TUSB6250 Bootcode Application Note

H. M. Tai *CS Peripheral*

ABSTRACT

The TUSB6250 bootcode is start-up code stored as a ROM image within the TUSB6250 device. An embedded microcontroller starts executing this code after power-on reset. Once the code is running, it loads application firmware from either an external memory device or the USB host controller. After the code has finished downloading, the bootcode releases its control to the application firmware.

This document describes in detail how the bootcode initializes the TUSB6250 device. In addition, several formats are listed as reference, including default USB descriptors, the I^2C device header format, and the USB host driver firmware downloading format. Examples are also given.

Contents

Figures

Tables

1 Overview

The TUSB6250 contains up to 32 KB of code space for the application firmware. During the power-up sequence, application firmware is downloaded from an external source to the internal code space. The bootcode is designed to search for the firmware source. Once the bootcode finds the source of firmware, it downloads it and then releases the control to the downloaded application firmware.

Application firmware can be downloaded either from an external I^2C EEPROM or from a USB host controller. If downloading the application firmware from an I^2C EEPROM is desired, use of a Cat3 I^2C EEPROM[1] is recommended because it normally has larger memory size. If downloading from a USB host controller is preferred, smaller I^2C EEPROM sizes like Cat2 devices can be used to store USB-related descriptors. The bootcode replaces its default USB descriptors with those stored in I^2C. Once the bootcode finishes downloading the application firmware from the USB host controller, it releases control to the downloaded firmware.

2 Bootcode Programming Flow

After power-on reset, the bootcode initializes the I^2C and USB registers, along with internal variables. It then checks whether the I^2C device contains the valid signature, 0x6250, which indicates that a properly formatted *header* exists in the device. A header contains USB descriptors and/or application firmware for the TUSB6250 to download.

If the I^2C device has the valid signature, the bootcode continues searching for descriptor blocks. If the checksum of a descriptor block is correct, the bootcode processes it. If the block contains a descriptor, it is used to overwrite the TUSB6250′s default. If a block contains binary firmware, it is downloaded after the entire header is processed. Application firmware can also exist in an auto-execution descriptor, in which case it is downloaded immediately and control is released to it.

If no firmware exists within the I^2C device, the bootcode connects to the USB host and waits for the host drivers to download application firmware. Once firmware is downloaded from the host, the bootcode releases control to it.

The flowcharts in Figure 1 and Figure 2 show the operational flow of the bootcode.

[1] See Section 8 for more detail.

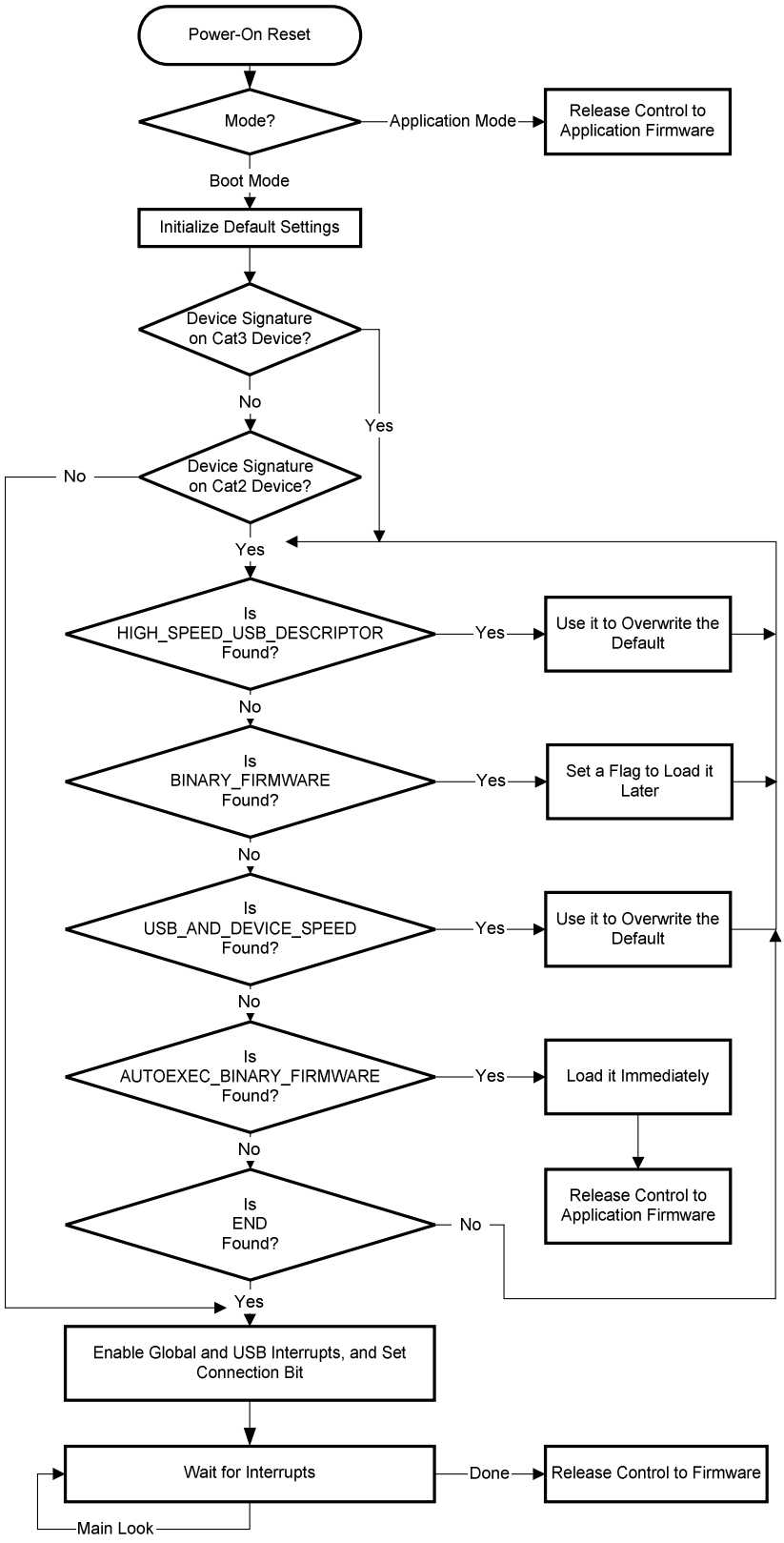

Figure 1. Bootcode Main

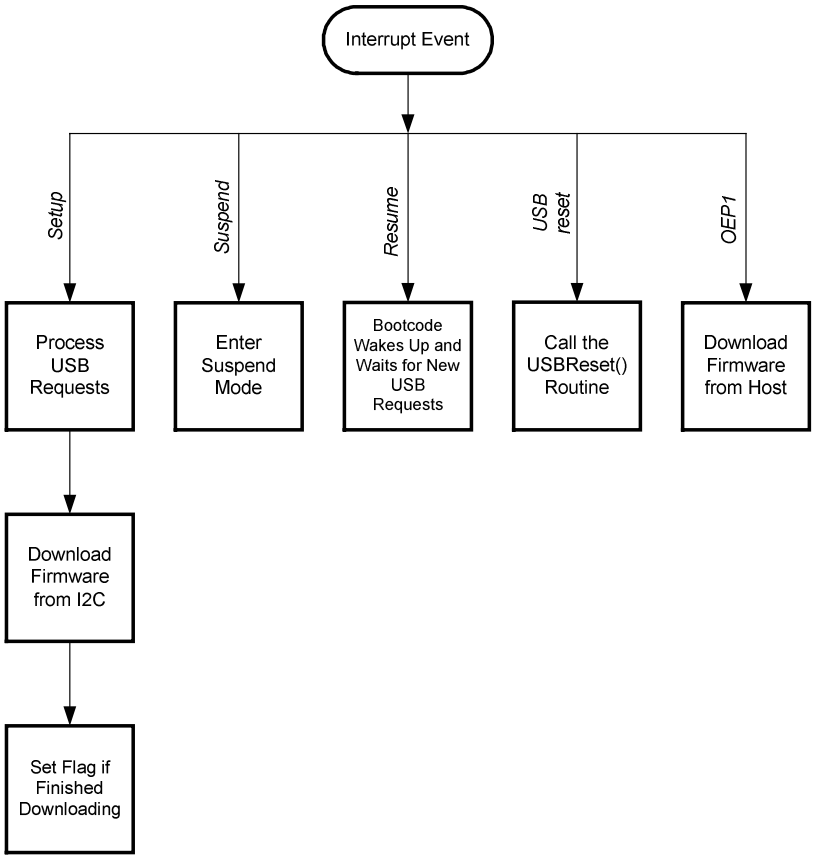

Figure 2. Bootcode Interrupt

The following mirrors the flowchart and provides additional detail.

- **Check operation mode.**
  - There are two operation modes: bootcode and application modes.
  - If in the application mode, the bootcode releases control to the application firmware. If not, the bootcode continues executing.

- **Initialize default settings.**
  - Call the CopyDefaultSettings() routine.
    - Set bootcode mode.
    - Set 40-KB RAM space with 32-KB code and 8-KB sector FIFO.
    - Set bDownloadFromI2c = FALSE.
  - Call the UsbDataInitialization() routine.
    - Set bFUNADR = 0.
    - Disconnect from the USB (bUSBCTL = 0x00).
    - Bootcode handles the USB reset.
    - Copy predefined device, configuration, and string descriptors to RAM.
    - Disable all endpoints and enable USB interrupts (SETUP, STPOW, RSTR, SUSPR, and RESUR).

- **Search for device signature.**
  - Check if a valid signature is found in I^2C. If not, stop processing I^2C.
    - Read 2 bytes from data address 0x0000 with type III (two-byte data address) EEPROM at device address 1. Stop searching type II EEPROM if the device signature is found.
    - Read 2 bytes from data address 0x0000 with type II (one-byte data address) EEPROM at device address 0.

