KB1
AT-PS/2 Keyboard Interface Chip

USER'S MANUAL

2012.06.01

Lucid Technologies
http://www.lucidtechnologies.info/
Email: info@lucidtechnologies.info

CONTENTS

1.0 Features

- Direct serial input from AT-PS/2 format keyboards
- Inexpensive single chip solution for keyboard input
- Pin-selectable interface modes
  - RS232 asynchronous serial interface
    - Pin-selectable baud rate: 9600 or 1200
    - Serial data format: 8 bit, No Parity, 1 Stop Bit
  - SPI synchronous serial interface
    - 100 kHz operation
    - Pin-selectable SPI modes: 0,1,2,3
  - I2C synchronous serial interface
    - 100 kHz operation
    - Pin-selectable seven bit address: 010XXXX
- Based on PIC16F1823 enhanced mid-range CPU
  - Operating Voltage: 5 VDC
  - Current Consumption: < 3 ma
  - No external oscillator or crystal required
  - 14 pin Dual Inline Package

2.0 Introduction

The KB1 was designed as a universal AT-PS/2 keyboard interface that can be used as a functional block in embedded projects. The AT-PS/2 keyboard interface involves complex timing and scan-code decoding. Incorporating the AT-PS/2 keyboard interface into a project's microcontroller can significantly increase the required program memory size and introduce timing problems. Instead of burdening your project's microcontroller with the AT-PS/2 keyboard interface, the KB1 provides an alternative solution for keyboard input. It can communicate with the Host via any of threes standard serial interface protocols supported by all microcontrollers.

The KB1 transmits the ASCII code assigned to each key, see appendix C. These ASCII codes include the standard (D7=0) ASCII and non-standard (extended) ASCII codes. Extended ASCII code bytes all have their most-significant bit set (D7=1). Examples of keys that produce extended ASCII output are PageUp, PageDown, function keys, and arrow keys. Not every key press will cause an ASCII code transmission to the Host. For example, the NumLock key is ignored, the KB1 always interprets the calculator keypad as numeric. Also, the CapsLock key affects a flag internal to the KB1 that changes the way future key presses are decoded, but does not itself cause an ASCII code transmission to the Host.

3.0 Pin Definitions

	MODE		
PIN	**DIAGNOSTIC & UART**	**SPI**	**I2C**
1 Vdd	+5V	+5V	+5V
2 RA5	SEL1	SEL1	SEL1
3 RA4	SEL0	SEL0	SEL0
4 MCLR	MCLR	MCLR	MCLR
5 RC5	RXD	CKE	ADR4
6 RC4	TXD	CKP	ADR3
7 RC3	nc	SS	ADR2
8 RC2	nc	SDO	ADR1
9 RC1	nc	SDI	SDA
10 RC0	Baud	SCK	SCL
11 RA2	nc	IRQ	IRQ
12 RA1	KBDAT	KBDAT	KBDAT
13 RA0	KBCLK	KBCLK	KBCLK
14 Vss	Ground	Ground	Ground

ADR4-ADR1 (Pins 5-8)
These pins are active in I2C mode. They are read at power up or when the KB1 is reset. Their state determines the I2C address of the KB1.

BAUD (Pin 10)
This pin is active in the UART and DIAGNOSTIC modes. This pin determines the BAUD rate at power up or reset; pullup to +5V is 9600, grounded is 1200.

CKE (Pin 5)
This pin is active in SPI mode. CKE selects the SPI transmit clock edge.

CKP (Pin 6)
This pin is active in SPI mode. CKP selects the SPI clock idle state polarity.

IRQ (Pin 11)
This pin is active in SPI and I2C modes. This is the Interrupt Request (IRQ) output to the Host.

KBDAT (Pin 12)
This pin is active in all modes. It is the bidirectional data line between the KB1 and the keyboard.

KBCLK (Pin 13)
This pin is active in all modes. It is the bidirectional clock line between the KB1 and the keyboard.
MCLR (Pin 4)
This pin is active in all modes. It is the Master Clear input of the KB1 microcontroller. Grounding this pin will reset the KB1 and any keyboard connected to it.

RXD (Pin 5)
This pin is active in the UART and DIAGNOSTIC modes. It is the UART receive data pin. This pin receives serial data from the Host at 5 volt TTL levels.

SCK (Pin 10)
This pin is active in SPI mode. This is the Serial Clock pin.

SCL (Pin 10)
This pin is active in I2C mode. It is the Serial Clock (SCL) input from the Host.

SDA (Pin 9)
This pin is active in I2C mode. It is the bidirectional Serial Data pin for communication with the Host.

SDI (Pin 9)
This pin is active in SPI mode. This is the Serial Data In pin.

SDO (Pin 8)
This pin is active in SPI mode. This is the Serial Data Out pin.

SEL1 (Pin 2), SEL0 (Pin 3)
These pins are active in all modes. They determining the operating mode of the KB1. They are read at power up or when the KB1 is reset.

SS (Pin 7)
This pin is active in SPI mode. This is the Slave Select pin.

TXD (Pin 6)
This pin is active in the UART and DIAGNOSTIC modes. It is the UART transmit data pin. This pin sends serial data to the Host at 5 volt TTL levels.

4.0 Operating Modes

The mode selection determines the serial protocol used by the KB1 to communicate with the Host. The four possible modes are shown in this Table.

SEL1	SEL0	MODE
0	0	Diagnostic
0	1	SPI
1	0	I2C
1	1	UART

4.1 UART Mode

UARTs allow rapid full duplex communication, usually in peer-to-peer topologies rather than master-to-slave topologies.

