In-Field Programming Using the SX In-System Programming Capability

Application Note 24
November 2000

1.0 Introduction

With minimal custom hardware and software, the In-System Programming (ISP) feature of Ubicom's SX communications provides the capability to update firmware in the field and allows production line programming. Users that have designed SX devices in their products can provide customer upgrades without the need for a site visit.

2.0 Ubicom's ISP

Ubicom ISP mode of programming requires just two of the SX device pins (OSC1 and OSC2). Once the chip is placed into ISP mode, data is streamed serially into and out of the device through the use of command op codes. The reader is urged to read SX In-System Programming Specifications for complete details of ISP mode. This appnote describes the additional hardware and software that may be used to implement ISP programming in an embedded design using Ubicom's SX family of devices.

3.0 Additional Hardware for ISP

In a typical SX design, a high-speed oscillator circuit is used to operate the device. This usually consists of a crystal/resonator oscillator with a feedback resistor and two load capacitors. Additional hardware is required if in-field ISP capability is desired. This hardware must be able to provide the signal levels and timing required for ISP, but must not interfere with the execution of the normal oscillations when the system is running. At high frequencies (e.g., 50 MHz), any significant changes in capacitance could cause the normal oscillations to shift in frequency, attenuate, or even stop.

3.1 Additional Hardware Requirements

The additional hardware must minimally provide:

- Between +12VDC to +12.5VDC (regulated) for the programming voltage (Vpp)
- Capabilities to switch Vpp "on" and "off" at a relatively fast rate
- Capability to "unload" the ISP circuit from the oscillator circuit during normal program execution
- One unidirectional line to control the OSC1 pin
- One bi-directional line to read/write the data and commands on the OSC2 pin
- An external storage/control device that provides the required ISP control program and application firmware.

It is also desirable to provide these additional capabilities:

- A programming current (Ipp) of greater than 10 mA
- A means to "enable" ISP mode
- Additional control/status lines to control and monitor the status of the ISP

Although the 10 mA programming current exceeds the Ubicom ISP specification, the higher current alleviates the need to provide the +5V signal that is normally specified for entering ISP mode (refer to In-System Programming Specifications Manual) therefore reducing hardware costs. A mechanism to prevent erroneously entering into ISP mode must also be implemented. Additional control and status lines provide more flexibility to the external monitoring/control device (as will be shown in the following example).

4.0 An Example of In-Field ISP

Ubicom has successfully implemented an In-Field ISP design with customer's end product in mind. This design uses low-cost hardware on an embedded system with the SX device being used in the system. A PC program is used to implement the ISP control. The PC parallel port is used as an interface. The software provides the ISP signaling directly over the parallel cable with no requirement for additional external hardware. Users of the end product can obtain firmware updates (e.g., from the customer's Internet site) and upgrade their product without adding or removing any of the normally-installed cables.

Figure 1 shows the example circuit. From a functional view, the circuit provides:

- A momentary pushbutton that "enables" ISP mode
- Means to switch the OSC1 voltage between Vpp, 0V, and "float"
- Means to change the OSC2 pin between input, output, and "float"
- Means to optionally monitor the ISP enable pushbutton

The momentary pushbutton switch enables the ISP circuitry, but only after the user presses the button and only

when the STOBE and INIT lines from the parallel port are in the correct state. Since the PC software sets the parallel port lines, the pushbutton is always disabled until ISP mode is “allowed”. When the PC software prompts the user to push the button, the STROBE and INIT lines both set low. Pushing the button clocks a low signal through the flip-flop which enables the ISP circuitry. Additionally, a high signal is clocked onto the PRGEN line which may optionally be used to disable other circuitry while the ISP is active. The RESET signal is provided by the CPU reset hold-off circuit (not shown) and provides an initial state for the flip-flop.