- **Process descriptor blocks in I^2C EEPROM.**
  - Process each descriptor block read from I^2C until an end-of-header byte is read. The bootcode ignores any descriptor block with the wrong checksum.
    - If the descriptor block is a HIGH_SPEED_USB_DESCRIPTOR, the bootcode overwrites the default descriptors.
    - If the descriptor block is BINARY_FIRMWARE, the bootcode makes a note and loads the firmware later.
    - If the descriptor block is a USB_AND_DEVICE_SPEED, the bootcode overwrites the default descriptors.
    - If the descriptor block is AUTOEXEC_BINARY_FIRMWARE, the bootcode loads it and releases control to the firmware.
    - If the descriptor block is an END, the bootcode stops searching.

- **Enable global and USB interrupts and set connection bit to 1.**
  - Assert the nPWR100 pin low.
  - Set global interrupt bit EA = 1.
  - Set internal interrupt bit EI5 = 1.
  - Set connection bit CONT = 1.

- **Process interrupts and USB enumeration.**
  - Process USB requests and return USB descriptors accordingly.
  - Suspend interrupt
    - Set USBCTL_LPEN = 1 to enter the suspend mode. A USB reset wakes up the bootcode.
  - Resume interrupt
    - The bootcode wakes up and waits for new USB requests.
  - USB reset interrupt
    - Call the UsbReset() routine.
  - Setup interrupt
    - Process USB requests.

- **Download application firmware.**
  - Download application firmware from I^2C.
    - Firmware download is done in device enumeration.
    - Disable the global interrupt, EA = 0.
    - The bootcode releases control to the application firmware.
  - Download application firmware from USB host.
    - Host driver sends header and application firmware to output endpoint 1.
    - The bootcode releases control to the application firmware once it is downloaded successfully.

- **Release control to firmware.**
  - Update the USB configuration and interface number.
  - Copy bConfigurationNumber to the bBOOTTMP register.
  - Copy bInterfaceNumber to the bBOOTACC register.
  - Release control to the application firmware.

- **Application firmware**
  - Either disconnect from the bus or continue responding to USB requests.

3 Default Bootcode Descriptors

The bootcode contains the default device, configuration, and string descriptors. These are sufficient to allow the device to be enumerated on a USB host and to enable the downloading of firmware from it. If descriptors of these same types are provided on the I^2C device, they are used to overwrite the defaults.

These descriptors are designed to support firmware download from a USB host. They can be overwritten by new descriptors specified in an external I^2C device. Note that these default descriptors are for use in lab evaluation only, because they contain TI-specific information. A vendor ID must be acquired from USBIF. More information can be found at their website, http://www.usb.org.

3.1 Device Descriptor

The device descriptor contains information common to the entire USB device, such as class and vendor information. The host driver parses this descriptor and selects the appropriate device driver accordingly.

The defaults device descriptor uses 0x0451 for the vendor ID, because this is TI's assigned value. The arbitrary product ID is 0x6250. This descriptor is used in full- and high-speed connections. Table 1 shows the default descriptor.

Table 1. Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x12	Size of this descriptor in bytes
1	bDescriptorType	1	1	*Device* descriptor type
2	bcdUSB	2	0x0200	USB spec 2.0
4	bDeviceClass	1	0xFF	Device class is vendor-specific.
5	bDeviceSubClass	1	0	We have no subclasses.
6	bDeviceProtocol	1	0	We use no protocols.
7	bMaxPacketSize0	1	64	Maximum packet size for endpoint 0
8	idVendor	2	0x0451	USB-assigned vendor ID = TI
10	idProduct	2	0x6250	TI part number = TUSB6250
12	bcdDevice	2	0x0300	Device release number = 3.0
14	iManufacturer	1	2	Index of string descriptor describing manufacturer
15	iProduct	1	3	Index of string descriptor describing product
16	iSerialNumber	1	1	Index of string descriptor describing device serial number
17	bNumConfigurations	1	1	Number of possible configurations

3.2 Configuration Descriptor

The configuration descriptor contains the number of interfaces supported by this configuration, its power configuration, and electrical current consumption.

The bootcode declares only one interface running in bus-powered mode. It consumes up to 100 mA at boot time. Table 2 lists the configuration descriptor. This descriptor is used in full- and high-speed connections.

Table 2. Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	9	Size of this descriptor in bytes
1	bDescriptorType	1	2	*Configuration* descriptor type
2	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
4	bNumInterfaces	1	1	Number of interfaces supported by this configuration
5	bConfigurationValue	1	1	Value to use as an argument to the SetConfiguration() request to select this configuration
6	iConfiguration	1	0	Index of string descriptor describing this configuration.
7	bmAttributes	1	0x80	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-0: Reserved (reset to 0)
8	bMaxPower	1	0x32	This device consumes 100 mA.

3.3 Interface Descriptor

The interface descriptor contains the number of endpoints supported by this interface as well as the interface class, subclass, and protocol.

The bootcode supports only one endpoint and uses the vendor-specific class. Table 3 lists the interface descriptor. This descriptor is used in full- and high-speed connections.

Table 3. Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	9	Size of this descriptor in bytes
1	bDescriptorType	1	4	*Interface* descriptor type
2	bInterfaceNumber	1	0	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	bAlternateSetting	1	0	Value used to select alternate setting for the interface identified in the prior field
4	bNumEndpoints	1	1	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
5	bInterfaceClass	1	0xFF	The interface class is vendor-specific
6	bInterfaceSubClass	1	0	
7	bInterfaceProtocol	1	0	
8	iInterface	1	0	Index of string descriptor describing this interface

3.4 Endpoint Descriptor

The endpoint descriptor contains the type and size of the communication pipe supported by this endpoint.

The bootcode supports one output endpoint with the size of 64 bytes in full-speed connection and 512 bytes in high-speed connection. Table 4 and Table 5 show both endpoint descriptors.

Table 4. Full-Speed Output Endpoint-1 Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	7	Size of this descriptor in bytes
1	bDescriptorType	1	5	*Endpoint* descriptor type
2	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
3	bmAttributes	1	2	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
4	wMaxPacketSize	2	0x0040	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected
6	bInterval	1	0	Interval for polling endpoint for data transfers, expressed in milliseconds

Table 5. High-Speed Output Endpoint-1 Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	7	Size of this descriptor in bytes
1	bDescriptorType	1	5	*Endpoint* descriptor type
2	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
3	bmAttributes	1	2	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
4	wMaxPacketSize	2	0x0200	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected
6	bInterval	1	0	Interval for polling endpoint for data transfers, expressed in milliseconds

3.5 String Descriptor

String descriptors contain unicode strings. These are usually used to show manufacturer name, product model, and serial number in human-readable format.

The bootcode supports three strings: manufacturer name, product name, and serial number. Table 6 lists the string descriptors.

String descriptors are used in full- and high-speed connections. The first string descriptor must be reserved for a 12-digit serial number.

Table 6. String Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x04	Size of string-0 descriptor in bytes
1	bDescriptorType	1	0x03	*String* descriptor type
2	wLANGID[0]	2	0x0409	English
4	bLength	1	0x1A	Size of string-3 descriptor in bytes
5	bDescriptorType	1	0x03	*String* descriptor type
6	bString	2	r5H,0x00	Unicode. r0H, r0L to r5H, r5L are ASCII-character representations of the hexadecimal high- and low-byte contents of registers SERNUM0 to SERNUM5. This string is a 12-digit hex number.
8		2	r5L,0x00	
10		2	r4H,0x00	
12		2	r4L,0x00	
14		2	r3H,0x00	
16		2	r3L,0x00	
18		2	r2H,0x00	
20		2	r2L,0x00	
22		2	r1H,0x00	
24		2	r1L,0x00	
26		2	r0H,0x00	
28		2	r0L,0x00	
30	bLength	1	0x24	Size of string-1 descriptor in bytes
31	bDescriptorType	1	0x03	*String* descriptor type
32	bString	2	'T',0x00	Unicode, 'T' is first byte.
34		2	'e',0x00	This string is 'Texas Instruments'.
36		2	'x',0x00	
38		2	'a',0x00	
40		2	's',0x00	
42		2	' ',0x00	
44		2	'I',0x00	
46		2	'n',0x00	
48		2	's',0x00	
50		2	't',0x00	
52		2	'r',0x00	
54		2	'u',0x00	

Offset	Field	Size	Value	Description
56		2	'm',0x00	
58		2	'e',0x00	
60		2	'n',0x00	
62		2	't',0x00	
64		2	's',0x00	
66	bLength	1	0x2A	Size of string 2 descriptor in bytes
67	bDescriptorType	1	0x03	*String* descriptor type
68	bString	2	'T',0x00	Unicode, 'T' is first byte.
70		2	'U',0x00	This string is 'TUSB6250 Boot Device'
72		2	'S',0x00	
74		2	'B',0x00	
76		2	'6',0x00	
78		2	'2',0x00	
80		2	'5',0x00	
82		2	'0',0x00	
84		2	' ',0x00	
86		2	'B',0x00	
88		2	'o',0x00	
90		2	'o',0x00	
92		2	't',0x00	
94		2	' ',0x00	
96		2	'D',0x00	
98		2	'e',0x00	
100		2	'v',0x00	
102		2	'i',0x00	
104		2	'c',0x00	
106		2	'e',0x00	

3.6 Device Qualifier Descriptor

The device qualifier descriptor contains information about high-speed device operation at the other speed. The bootcode uses settings similar to those in the device descriptor.

Table 7. Device Qualifier Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x0A	Size of this descriptor in bytes
1	bDescriptorType	1	0x06	*Device* descriptor type
2	bcdUSB	2	0x0200	USB spec 2.0
4	bDeviceClass	1	0xFF	Device class is vendor-specific.
5	bDeviceSubClass	1	0	We have no subclasses.
6	bDeviceProtocol	1	0	We use no protocols.
7	bMaxPacketSize0	1	0x40	Max. packet size for endpoint 0
8	bNumConfigurations	1	1	Number of possible configurations
9	bReserved	1	0	Reserved for future use, must be 0.

3.7 Other-Speed Configuration Descriptor

The other-speed configuration descriptor contains configuration information about high-speed device operation at the other speed.

The bootcode uses settings similar to those in the configuration descriptor.