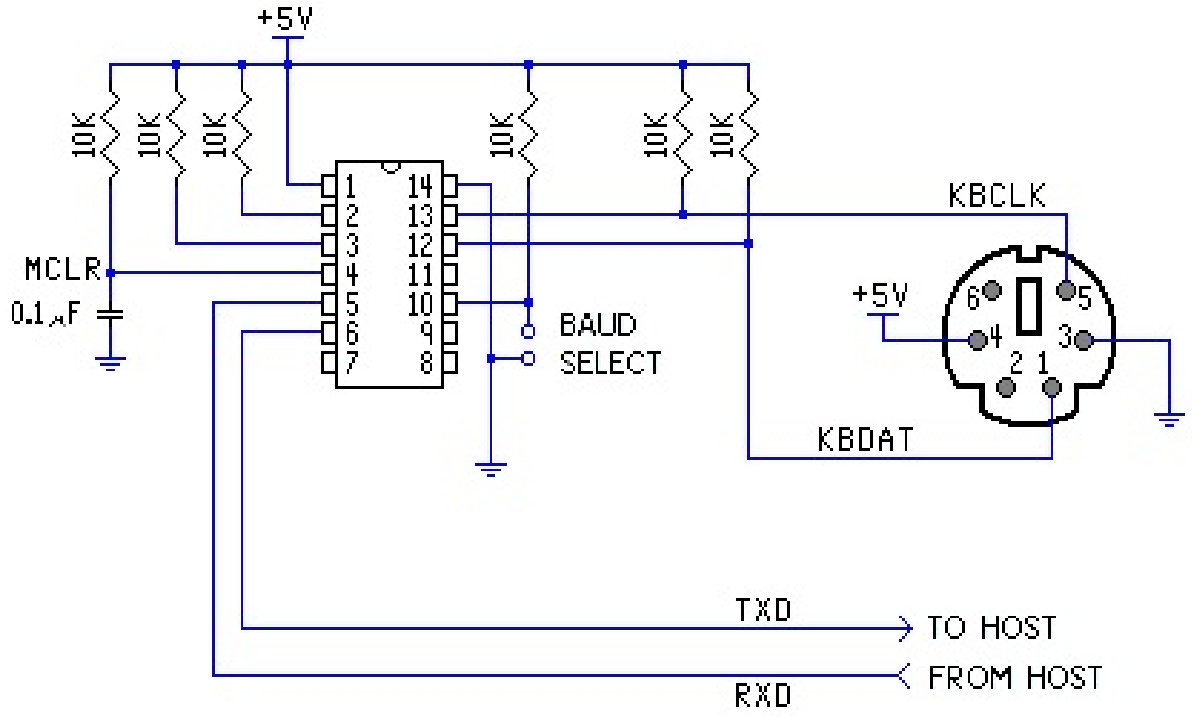

Figure 4.1 KB1 configuration for UART and Diagnostic modes.

The KB1 UART can operate at 1200 or 9600 baud; grounding the BAUD pin selects 1200, pulling it up to a logic high selects 9600. The TXD and RXD signals operate at 5 volt logic levels.

The KB1 transmits ASCII codes whenever a valid key press is decoded, see appendix C. The Host's UART should be programmed to produce an interrupt upon receipt of a byte from the KB1.

The KB1 will respond to two commands from the Host, all other received bytes will be ignored. The Reset command (0xFF) will reset the KB1 and attached keyboard. The Resend command (0xFE) will cause the KB1 to retransmit the last ASCII code byte sent to the Host.

4.2 SPI Mode

The Serial Peripheral Interface (SPI) Bus is a synchronous serial data link standard, named by Motorola, that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Only the master device can control the clock line, SCK. Multiple slave devices are allowed with individual slave select lines.

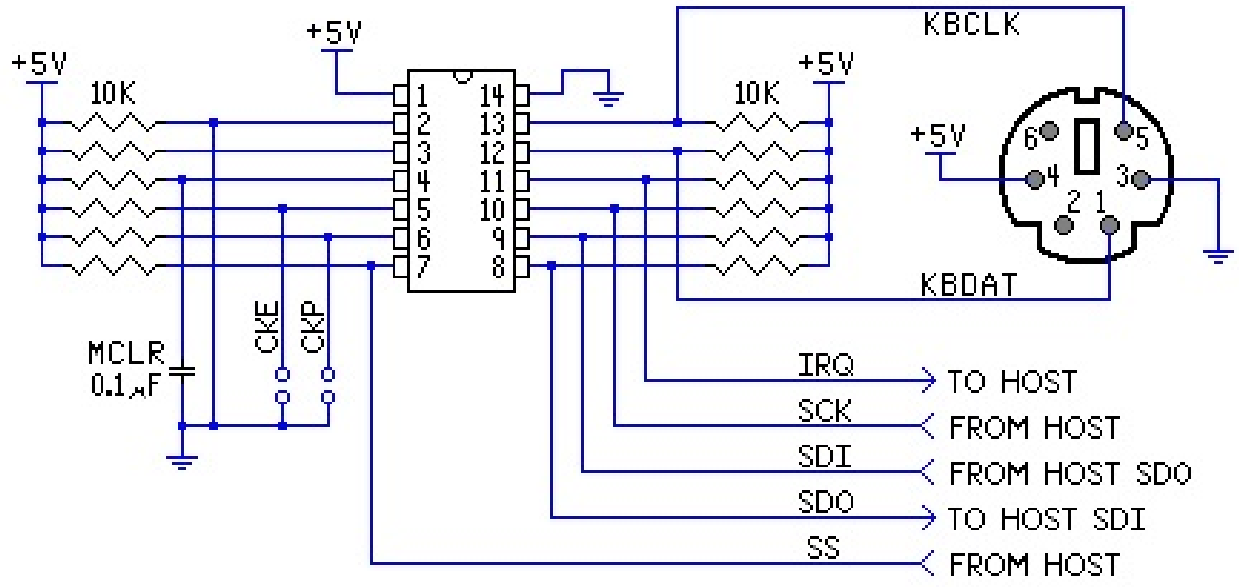

Figure 4.2a KB1 configuration for SPI mode.

The transmit clock edge and the idle state of the SPI clock (SCK) are set by CKE and CKP respectively. For full details see the Microchip data sheet for the PIC16F1823. These settings produce four possible SPI modes.

SPI Mode	CKE	CKP	SCK Idle	Shift edge	Read edge
0	1	0	0	Falling	Rising
1	0	0	0	Rising	Falling
2	1	1	1	Rising	Falling
3	0	1	1	Falling	Rising

As data is being shifted out, new data is also being shifted in. Data is always “exchanged” between devices. No device can just be a “transmitter” or just a “receiver” in SPI.

