letter, and underscore ('_'), or colon (':'), and may contain any combination of letters, digits, and underscores. A user-defined label may not be a reserved word.

A label may define a symbolic name for a program address, a data address, a macro, or an arbitrary 32-bit value. If used as a program address, a label may be either global or local. A global label must be unique in the entire program. A local label is written with an initial colon (':') character, and must be unique over the set of lines extending from the immediately preceeding global label to the next global label. Local labels will appear in the symbol table concatenated to the name of the immediately preceeding global label.

For example:

```
count     equ     $30
          org     $100
          reset   main
main      mov     count,#10
:loop     call    blink
          djnz    count,:loop
          sleep
blink                       ;define a blink function here
          ret
```

This routine defines labels 'count', 'main', 'main:loop', and 'blink'. The label 'count' refers to a data address. The global label 'main' refers to the program address $100 and is also the reset vector. The global label 'blink' is a function which blinks a light (whose implementation is left as an exercise for the reader). The local label ':loop' may be used again in other sections of the code, allowing for convenient nicknames for loops and other locations private to the implementation of a function.

Labels for program locations refer to the entire 12-bit address of the labeled instruction. Since the CALL and JMP instructions can only use 8 and 9 bits of the address, the assembler will silently truncate the target address to fit in the instruction. If possible, the assembler will generate a warning if the target address is not in the same page as set by the most recent PAGE instruction. To avoid PAGE mismatches automatically, a label may often be used in conjuction with an '@' symbol, which will cause the required PAGE instruction to be inserted. For example,

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
call @label
```

is assembled identically to

```
page      label
call      label
```

The same capability is available for any instruction which takes an 8-bit or 9-bit target address.

2.2.2 Mnemonic

The mnemonic field begins after the first white space in the source line and is terminated by the next white space. The field may contain an instruction mnemonic, assembler directive or macro.

2.2.3 Operand

The operand field begins immediately after the first white space following the mnemonic field and ends at the next white space. The field may contain one or more constants or expressions separated by commas.

2.2.4 Comment

The comment field begins immediately after the first white space following the operand field, or the mnemonic field for those mnemonics that do not require any operands. This is an optional field containing printable characters. Anything to the right of a semicolon (;) is treated as a comment and will be ignored by the assembler.

2.2.5 Constants

Constants are strings or numbers that SASM interprets as a fixed numeric value. SASM supports radix form character, hexadecimal, decimal, octal and binary. SASM uses decimal as the default radix which helps determine what value will be assigned to constants in the object file when they are not explicitly specified by a base descriptor.

2.2.6 Characters or String Constants

String constants always begin with a single or double quote, and end with a matching single or double quote. SASM converts the characters between the quotes to ASCII values. For example:

```
MOV  W,#'A'
RETW #'A'
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

2.2.7 Numeric Constants

A numeric constant in SASM consists of an arbitrary number of alphanumeric characters. The actual value of the constant depends on the radix you select to interpret it. Radices available in SASM are binary, octal, decimal, and hexadecimal, as shown below. If no radix is given, SASM uses the default radix as specified by the /R command-line option, or decimal if no /R option is present.

Hexadecimal numbers must always start with a decimal digit (0-9) if the trailing "H" notation is used. If necessary, put a leading 0 at the left of the number to distinguish it between hexadecimal numbers that start with a letter (A- F). The hexadecimal digits A through F can be either upper-case or lower-case. Constants can be optionally preceded by a plus or minus sign.

The formats for declaring a constant are shown in Tables2-1. The Radix descriptor is case insensitive. Also, either single-quote and double-quote charaters may be used where single-quotes are shown in Table 2-1.

Table2-1 Constants Declaration

TYPE	SYNTAX	EXAMPLE
Binary	<binary digits>B B'<binary digits> B"<binary digits> %<binary digits>	01000001B B'01000001 B"01000001 %11111011
Octal	<octal digits>O O'<octal digits> Q'<octal digits>	101O O'101 Q'101
Decimal	<digits>D D'<digits> D"<digits>	65D D'65 D"65
Hexadecimal	<digit><hex digits>H <digit><hex digits>X H'<hex digits> X'<hex digits> 0x<hex digits> $<hex digits>	41H 41X H' 41 X' 41 0x41 $41
Character	'<character>'	'A'

© 2000 Scenix Semiconductor, Inc. All rights reserved.

2.3 Symbols

A symbol represents a value, which can be a variable, an address label or an operand to an assembly instruction or directive.

2.3.1 Symbol Names

Symbol names are user-defined or predefined combination of letters (both uppercase and lowercase), digits and special characters. They are represented by a string of 1-32 alphanumeric characters with the first character being 'A' to 'Z', 'a' to 'z', '_', '@' or ':'. Valid characters for SASM are as follows:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 @ !_
```

NOTE: SASM accepts upper and lower case characters, and is case sensitive.

2.3.2 Symbol Types

Each symbol has a type that describes the characteristics and information associated with it. The way you define a symbol determines its type. SASM supports four symbol types:

DATA: A user-defined symbol that represents a data variable defined by EQU directive

VAR: A user-defined symbol that represents a data variable defined by SET directive

ADDR: A user-defined symbol that represents a code address or program counter location

RESV: A predefined symbol used internally by SASM

2.3.3 User-Defined Symbols

Symbols are used in both label and operand fields in the source statement. Symbols are defined in the label field as either the current program address or as the resulting value of an EQU or SET expression. These values can then be used symbolically in operand fields. All symbols must be defined at some point in the source code by appearing in the label field. A symbol may begin with a colon character, in which case it is appended to the most recently defined symbol not beginning with a colon to form the name which appears in the symbol table. This can be used to define locally-scoped lables within a larger region of code.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

2.3.4 Reserved Symbols

The assembler has internally defined the following reserved symbols

```
=          DS         RES        DW         FREG
EQU        ORG        END        SET        WATCH
ENDM       EXITM      IF         IFDEF      BREAK
IFNDEF     ELSE       ENDIF      EXPAND     CASE
NOEXPAND   LIST       DEVICE     ID         NOCASE
RESET      SPAC       ZERO       LOCAL      ERROR
LPAGE      MACRO      RADIX      TITLE
STITLE     INCLUDE    SUBTITL    LIST
PROCESSOR

W          M          OR         PC
RA         RB         RC         RD
RE         RL         RR         SB
SC         SZ         ADD        AND
CLC        CLR        CLZ        DEC
INC        JMP        MOV        NOP
NOT        RET        SNB        SNC
SNZ        SUB        WDT        XOR
BANK       CALL       CLRB       DATA
MODE       PAGE       RETI       RETP
RETW       SETB       SKIP       SWAP
TEST       DECSZ      INCSZ      IREAD
MOVSZ      RETIW      SLEEP      OPTION
```

2.4 Expressions

Expressions are used in the operand field of the source statement and may contain constants, symbols or any combination of constants and symbols separated by operators.Expressions are calculated with 32-bit arithmetic.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

2.4.1 Arithmetic Operators

The arithmetic operators available in expressions are listed in the following table. Operators are grouped by precedence, with earlier groups in the table at a higher precedence than later. Withing each group, precedence is strictly left to right.

Parenthesis may be used to modify the precedence arbitrarily, and may be nested to any required depth.

Note that some operators are not useful without parenthesis due to their actual precedence. For example, the bit position operator (".") is at a relatively high precedence which allows expressions such as 3.2+5 to evaluate to 7.2 which makes sense. If the intended result is 3.7, then the expression should be written as 3.(2+5).

All arithmetic is performed in 32-bit two's-complement integers, and intermediate results may be saved in the symbol table with 32 significant bits along with a 3-bit bit position. For consistency with the processor datasheets, an unspecified bit position is equivalent to bit zero. Of course, operand values are truncated as appropriate to fit the instructions with which they are used.

OPERATOR	DESCRIPTION	EXAMPLE	[VALUE]
Magic Values			
$	Current Program Counter		
%	Current repition counter		
Parenthesis and Not			
( )	Grouping	(10+5)/5	[3]
!	Logical Not	!(3= =5)	[TRUE]
Bit Number			
..	FR.BIT	3.2	
Multiplication and Division			
*	Multiplication	3*4	[12]
/	Division	3/4	[0]
//	Modulus	10//8	[2]
&	Bitwise And	10&3	[2]
^	Bitwise Exclusive Or	10^3	[9]
<<	Left Shift	10<<3	[80]
>>	Right Shift	10>>3	[1]

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Unary sign			
+	Unary Plus	+14	[14]
-	Unary Minus	-14	[-14]
~	Unary One's Complement	~1	[0xfffffffe]
Addition and Subtraction			
+	Addition	6+8	[14]
-	Subtraction	6-8	[-2]
|	Bitwise Inclusive Or	10|3	[11]
Logical Relations			
= =	Logical Equal	3= =5	[FALSE]
!=	Logical Not Equal	3!=5	[TRUE]
<	Less than	3<5	[TRUE]
>	Greater than	3>5	[FALSE]
<=	Less than or equal	3<=5	[TRUE]
>=	Greater than or equal	3>=5	[FALSE]
||	Logical Or	(x= =y)||(x!=y)	[TRUE]
&&	Logical And	(x= =y)&&(x!=y)	[FALSE]

2.4.2 Well-Defined Expressions

Some of the directives require well-defined expressions. These are expressions that can be evaluated on the first pass. This means any symbols used in the expression must be defined on an earlier line in the source file.

For example:

```
len     equ     4       ;length of an array
        org     $30
vara    ds      1       ;byte variable
varb    ds      len     ;array of len bytes
```

Each of the expressions above must be well-defined during pass 1. The value of 'len' given to the EQU directive must be known so that the symbol can be defined. The argument to the ORG directive must be known so that subsequent symbols can be defined. The argument to the DS directive must be known so that the actual value of varb and any subsequent symbols will be known.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Chapter 3

SASM Assembler Directive

3.1 Introduction

Directives are assembler commands that appear in the source code but are not translated directly into opcodes. They are used to control the program counter, allocation, and format listing outputs. Table3-1shows a summary of directives..

Table 3-1 Assembler Directives

Directive	Description	Syntax
BREAK	No effect (SXKey compatibility)	BREAK
CASE	No effect (SXKey compatibility)	CASE
DEVICE	Define device type and fuse options	DEVICE setting { setting, ... }
DS	Define memory space by incrementing the program memory address	Symbol ds 1 Symbols ds 3
DW	Define 16-bit data in program memory	DW data, { data... }
END	Mark the End of source code	END
EQU	Equate a symbol to an expression. The symbol cannot be reassigned	Symbol EQU expression
EXPAND or NOEXPAND	Specifies whether to expand the macro instructions in the list file	EXPAND or NOEXPAND
FREQ	No effect (SXKey compatibility)	FREQ expression
__FUSE __FUSEX	Define FUSE and FUSEX WORDs as explicit expression values. Not recommended for use.	__FUSE expression __FUSEX expression
FUSES	Synonym for DEVICE	FUSES setting ...
GLOBAL	Synonym for EQU	label GLOBAL expression
ID	Define an ID string up to 8 characters	ID ‘string’
IF {ELSE} ENDIF	Conditional assembly	IF expression { ELSE } ENDIF

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Table 3-1 Assembler Directives

Directive	Description	Syntax
IFDEF {ELSE} ENDIF	Conditional assembly	IFDEF symbol { ELSE } ENDIF
IFNDEF { ELSE } ENDIF	Conditional assembly	IFNDEF symbol { ELSE } ENDIF
INCLUDE	Insert external source file	INCLUDE ‘file’
LIST	Control the list format, set certain command-line options	LIST {P=processor} {R=radix} {F=format} {L=NONE|PAGE|NOPAGE} {X=ON|OFF} {C=cols} {N=lines}
LPAGE	Insert page eject in listing file	LPAGE
MACRO { EXITM } ENDM	Defines a macro	Label MACRO {argument, ...} { EXITM } ENDM
NOCASE	No effect (SXKey compatibility)	NOCASE
ORG	Set program origin	ORG expression
PROCESSOR	Synonym for DEVICE	PROCESSOR setting ...
RADIX	Set default radix	RADIX=[BIN|OCT|DEC|HEX] RADIX=[B|O|D|H]
REPT ENDR	Repeat block of program code a specified number of times	REPT count ENDR
RESET	Define reset vector (starting location) of program	RESET label
RES	Reserve storage in memory	RES expression
SET or =	Set a symbol equal to an expression. The symbol can be reassigned to new value	Symbol SET expression Symbol = expression
SPAC	Insert lines in listing	SPAC expression
STITLE	Synonym for TITLE	STITLE 'Title Text'
SUBTITLE'	Set a listing subtitle	SUBTITLE 'Subtitle Text
TITLE	Define program heading	TITLE ‘file’

© 2000 Scenix Semiconductor, Inc. All rights reserved.

Table 3-1 Assembler Directives

Directive	Description	Syntax
WATCH	No effect (SXKey compatibility)	WATCH {arguments}
ZERO	Synonym for RES	ZERO expression

3.1.1 FREQ, BREAK, WATCH, CASE, NOCASE (SXKey Compatibility)

Syntax: [<label>] FREQ <expression> [<comment>]

[<label>] BREAK [<comment>]

[<label>] WATCH <operands> [<comment>]

[<label>] CASE [<comment>]

[<label>] NOCASE [<comment>]

Description: Ignore the directive and any operands. These directives have a particular meaning to the Parallex SXKey assembler, but is unsupported by SASM. For better portability of code originally written with SXKey, SASM will ignore this directive. Note that SASM does not make any attempt to validate the operands expected by SXKey.

Example:
```
        FREQ 20000000
loop    inc     fr
BREAK
        jmp     loop
```

3.1.2 DEVICE or FUSES or PROCESSOR- Define Device Type and Fuse Bits

Syntax: [<lable>] DEVICE <settings> [, settings...] [<comment>]
FUSE <settings>
[<lable>] PROCESSOR <settings> [, settings...] [<comment>]

Description: Specifies the device type and fuse bits of both FUSE and FUSEX words to SASM assembler.

Example:
```
DEVICE      PINS28, BANKS8, OSCHS
DEVICE      TURBO, STACHKX, OPTIONX, CARRYX, PROTECT
```

There are different fuse settings for different device types.

NOTE: When using the SX18/20/28AC devices with the SX-ISD Debugger, the fuse bits that select the program memory size must be set to BANKS8 (2K program memory).

Table3-2, and Table3-3 show the FUSE/FUSEX bit settings for the SX18/20/28AC and SX48/52BD devices:

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Table3-2 FUSE/FUSEX Bit Settings for SX18/20/28AC

Option Bits	Description	Function	Default
PINS18/SX18AC PINS20/SX20AC PINS28/SX28AC PINS48/SX48BD PINS52/SX52BD	SX18AC SX20AC SX28AC SX48BD SX52BD	Specifies device type	PINS18
BANKS1 BANKS2 BANKS4 BANKS8	1 page, 1 bank 2 page, 1 bank 4 pages, 4 banks 4 pages, 8 banks	Configure memory size (should not be changed unless to reduce the amount of program memory)	BANKS8
OSCLP1 OSCLP2 OSCXT1 OSCXT2 OSCHS1 OSCHS2 OSCHS3 OSCRC	Ext Osc - LP1 Ext Osc - LP2 Ext Osc - XT1 Ext Osc - XT2 Ext Osc - HS1 Ext Osc - HS2 Ext Osc - HS3 Ext Osc - RC	Specifies external crystal / resonator or external RC oscillator	OSCRC
OSC4MHZ OSC1MHZ OSC128KHZ OSC32KHZ	Int RC Osc - 4MHz Int RC Osc - 1MHz Int RC Osc - 128kHz Int RC Osc - 32kHz	Specifies internal oscillator speed	4MHz
IFBD	0 = an ext feedback resistor is required between OSC1 and OSC2 pins. 1 = crystal/resonator OSC can rely on into feedback resistor between OSC1 and OSC2 pins	Internal Feedback Disable	Enable internal feedback resister
BOR42 BOR26 BOR22 BOROFF	Brown-out reset at 4.2V Brown-out reset at 2.6V Brown-out reset at 2.2V Disable Brown-out reset	Specifies brown-out reset function and threshold voltage	Disable brownout
TURBO	0 = Turbo mode (1:1) 1 = compatible mode (1:4)	Specifies turbo mode	Compatible mode
OPTIONX	0 = 8-bit option register and 8-level stack 1 = 6-bit option register and 2-level stack	Specifies Option register and stack extension	6 bits and 2-level

© 2000 Scenix Semiconductor, Inc. All rights reserved.

Table3-2 FUSE/FUSEX Bit Settings for SX18/20/28AC

Option Bits	Description	Function	Default
CARRYX	1 = ignore carry flag as input to ADD and SUB instruction	ADD and SUB instructions use Carry flag as input	Carry flag ignored
SYNC	0 = Enable synchronous inputs 1 = Disable synchronous inputs	Enable or disable isochronous input mode (for turbo mode operation)	Disabled
WATCHDOG	0 = Disable watchdog timer 1 = Enable watchdog timer	Enable or Disable Watchdog Timer	Disabled
PROTECT	0 = Code protect enabled 1 = Code protect disabled	Specified code protection	Disabled

.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Table3-3 FUSE/FUSEX Bit Settings for SX48/52BD

Option Bits	Descriptions	Function	Default
PINS18/SX18AC PINS20/SX20AC PINS28/SX28AC PINS48/SX48BD PINS52/SX52BD	SX18AC SX20AC SX28AC SX48BD SX52BD	Specifies device type	PINS18
OSCLP1 OSCLP2 OSCXT1 OSCXT2 OSCHS1 OSCHS2 OSCHS3 OSCRC	Ext Osc – LP1 Ext Osc – LP2 Ext Osc – XT1 Ext Osc – XT2 Ext Osc – HS1 Ext Osc – HS2 Ext Osc – HS3 Ext OSC – RC	Specifies external crystal / resonator Or external RC circuit	OSCRC
OSC4MHZ OSC1MHZ OSC128KHZ OSC32KHZ	Int Osc – 4MHz Int Osc – 1MHz Int Osc – 128kHz Int Osc – 32kHz	Specifies internal oscillator divider	4MHz
BOR42 BOR26 BOR22 BOROFF	Brown-out reset at 4.2V Brown-out reset at 2.6V Brown-out reset at 2.2V Disable Brown-out reset	Specifies brown-out reset	Disable brownout
CARRYX	1 = ignore carry flag as input to ADD and SUB instruction	ADD and SUB instructions use Carry flag as input	Carry flag ignored
SYNC	0 = Enable synchronous inputs 1 = Disable synchronous inputs	Enable or Disable synchronous input mode (for turbo mode operation)	Disabled
WATCHDOG	0 = Disable watchdog timer 1 = Enable watchdog timer	Enable or Disable Watchdog Timer	Disabled
PROTECT	0 = Code protect enabled 1 = Code protect disabled	Specified code protection.	Disabled

© 2000 Scenix Semiconductor, Inc. All rights reserved.

Table3-3 FUSE/FUSEX Bit Settings for SX48/52BD

Option Bits	Descriptions	Function	Default
SLEEPCLK	0 = Enable clock operation during sleep mode 1 = Disable clock operation during sleep mode	Sleep Clock Disable	Disable sleep clock
WDRT60 WDRT960 WDRT006 WDRT184	60 msec 1 sec 0.25 msec 18.0 msec (default)	Delay Reset Timer time-out period	18.0 msec

3.1.3 DS - Define Memory Space

Syntax: [<label>] DS <operand>

Description: Define memory space by incrementing the program memory address during assembly.

Example:
```
ORG         $10
Timers      =       $
timers_low  ds      1       ;   $10
timers_high ds      1       ;   $11
timers_accl ds      1       ;   $12
timers_array ds     3       ;   $13, $14, $15
```

3.1.4 DW - Define Data in Memory

Syntax: [<label>] DW <operand>

Description: Initialize one or more words of program memory with data. The data may be in the form of constants or ASCII character strings.

Example:
```
DW      10h, 20h, 30h
or
DW      'This is a test'
```

3.1.5 END - End of Source Program

Syntax: [<label>] END [<comment>]

Description: Mark the end of program.

Example:
```
END         ; terminate the program
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