Table 8. Other-Speed Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	9	Size of this descriptor in bytes
1	bDescriptorType	1	7	Other-speed configuration type
2	wTotalLength	2	9	Total length of data returned for this configuration.
4	bNumInterfaces	1	1	Number of interfaces supported by this configuration
5	bConfigurationValue	1	1	Value to use as an argument to the SetConfiguration() request to select configuration
6	iConfiguration	1	0	Index of string descriptor describing this configuration.
7	bmAttributes	1	0x80	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-0: Reserved (reset to 0)
8	bMaxPower	1	0x32	This device consumes 100 mA.

4 Header Format in External I^{2}C EEPROM

The header refers to all the readable information contained within the I^{2}C device. It consists of a *device signature* and one or more *descriptor blocks*. The device signature is a two-byte hex number that allows the bootcode to verify that a valid header exists in the I^{2}C device. The TUSB6250 always expects this to be 0x6250, and if it does not find it, the bootcode assumes there is no readable information in the I^{2}C device.

Each descriptor block contains a piece of information that defines the USB device, including USB descriptors and application firmware. Each block consists of a *descriptor prefix*, which identifies the block type, and data content. The prefix is four bytes long and contains the type, size and checksum of the data content. The data content itself contains the USB descriptors, application code, or other information.

To locate a particular descriptor block, the bootcode header-processing routine always counts from the first descriptor block until the desired block number is reached. That is, it reads a descriptor prefix and calculates the location of the next block according to the data in the prefix, and does this successively to find the desired block.

Note that the TUSB6250 looks only to the I^2C interface to find a valid header. It supports both Cat2 and Cat3 I^2C devices, but expects a different device address depending on which category it is accessing. For Cat2 devices, it expects an I^2C device address of 0. For Cat3 devices, it expects an I^2C device address of 1. In both cases, the data address starts at 0x0000. (See Section 8 for more information.)

4.1 Header Components

A header consists of two components: device signature and descriptor block. They are described in Sections 4.1.1 and 4.1.2.

4.1.1 Device Signature

The device signature is stored in the first two bytes of an I^2C device. The order of these two bytes must be LSB first. The TUSB6250 device signature is 0x6250; therefore, the first byte must be 0x50 and the second byte must be 0x62. If the first two bytes of the external I^2C device are not 0x6250, the bootcode stops looking for descriptor blocks. Instead, it waits for a firmware download from the USB host. (See Section 4.2.5.)

4.1.2 Descriptor Block

Each descriptor block contains a prefix and data content. The prefix is always four bytes in size. It contains the data type, data size, and a data checksum. The data content corresponds to the information specified in prefix. It can be as small as one byte or as large as 65,535 bytes.

The next descriptor immediately follows the previous descriptor block. If no other descriptor block follows, an extra byte with value zero is added to indicate the end of header.

4.1.2.1 Descriptor Prefix

The first byte of the descriptor prefix is the data type. This provides the bootcode with enough information to process the data content. The second and third bytes are the size of data content. The second byte is the low byte of the size and the third byte is the high byte. The last byte is the 8-bit arithmetic checksum of the data content.

The TUSB6250 bootcode supports the following descriptor blocks.

- High-speed USB descriptors (including full-speed USB descriptors)
- Binary firmware
- Autoexec binary firmware
- USB and header speed

4.1.2.2 Data Content

Information stored in data content can be USB information, a firmware binary image, or some other type of data. The size of the content must be no less than 1 byte but less than or equal to 65,535 bytes.

4.1.2.3 Checksum

Each descriptor prefix contains a 1-byte checksum of the data content. This checksum is equal to the least significant byte of the sum of the data content. For example, if the sum of the data content is 0x127856, then the checksum is 0x56. If the checksum is wrong, the bootcode simply ignores the descriptor block.

4.2 Header Examples

The header can be specified in different ways. The following examples show how to arrange the header for different configurations of descriptors and firmware.

4.2.1 USB Descriptor Header

Table 9 contains hi-/full-speed USB string descriptors with no firmware contained within the header.

For a full-speed device, configuration descriptors must be listed first with high-speed descriptors following. String descriptors for both speeds are always at the end.

An extra byte of zero value is always appended to the last descriptor block. This is to indicate the end of the header. The bootcode stops processing descriptor blocks once it reaches this end-of-header marker. Application-specific information can be placed after this position in the I^2C device.

Seeing no firmware, the bootcode connects to the USB host after it finishes processing the header. When the host enumerates the TUSB6250-based device, it returns the descriptors shown in Table 9. (For mode information, see Chapter 9 of USB specification version 2.0.) Please note that some rows in the table have shading. This is only for ease of reading.

Table 9. USB Descriptors Header

Offset	Type	Size	Value	Description
0	Signature0	1	0x50	FUNCTION_PID_L
1	Signature1	1	0x62	FUNCTION_PID_H
2	Data type	1	0x08	Hi-speed USB descriptors
3	Data size (low byte)	1	0x80	Size of descriptor block
4	Data size (high byte)	1	0x00	
5	Checksum	1	0x4C	Checksum of following data content
6	bLength	1	0x12	Size of device descriptor
7	bDescriptorType	1	0x01	*Device* descriptor type
8	bcdUSB	2	0x0200	USB spec 2.0
10	bDeviceClass	1	0xFF	Device class is vendor-specific
11	bDeviceSubClass	1	0x00	No subclasses
12	bDeviceProtocol	1	0x00	No protocols
13	bMaxPacketSize0	1	0x40	Max. packet size for endpoint 0
14	idVendor	2	0x0451	USB-assigned vendor ID = TI
16	idProduct	2	0x6251	PID = 0x6251
18	bcdDevice	2	0x0100	Device release number = 1.0

Offset	Type	Size	Value	Description
20	iManufacturer	1	0x00	Index of string descriptor describing manufacturer
21	iProduct	1	0x00	Index of string descriptor describing product
22	iSerialNumber	1	0x00	Index of string descriptor describing device serial number
23	bNumConfigurations	1	0x01	Number of possible configurations
24	bLength	1	0x09	Size of configuration descriptor
25	bDescriptorType	1	0x02	*Configuration* descriptor type
26	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
28	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
29	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
30	iConfiguration	1	0x00	Index of string descriptor describing this configuration
31	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-0: Reserved (reset to 0)
32	bMaxPower	1	0x02	This device consumes 4 mA.
33	bLength	1	0x09	Size of interface descriptor
34	bDescriptorType	1	0x04	*Interface* descriptor type
35	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
36	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field
37	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
38	bInterfaceClass	1	0xFF	The interface class is vendor-specific.
39	bInterfaceSubClass	1	0x00	
40	bInterfaceProtocol	1	0x00	
41	iInterface	1	0x00	Index of string descriptor describing this interface
42	bLength	1	0x07	Size of endpoint descriptor
43	bDescriptorType	1	0x05	*Endpoint* descriptor type

Offset	Type	Size	Value	Description
44	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
45	bmAttributes	1	0x02	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
46	wMaxPacketSize	2	0x0040	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected
48	bInterval	1	0x00	Interval for polling endpoint for data transfers, expressed in milliseconds
49	bLength	1	0x12	Size of device descriptor
50	bDescriptorType	1	0x01	*Device* descriptor type
51	bcdUSB	2	0x0200	USB spec 2.0
53	bDeviceClass	1	0xFF	Device class is vendor-specific
54	bDeviceSubClass	1	0x00	No subclasses
55	bDeviceProtocol	1	0x00	No protocols
56	bMaxPacketSize0	1	0x08	Max. packet size for endpoint 0
57	idVendor	2	0x0451	USB-assigned vendor ID = TI
59	idProduct	2	0x6252	PID = 0x6252
61	bcdDevice	2	0x0100	Device release number = 1.0
63	iManufacturer	1	0x02	Index of string descriptor describing manufacturer
64	iProduct	1	0x03	Index of string descriptor describing product
65	iSerialNumber	1	0x01	Index of string descriptor describing device serial number
66	bNumConfigurations	1	0x01	Number of possible configurations
67	bLength	1	0x09	Size of configuration descriptor
68	bDescriptorType	1	0x02	*Configuration* descriptor type
69	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
71	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
72	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
73	iConfiguration	1	0x00	Index of string descriptor describing this configuration
74	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-0: Reserved (reset to 0)
75	bMaxPower	1	0x02	This device consumes 4 mA.

Offset	Type	Size	Value	Description
76	bLength	1	0x09	Size of interface descriptor
77	bDescriptorType	1	0x04	*Interface* descriptor type
78	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
79	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field
80	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
81	bInterfaceClass	1	0xFF	The interface class is vendor-specific.
82	bInterfaceSubClass	1	0x00	
83	bInterfaceProtocol	1	0x00	
84	iInterface	1	0x00	Index of string descriptor describing this interface
85	bLength	1	0x07	Size of endpoint descriptor
86	bDescriptorType	1	0x05	*Endpoint* descriptor type
87	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
88	bmAttributes	1	0x02	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
89	wMaxPacketSize	2	0x0200	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected.
91	bInterval	1	0x00	Interval for polling endpoint for data transfers, expressed in milliseconds
92	bLength	1	0x04	Size of string for this descriptor
93	bDescriptorType	1	0x03	*String* descriptor type
94	wLANGID[0]	2	0x0409	English
96	bLength	1	0x1A	Size of string for this descriptor
97	bDescriptorType	1	0x03	*String* descriptor type
98	bString	2	'1',0x00	'199912311030'
100		2	'9',0x00	
102		2	'9',0x00	
104		2	'9',0x00	
106		2	'1',0x00	
108		2	'2',0x00	
110		2	'3',0x00	
112		2	'1',0x00	
113		2	'1',0x00	
116		2	'0',0x00	
118		2	'3',0x00	
120		2	'0',0x00	

Offset	Type	Size	Value	Description
122	bLength	1	0x06	Size of string for this descriptor
123	bDescriptorType	1	0x03	*String* descriptor type
124	bString	2	'T',0x00	Unicode, 'T' is first byte.
126		2	'I',0x00	'TI' = 0x54, 0x49
128	bLength	1	0x06	Size of string-2 descriptor in bytes
129	bDescriptorType	1	0x03	*String* descriptor type
130	bString	2	'u',0x00	'uC'
132		2	'C',0x00	
134	Data Type	1	0x00	End of header

4.2.2 USB Descriptor With Binary Firmware

Table 10 contains USB-device configuration and string descriptors along with binary firmware. After the bootcode loads the USB descriptors from the I^2C device, it then connects to a USB host.