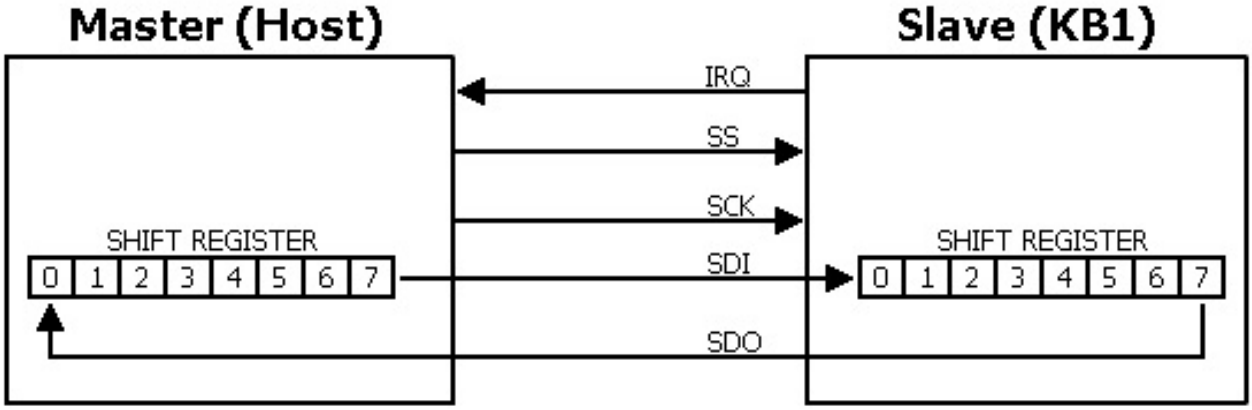

Figure 4.2B Data exchange nature of SPI. As data from Master is shifted into Slave, data from Slave is shifted into Master.

Because SPI data exchange is initiated by the Master (Host), and the KB1 is an SPI Slave, the KB1 can’t simply send ASCII codes to the Host. Therefore the KB1 uses the Interrupt Request (IRQ) output to signal the Host that data is ready. All SPI data exchanges with the KB1 are one byte in length. The KB1 keeps its SPI shift register loaded with the oldest unread ASCII code from its keyboard buffer. Upon detecting IRQ is low, the Host should load its SPI shift register with 0x11 and initiate a data exchange with the KB1. When the data exchange is completed the Host’s SPI shift register contains the ASCII code from the KB1. The KB1 will set IRQ high to acknowledge it has finished processing the byte exchanged with the Host. The Host should then set Slave Select (SS) high to terminate the data exchange.

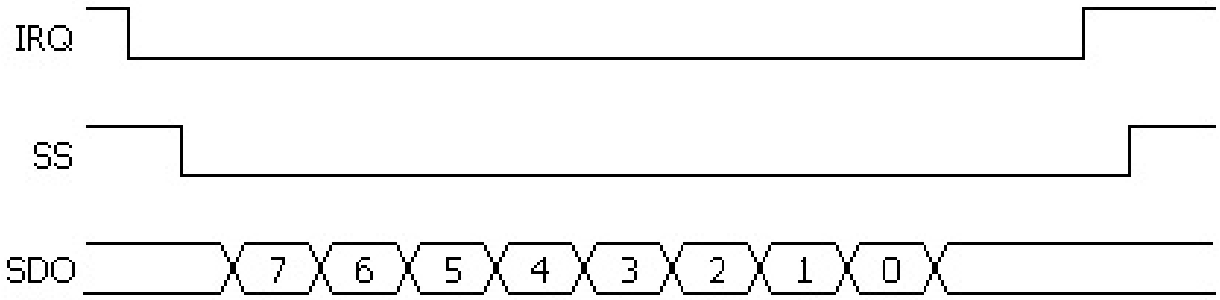

Figure 4.2C Data exchange timing between Host and KB1.

There are three commands that can be sent to the KB1 in SPI mode.

The Read command (0x11) should only be sent when IRQ is low. Because most commands will be the Read command, the KB1 keeps its SPI shift register loaded with the oldest unread ASCII code

from its keyboard buffer. If there are no unread ASCII codes the shift register contains 0x00. If any other command is received, valid or invalid, the shift register will be reloaded with the oldest unread ASCII code from its keyboard buffer after the other command is processed.

The Resend command (0xFE) may be sent regardless of the state of IRQ. It causes the KB1 to move its index pointer into the keyboard buffer so that the last read byte becomes the oldest unread byte. To actually read this ASCII code byte the Host must then send a Read command.

The Reset command (0xFF) may be sent regardless of the state of IRQ. The KB1, and attached keyboard will be reset.

4.3 I2C Mode

The Inter-Integrated Circuit (or I2C, I²C, IIC) bus is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a Host electronic device. Sometimes

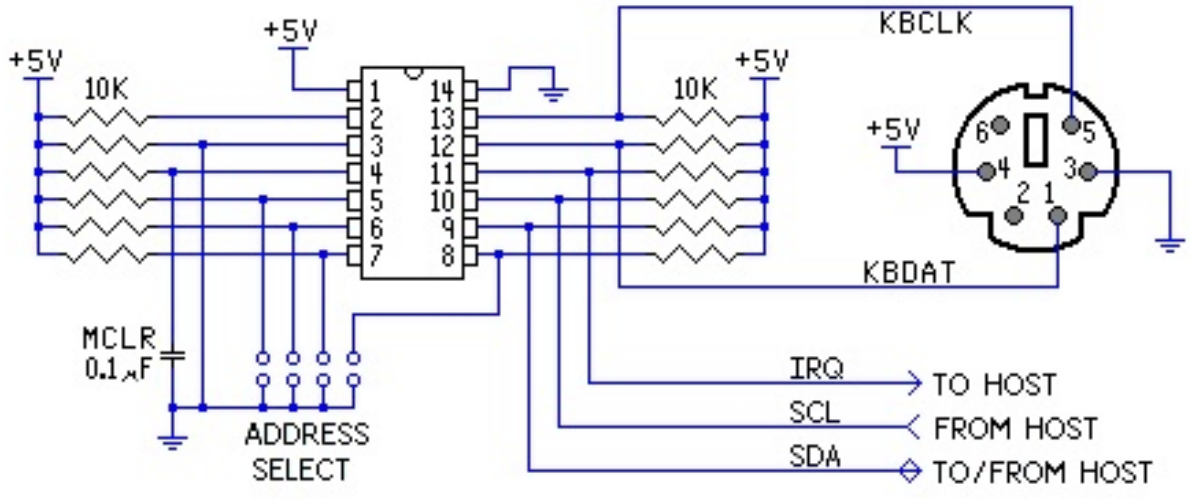

Figure 4.3 KB1 configuration for I2C mode.

