BL233

RS232 – I2C/SPI/1WIRE ADAPTOR & CONTROLLER

Features

- Universal Serial Bus Interface
- Four separate I2C busses
- SPI
- Dallas 1 Wire
- SCL Stretch to work with software & microprocessor slaves
- Interrupt input
- adjustable I2C speed
- ASCII protocol for ease of use
- Spare I/O pins for user control functions
- Hi Speed RS232: 1200bd - 921k
- Fully buffered RS232
- Compact 18 pin package
- Low power
- Low EMC
- Sleep
- Watchdog function detects and actions host computer failure
- 3-5V
- Cross-Platform

Applications

- Evaluation Boards for I2C chips
- Rapid Prototyping & PnP design
- PC based instrumentation and control
- ATE for I2C based equipment
- Isolated I/O
- Cheap and Easy Data logging
- PC and Network Watchdogs
- Handheld computer / calculator Analog/Digital I/O
- RF Systems with EMI constraints
- Education
- RS232 Parallel port chip
- Evaluation of I2C and SPI chips

Programmable

EEProm stores commands and settings

- Power On Reset actions
- Interrupt actions
- Watchdog timeout actions
- Macro's
- Autonomous actions
- Baud Rate (1200bd-921k)
- Settings

1 Description

BL233 is a simple and effective way to use I2C devices with any computer and OS. Fully built RS232/USB adaptors are available.

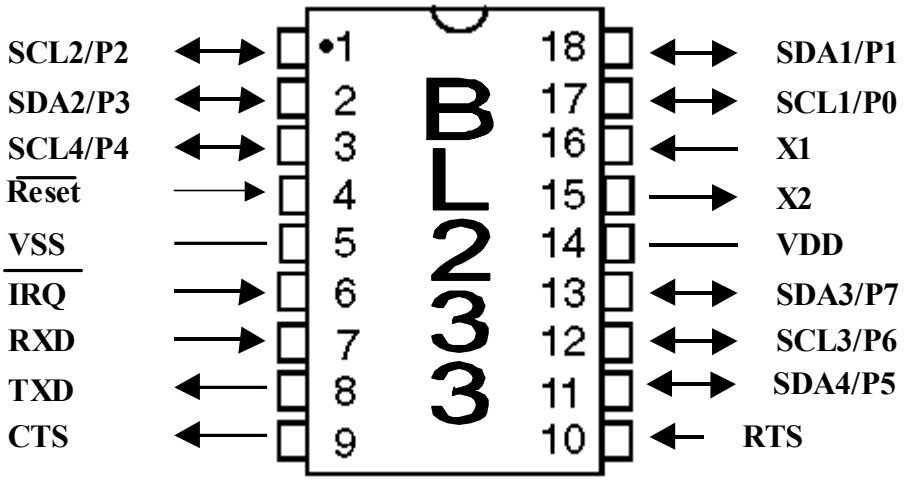


Figure 1 BL232/3 Pinout (DIP 18, SO-18)

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

BL233

RS232 – I2C/SPI/1WIRE ADAPTOR & CONTROLLER

Table Of Contents

FEATURES 1

APPLICATIONS 1

PROGRAMMABLE 1

1 DESCRIPTION 1

2 TABLE 1: BL233 PINOUT DESCRIPTION 5

3 I2C BUSSES 5

3.1 BIT TIMING 6
3.2 SCL STRETCH 6
3.3 FOUR WIRE BUS 6
3.4 FAST MODE 6
3.5 MULTIMASTER ARBITRATION 6

4 TABLE 2: COMMAND CHARACTER LIST 6

5 COMMAND FORMAT 8

5.1 A SIMPLE I2C TRANSACTION 8
5.2 STOP 8
5.3 READING I2C 8
5.3.1 Reading an indeterminate number of bytes: Pascal Strings *9*
5.4 10 BIT ADDRESSES 9
5.5 R AND W: FAST READ/WRITE COMMANDS 9
5.6 SELECTING THE I2C BUS 9
5.7 DELAYING 10
5.8 PAUSE COMMAND FOR TIMING CRITICAL SEQUENCES 10
5.9 I2C ACKNOWLEDGE 10
5.9.1 Writing *10*
5.9.2 Reading *11*

6 STATUS AND CONTROL REGISTERS 12

6.1 SETTING CONTROL REGISTER AND BUS TIMING 12
6.1.1 Bus Timing Value *12*
6.2 QUERYING STATUS REGISTER 13
6.3 INTERRUPT PIN 13
6.4 RETURN A CHAR 13

7 MESSAGE NUMBERS 14

8 DIRECT PIN I/O 14

8.1 READING PINS 14
8.2 WRITING PINS 15
8.2.1 Pin Timing *15*

9 SPI 15

9.1 CS/STROBE PINS 16
9.1.1 CS Pins for I2C-2-PC *16*
9.2 SHORT DATA 16
9.3 BIDIRECTIONAL (SIMULTAENEOUS) WRITE/READ 16

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

9.4 SPI CLOCK POLARITY 17
9.5 M5451 17

10 1WIRE BUS 17

10.1 PINS 18
10.2 SOFTWARE 18

11 EEPROM MEMORY 18

11.1 TABLE 3: EEPROM MEMORY MAP 18
11.2 MACROS 18
11.3 A MACRO FOR COMBINED 8 SWITCHES + 8 LEDS BOARD 19
11.4 A SIMPLE "PROGRAM" 19
11.5 WRITING THE EEPROM 19
11.5.1 Write Protecting the EEProm 20
11.6 DUMPING THE EEPROM 20

12 SERIAL COMMS 20

12.1 DATA FORMAT 20
12.2 HANDSHAKING 20
12.3 TABLE 4: SERIAL INITALISATION REGISTERS IN EEPROM 20
12.4 BAUD RATE 20
12.4.1 Special Pins Mode 21
12.5 MAX BAUD RATE AND SPEED 21
12.5.1 PC Serial Port Latency 21
12.5.2 RS232 Drivers 21
12.6 SERIAL BUFFER SIZE 21
12.7 USB 22
12.8 EOL AND SEPARATOR CHARS 22

13 OSCILLATOR 22

13.1 TIMER 22

14 WATCHDOG 22

14.1 USES 23

15 RESET 23

15.1 LOW VOLTAGE RESET 23
15.2 POWER ON RESET MACRO 23

16 SLEEP MODE 23

17 I2C BUS CONNECTORS AND PINOUTS 24

18 I2C BUS DEADLOCK 24

19 EXAMPLE APPLICATIONS 24

19.1 I2C-2-PC: DUAL INTERFACE USB & RS232 TO I2C ADAPTOR 24
19.2 SIMPLE I2C TO RS232 ADAPTOR 25
19.3 I2C EEPROMS: DUMPING 26
19.3.1 24C01 26
19.3.2 24C02 26
19.3.3 24C04 27
19.3.4 24C16 27
19.3.5 24C64 27
19.4 SIMPLE DATA LOGGER WITH NO PC SOFTWARE 27
19.5 ISOLATING AN I2C BUS 27

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

19.6 RS232 TO LCD MODULE ADAPTOR 28
19.7 HUGE RS232 PARALLEL PORT 28
19.8 HARDWARE WATCHDOG FOR NETWORK OR SERVER 28
19.8.1 Circuit 28
19.8.2 Operation 28
19.8.3 Code 28
19.9 TEMPERATURE LOGGER USING DS75/LM75/TMP101 28

20 MIGRATION FROM BL232 A, B 29

21 ERRATAE 29

21.1 XBL233-71 29
21.1.1 Wait for EOL (:) 29
21.1.2 1-Wire Buses 30
21.1.3 RTS Input 30
21.2 DATASHEET ERRATA 30
21.2.1 Set Control Register 30
21.2.2 Pin Number errors 30
21.2.3 Simple program example (11.4) 30

22 ORDERING INFORMATION 30

23 PACKAGE DIMENSIONS 31

23.1 18-LEAD PLASTIC DUAL IN-LINE (P) – 300 MIL (PDIP) 31
23.2 18-LEAD PLASTIC SMALL OUTLINE (SO) – WIDE, 300 MIL (SOIC) 32
23.3 SSOP ETC 32

