BL233B

RS232 – I2C/SPI/1Wire Adaptor & Controller

Features

- Versatile 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
- Automatic I2C Deadlock Recovery
- 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 from any computer and OS. Fully built RS232/USB adaptors are available. No DLL's or special libraries are needed to use it.

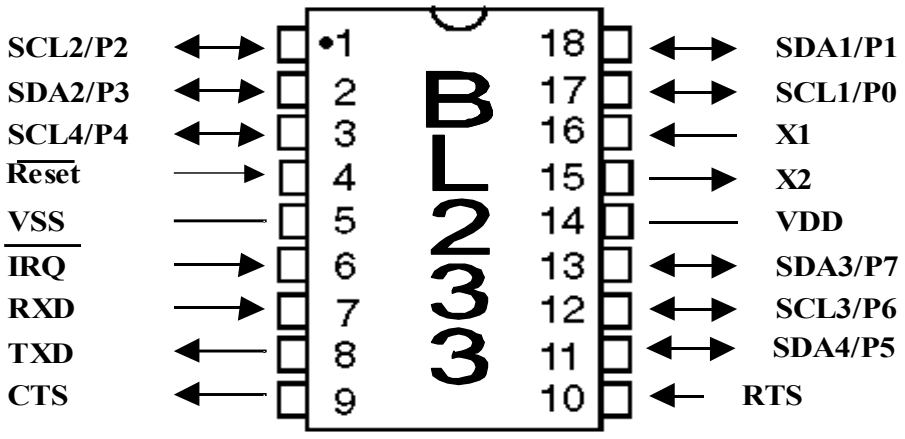


Figure 1: BL233 Pinout (DIP 18, SO-18)

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

BL233_C is available in DIP for testing from 2017. Customers should transition to BL233C during 2017.
Please check 26 Revision History for changes to this datasheet since you last read it.

Table of Contents

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	TTL	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)	

Unused pins should be pulled up.
See also 13.5 Special Pins mode to prevent accidental entry to SPM

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-Wire busses

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

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

3.1 Threshold Levels and Supply Voltage

ST/CMOS pins must be used with buses operating from the same voltage as the BL233. Ie if the BL233 is operating from 5V, the I2C bus must run from 5V. ST/CMOS pins have best noise immunity, and are the first choice, especially where cable lengths are longer. *Bus#1,2 are ST/CMOS*

TTL pins have low thresholds. They will work when the I2C bus is at a lower voltage than the BL233. Even if the I2C voltage is 1.8V, and BL233 is operating from 5V, the TTL pins will work. *Bus#3,4 are TTL*.
In the I2C-2-PC adaptor Bus#3 has a diode clamp to hold the pullup voltages down to the VDD voltage on the I2C connector, and space for an extra voltage regulator, so it is easy to use with any low voltage device.

3.2 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
To use fast mode timing, you select the fast bus numbers using the **G** command.
The timing of fast mode is *not* altered by the bit timings set using **J** command.
See section 3.5 Fast Mode below; section 5.6 Table 3: I2C Bus Pin Functions below

3.3 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 if the whole I2C byte/start/stop takes longer than 2^{18}/fXtal secs. (17ms @ 14.7MHz). A clock stretch timeout is flagged in the status register.
See BL233B errata: 24.1.3 Clock Stretch does not work on Stop pg 44 and 24.1.11 Short SCL Hi after Clock Stretch
See I2C Clock Stretch Probe[1]
[BL233_C can disable Timeout on clock stretch in extra control register]

3.4 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. Our I2C2PC adaptor is available in a galvanic isolated version.

3.5 Fast Mode

Busses can also be operated with Fast Mode (400k[4]) timing. This improves throughput somewhat when high baud rates are used.
To use fast mode timing, you select the fast bus numbers using the **G** command.
The timing of fast mode is *not* altered by the bit timings set using **J** command.

See section 5.6 Table 3: I2C Bus Pin Functions below for the fast bus numbers

3.6 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.

[3] http://www.nve.com/
[4] The pulse timing is designed to comply with the NXP requirements for Fast Mode. The actual clock frequency is about 200kHz at 14.76MHz fXtal.

4 Table 2: Command Character List

Char	Command		Chars to 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. *[BL233_C – Resume Char for Pause ":"]*	
<	Return		0	Execution returns from a macro, either to calling macro or RS232 buffer	
=	unused			*[BL233_C]*	
>	Jump to EE		2	Execution jumps to macro in EE	
?	Query Status		0	returns status register	6.2
@	unused			*[BL233_C]*	
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		2,4		6.1
K				[BL233_C – I2C EEProm Simple Read Write]	
L	Delay NNNN ms	Lnnnn	4		5.7
M	Set Message Number	Mnn	0,2		8
N	Nack			send nack during special reads	5.9.2
O	Output Direct to Pins		1-∞	write direct to pins and tri-state registers	9
P	I2C Stop	P	0		5.2
Q	Query Direct from Pins	Q	0	Reads pins directly	9
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	7
U	Dump EEProm	U	0	Dumps *whole* EEProm	12.6
V	Write EEProm	Vaadd …		Write eeprom from address *aa*	12.5
W	Write	Wdd…		Writes bytes to previous address (I2C) or SPI	5.5
X	Software Reset	5A		"X5A" forces a power on reset	16
Y	SPI Mode	n		Set SPI Mode. N sets bits to send for next write to do short writes.	10

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

Char	Command		Chars to Follow		
Z	sleep			*Not implemented in BL233B* *[BL233_C – Implemented]*	17
.	(fullstop)			echo EOL back for formatting	13.9
,	(comma)			echo comma back for formatting	13.9
	(space) ignored				
0x80 0xFF	Chars>=0x80 are used as ASCII			Special mode, enabled via fSerial	
!()[]#=@'/\+-*&$%				*[BL233_C – Stack Operations]*	

BL233_C Uses additional chars, please avoid them for future compatibilty. Space and all lower chars are unused.

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. Spaces and unrecognised chars are generally ignored[7]. Comma, Fullstop and the T command, can be used to format the returned data (sec 13.9 below).
Chars are generally processed and acted on immediately, except when the Pause Command for Timing Critical Sequences is used.
The basic form follows that shown in the NXP I2C documentation[2].
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 NXP 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 NXP documents.
To send a string of 80/81/80..… to the port:
S 40 80 81 80 81 P
Repeated Start:
S 40 80 81 S 82 S83 P
see errata: 24.1.12 Repeated start does not meet Tsu:sta specification

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] 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 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.

When an I2C bus *STOP* fails because SDA remains low, BL233B now sends up to 8 extra SCL pulses to try and clear it automatically. See 19 I2C Bus Deadlock to understand this.

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][8]

Repeated Start:
S407D S4101P
7D[eol]
[start][write 0x7D][repeated start][read 1 byte][stop] [adaptor returns 1 byte]

You can suppress the automatic EOL after every read, and format the data (sec 13.9 below).

5.3.1 Reading more than 255 bytes.

You can read more than 255 bytes in a single block if you need. Each time you send the byte count, another read block will be performed, ***but*** NACK is sent after each read block and most slaves will stop sending data after the NACK. You need to use the manual NACK mode.
See: Reading more than 255 bytes / Read without NACK section 5.9.2 below.
eg Read 256 bytes from 0x40:
J8C S41 FF N01 P J88
Read 1024 bytes:
J8C S41 FF FF FF FF N04 P J88

5.3.2 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]

[8] [EOL] is the End of Line char. It defaults to LF (0x0A).

5.4 10 Bit Addresses

Carefully study the NXP 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: Concise Read/Write Commands

Where you want to read or write to the previous address use R and W[9]. 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).