3.1.6 EQU or GLOBAL- Equate a Symbol to an Expression

Syntax: <label> EQU <expression> [<comment>]
<lable> GLOBAL <expression>[<comment>

Description: A constant value or the value of a well-defined expression is assigned to the given label. Note that any value defined with an EQU directive is fixed and may not be redefined.

Example:
```
COUNT   EQU   19h
```

To support semi-direct addressing mode for SX48/52BD devices and differentiate between global registers and bank 0 registers. The bank 0 registers must be defined as the 9-bit values $100 through $10F. In effect, SASM treats the imaginary BANK 16 as identical to BANK 0. In addition, banks 1 through 15 may also be reffered to by addresses $110 through $1FF.

3.1.7 ID - Set an ID String in Program Memory

Syntax: ID “Text”

Decription: Assigns an ID text string at the end of program memory. The string may be up to 8 characters and should be in quotes

Example: ID ‘Demo28’

3.1.8 IF.ELSE.ENDIF - Conditional Assembly

Syntax:
```
IF <expression>
        <source lines>
ELSE
        <source lines>
ENDIF
```

Description: ELSE is used in conjunction with IF directive to provide an alternative path. If IF tests false, the alternative path noted by the ELSE directive is taken, providing conditional assembly. The IF statement requires a matching ENDIF statement.

Example:
```
count   equ     12h
IF      (count >    10h)
        INC     4
ELSE
        DEC     4
ENDIF
```

3.1.9 IFDEF.ELSE.ENDIF - Conditional Assembly

Syntax:
```
IFDEF <symbol>
        <source lines>
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
ELSE
        <source lines>
ENDIF
```

Description: ELSE is used in conjunction with IFDEF directive to provide an alternative path. If symbol is not defined, the alternative path noted by the ELSE directive is taken, providing conditional assembly. The IFDEF statement requires a matching ENDIF statement.

Example:
```
varl    equ     10h
.
.
.
IFDEF   varl
        INC     4
ELSE
        DEC     4
ENDIF
```

3.1.10 IFNDEF.ELSE.ENDIF - Conditional Assembly

Syntax:
```
IFNDEF  <symbol>
        <source lines>
ELSE
        <source lines>
ENDIF
```

Description: ELSE is used in conjunction with IFNDEF directive to provide an alternative path. If symbol is defined, the alternative path noted by the ELSE directive is taken, providing conditional assembly. The IFNDEF statement requires a matching ENDIF statement.

Example:
```
IFNDEF      varl
        INC     4
ELSE
        DEC     4
ENDIF
```

3.1.11 INCLUDE - Insert External Source File

Syntax: [<label>] INCLUDE “<filename>” [<comment>]

Description: To read in the specified file as source code. A path name can be provided if the file resides in another directory.

Example:
```
INCLUDE         “SXREG. INC”
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

3.1.12 LIST -- Control the list file format

Syntax: [<label>] LIST [P=<processor>]
[<label>] LIST [R=<radix>]
[<label>] LIST [F=<format>]
[<label>] LIST [L=<list>]
[<label>] LIST [X=<on/off>]
[<label>] LIST [C=<cols>]
[<label>] LIST [N=<lines>]

Description: The LIST directive sets certain command-line options within the source file, and allows additional control of the list file format.
The first four options mirror the command-line options /P, /R, /F, and /L, respectively.
Use any of SX18, SX18AC, SX20, SX20AC, SX28, SX28AC, SX48, SX48BD, SX52, SX52BD, or OLDREV for <processor>.
Use any of BIN, B, OCT, O, DEC, D, HEX, or H for <radix>.
Use any of NONE, PAGE, or NOPAGE for <list>.
LIST X=ON is a synonym for EXPAND, and LIST X=OFF is a synonym for NOEXPAND.
LIST C=<cols> and LIST N=<lines> sets the number of columns and lines on a listing page, respectively.
Note that SASM does not make any attempt to validate the operands expected by SXKey.

3.1.13 LPAGE - Insert Page Eject in Listing File

Syntax: [<label>] LPAGE [<comment>]

Description: Insert a form feed at this point in the listing file.

Example: LPAGE

3.1.14 ORG - Set Program Origin

Syntax: [<label>] ORG <expression> [<comment>]

Description: Set program origin for subsequent code at the expression value. The expression must be well-defined.

Example:
```
ORG     0
or
ORG     $100
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

3.1.15 RADIX -- Set default radix

Syntax: [<label>] RADIX=<radix> [<comment>]

Description: Set the default radix for constants to one of binary, octal, decimal, or hexidecimal. The default default radix is decimal, unless modified by the RADIX directive, the LIST R=<radix> directive, or the /R command-line option.