I2C is called a “two-wire” bus.

Because I2C data exchange is initiated by the Master (Host), and the KB1 is an I2C Slave, the KB1 can’t simply send ASCII codes to the Host. Therefore the KB1 uses the Interrupt Request (IRQ) output to signal the Host that data is ready. When IRQ is low the KB1 has unread data for the Host. All I2C communications with the KB1 are two bytes in length. The first byte sent by the Host is always the 7 bit address of the KB1 and a Read/Write bit. The KB1 can be assigned to any of sixteen addresses using the four address select pins, 010XXXX.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	Pin 5	Pin 6	Pin 7	Pin 8	R/W

If the R/W bit is 1 (Read) the KB1 will send the second byte to the Host. The byte sent will be the oldest unread ASCII code from its keyboard buffer, or 0x00 if the buffer is empty. The Host should only send a Read request if IRQ is low.

If the R/W bit is 0 (Write) the Host will send a command as the second byte. Valid command bytes are 0xFE and 0xFF, other values will be ignored. Commands may be sent without regard to the state of IRQ. The Reset command (0xFF) will reset the KB1 and attached keyboard. The Resend command (0xFE) causes the KB1 to move its index pointer into the keyboard buffer so that the last read byte becomes the oldest unread byte. To actually read this ASCII code byte the Host must then send a Read request.

4.4 Diagnostic Mode

The hardware configuration for the Diagnostic mode is almost the same as the UART mode, see Figure 4.1. The only difference is that the mode select pins (SEL1 and SEL0) are both grounded. To use the Diagnostic mode with a computer running a terminal program (like HyperTerm, etc.) the RXD and TXD signals must be converted to RS232 levels. This can easily be done with a MAX232 chip. The KB1 will report all data sent to and from the KB1 in the following format.

XX = Hexadecimal value of byte sent to the KB1 from the keyboard
<XX> = Hexadecimal value of byte sent to the keyboard from the KB1
{XX} = Hexadecimal value of byte sent to the Host from the KB1
[XX] = Hexadecimal value of byte sent to KB1 from Host

Here is what the diagnostic mode will show when the KB1 tells the keyboard to turn on the Num Lock LED.

<ED> FA<02> FA
ED = Set status LED command, sent to keyboard
FA = Acknowledge byte from keyboard
02 = Value for Num Lock LED, sent to keyboard
FA = Acknowledge byte from keyboard

Here is what the diagnostic mode will show when the space bar is pressed and released.

29{20} F0 29
29 = Make scan code for space bar from keyboard
20 = ASCII code for space sent to Host
F0 29 = Break scan code for space bar from keyboard

5.0 Keyboards

Although most new PC keyboards use the Universal Serial Bus (USB) protocol, the software overhead to implement a USB interface is prohibitive for small projects. PS/2 keyboards are still widely

available and a much better choice for quick prototyping and simple user interface. PS/2 keyboards use a 6-pin mini-DIN connector, see Appendix E. The KB1 is compatible with both PS/2 and AT keyboards. AT keyboards use a 5-pin DIN connector, but beware, older XT keyboards used the same 5-pin DIN connector but they are not compatible with the KB1.

APPENDIX A

ASCII CHARACTER CODES

The first 32 characters in the ASCII (American Standard Code for Information Interchange) table are unprintable control codes and are used to control peripherals such as printers.

ASCII control characters (character codes 0x00-0x1F)

DEC	HEX	Symbol	Description
0	0	NUL	Null char
1	1	SOH	Start of Header
2	2	STX	Start of Text
3	3	ETX	End of Text
4	4	EOT	End of Transmission
5	5	ENQ	Enquiry
6	6	ACK	Acknowledgment
7	7	BEL	Bell
8	8	BS	Backspace
9	9	HT	Horizontal Tab
10	0A	LF	Line Feed
11	0B	VT	Vertical Tab
12	0C	FF	Form Feed
13	0D	CR	Carriage Return
14	0E	SO	Shift Out
15	0F	SI	Shift In
16	10	DLE	Data Line Escape
17	11	DC1	Device Control 1 (often XON)
18	12	DC2	Device Control 2
19	13	DC3	Device Control 3 (often XOFF)
20	14	DC4	Device Control 4
21	15	NAK	Negative Acknowledgment
22	16	SYN	Synchronous Idle
23	17	ETB	End of Transmit Block
24	18	CAN	Cancel
25	19	EM	End of Medium
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	File Separator
29	1D	GS	Group Separator
30	1E	RS	Record Separator
31	1F	US	Unit Separator

Codes 32-127 are called printable characters. They represent letters, digits, punctuation marks, and a few miscellaneous symbols. You will find almost every character on your keyboard.