Table 3: I2C Bus Pin Functions

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
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									Unused,[BL233_C Bus2 SDA]
C	18			17					1Wire#1 (Bus1 pins)
D	3								1Wire#2 (CS2 pin)
E	13			12					1Wire#3 (on Bus 3 pins)
F	11								1Wire#4 (CS3 pin)

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

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

5.7 Delaying

The de**L**ay command puts a delay in execution. You should always use this in preference to trying to delay commands being sent by you software.
To output a pulse we can use Delay (**L**). Delay is up to 65535 milliseconds[10].

```
S4001 L0400 W00
```

[Set bit0 of PCF8574 HI][Delay 0x400ms ie ~1 sec][Set bit 0 LO]
The delay timer runs continuously, so will have jitter. A delay of 4ms can be 4-5ms.s
Note that this delay can only be used between separate commands, not within a single command. [BL233_C allows **L** inside other operations e.g. write, read, typeback. Jitter is fixed]

5.7.1 Delaying WITHIN a single write/read sequence

There are a few rare reasons you might need to delay between the bytes of a single write sequence. e.g. You have a software/microprocessor slave which does not have (working) SCL stretch capability, and needs sometime between bytes.

- Short Delays: you can pad commands with any invalid character e.g. any lowercase character or space. BL233B will ignore them. e.g. the command S40xxxx55P will take 1ms (6chars@57600) between writing the 40 and the 55.
- Long delays can use the SPI "Y" command to do I2C write without starts. (See 10.6 Writing/Reading I2C without Start and Address)

5.8 Pause Command for Timing Critical Sequences

'**:**' (colon) pauses execution until an EOL char (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 after '**:**' and including EOL must fit in the RX buffer, before handshaking stops the flow). This is 32 chars in BL233B.
[BL233C has larger buffer, and adds the resume char ";"]

An alternative for timing critical sequences is to put them in the EEPROM as a macro.
Note that once the Tx buffer (80 chars) is full, read operations will only proceed as the buffer is emptied.

5.9 I2C Acknowledge

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

[10] Timer tick rate can be changed in EEProm

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
- 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 on writes. 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.
When reading, the last byte is automatically terminated.

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. If 0, Writes halt when a nack occurs. Subsequent bytes are ignored. Writes "N" back to host
2	0	RdBlockAck	nack won't be sent automatically when last byte is read , **N** command must be used to manually nack last byte read
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 more than 255 bytes / Read without NACK

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. Slaves usually stop returning data at this point.
The BL233 normally 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 (eg 2 byte words for formatting), without signalling end-of-data to the slave.

In this case you can set the flag *fControlReadBlockAck*. In this case *all* reads will be ack'd, and you must take care to nack the last read yourself with an N command before the final read. Note that you can read multiple blocks by sending a series of byte counts.
Don't forget to change *fControlReadBlockAck* back when you are done. You will probably want to use *SuppressEOL* and formatting chars "," and "." as well.

"**N**" makes the last read NOT ack, thus signalling the end to the slave. Eg to read 512 bytes from an 8574:
J8C S41 FF FF N02 P J88
[Setup Control register, read 255 byte blocks with ack 2 times. Read 2 bytes, N makes last byte nack to end transfer, Restore control register]

Read 256 bytes from 8574, and restore control register when done:
J8C S41 FF N01 P J88
Read 1024 bytes:
J8C S41 FF FF FF FF N04 P J88
Read 256 bytes in 64 byte lines:
J8C S41 40.40.40.N40 P J88

If fControl*ReadBlockAck* is set, then you *must* use the **N** command, followed by another read (eg **N01**). Otherwise it may be impossible to send a STO**P.** This will happen if the first bit of the next byte is 0. In this case the slave will be asserting SDA=0, and the master will be unable to send stop.

Note that the BL233B Automatic STOP Recovery will send extra clock pulses to force a stop onto the bus regardless. However this is slow, and is doing extra operations to the slave (depending on the data).

Note that when you have missed out nack, the slave may not return data for the first byte, when read next.

5.9.3 Retry until ACK

BL233_C Can retry Start+Address until slave acknowledges. This is used by eeproms when busy writing, and some processor based slaves.

6 Status and Control Registers

fControl (write with J)			
Bit	Default	Name	Description
7	0	SuppressEOL	suppress EOL following data reads
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	ShowAck	returns "K" or "N" as each byte written to I2C is/not acknowledged by the slave

0	0	MessageNumsEnabled	enables return of Message Numbers

fStatus (read with ?)				
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	[BL233_C – Used with I2C EEProms writes]
2	*		FifoOverflow	Either RX or TX fifo overflowed
1	*		ClkStretchTimeout	I2C timed out waiting for SCL to go Hi. [BL233_C Also used for RetryAddressUntilAck timeout]
0			Nack	result of last I2C Nack

6.1 Setting Control Register and Bus Timing

To set the control register only.
J [control register value]

The default at power on is:
J08

To change the I2C Bus timing. 0 is fastest, 0xFF (255) is very slow
J [control register value] [I2C Timing value]

e.g. Write only fControl to suppress the automatic EOL char after every read [we recommend using J88, and using "." to put linefeeds where you want them]
J88

To suppress EOL *and* set I2C bit time to 15,
J880F

Write only fControl to enable interrupts (and WriteIgnoreNack)
J18
To Enable Ints *and* set I2Ctiming to 15
J180F

Notes: You cannot read back the control register or timing values. The default Control register value can be set directly in eeprom see Table 5: EEProm Memory Map. Bit Timing can only be permanently set by putting the **J** command into the startup macro.(see Macros)

6.1.1 Bus Timing Value

Bit/Bus timing is set by **J** command. The Timing value is 0 – 0xFF. For Normal I2C buses each Timing count is approx 1.6µs. Experiment to see what the timing is.

* Cleared by reading status register

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.

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.

Remember that timing is the byte after the control register value eg to set timing to 15 (ie 0x0F) send **`J 08 0F`**
See errata: 24.1.11 Short SCL Hi after Clock Stretch
[BL233_C adds a second Control2 register. J command can take 3 bytes J[Control][Delay][Control2]]

6.1.2 *Table of Bit Times*

Bit Time of I2C Bus			
Timing Value (Dec)	***Timing Value (Hex)***	***Normal I2C Period (T_{HI})***	***Fast I2C***
default	00	9.2µs (3.7)	5.5µs
1	01	9.6µs (2.9)	5.5µs
2	02	11.2µs (3.7)	5.5µs
3	03	12.8µs (4.6)	5.5µs
4	04	14µs (5.4)	5.5µs
8	08	21µs (8.6)	5.5µs
16	10	34µs	5.5µs
32	20	61µs	5.5µs
64	40	122µs	5.5µs
128	80	226µs	5.5µs
255	FF	424µs (212)	5.5µs

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

[errata: IRQ low during reset / power on appears to stop BL233 working, BL233_C – fixed]

- 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 12 for more details.

