Systronix
Expansion
Hardware

■ Technical Reference for
Systronix SBX2

SBX2

LIMITED WARRANTY

The information in this manual is subject to change without notice and does not represent a commitment on the part of Systronix, Inc. Systronix, Inc. makes no warranty, express or implied, for the use or misuse of its products, which are provided with the understanding that you, the user, will determine fitness for a particular application. Systronix assumes no responsibility for any errors which may appear in this manual. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Systronix, Inc.

Systronix reserves the right to revise this documentation and the software and hardware described herein or make any changes to the specifications of the product described herein at any time without obligation to notify any person of such revision or change.

TRADEMARKS

Systronix is a registered trademark of Systronix Inc, TILT, STEP, STEP+, STEP.IR, SaJe and SBX2 are trademarks of Systronix Inc, INT_EL and Intel are registered trademarks of Intel Corporation. Microsoft, Windows, and MS-DOS are registered trademarks of Microsoft Corporation. TINI is a trademark of Dallas Semiconductor.

Systronix®, Inc.
555 South 300 East
Salt Lake City, UT 84111
TEL: 801-534-1017
FAX: 801-534-1019
www.systronix.com
email: support@systronix.com

Revised - November 8, 2001

Table of Contents

SBX Standard . . . . . Page -1-
SBX2 Addressing and I/O MAP . . . . . Page -1-
SBX2 with TINI and STEP boards . . . . . Page -1-
SBX Interrupts with TINI and STEP . . . . . Page -1-
SBX2 UART with TINI . . . . . Page -1-
SBX2 with HSM550 . . . . . Page -2-
SBX2 with uCAN2 . . . . . Page -2-
SBX2 with SaJe . . . . . Page -2-
Addressing . . . . . Page -2-
Chip Selects . . . . . Page -2-
Interrupts . . . . . Page -2-
SBX2 Address and Data Map . . . . . Page -2-
KEYPAD INTERFACE . . . . . Page -5-
How the Keypad is Scanned and Encoded . . . . . Page -5-
Reading the Keypad Register . . . . . Page -5-
Polling the Keypad . . . . . Page -6-
Using Keypad Interrupts . . . . . Page -6-
Connecting a Keypad . . . . . Page -6-
KEYPAD LEGENDS . . . . . Page -9-
DIGITAL I/O (24 BITS) . . . . . Page -9-
LCD INTERFACE . . . . . Page -9-

SBX2 Java API . . . . . Page -10-

SBX Standard

SBX2 conforms to the iSBX specification as originally defined by Intel as part of Multibus, also defined in Standard IEEE-796. There are reportedly some minor differences between the Intel specification and the IEEE Standard.

We have a copy of the SBX specification in an old Intel *Multibus I Architecture Reference Book*, order number 210883-002, probably long out of print. The IEEE-796 specification is still available (for a fee) from the IEEE; try them on the web at www.ieee.org.

The SBX interface uses 2 chip selects (MCS0 and MCS1), three address lines (A2, A1, and A0), and read (RD) and write (WR) strobes.

SBX2 Addressing and I/O MAP

SBX2 is intended to be memory-mapped into a host board, as part of its data space.

SBX2 with TINI and STEP boards

STEP sockets boards map the SBX connector into data space at 0x38006X for all but the UART, and 0x38008X for the SBX2 UART. This is derived from these equations:

MCS0 = CE3 * A19 * /A16 * /A7 * A6 * A5
MCS1 = CE3 * A19 * /A16 * A7 * /A6 * /A5

Other addresses are don't cares. So making don't care bits '0',
MCS0 = 0x38006[0-7]
MCS1 = 0x38008[0-7]

Notes for above:
* = logical AND
/ = logical negation (not asserted)

The other part of the puzzle is where these addresses [0-7] refer at each chip select. That is described in the SBX2 Address and Data Map.

SBX Interrupts with TINI and STEP

TINI only has one external interrupt, so the two SBX2 interrupts MINT0 and MINT1 are logically ORed by STEP hardware into a single active low EXTINT for TINI.

On SBX2, MINT1(H) is driven directly by the UART interrupt output with no additional logic. MINT0(H) is driven by the keypad scanner logic. When you use SBX2 with STEP, and both the keypad and UART can generate interrupts, you will need to determine the source of the interrupt with a firmware routine which checks both the keypad and UART.

SBX2 UART with TINI

SBX2 UART is TINI's serial2. The standard TINI and javaxcomm serial drivers support SBX2's UART. Since SBX2 uses a 16C550 UART chip which is the same as the former Dallas e50 socket

board's added UARTs, it works exactly the same as the e50 did.