ASCII printable characters (character code 0x20-0x7F)

DEC	HEX	Symbol	Description
32	20		Space
33	21	!	Exclamation mark
34	22	"	Double quotes
35	23	#	Number
36	24	$	Dollar
37	25	%	Percent
38	26	&	Ampersand
39	27	`	Single quote
40	28	(	Open parenthesis
41	29	)	Close parenthesis
42	2A	*	Asterisk
43	2B	+	Plus
44	2C	,	Comma
45	2D	-	Hyphen
46	2E	.	Period, dot or full stop
47	2F	/	Slash or divide
48	30	0	Zero
49	31	1	One
50	32	2	Two
51	33	3	Three
52	34	4	Four
53	35	5	Five
54	36	6	Six
55	37	7	Seven
56	38	8	Eight
57	39	9	Nine
58	3A	:	Colon
59	3B	;	Semicolon
60	3C	<	Less than
61	3D	=	Equals
62	3E	>	Greater than
63	3F	?	Question mark
64	40	@	At symbol
65	41	A	Uppercase A
66	42	B	Uppercase B
67	43	C	Uppercase C
68	44	D	Uppercase D
69	45	E	Uppercase E
70	46	F	Uppercase F
71	47	G	Uppercase G
72	48	H	Uppercase H
73	49	I	Uppercase I
74	4A	J	Uppercase J
75	4B	K	Uppercase K
76	4C	L	Uppercase L
77	4D	M	Uppercase M

DEC	HEX	Symbol	Description
78	4E	N	Uppercase N
79	4F	O	Uppercase O
80	50	P	Uppercase P
81	51	Q	Uppercase Q
82	52	R	Uppercase R
83	53	S	Uppercase S
84	54	T	Uppercase T
85	55	U	Uppercase U
86	56	V	Uppercase V
87	57	W	Uppercase W
88	58	X	Uppercase X
89	59	Y	Uppercase Y
90	5A	Z	Uppercase Z
91	5B	[	Opening bracket
92	5C	\	Backslash
93	5D	]	Closing bracket
94	5E	^	Caret - circumflex
95	5F	_	Underscore
96	60	`	Grave accent
97	61	a	Lowercase a
98	62	b	Lowercase b
99	63	c	Lowercase c
100	64	d	Lowercase d
101	65	e	Lowercase e
102	66	f	Lowercase f
103	67	g	Lowercase g
104	68	h	Lowercase h
105	69	I	Lowercase I
106	6A	j	Lowercase j
107	6B	k	Lowercase k
108	6C	l	Lowercase l
109	6D	m	Lowercase m
110	6E	n	Lowercase n
111	6F	o	Lowercase o
112	70	p	Lowercase p
113	71	q	Lowercase q
114	72	r	Lowercase r
115	73	s	Lowercase s
116	74	t	Lowercase t
117	75	u	Lowercase u
118	76	v	Lowercase v
119	77	w	Lowercase w
120	78	x	Lowercase x
121	79	y	Lowercase y
122	7A	z	Lowercase z
123	7B	{	Opening brace
124	7C	|	Vertical bar
125	7D	}	Closing brace
126	7E	~	Equivalency sign - tilde
127	7F		Delete

APPENDIX B

KEYBOARD SCAN CODES

The scan codes shown here are Set 2 for IBM/Compatible keyboards - Also known as "AT" or "PS/2" keyboards. There are multiple sets of scan codes in use but Scan Code Set 2 is the default scan code set for all modern keyboards. The scan codes shown below are in hex. There are two scan codes for each key; the “make” code, sent when the key is depressed, is the one shown in the diagrams below. The “break” code, sent when the key is released, adds an 0xF0 byte. Thus the F1 key has a make code of 0x05 and a break code of 0xF0, 0x05. Extended scan codes (those with an 0xE0 prefix) add the 0xF0 after the 0xE0. Thus the Up Arrow key has a make code of 0xE0, 0x75 and a break code of 0xE0, 0xF0, 0x75.

The important thing to remember about scan codes is that they only identify which key is depressed not the character associated with the key! The computer operating system, or in this case the KB1 chip, must map each key to an appropriate symbol or action. For more details on scan codes see the references in Appendix F.

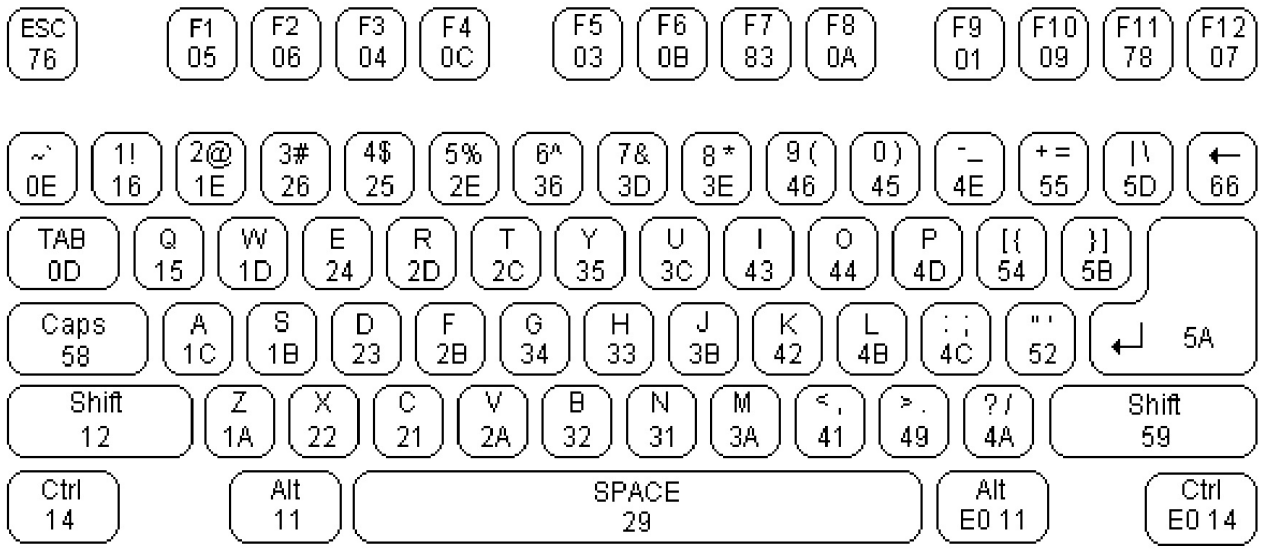

Figure B1 Scan codes for the left side of a typical keyboard.