7 Typeback Characters and Strings

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

```
S407D T563D R01
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: EOL, Separator Chars, and Data Formatting, 13.9below)
[BL233_C – "Quoted Strings" are allowed, not just hex]

The default start-up welcome "Hi I2Cad VXXX" message works this way. When you receive your adaptor it has a start-up macro in eeprom, that contains a **T** string to type the message.

	Hex	Hi Char
Space	1 0	A0
Quote	2 2	A2
Tab	0 9	89
CR	0 D	8D
0x	3 0 7 8	B0 F8

7.1 Typeback Synchronising Chars

If you use the typeback chars for synchronising the data, then 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.

7.2 Long Strings

[BL233_C – "Quoted Strings" are allowed, not just hex]
Each char typed back, requires 2 chars to be sent to the adaptor. If this is slowing your application down you can do two things.

Store the **T** sequence in the EEProm. This will reduce a long sequence to 3 chars. Try running the startup macro by typing
`>00`
The adaptor replies with the welcome message:
HI I2C v118

Use **HiCharsAsAscii.** See HiCharsAsAscii,sec13.1.1,below
Add 0x80 or set bit 7 of any ascii char you want to send. Now you only send 1 char for each char typed back. Note that this also works inside macros.

8 Message Numbers

[If you use Message Numbers, please email sales@i2cchip.com and tell us]
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 R01 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 zero 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.
After sending **M** the only way to disable message numbers is by writing the control register.
The message number only increments when it is enabled. You can turn it on for 1 read, then off for 5 reads, then on for 1 read. It will only be incremented each time it is output, not for the reads where it was disabled.

9 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.

9.1 Reading Pins

'**Q**' *queries* the actual pin state. In the I2C-2-PC adaptor there are pullup resistors on the pins so they idle high.
One use for **Q** is checking for bus faults and the deadlock condition, after an *I2C Stop*.
The INT pin can be read using the **?** status command.

9.2 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.

9.2.1 Pin Timing

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

10 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 sec 9 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]

10.1 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 I2C2PC to use CS#2,3 as outputs. (only need to do this once)
P O 30 CF
[I2C Stop] [Set Bus2,3 CS to outputs 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]

10.1.1 CS Pins and commands for I2C-2-PC

The I2C-2-PC uses Buses 2 & 3 for SPI transfers if the jumpers are changed to the CS position. This makes pin 5 of the connector CS.
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 10	
CS on Bus #3 Only	11	5	DF	O 20 DF	O 20	
CS on Bus #2 & #3			CF	O 30 CF	O 30	O 00

So if you are using I2C2PC and want to use CS 2,3 then use these commands to initialise and control the CS pins.

	CS2	*CS3*	*Command*
Initialise CS pins	1	1	`O03CF`
Change CS	0	0	`O00`
Change CS	1	0	`O10`
Change CS	0	1	`O20`
Change CS	1	1	`O30`

Note that *any* unused lines can operate as additional CS pins. This means unused SDA and SCL pins. For example Bus 2 might be used for SDA, SCL, CS1, while Bus 3 is used as CS2, CS3, CS4

10.2 The Data Pins SDA vs MOSI / MISO

I2C uses a common data pin for in and out (SDA). Many SPI chips have separate pins, often called MISO (master in, slave out) and MOSI (master out, slave in).

Even though there are separate in and out pins, many SPI devices have tri-state outputs on the MISO pin. These are only enabled during that part of the transfer when data is being read out of the slave. In this case, the MISO and MOSI pins can simply be connected together, and connected to the SDA pin of the BL233. (It is obvious that spi chips have tristate outputs, or else only a single slave could connect to the MISO line)

However some devices really do require separate pins. In this case you use the half-split bus. (command G9 or GA). This gives you separate SDA_In (MISO) and SDA_Out (MOSI) pins.

If you are using the I2C-2-PC then these are the split data pins that are possible:

Bus#	*Pin#*	*Levels*	*G8*	*G9*	*GA*
Bus 2 SDA	1		SDA Out MISO	SDA Out MOSI	CS2
Bus 2 SCL	4		SCL	SCL	CS3
Bus 2 CS/IRQ	5	CMOS	CS1	SDA In/	CS1
Bus 3 SDA	1	TTL	SDA In MOSI	CS2	SDA Out MOSI
Bus 3 SCL	4		unused	CS3	SCL
Bus 3 CS/IRQ	5		CS2	CS1	SDA In MISO

10.3 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]

10.4 Bidirectional (simultaneous) 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.

Normally you would be using one of the split buses for this.

10.5 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.

10.6 Writing/Reading I2C without Start and Address

The I2C commands, implicitly use / require the I2C start and addressing. You might need to force some timing gaps into an I2C transaction (Error: Reference source not found Error: Reference source not found, or you might have some bastardised-almost-I2C protocol like Sensirion SHT7X use.
You can do an isolated 9 bit I2C write, without the start using the **Y** command

To write the byte 55.

YW55Y1FF
(the "Y1FF" just clocks the ACK bit)

To read you will read a byte and send the ACK (0) for all bytes except the last one. So to read 3 bytes you would send
YR01Y1W00 YR01Y1W00 YR01Y1WFF P
"Y100" sends ACK, and "Y1FF" sends NACK for the last byte only.

10.7 M5451 LED/LCD Driver & I2CCHIP Display Modules

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.

11 1Wire Bus

The BL233 has support for Dallas/Maxim 1 Wire bus, using the standard timing. 1Wire devices can be used in either unaddressed mode, with a single device per bus, or addressed by the RomID, with multiple devices on a single bus.

There are two places where 1Wire is an especially useful addition to systems:

- Unique serial numbers are lasered into all 1Wire devices, and make an easy ID device.
- DS1820 temperature sensors are commonly available as packaged, wired, sealed sensors.

Commands:

- 1Wire buses are selected by bus "G" command (GC-GF), followed by a Start.
- **S**tart sends 1Wire Reset, and samples the presence and interrupt responses into the fStatus register
- **W**rite and **R**ead 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.

11.1 Reset, Start and Stop

The 1Wire bus begins transactions with a Reset. The Start command (S) is used to send this. Unlike I2C, there is no Stop command required

11.2 Reading Rom ID or DS2401 Serial Number

The 8 byte RomID, contains a 48 bit unique serial number, 8 bit CRC, and 8 bit device type. To read it send:

GD S W33 R08
[Select 1 wire bus#1 (bus #C)][Start(reset)][ReadRom Command][Read 8 bytes of rom data]
Return data is
28FF20A314140092
[Chip (28=DS1820, 01=DS2401)], [48 bit Serial Number (12 chars)], [CRC 8 bit]
In this example the 48bit serial number is FF20A3141400

11.3 Checking Presence and 1Wire Interrupts

The 1Wire Reset (Start) checks for interrupts[11], and presence of any chips. Read the fStatus ("?") to check these flags after sending the 1Wire Reset. If you are using an iButton application where it is possible to short the bus, you can directly query the bus pin with the "Q" command, if 1WireInterrupt and 1WirePresence are both true. Eg send:
S?

fStatus (read)				
Bit		Default	Name	Description
6			1WirePresence	if a presence pulse exists during reset
5			1WireInterrupt	A device signals INT during reset pulse

Checking the CRC value of any RomID, or comparing multiple reads will ensure that your are reading valid data.

11.4 Reading DS1820 Sensor

Reading DS1820 Temperature Sensor
The sensor defaults to 12bit resolution, and so nothing need be written to it. Only the two temperature bytes need to be read.

:S W CC 44 L0400 S W CC BE R02[LF]
[Reset][SkipRom][Convert Temp] [wait for conversion] [Reset][SkipRom][Read Temp 2 bytes]

11.5 Addressing with Rom IDs

If you have multiple devices on the bus then you need to know the RomID of each device. You can find this, by reading it individually (ie when there is only one device on the bus at a time). Once you have the 16 Hex character RomID, replace the SkipRom command 'CC' above with the match rom command '55' and the Rom ID, eg
SW5528FFB6381514006744L0400SW5528FFB63815140067BER02

11.6 1Wire Buses and 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.

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

The "G" command selects 1Wire buses. Note that is you get a pre-assembled 1Wire device from I2CCHIP, it will probably be wired to the CS/INT pin 5, and so you would use it with bus D or bus E.

Table 4: 1Wire Bus Pins

Bus #	SDA	I2C2PC Pin	SDAOut	I2C2PC Pin	
B		[BL233_C adds Bus2 SDA]			
C	18	Bus1 SDA	17	Bus1 SCL	1Wire#1
D	3	Bus2 CS			1Wire#2
E	13	Bus3 SDA	12	Bus3 SCL	1Wire#3
F	11	Bus3 CS			1Wire#4

11.7 Software

Realterm has some basic 1Wire commands on the I2CMisc tab.

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.

12 EEProm Memory

[See application note: Bl233B_EEProm_Programming.pdf]

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.

12.1 Table 5: EEProm Memory Map

Address	~Add	Register	Description	see
0xF7	08	fSerial	RS232 Control Register	13.3
0xF8	07	Baud_Div	Baud rate divisor reset value	13.4
0xF9	06	TimerDivL	***Low*** byte of timer divisor (TimerDiv)	14.1
0xFA	05	TimerDivH	***Hi*** byte of timer divisor	14.1
0xFB	04	fControl	Control Register	6
0xFC	03		*Internal use only* *[BL233C - fControl2]*	
0xFD	02		*Internal use only*	
0xFE	01	Watchdog Vector	Macro address for watchdog	15
0xFF	00	IRQ1Vector	Macro address for interrupt	6.3
0x00	FF	Base / POR	First macro instruction. Power-On jumps to this location.	

Address	~Add	Register	Description	see
0x76	89	end	Generally contains a return char "<"	

12.2 Macros

Macros are stored command chars.

- Commands to execute at power on (see 16)
- 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

Note: If the Power-On macro is bad, then the BL233 can become unusable. Use Special Pins Mode to regain control and correct the bad macro.

The IRQ and Watchdog vectors are binary values that point at the first char of the macro. Obviously these are in the range 0-0x76

12.3 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]

12.4 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[12], that captures it with a terminal program[13].
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]

12.5 Writing the EEProm

BL233B ignores characters during each 10ms eeprom write. Failure to time correctly will result in corrupted eeprom. See notes below [BL233_C – fixed, send chars during ee writes].

[12] You could use a TI or HP graphing calculator too.
[13] REALTERM.EXE from http://realterm.sourceforge.net or http://www.i2cchip.com/realterm

[See application note: Bl233B_EEProm_Programming.pdf]
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
```