24 CO-OPERATION 33

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

2 Table 1: BL233 Pinout Description

Name	DIP Pin #	SSOP Pin #	I/O/P Type	Buffer Type	User Pin	Description	Bit #
SCL1	17	19	I/oc[1]	ST[2]	I/0	I2C Bus #1	P0
SDA1	18	20	I/oc	ST	I/0	I2C Bus #1, 1 Wire	P1
SCL2	1	1	I/oc	ST	I/0	I2C Bus #2	P2
SDA2	2	2	I/oc	ST	I/0	I2C Bus #2	P3
SCL4	3	3	In	ST	I/oc	I2C#4, SPI#2 CS pin, 1Wire	P4
Reset	4	4	In	ST	In		
Vss	5	5,6					
Int	6	7	In	ST	I/0	0=interrupt	
RXD	7	8	In	ST	--	RS232 data IN, Connect to PC TXD	
TXD	8	9	Out		--	RS232 data Out, Connect to PC RXD	
CTS	9	10	Out		I/0	Connect to PC CTS, Tells PC to send	
RTS	10	11	IN	TTL	I/0	Connect to PC RTS, PC controls me.	
SDA4	11	12	I/O	TTL	I/0	I2C#4, SPI#3 CS pin, 1Wire	P5
SCL3	12	13		TTL	I/0	I2C Bus #3	P6
SDA3	13	14		TTL	I/0	I2C Bus #3, 1 Wire	P7
VDD	14	15,16					
X2 Out	15	17				Crystal / Oscillator Out	
X1 In	16	18				Crystal / Oscillator In (14.7456)	

3 I2C Busses

The BL233 supports up to 4 physical 2 wire I2C Busses. (1-4)
Two operate with Schmitt-CMOS levels, and 2 with TTL levels.
Only one bus can be active at one time. The bus will be put into Stop (P) state before selecting the new bus.
Bus numbers above 4 are logical busses, ie they map to the same pins as another bus, or work differently

- Full Split Bus, Separate in and out pins for both SDA and SCL
- Half split, Separate SDA IN pin
- *FAST* (400kHz) timing busses

[1] Open Collector drive when used for I2C bus
[2] Schmitt Trigger

- 1-Wire busses

SPI works on I2C busses, by setting the SPI mode. All busses operate in an open collector fashion, and **require external pullups**.

3.1 Bit Timing

Timing complies with the specs for standard 100kHz I2C. To slow the bus you may load another value into the I2Ctiming byte with the **J** command.
see 6.1:Setting Control Register and Bus Timing

3.2 SCL Stretch

SCL Stretch is supported. It is implemented on a bit-by-bit basis.
It is tested at the *start* of a cycle, or S or P states.
Why test at the *Start* when the slave asserts stretch at the end of a cycle?
The stretch is asserted by the slave after the ACK, however it does not impact the master until it attempts to begin the next byte/S/P. By testing at the beginning of the next byte, most SCL stretches will have no affect on throughput.
SCL Stretch should be asserted for <10ms. Clock stretch will timeout after approx. 2^{18}/fXtal secs. (17ms @ 14.7MHz). A clock stretch timeout is flagged in the status register.

3.3 Four Wire Bus

It is significantly easier to isolate a bus when the transmit and receive are split. A 4 wire logical bus can be used where galvanic isolation is required.
For most applications where SCL stretch is not used, 2 isolators are needed *from* the BL233, and only a single return SDA isolator. In this case a half-split bus can be used. You can use a single IL716 magnetic coupler from NVE[3].
Note that it is often easier to isolate the RS232 bus.

3.4 Fast Mode

Busses can also be operated with Fast Mode (400k[4]) timing. This improves throughput somewhat when high baud rates are used.

3.5 Multimaster Arbitration

There is no support for multimaster. BL233 assumes it is the only master on the bus. Where multiple masters are desired on a bus, for example in a ATE set, I/O pins can be used to handshake between masters, or to reset/disable the other master when the BL233 wants the bus.

4 Table 2: Command Character List

Char	Command	Chars to

[3] http://www.nve.com/
[4] The pulse timing is designed to comply with the Philips requirements for Fast Mode. The actual clock frequency is about 200kHz at 14.7MHz fXtal.

			Follow		
0-9 A-F	Hex		1-∞	lower case **a-f** are NOT hex	
:	PauseUntilEOL ForceReturnToRS232			Pause execution until EOL char received. (except in EE, see below) Execution *unconditionally* returns to RS232 buffer if in EE	5.8
;	ReturnToRS232IfChars			Execution returns to RS232 buffer *if there are chars in buffer*, otherwise continues in EE	
<	Return		0	Execution returns from a macro, either to calling macro or RS232 buffer	
=	unused				
>	Jump to EE		2	Execution jumps to macro in EE	
?	Query Status		0	returns status register	6.2
@	unused				
G	Set Bus Number	N	1-F	Sets the I2C Bus to use. Will execute STOP (P) before changing bus.	5.6
H	Hi-speed start			Future Use for HS mode	
I	Check Interrupt	I		Checks Int pin, and execute Int macro regardless of fInterruptEnable	6.3
J	Write Control Flags & Timing				6.1
K					
L	Delay NNNN ms	Lnnnn	4		5.7
M	Set Message Number	Mnn	0,2		7
N					
O	Output Direct to Pins		1-∞	write direct to pins and tri-state registers	8
P	I2C Stop	P	0		5.2
Q	Query Direct from Pins	Q	0	Reads pins directly	8
R	Read nn bytes	Rnn	2	Reads NN bytes from previous address (I2C) or SPI, 1Wire	5.5
S	I2C Start	S	0-∞		5.1
T	Type Char	Tnn…	2-∞	Types data back	6.3
U	Dump EEProm	U	0	Dumps *whole* EEProm	11.6
V	Write EEProm	Vaadd…		Write eeprom from address *aa*	11.5
W	Write	Wdd…		Writes bytes to previous address (I2C) or SPI	5.5
X	Software Reset	5A		"X5A" forces a power on reset	15
Y	SPI Mode	n		Set SPI Mode. N sets bits to send for next write to do short writes.	9
Z	sleep				16
,	(comma) ignored			avoid using for forward compatibility	

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

	(space) ignored				
0x80 0xFF	Chars>=0x80 are used as ASCII			Special mode, not normally enabled	

5 Command Format

The BL233 uses a simple printable **ASCII** format[5]. **HEX**[6] characters are used for all data and I2C addresses. Other characters are used as single char commands. Unrecognised chars are generally ignored[7].
Chars are generally processed and acted on immediately.
The basic form follows that shown in the Philips I2C documentation.
I2C Addresses: Our convention is to write addresses as 8 bit numbers including the R/W bit position. So an EEProm has a base address of 0xA0

5.1 A simple I2C Transaction

Consider writing 0xD7 to a Philips PCF8574 8 bit Port. The base address is 0x40.
Send:
S40D7P
[I2C Start][Address:0x40][Data:0xD7][I2C Stop]
The command S sends an I2C Start. P is an I2C Stop, just like the Philips documents.
To send a string of 80/81/80..… to the port:
S 40 80 81 80 81 P[8]
Repeated Start:
S 40 80 81 S 82 S83 P

5.2 Stop

P returns the I2C bus to the idle state. You should leave the bus stopped when it is idle. Interrupts and other automatic actions need the bus to be stopped, so they don't split an indivisible operation

5.3 Reading I2C

The adaptor looks at bit 0 (R/W bit) of the address byte. If 1, this is a read.
To read, set bit 0 of the address eg: read 2 successive bytes from address 0x40
S4102
[I2C start][Address:0x40 R/W=Read][Number Bytes to read: 02 (0x02)]
The adaptor replies with 0x83 both times:
8383[eol][9]

Repeated Start:
S407D S4101P

[5] Using printable ASCII is easy to understand and debug, and is easily passed through the Internet to a remote BL233.
[6] Only upper case A-F are considered hex. Lower case a-f will be ignored or recognised as commands
[7] Spaces and Commas are always ignored. You may freely use them to make strings more readable (if slower). You can use CR and/or LF if you wish.
[8] Spaces and commas will be ignored, and can be used for clarity (but will slow down comms)
[9] [EOL] is the End of Line char. It defaults to LF (0x0A), but can be changed in EEProm

7D[eol]
[start][write 0x7D][repeated start][read 1 byte][stop] [adaptor returns 1 byte)]

5.3.1 Reading an indeterminate number of bytes: Pascal Strings

Due to the way the I2C bus works, (master controls reads), the master has to know how many bytes to read. If the device is returning an indeterminate number, eg a string, then we have a problem.
The BL233 has support for reading pascal style strings. If the *NumberOfBytes* is 0, then it will take the first byte read from the I2C slave, to be the string length.
eg an I2C slave (0x40) has the pascal string "Hi", ie 0x 02 48 49
S4100
4849[eol]

5.4 10 Bit Addresses

Carefully study the Philips notes on 10 bit addressing. In short: you *write* 2 address bytes, so any *read* is done by a *write(2 address bytes)*, then a *read*, with only the first address byte.
The R and W commands are particularly useful with 10 bit addresses to reduce traffic.