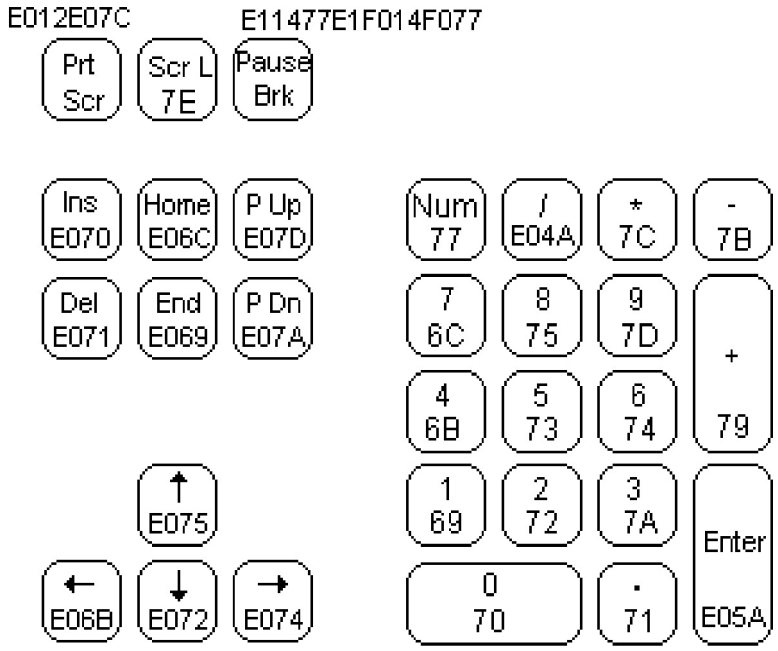

Figure B2 Scan codes for the right side of a typical keyboard.

APPENDIX C

KB1's KEY TO ASCII CODE TRANSLATION

(All values are in Hexadecimal)

KEY	Normal Output	Shifted Output	KEY	Normal Output	Shifted Output
Esc	1B	1B	F1	80	80
F2	81	81	F3	82	82
F4	83	83	F5	84	84
F6	85	85	F7	86	86
F8	87	87	F9	88	88
F10	89	89	F11	8A	8A
F12	8B	8B			
` ~	60	7E	1 !	31	21
2 @	32	40	3 #	33	23
4 $	34	24	5 %	35	25
6 ^	36	5E	7 &	37	26
8 *	38	2A	9 (	39	28
0)	30	29	- _	2D	5F
= +	3D	2B	Backspace	08	08
Tab	09	09	q Q	71	51
w W	77	57	e E	65	45
r R	72	52	t T	74	54
y Y	79	59	u U	75	55
I I	69	49	o O	6F	4F
p P	70	50	[{	5B	7B
] }	5D	7D	\ |	5C	7C
a A	61	41	s S	73	53
d D	64	44	f F	66	46
g G	67	47	h H	68	48
j J	6A	4A	k K	6B	4B
l L	6C	4C	; :	3B	3A

KEY	Normal Output	Shifted Output	KEY	Normal Output	Shifted Output
` ~	27	22	Enter	0D	0D
z Z	7A	5A	x X	78	58
c C	63	43	v V	76	56
b B	62	42	n N	6E	4E
m M	6D	4D	, <	2C	3C
. >	2E	3E	/ ?	2F	3F
Left Windows	99	99	Left Alt (1)	96	96
Space	20	20	Right Alt (1)	96	96
Right Windows	9A	9A	Menu	9B	9B
Print Screen	9D	9D			
Insert	8C	8C	Home	94	94
Page Up	8D	8D	Delete	7F	7F
End	93	93	Page Down	8E	8E
Up Arrow ↑	91	91	Left Arrow ←	8F	8F
Down Arrow ↓	92	92	Right Arrow →	90	90
Keypad /	2F	2F	Keypad *	2A	2A
Keypad -	2D	2D	Keypad 7	37	37
Keypad 8	38	38	Keypad 9	39	39
Keypad 4	34	34	Keypad 5	35	35
Keypad 6	36	36	Keypad +	2B	2B
Keypad 1	31	31	Keypad 2	32	32
Keypad 3	33	33	Keypad 0	30	30
Keypad .	2E	2E	Keypad Enter	0D	0D

(1) A 0x96 is sent when either Alt key is depressed, a 0x97 is sent when both Alt keys are released.

The numeric keypad portion of the keyboard is always in the numeric mode. The Num Lock key has no effect.

The Scroll Lock and Pause/Break keys are ignored by the KB1-IC.

APPENDIX D

CONTROL CHARACTERS

The KB1 will generate the thirty-two ASCII control codes when the appropriate character is depressed while holding down either control (Ctrl) key. For example, typing a capital A while holding down either Ctrl key will cause the KB1 to send 0x01.

Character	Output (hex)	Symbol	Description
@	00	NUL	Null character
A	01	SOH	Start of Header
B	02	STX	Start of Text
C	03	ETX	End of Text
D	04	EOT	End of Transmission
E	05	ENQ	Enquiry
F	06	ACK	Acknowledgment
G	07	BEL	Bell
H	08	BS	Backspace
I	09	HT	Horizontal Tab
J	0A	LF	Line feed
K	0B	VT	Vertical Tab
L	0C	FF	Form feed
M	0D	CR	Carriage return
N	0E	SO	Shift Out /
O	0F	SI	Shift In
P	10	DLE	Data Link Escape
Q	11	DC1	Device Control 1 (often XON)
R	12	DC2	Device Control 2
S	13	DC3	Device Control 3 (often XOFF)
T	14	DC4	Device Control 4
U	15	NAK	Negative Acknowledgment
V	16	SYN	Synchronous idle
W	17	ETB	End of Transmission Block
X	18	CAN	Cancel
Y	19	EM	End of Medium
Z	1A	SUB	Substitute
[	1B	ESC	Escape
\	1C	FS	File Separator
]	1D	GS	Group Separator
^	1E	RS	Record Separator
_	1F	US	Unit Separator

APPENDIX E

PS/2 CONNECTOR

PS/2 Keyboards have a male 6-pin mini-DIN connector. Therefore the KB1-IC usually connects to a female 6-pin mini-DIN connector like that shown below. Note that various manufacturers may number the pins in different orders. Of course the position of the signals on the connector remains the same no matter how the pins are numbered.

Figure E1 Female 6-pin mini-DIN connector viewed from the front.

Table E1 Signals for the female 6-pin mini-DIN connector shown above.