Example:
```
RADIX=HEX
org     100
```

3.1.16 REPT-ENDR – Repeat Code Block

Syntax:
```
REPT    count
Codeblock
ENDR
```

Description: Used to indicate a block of code to be repeated a specified number of times during assembly.

Example:
```
REPT    3
add     $12,#1
ENDR
```

will be expanded to the following sequence during program assembly:
```
add     $12,# 1
add     $12,# 1
add     $12,# 1
```

Within the block, the % sign may be used to refer to the current iteration(1-n), i.e. % equal to 1 the first time through the repeat block, % equal to 2 the second time through the loop etc. For example:
```
REPT    3
Add     $12,# %
ENDR
```
will be expaned tothe following sequence during assembly:
```
Add     $12,# 1
Add     $12,# 2
```

```
Add     $12,# 3
```

3.1.17 RESET - Set Reset Vector Address

Syntax: RESET <expression> [<comment>]

Description: Put the instruction opcode [JMP <expression>] at the reset vector memory location. The reset vector location depends on the chip's configured memory size, and defaults to $7FF.

Example: Define PAGESx in FUSES Reset Vector

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
FUSES PAGES1        0x1FF
FUSES PAGES2        0x3FF
FUSES PAGES4        0x7FF
FUSES PAGES8        0x7FF

DEVICE              PINS18
RESET               Start
This is equivalent to:
ORG                 1FFh
JMP                 Start
```

NOTE: The expression must evaluate to a destination address in Page 0.

3.1.18 RES or ZERO- Reserve Storage in Memory

Syntax: [<label>] RES <expression> [<comment>]
[<label>] ZERO <expression> [<comment>]

Description: The program counter will be advanced by the amount of the expression.

Example: RES 10

© 2000 Scenix Semiconductor, Inc. All rights reserved.

3.1.19 SET or = - Set a Symbol Equal to an Expression

Syntax: [<label>] SET <expression> [<comment>]

Description: To assign the value of a well-defined expression to a label. Unlike the EQU directive, SET can be used more than once on the same symbol; with the most recent SET statement determining the value of the label.

Example:
```
FIVE    SET     5
or
FIVE    =       5
```

3.1.20 SPAC - Insert Lines in Listing File

Syntax: [<label>] SPAC <expression> [<comment>]

Description: Insert the number of blank lines given by the expression into the listing file.

Example:
```
SPAC    5
```

3.1.21 TITLE or STITLE- Define Program Heading

Syntax:
```
[<label>]  TITLE "<string>"  [<comment>]
[<label>]  STITLE "<string>"  [<comment>]
```

Description: Set up the text to be used in top line of listing file.

Example:
```
TITLE   "SAMPLE.ASM"
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Chapter 4

Macros

4.1 Introduction

Macros consist of sequences of assembler instructions and directives that can be inserted in the assembly source code by using a macro call. The macro must first be defined then it can be call upon later at any point within the source program.

4.2 Macro Definition

A macro definition is divided into three areas:

- Macro Heading,
- Macro Body
- Macro Terminator

4.3 Macro Heading

The format of the macro heading is as follows

<label> MACRO [<parameter> ... <parameter>][<comment>]:

Where <label> is the name of the macro, and <parameter> is an input argument passed into the macro call by value. Parameter can only be operand values, not instructions.

OR

```
<lable>     MACRO       {Argcount}
            codeblock
            {EXITTM}
            ENDDM
```

Where {Argcount} specifies the exact number of arguments required by the macro and must range from 1 to 64.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

The comment field is permitted in the heading whether or not there are parameters. The name of the macro must comply with SASM label rules. If a macro name is identical to a mnemonic or an assembler directive, the assembler will generate an error.

4.4 Macro Body

The macro body begins immediately after the macro definition and continues until the macro terminator. The macro body consists of a sequence of source lines that may contain a formal parameter in any field. When the macro is instantiated, all parameters will be replaced by the corresponding arguments provided by the macro call.

4.5 Macro Terminator

The ENDM directive terminates the macro definition. ENDM must exist before another MACRO statement is found. The format of the macro terminator is as follows:

ENDM [<comment>]

4.6 Macro Call

Once the macro has been defined, it can be instantiated at any point within the source module by using a macro call as described below

<label>] <name> [<arg> [,<arg>] ...][<comment>]

<label> is assigned the current value of the location counter
<name> is the name of the macro to be instantiated<arg> is any symbol or constant passed as a parameter to the macro

The macro call itself will not occupy any locations in memory. However, the macro instantiation will begin at the current memory location. Commas may be used to reserve an argument position. In this case the argument will be null. The argument list is terminated by white space or a semi-colon.

4.7 Parameters

All arguments are passed into the macro instantiation by value. Currently SASM supports symbols, constants and fr.bit type as macro parameters. However, it does not allow string operands or reserved symbols as macro arguments.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

4.8 Local Symbols

Local symbols are labels declared within macros only. These symbols are local to the particular macro and are differentiated from regular labels which are global to the entire program. Each time the macro is called, SASM will assign each local symbol a system generated symbol of the form ??0001, ??0002, ??0003. etc. All Local definitions must occur immediately after the MACRO heading and before the first line of the macro body with a syntax as followed:

These local macro labels do not have to start at column 1, as the global labels.

```
LOCAL  <label>  [,<label>] ...
```

4.9 Macro Examples

Example 1:

Definition

```
;   Define macro
CLRREG          MACRO   reg
                LOCAL   again
again           clr     reg
                jmp     again
                ENDM
```

Macro Call

```
CLRREG          20h
```

Macro expansion

```
0046                                    CLRREG          08h
0046    0006    0068    m   ??0000      clr             08h
0046    0007    0A06    m               jmp             ??0000
0047
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Example 2:

Definition

```
;  Define macro
ANDREG      MACRO       3
            And         W,# 1
            And         W,# 2
            And         W,# 3
            ENDM
```

Macro Call

```
ANDREG      5, 6, 7,
```

Macro expansion

```
0046                                    ANDREG      5, 6, 7
0046    0006    0E05    m               and         W,# 5
0046    0007    0E06    m               and         W,# 6
0047    0008    0E07    m               and         W,# 7
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Chapter 5

Assembler Output Files

5.1 Introduction

When SASM is activated, you will see the following:

SASM Cross-Assembler for Scenix SX-based Microcontrollors Version xxx
Copyright (c) Advanced Transdata Corporation 1999

xxx lines compiled in xxx seconds
xxx symbols
< error status >

For each source file submitted, the SASM will produce the following files:
HEX: object file
LST: listing file, unless the /L switch is given to suppress its output
SYM: symbol file
MAP: map file
ERR: error message file

5.2 Object File (HEX or OBJ)

The object file can be in different formats and contains data that can be loaded and executed. SASM outputs INHX8M (Intel 8-bit Hex file) format as the default. This file will be used by the device programmer and the debug tool for programming/debugging purposes.

The other formats: BIN16, INHX16, INHX8S, and IEEE695 are provided to support other programmers. See Appendix A for more information on the individual object file formats.

5.3 Listing File (LST)

The listing file contains the source code along with some useful information about the output addresses and corresponding object code. Each line from the source code will be reproduced in the listing file and accompanied by the listing file line number, program counter and the object code (OPCODE).

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

Example

```
LINE    PC      OPCODE
0011    0000    0C02    mov     W,#00000010b
0012    0001    01A6    xor     rb,W            ; toggle    rb.1
0013    0002    0CEC    mov     w,# - 20
0014    0003    000F    retiw
```

The first field is a 4-digit decimal number that represents the line number at which the source line appears in the source code. The second field is a 4-digit hex number that represents the current program counter. The third field is a 4-digit hex number that represents the opcode generated from the source line. This is the actual value that will appear in the object code.