5.5 R and W: Fast Read/Write Commands

Where you want to read or write to the previous address use R and W[10]. eg instead of
S4083 S4101 S407D P S407E S4101 P
Using R and W to read and write the previous addresses is much shorter.
S4083 R01 W7D W7E R01 P

10 Bit addresses will be correctly handled.

5.6 Selecting the I2C Bus

G selects the I2C Bus. If the bus is not in the stop (P) state, then the adaptor will execute a P before changing the bus.
Busses 1-4 are separate busses. Busses > 4 use the same pins in different ways. It is important to write your code to allow bus numbers to be easily changed to allow for future BL23X chips.

S4083 G2 S4084 G1
[Write 0x83 to PCF8574 on Bus1][Select Bus 2][Send 84 to another 8574 on Bus2][return to bus 1]
Multiple I2C Busses allows you to have more of one type of chip than sub addressing allows. (eg 24 PCF8574's).

Bus #	SDA	SCL	SDA In	SDAOut	SCLIn	SCLOut	CS	Fast	
1	18	17							Bus 1 Standard I2C
2	2	1					3		Bus 2 Standard I2C
3	13	12					11		Bus 3 Standard I2C
4	11	3							Bus 4 Standard I2C

[10] R & W are also used for SPI operations

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

5	18	17						#1	Bus 1 Fast I2C
6	2	1					3	#2	Bus 2 Fast I2C
7	13	12					11	#3	Bus 3 Fast I2C
8			2	13	1	12			Full split, Bus2 is inputs, Bus 3 outputs
9		1	3	2					Bus 2 Half Split
A		12	11	13					Bus 3 Half Split
B									
C	18			17					1Wire#1
D	3								1Wire#2
E	13			12					1Wire#3 *See Errata!*
F	11								1Wire#4 *See Errata!*

Some bus numbers use some or all of the same pins as another bus.

5.7 Delaying

To output a pulse we can use Delay (**L**). Delay is up to 65535 milliseconds[11].

```
S4001 L0400 W00
```

[Set bit0 of PCF8574 HI][Delay 0x400ms ie ~1 sec][Set bit 0 LO]

5.8 Pause Command for Timing Critical Sequences

'**:**' (colon) pauses execution until an EOL char (CR or LF) is received. Chars after the '**:**' are stored in the buffer, but not processed until an EOL char is received. This ensures that the sequence of operations between '**:**' and EOL will always take the same time, irrespective of baud rate or undefined computer delays.

```
:S4001 L0001 W00 [eol]
```

[pulse bit0 high for 1ms]
(hint: using ":" can make it easier to watch data sequences on a oscilloscope)
Obviously the string between '**:**' and EOL must fit in the RX buffer.
Be careful when using wait for EOL with RTS/CTS handshaking. If the handshake line is raised after ":" is sent, then it will be impossible to send the matching linefeed, so a deadlock will result. Setting handshake to NONE during the wait for EOL phrase can avoid this.
An alternative for timing critical sequences is to put them in the EEPROM as a macro.

5.9 I2C Acknowledge

Read the Philips I2C documentation closely.
At the end of each byte transfer there is an acknowledge.

5.9.1 Writing

When writing the *slave* generates ACK. (pulls the bus LOW)
If the slave does not ack, then either

- It is not present

[11] Timer tick rate can be changed in EEProm

- It is not ready or prepared to accept any more data

Most hardware designs and hardware type I2C slaves are deterministic.
- The device is always present if it was present once.
- Most devices always accept the number of bytes you will try to send

For this reason the default is to ignore nack. You can check the result of the last writes by QueryStatus (?). This is an easy way to check the bus for I2C devices at power on. *Nack* is cleared at each START. Note that "?" can be put between data bytes.

Two fControl flags offer more sophisticated nack handling

fControl			
Bit	Rst	Name	Description
3	1	WrIgnoreNack (default)	makes I2C write routines ignore nack. The interface will continue to write bytes to the bus even after a nack. Writes halt when nack occurs. Subsequent bytes are ignored. Writes "N" back to host
1	0	AckWrites	writes K/N as each byte written to I2C is/not acknowledged by the slave "K" nack'd OK "N" failed to nack Useful for debugging. Clears Nack at each byte

5.9.2 Reading

When reading from a slave the nack is generated by the master (BL233) as each byte is received.
The I2C Bus requires that after receiving the last byte a master doesn't nack. This signals end-of-data to the slave.
The BL233 handles this automatically.

However you might wish to read a large number of bytes from a slave (eg >255), or read as a series of shorter blocks, without signalling end-of-data to the slave.
In this case you can set the flag fControl*ReadBlockAck*. In this case *all* reads will be ack'd, and you must take care of the last read yourself with an N before the final read. Note that you can read multiple blocks by sending a series of byte counts. So you could use "R 02 03 04 N 01"

"**N**" makes the last read NOT ack, thus signalling the end to the slave.

Note that this all seems a bit complicated, and for most slaves you can probably just end with a stop.

6 Status and Control Registers

fControl			
Bit	Default	Name	Description
7			
6	0	EmitWriteResult	returns data read as byte written (see SPI)
5	0	RxWatchdogEnable	enables the watchdog on rx chars
4	0	InterruptEnable	enables interrupt response
3	1	WrIgnoreNack	makes I2C write routines ignore nack
2	0	RdBlockAck	nack won't be sent when last byte is read
1	0	AckWrites	writes K/N as each byte written to I2C is/not acknowledged by the slave
0	0	MessageNumsEnabled	enables return of message numbers

fStatus				
Bit		Default	Name	Description
7				
6			1WirePresence	if a presence pulse exists during reset
5			1WireInterrupt	A device signals INT during reset pulse
4			Int	Current state of Interrupt pin (0=active int)
3	*		TimerTimedOut	
2	*		FifoOverflow	Either RX or TX fifo overflowed
1	*		ClkStretchTimeout	I2C timed out waiting for SCL to go Hi
0			Nack	result of last I2C Nack

6.1 Setting Control Register and Bus Timing

To set the control register only.
J [control register]
To change the I2C Bus timing. 0 is fastest, 255 is very slow.
J [control register] [I2C Timing]
eg Write only fControl to enable interrupts (and WriteIgnoreNack)
`J14`
eg To Enable Ints *and* set I2Ctiming to 15
`J140F`

Note: You cannot read back the control register or timing values.

6.1.1 Bus Timing Value

Bit/Bus timing is set by the **J** command. The Timing value is 0 – 0xFF. Experiment to see what the timing is.
Different bus modes (eg normal, fast, 1-wire etc) have different bit timing for the same value. The timing is also non linear at small values eg 0,1,2 can be quite different.

* Cleared by reading status register

If you use different timing for different buses, then you have to change it when you change bus#, as the timing value is global to all buses.

6.2 Querying Status Register

? returns the status register. This is most commonly used to

- check NACK to see if the last write was acknowledged by the slave.
- check state of the interrupt pin (0 is INT)
- check flags after a 1Wire reset

"?" does not interfere with the command currently being executed, so you can put it in the middle of a series of bytes being written, or between reads.

