i2cchip.com

Reading and Programming I2C EEProms

Introduction

The programming and reading of I2C EEProms is one of the major uses of the I2C2PC and BL233.

The revised BL233C [1] (release 2017) adds dedicated I2C EEProm command K, whch makes the process much simpler. New applications should use BL233C, existing BL233B commands will still work correctly, there is no reason to change. If you have I2C2PC adaptors, you can replace the BL233.

Contents

Just want to Dump an eeprom to a file? Send the K-Dump command (Table of EEProms, sec 1.3) using Realterm or see Programming and Dumping Hex Files, sec 5

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: SPI_App [2]

Other References

The primary software reference is the BL233 Datasheet [3]

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

1. Addressing the EEProm Memory Space

*BL233Cs **K** command hides this complexity especially for large reads and writes.*

I2C EEProms have an internal address pointer. This must be written (an I2C write cycle) before data can be read or written. The address pointer does not reset itself automatically. It does auto-increment, but only an 8 bit counter (256 bytes) on many 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. Sequential read operations can read 256 bytes
- 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. Sequential read operations can read up to 65536 bytes
- 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. Sequential read operations can read up to 65536 bytes
- Vast: 8M+ devices will need to have 3 pointer bytes, but do not currently exist.

So the read and write I2C commands will be thus:

Start {i2c address−Write} {1 or 2 pointer bytes} Start {I2C Address−Read} {Read N data} stoP

Start {i2c address−Write} {1 or 2 pointer bytes} {Write 1 to Page−size bytes} stoP

If you have not used I2C before, understand that a read involves writing the pointer, then restarting and re-addressing to read the data. This is confuses many newcomers.
The BL233Cs **K** command simplifies this for you.

1.1. I2C Address Pins

Most of the chips <=2kBytes (24XX00 - 24XX16), have labelled address pins A0-2, but *they do nothing*. (24xx32 and larger *do* decode A0-2)

- 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 decode the unused sub-addresses, so multiple chips can be used
- There are a few small memories that obey address pins, and so you can have multiple memories on a single I2C bus, eg 24LC025.

You can use the "Scan Bus" button in Realterm (on I2C tab), to see what addresses your memory actually responds to.

Make sure you set the unused Address and WP pins correctly.
Microchip writes about 24aa1025 TABLE 2-1: PIN FUNCTION TABLE, pin A2: "Non-Configurable Chip Select. This pin ***must be hard-wired to logical 1*** state (VCC). Operation will be undefined with this pin left floating or held to logical 0 (VSS)."

1.2. Sequential Reads and Writes

Sequential reads can be the size of the address pointers i.e. 256 bytes for up to 24xx16 and 65536 for 24xx32 and up. Microchip writes: "To provide sequential reads, the 24XX1025 contains an internal Address Pointer which is incremented by one at the completion of each operation. This Address Pointer allows half the memory contents to be serially read during one operation. Sequential read address boundaries are 00000h to 0FFFFh and 10000h to 1FFFFh. The internal Address Pointer will automatically roll over from address 0FFFFh to address 00000h if the master acknowledges the byte received from the array address, 0FFFFh. The internal address counter will automatically roll over from address 1FFFFh to address 10000h if the master acknowledges the byte received from the array address, 1FFFFh."

Sequential writes roll over at the wite page size.

1.3. Table of EEProms

This is a summary of common eeprom, but do check the datasheet for your memory

Blocks is the number of sub-addresses used by this chip
Addrs is number of I2C Sub-**Addr**esses decoded, i.e. the number of chips you can use on the same I2C bus
Ptr is how many Pointer bytes are used, 1 will be 256B blocks, 2 for 65536B blocks
Page Size for writes, **Time** is max write **Time** (ms),
CB is ControlByte used by BL233Cs **K** eeprom command. (Page size and #Index)

N-all is the special byte-count value used after **R** to read out the *entire* eeprom at once with **K** command.
K-Dump Command to read entire eeprom using **K** command