Pin 1	Data
Pin 2	Not Connected
Pin 3	Ground
Pin 4	+5V DC at 275 mA
Pin 5	Clock
Pin 6	Not Connected

APPENDIX F

REFERENCES

http://www.electronic-engineering.ch/microchip/index.html
AT Keyboard Interface V1.04 for Microchip 16F84 Microcontroller. Peter Luethi

http://www.Computer-Engineering.org
The PS/2 Keyboard Protocol. Adam Chapweske.

http://www.lima.com.tr/Support/Lima/PS/2Keyboard_EN.PDF
Using PS/2 Keyboards in Embedded Systems. Cem Celik

http://retired.beyondlogic.org/keyboard/keybrd.htm
Interfacing the AT keyboard. Craig Peacock

Motorola Semiconductor Application Note, AN1723
Interfacing MC68HC05 Microcontrollers to the IBM AT Keyboard Interface.

THE I2C-BUS SPECIFICATION, VERSION 2.1, JANUARY 2000
Philips Semiconductors, http://www.semiconductors.philips.com/i2c

Microchip PIC12F/LF1822/PIC16F/LF1823 Data Sheet, DS41413B

SPI Protocol and Bus Configuration, Intersil Application Note AN1340.0, August 20, 2007

Appendix G

Sample Host code - SPI mode

```
; Example code for a host to communicate with a KB1.
; This example assumes:
;   host microcontroller is also a PIC16F1823,
;   the KB1 pins are set for:
;     SPI communications (SEL1=0, SEL0=1),
;     SPI mode zero (CKE=1, CKP=0).
;
; PIC16F1823 PIN USAGE - HOST
; -- Pin --       -- SPI -
;  1   Vdd  +5V
;  2   RA5  -
;  3   RA4  -
;  4   RA3  MCLR   In
;  5   RC5  -
;  6   RC4  -
;  7   RC3  SS    Out
;  8   RC2  SDO   Out
;  9   RC1  SDI    In
; 10   RC0  SCK   Out
; 11   RA2  IRQ    In
; 12   RA1  -
; 13   RA0  -
; 14   Vss  Gnd
;
;--------------------------------------------------------------------
-
SPI_init                    ; Mode 1, SPI mode
      banksel  LATC           ; select bank 2
      movlw    B'00011101'      ; 1 is idle state of serial lines
      movwf    LATC             ; init output LATCH C
      banksel  TRISA          ; select bank 1
      movlw    B'11111111'      ; RA2 = input = IRQ1 from KB1
      movwf    TRISA            ; PORTA direction
      movlw    B'11100010'      ; direction for SPI port lines
      movwf    TRISC            ; PORTC direction
      banksel  SSP1ADD        ; select bank 4
      movlw    0x27             ; set SPI clock rate (based on Fo=16MHz)
      movwf    SSP1ADD          ;   to 100 kHz
      bcf      SSP1CON1,5       ; SSP1EN=0
      movlw    B'00001010'      ; CKP=0
      movwf    SSP1CON1         ;   Master mode
      movlw    B'01000000'      ; SMP=0, CKE=1
      movwf    SSP1STAT         ; SPI mode 0
      bsf      SSP1CON1,5       ; SSP1EN=1, SPI enabled
      movf     SSP1BUF          ; read buffer to clear BF bit

;--------------------------------------------------------------------
-
; Subroutine to read an ASCII code byte from the KB1.
; This routine should be called only after the KB1 signals data is
```

```
; available by lowering IRQ. The host program can link IRQ to an
; interrupt, or poll it to determine when data is available.
SPI_c11
; Clear Slave Select to KB1
      banksel  LATC           ; select bank 2
      bcf      LATC,3           ; 0 --> RC3 = SS output
; Send command byte to KB1
      banksel  SSP1BUF        ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      movf     SSP1BUF,w        ; read SPI buffer, clears BF
      movlw    0x11             ; Device Code 1 (DC1)
      movwf    SSP1BUF          ; send data request command
; Wait for command byte to shift-out, data from KB1 to shift-in
SPI_12
      btfss    SSP1STAT,BF      ; wait until receive buffer full
      bra      SPI_12           ;  flag (BF) is set
      movf     SSP1BUF,w        ; read SPI buffer, clears BF
      banksel  PORTA           ; select bank 0
      movwf    rxbuf1           ; store byte from KB1
; Wait for IRQ from KB1 to go Hi
SPI_14
      btfss    PORTA,2          ; test interrupt request input
      goto     SPI_14           ; loop until IRQ = 1
; Set Slave Select to KB1
      banksel  LATC           ; select bank 2
      bsf      LATC,3           ; 1 --> RC3 = SS output
; Communication with KB1 is done. Data from KB1 in rxbuf1
      return

;--------------------------------------------------------------------------
-
; Subroutine to send 0xFE (Resend) command byte to KB1.
SPI_cFE
; Clear Slave Select to KB1
      banksel  LATC           ; select bank 2
      bcf      LATC,3           ; 0 --> RC3 = SS output
; Send command byte to KB1
      banksel  SSP1BUF        ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      movf     SSP1BUF,w        ; read SPI buffer, clears BF
      movlw    0xFE             ;
      movwf    SSP1BUF          ; send data request command
      goto     SPI_12

;--------------------------------------------------------------------------
-
; Subroutine to send 0xFF (Reset) command byte to KB1.
SPI_cFF
; Clear Slave Select to KB1
      banksel  LATC           ; select bank 2
      bcf      LATC,3           ; 0 --> RC3 = SS output
; Send command byte to KB1
      banksel  SSP1BUF        ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      movf     SSP1BUF,w        ; read SPI buffer, clears BF
```

```
        movlw   0xFF            ;
        movwf   SSP1BUF         ; send data request command
        goto    SPI_12
```

