Reading and Programming I2C EEProms

1 Introduction

The programming and reading of I2C EEProms is one of the major uses of the I2C2PC and BL233B.

The revised BL233C (release late 2016) adds built in support for reading and writing I2C EEPROMs, and makes the process much simpler.

This document is about

- I2C EEProm types, addressing etc.
- I2C-Chip tools for programming and dumping eeprom,
- Tools for joining, trimming, comparing, viewing eeprom data files
- Details of eeproms, reading and programming to do it yourself

For SPI eeproms see: http://www.i2cchip.com/pdfs/SPI_App.pdf

Contents

Other References

The primary software reference is the BL233B Datasheet http://www.i2cchip.com/pdfs/bl233_b.pdf. See Section

This document is commentary, the BL233 datasheet is the primary command reference you need to follow.

A good summary of using various I2C EEProms is Microchip's "AN536 Basic Serial EEPROM Operation"

2 Memory Addressing of EEProms

The memory addressing arrangement can be very confusing when you first meet it. The internal eeprom address pointer is set using one or two pointer bytes, combined with 0-3 of the lower I2C address bits as block select bits. The memories can be classified as:

- Small: 512 to 2k bytes (24C00 - 24C16), have only a single pointer byte, and use the 3 device subaddress bits to select blocks of 256 bytes. The pointer is set by the sub-address bits+pointer byte. These can be treated as 2,4 or 8 separate 24C02 memories with sequential I2C sub-addresses. These do *not* decode the address pins (even though the pinout has them labelled), they respond to all 8 sub-addresses, and thus only one chip can be used on an I2C bus.
- Medium: 4K to 64K bytes, (24C64 - 24C512), they have 2 pointer bytes, use a single I2C address, and do recognise and decode the sub-address bits.
- Large: 128k - 512k bytes (24LC1025=24M02) still have 2 pointer bytes, they return to using I2C subaddress bits for blocks of 64kbytes. However they do decode the unused subaddress bits, and you can have multiple.
- Vast: 8M+ devices will need to have 3 pointer bytes, but do not current exist.

2.1 I2C Address Bits

Most of the chips <=2kBytes (24XX00 - 24XX16), have labelled address pins A0-2, but *they are not connected*.

- The chips respond to all 8 sub-addresses, the data is repeated on different subaddresses.
- Only one of these chips can be used on an I2C bus at a time.
- Most >=4KByte chips do use the unused sub-addresses, so multiple chips can be used
- There are a few small memories that use address pins, and so you can have multiple memories on a single I2C bus, eg 24LC025.

2.2 Table of EEProms

Part	Bits	Bytes	Blocks	# I2C sub-add	Pointer Bytes	Page Size	Time	CC	N-all
24LC00	512	64	1	1	1	1	5	00	40
						4		01	
24C01	1k	128	1	1	1	8	5	02	80
24LC02	2k	256	1	1	1	8	5	02	FF
24LC025	2k	256	1	8	1	16	5	03	FF
24LC04B	4k	512	2	1	1	16	5	03	FE
24LC08	8k	1k	4	1	1	16	5	03	FD
24LC16	16k	2k	8	1	1	16	5	03	FC
24LC32	32k	4k	1	8	2	32	5	0C	FB
24LC64	64k	8k	1	8	2	32	5	0C	FA
24LC128	128k	16k	1	8	2	64	5	0D	F9
24LC256	256k	32k	1	8	2	64	5	0D	F8
24LC512	512k	64k	1	8	2	128	5	0E	F7
24LC1025	1M	128k	2	4	2	128	5	0E	F6
24M02	2M	256k	4	2	2	256	10	0F	F5
?	4M	512k	8	1	2	256	?	0F	F4

CC, N-all values used by BL233C eeprom command

2.3 I2C Address Pins, Page Sizes and Write Times:

Parts of the same capacity can have very different parameters, so *the data sheet must be checked for your exact part*. The following are all Microchip 256byte eeproms.

Part	Bits	Bytes	Blocks	Usable I2C Addr	Pointer Bytes	Write Page	Time
24LC02, 24AA02	2k	256	1	1	1	8	5
24AA025,24LC025	2k	256	1	8	1	16	5
24C02C	2k	256	1	8	1	16	1
24LC22	2k	256	1	1	1	16	10

3 Writing

Writing the EEProm cells is initiated by the I2C STOP. While the data is being written, the memory goes off-line, and does not respond to addressing. Following each write by a delay command ensures that the memory is online, when the next I2C command begins. Make sure that you use the maximum time from the datasheet (eg 5ms) not the nominal (3ms) time. Make sure you are using RTS/CTS handshaking.

3.1 Single Byte Writes

The easiest method to write is simply to repeatedly write each single byte, and increment the pointer byte each time.

Size	Command	eg: location 0x0234, data =5A 5B
<=2kB	S [I2CAdd-w] [pointer] [data byte] P L[delay in ms]	SA2 34 5A P L0005 W 35 5B P L0005
>=4kB	S [I2CAdd-w] [pointer H,L] [data bytes] P L[delay]	SA0 0234 5A P L0005 W 0235 5B P L0005

W command writes to last address, ie "W" replaces the "S A2" for subsequent writes.

3.2 Paged Writes

EEProm data is written in pages. The write time the the same for a page, as for a single byte.

The data must not straddle page boundaries. You can do partial page writes, eg you can write a memory in 8 byte blocks, when it has 16 byte pages. This may be useful, as you will see, almost all of the memories have >=8byte pages.