5.4 Cross Reference Listing

A cross-reference table is generated at the end of the listing file. This table contains a list of every symbol used in the source file along with its symbol type, value and the source line number.

For example:

```
SYMBOL      TYPE    VALUE   LINE
W           RESV    0000    0006
LOOPB       ADDR    0109    0044
XCNT        DATA    0010    0011
YCNT        VAR     0011    0045
```

Where

DATA: A user-defined symbol that represents a data variable defined by EQU directive

VAR: A user-defined symbol that represents a data variable defined by SET directive

ADDR: A user-defined symbol that represents a code address or program counter location

RESV: A predefined symbol used internally by SASM

5.5 Symbol File (SYM)

The symbol file is identical to the cross reference portion of the listing file. It lists all symbols found in the source file, provides information on their type, value and the specific line numbers where they are found. The symbol is required to define watch variables and to specify breakpoint at address label for the debug tool.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

5.6 Map File (MAP)

The map file contains line correspondence between source file, program counter and file number. This file is necessary to enable source level debugging with the emulator. The contents of the map file vary, depending on which switch is used during compilation.

SASM generates correspondence between source file (.ASM) and program counter. It enables Emulators to load the source file to the Source Window during debugging.

5.7 Error File (ERR)

The error file contains all error messages generated during program compilation. If there is no error, the file will have zero byte.

5.8 Error Messages

Error messages are displayed at the terminal and in the listing file. They all have the following format:

<List Line#> <File (Source Line#) > <Error/Warning Count> : <Pass#> : <message>

A list of error/warning messages is given in Appendix C.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Appendix A

Summary of SX Instruction Set

Mnemonics, Operands	Flags	Description

A.1 Logical Operations

AND	fr,W	Z	AND W into fr
AND	W,fr	Z	AND fr into W
AND	W,#lit	Z	AND literal into W
NOT	fr	Z	One's complement of fr into fr
NOT	W	W, Z	One's complement of W into W
OR	fr,W	Z	OR W into fr
OR	W,fr	Z	OR fr into W
OR	W,#lit	Z	OR literal into W
XOR	fr,W	Z	XOR W into fr
XOR	W,fr	Z	XOR fr into W
XOR	W,#lit	Z	XOR literal into W

A.2 Arithmetic and Shift Operations

ADD	fr,W	C,DC,Z	Add W to fr into fr
ADD	W,fr	C,DC,Z	Add fr to W into W
CLR	fr	Z	Clear fr to 0
CLR	W	Z	Clear W to 0
CLR	!WDT	TO,PD	Clear WDT and prescaler
DEC	fr	Z	Decrement fr
DECSZ	fr	-	Decrement fr, skip if zero
INC	fr	Z	Increment fr
INCSZ	fr	-	Increment fr, skip if zero
NOP		-	No operation
RL	fr	C	Rotate left fr into fr
RR	fr	C	Rotate right fr into fr
SUB	fr,W	C,DC,Z	Subtract W from fr
SWAP	fr	-	Swap nibbles in fr into fr

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

A.3 Bitwise Operations

CLRB	fr.bit	-	Clear bit to 0
CLC		C	Clear carry
CLZ		Z	Clear zero
SB	bit	-	Skip if bit = 1
SETB	fr.bit	-	Set bit to 1
SNB	bit	-	Skip if bit = 0

A.4 Data Movement Operations

MOV	fr,W	-	Move W into fr
MOV	W,fr	Z	Move fr into W
MOV	W,fr-W	C,DC,Z	Move fr-W into W
MOV	W,#lit	-	Move literal into W
MOV	W,/fr	Z	Move 1's complement of fr to W
MOV	W,--fr	Z	Move fr-1 into W
MOV	W,++fr	Z	Move fr+1 into W
MOV	W,<<fr	C	Move left-rotated fr into W
MOV	W,>>fr	C	Move right-rotated fr into W
MOV	W,<>fr	-	Move nibble-swapped fr into W
MOV	W,M	-	Move MODE into W
MOV	M,W	-	Move W into MODE
MOV	M,#lit	-	Move literal into MODE
MOV	!rx,W	-	Move W into Port Rx control register
MOV	!OPTION,W	-	Move W into OPTION
MOVSZ	W,--fr	-	Move fr-1 into W, skip if zero
MOVSZ	W,++fr	-	Move fr+1 into W, skip if zero
SC		C	Skip if carry bit is set
TEST	fr	Z	Test if fr equal to 0

A.5 Control Transfer Operations

CALL	addr8	-	Call to address
JMP	addr9	-	Jump to address
JMP	W	-	Move W into PC(L)
JMP	PC+W	C,DC,Z	Add W into PC(L)
RET		-	Return from call without affecting W
RETP		-	Return from call, write to PA2:PA0
RETI		-	Return from interrupt
RETIW		-	Return from interrupt, subtract W from RTCC
RETW	#lit	-	Return from call, move literal in W
SKIP		-	Skip the following instruction

© 2000 Scenix Semiconductor, Inc. All rights reserved.

A.6 System Control Operations

BANK	n	-	Transfer n to FSR7:FSR5
IREAD		-	Read instruction at MODE:W into MODE:W
MODE	n	-	Transfer n into MODE
M	n	-	Transfer n into MODE
PAGE	n	-	Transfer n to PA2:PA0
SLEEP		TO,PD	Clear WDT and enter sleep mode

A.7 Multi-Byte Instructions

Mnemonics, Operands		Affects	Description
add	fr,#lit	fr,W,C,DC,Z	ADD lit into fr
add	fr,fr2	fr,W,C,DC,Z	ADD fr2 into fr
addb	fr,frbit	fr,Z	ADD frbit into fr
addb	fr,/frbit	fr,Z	ADD ~frbit into fr
and	fr,#lit	fr,W,Z	AND lit into fr
and	fr,fr2	fr,W,Z	AND fr2 into fr
cja	fr,#lit,addr9	W,C,DC,Z	JUMP if fr > lit
cja	fr,fr2,addr9	W,C,DC,Z	JUMP if fr > fr2
cjae	fr,#lit,addr9	W,C,DC,Z	JUMP if fr >= lit
cjae	fr,fr2,addr9	W,C,DC,Z	JUMP if fr >= fr2
cjb	fr,#lit,addr9	W,C,DC,Z	JUMP if fr < lit
cjb	fr,fr2,addr9	W,C,DC,Z	JUMP if fr < fr2
cjbe	fr,#lit,addr9	W,C,DC,Z	JUMP if fr <= lit
cjbe	fr,fr2,addr9	W,C,DC,Z	JUMP if fr <= fr2
cje	fr,#lit,addr9	W,C,DC,Z	JUMP if fr == lit
cje	fr,fr2,addr9	W,C,DC,Z	JUMP if fr == fr2
cjne	fr,#lit,addr9	W,C,DC,Z	JUMP if fr != lit
cjne	fr,fr2,addr9	W,C,DC,Z	JUMP if fr != fr2
csa	fr,#lit	W,C,DC,Z	SKIP if fr > lit
csa	fr,fr2	W,C,DC,Z	SKIP if fr > fr2
csae	fr,#lit	W,C,DC,Z	SKIP if fr >= lit
csae	fr,fr2	W,C,DC,Z	SKIP if fr >= fr2
csb	fr,#lit	W,C,DC,Z	SKIP if fr < lit
csb	fr,fr2	W,C,DC,Z	SKIP if fr < fr2
csbe	fr,#lit	W,C,DC,Z	SKIP if fr <= lit
csbe	fr,fr2	W,C,DC,Z	SKIP if fr <= fr2
cse	fr,#lit	W,C,DC,Z	SKIP if fr == lit
cse	fr,fr2	W,C,DC,Z	SKIP if fr == fr2
csne	fr,#lit	W,C,DC,Z	SKIP if fr != lit
csne	fr,fr2	W,C,DC,Z	SKIP if fr != fr2
djnz	fr,addr9	fr	Decrement fr, JUMP if not zero