Once the ISP circuitry is enabled, the PC software uses data bits D0 and D2 to control the voltage on the SX OSC1 pin. The PC uses the following truth table:

D2	D0	OSC1
0	0	“float”
0	1	pulled low
1	0	pulled high to Vpp
1	1	unused

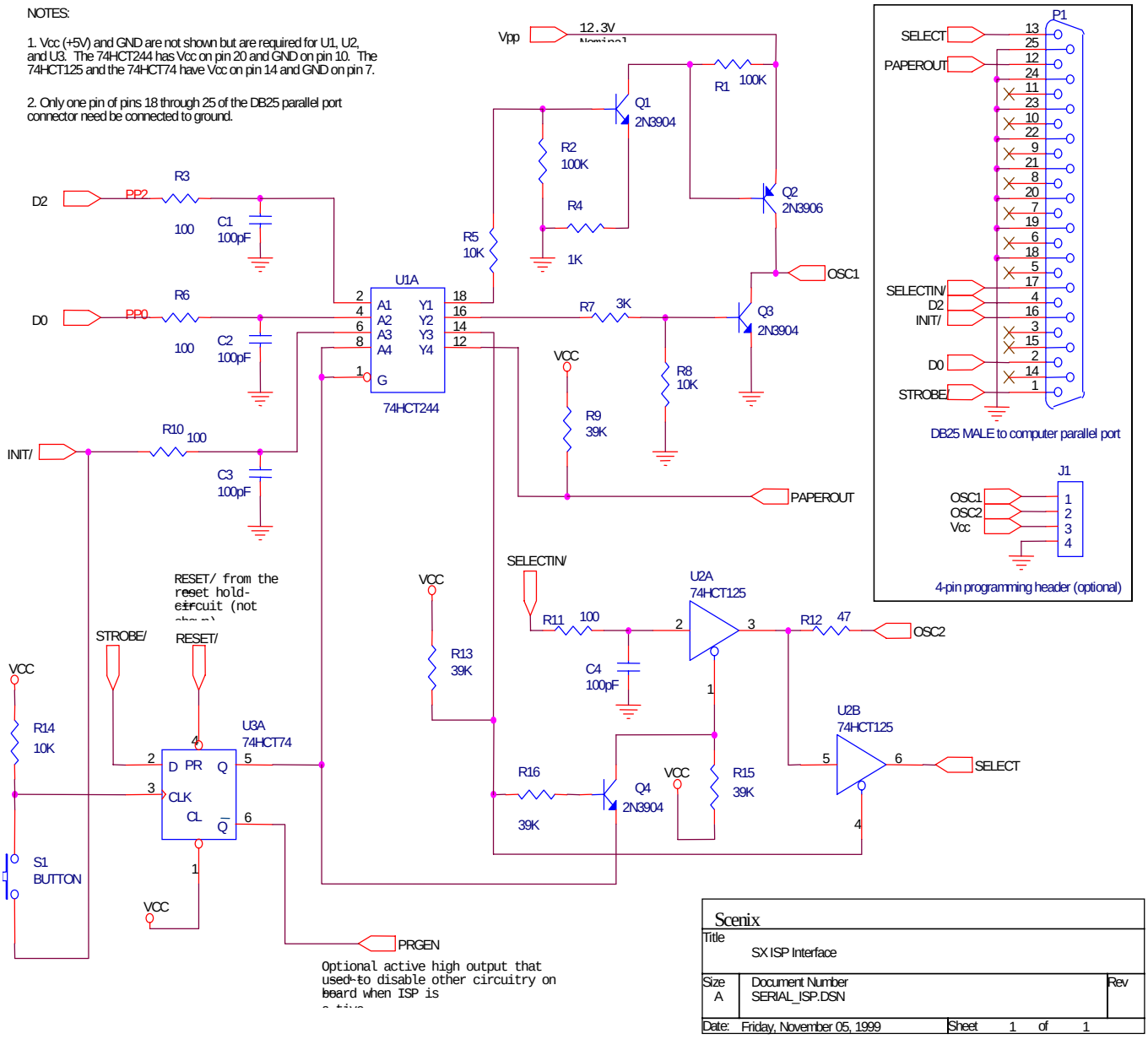

Figure 4-1. SX-ISP Hardware Schematic

After the user presses the pushbutton to enable the ISP circuitry, the PC software uses the parallel port INIT sig-

nal to change the direction of the data stream on the SX OSC2 pin as follows:

$\overline{\text{INIT}}$	OSC2
0	output
1	input

When OSC2 is in the output state, the PC software uses the SELECT line to read the data stream from the CPU. When OSC2 is in the input state, the PC software uses the SELECTIN line to write data to the CPU.

The PAPEROUT pin is provided as a means for the PC software to determine the state of the pushbutton during various states of programming the CPU. The current PC software does not use this function.