6.3 Interrupt Pin

- You can check the interrupt pin by reading the status register with **?** at any time
- **I** command checks the interrupt pin, and executes interrupt macro in EEProm, regardless of the state of the fInterruptEnable. The address of the interrupt macro is set in the IRQ1Vector in EEProm.
- If fInterruptEnable is set, then interrupt pin is checked at each *Stop* (**P**) command, and the interrupt macro executed if it is low.

The default interrupt routine just types "i" back.
To enable interrupts set the InterruptEnable bit of fControl register eg.

```
J18
```

[enable interrupts]

```
J08
```

[disable interrupts]

If you want to wait for interrupts, then a simple macro in eeprom will loop waiting for interrupts.

```
I;>60
```

(start is at location 60)[Check Ints][return if the PC sends a char][jump to 10 (start)]
Change the interrupt macro to be

```
T69:
```

[type "I"][leave EEProm]

Then to run the "wait for interrupt" macro

```
>60
```

Now when INT goes low, a single "i" is typed. (with the the original macro "T69<" it returns, and prints "i" repeatedly while INT is low)

The Interrupt macro, and the InterruptVector are in EEProm. See 11 for more details.

6.4 Return a Char

T types a char back to the PC. It is very useful for formatting and synchronising data.

```
S407D T563D R01
```


ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

V=7D[eol]
[set 8574 to 0x7D][Type "V=" back to PC][reads 8574: 0x7D]

- make return data more readable or parseable
- you don't have to wait for a specific reply (see also: 7 Message Numbers)

The start-up welcome "Hi I2Cad VXXX" message works this way.
Remember that the I2C adaptor is going to return upper-case hex chars, and a few special upper case chars in certain cases (eg N,K)
Type different chars or lower case chars back, then it is very easy to separate them out of the data stream.

7 Message Numbers

Message Numbers provide a way of dealing with latency. The message number is 8 bits
Without message numbers it can be difficult to pair up commands and returned data, especially when long latencies are involved. eg

```
S407D R01 L0100 R01
7D[eol]7D[eol]
```

If we turn on message numbers, and set the message number to 0x05

```
S407D M05 R01 D0100 R01 D0100 R01
057D[eol]067D[eol]077D[eol]
```

Now it is easy to pair each query to its reply.
Message numbers are 8 bits, autoincrement, and wrap around from 0xFF -> 0
Message numbers are enabled by **fControlMessageNumsEnabled**, or by the first occurrence of an M.
M without data will clear the message number.

```
S407D M05 R01 M R01 R01
057D[eol]007D[eol]017D[eol]
```

Another way to synchronise, is to use "**T**" to type specific chars back.

8 Direct Pin I/O

The BL233 makes a great 8 bit I/O port for a PC! You can read and write 8 pins directly.

8.1 Reading Pins

'**Q**' *queries* the actual pin state. In the I2C-2-PC adaptor there are pullup resistors on the pins so the idle high.
One use for **Q** is checking for bus faults and the deadlock condition, after an *I2C Stop*.

8.2 Writing Pins

The '**O**' output command can be used to write to the output pins, and the data direction registers. The bit number of each pin is given in 2 Table 1: BL233 Pinout Description. Before you can use a pin as an output (or CS for SPI) you must set its tristate bit to 0.

Expects sequence of **O** Data, Tristate , Data, Tristate …..
You can write 1 or more bytes in this sequence

`O 0F 0E 5A`
[write][Data=0x0F][tristate=0x0E][Data=5F]

Inputs have tristate register bits = '1'. Outputs have tristate bits=0.

Once you have set the tristate register you can simply write to the data register:
`O 1F O FF`
You have to take great care using direct pin I/O when you are also using serial busses, as the bus functions of the chip expect the registers to be set up a certain way.
All bus pins have DATA bits=0, TRIS bits=1.

If you are only using the two CS pins see below.

To restore the idle condition send
`O 00 FF`.
CS pins on the I2C-2-PC will idle high as they have pullups.

8.2.1 Pin Timing

Writes and reads to the pins are done with bits 0-4 changing first, and bits 3-7 ~1µs later.

9 SPI

8 bit bytes are written using SCL and SDA. Data is shifted out MSB first, (as is I2C). The bus must be in *STOP* before using SPI write. The SPI **strobe** or **CS** pin function is done using direct I/O above.
SPI Mode is entered by the "Y" command. SPI mode is exited by "Y0"

`P Y W 01 02 03`
[I2C Stop][SPI Mode] [write 0x01,0x02,0x03]

`P Y R 03`
[I2C Stop][SPI read 3 bytes]

`P Y W 01 02 03`
[I2C Stop][SPI write 0x01,0x02,0x03]

9.1 CS/Strobe Pins

These are driven by simple bit bashing using the direct I/O command "**O**" above.
Don't forget that the direction/tristate byte must be set before writing data bytes
First initialise to use CS#2 as an output:

```
P O 10 EF
```

[I2C Stop] [Set Bus2 CS to output and high]
Then we can write to the port.

```
O 00    Y W 03 04    O 10
```

[Set bit 4 (Bus#2 CS) LOW][SPI write 2 bytes (0x03,0x04)][Set bit 4 (Bus#2 CS) HI]

9.1.1 CS Pins for I2C-2-PC

The I2C-2-PC can use Buses 2 & 3 for SPI transfers if the jumpers are changed to select the CS pins.
Before using the CS pins you need to write 0 into the appropriate TRISTATE bits.

	Pin#	Bit#	TRIS Value	Initalise (CS=1)	CS=1	CS=0
CS on Bus #2 Only	3	4	EF	O 10 EF	O 80	
CS on Bus #3 Only	11	5	DF	O 20 DF	O 10	
CS on Bus #2&#3			CF	O 30 CF	O 18	O 00

9.2 Short Data

Unlike I2C, SPI can have any data length. You can short transfer by specifying the bit count after Y as a single hex digit. This only shortens the next byte. Subsequent transfers will be 8 bits. Data is in the least significant bits of the byte, and data read will have 0 in the unused bits.

```
Y W 5A Y2 W 03 AA Y2 R 02
```

[SPI][write byte 0x5A][Set for 2 bits][Write 2 bits 0x3][write fullbyte 0xAA][read 2 bits, then 1 full byte]

Bit count=0 exits SPI mode.
Bit count 8-15 is for bidirectional mode below.

Note you can write directly after the Yn command eg

```
Y2 03 AA 55
```

[write 2 bits (0x03), then full bytes 0xAA, 0x55]

9.3 Bidirectional (simultaeneous) Write/Read

Some SPI and 1Wire operations need to read in the data pin as data is written out. This is supported by the *fControl* bit *fControlEmitWriteResult.*

To enter this mode use "Y" with bit count>=8
To exit the mode use "Y" with no bit count.
For each byte written, a byte (2 hex digits) is returned.

Alternatively you can write *fControl* directly. Note that changing the bus will clear this bit.

9.4 SPI Clock Polarity

BL233 does not change the data and clock at the same time. So it will work writing data to both rising and falling edge clocked devices. There is more timing margin at the rising edge, so this is preferred.
Read data is sampled when SCL is Hi, just before the falling edge.
SCL idles LO between transfers.

9.5 M5451

The M5451 devices do not use a CS or STB line. They take 36 bits data, the first bit is 1. To reset/synchronise them after power on write 5 bytes of 00.
To write to them it is fastest and easiest to send 5 bytes of data. The first data bit should be one, and all trailing unused bits must be 0.
Only one M5451 can be connected to each bus.

If using 7 segment displays with an M5451, we recommend the BL301 I2C Versatile Display Driver. It handles the 7 segment decoding and bit re-ordering for you, and connects multiple M5451's to a single I2C bus.

10 1Wire Bus

The BL233 has support for Dallas 1 Wire bus, using the standard timing.