The page size depends on the chip. Do not assume that all manufacturers and versions of a particular chip will have the same page size - check your exact parts datasheet.

Size	Command	eg: @ 0x0230, data =5A 5B 5C 5D
<=2kB	S [I2CAdd-w] [pointer byte] [data bytes] P L[delay in ms]	SA2 30 5A 5B 5C 5D P L0005
>=4kB	S [I2CAdd-w] [pointer H,L] [data bytes] P L[delay]	SA0 02 30 5A 5B 5C 5D P L0005

3.3 Programming constant values

Many applications involve simply initialising a memory with a standard set of data. As this data will never change, the easiest process is:

- Convert the data to a text file of BL233/I2C2PC Commands. This only needs to be done once
- Send the command file to the I2C2PC adaptor

4 Reading

To read an I2C memory, you must first write the memory pointer, then initiate a read operation. Data is returned as hex bytes.

Size	Command	eg: 16 bytes from 0x0230
<=2kB	S [I2CAdd-w] [pointer byte] R [Num Bytes] P R [Num Bytes] P ...	S A2 30 R 10 P . R10 P
>=4kB	S [I2CAdd-w] [pointer H,L] R [Num Bytes] P R [Num Bytes] P ...	S A0 02 30 R 10 P . R10 P

- You can read up to 256 bytes in a single read.
- The pointer auto-increments, so you can repeat a read command to read more bytes, as long as you stay within a block (ie 256 bytes for <=2kB mem)
- Use . and , commands to format the data into lines if you need, see: http://www.i2cchip.com/pdfs/BL233_Data_Formatting.pdf

Speed

When reading large memories, baud rate controls the time taken to read. For large memories changing the baud rate will reduce the time.

Baud	1KB	Notes
57.6k	0.4s	default
115k	0.2s	
307.2k	0.1s	In USB mode, but only in some RS232 ports

5 Handshaking & Baud Rate

As programming eeproms involves delay periods, and dumping eeproms sends a lot of characters back, you *must* use handshaking on the serial connection. If you are using the Realterm application from a batch file, use the RTS=1 commandline parameter.

For large eeproms, there is a lot of data to transfer, and you may want to increase the baud rate from the factory default of 57600. If you are using the USB interface, then you can use 307200 bd. If you are using RS232, then 230400 is commonly available.

6 Programming from Hex Files

Complete scripts to convert hex files into BL233B / I2C2PC commands, and program I2C eeproms, are installed with Realterm in %ProgramFiles%\BEL\Realterm\examples\I2C_EEProm_Programming.

You should edit ProgramI2CEEProm.bat to set the serial port to use, and point at any utilities that are not in the standard place. It defaults to common values for a range of chips.

You need to specify the number of bytes in the pointer address for your memory type, see

Chip	Command Line	pagesize
24xx01 - 24xx16	`ProgramI2CEEProm.bat <YourHexFile> 1`	8
24xx32 - 24LC1025	`ProgramI2CEEProm.bat <YourHexFile> 2`	32
24M02	`ProgramI2CEEProm.bat <YourHexFile> 2 A`	32
24C00	`ProgramI2CEEProm.bat <YourHexFile> 1 5 1`	1

You can also change the pagesize to optimise the writing for particular memories.

6.1 Hex File Tools

These tools are installed with Realterm.

Name		
SRecord	Universal Hex Eprom file converting / combining / comparing	
Hex2CSV	Convert hex data files, to (meaningful) CSV numbers	

6.2 Convertor Script

Hex file data lines must fit evenly into eeprom page size. Commonly files have 16 bytes per line, which works with 24LC04 up. Use SRecord to resize lines.

srec_cat <YourHexFile> -Output_Block_Size <pagesize> -Output_Block_Alignment -o OutputFile.srec -address-length=3

```
sed -n -f S2bl233b.sed <your hex file>
```

- Set the of one/two byte pointer for your eeprom
- Most eeproms work with 5ms delay, but check and set the delay if needed.

6.3 Converting Hex Files to BL233 Commands

If you want to do this conversion yourself, this is what happens. You do not need to interpret or understand the file contents. You can just select and pass through selected hex characters from the input file to the bl233 file. This is only string processing, and can be done easily with SED, Perl, Python etc.

In	SRecord File - S1	**S110022148656C6C6F2C20576F726C640AA1**
In	InHex8 File	:0D022100**48656C6C6F2C20576F726C640A**A1
Out	I2C 24LC16 (1 byte pointer)	**SA4 21 48656C6C6F2C20576F726C640A P L0005**
Out	I2C 24LC64 (2 byte pointer)	**SA0 0221 48656C6C6F2C20576F726C640A P L0005**

Key:

Address Data Write Time in milliseconds (hex)

Upper nibble of Address, shift left and pushed into I2C address.

For eeproms >=128k, you need to use SRecord files with 3 or 4 byte addresses ie S2, S3 records.

In	SRecord File - S2	**S31003022148656C6C6F2C20576F726C640AA1**
In	SRecord File - S3	**S3100003022148656C6C6F2C20576F726C640AA1**
Out	I2C 24LC64 (2 byte pointer)	**SA6 0221 48656C6C6F2C20576F726C640A P L0005**

7 Revision History

Rev	Date	Changes
1	31 Mar 14	New
2	9 May 2014	Added section on converting hex to programming files
3	8 Apr 2016	Add link to SPI app note
4	20 July 2016	Add BL233C information to table.