One caution: in TINI firmware version 1.02, the baud rate of external UARTs is limited to 57600. This limit may be removed in future TINI firmware revisions.

SBX2 with HSM550

MCS0 = 0xFC[0-7]X -- parallel port, keypad, LCD, buzzer, etc.
MCS1 = 0xFC[8-F]X -- UART

SBX2 with uCAN2

MCS0 = 10:0C[0-7]X -- parallel port, keypad, LCD, buzzer, etc.
MCS1 = 10:0C[8-F]X -- UART

SBX2 with SaJe

Addressing

SaJe uses a 32-bit processor with a 32-bit data path. SaJe addresses 0 and 1 refer to bytes within a 32-bit word. SBX2 on SaJe is right-justified within the 32-bit data word. On SaJe, SBX2 shifts its addresses left two bits like this:

SBX2 A0 = SaJe address A2
SBX2 A1 = SaJe address A3
SBX2 A2 = SaJe address A4

This enables addressing SBX2 as if it were a 32-bit wide data device and discarding the upper 24 data bits. Instead of occupying low address locations [0x00..0x07], SaJe accesses SBX2 at low addresses [0x00..0x1C] with only every fourth location used.

Chip Selects

On SaJe, SBX2 chip selects occupy the lower portion of CS5. CS5 defines 4 MBytes from 0x0140_000 through 0x017F_FFFF.

MCS0 = 0x0140_0000 through 0x0140_FFFF or 0x0140_XXXX -- parallel port, keypad, LCD, buzzer, etc.

MCS1 = 0x0141_0000 through 0x0141_FFFF or 0x0414_XXXX -- UART

Interrupts

The two SBX interrupts are mapped as follows:

MINT0(H) is tied to GPIOA1. This is the SBX2 keypad interrupt.
MINT1(H) is tied to GPIOA2. This is the SBX2 UART interrupt.

SBX2 Address and Data Map

The following table describes the addresses used by SBX2 and the data mapping at those addresses.

SBX2 Address and Data Map

note that 0 =not asserted and 1= asserted, regardless of voltage level. For example, address lines are asserted high, but chip selects MCSX, RD and WR are asserted low. MCS0=1 means this chip select is asserted (and happens to be active low), while A2=1 is also asserted but active high. Don't confuse voltage levels with the boolean assertion level.

MCS0	MCS1	A[2..0] SaJe A[4..2]	RD	WR	Function
1	0	000	1	0	DINPRD0 - (note 1) read data input register0. Reads the I/O bit state with no inversion. DIO(H) reads as DINP=1. DINP0.7 - DIO7(H) DINP0.6 - DIO6(H) ... DINP0.0 - DIO0(H)
1	0	000	0	1	DOUTWR0 - (note 1) write data output register0, note that setting a bit to a '1' drives that open-drain I/O bit LOW, DOUT0.X=1 means DIOX(L) DOUT0.7 - DIO7(L) DOUT0.6 - DIO6(L) ... DOUT0.0 - DIO0(L)
					DINPRD1
1	0	011	1	0	Keypad read: (note 3) KEY.7 - Keypress now KEY.6 - unused KEY.5 - unused KEY.4 - Y msb KEY.3 - Y KEY.2 - Y lsb KEY.1 - X msb KEY.0 - X lsb
1	0	100	1	0	LCD data read
1	0	100	0	1	LCD data write
1	0	101	1	0	LCD instruction read
1	0	101	0	1	LCD instruction write
1	0	110	1	0	No read function
1	0	110	0	1	MISC data output register U5, with eight bits defined as follows: (note 4) MISC.7 - local LED on (L) MISC.6 - LCD backlight bit 1 (L) MISC.5 - LCD contrast clock MISC.4 - LCD contrast up (H) MISC.3 - LCD backlight bit 0 (L) MISC.2 - buzzer on (L) MISC.1 - external indicator on (L) MISC.0 - UART DCE DCD (L)

1	0	111	1	0	Not used
0	1	XXX	X	X	read and write 16C550 UART when CS1. See UART info for details.

Note 1: the output and input registers are tied together, allowing the input register to read the value of the output register. The output registers are octal, open-drain FET registers. Writing a '0' to an output bit clears the bit, allowing the output to float. All outputs are pulled high through 10 Kohms to 5V. If you want to tie an output to a 12V load that's fine. The existing 5V pullups will simply draw (12-5)/10K = 700 uA of current per bit. Writing a '1' to an output bit causes that open-drain output to go asserted low. An individual FET can sink at least 150 mA, multiple outputs can be limited by the total device power dissipation discussed in the digital I/O section of this manual. The input register reads the state of the I/O bit, through some current-limiting resistors. To use an I/O bit as an input, first drive the output bit with a '0' to let it float high and then pull the I/O bit low with an external switch, relay, or other contact closure. Because the open-drain output buffer inverts, if you write 0xA5 to the output, you will read back 0x5A, the complement, on the input buffer.