- 1Wire busses are selected by bus "G" command, followed by a start
- Start sends bus reset, and samples the presence and interrupt responses
- Write and Read do data transfers
- short bit count transfers and bidirectional Wr/Rd's can be done with the "Y" command as per SPI. This is useful for searching for unknown devices.

Eg to read a DS2401 serial number
`GC S ? W33 R04`
[Select 1 wire bus#1 (bus #C)][Start(reset)][Get status to check presence][ReadRom Command][Read 4 bytes of rom data]
Since **S**tart can be followed immediately by write data, a shorter version is:
`GC S 33 R04`
[Select 1 wire bus#1][Start][ReadRom command][read 4 bytes]
The 1Wire Reset (Start) checks for interrupts[12], and presence of a chip. Read the fStatus ("?") to check these flags. If you are using an iButton application where it is

[12] Note we mean 1wire interrupts, which are signalled by devices stretching the 1wire reset pulse. Not the INT pin that supports the I2C bus

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

possible to short the bus, you can directly query the bus pin with the "Q" command, if interrupt and presence are both true.

10.1 Pins

1Wire uses a single open collector I/O pin, and a 1k5 pullup resistor. However we also have a second output-only pin on some buses. This allows for easy galvanic isolation, or for improved bus driving hardware. For a simple bus, you can just use the SDA pin, and ignore SDA_OUT

10.2 Software

You can easily use BL233 with the Dallas "Open Source SDK" for high level 1Wire software. BL233 provides support for byte-read, byte-write, N-bit read/write, "touch" (write & read). With these you can get complete 1 wire functionality.

11 EEProm Memory

The 128 Bytes of EEProm contains:

- Stored Settings
- Vectors
- Macros

Note that these are mostly initialisation values loaded at reset, and may not change or reflect the present value or have any effect until reset.

11.1 Table 3: EEProm Memory Map

Address	~Add	Register	Description	see
0xF7	08	fSerial	RS232 Control Register	12.3
0xF8	07	Baud_Div	Baud rate divisor reset value	12.4
0xF9	06	TimerDivL	***Low*** byte of timer divisor (TimerDiv)	13.1
0xFA	05	TimerDivH	***Hi*** byte of timer divisor	13.1
0xFB	04	fControl	Control Register	6
0xFE	01	IRQ1Vector	Macro address for interrupt	6.3
0xFF	00	Watchdog Vector	Macro address for watchdog	14
0		Base / POR	First macro instruction. Power-On jumps to this location.	
0x76		end	Generally contains a return char "<"	

11.2 Macros

Macros are stored command chars.

- Commands to execute at power on (see 15)
- Commands to execute when Interrupt is detected
- Commands to execute when there is a watchdog timeout
- macros to speed up common tasks
- very simple "programs" for autonomous actions

11.3 A Macro for combined 8 switches + 8 Leds board

This board uses (you guessed it) an 8574. The 8574 is a open-drain driver. We can connect switches across the drivers, and read the switches if we momentarily turn the LEDs off (write 1's)

```
S40FF R01 W <
```

[write 0xFF to turn leds off][read 1 byte][start write][return] we now expect the LED data to follow.
To use the macro:

```
>20 00 P
```

[goto macro][new led data][stop]

11.4 A simple "program"

This is a simple data-logging application. It reads the 4 A-D channels of a PCF8591, and 8 switches with an 8574, 4 times a second, and streams the data to a PC[13], that captures it with a terminal program[14].
start location is 0x10

```
S9104 S4101 P L00F9 ; >10
```

[read 4 a-d][read 8574][delay 249ms][return if the PC sends a char][jump to start]

11.5 Writing the EEProm

To prevent inadvertent writes to the EEProm, the start address is sent twice. First normally, then inverted. If they don't match, the whole operation is aborted.
The format is V[address][not addr][byte1][byte2]…..

Address 0 is the Base / POR location. So to write F1,01,11 to the POR location:

```
V00FF F1 01 11
```

Each byte written takes 10ms. You need to either send data slowly enough, or in small packets eg 8 locations to avoid overflowing the buffer.
Most settings will only change after a reset.

The EEProm write command is intended for configuration, rather than routine use. You must check that vital data is correct. In particular if the serial parameters are wrong, you will be unable to regain control after the next reset.

[13] You could use a TI or HP graphing calculator too.
[14] REALTERM.EXE from http://realterm.sourceforge.net

(hint: to find the hex for the char string you want to put in EEProm, set Realterm's display mode to hex+ascii, half-duplex. Now type the string, and you will see the hex.)

11.5.1 Write Protecting the EEProm

If fSerialEEWriteProtect is set, further writes to EEProm are blocked unless in Special Pins Mode

11.6 Dumping the EEProm

U will dump all 128 bytes of EEProm in the order shown in Table 3: EEProm Memory Map. ie the first byte is fSerial, not Base/POR

12 Serial Comms

12.1 Data Format

- 8 bits
- No parity
- 1 stop bit

Data format cannot be changed.

12.2 Handshaking

RTS/CTS hardware handshaking is supported. It is controlled by bit fSerial_UseRTS. CTS will always be output, regardless of *UseRTS.* RTS:See errata
XON/XOFF handshaking is not supported.

12.3 Table 4: Serial Initalisation Registers in EEProm

	bit 7	bit 6	bit5	bit4	bit 3	bit 2	bit 1	bit 0	
fSerial	EEWr Prot.	Enable Sleep				UseXon	UseRTS	Baud Rate HI	
BaudDiv	Baud Rate Divisor								

12.4 Baud Rate

The baud rate is set by writing the ee_BaudDiv and ee_fSerial locations in EEProm. The changes will become active when the part is next reset.

$$\text{if BaudRateHi} = 1: BaudRate = \frac{Fxtal}{16 \times (BaudDiv + 1)}$$

$$\text{if BaudRateHi} = 0: BaudRate = \frac{Fxtal}{64 \times (BaudDiv + 1)}$$

***BaudDiv* Values for other Baud Rates (14.7456MHz)**							
BaudDiv: 0x	03	07	0F	17	2F	5F	BF
Baud Hi	230k	115K	57.6k	38.4k	19.2k	9600	4800
Baud Lo				9600	4800	2400	1200

Baud rates can be calculated at other crystal frequencies from:
BaudRate HI= Fxtal/(16(N+1))
BaudRate Lo= Fxtal/(64(N+1))

For example to change the baud rate to 9600,
V F807 5F
[write eeprom][address, ~address][BaudDiv=0x5F]
Changes are active from the next power on, not immediately

12.4.1 Special Pins Mode

As you can cock the baud rate up, and be left with an unusable chip, we have provided a special mode where the baud rate and fSerial are in their factory state. To enter SPM, force Pins 1,2,17,18, and 6 to 0V, during and 500ms after reset / power on. Now you can change the EEProm values.

12.5 Max Baud Rate and Speed

The BL233 can have baud rates as high as 921kbd. However it cannot process commands at this rate. Different commands take differing amounts of time to execute, and so you should experiment to find the optimal rate for your application. It is likely to have an appropriate throughput for 115kbd.
Where speed is an issue, make use of macros stored in EE to reduce the serial traffic.

12.5.1 PC Serial Port Latency

While Windows PC's, USB devices, and anything running over the internet, have high data throughput (bytes per second), they have long latencies or transit times.
On a Windows PC that can send 10,000 bytes/sec, it may take as long as 300ms for a single char to loop back.

12.5.2 RS232 Drivers

At high baud rates (20k +) you need to pay attention to the RS232 Drivers. Driver IC's vary in their ability to drive longer cables at high speed. Variants, eg xxxE parts can have different speeds. Different manufacturers versions of the same part have different speeds. Also beware that at the PC end, some serial ports eg older laptops, may use low power drivers that are rated to 20k only.

12.6 Serial Buffer Size

It is best to keep command strings within the buffer. When reading a port, avoid reading more chars than will fit in the buffer.

- RXD Buffer: 48 Chars *from PC*
- TXD Buffer: 80 Chars *to PC* (each byte read will be 2 hex chars in the buffer)

12.7 USB

The usb has a drawback. Data is passed in 1ms frames. This means that there is a minimum 1ms turnaround for even a single char. Structure your data so that you send as many commands as possible before your program requires a response.