4.1 Vpp

If a source of +12VDC is available on the user circuit, providing the required source for Vpp may be as simple as adding an adjustable voltage regulator tuned to +12.3V. If the higher voltage is not available, it may be possible to generate the required source from the logic supply voltage (Vcc). A boost circuit may be used to increase the voltage to the Vpp level. The topology of the boost circuit is highly dependent on the characteristics of the available Vcc. Options include (but are not limited to) simple voltage multiplier circuits, analog boost circuits, and “off-the-shelf” boost ICs such as those provided by Maxim and National Semiconductor.

5.0 SXISP Software

The SXISP software provides Windows 95/98 users with a simple, single-window interface for programming the SX28AC and SX52BD devices. The window consists of several frames and buttons that allow the user to program the SX program memory, read the contents of the program memory, and verify that the contents of the program memory are equal to a specified file.

Files used for programming are in INTEL HEX format and follow the structure of the assembled output of the SX assembler.

Refer to Section 10.0 for instructions on operating the PC software.

5.1 User Interface

SXISP presents a simple dialog interface subdivided into several frames. A set of command buttons serves to initiate the three primary functions: Programming, Verifying and Reading the SX device.

5.2 Program (or Hex) File Selection

When programming or verifying the SX device, a program (hex) file must be selected. There are four separate controls to assist the user in selecting the desired file: File name text box (upper left corner of frame), file list box (lower left corner of frame), folder list box (upper right) and drive list box (lower right). First, select the appropriate drive; click on the down arrow for a drop-down list and click on the desired drive. This will initiate a refresh of the folder list box, showing the folders on the selected drive. Click on the folder where the program hex file exists (double clicking expands the tree view of sub folders). Clicking on a folder causes a refresh of the file list box, showing a list of .SXH and .HEX files in that folder. Click on the desired program hex file; the file text box will be updated with the specified program file name.

If a special file (one which does not have the .SXH or .HEX file extension) is desired, select the appropriate drive and folder location and type the full file name in the file text box at the upper left corner of the frame.

5.3 Chip Type Selection

If the target SX device is a 28-pin, 2K program word device, select the SX28AC radio button in the Chip Type frame. Else, if the target is a 52-pin, 4K program word device, click on the SX52AC radio button.

5.4 Parallel Port Selection

The prototype board must be attached to parallel port on the host computer via a standard DB-25 cable. If the host computer has more than one parallel port, note which port the prototype board was attached to (i.e. LPT1, LPT2 or LPT3). Select the parallel port device for SXISP to use by clicking on the appropriate radio button.

5.5 Program Time Configuration

SXISP allows the user to configure the minimum times for

1. Erasing the device.
2. Programming the FUSEX and FUSE words.
3. Programming a data word

These values equate to the number of milliseconds that SXISP will repetitively send the respective ISP command to the SX device. Note that the *Ubicom SX Device Programming Specification* clearly states the minimum times for reliable erasure and programming. Take care when altering these values to be less than those specified by Ubicom as programming failures may occur. SXISP will do a read-back-verify step for each word programmed in order to limit failures.

6.0 Programming the SX Device

To program the SX Device, the following parameters must be configured:

1. Select the program hex file containing the code to be programmed.
2. Select the SX Chip Type – either an SX-28 or an SX-52.
3. Select the appropriate parallel port to which the prototype board is attached.
4. Optionally set the minimum program times for Erasure, FUSEX and FUSE word writes and program data word writes.
5. Click on the *Program* button.

SXISP will configure the port and board appropriately, in preparation for the SX device entering ISP mode. A status dialog is displayed and the user is then prompted to press the switch/button on the prototype. Once the button is pressed and the user clicks the *Continue* button on the

SXISP Status dialog, the SX device is placed into ISP mode.

If SXISP is unable to set the device into ISP mode, an error message will appear. In this case, ensure the following:

1. The prototype board has been powered up.
2. The cable is properly connected to the prototype board and also to the appropriate parallel port on the host computer.
3. The parallel port setting on the SXISP window properly reflects the parallel port in use.
4. There are no other devices, such as printers, external disk or tape drives, software dongles or communication devices attached to the parallel port.
5. The specified parallel port is properly configured in the host computer BIOS.
6. The specified parallel port is properly configured in Windows. In Control Panel, System, Device Manager, Ports, ensure that the specified parallel port exists in the list and that an exclamation point encircled in yellow does not appear over the port device icon in this list.