Most settings will only change after a reset.

Each byte written takes 10ms. During the write period, the BL233B will ignore or lose any characters sent to it. [BL233_C – fixed]
To write more than a single byte you need to either

- Send the whole eeprom configuration with character padding of 10ms. Put programming commands and data in a file, and use Realterm's "Send File" function and set the character delay to 10ms. This is an easy way for production configuration. You will have no issues with buffer size etc.
- Use the ":" command before "V" to pause execution until the whole eeprom write string is in the buffer. After the LF character there must be a pause of 10ms per byte written. This method is easy from the keyboard. (In Realterm <ctrl>+ENTER sends LF

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, or an invalid start-up macro is loaded, you will be unable to regain control after the next reset. *Special Pins Mode 13.5 below* exists to regain control.

(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.)

12.5.1 Write Protecting the EEProm

If fSerialEEWriteProtect is set, further writes to EEProm are blocked.
BL233B: for ever – see errata 24.1.8; [BL233_C unless in Special Pins Mode 13.5 below]

12.6 Dumping the EEProm

U will dump all 128 bytes of EEProm in the order shown in Table 5: EEProm Memory Map. ie the first byte is fSerial, not Base/POR

13 Serial Comms

[See also application note: BL233B_Setting_Baud_Rate.pdf]
Note: Changes to fSerial and Baud Rate will only take effect after a reset, either power-on or software.

Default is **57,600 Baud N81** with a **14.7456MHz** crystal. RTS/CTS handshaking is recommended.

13.1 Data Format

- 8 bits
- No parity
- 1 stop bit

Data format cannot be changed.

13.1.1 HiCharsAsAscii

The BL233 uses two hex digits to represent each single data byte. Sometimes, such as when sending long ASCII strings to displays, this is a significant overhead.
When the *HiCharsAsAscii* flag in fSerial is set, any chars with bit 7 set, will be used as ascii chars with bit 7 clear.

Eg to send the string "Hi" using the T command
`T[0xC8][0xC9]`
ie oxC8=0x48+0x80. This is 3 chars, instead of the normal 5 chars, so long strings will be much shorter.
`T4849`

This is enabled by default in the Bl233B.

13.2 Handshaking

RTS/CTS hardware handshaking is supported. CTS output controls the flow *to the BL233* , and will always be output, regardless of *UseRTS.*
RTS input controls sending *from the BL233.* It is controlled by bit fSerial_UseRTS. RTS is ignored by default: (See errata)

CTS/XOFF is asserted when the buffer contains 32 chars, and CTS is released / XOFF when buffer has <16 chars.

13.2.1 Testing Handshake

To test you can send this command string:
`T546573742048616E647368616B696E673F L0100 T2E2E2E2E2E2E48616E647368616B65207465737473204F4B`
OK: `Test Handshaking?.......Handshake tests OK`
Fail: `Test Handshaking?`

13.2.2 XON / XOFF

BL233_C Significantly improves XON/XOFF and should be used instead

XON/XOFF is available to supplement CTS for controlling the flow of data *from* the PC. When XON/XOFF is enabled, CTS is still asserted. Sending of the XON/XOFF chars will always ignore the RTS pin.
The BL233 ignores XON/XOFF characters – they are only used to control the flow of data *from* the PC.

It is recommended that you *only* use XON/XOFF where handshaking lines are not available. It has extra overhead and is not as reliable a technique as RTS/CTS. When using XON/XOFF, it is possible for the flow to lock up if an XON is lost, or if the BL233's RX buffer is overflowed. See Watchdog Examples 15.2
The sending device has 16chars time to respond to XOFF before the buffer overflows. Where software is implementing XON/XOFF, this may be too short. WR703 Routers running OpenWRT tested at a max of 20chars response time at 57,600. The buffer overflows, and BL233 locks up. To avoid this baud rate must be reduced.

13.3 Table 6: Serial Initalisation Registers in EEProm

	bit 7	bit 6	bit5	bit4	bit 3	bit 2	bit 1	bit 0	default
fSerial (F708)	EEWr Prot.	Enable Sleep	HiChars AsAscii			UseXon	UseRTS	Baud Rate HI	x21
BaudDiv (F807)	Baud Rate Divisor								x0F

Unused bits should be 0. [BL233_C uses unused bits]

13.4 Baud Rate

*Default is **57,600 Baud N81** with a **14.7456MHz** crystal. RTS/CTS handshaking is recommended.*

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

13.5 Special Pins Mode and Baud Rate Changing Problems

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, and the start-up macro is not executed

Special Pins Mode of BL233A,B should be entered by *either*

- IRQ held low during power on, *or*
- SDA1,SCL1,SDA2,SCL2 are all held down during power on

This is contrary to earlier descriptions of SPM which said "force Pins 1,2,17,18, and 6 to 0V, during and 500ms after reset / power on"
Now you can change the EEProm values and fix the problem.
[BL233_C - IRQ,SDA1,SCL1,SDA2,SCL2 are all held down during power on. All critical eeprom values are ignored in SPM]

If you leave unused pins floating, the the part may be in special pins mode during start up. If this happens the baud rate will remain at default, even though you have sucessfully changed the eeprom values. Unused pins should be pulled up.

13.6 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.

13.6.1 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.

13.6.2 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.

13.7 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, in any timing critical sequence.
BL233-B

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

BL233-C

- RXD Buffer: 80 Chars *from PC*

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

13.8 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.