Note 3: The keypad can be read at any time. Reads are non-destructive of data but do clear an active keypad interrupt.

Note 4: The LCD is assumed to use an LED backlight with Vcc on Pin 15 of LCD connector P3, and ground on Pin 16. On SBX2, Pin 16 (the LED cathode) is pulled to ground through two 10-ohm current-limiting resistors, one of which may be bypassed by JP1.

LED Backlight Intensity Control

MISC. 6	JP 1	MISC. 4	BRIGHTNESS	COMMENT
0	off	0	off	
0	off	1	low	
0	on	0	off	
0	on	1	medium	
1	off	X	medium	
1	on	X	max	Caution! See note below

Caution: only install JP1 if the LED backlight has a built-in current-limiting resistor, otherwise you will draw too much current through the backlight. This could cause SBX2 to overload the system power bus, reset the host board and/or damage the LED backlight.

KEYPAD INTERFACE

How the Keypad is Scanned and Encoded

The keypad Y inputs are all pulled high through 10 Kohms. The X lines of the keypad are sequentially driven low with a 2-bit counter running at 3.6864 MHz. This creates 272 nsec-wide low pulses on the X lines with a 25% duty cycle. As the X lines are driven, the Y lines are scanned to see if any of the X line pulses appear in the Y inputs. As soon as one appears, then a key is assumed to be making the connection between that X output and that Y input. The value of the X counter gives the encoded row value (0x0, 0x1, 0x2, 0x3) and the value of the Y input on which the pulse was detected is encoded into 3 bits (0x0 - 0x5). This supports a keypad of up to four X lines and five Y lines.

After a power up or reset (via the SBX RST signal), the value in the keypad register is 00. As soon as a keypress is detected, scanning stops and a debounce counter of 16 msec starts. If the keypress bounces or is released completely, the scanner starts over. If the key remains pressed for the debounce period, its value is latched into the keypad register, and the keypad interrupt asserts, driving SBX MINT0 to an asserted state. Keypad data in the register is always debounced. If the key is released, INT0 negates at the next scan clock (270 nsec) and cannot assert again until it has been re-debounced.

Since the data is latched and can't change for at least a 16 msec debounce delay, keypad data cannot possibly change more frequently than every 16 msec. If you are driving the keypad inputs with some kind of digital input (a hall effect switch perhaps), the maximum frequency which can be sampled is 62.5 Hz.

KEYPAD REGISTER (location 3) BIT DEFINITION							
7	6	5	4	3	2	1	0
1=Keypress	0	0	Y msb	Y	Y lsb	X msb	X lsb

Keypad data reads are non-destructive. As long as a key is being pressed, and after it is debounced, the fresh bit (bit 7) will be a 1. If you are polling the keypad, this is how you can detect new keypad data. The keypad interrupt MINT0 asserts HIGH but is cleared by reading the keypad register, or by releasing the key. Data in the keypad register is always debounced and "clean".

If a key is being held down, and the controller responds to it by reading the keypad register, the keypad interrupt MINT0 will be cleared. If the key remains pressed, another interrupt will occur one debounce period later. During this time, as long as the key remains pressed, the Key Fresh bit will remain a 1.

If a key is pressed, but the key is released before the controller responds to it, the keypad data will still be present, but the Key Fresh bit will have cleared to a 0, and MINT0 will no longer be asserted. The last-debounced value will remain readable in the keypad register, until a new key is debounced.

Reading the Keypad Register

The keypad scanner runs off the SBX2 3.6864 MHz crystal, which is asynchronous to the host processor which is running your application. Even though the keypad is debounced and the data in the keypad register is always valid in relation to the keypad, if you are polling the keypad, it could be changing in relation to the host processor just at the moment you read it. The chance this will happen

is very small, but the probability is finite. If the controller data read time is 50 nsec, then the chance of reading a changing value is about 50 nsec out of 16 msec, or about 3 parts per million. This assumes the keypad data changes every 16 msec which of course it does not for manual keypad presses.

Therefore, to be certain your polling controller is reading a valid value, you might consider requiring that the same value be successively read twice.