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

ijnz	fr,addr9	fr	Increment fr, JUMP if not zero
jb	frbit,addr9	-	JUMP if bit set
jc	addr9	-	JUMP if carry
jnb	frbit,addr9	-	JUMP if bit clear
jnc	addr9	-	JUMP if no carry
jnz	addr9	-	JUMP if not zero
jz	addr9	-	JUMP if zero
mov	fr,#lit	fr,W	MOVE lit into fr
mov	fr,fr2	fr,W,Z	MOVE fr2 into fr
mov	fr,m	fr,W	MOVE M into fr
mov	m,fr	W,M,Z	MOVE fr into M
mov	!option,fr	W,Z,OPT	MOVE fr into OPTION
mov	!option,#lit	W,OPT	MOVE lit into OPTION
mov	!rx,fr	W,Z,!Rx	MOVE fr into Port Rx control register
mov	!rx,#lit	W,!Rx	MOVE lit into Port Rx control register
movb	frbit,frbit2	frbit	MOVE frbit2 to frbit
movb	frbit,/frbit2	frbit	MOVE ~frbit2 to frbit
or	fr,#lit	fr,W,Z	OR lit into fr
or	fr,fr2	fr,W,Z	OR fr2 into fr
sub	fr,#lit	fr,W,C,DC,Z	SUBTRACT lit from fr
sub	fr,fr2	fr,W,C,DC,Z	SUBTRACT fr2 from fr
subb	fr,frbit	Z	SUBTRACT frbit from fr
subb	fr,/frbit	Z	SUBTRACT ~frbit from fr
xor	fr,#lit	fr,W,Z	XOR lit into fr
xor	fr,fr2	fr,W,Z	XOR fr2 into fr

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Appendix B

Object File Format

B.1 Intel Hex file formats

This is the most commonly used format for file interchange with
EPROM programmers. A complete Intel Hex file contains one or more hexadecimal records.
The file ends with an end of file record.

Each data record begins with a nine-character prefix and ends with a two-character checksum. Each letter corresponds to one hexadecimal digit in ASCII representation.

Example :BBAAAATTHHHH...HHHCC

Definitions

:	Record start character
BB	Byte count – the hexadecimal number of data bytes in the record.
AAAA	Load address in hexadecimal of first data byte in this record.
TT	Record type. The record type is 00 for data records and 01 for the end record.
HH	One hexadecimal data byte.
CC	Record checksum. This is the 2's complement of the summation of all the bytes in the record from the byte count through the last byte. While the summation is calculated, it is always truncated to a one byte result.

B.1.1 INHX8M: Merged 8-bit Intellex Hex Format

This is the default hex file that will be generated by the SASM cross assembler.

This format produces one 8-bit Hex file with a low-byte/high-byte combination. Since each address can only contain 8 bits in this format, all addresses will be doubled. File extensions for the object code will be '.HEX'.

Example

```
: 0800000001024307000864 0C33
: 080008002100A502040000081C
: 08020000000C0500250026009A
: 080208000600030C0200640C67
: 080210002100A6020A0C3000D7
: 080218000009F0020C0B090BB8
: 08FFE000000000000000000019
: 043FFE00FF0FFF0FA3
: 00000001FF
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

B.1.2 INHX16: 16-bit Hex Format

This format will be output if the INHX16 option is used with the LIST F directive or with the '/f' option on the command line.
This format produces one 16-bit Hex file with a high-byte/low-byte combination. File extension for the object code will be '.HEX'.

Example:

```
: 0800000002010743080000C64002102A5000408005F
: 080100000C0000050025002600060C0300020C6414
: 08010800002102A60C0A0030090002F00B0C0B09BA
: 047FF000000000000000008D
: 021FFF000FFF0FFFC4
: 00000001FF
```

B.1.3 INHX8S: Split 8-bit Intel Hex File Format

This format will be output if the INHX8S option is used with the LIST F directive or with the '/f' option on the command line.

This format produces two 8-bit Hex files, one containing the address/data pairs for the high order 8 bits and the other will contain the low-order 8 bits. File extensions for the object code will be '.HXL' and 'HXH' for low and high order files respectively.

Example:

SAMPLE.HXL:	SAMPLE.HXH:
: 080000000143006421A5040086	: 080000000207080C00020008D1
: 0801000000525260603026438	: 080100000C000000000C000CD3
: 0801080021A60A3000F00C09E9	: 0801080000020C0009020B0BC0
: 047FF000000000008D	: 047FF000000000008D
: 021FFF00FFFFE2	: 021FFF000F0FC2
: 00000001FF	: 00000001FF

B.2 Binary File Format

This format will be output if the BIN16 option is used with the LIST F directive or with the '/f' option on the command line. A pure 16-bit binary file will be generated. A screen dump of SAMPLE.OBJ using DEBUG will be as follows:

```
Debug SAMPLE.OBJ
- d 300 32f

1846:0100   01  02  43  07  00  08  64  0C-21  00  A5 02  04  00  00  08  . . c . . . d . ! . . . . . .
1846:0110   FF  0F  FF  0F  FF  0F  FF  0F-FF 0F  FF  0F  FF  0F  FF  0F  . . . . . .
1846:0120   FF  0F  FF  0F  FF  0F  FF  0F-FF 0F  FF  0F  FF  0F  FF  0F  . . . . . .
1846:0130   FF  0F  FF  0F  FF  0F  FF  0F-FF 0F  FF  0F  FF  0F  FF  0F  . . . . . .
1846:0140   FF  0F  FF  0F  FF  0F  FF  0F-FF 0F  FF  0F  FF  0F  FF  0F  . . . . . .
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

B.3 IEEE-695 File Format

If the /F:IEEE695 switch is given to SASM, the object code and debug information are written to a file which is compliant to the IEEE-695 standard. This output file is named after the source file with the '.SXE' extension.

B.3.1 Target Device

SASM can generate code for either the SX18/SX20/SX28AC or the SX48/SX52BD devices. Since there are subtle differences between these two processors, the specific device specified to SASM is documented in the IEEE-695 file.

The very first record in the IEEE-695 file is the module begin record. One of the parameters in that record is the target device name as an ASCII text string. One of the following strings will appear, indicating the specific device specified in the source file (or on the SASM command line): 'SX18AC', 'SX20AC', 'SX28AC', 'SX48BD', or 'SX52BD'.

B.3.2 Symbols

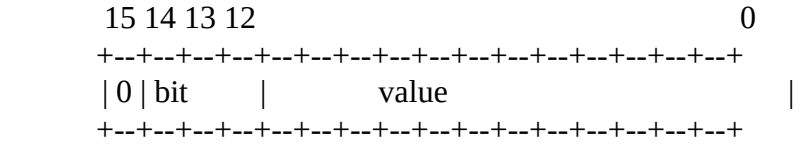
SASM permits symbol table entries to be 32-bit integer values, withan additional 3-bit field to define a bit number. It was determined that some assembly-time calculations are easier if intermediate 32-bitvalues can be saved.

However, the CPU architecture defines at most a 12-bit instruction address and an 8-bit data address.

Since a debugger is most interested in symbols which represent addresses of things, symbol values in the IEEE-695 files are suitably truncated.

In particular, the 32-bit integer value is trucated to 12 bits. If present, the bit number field is placed in bits 12, 13, and 14. Bit 15 is set to zero so that the resulting 16-bit values are guaranteed to be positive.

The following bitmap shows the layout of a SASM symbol value as found in the 16-bit entry in the IEEE-695 file debug section:



B.3.3 Address Spaces

The following conventions are used in symbol values and the actual code segment stored in the IEEE-695 file.

Code addresses are in the range 0x000 to 0x7ff on the SX28AC, and 0x000 to 0xfff on the SX52BD. In both cases, the reset address is at the top of memory. Regardless of the processor, locations 0x0000 through 0x1013 are written to the object file since the 8KB used is likely to be small compared to the symbols and line number information. Unused locations are filled with the value 0xfff.

The ID string is stored one nybble at a time in the low nybbles of addresses 0x1000 to 0x100f. The string is packed high-nybble first.

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

For example, "1234ABCD" is stored as follows:

```
1010:  FF3 FF1 FF3 FF2
1014:  FF3 FF3 FF3 FF4
1018:  FF4 FF1 FF4 FF2
101C:  FF4 FF3 FF4 FF4
```

The FUSE and FUSEX word are part of a 4-location record at 0x1010. The FUSE word is at 0x1010, FUSEX at 0x1011, 0x1012 is unused, and an undocumented code representing the DEVICE is at 0x1013.