13.9 EOL, Separator Chars, and Data Formatting

EOL (LF 0x0A by default) is sent after any data to separate commands. eg

```
S41 R01 SA1 R02 P L0100 S41 R01 S42 R02 P
01[eol]
        FE02[eol]
                01[eol]
                        FE02[eol]
```

This is convenient for program control, but may be awkward where you want to capture files with tables of data. For example Excel will find it much easier to read comma delimited lines of data.

fControlSuppressEOL allows you to suppress the automatic EOL character.
Two shortcuts are available: Comma is echoed back and fullstop returns EOL.

```
J88
[set fControl to suppress EOL char]
S41 R01, SA1 R02 P. L0100 S41 R01, S42 R02 P.
01,FE02[eol]
01,FE02[eol]
```

You can explicitly insert any char or string using the **T** command. These can include control chars. eg TAB, CR etc. The T command is followed by the hex value of the characters you want to send.

```
J88
[set Control to suppress EOL char]
T566F6C747320 R02, T20416D707320 R01.
Volts 5A5A, Amps 5A[eol]
```

The unix/mac utility TR (available on the PC in UnixUtils package) is useful for translating formatting chars in files eg commas to spaces, LF to CR etc.
fControlSuppressEOL will affect all operations except d**U**mp, which always has an EOL.

14 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

14.1 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[14]. ***Setting the Timer eeprom value to less than 0x0030 will fatally lock up the BL233B. Special Pins mode will not recover from this.*** [BL233_C fixed]
The TimerTimedOut bit is normally set. [BL233_C: I2C EEProm support WriteI2CAddWaitLoop waits until eeprom memory comes back online after a write cycle. TimerTimedOut will be clear if it exited normally, and set if timedout (write)]

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

Timer Rates @ 14.7456MHz						
Timer Div	0x0E66	0x10CC	0x3840	0x4800	0x9000	0xFFFF
Timer	1ms	2ms	256Hz[15]	5ms	10ms	17.8ms
Watchdog	66s	2m 11s	4m 15s	5m 28s	10m 55s	19m 25s

14.2 1-Wire Timing

1-Wire bus timing cannot be adjusted and will scale with the clock frequency.

15 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.
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 fControl register bit **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. *Errata: BL233B does not unblock pause ":" sequences.*

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

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", and will only fire once.
To make the watchdog repeat, the watchdog macro must include a J command to re-enable the watchdog. Eg “J28”

15.1 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 flow of chars stops.
- Send XON character in case XON/XOFF handshaking gets locked up.
- Wake up wifi router from sleep,

15.2 Examples

“**T11J28<**” Types XON and renables watchdog. Used to clear lockups in XON/XOFF handshaking.

“**T77J28<**” Types “w” and re-enables watchdog.

16 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.

16.1 Low Voltage Reset

BL233B no longer has low voltage reset, and will operate to <3V.
4V Reset is still available as a special factory option if required.

16.2 Power On Reset Macro

At power on or reset execution begins at 0 in the eeprom. The factory default macro types "HI I2C v118" using the **T** command.
Caution: If you write a macro that has an endless loop as the POR/Reset macro, you may be unable to recover the chip. Special Pins Mode can be used to regain control.

17 Sleep Mode

not implemented in BL233B.
See 25 BL233_C
Due to the perils of putting the part to sleep this function is not enabled.
If *fSerial.EnableSleep* is set, **Z** will put the part to sleep. (by default sleep is disabled).
There is Powerdown Sleep and WaitForString sleep.

17.1 Pause vs Sleep

When the Sleep bit is set, the clock oscillator is stopped and the BL233 current consumption drops.
The BL233 is unable to receive any further chars, and you could expect chars to be garbled if they occur during transition times.
Execution will resume after some delay depending on how the part exits Sleep.
During pause command execution is paused, but there is no change in current consumption or oscillator. The part will exit pause immediately. Timing during pause is accurate.

18 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.

19 I2C Bus Deadlock

A known issue with the I2C Bus is the possibility of a deadlock. SMBUS has addressed this with a requirement for an automatic timeout. However the possible causes of bus deadlocks are seldom explained.
I2C relies on the two special conditions sent by the master: *START* and *STOP*, where SDA changes state while SCL is high.
If (for some reason) a slave holds SDA low, then the master is unable to send the *STOP* condition, and the bus state cannot be reset. In some situations the bus can remain busy indefinitely, in others there will be an indeterminate number of faulty operations performed until the bus comes free.

We know of two situations where this can arise. Both involve reading a slave, and the slave outputting a 0 data bit.

19.1 Added / Dropped Clock Pulses

In the first case a clock pulse is added or dropped, (eg by noise) during a transfer. Now we find that when the master thinks that it is at the end of a byte, the slave is still outputting a data bit. If this data bit is a 0, then SDA is held low, and *STOP* cannot be sent.

This situation could arise either from noise, or perhaps from the master software being interrupted part way through a transfer.

19.2 Failure to send NACK after last read

In the second case a programming error is too blame. The master does not send NACK (ie leave the NACK bit=1) at the end of a byte read from a slave. At the end of the clock pulse for the NACK bit, the slave will begin outputting the next data bit. If this is 0, then the master will be unable to raise SDA.

Note that both these are dependent on the data in the slave. If the slave does not have a 0 in the correct bit, then there will be no problem. This helps to make the problem intermittant.
Note that normally the BL233 takes care of sending the NACK correctly. Only if you override the NACK control bit will you have to watch this.

19.3 BL233B Automatic STOP Recovery

The BL233B checks SDA after a *STOP*. If SDA is still low, then it will clock the SCL line up to 9 more times to try and clear the deadlock.

[Where there is excessive capacitance on the SDA line, then SDA can be still low when tested, and extra clocks will be sent. Slow the bit timing to fix this]

19.4 Demonstrating the Problem

Connect a PCF8574 to I2Cbus1

19.4.1 Extra Clock Pulse

```
J0C
```

[set mode to not send ack after reads. this gets us halfway through a read operation]

```
:S4000S4101
```

[write all 0's to a PCF8574. Now read back 1 byte. BL233 sends ack, so slave expects another read to follow]

```
:O 00FE 00FF 00FE
```

[this sends an extraneous clock pulse, reading the next data bit (a 0) from the 8574]

```
P
```

[STOP fails because 8574 is holding SDA low. On BL233B, 8 extra clock pulses will be seen]

19.4.2 Failure to send NACK

```
J0C
```

[set mode to not send ack after reads. this gets us halfway through a read operation]

```
:S4000S4101
```

[write all 0's to a PCF8574. Now read back 1 byte. BL233 sends ack, so slave expects another read to follow. At this point 8574 is outputting 1st data bit (0)]

```
P
```

[STOP fails because 8574 is holding SDA low. On BL233B, 8 extra clock pulses will be seen]

19.5 Manually Checking SDA and SCL

Software can check for this condition (and any other bus faults) 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. With the BL233A, software could use the **O** command to explicitly toggle SCL 8 times to clear the problem.

20 Formatting Data

Note: This section applies to BL233B not BL233A
[See also application note: BL233_Data_Formatting.pdf]
The default behaviour for the BL233B is to return each data read followed by a linefeed (ox0A). This is the same as the BL233A. This is fine for direct reading of each value as it is returned.
Frequently, for example when datalogging, you would like the data to be formatted into lines or have frame-sync characters etc.

- "J88" command sets the Suppress EOL bit in control register. Now you have to format the returned data yourself
- "," returns a comma in the data stream
- "." returns a linefeed
- "T" command lets you type any character you like back

With these it is very easy to format your data into comma separated value CSV form
End of Line Characters:
BL233B uses [LF] (linefeed) as the end of line. "." types [LF] back
This is common on Unix. Note that Windows Notepad will not correctly handle LF. Don't use it.