If you are using an interrupt driven keypad handler, then the keypad data is much less likely to be changing when you respond to the interrupt. As long as you respond within 16 msec the data cannot possibly be changing since 16 msec is the keypad debounce period. If you take more than a debounce period to respond, there are two possibilities. First, the key could be released in which case the Key Fresh bit only might be changing. Second, a different key could have been pressed which will change the keypad data.

Polling the Keypad

If you will be polling the keypad, check the Key Fresh bit to see if a key is currently being pressed. It's hard for a human to press a key more than about twice per second. If you poll for a key 10 times per second you will give the user the feeling of a responsive system. It's possible that the keypad register could be in the process of changing at the moment you poll it, so it is sensible to require two identical reads before trusting the data. (Don't confuse this asynchronous keypad data changing with the key debouncing which is always performed by the SBX2 keypad scanner.)

To implement a spinner, 'jog' a value, or increment/decrement a value at a variable rate, count the number of successive polls for which Key Fresh is asserted and the key data is the same as the last poll. After half a second or so you can switch to a faster update or increment/decrement rate. Clear the counter as soon as Key Fresh is not a 1, or the data changes.

In Java, a thread could be used to poll the keypad every 100-200 msec. If you implement this before I get around to it let me know how you did it and how it worked via email: support@systronix.com.

Using Keypad Interrupts

You can implement a keypad routine based on either an edge- or level-sensitive interrupt. MINT0 is asserted high. Since there will always be some latency from the assertion of the interrupt until you read the register, the data in the register will always be stable when read. As long as you respond within one debounce period (16 msec), there is no way the keypad data can be changing at the instant you read the register. It takes one period to debounce the key, latch the data, and assert the interrupt. If the key is released, or a different key pressed, it will take another debounce period (16 msec) for the data to change from the value it was when the interrupt asserted.

Connecting a Keypad

Connect a "matrix-pinout" keypad with up to 4 rows and 5 columns (or vice versa) to the 1x9 header P1. Keypads are available from Systronix and many electronic supply distributors. Membrane keypads with a flex cable tail usually have a color dot or arrow on pin 1 of their receptacle. J1 is provided to let you add other keypad connectors, or to cut and reroute keypad signals. J1 has straight-through connections on the back side of the PC board connecting one row of pins (which come from the keypad header) to its other row (which are connected to the keypad scanner).

If your keypad requires significantly different row and column wiring, you can cut the traces on the back of the PC board and wire-wrap, solder, or jumper whatever connections you need.

EXAMPLE KEYPAD CONNECTOR PINOUT								
9	8	7	6	5	4	3	2	1
COL5	COL4	COL3	COL2	COL1	ROW4	ROW3	ROW2	ROW1

"Row" and "column" are relative. The keypad scanner doesn't care if you swap them. What is important is that all the row outputs be connected to keypad lines which are orthogonal to the column inputs. You can't have some of the SBX2 row outputs going to keypad rows and some to columns, since the scanner can only detect a connection between the SBX2 row outputs and the column inputs.

If your keypad does not have the above pinout, don't panic! As long as your keypad rows and columns are connected to any rows and columns of the keypad header, you can correct other wiring easily in a keypad lookup table. For example, you can fix a swap between row 1 and row 4 in a lookup table. But if a keypad *row* is swapped with a keypad *column* it may not be possible to sort this out in a lookup table. In order for the keypad to be scanned correctly,

KEYPAD ENCODER MAPPING

This is what the output of the SBX2 keypad scanner would be for an ideal keypad wired straight to SBX2. The Key Fresh bit is not shown. Values are binary on the top (just the five lower bits, grouped YYY XX) and hexadecimal below that. Add 0x80 to each hex number to get its value with the Key Fresh bit asserted. Note that there are several unused, 'impossible' values -- anything with register bits 6 and 5 set. Also note that a binary Y value of 000 means no key has been pressed, valid values are 001-101. Valid X values are 00-11.

ROW/COL	Col Y1	Col Y2	Col Y3	Col Y4	Col Y5
Row X1	001 00 0x04	010 00 0x08	011 00 0xC	100 00 0x10	101 00 0x14
Row X2	001 01 0x05	010 01 0x09	011 01 0x0D	100 01 0x11	101 01 0x15
Row X3	001 10 0x06	010 10 0x0A	011 10 0x0E	100 10 0x12	101 10 0x16
Row X3	001 11 0x07	010 11 0x0B	011 11 0x0F	100 11 0x13	101 11 0x17

SYSTRONIX 4x4 KEYPAD #2702

This map shows the value of each key as if it were written on the keypad. As you press keys on the keypad and read their hexadecimal value from SBX2, this is what you would obtain. Note that rows 1 and 2 are swapped in the keypad. Note that this mapping assumes pin 1 of the keypad connector to be the rightmost pin as you face the keypad with the connector tail emerging from the top of the pad. On SBX2, this pin1 is aligned with P1.2, the "Y2" input. The key fresh bit is masked off, just the lower 5 bits of the keypad register are shown. Note that for this keypad, the upper left corner of the keypad is X2 and Y4.