12.8 EOL and Separator Chars

EOL (LF 0x0A by default) is sent back to separate commands.
Unimplemented Features:
Both the EOL char and the separator char can be set in EEProm. They can be changed to get a different data format that is easier to use.

The separator char is not used by default but may be set to any value that suits the particular program interpreting the data. Eg comma, space, or tab. This will add 50% comms overhead of course.
e.g. Change separator to TAB, and EOL to CR, to make Excel readable files

13 Oscillator

The standard oscillator frequency is 14.7456MHz. This permits standard baud rates as high as 921kbd.
As the BL233 is much faster than many applications will require, you can change the crystal for a lower frequency. This will allow operation at lower voltages, lower supply current, and reduced EMI.
Both BaudDivisor and TimerDivisor can be set in EEProm.
The max oscillator frequency is 20MHz. The I2C Bus timing is set for *Standard Mode* at 14.74MHz

13.1 Timer

The timer ticks run at 1ms with a 14.7456MHz crystal. You can change the timer rate using the 16 bit EEProm register TimerDiv[15].

$$TimerDiv = \text{round}\left(\frac{\text{Fxtal}}{4 \times TimerRate}\right)$$

Timer Rates @ 14.7456MHz						
Timer Div	0x0E66	0x01CC	0x3840	0x4800	0x9000	0xFFFF
Timer	1ms	2ms	256Hz[16]	5ms	10ms	17.8ms
Watchdog	66s	2m 11s	4m 15s	5m 28s	10m 55s	19m 25s

14 Watchdog

The chip has a watchdog function that is used to monitor correct functioning of the host computer. A typical use might be to force a hardware reset of the host PC, or to force outputs into a safe idle state if the controlling PC crashes.

[15] numbers are stored low byte first in EEProm
[16] High byte if TimerDiv will be in secs at 256Hz Timer rate

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

The watchdog period is fixed at approx 2^{16} timer ticks, normally ~65secs.
Each time a character is received from the host, the watchdog is restarted. After 2^{16} ticks without a character received, a watchdog event can be generated.

If the watchdog vector is not 0, and the RxWatchdogEnable is set, then execution jumps *immediately* to the watchdog macro. The current character will be processed, then the next char is fetched from the watchdog location.

The RxWatchdogEnable flag is cleared, and any pending chars in the input fifo that are waiting for an EOL char, are unblocked.
You need to bear in mind that bus transactions may be left half completed, and so you may need to use **P,S,G** or other commands before using the I2C bus.

The default Watchdog macro just types "w"

14.1 Uses

- Use a relay board to reset the PC if the flow of chars stops.
- Attach to a PC on the far side of a router getting chars through the network. Turns the router power off for a few seconds when the flo of chars stops.

15 Reset

X5A causes the part to do a power-on-initialise. (5A is required to prevent inadvertent operation).

- any characters following the X5A will be flushed from the buffer and lost.
- reset will take several milliseconds, so you must pause before sending further commands
- The POR macro will execute at power on *and* X5A reset.

15.1 Low Voltage Reset

As standard internal reset is generated at ~4V. The BL233 will operate to <3V, but you need to request the "Low Voltage Option" when ordering.

15.2 Power On Reset Macro

At power on or reset execution begins at 0 in the eeprom. The factory default macro types "HI I2C v104" using the **T** command.
Caution: If you write a macro that has an endless loop as the POR/Reset macro, you will be unable to recover the chip.

16 Sleep Mode

not implemented
If *fSerial.EnableSleep* is set, **Z** will put the part to sleep. (by default sleep is disabled) It will wake on pin change. This could be from an I2C interrupt, or by the PC toggling RTS. If you need the PC to wake up the via RXD, connect RXD to SDA4 through a resistor.
You need to allow 10ms for wakeup from sleep.

If you are waking it with the RXD line you need to be aware of the potential for waking up part way through the char used to wake it. For this reason 0xFF is recommended as the wakeup char.
Due to the perils of putting the part to sleep this function is not enabled. Contact the factory if your application needs this function.

17 I2C Bus Connectors and Pinouts

We use this pinout, and recommend that you do also. More details, and connector part#'s, suppliers etc see: http://www.i2cchip.com/i2c_connector.html

Pin#	6 Way	4 Way	
1	SDA	SDA	
2	+5	+5	
3	Gnd	Gnd	
4	SCL	SCL	
5	INT		Interrupt input (active low). Can be used as CS when being used for an SPI bus.
6	VAux		Aux supply (eg 12V), or other uses.

18 I2C Bus Deadlock

A known issue with the I2C Bus is the possibility of a lockup. SMBUS has addressed this with a requirement for an automatic timeout.
This condition can arise when a master reads a slave, and (for whatever reason eg noise pulses) the slave gets too many or too few clock pulses, and the slave is left asserting a 0 on the bus. In this case the I2C Stop cannot be sent to the part, as it is holding the bus low.
Software can check for this condition by using the Q command to directly read the port pins and check that the SCL and SDA pins are high after an I2C Stop has been sent. You may be able to clear this condition by sending a series of start and stops. This will have the effect of clocking the slave until its output is high, then it will recognise STOP. So 9 starts and stops should be able to clear the bus. ie
SPSPSPSPSPSPSPSPSP
(note: We have not tested this: it is a suggestion in response to a user query)

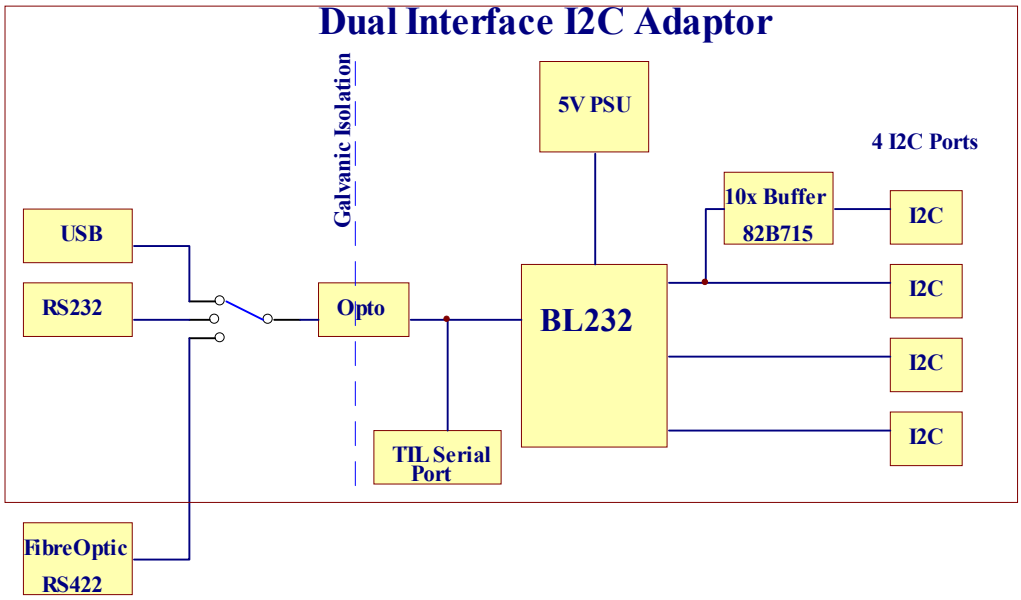
19 Example Applications

19.1 I2C-2-PC: Dual Interface USB & RS232 to I2C Adaptor

This 45x80 mm module can be used standalone in its enclosure, or as an OEM module in your equipment to provide a complete USB ***and*** RS232 interface with 3 I2C ports, and 1 buffered high current I2C.
Galvanic isolation can be installed where the I2C busses need to be isolated from the computer.

There is an internal 5V power supply to run the interface and target hardware, but for USB bus power can be used. Building an I2C based instrument needs nothing more than your I2C chip, everything else is included.

An optional Fibre Optic interface is available where very high voltage or totally secure isolation is required.
An RS422 interface is available for long distance or high noise immunity.
Ferrite beads are fitted to all data lines for EMC protection.