Part	Bits	Bytes	Blocks	Addrs	Ptr	Page	Time	CB	N-all	K-Dump
24LC00 [4]	512	64	1	1	1	1	5	00	40	K00R40
<none>						4		01		
24C01 [5]	1k	128	1	1	1	8	5	02	80	K0280
24LC02	2k	256	1	1	1	8	5	02	FF	K02RFF
24LC025	2k	256	1	8	1	16	5	03	FF	K03RFF
24LC04B	4k	512	2	1	1	16	5	03	FE	K03RFE
24LC08	8k	1k	4	1	1	16	5	03	FD	K03RFD
24LC16	16k	2k	8	1	1	16	5	03	FC	K03RFC
24LC32	32k	4k	1	8	2	32	5	0C	FB	K0CRFB
24LC64 [6]	64k	8k	1	8	2	32	5	0C	FA	K0CRFA
24LC128	128k	16k	1	8	2	64	5	0D	F9	K0DRF9
24LC256	256k	32k	1	8	2	64	5	0D	F8	K0DRF8
24LC512	512k	64k	1	8	2	128	5	0E	F7	K0ERF7
24AA1025 [7]	1M	128k	2	4	2	128	5	0E	F6	K0ERF6
24M01 [8]	1M	128k	4	2	2	256	5	0F	F5	K0FRF6
24M02 [9]	2M	256k	4	2	2	256	10	0F	F5	K0FRF5
?	4M	512k	8	1	2	256	?	0F	F4	K0FRF4

1.4. 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	#Addr	#Index	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

2. Easy I2C EEProm K Command (BL233C)

BL233C can manage the process of reading and writing I2C EEProms, handling the data pointer, paged writes and timing for you. This turns the reading and writing into trivial sequential reads and writes.
The data pointer is persistent, making it possible to resume reads and writes at the previous location, and (with the stack commands) to move data from an I2C device directly into an I2C memory.

I2C EEProms have up to 20 bits of data pointer. They use a one or two byte pointer, and up to 3 bits mixed into the lower bits of the address byte. When writing they have write pages from 1 to 256bytes long. The Control Byte configures K command for 1 or 2 address bytes and the write page size. (see section **??**)

The **K** command sets up the pointers. **R** & **W** commands read and write the eeprom. The basic command format is:

K {Hi 3 Pointer Bits}{Pointer Byte M}{Pointer byte L}{Control Byte}{Rnn or W [data]}

The Control Byte (CB) is specific to the type of eeprom. Unused parameters can be omitted and will default in a sensible way. See EEProm table for CB values.

I2C base address defaults to 0xA0. The merging of data pointer bits into the I2C address is taken care of for you. The address pins, if used are simply the upper 3 bits of the pointer.

K R02 ˜ resume reading from previous data pointer
K 02 R04 ˜ ControlByte=02, Data Pointer id cleared to 00, use default I2C address of A0, read 4 bytes
K 40 02 RFF ˜ CB=02, DP=0x40, read 256bytes
K 21 40 0D RFE ˜CB=0D, DP= 0x02140, read 512 bytes
K 03 21 40 0D RFD ˜CB=0D, DP= 0x032140, read 1024 bytes

If you are using different address pins, e.g A0-2 of 24LC512 are held high, the Hi pointer will be 7

K 07 21 40 0D RFD ˜CB=0D, BaseAddress=AE −Address pins tied high, DP= 0x2140, read 1024 bytes

All known eeproms have base address Ax, and the K command will address them by default. If for some reason you have some special chip with a different base address you can set the high nibble.

K B0 21 40 0D RFD ˜CB=0D, BaseAddress=B0, DP= 0x2140, read 1024 bytes

2.1. Control Byte

see section **??** for actual values for eeproms.

The control byte is:

Bit	7	6	5	4	3	2	1	0
Fn	0	0	0	0	2ByteAdd	Write Page Value		

The PageWriteSize

Write Page Value	0	1	2	3	4	5	6	7
Write Page Size	1	4	8	16	32	64	128	256

2.2. Read Command

Read is done with R <nn bytes> . See section **??** for command to dump entire eeprom.

The number of bytes to read (nn) is modified from the normal read command. From 1 to 0xF3 it is simply the number of bytes to read, as normal. From FF to F4 it goes in binary steps: FF=256, FE=512, ... F4=512k.

Thus the whole memory can be read with a single command.