Appendix H

Sample Host code - I2C mode

```
; Example code for a host to communicate with a KB1.
; This example assumes:
;   host microcontroller is also a PIC16F1823,
;   the KB1 pins are set for:
;     I2C communications (SEL1=1, SEL0=0),
;     the lower four address bits (ADR4-1) are set to 1010.
; This makes the 7-bit address of the KB1 0101010X (X is the R/W bit).
;
; PIC16F1823 PIN USAGE - HOST
; -- Pin --       -- I2C --
;  1   Vdd  +5V
;  2   RA5  -
;  3   RA4  -
;  4   RA3  MCLR   In
;  5   RC5  -
;  6   RC4  -
;  7   RC3  -
;  8   RC2  -
;  9   RC1  SDA    In
; 10   RC0  SCL    In
; 11   RA2  IRQ    In
; 12   RA1  -
; 13   RA0  -
; 14   Vss  Gnd
;
;--------------------------------------------------------------------------
-
I2C_init                    ; Mode 2, I2C mode
; SCL and SDA should be set to inputs by setting the appropriate TRIS
bits.
      banksel  SSP1ADD       ; select bank 4
      movlw    0x27            ; set SPI clock rate (based on Fo=16MHz)
      movwf    SSP1ADD         ;   to 100 kHz
      movlw    B'10000000'     ; standard slew rate
      movwf    SSP1STAT        ; SMP=1
      movlw    B'00111000'     ; I2C master mode, 7-bit address
      movwf    SSP1CON1        ; CKP=1
      clrf     SSP1CON2        ;
      clrf     SSP1CON3        ;

;--------------------------------------------------------------------------
-
; Subroutine to generate I2C start condition.
I2C_start
      banksel  SSP1CON1       ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      bcf      SSP1CON1,SSPOV   ; clear overflow bit
      bsf      SSP1CON2,SEN     ; start condition enabled
      nop                       ;
      btfsc    SSP1CON2,SEN     ; loop until SEN bit is cleared
```

```
      bra      $-1              ;  by end of start condition
      return

;--------------------------------------------------------------------
-
; Subroutine to generate I2C stop condition.
I2C_stop
      banksel  SSP1CON1       ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      bcf      SSP1CON1,SSPOV   ; clear overflow bit
      banksel  PIR1            ; select bank 0
      bcf      PIR1,SSP1IF      ; clear I2C interrupt flag
      banksel  SSP1CON2       ; select bank 4
      bsf      SSP1CON2,PEN     ; stop condition enabled
      nop                       ;
      btfsc    SSP1CON2,PEN     ; loop until PEN bit is cleared
      bra      $-1              ;  by end of stop condition
      banksel  PIR1            ; select bank 0
      bcf      PIR1,SSP1IF      ; clear I2C interrupt flag
      return

;--------------------------------------------------------------------
-
; Subroutine to transmit the byte in W via I2C interface.
I2C_tx
      banksel  SSP1CON1       ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      bcf      SSP1CON1,SSPOV   ; clear overflow bit
      movwf    SSP1BUF          ; start transmission
      banksel  PIR1            ; select bank 0
      bcf      PIR1,SSP1IF      ; clear I2C interrupt flag
      nop
      btfss    PIR1,SSP1IF      ; wait for data byte to be sent
      bra      $-1              ;  and ACK bit to be received
      return

;--------------------------------------------------------------------
-
; Subroutine to read an ASCII code byte from the KB1.
; This routine should be called only after the KB1 signals data is
; available by lowering IRQ. The host program can link IRQ to an
; interrupt, or poll it to determine when data is available.
I2C_RD
      call     I2C_start        ; generate a start condition
      movlw    0x55             ; address/read byte
      call     I2C_tx           ; send byte in W
      banksel  SSP1CON2       ; select bank 4
      btfsc    SSP1CON2,6       ; test ACKSTAT bit
      bra      I2C_RD9          ; branch if no slave acknowledges
      banksel  PIR1            ; select bank 0
      bcf      PIR1,SSP1IF      ; clear I2C interrupt flag
      banksel  SSP1CON2       ; select bank 4
      bcf      SSP1CON1,WCOL    ; clear write collision bit
      bcf      SSP1CON1,SSPOV   ; clear overflow bit
      bsf      SSP1CON2,RCEN    ; set receive enable bit
```

```
      banksel  PIR1           ; select bank 0
      btfss    PIR1,SSP1IF     ; wait for data from slave
      bra      $-1             ;  to be received
      banksel  SSP1BUF        ; select bank 4
      movf     SSP1BUF,w       ; read I2C buffer, clears BF
      banksel  PIR1           ; select bank 0
      movwf    rxbuf1          ; store byte from KB1
      bcf      PIR1,SSP1IF     ; clear I2C interrupt flag
      banksel  SSP1CON2        ; select bank 4
      bsf      SSP1CON2,ACKDT  ; data transfer complete,
      bsf      SSP1CON2,ACKEN  ;  send not acknowledge bit
      banksel  PIR1           ; select bank 0
      btfss    PIR1,SSP1IF     ; wait for data from slave
      bra      $-1             ;  to be received
I2C_RD9
      call     I2C_stop        ; end I2C communication
; Communication with KB1 is done
      return

;-------------------------------------------------------------------
-
; Subroutine to send 0xFE (Resend) command byte to KB1.
I2C_FE
      call     I2C_start       ; generate a start condition
      movlw    0x54            ; address/write byte
      call     I2C_tx          ; send byte in W
      banksel  SSP1CON2      ; select bank 4
      btfsc    SSP1CON2,6      ; test ACKSTAT bit
      bra      I2C_FE9         ; branch if no slave acknowledges
      movlw    0xFE            ; repeat data command
      call     I2C_tx          ; send byte in W
I2C_FE9
      call     I2C_stop        ; end I2C communication
; Communication with KB1 is done
      return

;-------------------------------------------------------------------
-
; Subroutine to send 0xFF (Reset) command byte to KB1.
I2C_FF
      call     I2C_start       ; generate a start condition
      movlw    0x54            ; address/write byte
      call     I2C_tx          ; send byte in W
      banksel  SSP1CON2      ; select bank 4
      btfsc    SSP1CON2,6      ; test ACKSTAT bit
      bra      I2C_FF9         ; branch if no slave acknowledges
      movlw    0xFF            ; reset command
      call     I2C_tx          ; send byte in W
I2C_FF9
      call     I2C_stop        ; end I2C communication
; Communication with KB1 is done
      return
```