Mac uses [CR]. "T0D" will return a carriage return
PC often uses [CRLF] "T0D." will return [CR LF]

An alternative is to use a utility such as TR or SED to translate the [LF] into whatever sequence you want.

Lets say that the command sequence "R01" returns some data
R01,R01.
5A,5B[LF]

You might want to insert the time into the file as you collect data. So we put a placeholder character T where we want the time to go.

You can use the "T" command to type back a header line if you want to load your data into a spreadsheet. However this may not be the most efficient method.

Reading hex in spreadsheets.
OpenOffice:Calc uses the "DECIMAL" function to convert

21 Getting Started

It is best to get the "I2C Starter Kit". This has a built up I2C-2-PC adaptor, cables and connectors and a simple I/O Slave. This will save you time, and it is fully tested!.

Baud Rate is ***57600 baud, N81, RTS/CTS*** handshaking recommended.
Crystal should be 14.7456 Mhz
Unused pins should be pulled up, and *not* left floating.
Serial data pins must be inverted by a driver chip such as MAX202 before you connect to RS232 cables.
Serial data pins can be directly connected to microprocessor UART pins.
When you power-up or rest the BL233 it sends its string "`HI I2C v118`"
When you send a "`?`" it will reply with 2 chars eg "`18`"

Install Realterm on a windows machine. Realterm has special tabs for sending commands to the BL233. These buttons send the commands to the BL233 as plain ASCII text. You will see all the commands sent on the terminal screen and all the replies from the BL233 in a different color. Remember: there is no special api, and no special magic in Realterm, it is only sending the simple ASCII that you see on the screen.

You can use Realterm to send your commands to the BL233. Once you are able to control your chip and you know what the returned data is like, then you are ready to begin programming your application. It is very easy and quick to get your application going when you know all the commands work, and you have samples of all the returned data.

22 Example Applications

22.1 I2C2PC: 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.
Galvanic isolation can be installed where the I2C busses need to be isolated from the computer.
There is an internal 5V and 3.3V 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.
A Bluetooth interface is available for Wireless adaptors
.

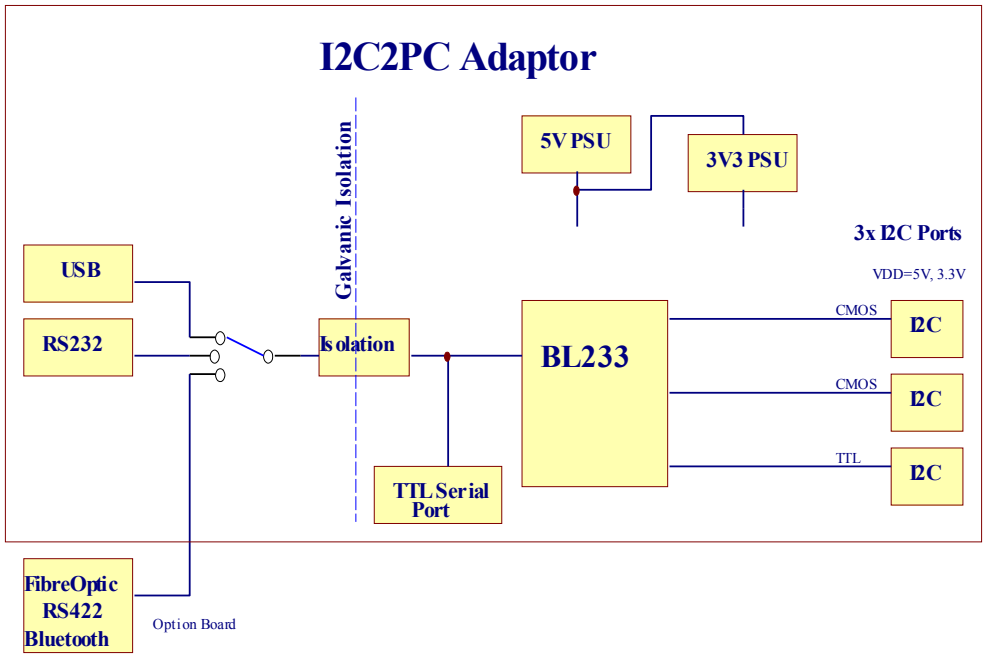


Figure 2:

22.2 Simple I2C to RS232 Adaptor

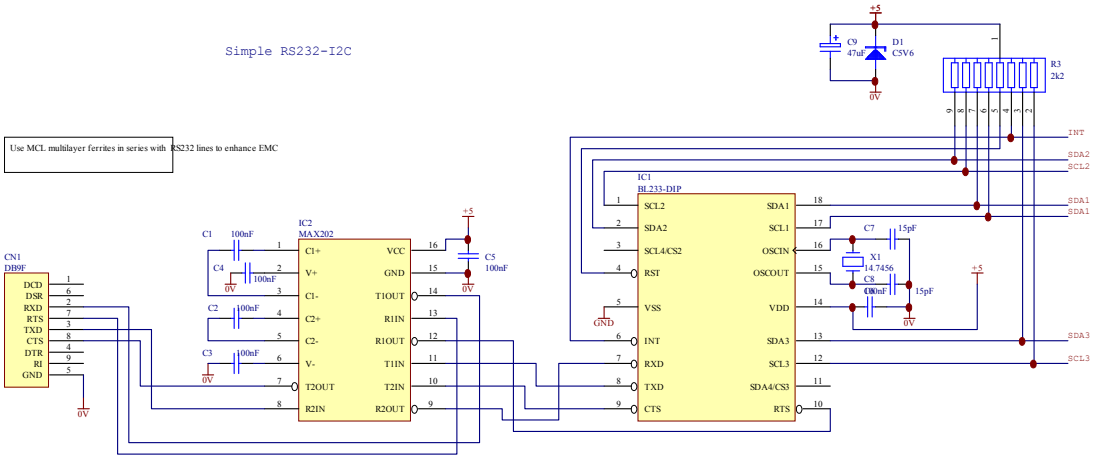


Figure 3

22.3 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.

22.3.1 24C01

128 Byte EEProm.

```
SA0 00
R40 R40 P
```

22.3.2 24C02

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

22.3.3 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
```

22.3.4 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
```

22.3.5 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….

22.4 Simple Data logger with no PC Software

Use a NXP 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.

22.5 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

22.6 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.

22.7 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

22.8 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.

22.8.1 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.

22.8.2 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.

22.8.3 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]

22.9 Temperature Logger using DS75/LM75/TMP101[16]

To read the 2 byte temperature:
```
S 91 02 P
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
```

23 Migration to BL233_B

BL233_A has been obsoleted. It is still available by special order for existing customers only.

23.1 BL233_A to BL233_B

BL233_B identified by welcome string “Hi I2C v118”.
Known bugs have been corrected. It is not expected that existing applications will see any incompatibilities, as the default behaviour remains the same.

23.1.1 Wait for EOL ":"

Fixed. This is the only major visible improvement for most users. Wait for EOL can now be used freely, and this makes timing dependent sequences much easier. However *only* LF is recognised as the resume char now. CR will be ignored.

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

23.1.2 I2C Stop Deadlock Automatic Recovery

When an I2C bus *STOP* fails because SDA remains low, BL233B now sends up to 8 extra SCL pulses to try and clear it automatically. See 19

23.1.3 1-Wire Buses Fixed

All 1-Wire buses are working.

23.1.4 RTS Input Fixed

Default is to ignore RTS for backward compatibility with previous versions.

23.1.5 XON/XOFF Added

A consequence of this has been a slight change in the CTS pin behaviour. Previously the CTS pin fifo threshold had no hysteresis. Now there is 16 chars of hysteresis.