During the USB enumeration period, it returns the USB descriptors specified in the descriptor blocks below. Following enumeration, after *the Set Configuration request* is served, the bootcode begins releasing control to the binary firmware. The application firmware continues responding to any further USB requests once it gains control of execution.

Table 10. USB Descriptors With Binary Firmware

Offset	Type	Size	Value	Description
0	Signature0	1	0x50	FUNCTION_PID_L
1	Signature1	1	0x62	FUNCTION_PID_H
2	Data type	1	0x08	USB device descriptor
3	Data size (low byte)	1	0xC2	Device descriptor size
4	Data size (high byte)	1	0x00	
5	Checksum	1	0x29	Checksum of the following data
6	bLength	1	0x12	Size of device descriptor
7	bDescriptorType	1	0x01	*Device* descriptor type
8	bcdUSB	2	0x0200	USB spec 2.0
10	bDeviceClass	1	0xFF	Device class is vendor-specific
11	bDeviceSubClass	1	0x00	No subclasses
12	bDeviceProtocol	1	0x00	No protocols
13	bMaxPacketSize0	1	0x40	Max. packet size for endpoint 0
14	idVendor	2	0x0451	USB-assigned vendor ID = TI
16	idProduct	2	0x6253	Test PID
18	bcdDevice	2	0x0100	Device release number = 1.0
20	iManufacturer	1	0x00	Index of string descriptor describing manufacturer
21	iProduct	1	0x00	Index of string descriptor describing product
22	iSerialNumber	1	0x00	Index of string descriptor describing device serial number
23	bNumConfigurations	1	0x01	Number of possible configurations

Offset	Type	Size	Value	Description
24	bLength	1	0x09	Size of configuration descriptor
25	bDescriptorType	1	0x02	*Configuration* descriptor type
26	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
28	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
29	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
30	iConfiguration	1	0x00	Index of string descriptor describing this configuration
31	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-0: Reserved (reset to 0)
32	bMaxPower	1	0x02	This device consumes 4 mA.
33	bLength	1	0x09	Size of interface descriptor
34	bDescriptorType	1	0x04	*Interface* descriptor type
35	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
36	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field
37	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
38	bInterfaceClass	1	0xFF	The interface class is vendor-specific.
39	bInterfaceSubClass	1	0x00	
40	bInterfaceProtocol	1	0x00	
41	iInterface	1	0x00	Index of string descriptor describing this interface
42	bLength	1	0x07	Size of endpoint descriptor
43	bDescriptorType	1	0x05	*Endpoint* descriptor type
44	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
45	bmAttributes	1	0x02	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
46	wMaxPacketSize	2	0x0040	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected.

Offset	Type	Size	Value	Description
48	bInterval	1	0x00	Interval for polling endpoint for data transfers, expressed in milliseconds
49	bLength	1	0x12	Size of device descriptor
50	bDescriptorType	1	0x01	*Device* descriptor type
51	bcdUSB	2	0x0200	USB spec 2.0
53	bDeviceClass	1	0xFF	Device class is vendor-specific
54	bDeviceSubClass	1	0x00	No subclasses
55	bDeviceProtocol	1	0x00	No protocols
56	bMaxPacketSize0	1	0x08	Max. packet size for endpoint 0
57	idVendor	2	0x0451	USB-assigned vendor ID = TI
59	idProduct	2	0x6254	Test PID
61	bcdDevice	2	0x0100	Device release number = 1.0
63	iManufacturer	1	0x02	Index of string descriptor describing manufacturer
64	iProduct	1	0x03	Index of string descriptor describing product
65	iSerialNumber	1	0x01	Index of string descriptor describing device serial number
66	bNumConfigurations	1	0x01	Number of possible configurations
67	bLength	1	0x09	Size of configuration descriptor
68	bDescriptorType	1	0x02	*Configuration* descriptor type
69	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
71	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
72	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
73	iConfiguration	1	0x00	Index of string descriptor describing this configuration.
74	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-0: Reserved (reset to 0)
75	bMaxPower	1	0x02	This device consumes 4 mA.
76	bLength	1	0x09	Size of interface descriptor
77	bDescriptorType	1	0x04	*Interface* descriptor type
78	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
79	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field

Offset	Type	Size	Value	Description
80	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
81	bInterfaceClass	1	0xFF	The interface class is vendor-specific.
82	bInterfaceSubClass	1	0x00	
83	bInterfaceProtocol	1	0x00	
84	iInterface	1	0x00	Index of string descriptor describing this interface
85	bLength	1	0x07	Size of endpoint descriptor
86	bDescriptorType	1	0x05	*Endpoint* descriptor type
87	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
88	bmAttributes	1	0x02	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
89	wMaxPacketSize	2	0x0200	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected.
91	bInterval	1	0x00	Interval for polling endpoint for data transfers, expressed in milliseconds.
92	bLength	1	0x04	Size of string this descriptor
93	bDescriptorType	1	0x03	*String* descriptor type
94	wLANGID[0]	2	0x0409	English
96	bLength	1	0x1A	Size of string this descriptor
97	bDescriptorType	1	0x03	*String* descriptor type
98	bString	2	'6',0x00	'625020011101'
100		2	'2',0x00	
102		2	'5',0x00	
104		2	'0',0x00	
106		2	'2',0x00	
108		2	'0',0x00	
110		2	'0',0x00	
112		2	'1',0x00	
113		2	'1',0x00	
116		2	'1',0x00	
118		2	'0',0x00	
120		2	'1',0x00	
122	bLength	1	0x24	Size of string for this descriptor
123	bDescriptorType	1	0x03	*String* descriptor type
124	bString	2	'T',0x00	'Texas Instruments'
126		2	'e',0x00	
128		2	'x',0x00	
130		2	'a',0x00	
132		2	's',0x00	

Offset	Type	Size	Value	Description
134		2	' ',0x00	
136		2	'I',0x00	
138		2	'n',0x00	
140		2	's',0x00	
142		2	't',0x00	
144		2	'r',0x00	
146		2	'u',0x00	
148		2	'm',0x00	
150		2	'e',0x00	
152		2	'n',0x00	
154		2	't',0x00	
156		2	's',0x00	
158	bLength	1	0x2A	Size of string-2 descriptor in bytes
159	bDescriptorType	1	0x03	*String* descriptor type
160	bString	2	'T',0x00	'TUSB6250 Boot Device'
162		2	'U',0x00	
164		2	'S',0x00	
166		2	'B',0x00	
168		2	'6',0x00	
170		2	'2',0x00	
172		2	'5',0x00	
174		2	'0',0x00	
176		2	' ',0x00	
178		2	'B',0x00	
180		2	'o',0x00	
182		2	'o',0x00	
184		2	't',0x00	
186		2	' ',0x00	
188		2	'D',0x00	
190		2	'e',0x00	
192		2	'v',0x00	
194		2	'i',0x00	
196		2	'c',0x00	
198		2	'e',0x00	
200	Data Type	1	0x06	Binary firmware
201	Data Size (low byte)	1	0x0E	
202	Data Size (high byte)	1	0x00	
203	Check Sum	1	0xE8	Checksum of the following data
204	Program	1	0x02	Binary application code
205		1	0x20	LJMP 0x2003
206		1	0x03	
207		1	0x90	MOV DPTR, 0xF006
208		1	0xF0	
209		1	0x06	

Offset	Type	Size	Value	Description
210		1	0x74	MOV A, 20h
211		1	0x20	
212		1	0xF0	MOVX @DPTR, A
213		1	0x75	MOV P2, 27h
214		1	0xA0	
215		1	0x27	
216		1	0x80	SJMP 0xFD - loop forever
217		1	0xFD	
218	Data Type	1	0x00	End of header

4.2.3 Autoexec Binary Firmware

If an application requires firmware to be loaded prior to USB connection, the following header can be used. The bootcode loads the firmware and releases the control to the firmware directly without connecting to the USB. However, per USB specification requirement, any bus-powered USB device must connect to the host within the first 100 ms after drawing power from the USB cable. Table 11 shows an example of an autoexec binary firmware header.

Table 11. Autoexec Binary Firmware

Offset	Type	Size	Value	Description
0x0000	Signature0	1	0x50	FUNCTION_PID_L
0x0001	Signature1	1	0x62	FUNCTION_PID_H
0x0002	Data type	1	0x07	Autoexec binary firmware
0x0003	Data size (low byte)	1	0xZW	0xXYZW bytes of application code
0x0004	Data size (high byte)	1	0xXY	
0x0005	Checksum	1	0xMN	Checksum of the following firmware
0x0006	Program	0xXYZW		Binary application code
0x456D	Data type	1	0x00	End of header

4.2.4 USB and Header Speed With Autoexec Binary Firmware

The USB-and-header-speed descriptor block sets the USB connection speed and header device speed. To accelerate firmware downloading, 400 kHz can be specified.

If an application requires full speed only, USB[1:0] can be set to 01b so the TUSB6250 device only connects to USB at full speed. Table 12 shows a typical example using 400 kHz with a full-speed connection. Table 13 shows the bit definitions of the USB-device-and-header-speed descriptor block.