Once the SX device is successfully placed into ISP mode, the device will first be erased. In the Status dialog, a count from 0 to 100 percent of the word locations erased, indicates the erase progress. An error count also appears showing the number of erase commands that failed.

If errors occur during the erase process, it does not necessarily mean that the erasure failed. The number of acceptable errors can depend upon the erase time specified and the device type. The only definitive way to know if the erase failed is during the program phase; If there are frequent failures during the program phase, it is likely that the erase phase failed.

Once the device is erased, the next step is to program the FUSEX and FUSE words respectively. It should take a bit longer to program these words and several retries may be involved depending on the program times specified. SXISP will continue to try to program the word until it successfully reads and verifies the value.

The next phase it to program each data word. The status dialog will display the progress, i.e. words programmed and number of errors. In this phase, an error indicates that a word or words was unsuccessfully programmed. It is likely that the program will not run as intended or may not run at all. The device should be reprogrammed. If errors persist on subsequent programming attempts, try increasing the erase time.

When each word is programmed, the user is prompted again to press the button on the prototype board and click the *Continue* button. The SX device is then taken out of ISP mode and is reset.

7.0 Verifying an SX Device

To verify an SX Device, the following parameters must be configured:

1. Select the program hex file containing the code to be compared to the existing code burned into the SX flash.
2. Select the SX Chip Type – either an SX28AC or an SX52BD.
3. Select the appropriate parallel port to which the prototype board is attached.
4. Click on the *Verify* button.

The SXISP Status dialog is displayed and the user is prompted to press the button on the prototype board. Once the button is pressed and the user clicks the *Continue* button on the status dialog, the SX device is placed into ISP mode.

SXISP will first read and compare the FUSEX word, then the FUSE word and the each data word in the SX flash and compare it to its corresponding word in the program hex file. If any mismatches are detected, the error count on the status dialog is incremented to reflect the discrepancy. SXISP also reads and verifies the 16 words after the last word of program memory.

When each word is read and verified, the user is prompted again to press the button on the prototype board and click the *Continue* button. The SX device is then taken out of ISP mode and is reset.

8.0 Reading the SX Device

To read an SX Device, the following parameters must be configured:

1. Select the SX Device Type or Chip Type – either an SX28AC or an SX52BD.
2. Select the appropriate parallel port to which the prototype board is attached.
3. Click on the *Read* button.

The SXISP Status dialog is displayed and the user is prompted to press the reset button on the prototype board. Once the reset button is pressed and the user clicks the *Continue* button on the status dialog, the SX device is placed into ISP mode.

SXISP will first read the FUSEX word, then the FUSE word and then each word of the SX program flash. The contents of the flash is displayed on the SXISP window within the EEPROM Contents frame.

After each word is read the user is prompted again to press the reset button on the prototype board and click the *Continue* button. The SX device is then taken out of ISP mode and is reset.

The FUSEX, FUSE and data words are displayed in 3-digit hexadecimal format, 16 words per line. Each line of data words is preceded by the address of the first data word in the line. Addresses are displayed in hexadecimal format as well. The contents also include the 16 extra words past the last word of program memory.

9.0 Errors and Troubleshooting

SXISP makes every effort to detect errors and report the cause and possible solution in a pop-up message box or in the message area of the SXISP Status dialog. If at any time during a program cycle, execution seems unusually slow or frequent errors are occurring, the user can click the *Cancel* button on the SXISP Status dialog to abort the cycle.

When frequent errors occur during programming, first try cycling power to the prototype board, double-check the cable connection between the prototype board an the PC's parallel port and ensure that another process on the host Windows PC has not taken control of your parallel port.

10.0 SXISP Dynamic Link Library Exported Function Descriptions

10.1 Introduction

This section includes manual pages for the Application Programming Interface (API) of the SXISP 32-bit Windows® dynamic link library (DLL) for use on a Windows 95 or Windows 98 platform. This DLL will not function on a Windows NT platform nor does it support the UNICODE or "wide" character set at this time.

Note:The "USHORT" or "short" data type definition describes a 16-bit integer value ranging from 0 to 65,535. This corresponds to the "Integer" data type in Microsoft Visual Basic. LPSTR indicates a 32-bit pointer to a null-terminated string of characters. In Visual Basic, this data type is a "String". All function arguments from Visual Basic should be passed by value ("ByVal").