19.2 Simple I2C to RS232 Adaptor

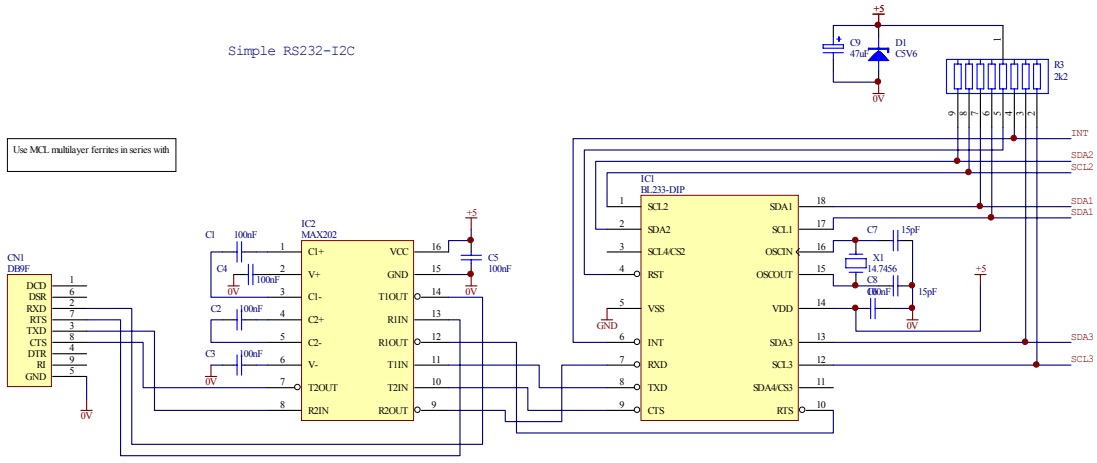


Figure 2

19.3 I2C EEProms: Dumping

These differ according to the amount of memory. Up to 16k (2kB) there is a single byte word address to write.
The examples below dump the data as 64 bytes per line for convenience.

19.3.1 24C01

128 Byte EEProm.

```
SA0 00
R40 R40 P
```

19.3.2 24C02

```
SA0 00
R40 R40 R40 R40P
```

19.3.3 24C04

The 512x8 and up chips use multiple addresses for each 256 byte page.

```
SA0 00 R40 R40 R40 R40P
SA2 00 R40 R40 R40 R40P
```

19.3.4 24C16

```
SA0 00 R40 R40 R40 R40P
SA2 00 R40 R40 R40 R40P
SA4 00 R40 R40 R40 R40P
SA6 00 R40 R40 R40 R40P
SA8 00 R40 R40 R40 R40P
SAA 00 R40 R40 R40 R40P
SAC 00 R40 R40 R40 R40P
SAE 00 R40 R40 R40 R40P
```

19.3.5 24C64

Starting from 24C32 there are 2 address bytes to write before reading

```
SA0 00 00
R40 R40 R40 R40P
R40 R40 R40 R40P
```

and repeat for all the data….

19.4 Simple Data logger with no PC Software

Use a Philips PCF8591 4 channel 8 bit A-D chip.
Store this macro in the EEProm at location 0.

```
S90 04 R04 T0A L0200 >00
```

[Start][address 0x90 (8591)][Control register=0x04][Read 4 channels of A-D] [Type LF (linefeed)][delay 512 ms][goto start]

The macro will run automatically at power on, sending 4 channels of A-D data in hex, followed by LF.
Capture the data to file with Realterm.
Plot it with Excel, Matlab etc.
You don't need to send anything from the PC, or write any software. Almost any A-D convertor could be used.

19.5 Isolating an I2C Bus

The I2C Bus can be isolated in two ways.
The easiest is to isolate the serial data stream, and the I2C-2-PC adaptor can be fitted with an isolator to do this.
However sometimes you want to isolate one bus from others. An example is a current sensor in the positive rail of a power supply.
In this case you can use the Split Bus feature. This eliminates the need for an expensive part like the 82B96

19.6 RS232 to LCD Module adaptor

A single PCF8574 can drive a standard LCD module. This circuit is available built up. See the BL301 Versatile Display driver for an easier way to do this.

19.7 Huge RS232 Parallel Port

To get are large number of cheap and easy output pins with good drive capability, use the SPI output, and 74HC4094 shift registers. Eg a 32 bit output with 4x 4094's. SDA4 is used as STB for the 4094's

```
G3 Y W01020408 OFFFD OFFFC
```

[select bus3][SPI mode][send 4 bytes to 4094's][pulse STB pin to latch data] nb **O** not zero

19.8 Hardware Watchdog for Network or Server

Servers and Routers commonly lock up, and need to be reset or de-powered to restore service to whole areas of offices and factories. A BL233 makes an easy watchdog to restore service.

19.8.1 Circuit

Connect a relay energised by the P4 pin going low. Put the normally-closed contact in series with the mains lead of the server or router. ie When the relay is energised, the server is turned OFF.

19.8.2 Operation

To watchdog a server, run a program on the server to send a char to the I2C-2-PC every few seconds. After 65 secs without a signal, it will reset the machine. Changing the TimerDiv adjusts the period up to 20 minutes.
To watchdog a router, use a program to check the router connection (eg ping across the router), and send a char to the watchdog when it is OK. Not that both the router, and the PC should be on the watchdog.
Realterm can be used to send chars to the watchdog from a batchfile.

19.8.3 Code

Put this string in the Watchdog macro of the BL232.

```
O EF EF L1000 OFF EF J82;
```

[turn relay ON][wait 4 secs][turn relay off][enable watchdog][return]
At 0, (power-on macro) enable the RX watchdog.

```
J82
```

[enable RX watchdog]

19.9 Temperature Logger using DS75/LM75/TMP101[17]

To read the 2 byte temperature:

```
S 91 02 P
```

[17] Examples used DS75 datasheet. LM75 and TMP101 have similar functions, but may be somewhat different

1910[LF]
[start][read 0x90][2 bytes][stop] [replies with 0x1910 = 25.0625°C]

To initialise the DS75 to maximum resolution (as it powers on at 9 bit resolution)
S 90 01 60 W 00 P
[Start][Add 0x90][point to config register][set config for max resolution][move point to temperature register]

A macro to read the temperature every second is:
S 91 02 P L0400 >0B

Putting it all together (and removing the spaces), you could store this macro at 0. It runs at power on, initialising the DS75, then loops reading the temperature every second.
S900160W00PS9102PL0400 >0B

To capture the data on the pc just run realterm from the command line.
realterm.exe capfile=temperature.dat capture

20 Migration from BL232 A, B

BL232-A is very similar for I2C use. BL232-B is a subset of BL233 for cost sensitive high volume applications, and works the same

- More Busses with different numbers
- Full and half-split busses
- Added Support for SPI Read, Write-Read
- SPI arbitrary bit lengths
- Dallas 1 Wire bus support
- Direct port I/O is simplified and more reliable

21 Erratae

21.1 XBL233-71

21.1.1 Wait for EOL (:)

Only a single wait for EOL phrase (colon) should be sent at once. A second should not be allowed to get into the buffer. This problem particularly arises when phrases can't be executed at once eg where the delay operator is used, or with slow I2C bit rates.
:L0100?[LF] is OK
:L0100?[LF]:L0100?[LF] is NOT OK
:S420102P[LF]:S42F00FP[LF] is normally OK, because the first phrase is executed while the ":" is arriving.
(see also 5.8)

21.1.2 1-Wire Buses

Only 2 1-Wire buses are implemented.
#1 (GC to select) is on I2C bus1, pin18 &17. Not bus2, pins 1,2.
#2 (GD) is on pin pin3. GE,GF are unimplemented and should not be used.

21.1.3 RTS Input

The RTS input is ignored regardless of *fSerial_UseRTS*. This means that the PC cannot stop the BL233 from sending.
CTS works correctly, ie the BL233 will stop the PC sending.

21.2 Datasheet Errata

Corrections have been made in these areas compared to earlier revisions.

21.2.1 Set Control Register

"J04" in example of setting control register to enable interrupts corrected to "J14"

9.1 SPI Operation and CS Pins

Amended to add detail and correct errors.