Table 12. USB and Header Speed With Autoexec Binary Firmware

Offset	Type	Size	Value	Description
0x0000	Signature0	1	0x50	FUNCTION_PID_L
0x0001	Signature1	1	0x62	FUNCTION_PID_H
0x0002	Data type	1	0x09	USB and header speed
0x0003	Data size (low byte)	1	0x01	Data content size: 1 byte

Offset	Type	Size	Value	Description
0x0004	Data size (high byte)	1	0x00	
0x0005	Checksum	1	0x12	Checksum
0x0006	Speed	1	0x12	Use full-speed and 400-kHz I^2C
0x0007	Data type	1	0x07	Autoexec binary firmware
0x0008	Data size (low byte)	1	0xZW	0xXYZW bytes of application code
0x0009	Data size (high byte)	1	0xXY	
0x000A	Checksum	1	0xMN	Checksum of the following firmware
0x000B	Program	0xXYZW		Binary application code
0x100B	Data type	1	0x00	End of header

Table 13. USB Device and Header Speed Definition

7	6	5	4	3	2	1	0
Imm	RSV	USB1	USB0	RSV	RSV	HDR1	HDR0

Bit	Name	Default	Function
1-0	HDR[1:0]	0	Header device speed HDR[1:0] = 00b Use bootcode default device speed. HDR[1:0] = 01b Use 100-kHz I^2C. HDR[1:0] = 10b Use 400-kHz I^2C. HDR[1:0] = 11b Reserved.
3-2	RSV	0	Reserved; must be 0 for feature compatibility.
5-4	USB[1:0]	00	USB connection speed. USB[0:0] = 00b Use bootcode default USB connection speed. USB[1:0] = 01b Use full speed. USB[1:0] = 10b Do not use this. USB[1:0] = 11b Do not use this.
6	RSV	0	Reserved; must be 0 for feature compatibility.
7	Imm	0	Used for early enumeration. 0: SetConfig release. The bootcode releases control to application firmware after it finishes the *Set Configuration* request from the host. 1: Imm release. The bootcode releases control to application firmware after it finishes downloading firmware and the current USB request.

4.2.5 USB and Header Speed With Binary Firmware

The bootcode normally takes more than 100 ms to download the entire application firmware from an I^2C device. It violates the USB specification if it continues downloading firmware without signaling connection to the host controller. To meet this requirement, the bootcode must connect to the USB and respond to host requests as well as download firmware from the I^2C device. Table 14 shows an example of a typical bus-powered device. In this example, the first descriptor block tells the bootcode to use a 400-kHz I^2C device for firmware download and *Imm* release for releasing control to the application firmware.

The bootcode takes advantage of standard USB requests with data stages to download application firmware. For example, the *Get Device* request can have one data and status stage. In each data stage, the bootcode uses around 460 ms to download the application firmware while the USB specification allows 500 ms. In each status stage, the bootcode uses around 46 ms for download while the specification allows 50 ms. Therefore, the bootcode has more than 500 ms to download firmware on each standard USB request with a data stage. It takes two requests to download 32 KB of code at 400-kHz speed. Because device enumeration takes at least two USB requests (*Get Device* and *Get Configuration Descriptor* requests), the bootcode can download the entire application code before it finishes device enumeration.

The bootcode provides two ways to release control to the application firmware. One is called *SetConfig* release. This is the default firmware release method. The other method is *Imm* release as shown in Table 14. If *SetConfig* release is chosen, the bootcode releases control to the application firmware after it serves the *Set Configuration* request. On the other hand, *Imm* release allows the bootcode release control to the application firmware after it finishes firmware download. It then releases control at the status stage.

Table 14. USB and Header Speed With Binary Firmware

Offset	Type	Size	Value	Description
0	Signature0	1	0x50	FUNCTION_PID_L
1	Signature1	1	0x62	FUNCTION_PID_H
2	Data type	1	0x09	USB and header speed
3	Data size (low byte)	1	0x01	Data content size: 1 byte
4	Data size (high byte)	1	0x00	
5	Checksum	1	0x82	
6	Imm. and 400 kHz	1	0x82	Imm. release with 400-kHz I^2C
7	Data type	1	0x08	USB device descriptor
8	Data size (low byte)	1	0xC2	Device descriptor size
9	Data size (high byte)	1	0x00	
10	Checksum	1	0x29	Checksum of data below
11	bLength	1	0x12	Size of device descriptor
12	bDescriptorType	1	0x01	*Device* descriptor type
13	bcdUSB	2	0x0200	USB spec 2.0
15	bDeviceClass	1	0xFF	Device class is vendor-specific
16	bDeviceSubClass	1	0x00	No subclasses
17	bDeviceProtocol	1	0x00	No protocols
18	bMaxPacketSize0	1	0x40	Max. packet size for endpoint 0
19	idVendor	2	0x0451	USB-assigned vendor ID = TI
21	idProduct	2	0x6253	Test PID
23	bcdDevice	2	0x0100	Device release number = 1.0
25	iManufacturer	1	0x00	Index of string descriptor describing manufacturer
26	iProduct	1	0x00	Index of string descriptor describing product
27	iSerialNumber	1	0x00	Index of string descriptor describing device serial number
28	bNumConfigurations	1	0x01	Number of possible configurations

Offset	Type	Size	Value	Description
29	bLength	1	0x09	Size of configuration descriptor
30	bDescriptorType	1	0x02	*Configuration* descriptor type
31	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
33	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
34	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
35	iConfiguration	1	0x00	Index of string descriptor describing this configuration.
36	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-D0: Reserved (reset to 0)
37	bMaxPower	1	0x02	This device consumes 4 mA.
38	bLength	1	0x09	Size of interface descriptor
39	bDescriptorType	1	0x04	*Interface* descriptor type
40	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
41	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field
42	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
43	bInterfaceClass	1	0xFF	The interface class is vendor-specific.
44	bInterfaceSubClass	1	0x00	
45	bInterfaceProtocol	1	0x00	
46	iInterface	1	0x00	Index of string descriptor describing this interface
47	bLength	1	0x07	Size of endpoint descriptor
48	bDescriptorType	1	0x05	*Endpoint* descriptor type
49	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
50	bmAttributes	1	0x02	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
51	wMaxPacketSize	2	0x0040	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected.

Offset	Type	Size	Value	Description
53	bInterval	1	0x00	Interval for polling endpoint for data transfers, expressed in milliseconds
54	bLength	1	0x12	Size of device descriptor
55	bDescriptorType	1	0x01	*Device* descriptor type
56	bcdUSB	2	0x0200	USB spec 2.0
58	bDeviceClass	1	0xFF	Device class is vendor-specific
59	bDeviceSubClass	1	0x00	No subclasses
60	bDeviceProtocol	1	0x00	No protocols
61	bMaxPacketSize0	1	0x08	Max. packet size for endpoint 0
62	idVendor	2	0x0451	USB-assigned vendor ID = TI
64	idProduct	2	0x6254	Test PID
66	bcdDevice	2	0x0100	Device release number = 1.0
68	iManufacturer	1	0x02	Index of string descriptor describing manufacturer
69	iProduct	1	0x03	Index of string descriptor describing product
70	iSerialNumber	1	0x01	Index of string descriptor describing device serial number
71	bNumConfigurations	1	0x01	Number of possible configurations
72	bLength	1	0x09	Size of configuration descriptor
73	bDescriptorType	1	0x02	*Configuration* descriptor type
74	wTotalLength	2	0x19 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, and endpoint) returned for this configuration.
76	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
77	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
78	iConfiguration	1	0x00	Index of string descriptor describing this configuration.
79	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to 1) D6: Self-powered D5: Remote wake-up is supported D4-D0: Reserved (reset to 0)
80	bMaxPower	1	0x02	This device consumes 4 mA.
81	bLength	1	0x09	Size of interface descriptor
82	bDescriptorType	1	0x04	*Interface* descriptor type
83	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
84	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field

Offset	Type	Size	Value	Description
85	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint 0). If this value is zero, this interface only uses the default control pipe.
86	bInterfaceClass	1	0xFF	The interface class is vendor-specific.
87	bInterfaceSubClass	1	0x00	
88	bInterfaceProtocol	1	0x00	
89	iInterface	1	0x00	Index of string descriptor describing this interface
90	bLength	1	0x07	Size of endpoint descriptor
91	bDescriptorType	1	0x05	*Endpoint* descriptor type
92	bEndpointAddress	1	0x01	Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint Bits 3-0: The endpoint number
93	bmAttributes	1	0x02	Bits 1-0: Transfer type 10 = Bulk 11 = Interrupt
94	wMaxPacketSize	2	0x0200	Max. packet size this endpoint is capable of sending or receiving when this configuration is selected.
96	bInterval	1	0x00	Interval for polling endpoint for data transfers, expressed in milliseconds.
97	bLength	1	0x04	Size of string this descriptor
98	bDescriptorType	1	0x03	*String* descriptor type
99	wLANGID[0]	2	0x0409	English
101	bLength	1	0x1A	Size of string this descriptor
102	bDescriptorType	1	0x03	*String* descriptor type
103	bString	2	'6',0x00	'625020011101'
105		2	'2',0x00	
107		2	'5',0x00	
109		2	'0',0x00	
111		2	'2',0x00	
113		2	'0',0x00	
115		2	'0',0x00	
117		2	'1',0x00	
119		2	'1',0x00	
121		2	'1',0x00	
123		2	'0',0x00	
125		2	'1',0x00	
127	bLength	1	0x24	Size of string for this descriptor
128	bDescriptorType	1	0x03	*String* descriptor type
129	bString	2	'T',0x00	'Texas Instruments'
131		2	'e',0x00	
133		2	'x',0x00	
135		2	'a',0x00	
137		2	's',0x00	

Offset	Type	Size	Value	Description
139		2	' ',0x00	
141		2	'I',0x00	
143		2	'n',0x00	
145		2	's',0x00	
147		2	't',0x00	
149		2	'r',0x00	
151		2	'u',0x00	
153		2	'm',0x00	
155		2	'e',0x00	
157		2	'n',0x00	
159		2	't',0x00	
161		2	's',0x00	
163	bLength	1	0x2A	Size of string-2 descriptor in bytes
164	bDescriptorType	1	0x03	*String* descriptor type
165	bString	2	'T',0x00	'TUSB6250 Boot Device'
167		2	'U',0x00	
169		2	'S',0x00	
171		2	'B',0x00	
173		2	'6',0x00	
175		2	'2',0x00	
177		2	'5',0x00	
179		2	'0',0x00	
181		2	' ',0x00	
183		2	'B',0x00	
185		2	'o',0x00	
187		2	'o',0x00	
189		2	't',0x00	
191		2	' ',0x00	
193		2	'D',0x00	
195		2	'e',0x00	
197		2	'v',0x00	
199		2	'i',0x00	
201		2	'c',0x00	
203		2	'e',0x00	
205	Data Type	1	0x06	Binary firmware
206	Data Size (low byte)	1	0x0E	
207	Data Size (high byte)	1	0x40	
208	Check Sum	1	0xWX	Checksum of application code
209	Program	16399	0xYZ	Binary application code
16608	Data Type	1	0x00	End of header

5 Host Driver Downloading Header Format

If it is preferable to download the firmware from a driver on the USB host, the host driver must follow the format in Table 15. TI's AppLoader driver can be used for this function. Users only need to provide a binary image of the application firmware for the driver.