23.1.6 Reset Macro not executed in Special Pins Mode

As it is possible to accidentally put an endless loop in the reset macro, it is no longer executed in special pins mode to prevent the effective destruction of the part.

23.1.7 Stop always follows reads correctly

23.1.8 Improved Formatting of Data

To assist with formatting the returned data some changes have been made.
fControlSuppressEOL can be used to suppress the LF that follows all data reads.
',' is no longer ignored, now it echoes a comma back.
'.' is no longer ignored, now it returns a linefeed.

23.1.9 Hi Chars can be used

Chars with bit 7 set will be used as data bytes with bit 7 cleared. This is useful when a lot of ascii chars are going to be sent as they can be sent with just 1 char not as 2 hex chars. EEProm bit fSerialUseHiChars will enable this feature.

23.1.10 N Command documented

The Nack (**N**) command is documented, and works correctly.

23.1.11 Internal Reset

The internal reset arrangements are slightly different. The part no longer resets when VCC<4V. A watchdog has been added to avoid the part locking up due to brownouts etc.
This also means there is no special longer a low-voltage variant.

23.1.12 BL232 B

BL232 is a subset of the basic I2C functions of BL233B for cost sensitive *high volume applications*.

24 Erratae

24.1 BL233_B

24.1.1 IRQ low during reset / power on *appears* to stop BL233 working

Special Pins Mode of BL233A,B is entered by *either*

- IRQ held low during power on, *or*
- SDA1,SCL1,SDA2,SCL2 are all held down during power on

This is contrary to earlier descriptions of SPM.
This can give the appearance of the BL233 not working if IRQ is held low during power on, as SPM stops the reset macro from running (so there is no hello message), and forces baud to 57600.
The BL233B is running normally however. See Special Pins Mode for more details.

24.1.2 RdBlockAck with R command

The flag RdBlockAck behaves as documented with reads that began with **S** command. However if you are using the **R** command, the operation may not be as expected. Don't use **R** if you need to use this function.

24.1.3 Clock Stretch does not work on Stop

Clock stretching does not occur correctly for the STOP command. BL233_C behaves correctly.

24.1.4 Peculiarity in Timing set by J command

For Timing values of 1 and 2, T_{HI} is not actually longer. See table of values.

24.1.5 Watchdog does not unblock Pause

The watchdog function does execute until EOL is received when in Pause (":" command)

24.1.6 1Wire

"W" should be used before all writes. Behaviour without it is not as expected for longer sequences of bytes.
Previously the datasheet stated: "Since **S**tart can be followed immediately by a single byte of write data, a shorter version is: GC S 33 R08"

24.1.7 Xon/Xoff lockup modes

XON/Xoff handshaking is always susceptible to locking up when Xoff has been sent but Xon cannot be sent.

- RX buffer overflows will result in Xon not being sent
- Watchdog can be programmed to send Xon

24.1.8 fSerialEEWrProtect

EEProm cannot be written if this is set, even in Special Pins Mode

24.1.9 Timer EEProm Values can Lock BL233B

Setting the Timer eeprom value to less than 0x0030 will fatally lock up the BL233B. Special Pins mode will not recover from this.

24.1.10 Unexpected Extra Clock Pulses at Stop

When the SDA line has a high capacitance on it, then the anti-deadlock featureI2C Bus Deadlock section 19 can result in an unexpected 8 clock pulses when the stoP command is sent. You should slow the Bus Timing Value 6.1.1 using the J command.

24.1.11 Short SCL Hi after Clock Stretch

When SCL is stretched by the slave, the high time, THD, can be shortened to 2.2 – 3.5us (normally 4us), regardless of the I2Ctiming value set with the J command. This will be significant when longer bit times are being used.
Start and Stop THD will be shortened to 1.4 – 2.4us.

24.1.12 Repeated start does not meet TSU:STA specification

When performing a repeated start eg "**S40S41**" the second repeated start has short start setup time TSU (SDA rises 270ns before SCL rises). This occurs after writes not reads as nack leaves If this is a problem, use the *output* command to manually raise SCL before issuing stop. e.g for Bus1 use "**O00FES**" instead of "**S**" for the repeated start.
TSU:STA specification is 4.7us (standard) 600ns fast, and THD:STA is 4us or 600ns

25 BL233_C

BL233C is available in DIP for testing from Sept 2016. Customers should transition to BL233C during 2017. Request pre-release samples for testing from sales@i2cchip.com.

BL233C is intended to be a drop in replacement for BL233B. Normal operation of existing commands should operate the same.

These are the changes. See BL233C New Features for detailed description of the new features.

- Special Pins Mode works as originally planned, therefore IRQ pin does not affect startup.
- Quoted Strings supported. Makes it easier and faster to send ASCII text through to I2C devices such as display modules.
- Handshake threshold changed to increase apparent serial fifo size.
- RdBlockAck control bit obeyed in **R** mode
- Clock Stretch on Stop works correctly.
- Special Pins Mode also applies to eeTimerDiv and eefControl, eefControl2
- TimerDivH forced to minimum of 1 to prevent fatal lockup
- ";" printable resume/EOL char for pause command.

- XON sent at Reset (if enabled) and on RX buffer overflow
- Watchdog will escape from pause & xoff & quoted string
- fStatus-FifoOverflow correctly shows RX fifo overflows
- RX buffer overflow sends XON, resumes and exits quoted strings.
- Swap RX/TX fifos: RX expands to 80, TX shrinks to 48chars. Space before CTS 32→53. Allows XON/XOFF where flow control allows 16→25 chars after XOFF
- EEADR is parked outside eeprom, at reset, after eewrites, and dump.
- EEProm can be written in Special Pins mode, ignores EEWrProtect bit.
- I2CEEProm complex read/write commands make eeproms simple
- Stack Data Commands to manipulate data and do decimal conversions.
- Sleep mode. Power-down sleep with wake on time or interrupt
- Sleep mode. Wake on character match for addressing or TTYS0 problems
- fixed short scl and stop after stretch
- fixed Tsu in repeated start after write
- Delay "L" can be inside read/write sequences eg S405A L0001 5B P
- Delay "L" Jitter has been reduced from 1ms to 100us.
- Read rate improved for baud rates >230k. e.g. read 255bytes 32→23ms (G5, 307200bd)
- J command gets 3rd byte: fControl2

fControl2 (write with J)			
Bit	Default	Name	Description
7	0	BlockTimeout	SCL Clock Stretch is unlimited
6	0	RetryAddressUntilAck	Retries Start+Address until ACKs

- BlockTimeout flag to allow unlimited SCL Stretch.
- Auto-Retry-Unitl-Ack mode for devices that go off the bus while busy e.g. I2CEEProm writing: retries Start+Address sequence until acknowledge.
- Add extra 1Wire bus on SDA2 at unused bus GB
- RTS is AND gate for linux ttys0 operation (special version)

Todo??:

- Duplex? would allow duplex?.
- Explain TimerTimedOut bit in fStatus
- I2Ceeprom does TimerTimedOut work if it is read after a pause or is it getting set all the time?
- Useful macros
- Multimaster start – wait until stop to try starting?????