For the SX28AC, data addresses are the 8-bit values given in the datasheet, where bits 5, 6, and 7 identify the bank, and the 16 global registers are multiply mapped to the first 16 locations of every bank.

For the SX52BD, SASM uses a 9-bit address which better describes the 256 banked registers and the 16 global locations. In this mapping, addresses 0x000 to 0x00f are the global registers, and 0x010 to 0x10f are the banked registers (bank 1 thru 15 and then bank 0). In addition, SASM allows the user to use addresses from 0x110 to 0x1ff as a second mapping of the first 15 banks.

B.3.4 Assembly-Time Environment

SASM puts records in the IEEE-695 file documenting some trivia about the runtime environment at the moment SASM is invoked. The detailed content of some of these fields will change from run to run, making it difficult to compare the resulting .SXE files even when no source changes have been made.

In addition, some of these fields will differ in the 16-bit DOS executable build of SASM as compared to the Win32 build.

The variable environment information includes a time stamp documenting when the assembler was run, a copy of the command line (including the name by which SASM itself was invoked), and the success/failure of the assembly.

B.3.5 Line Numbers

SASM will include records in the line number table in the IEEE-695 file for each source line (including macro expansions) which generates any words in the code segment. Line number records will correctly reflect the actual source file, line number, and code offset.

© 2000 Scenix Semiconductor, Inc. All rights reserved.

SCENIX

Appendix C

SXREG.INC Definition File

```
0001                            ;SX52INST.SRC
0002                            ; Demonstrate every mnemonic of the SX52
0003
0004    =00000042               lit     equ     $42
0005    =0000001F               fr      equ     $1f
0006    =0000001E.7             frbit   equ     $1e.7
0007    =0000001D               fr2     equ     $1d
0008    =0000000F               imm4    equ     $f
0009
0010    0FFB    0FFF                    device  sx52
0011    =00000000                       org     $0
0012
0017                            ; Logical Operations
0018    0000    017F                    and     fr,w
0019    0001    015F                    and     w,fr
0020    0002    0E42                    and     w,#lit
0021    0003    027F                    not     fr
0022    0004    013F                    or      fr,w
0023    0005    011F                    or      w,fr
0024    0006    0D42                    or      w,#lit
0025    0007    01BF                    xor     fr,w
0026    0008    019F                    xor     w,fr
0027    0009    0F42                    xor     w,#lit
0028
0029                            ; Arithmetic and Shift Operations
0030    000A    01FF                    add     fr,w
0031    000B    01DF                    add     w,fr
0032    000C    007F                    clr     fr
0033    000D    0040                    clr     w
0034    000E    0004                    clr     !wdt
0035    000F    00FF                    dec     fr
0036    0010    02FF                    decsz   fr
0037    0011    02BF                    inc     fr
0038    0012    03FF                    incsz   fr
0039    0013    037F                    rl      fr
0040    0014    033F                    rr      fr
0041    0015    00BF                    sub     fr,w
0042    0016    03BF                    swap    fr
0043
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
0044                        ; Bitwise Operations
0045    0017    04FE            clrb    frbit
0046    0018    07FE            sb      frbit
0047    0019    05FE            setb    frbit
0048    001A    06FE            snb     frbit
0049
0050                        ; Data Movement Instructions
0051    001B    003F            mov     fr,w
0052    001C    021F            mov     w,fr
0053    001D    009F            mov     w,fr-w
0054    001E    0C42            mov     w,#lit
0055    001F    025F            mov     w,/fr
0056    0020    00DF            mov     w,--fr
0057    0021    029F            mov     w,++fr
0058    0022    035F            mov     w,<<fr
0059    0023    031F            mov     w,>>fr
0060    0024    039F            mov     w,<>fr
0061    0025    0042            mov     w,m
0062    0026    02DF            movsz   w,--fr
0063    0027    03DF            movsz   w,++fr
0064    0028    0043            mov     m,w
0065    0029    005F            mov     m,#imm4
0066    002A    0005            mov     !ra,w
0067    002B    0002            mov     !option,w
0068    002C    023F            test    fr
0069
0070                        ; Program Control Instructions
0071    002D    09FF            call    addr8
0072    002E    0BFF            jmp     addr9
0073    002F    0000            nop
0074    0030    000C            ret
0075    0031    000D            retp
0076    0032    000E            reti
0077    0033    000F            retiw
0078    0034    0842            retw lit,lit+1,lit+2
0079    0035    0843
0080    0036    0844
0081
0082                        ; System Control Instructions
0083    0037    0019            bank    fr
0084    0038    0041            iread
0085    0039    0017            page    addr12
0086    003A    0003            sleep
0087
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
0088                        ; Equivalent Assembler Mnemonics
0089    003B    0403            clc
0090    003C    0443            clz
0091    003D    0022            jmp     w
0092    003E    01E2            jmp     pc+w
0093    003F    005F            mode    imm4
0094    0040    0FFF            not     w
0095    0041    0703            sc
0096    0042    0702            skip
0097    0043    0602            skip
0098
0099
0100                        ;----------------------------------
0101                        ;Parallax multi-opcode instructions
0102
0103    0044    0C42            add     fr,#lit
0104    0045    01FF
0105    0046    021D            add     fr,fr2
0106    0047    01FF
0107    0048    06FE            addb    fr,frbit
0108    0049    02BF
0109    004A    07FE            addb    fr,/frbit
0110    004B    02BF
0111    004C    0C42            and     fr,#lit
0112    004D    017F
0113    004E    021D            and     fr,fr2
0114    004F    017F
0115    0050    0CBD            cja     fr,#lit,addr9
0116    0051    01DF
0117    0052    0603
0118    0053    0BFF
0119    0054    021F            cja     fr,fr2,addr9
0120    0055    009D
0121    0056    0703
0122    0057    0BFF
0123    0058    0C42            cjae    fr,#lit,addr9
0124    0059    009F
0125    005A    0603
0126    005B    0BFF
0127    005C    021D            cjae    fr,fr2,addr9
0128    005D    009F
0129    005E    0603
0130    005F    0BFF
0131    0060    0C42            cjb     fr,#lit,addr9
0132    0061    009F
0133    0062    0703
0134    0063    0BFF
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
0135    0064    021D            cjb     fr,fr2,addr9
0136    0065    009F
0137    0066    0703
0138    0067    0BFF
0139    0068    0CBD            cjbe    fr,#lit,addr9
0140    0069    01DF
0141    006A    0703
0142    006B    0BFF
0143    006C    021F            cjbe    fr,fr2,addr9
0144    006D    009D
0145    006E    0603
0146    006F    0BFF
0147    0070    0C42            cje     fr,#lit,addr9
0148    0071    009F
0149    0072    0643
0150    0073    0BFF
0151    0074    021D            cje     fr,fr2,addr9
0152    0075    009F
0153    0076    0643
0154    0077    0BFF
0155    0078    0C42            cjne    fr,#lit,addr9
0156    0079    009F
0157    007A    0743
0158    007B    0BFF
0159    007C    021D            cjne    fr,fr2,addr9
0160    007D    009F
0161    007E    0743
0162    007F    0BFF
0163    0080    0CBD            csa     fr,#lit
0164    0081    01DF
0165    0082    0703
0166    0083    021F            csa     fr,fr2
0167    0084    009D
0168    0085    0603
0169    0086    0C42            csae    fr,#lit
0170    0087    009F
0171    0088    0703
0172    0089    021D            csae    fr,fr2
0173    008A    009F
0174    008B    0703
0175    008C    0C42            csb     fr,#lit
0176    008D    009F
0177    008E    0603
0178    008F    021D            csb     fr,fr2
0179    0090    009F
0180    0091    0603
0181    0092    0CBD            csbe    fr,#lit
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
0182    0093    01DF
0183     0094    0603
0184    0095    021F            csbe    fr,fr2
0185    0096     009D
0186    0097    0703
0187    0098     0C42           cse     fr,#lit
0188    0099     009F
0189    009A     0743
0190    009B     021D           cse     fr,fr2
0191    009C     009F
0192    009D     0743
0193    009E    0C42            csne    fr,#lit
0194    009F     009F
0195    00A0     0643
0196    00A1     021D           csne    fr,fr2
0197    00A2     009F
0198    00A3     0643
0199    00A4     02FF           djnz    fr,addr9
0200    00A5     0BFF
0201    00A6     03FF           ijnz    fr,addr9
0202    00A7     0BFF
0203    00A8     06FE           jb      frbit,addr9
0204    00A9     0BFF
0205    00AA    0603            jc      addr9
0206    00AB     0BFF
0207    00AC     07FE           jnb     frbit,addr9
0208    00AD     0BFF
0209    00AE     0703           jnc     addr9
0210    00AF     0BFF
0211    00B0     0743           jnz     addr9
0212    00B1     0BFF
0213    00B2     0643           jz     addr9
0214    00B3     0BFF
0215    00B4     0C42           mov     fr,#lit
0216    00B5     003F
0217    00B6     021D           mov     fr,fr2
0218    00B7     003F
0219    00B8     0042           mov     fr,m
0220    00B9     003F
0221    00BA     021F            mov     m,fr
0222    00BB     0043
0223    00BC     021F           mov     !option,fr
0224    00BD     0002
0225    00BE     0C42           mov     !option,#lit
0226    00BF     0002
0227    00C0     021F           mov     !ra,fr
0228    00C1     0005
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
0229    00C2    0C42            mov     !ra,#lit
0230    00C3    0005
0231    00C4    07FE            movb    frbit,frbit
0232    00C5    04FE
0233    00C6    06FE
0234    00C7    05FE
0235    00C8    06FE            movb    frbit,/frbit
0236    00C9    04FE
0237    00CA    07FE
0238    00CB    05FE
0239    00CC    0C42            or      fr,#lit
0240    00CD    013F
0241    00CE    021D            or      fr,fr2
0242    00CF    013F
0243    00D0    0C42            sub     fr,#lit
0244    00D1    00BF
0245    00D2    021D            sub     fr,fr2
0246    00D3    00BF
0247    00D4    06FE            subb    fr,frbit
0248    00D5    00FF
0249    00D6    07FE            subb    fr,/frbit
0250    00D7    00FF
0251    00D8    0C42            xor     fr,#lit
0252    00D9    01BF
0253    00DA    021D            xor     fr,fr2
0254    00DB    01BF
0255
0256
0257    =000000FF               org     $ff
0258    =000000FF       addr8
0259    =000001FF               org     $1ff
0260    =000001FF       addr9
0261    =00000FFF               org     $fff
0262    =00000FFF       addr12
0263
0264                    end
Cross Reference
9 symbols

Symbol                          Type    Value       Line
addr12                          ADDR    00000FFF    0164
addr8                           ADDR    000000FF    0160
addr9                           ADDR    000001FF    0162
fr                              DATA    0000001F    0005
fr2                             DATA    0000001D    0007
frbit                           DATA    0000001E.7  0006
imm4                            DATA    0000000F    0008
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
lit                             DATA        00000042    0004
w                               RESV        00000000    0053
;SX52INST.SRC
; Demonstrate every mnemonic of the SX52

lit         equ         $42
fr          equ         $1f
frbit       equ         $1e.7
fr2         equ         $1d
imm4        equ         $f

            device      sx52
            org         $0

;
; Logical Operations
            and         fr,w
            and         w,fr
            and         w,#lit
            not         fr
            or          fr,w
            or          w,fr
            or          w,#lit
            xor         fr,w
            xor         w,fr
            xor         w,#lit

; Arithmetic and Shift Operations
            add         fr,w
            add         w,fr
            clr         fr
            clr         w
            clr         !wdt
            dec         fr
            decsz       fr
            inc         fr
            incsz       fr
            rl          fr
            rr          fr
            sub         fr,w
            swap        fr

; Bitwise Operations
            clrb        frbit
            sb          frbit
            setb        frbit
            snb         frbit
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
; Data Movement Instructions
        mov     fr,w
        mov     w,fr
        mov     w,fr-w
        mov     w,#lit
        mov     w,/fr
        mov     w,--fr
        mov     w,++fr
        mov     w,<<fr
        mov     w,>>fr
        mov     w,<>fr
        mov     w,m
        movsz   w,--fr
        movsz   w,++fr
        mov     m,w
        mov     m,#imm4
        mov     !ra,w
        mov     !option,w
        test    fr

; Program Control Instructions
        call    addr8
        jmp     addr9
        nop
        ret
        retp
        reti
        retiw
        retw lit,lit+1,lit+2

; System Control Instructions
        bank    fr
        iread
        page    addr12
        sleep

; Equivalent Assembler Mnemonics
        clc
        clz
        jmp     w
        jmp     pc+w
        mode    mm4
        not     w
        sc
        skip
        skip
```