If the checksum is wrong, the application code does not execute. Eventually, the watchdog timer resets the device.

Table 15. Host Driver Downloading Format

Offset	Type	Size	Value	Description
0x0000	Firmware size (low byte)	1	0xYZ	Application firmware size
0x0001	Firmware size (high byte)	1	0xWX	
0x0002	Checksum	1	0xJK	Checksum of binary application code
0x0003	Program	0xWXYZ		Binary application code

6 Bootcode Programming Considerations

6.1 Servicing USB Requests

For each USB request, the bootcode follows these steps to ensure proper operation:

1. Service the setup interrupt.
2. Determine the direction of the request by checking the MSB of bmRequestType field and set the USBCTL_DIR bit accordingly.
3. Decode the command
4. If another setup is pending, then return. Otherwise, serve the request.
5. Check again if another setup is pending. If so, go to Step 2; if not, proceed to Step 6.
6. Clear the interrupt source and then the VECINT register.
7. Exit the interrupt routine.

6.1.1 USB Requests

USB requests consist of three types of transfers: control-read-with-data-stage, control-write-without-data-stage, and control-write-with-data-stage transfers. These are shown in Figure 3. In each transfer, arrows indicate interrupts generated after receiving a setup packet or acknowledge packet. Only two of the transfers are supported by the bootcode.

Figure 3 and Figure 4 show the USB data flow and hardware and firmware response to USB requests. Figure 3 and Table 17 list responses to standard USB requests by the bootcode.

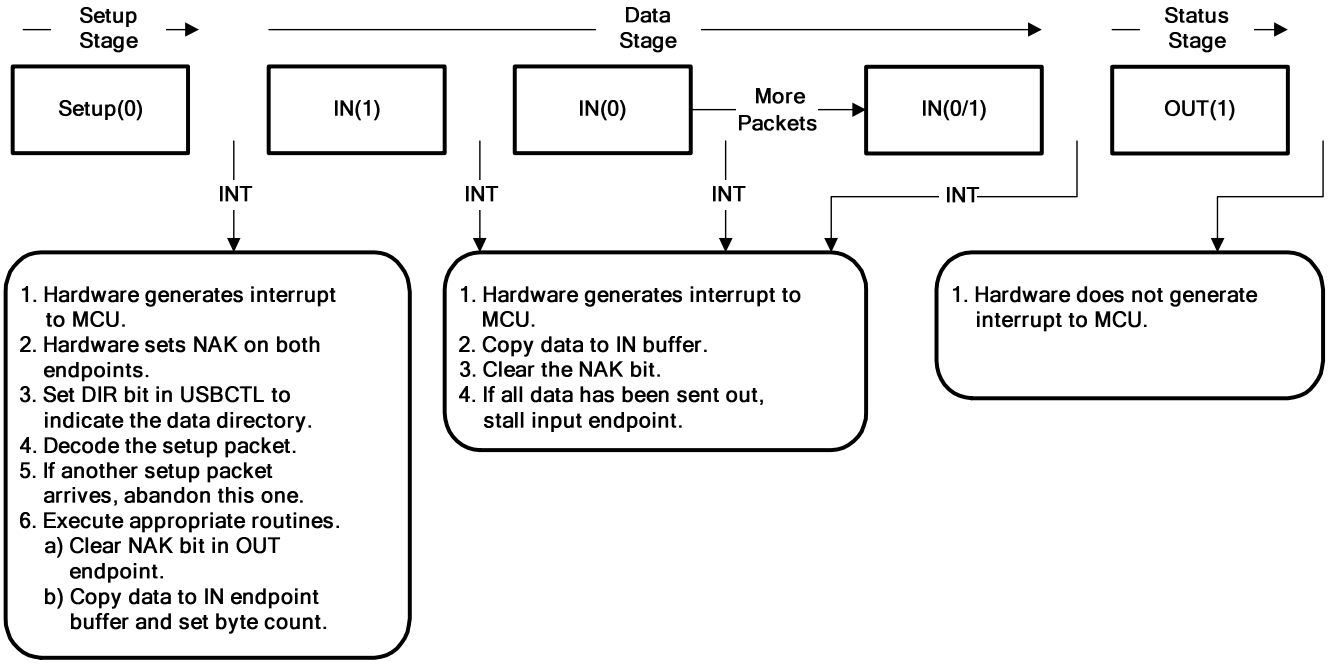

Figure 3. Control Read Transfer

Table 16. Bootcode Response to Control Read Transfer

Control Read	Action in Bootcode
Get status of device	Return power and remote wake-up settings
Get status of interface	Return two bytes of 0s
Get status of endpoint	Return endpoint status
Get descriptor of device	Return device descriptor
Get descriptor of configuration	Return configuration descriptor
Get descriptor of string	Return string descriptor
Get descriptor of interface	Stall
Get descriptor of endpoint	Stall
Get configuration	Return bConfiguredNumber value
Get interface	Return bInterfaceNumber value
Get device qualifier	Return device qualifier descriptor
Get other-speed configuration	Return other-speed configuration descriptor

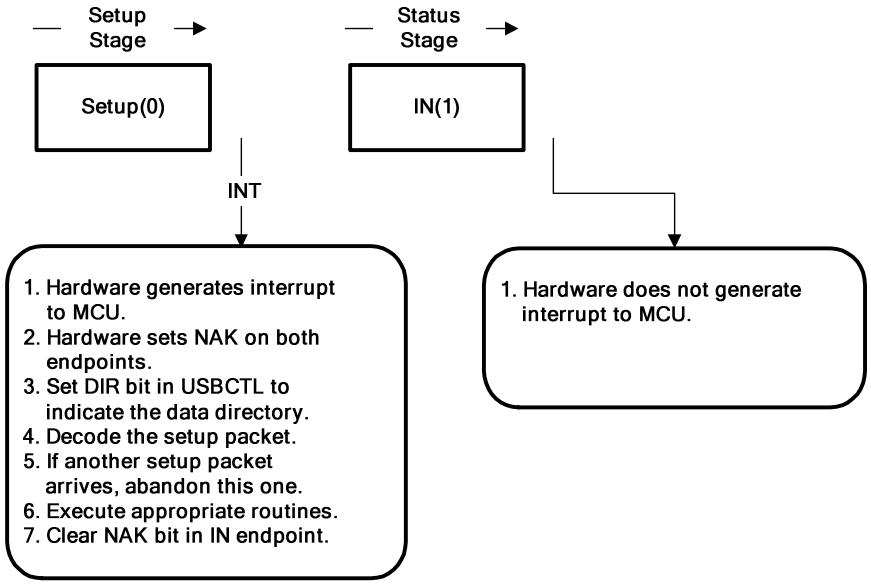

Figure 4. Control Write Transfer Without Data Stage

Table 17. Bootcode Response to Control Write Without Data Stage

Control Write Without Data Stage	Action in Bootcode
Clear feature of device	Clear remote wake-up
Clear feature of interface	Stall
Clear feature of endpoint	Clear endpoint stall
Set feature of device	Set remote wake-up
Set feature of interface	Stall
Set feature of endpoint	Stall endpoint
Set address	Set device address
Set descriptor	Stall
Set configuration	Set bConfiguredNumber
Set interface	Set bInterfaceNumber
Synchronize frame	Stall

6.2 Bootcode Interrupt Latency

Because the bootcode resides at 0x0000 to 0x1FFF, interrupt entry points are part of the bootcode and cannot be changed. Therefore, some latency is introduced for each interrupt. Table 18 lists the latency of each interrupt in instruction clock cycles.

An application firmware interrupt vector starts at address 0x2000. For example, the entry point of EX1 (0x0013) in the application firmware is at address 0x2013. The bootcode jumps to the address in application mode.

Table 18. Bootcode Interrupt Latency

Interrupt Source	Description	Interrupt Vector	Latency (Instruction Cycles)
EI6	RTK interrupt	0x2033	2
EI5	Internal INT5 (Used for internal vector interrupt)	0x202B	8
ES	8052 UART interrupt	0x2023	2
ET1	Timer-1 interrupt	0x201B	2
EX1	External $\overline{\text{INT1}}$	0x2013	2
ET0	Timer-0 interrupt	0x200B	2
EX0	External $\overline{\text{INT0}}$	0x2003	2

7 Header Utility

TI's header generation utility provides an easy way to generate a proper image file that can be programmed directly into an I^2C device. The inputs for this DOS utility are a configuration file and a binary object code file. The configuration file contains commands and comments and can be edited by a regular text editor.

The configuration file syntax is straightforward and can be observed in the following examples, listed for reference.

7.1 Usage

The executable filename of the header generation utility is "header.exe", and the usage is as follows.

```
header input_file_name output_file_name [-v] [-o offset]

where:
       -v                       set debug mode on (optional)
       -o offset                set the firmware offset (if NULL, default is 0x2000)
       input_file_name          configuration file name (for example, tusb6250.cfg)
       output_file_name         output image file name  (for example, tusb6250.bin)
```

The first parameter is the name of the configuration file. This file defines the overall structure of the image file. An example file named TUSB6250.cfg accompanies the utility and is an example of a typical configuration file.

The second parameter is the I^2C image output file name. If the file extension is .bin, the header utility generates the output file in binary format. If .hex is specified in the file extension, the header utility generates the image in hex format.

To deal with the fact that different compilers may begin executable code at different locations, the -o option can be used to adjust the firmware offset. For example, some compilers generate the firmware binary starting at 0x2000, including start-up code. Other compilers allocate the entry point of the start-up code, usually three bytes for a long jump, starting at address 0x0000, and the rest of the firmware starting at address 0x2000. The offset option can be used to accommodate this difference.