10.2 High-Level Interface

The high-level API provides the easiest facility for programming an SX device. Just three functions program the three basic operations of Programming, Verifying and Reading a device.

10.2.1 Program SX Device

Prototype:

```
int SxProgram(USHORT  usType,
    USHORT  usPort,
    LPSTR   lpszHexFile,
    USHORT  usEraseTime,
    USHORT  usProgramTime,
    USHORT  usFusexTime)
```

Function Arguments:

usType	SX Device Type
usPort	Parallel port number (1 = LPT1, 2 = LPT2, 3 = LPT3)
lpszHexFile	Full Drive:\Path\FileName of the program hex file
usEraseTime	Number of milliseconds to send ISP Erase commands
usProgramTime	Number of milliseconds to write a data word
usFusexTime	Number of milliseconds to write the FUSEX and FUSE words.

Return Value:

Function returns zero if successful, else returns SXISP_ERROR (defined in SXISP.H).

Comments:

Function displays a status dialog during programming. On errors, a message box is presented containing information about the error and possible solutions.

10.2.2 Verify SX Device

Prototype:

```
int SxVerify(USHORT  usType,
             USHORT  usPort,
             LPSTR   lpszHexFile)
```

Function Arguments:

usType	SX Device Type
usPort	Parallel port number (1 = LPT1, 2 = LPT2, 3 = LPT3)
lpszHexFile	Full Drive:\Path\FileName of the program hex file

Return Value:

Function returns zero if successful, else returns SXISP_ERROR (defined in SXISP.H).

Comments:

Function displays a status dialog during verification. On errors, a message box is presented containing information about the error and possible solutions.

10.2.3 Read SX Device

Prototype:

```
int SxRead(USHORT  usType,
           USHORT  usPort,
           LPSTR   lpszOutFile)
```

Function Arguments:

usType	SX Device Type
lpszOutFile	Parallel port number (1 = LPT1, 2 = LPT2, 3 = LPT3)
lpszOutFile	Full Drive:\Path\FileName of the output file

Return Value:

Function returns zero if successful, else returns SXISP_ERROR (defined in SXISP.H).

Comments:

Function displays a status dialog during read. On errors, a message box is presented containing information about the error and possible solutions.

The output data words are formatted as three-digit hexadecimal values. Each line is preceded by the program memory address expressed as four-digit hexadecimal values. The FUSEX and FUSE words appears in the beginning.

Example Output:

```
FUSEX: FDE
FUSE: 4FA

0000: 019 21B 743 0FB 01A 216 743 0F6 C01 08B
643 A37 019 2B8 216 098
0010: 703 A1F C00 038 2B5 217 036 C00 099 643
A1F 219 93D 036 0F9 01C
0020: 2B0 C32 090 703 A37 C00 030 2B1 CFA 091
703 A37 C00 031 403 332
```

10.3 Mid-Level Interface

This API provides the caller with much greater control over the programming process, affording the user interface with the ability to:

1) manipulate the program image after reading it from the program hex file.
2) manually control each step of programming an SX device.

Standard Return Values

Each of these functions returns a standard set of values under certain conditions. Some functions return specific error values that are documented where necessary. The standard return values are:

0 = Success

-1 = Memory Allocation Error. Fatal error - the process heap is full. SXISP uses 4-5 kilobytes of heap memory. Under most scenarios, this error should not occur.

OR

A necessary object has not yet been initialized. Fatal error – the appropriate object initialization function was not called prior to called this function. For example, before calling SxFileGetWord() to read a word from the program image, the caller must first call SxFileOpen() to read the program hex file.

-2 = The specific intent of the function failed. For example, SxFileClose() returns –2 if a file has not been opened. Or, SxIspEnter() returns –2 when the SX device fails to enter ISP mode.

-3 = The object has already been initialized. This can occur when SxIspInit() is called twice without an intervening SxIspDone() call.

10.3.1 Program Hex File Management

10.3.1.1 Open Program Hex File

Prototype:

```
short  SxFileOpen(LPSTR lpszFileName)
```

Purpose:

Opens the program hex file for reading – ensures that the specified files exists. Reads the program data, parses it and creates a program image. The program image is saved in memory and becomes the default image for all subsequent program operations.