K 02 R04 ˜ ControlByte=02 (24C02), Default Data Pointer to 0, use default address of A0, read 4 bytes
K 40 02 RFF ˜ CB=02, DP=0x40, read 256bytes
K 21 40 0D RFE ˜CB=0D, DP= 0x02140, read 512 bytes
K 03 21 40 0D RFD ˜CB=0D, DP= 0x32140, read 1024 bytes

You can dump the entire eeprom with a single command (see section **??**) Reading large memories is a lot of data. You may want to change baud rate.

```
K 03 RFC              ~ dump entire 2kB of 24LC16
K 0E RF7          ~ dump entire 64kB of 24LC512
K 07 00 00 0E RF7  ~ dump entire 64kB of 24LC512 at I2C address A0−2=111
```

The data pointer is persistent so you can resume reading from the same point later if **R** or **W** immediately follows the **K**.

```
K R02          ~ resume reading from previous data pointer
```

You can format the data. e.g. insert commas,linefeeds,comments (data at 0x00 is A0 A1 - A7)

```
K 02 R02,02,02,02. ~ two bytes values separated by commas (returns A0A1,A2A3,A4A5,A6A7<lf>)
T"Volts"K 02 R02P. ~ (returns Volts=A0A1<lf>)
```

The BL233C Stack Commands will let you manipulate the data - swap bytes, convert to decimal, scale sensor values etc. (@ reads onto stack, # types hex from stack)

```
K 02 R2@##,2@##. ~ swap 2 bytes values comma separated (returns A1A0,A3A2<lf>)
T"Swapped="K 02 R2@##,T"Unswapped="KR02P.  ~ (returns: Swapped=A2A1,Unswapped=A2A3)
```

You can also use it to read text out of the eeprom as ascii (= pops the ascii value off stack)

```
K 18 03 W "Volts" P      ~write "Volts" at 0x18
K 18 03 R1@=@=@=@=@=P.  ~ (returns: Swapped=A2A1,Unswapped=A2A3)
```

BL233 Can also read out variable length data, where the first byte is the total length

```
K 18 03 W 05 A1A2A3A4 ~write 5 bytes at 0x18
K 18 03 R 00.  ~R00 reads variable length (returns: 05A1A2A3A4 )
```

2.2.1. Speed Improved

Serial throughput has been improved in BL233C for data read back at baud rates >=230k. Using fast bus (G5) at 307kbd, 255 byte read improves to 23ms from 32ms.

2.3. Write Command

The **K** command automatically handles write paging and data pointers, you can write the whole memory in a single operation if you wish.

```
K 00 03 W A0 A1 A2 A3 P  ~write A1−A3 into bottom (Pointer=00) of 24LC16 (CB=03)
K 04 03 W A4 A5 P  ~write A4 A5 at 04
K W A6 A7 P        ~resume writing (at 06)
K03R08P              ~reading it all returns: A0A1A2A3A4A5A6A7
K 10 03 W "This is a literal text string written at address 0x10"
```

Writing is automatically executed when a write page boundary is crossed or the sto**P** command is sent. The eeprom goes offline while writing, and the BL233 loops addressing it until it returns, or times out. You can start and finish writing at any address.

You can use as much or as little of the address as you need.

```
K03 W A0A1P  ˜writing at 0x00 in 24C16
K 40 0E W A0A1P ˜writing at 0x40 in 24LC512
K 2140 0E W A0A1P ˜writing at 0x2140 in 24LC512
K 2140 0E W A0A1P ˜writing at 0x2140 in 24LC512
K 07 2140 0E W A0A1P ˜writing at 0x2140 in 24LC512 with I2C Address=AE
K 032140 0F W A0A1P ˜writing at 0x032140 in 24M02
```

The stack operations let you move data directly from a sensor to eeprom . e.g Reading PCF8574 8 bit port at 0x40, to a 24LC512 eeprom

```
S41@P  ˜read port to stack
K 28 0E W @ P   ˜write from stack (@) to 0x28 in eeprom
```

Since the eeprom pointer is persistent we can log it sequentially into the eeprom

```
˜ Initialise once at the start...
S40FFP ˜initialise port to inputs
K0EP ˜will set eeprom pointer to 0

˜ do this each time – could be a loop inside a macro
S41@P  ˜read port to stack
KW@P   ˜write at next location in eeprom
```

With the delay and sleep commands, a simple data logger will run in the BL233Cs power-on macro

2.3.1. Offline after Writes

EEProms go off-line during the write cycle, and do not *ACK* until finished. The **K** command handles this during the write cycle, and during a subsequent **K** read. However beware if you want to use a basic operation immediately following a write. You may need to insert a delay.

The BL233C extended Control2 register has bit fRetryAddressUntilNack if you need to use basic I2C commands to implement eeprom writing without delays.