The following is a typical hex file firmware output from a C compiler. If the offset is set to 0x2000, it parses this file as follows. The first line contains three bytes beginning at address 0x0000. Because this is less than 0x2000, the header generator utility does not change the address when it parses it. (This is actually a long-jump instruction to address 0x2E64 in an 8051 microcontroller.) The second line contains 32 bytes starting at address 0x2111. Because this address is above 0x2000, the header generator utility subtracts 0x2000 at the address field. As a result, these 32 bytes are allocated starting at address 0x0111. The same rule applies to the rest of the lines.

The reason for this subtraction is to relocate the firmware within the I^2C image. The firmware still runs in logical code space beginning at address 0x2000 once it is loaded to internal SRAM, because it does not occupy code space from 0x0000 to 0x1FFF. Therefore, there is no need to save this code space in an I^2C device.

```
:0300000012642E59
:20211100FB90EB7FE0232354FCFC2430F582E434EBF583EAF0EBA3F090EB7E1267EAE0FC71
:17213100A3E0FD24ECEC6480347F500790EB7EE4F0A3F0D2AFC32267
:2021480075FB01C2AF90EB81E0232354FCFA242EF582E434EBF583AA83AB82E0F526A3E00D
:20216800F5274526605E8B828A83E4F0A3F090EB81E0232354FCFA2430F582E434EBF583DF
```

7.2 Command and Comment in Configuration File

A configuration file has three types of commands and one type of comment. All commands must be in capital letters. A comment starts with a semicolon (;) character. Any unsupported commands or characters are treated as an error. The utility stops immediately and prompts with a corresponding error message once it detects an error.

7.2.1 Command: DEVICE_NAME

The DEVICE_NAME command instructs the header utility to generate a proper signature. For example, the header utility produces 0x50, 0x62 for the first two bytes of an I^2C image file as the header signature if it reads *DEVICE_NAME = TUSB6250* from the configuration file.

7.2.2 Command: DESCRIPTOR_BLOCK

The descriptor block consists of data type, data length, data checksum, and data.

Data type, data length, and checksum are automatically generated by the utility. It is recommended that data be provided in the file as shown in the following example.

If the descriptor block is END, the utility terminates the program and simply attaches a byte of zero to the end of image file, as shown in the following example.

```
DESCRIPTOR_BLOCK USB_HIGH_SPEED_DESCRIPTOR
0x12, 0x00, 0x02, 0xff, 0x00, 0x00, 0x08
0x51, 0x04, 0x50, 0x62, 0x00, 0x01
0x01, 0x02, 0x03, 0x01
DESCRIPTOR_BLOCK END
```

7.2.3 Command: LOAD_BINARY_FILE or LOAD_HEX_FILE

LOAD_BINARY_FILE allows data to be stored in a binary file. Following is an example showing binary firmware loaded from the 6250.bin file.

```
DESCRIPTOR_BLOCK BINARY_FIRMWARE
LOAD_BINARY_FILE = 6250.bin
```

Following is another example showing AUTOEXEC_BINARY_FIRMWARE loaded from the 6250.hex file.

```
DESCRIPTOR_BLOCK AUTOEXEC_BINARY_FIRMWARE
LOAD_HEX_FILE = 6250.hex
```

7.2.4 Comment

A comment starts with the semicolon (;) character. It can be placed in any location of a configuration file. Any characters, letters, and commands following the semicolon on the same line are ignored. This comment feature is primarily designed for readability in a configuration file.

7.3 Example 1: 6250a.cfg Configuration File

This example shows a typical configuration file that supports USB device, configuration, and string descriptors. Firmware is downloaded from USB host.

```
;
; Commands:
;
; END                           // All Devices
; BINARY_FIRMWARE               // TUSB3410,6250
; AUTOEXEC_BINARY_FIRMWARE,     // TUSB3410,6250
; HISH_SPEED_USB_DESCRIPTOR,    // TUSB6250 (contains both full and high speed
;                                          descriptors)
; USB_AND_DEVICE_SPEED,         // TUSB6250
;
; '@' is used to generate SPACE character
;
;
;-------------------------------------------------------------------------
;
; Step 1 - Device signature
;
DEVICE_NAME = TUSB6250                                      ; USB to ATAPI device
;
;-------------------------------------------------------------------------
;
; Step 2 - Descriptor Blocks
;
DESCRIPTOR_BLOCK HIGH_SPEED_USB_DESCRIPTOR
;
; Order in this descriptor block:
; Full speed device descriptor
; Full speed Configuration descriptor
; High speed device descriptor
; High speed Configuration descriptor
; String descriptor
;
;
; Full speed device descriptor
;
0x12,               ; size of this descriptor in bytes
0x01,               ; device descriptor type
0x00, 0x02          ; USB spec 2.0
0xFF,               ; device class is vendor-specific
0x00                ; no sub-classes
0x00                ; no protocol
0x40                ; 64 bytes in endpoint 0
0x51, 0x04          ; vendor ID: 0x0451
0x51, 0x62          ; product ID: 0x6251
0x00, 0x01          ; device release number = 1.0
0x00,               ; index of string descriptor describing manufacturer
```

```
0x00,               ; index of string descriptor describing product
0x00,               ; index of string descriptor describing device's serial number
0x01,               ; number of possible configuration
;
; Full speed configuration descriptor
;
; configuration descriptor, size = 0x09
0x09,               ; bLength
0x02,               ; bDescriptorType - DESC_TYPE_CONFIG
0x19, 0x00,         ; wTotalLength = 25 bytes
0x01,               ; bNumInterfaces
0x01,               ; bConfigurationValue
0x00,               ; iConfiguration
0xE0,               ; bmAttributes, self-powered and remote wakeup
0x02,               ; Max. Power Consumption at 2mA unit
;
; Full speed interface descriptor, size = 0x09
;
0x09,               ; bLength
0x04,               ; bDescriptorType - DESC_TYPE_INTERFACE
0x00,               ; bInterfaceNumber
0x00,               ; bAlternateSetting
0x01,               ; bNumEndpoints
0xFF,               ; bInterfaceClass - vendor-specific
0x00,               ; bInterfaceSubClass, zero for hub
0x00,               ; bInterfaceProtocol
0x00,               ; iInterface
;
; Full speed endpoint descriptor, size = 0x07 for OEP1
;
0x07,               ; bLength
0x05,               ; bDescriptorType - DESC_TYPE_ENDPOINT
0x01,               ; bEndpointAddress; bit7=1 for IN, bits 3-0=1 for ep1
0x02,               ; bmAttributes, bulk transfer
0x40,0x00,          ; wMaxPacketSize, 512 bytes for high and 64 bytes for full speed
0x00                ; bInterval
;
;
; High speed device descriptor
;
0x12,               ; size of this descriptor in bytes
0x01,               ; device descriptor type
0x00, 0x02          ; USB spec 2.0
0xFF,               ; device class is vendor-specific
0x00                ; no sub-classes
0x00                ; no protocol
0x40                ; 64 bytes in endpoint 0
0x51, 0x04          ; vendor ID: 0x0451
0x52, 0x62          ; product ID: 0x6252
0x00, 0x01          ; device release number = 1.0
0x02,               ; index of string descriptor describing manufacturer
0x03,               ; index of string descriptor describing product
0x01,               ; index of string descriptor describing device's serial number
0x01,               ; number of possible configuration
;
; High speed configuration descriptor
;
; High speed configuration descriptor, size = 0x09
0x09,               ; bLength
0x02,               ; bDescriptorType - DESC_TYPE_CONFIG
```

```
0x19, 0x00,         ; wTotalLength = 25 bytes
0x01,               ; bNumInterfaces
0x01,               ; bConfigurationValue
0x00,               ; iConfiguration
0xE0,               ; bmAttributes, self-powered and remote wakeup
0x02,               ; Max. Power Consumption at 2mA unit
;
; High speed interface descriptor, size = 0x09
;
0x09,               ; bLength
0x04,               ; bDescriptorType - DESC_TYPE_INTERFACE
0x00,               ; bInterfaceNumber
0x00,               ; bAlternateSetting
0x01,               ; bNumEndpoints
0xFF,               ; bInterfaceClass - vendor-specific
0x00,               ; bInterfaceSubClass, zero for hub
0x00,               ; bInterfaceProtocol
0x00,               ; iInterface
;
; Endpoint Descriptor, size = 0x07 for OEP1
;
0x07,               ; bLength
0x05,               ; bDescriptorType - DESC_TYPE_ENDPOINT
0x01,               ; bEndpointAddress; bit7=1 for IN, bits 3-0=1 for ep1
0x02,               ; bmAttributes, bulk transfer
0x00,0x02,          ; wMaxPacketSize, 512 bytes for high speed and 64 bytes for full
                    ; speed
0x00                ; bInterval
;
; string descriptors
;
;
; string index 0, language ID
;
0x04,                               ; 4 bytes
0x03,                               ; DESC_TYPE_STRING
0x09,0x04,                          ; English 0x0409
;
; string index 1, serial number
;
0x1A,                               ; 26 bytes
0x03,                               ; DESC_TYPE_STRING
'1',0x00,'9',0x00,'9',0x00,'9',0x00,
'1',0x00,'2',0x00,'3',0x00,'1',0x00,
'1',0x00,'0',0x00,'3',0x00,'0',0x00,
;
; string index 2, manufacturer: Texas Instruments
;
0x06,                               ; Length of this string descriptor
0x03,                               ; DESC_TYPE_STRING,
'T',0x00,'I',0x00,
;
; string index 3, product name: TUSB6250
;
0x06,                               ; Length of this string descriptor
0x03,                               ; DESC_TYPE_STRING,
```

```
'u',0x00,'C',0x00
;
;-------------------------------------------------------------------------
;
; Step 3 - End of Descriptor Block
;
DESCRIPTOR_BLOCK END
```