ROW/COL	Col 4	Col 3	Col 2	Col 1
Row 1	0x11	0x0d	0x09	0x05
Row 2	0x10	0x0x	0x08	0x04
Row 3	0x12	0x0e	0x0a	0x06
Row 4	0x13	0x0f	0x0b	0x07

DESIRED KEYPAD LAYOUT

This possible keypad layout is similar to the numerical keypad on your PC. The left arrow is the backspace/rubout key. This is the standard keypad legend layout we provide with our 4x4 keypad 2702.

7	8	9	7
4	5	6	8
1	2	3	9
ESC	0	.	ENTR

You can remap keys in a Java or C switch-case construct, or with an array lookup. The easiest way to remap keys in 8051 assembly code is in a lookup table such as this.

```
key_get_val:
    gosub key_get_data
#ASM
    mov     DPTR,#__KEY_DAT     ; addr in DPTR
    movx    A,@DPTR             ; put value into acc
    add     A,#02H              ; adjust for jump
    movc    A,@A+PC             ; get lookup value into acc
    sjmp    OVER_KEY_TABLE      ; jump over data
;
; lookup data entries
; this translates Systronix keypad 2702 into the keys used
; in our 8051 based systems.  Note that pin 1 of the keypad connector is
; the rightmost pin as you face the keypad with the flex cable
; extending away from you, out of the top of the keypad.
;
; 1xH are the non-numeric keys
    DB      14H         ; 0 - del/backspace
    DB      15H         ; 1 - up arrow
    DB      13H         ; 2 - down arrow
    DB      12H         ; 3 - enter
    DB      06H         ; 4 -
    DB      09H         ; 5 -
    DB      03H         ; 6 -
    DB      11H         ; 7 - dec point
    DB      05H         ; 8 -
    DB      08H         ; 9 -
    DB      02H         ; A -
    DB      00H         ; B -
```

```
    DB      04H         ; C -
    DB      07H         ; D -
    DB      01H         ; E -
    DB      10H         ; F - escape
OVER_KEY_TABLE:
    movx    @DPTR,A     ; store value in KEY_DAT
#ASM_END
    return
```

KEYPAD LEGENDS

If your ordered a keypad and enclosure from us, they will probably be delivered assembled and tested, with the keypad legends shown on the attached mechanical drawing.

The keypad #2702 also available with no imprinted legends. In this case, it comes in two pieces: the actual membrane keypad and a protective overlay with a clear area for each key. You can easily create your own legends on a word processor or drawing program. Copy them onto 20 lb bond paper on a laser printer (heavier paper reduces tactile feedback). Or you can use a color photocopy or a color inkjet print for a more elaborate custom appearance. You can easily and quickly make low-volume custom keypads which look like they are an expensive full-custom design.

The keypad and its protective overlay are intended to be held together by the adhesive strips on the keypad. Position your legends (either in strips or a whole sheet) over the keypad. Peel off the adhesive release liner and position your legends. Be sure your legends are small enough to leave a border of adhesive around them to hold the overlay. Press the overlay carefully in place. This adhesive is very tenacious, so you only get one attempt at this!

Finally, the keypad back also has laminated adhesive for permanent mounting to an enclosure or panel. Be sure the mounting surface is clean - some isopropyl alcohol will remove any grease or oil from plastic and metal surfaces. Line up the keypad carefully since the adhesive is very permanent and cannot be easily repositioned.

DIGITAL I/O (24 BITS)

At the moment the best digital I/O documentation in addition to the address map above is in the javadocs in the SBX2 JAR file at http://www.systronix.com/expansion/sbx2/sbx2.htm

LCD INTERFACE

At the moment the best LCD documentation in addition to the address map above is in the javadocs in the SBX2 JAR file at http://www.systronix.com/expansion/sbx2/sbx2.htm

SBX2 Java API

See the documentation and JavaDocs online at http://www.systronix.com/expansion/sbx2/sbx2.htm. An extensive Java library for TINI and SaJe is under development for release 4Q 2001. Preliminary versions will be posted online in the SBX2 area of our website.