2.4. Timeout

K operations retry the address for up to 255ms. If they fail (e.g. eeprom missing), fSCLTimeout in Status is set. The timeout is clears at the K, and all read and writes will fail until the next K. The Status command **?** commonly returns 18 for success, 19 or 1B for missing or timeout.

3. Using Basic I2C Commands (BL233B)

Note basic commands will still work properly on BL233C.

3.1. Reading

To read an I2C memory, you must first write the address pointer (one or two bytes - see table), then initiate a separate read operation with a repeated start. 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 255 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 <=24xx16, 65536 for >=24xx32)
- Use "**.**" and "**,**" commands to format the data into lines if you need, see: BL233_Data_Formatting.pdf [10]
- J88 stops the automatic linefeeds after each read use "**.**" command to insert when you want them

To read more than 255 bytes you can repeat the byte count

```
J88  ~ suppress linefeeds (only once at start)
S A0 00 R FF 01 P  ~read 255+1 bytes
S A0 00 R 80 80 P  ~read 128+128 bytes
S A0 00 R 80 80 SA3 80 80 P ~read 512bytes, SA3 reads 2nd block
```

With 24xx32 upwards you can read up to 65536 bytes

```
J88  ~ suppress linefeeds (only once at start)
S A0 0000 R FF FF FF FF 04 P  ~read 1kB
S A0 0000 R FF {...257 FFs total...} FF 01 P  ~read entire 64kB 24aa512
```

3.2. 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 (acknowledge) addressing. Only data within a single write page can be written at once.

BL233B should follow each write by a delay command to ensure 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.

BL233C Extended Control Register has a bit fRetryUntilAcknowledge that can be used instead of delays - if you have some reason not to use the K command

3.2.1. Single Byte Writes

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

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.

3.2.2. 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

3.2.3. Programming constant values

Many applications involve simply initialising an eeprom 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. Handshaking, Baud Rate & Speed

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.

Speed

For large eeproms, there is a lot of data to transfer. To get best speed:

- Increase the baud rate from the factory default of 57600 to 230,400Bd.
- Use Fast I2C Bus timing with **G** command
- Use BL233C - Serial throughput has been improved for data read back at baud rates >230k. Using fast bus (G5), 255 byte read improves to 23ms from 32ms., **K** command is more compact

Baud	1KB	Notes
57.6k	0.4s	default
115k	0.2s	
203.4k	0.1s	recommended for speed

Use one of the fast I2C buses

Bus	Normal	Fast
1	G1	G5
2	G2	G6
3	G3	G7

5. Programming and Dumping Hex Files

Scripts to read and write eeproms are installed with Realterm in Examples Directory [11] or in I2C_EEProm_Scripts.zip [12] . There is a directory **I2CCmdFiles** which contains the actual BL233 commands to read specific eeproms in S19 and HEX format. [BL233B tools below will still work for BL233C]

The SRecord utility can convert between many different common hex formats like inhx8 as well as S19.

- **ReadI2CEEPromSREC.bat** reads to formatted SREC/S19 hex files
- **ReadI2CEEPromHEX.bat** reads to plain unformatted hex files
- **UtilsSetup.bat** is a common place to set the port, directories etc
- **I2CEEPromTypes.bat** sets up the eeprom parameters
- **ConvertHexFile2BL233.bat** converts a formatted SREC/S19 file to I2C commands

5.1. Dumping to Hex File

The simplest is to dump to a simple, unformatted hex file (no addresses, nothing but continuous hex data)

Batch file **ReadI2CEEPromHEX.bat** does this.

First you will need to edit **UtilsSetup.bat** to set your serial port, directories etc.

Usage: ReadI2CEEProm.bat YourDataFile EEProm-Type

Edit the batch file to add your post-processing e.g. there are SED scripts to space and comma separate the data inside it.

You can do the same thing directly. Use any tool or language that can send to the port and capture the data.

- Port must use handshaking (RTS/CTS)
- Send the appropriate eeprom i2c command file e.g. ReadIMem_Hex_24xx32.i2c
- Capture the returning data to file. There is a linefeed at the end.

- Modify or use the data file. There are SED scripts to space and comma separate the data.