Function Arguments:

`lpszFileName`: The program hex file name (pointer to null-terminated character string). Includes Drive:\Path\Filename.Ext if not in the current directory. The file must exist.

Return Value:

-2 File does not exist or could not be read.

Comments:

Only one program hex file can be used at a time. The actual file does not remain open – it is opened, read, parsed and then closed before returning.

If a hex file is already loaded, the currently-loaded image is overwritten. It is, however, recommended that a call to SxFileClose() precede a subsequent call to SxFile-Open().

10.3.1.2 Close Program Hex File

Prototype:

```
short  SxFileClose(void)
```

Purpose:

Releases the currently-loaded program hex file image from memory.

Function Arguments:

None.

Return Value:

-2 No program hex file is currently loaded.

10.3.1.3 Get Program Word

Prototype:

```
short  SxFileGetWord(short sWord)
```

Purpose:

Fetches the specified program word from the loaded program image.

Function Arguments:

`sWord`: The address location (zero-relative) of the word.

Non-program device words are specified using the following values:

-1 = FUSEX

-2 = FUSE

Return Value:

The 12-bit value of the specified program word or a standard error code.

10.3.1.4 Put Program Word

Prototype:

```
short  SxFilePutWord(short sWord, short sValue)
```

Purpose:

Write a program data word to the program image at the specified location.

Function Arguments:

`sWord`: The address location (zero-relative) of the word.

Non-program device words are specified using the following values:

-1 = FUSEX

-2 = FUSE

`sValue`: New 12-bit program data word value.

Return Value:

The NEW 12-bit value of the specified program word or a standard error code.

10.3.1.5 Program Image Size

Prototype:

```
short  SxFileGetCount(void)
```

Purpose:

Get a count of the number of program data words in the program image.

Function Arguments:

None.

Return Value:

Number of program data words in the program image.

10.3.1.6 Calculate Checksum

Prototype:

```
USHORT SxFileChecksum(short sWords)
```

Purpose:

Calculate a 16 bit twos complement checksum of the program image.

Function Arguments:

sWords: Number of words to include in the checksum.

Return Value:

16 bit twos complement checksum value.

Comments:

The number of words to include in the checksum calculation assumes that the calculation begins with word location zero and proceeds until *sWords* have been included.

10.3.2 In System Program (ISP) Mode Control

10.3.2.1 Initialize

Prototype:

```
short  SxIspInit(short sPort, short sErase,
short sProgram, short sFusex)
```

Purpose:

Initialize “Isp” object in the following steps:

1. Allocate storage for the Isp class object.
2. Load and display the status dialog
3. Load and initialize the virtual device driver (VSX-ISPD.VXD)
4. Open the specified parallel port and configured it to standard bi-directional mode.
5. Configure the program write timing values.
6. Instruct user to press the button on the prototype board.
7. Enter ISP mode

Function Arguments:

sPort:	Parallel port number (1 = LPT1, 2 = LPT2, 3 = LPT3)
sErase:	Number of milliseconds to erase the SX device
sProgram:	Number of milliseconds to write a program data word
sFusex:	Number of milliseconds to write the FUSEX and FUSE words

Return Value:

Standard error codes and:

-2 Initialization failed.

Comments:

At this time, the specific step that failed is not known.

10.3.2.2 Shut Down

Prototype:

```
short  SxIspDone(void)
```

Purpose:

Stop, shut down, release, deallocate, etc. Signals component that the caller is finished with this iteration of an ISP event.

Function Arguments:

None.

Return Value:

-1 Not initialized.

10.3.2.3 Enter ISP Mode

Prototype:

```
short  SxIspEnter(void)
```

Purpose:

Places the SX device into the In-System Programming (ISP) mode.

Function Arguments:

None.

Return Value:

Standard error codes.

-2 The device failed to enter ISP mode.

10.3.2.4 Exit ISP Mode

Prototype:

```
short  SxIspExit(void)
```

Purpose:

Reverts device from In-System Programming (ISP) mode to normal operating mode (generates and internal reset).

Function Arguments:

None

Return Value:

Standard return codes

-2 Device was not in ISP mode.

10.3.2.5 Erase Device

Prototype:

```
short  SxIspErase(void)
```

Purpose:

Erase the SX device

Function Arguments:

None.