21.2.2 Pin Number errors

SDA4 and SCL4 swapped.
One-wire bus#1 (GC to select) is on I2C bus1, pin18 &17. Not bus2, pins 1,2.

21.2.3 Simple program example (11.4)

Delay command is L not D

22 Ordering Information

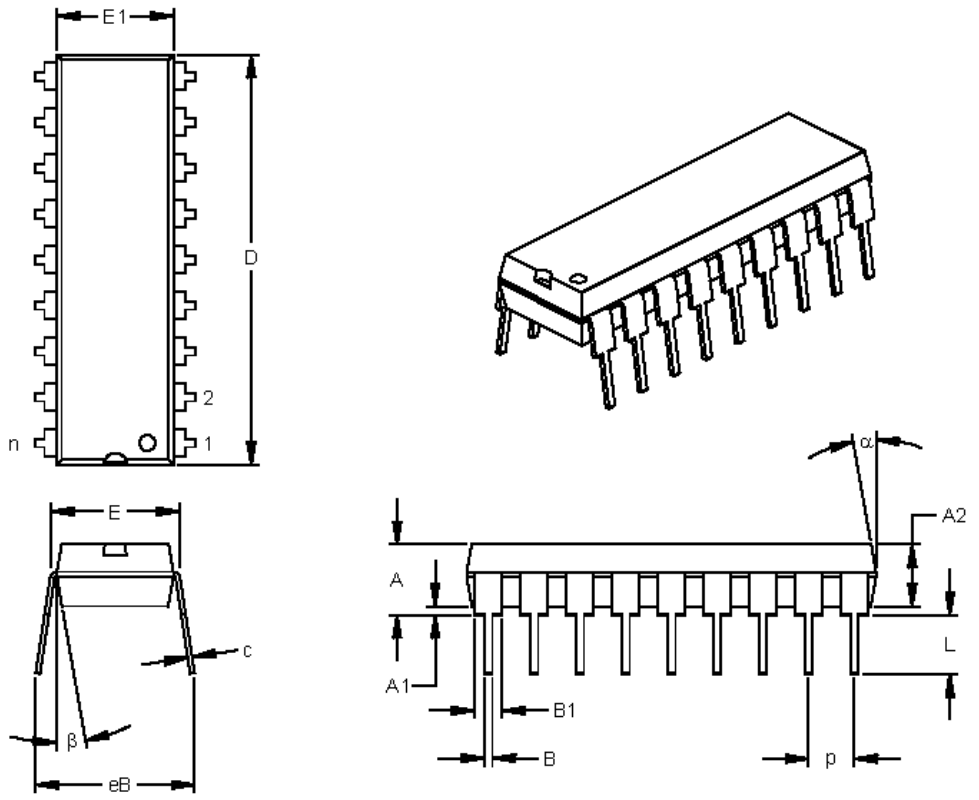
We recommend buying one built up I2C-2-PC adaptor to save time and hassles.
DIP parts are available in low volume, SOIC in medium volume, and SSOP20 and other packages are only available for volume orders at this time.
DIP18: BEL233-P
SO18: BEL233-SO
SSOP20: BEL233-SS

Low-voltage option is available for <4V operation.
EEProm settings can be customised at the factory.
BL232 has fewer functions and is slightly cheaper for high volume applications.

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

23 Package Dimensions

23.1 18-Lead Plastic Dual In-line (P) – 300 mil (PDIP)



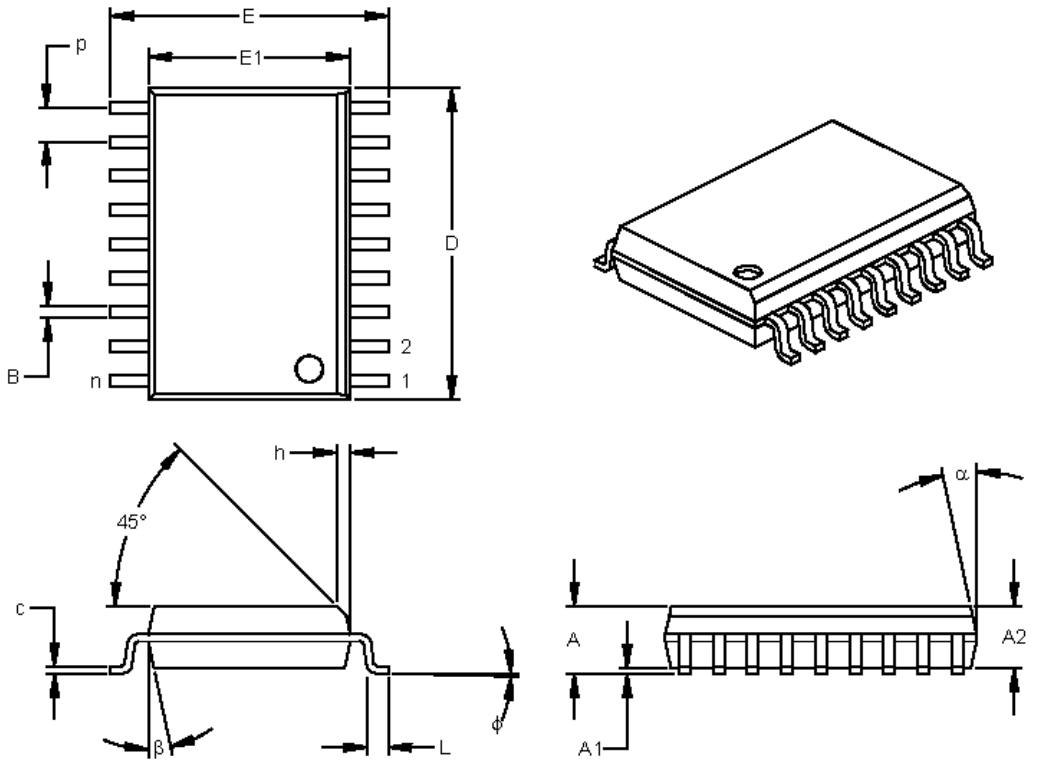
Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		18			18	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.155	.170	3.56	3.94	4.32
Molded Package Thickness	A2	.115	.130	.145	2.92	3.30	3.68
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.300	.313	.325	7.62	7.94	8.26
Molded Package Width	E1	.240	.250	.260	6.10	6.35	6.60
Overall Length	D	.890	.898	.905	22.61	22.80	22.99
Tip to Seating Plane	L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.045	.058	.070	1.14	1.46	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.310	.370	.430	7.87	9.40	10.92
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

*

Controlling Parameter
Notes:
.010" (0.254mm) per side.
JEDEC Equivalent: MS-001
Drawing No. C04-007
§ Significant Characteristic

ph +64 21 623-402 www.i2cchip.com sales@i2cchip.com

23.2 18-Lead Plastic Small Outline (SO) – Wide, 300 mil (SOIC)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		18			18	
Pitch	p		.050			1.27	
Overall Height	A	.093	.099	.104	2.36	2.50	2.64
Molded Package Thickness	A2	.088	.091	.094	2.24	2.31	2.39
Standoff §	A1	.004	.008	.012	0.10	0.20	0.30
Overall Width	E	.394	.407	.420	10.01	10.34	10.67
Molded Package Width	E1	.291	.295	.299	7.39	7.49	7.59
Overall Length	D	.446	.454	.462	11.33	11.53	11.73
Chamfer Distance	h	.010	.020	.029	0.25	0.50	0.74
Foot Length	L	.016	.033	.050	0.41	0.84	1.27
Foot Angle	ϕ	0	4	8	0	4	8
Lead Thickness	c	.009	.011	.012	0.23	0.27	0.30
Lead Width	B	.014	.017	.020	0.36	0.42	0.51
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter
Notes:
.010" (0.254mm) per side.
JEDEC Equivalent: MS-013
Drawing No. C04-051
§ Significant Characteristic

23.3 SSOP etc

Contact factory for availability of other packages.

24 Co-operation

We offer all customers a link page on our web site, where others can find out about ***your*** products. We encourage you to use this.