A Realterm commandline is:

realterm.exe port=\vcp0 flow=2 capture=myhexdata.hex capsecs=-30 sendquit=readimem_hex_24xx32.i2c

Within a batch file, you need to use *start* and *wait* so that the batch file pauses until Realterm exits.

start "Reading I2C EEProm" /wait realterm.exe

5.2. Dumping to Formatted SREC/S19 File

Batch file **ReadI2CEEPromSREC.bat** does this. Each type of eeprom has a read command file e.g. ReadIMem_24xx512.i2c

First you will need to edit **UtilsSetup.bat** to set your serial port, directories etc.

Usage:

- ReadI2CEEProm.bat YourDataFile
- ReadI2CEEProm.bat YourDataFile EEProm-Type
- I2CEEPromTypes.bat to see known types

You can add to this to do your own processing or to change the output data format.

5.3. Reading with BL233C

BL233B commands (above) still work, and are just as easy to send from file.

Send the dump command for your eeprom from Table of EEProms, sec 1.3

```
realterm port=\vcp0 flow=2 sendstring=K0EF7P capsecs=-30 capquit=''c:\temp\
```

5.4. Programming from File

You should edit ProgramI2CEEProm.bat to set the serial port to use (must use handshaking), 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	Wr time
24xx01 - 24xx16	**ProgramI2CEEProm.bat <YourHexFile> 1**	8	5
24xx32 - 24LC1025	**ProgramI2CEEProm.bat <YourHexFile> 2**	32	5
24M02	**ProgramI2CEEProm.bat <YourHexFile> 2 A**	32	10
24C00	**ProgramI2CEEProm.bat <YourHexFile> 1 5 1**	1	5

You can also change the pagesize to optimise the writing for particular memories - look inside ProgramI2CEEProm.bat for more info.

5.5. Programming with BL233C

BL233C programming is much simpler using the K command: Make a new file which concatenates

K write command for your eeprom eg “K0EW” for 24LC512.

The Hex data file

Append “P” to finish last write.

Realterm will get a direct eeprom write/dump command late 2017

5.6. Hex File Tools

These commandline tools are installed with Realterm. They can be used in batch files to process the hex.

Name	Function	
SRecord [13]	Universal Hex Eprom file converting / combining / comparing	
HexCSV2Dec [14]	Convert hex data files, to decimal CSV numbers	
SED [15]	Stream EDitor modify, split,extract and manipulate files	

There are SED scripts to turn plain hex into space-separated, comma-separated, and ascii-space-hex format that **srecord** can read e.g.

```
sed -n -f hex2asciispacehex.sed <in.hex> > <out.hex>
```

5.7. Write 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.

5.8. Converting SREC/S19 Hex Files to basic BL233B 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	**S1100221 48656C6C6F2C20576F726C640A A1**
In	InHex8 File	:0D0221 00 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**

A. Revision History

Rev	Date	Changes
1	31 Mar 2014	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.
5	5 Sep 2017	Add BL233C K command, add auto-nameddestinations and url tooltips
7	11 Sep 2017	Correct K command for reading eeproms with A0-2 set

B. URL References

Notes

1. BL233C https://www.i2cchip.com/pdfs/BL233C_New_Features.pdf
2. SPI_App https://www.i2cchip.com/pdfs/SPI_App.pdf
3. BL233 Datasheet https://www.i2cchip.com/pdfs/BL233_Datasheet.pdf
4. 24LC00 http://ww1.microchip.com/downloads/en/DeviceDoc/21178H.pdf
5. 24C01 http://ww1.microchip.com/downloads/en/DeviceDoc/21711J.pdf
6. 24LC64 http://ww1.microchip.com/downloads/en/DeviceDoc/21189T.pdf
7. 24AA1025 http://ww1.microchip.com/downloads/en/DeviceDoc/20001941L.pdf
8. 24M01 http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8812-SEEPROM-AT24CM01-Datasheet.pdf
9. 24M02 http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8828E-SEEPROM-AT24CM02-Datasheet.pdf
10. BL233_Data_Formatting.pdf https://www.i2cchip.com/pdfs/BL233_Data_Formatting.pdf
11. Examples Directory file:%ProgramFiles%%5CBEL%5CRealterm%5Cexamples%5CI2C_EEProm_Programming

12. I2C_EEProm_Scripts.zip https://www.i2cchip.com/pdfs/I2C_EEProm_Scripts.zip

13. SRecord http://srecord.sourceforge.net/

14. HexCSV2Dec https://realterm.sourceforge.io/#HexCSV2Dec

15. SED http://www.grymoire.com/Unix/Sed.html