Standard return codes

-2 Device may not have been successfully erased. Two conditions may cause such a return value: 1) Device does not appear to be in ISP mode, 2) At least 50 erase command write attempts failed. Note that in some cases, the device may have been successfully erased if a sufficient number of erase command were received.

10.3.2.6 Increment Address

Prototype:

```
short  SxIspIncrement(void)
```

Purpose:

Increment the SX device internal address pointer to the next memory location.

Function Arguments:

None

Return Value:

Standard return codes

-2 Pointer was not advanced – increment command could not be sent. Device may have exited ISP mode.

Comments:

After entering ISP mode, the address pointer points to the FUSE word. The first increment then points the internal address pointer to program location zero.

The address pointer cannot be decremented nor “wrapped” around to address location zero. To accomplish these tasks, the device must be taken out of ISP mode and then placed back into ISP mode and the address pointer incremented appropriately.

10.3.2.7 Read Word

Prototype:

```
short  SxIspRead(short sType)
```

Purpose:

Read a word from the SX device

Function Arguments:

sType: An identifier indicating which word is to be read:

0 = Program Data word

-1 = FUSEX word

-2 = FUSE word

-3 = DEVICE word

The program data word is read from the location that the internal address pointer currently points to.

Return Value:

The 12-bit value of the specified word. Otherwise, a standard return code. Caller should test the return code to see if it is less than zero in which case an error occurred.

-2 The device was not in ISP mode

10.3.2.8 Write Word

Prototype:

```
short  SxIspWrite(short sType, short sValue)
```

Purpose:

Write a word to the SX device.

Function Arguments:

sType: An identifier indicating which word is to be read:

0 = Program Data word

-1 = FUSEX word

-2 = FUSE word

The program data word is written in the location that the internal address pointer currently points to.

sValue: New word value

Return Value:

Standard return codes:

-2 The device was not in ISP mode.

Comments:

The Device word cannot be altered. The address pointer is NOT incremented after the program data word is written.

10.3.2.9 Display Message

Prototype:

```
short  SxIspMessage( LPSTR lpszMessage, short
                     sWord, short sWords, short
                     sErrors, BOOL bWait)
```

Purpose:

Displays a message string on the SX ISP Status dialog presented by the SXISP component.

The Status dialog has four items that can be altered by the caller: the text message, the current word (x) and the total word count (n) (Word x of n), and the error count.

Function Arguments:

lpszMessage:	The message string – null-terminated string of characters. A NULL value is permitted in which the text will remain the same but the following values are to be altered.
sWord:	Current program data word that the operation is processing (-1 if the value should not be altered).
sWords:	Total number of words to be processed. (unchanged when sWord is -1).
sErrors:	Total number of errors encountered.
bWait:	Boolean flag indicating whether function should block until user clicks the *Continue* or *Cancel* button on the SX ISP Status dialog.

Return Value:

Standard return code or:

Non-zero positive value if the *Cancel* button has been clicked by the user.

10.3.3 User-Specific API

10.3.3.1 Ubicom Application-Specific Device Program

Prototype:

```
short  UbicomProgram(USHORT usPort, LPSTR
lpszHexFile,
                     LPSTR lpszSerialNum,
                     USHORT bProtect,
                     USHORT usEraseT, USHORT
                     usProgramT,
                     USHORT usFusexT)
```

Purpose:

End product program function.

Function Arguments:

usPort:	Parallel port number (1=LPT1, 2=LPT2, 3=LPT3)
lpszHexFile:	Program Hex File (Path) Name
lpszSerial:	Product Serial Number
bProtect:	Code protect (TRUE=Code Protect ON, FALSE=Off)
usEraseT:	Erase time in milliseconds
usProgramT:	Program data word write time in milliseconds
usFusexT:	FUSEX/FUSE word write time in milliseconds

Return Value:

0	Success
-1	Failed

Comments:

Function reads program hex file, calculates checksum and inserts checksum into program image, parses serial number and inserts serial into program image, then programs the device.

Lit#: AN24-02

For the latest contact and support information on SX devices, please visit the Ubicom website at www.ubicom.com. The site contains technical literature, local sales contacts, tech support and many other features.

1330 Charleston Road
Mountain View, CA 94043
(650) 210 - 1500
http://www.ubicom.com