26 Revision History

Rev	Date	Change
29		Added section on formatting data
30		Corrected 1 wire ID read command to 8 bytes
33		Clarified EEProm write delay issues
37		Corrected Special Pins Mode
43	24/02/08	Small changes in Formatting related topics
44		Clarify how Fast Mode is selected.
48		Errata added for RdBlockAck
49	15/02/12	Errata added for Clock Stretch on Stop Clarify Bus4 is TTL
54	11/07/13	IRQ Vector and Watchdog vector addresses in eeprom corrected. Handshaking clarified. Add Watchdog examples.
56	31/10/13	Correct control register examples from J14 to J18
59	31/3/14	Add high times to table of timing values for J command. Add links to BL233 App Notes
62	6 July 14	Extend and correct 1Wire section: Add DS1820 example. Remove suggestion that "W" is not needed for some writes. Realterm 1Wire features improved at same time.
66	6 Feb 15	Improve explanations of reading more than 255 bytes in a continuous read: Reading more than 255 bytes / Read without NACK section 5.9.2 above.
68	24 Apr 15	EEProm min voltage added
77	15 Aug 16	Add 13.2.1 Testing Handshake section to serial comms Add Warning and Errata 45 Timer EEProm Values can Lock BL233B Errata: Eeprom cannot be written in Special Pins mode when EEWrProtect is set 24.1.8 fSerialEEWrProtect . Update BL233C changes, add release date and document links.
83	13 Jan 17	Add Errata Unexpected Extra Clock Pulses at Stop Add 13 Delaying WITHIN a single write/read sequence minor changes in 6 Status and Control Registers Add 10.6 Writing/Reading I2C without Start and Address
84	17 Jan 17	Clarify that clock stretch timeout is 17ms total Add errata 24.1.11 Short SCL Hi after Clock Stretch

86	18 Jan 17	Add errata 24.1.12 Repeated start does not meet Tsu:sta specification
95	23 Jan 17	Add more BL233C notes into text.

27 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: BEL233B-P (multiples of 10 and 25pcs)
SO18: BEL233B-SO (multiples of 50 pcs) Tape and Reel.
SSOP20: BEL233B-SS (multiples of 50 pcs) Tape and Reel

EEProm settings can be customised at the factory.

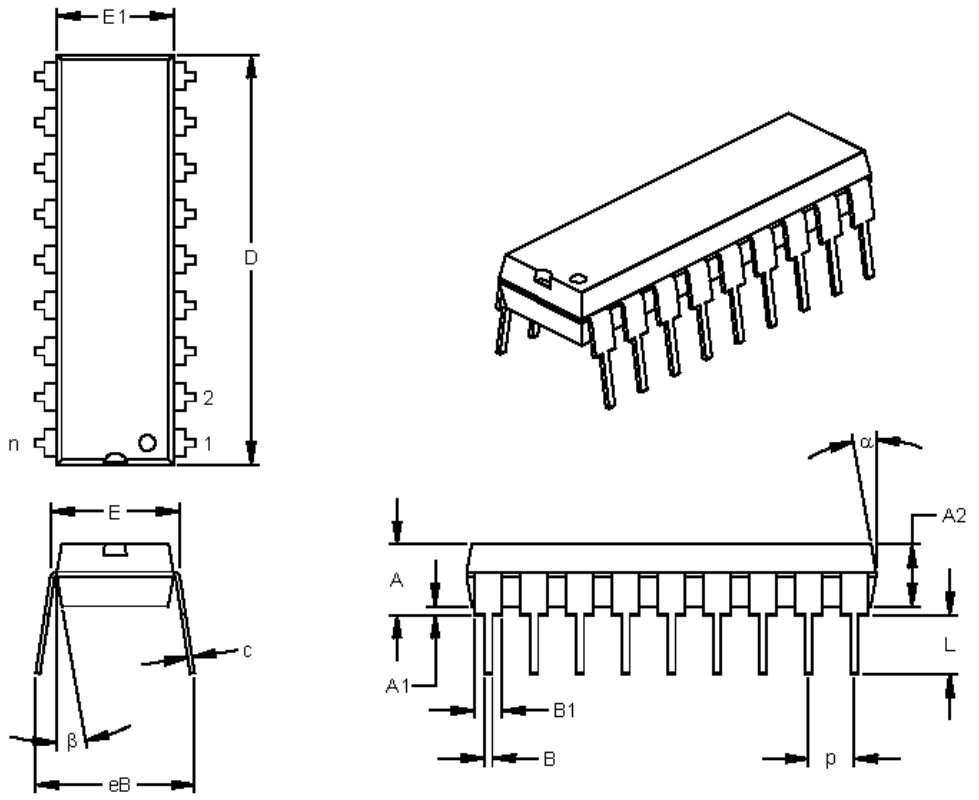
27.1 RoHS and Lead Free

All packages are ***Lead-Free*** and ***RoHS compliant***.
Parts are compatible with lead-free reflow profiles up to 260ºC

Pb-Free Finish Category: e3 (matte Tin)

27.2 Package Dimensions

27.2.1 18-Lead Plastic Dual In-line (P) – 300 mil (PDIP)

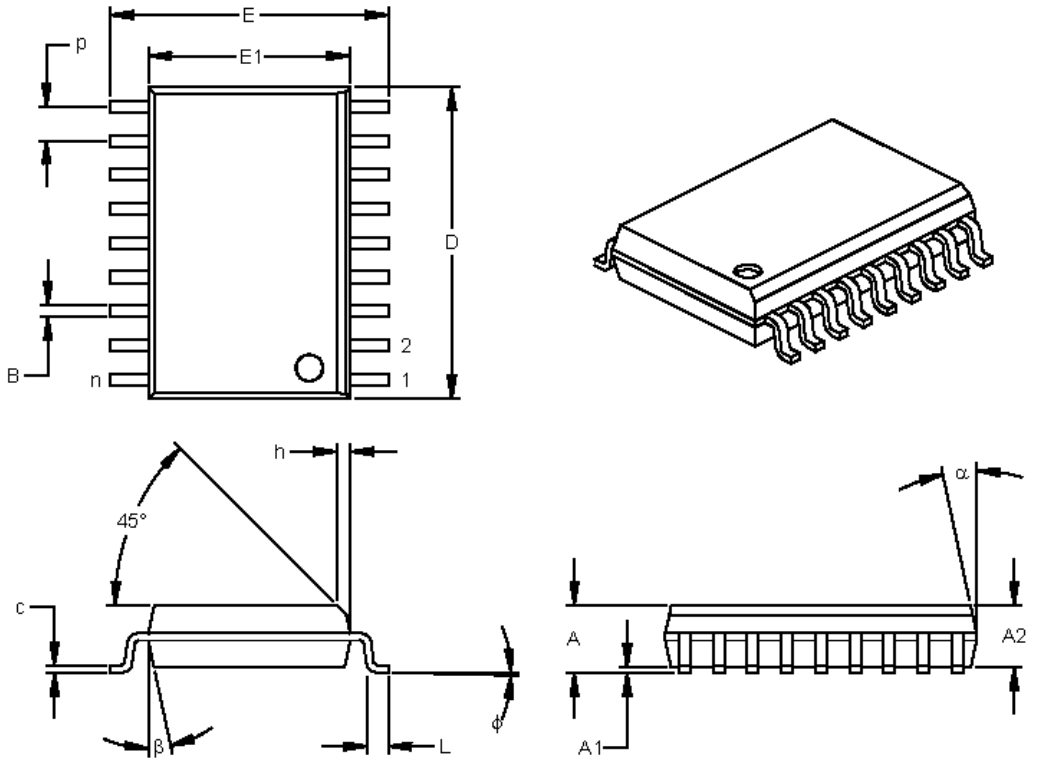


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		18			18	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.155	.170	3.56	3.94	4.32
Molded Package Thickness	A2	.115	.130	.145	2.92	3.30	3.68
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.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

27.3 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

27.4 SSOP etc

Contact factory for availability of other packages.

28 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.

Bibliography

1;I2C Clock Stretch Probe,I2CCIP
;http://www.i2cchip.com/pdfs/I2C_Clock_Stretch_Probe.pdf
2;I2C-bus specification and user manual,
;http://www.nxp.com/documents/user_manual/UM10204.pdf