© 2000 Scenix Semiconductor, Inc. All rights reserved.

```
;;Parallax multi-opcode instructions

        add     fr,#lit
        add     fr,fr2
        addb    fr,frbit
        addb    fr,/frbit
        and     fr,#lit
        and     fr,fr2
        cja     fr,#lit,addr9
        cja     fr,fr2,addr9
        cjae    fr,#lit,addr9
        cjae    fr,fr2,addr9
        cjb     fr,#lit,addr9
        cjb     fr,fr2,addr9
        cjbe    fr,#lit,addr9
        cjbe    fr,fr2,addr9
        cje     fr,#lit,addr9
        cje     fr,fr2,addr9
        cjne    fr,#lit,addr9
        cjne    fr,fr2,addr9
        csa     fr,#lit
        csa     fr,fr2
        csae    fr,#lit
        csae    fr,fr2
        csb     fr,#lit
        csb     fr,fr2
        csbe    fr,#lit
        csbe    fr,fr2
        cse     fr,#lit
        cse     fr,fr2
        csne    fr,#lit
        csne    fr,fr2
        djnz    fr,addr9
        ijnz    fr,addr9
        jb      frbit,addr9
        jc      addr9
        jnb     frbit,addr9
        jnc     addr9
        jnz     addr9
        jz      addr9
        mov     fr,#lit
        mov     fr,fr2
        mov     fr,m
        mov     m,fr
        mov     !option,fr
        mov     !option,#lit
        mov     !ra,fr
```

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

```
        mov     !ra,#lit
        movb    frbit,frbit
        movb    frbit,/frbit
        or      fr,#lit
        or      fr,fr2
        sub     fr,#lit
        sub     fr,fr2
        subb    fr,frbit
        subb    fr,/frbit
        xor     fr,#lit
        xor     fr,fr2


        org     $ff
addr8
        org     $1ff
addr9
        org     $fff
addr12

end
```

SCENIX

Appendix D

Error Message

1	Bad instruction statement
2	Redefinition of symbol
3	Symbol is not defined
4	Symbol is a reserved word
5	Missing operand(s)
6	Too many operands
7	Missing file register
8	Missing literal
9	Missing Label
10	Missing right parenthesis
11	Missing expression
12	Redefinition of MACRO label
13	Bad expression
14	Bad argument
15	Bad MACRO expression
16	Macro argument do not match
17	Unmatched MACRO
18	Bad IF-ELSE-ENDIF statement
19	Unmatched ELSE
20	Unmatched ENDIF
21	File nesting error - too deep
22	If.else.endif nesting error - too deep
23	Bad numeric string format
24	Value is out of range
25	Bad radix value
26	Unknown microcontroller type
27	Unknown output format
28	Unknown listing parameter
29	Bad string syntax
30	Overwriting same program counter location
31	Expected an \'=\' sign
32	Unexpected EOF
33	Assume value is in HEXADECIMAL
34	Token length exceeds limit
35	Illegal character - Ignored
36	File register truncated to 5 bits
37	Literal truncated to 8 bits
38	Missing RAM Bank bits
39	No destination bit
40	Destination bit can only be 0 or 1

 © 2000 Scenix Semiconductor, Inc. All rights reserved.

41	Bit number out of range
42	Address change across page boundary
43	Address exceeds memory limit
44	Address is not within lower half of memory page
45	Label must begin at column 1

© 2000 Scenix Semiconductor, Inc. All rights reserved.