7.4 Example 2: 6250b.cfg Configuration File

This example shows a typical configuration file that supports USB device, configuration, and string descriptors, as well as binary firmware. The bootcode updates its internal descriptors with those in this header, replacing its internal defaults, and then connects to the USB host. The bootcode handles requests from the host during the enumeration process. Once enumeration is complete, the bootcode releases control to the firmware.

```
;
; Commands:
;
; END                           // All Devices
; BINARY_FIRMWARE               // TUSB3410,6250
; AUTOEXEC_BINARY_FIRMWARE,     // TUSB3410,6250
; HISH_SPEED_USB_DESCRIPTOR,    // TUSB6250 (contains both full and high speed
;                                          descriptors)
; USB_AND_DEVICE_SPEED,         // TUSB6250
;
; '@' is used to generate SPACE character
;
;
;-------------------------------------------------------------------------
;
; Step 1 - Device signature
;
DEVICE_NAME = TUSB6250                                      ; USB to ATAPI device
;
;-------------------------------------------------------------------------
;
; Step 2 - Descriptor Blocks
;
DESCRIPTOR_BLOCK HIGH_SPEED_USB_DESCRIPTOR
;
; Order in this descriptor block:
; Full speed device descriptor
; Full speed Configuration descriptor
; High speed device descriptor
; High speed Configuration descriptor
; String descriptor
;
;
; Full speed device descriptor
;
0x12,               ; size of this descriptor in bytes
0x01,               ; device descriptor type
0x00, 0x02          ; USB spec 2.0
0xFF,               ; device class is vendor-specific
0x00                ; no sub-classes
0x00                ; no protocol
0x40                ; 64 bytes in endpoint 0
```

```
0x51, 0x04          ; vendor ID: 0x0451
0x53, 0x62          ; product ID: 0x6253
0x00, 0x01          ; device release number = 1.0
0x00,               ; index of string descriptor describing manufacturer
0x00,               ; index of string descriptor describing product
0x00,               ; index of string descriptor describing device's serial number
0x01,               ; number of possible configuration
;
; Full speed configuration descriptor
;
; configuration descriptor, size = 0x09
0x09,               ; bLength
0x02,               ; bDescriptorType - DESC_TYPE_CONFIG
0x19, 0x00,         ; wTotalLength = 25 bytes
0x01,               ; bNumInterfaces
0x01,               ; bConfigurationValue
0x00,               ; iConfiguration
0xE0,               ; bmAttributes, self-powered and remote wakeup
0x02,               ; Max. Power Consumption at 2mA unit
;
; Full speed interface descriptor, size = 0x09
;
0x09,               ; bLength
0x04,               ; bDescriptorType - DESC_TYPE_INTERFACE
0x00,               ; bInterfaceNumber
0x00,               ; bAlternateSetting
0x01,               ; bNumEndpoints
0xFF,               ; bInterfaceClass - vendor-specific
0x00,               ; bInterfaceSubClass, zero for hub
0x00,               ; bInterfaceProtocol
0x00,               ; iInterface
;
; Full speed endpoint descriptor, size = 0x07 for OEP1
;
0x07,               ; bLength
0x05,               ; bDescriptorType - DESC_TYPE_ENDPOINT
0x01,               ; bEndpointAddress; bit7=1 for IN, bits 3-0=1 for ep1
0x02,               ; bmAttributes, bulk transfer
0x40,0x00,          ; wMaxPacketSize, 512 bytes for high speed and 64 bytes for full
                    ; speed
0x00                ; bInterval
;
;
; High speed device descriptor
;
0x12,               ; size of this descriptor in bytes
0x01,               ; device descriptor type
0x00, 0x02          ; USB spec 2.0
0xFF,               ; device class is vendor-specific
0x00                ; no sub-classes
0x00                ; no protocol
0x40                ; 64 bytes in endpoint 0
0x51, 0x04          ; vendor ID: 0x0451
0x54, 0x62          ; product ID: 0x6254
0x00, 0x01          ; device release number = 1.0
0x02,               ; index of string descriptor describing manufacturer
0x03,               ; index of string descriptor describing product
0x01,               ; index of string descriptor describing device's serial number
0x01,               ; number of possible configuration
;
; High speed configuration descriptor
```

```
;
; High speed configuration descriptor, size = 0x09
0x09,               ; bLength
0x02,               ; bDescriptorType - DESC_TYPE_CONFIG
0x19, 0x00,         ; wTotalLength = 25 bytes
0x01,               ; bNumInterfaces
0x01,               ; bConfigurationValue
0x00,               ; iConfiguration
0xE0,               ; bmAttributes, self-powered and remote wakeup
0x02,               ; Max. Power Consumption at 2mA unit
;
; High speed interface descriptor, size = 0x09
;
0x09,               ; bLength
0x04,               ; bDescriptorType - DESC_TYPE_INTERFACE
0x00,               ; bInterfaceNumber
0x00,               ; bAlternateSetting
0x01,               ; bNumEndpoints
0xFF,               ; bInterfaceClass - vendor-specific
0x00,               ; bInterfaceSubClass, zero for hub
0x00,               ; bInterfaceProtocol
0x00,               ; iInterface
;
; Endpoint Descriptor, size = 0x07 for OEP1
;
0x07,               ; bLength
0x05,               ; bDescriptorType - DESC_TYPE_ENDPOINT
0x01,               ; bEndpointAddress; bit7=1 for IN, bits 3-0=1 for ep1
0x02,               ; bmAttributes, bulk transfer
0x00,0x02,          ; wMaxPacketSize, 512 bytes for high and 64 bytes for full speed
0x00                ; bInterval
;
; string descriptors
;
;
; string index 0, language ID
;
0x04,                               ; 4 bytes
0x03,                               ; DESC_TYPE_STRING
0x09,0x04,                          ; english 0x0409
;
; string index 1, serial number
;
0x1A,                               ; 26 bytes
0x03,                               ; DESC_TYPE_STRING
'6',0x00,'2',0x00,'5',0x00,'0',0x00,
'2',0x00,'0',0x00,'0',0x00,'1',0x00,
'1',0x00,'1',0x00,'0',0x00,'1',0x00,
;
; string index 2, manufacture: Texas Instruments
;
0x24,                               ; Length of this string descriptor
0x03,                               ; DESC_TYPE_STRING,
'T',0x00,'e',0x00,'x',0x00,'a',0x00,'s',0x00,'@',0x00,  ; use @ for SPACE
'I',0x00,'n',0x00,'s',0x00,'t',0x00,'r',0x00,'u',0x00,
'm',0x00,'e',0x00,'n',0x00,'t',0x00,'s',0x00,
;
; string index 3, product name: TUSB6250
;
```

```
0x2A,                               ; Length of this string descriptor
0x03,                               ; DESC_TYPE_STRING,
'T',0x00,'U',0x00,'S',0x00,'B',0x00,
'6',0x00,'2',0x00,'5',0x00,'0',0x00,'@',0x00,
'B',0x00,'o',0x00,'o',0x00,'t',0x00,'@',0x00,
'D',0x00,'e',0x00,'v',0x00,'i',0x00,'c',0x00,'e',0x00
;
;-------------------------------------------------------------------------
;
; Step 3 - Descriptor Blocks
;
DESCRIPTOR_BLOCK BINARY_FIRMWARE
;
LOAD_HEX_FILE = 6250.hex
;
;-------------------------------------------------------------------------
;
; Step 4 - End of Descriptor Block
;
DESCRIPTOR_BLOCK END
```

7.5 Example 3: tusb6250d.cfg Configuration File

This example shows a typical configuration file that supports autoexec binary firmware. Once the bootcode finds the AUTOEXEC_BINARY_FIRMWARE descriptor block, it starts downloading firmware from an external EEPROM without connecting to the USB. After the firmware is downloaded and the checksum is correct, the bootcode releases control to the firmware.

```
;
; Bootcode will load autoexec binary firmware from i2c at 400 kHz.
; The firmware will connect to USB using PID 0xFFFE and should run on full speed only.
;
; Commands:
;
; END                           // All Devices
; BINARY_FIRMWARE               // TUSB3410,6250
; AUTOEXEC_BINARY_FIRMWARE,     // TUSB3410,6250
; HISH_SPEED_USB_DESCRIPTOR,    // TUSB6250 (contains both full and high speed
;                                             descriptors)
; USB_AND_DEVICE_SPEED,         // TUSB6250
;
; '@' is used to generate SPACE character
;
;
;-------------------------------------------------------------------------------
;
; Step 1 - Device signature
;
DEVICE_NAME = TUSB6250                                          ; USB to ATAPI device
;
;-------------------------------------------------------------------------------
;
; Step 2 - Descriptor Blocks
;
DESCRIPTOR_BLOCK USB_AND_DEVICE_SPEED
0x12    ; Full speed and 400-kHz i2c
;
;
;-------------------------------------------------------------------------------
;
; Step 3 - Descriptor Blocks
;
DESCRIPTOR_BLOCK AUTOEXEC_BINARY_FIRMWARE
;
; This program will be used to test all endpoints.
;
LOAD_BINARY_FILE = UARTBOOT.bin
;
;-------------------------------------------------------------------------------
;
; Step 4 - End of Descriptor Block
;
DESCRIPTOR_BLOCK END
```

8 I²C EEPROM Categories

I²C EEPROM devices can be divided into three categories, based on data memory size. Category 1 (Cat1) devices have small memory, typically in the 16- to 128-byte range. Unlike Cat2 and Cat3 devices, it does not have a device address. Therefore, only one master and one slave can be on the bus. The bootcode does not support this type of device.

A Cat2 device has a memory size equal to or larger than that of a Cat1 device. This type of device normally has three address pins. Therefore, up to 8 devices can be on the same bus. Some devices with larger memory size might only have one or two address pins. Therefore, the rest of the address bits in the device address byte become part of the data address. Normally, a Cat2 device has a one-byte data address in the protocol. There could be from 8 to 11 addressing bits, depending on the vendor's implementation. Usually, a Cat2 device has no more than 2K bytes of data memory.

A Cat3 device has a larger memory size. This type of device normally also has three address pins. Therefore, up to 8 devices can be on the same bus. Some devices with larger memory sizes might only have one or two address pins. Therefore, the rest of the address bits in the device address byte become part of the data address. Normally, a Cat3 device has a two-byte data address in the protocol. There could be from 16 to 19 addressing bits, depending on the vendor's implementation.

The bootcode supports Cat2 and Cat3 devices.

8.1 CAT2 Device

Figure 5 shows the SDA line during access of a Cat2 device. Three bytes of data are written to addresses i, i+1, and i+2, respectively. P0, P1, and P2 represent device address pins A2, A1, and A0 on the chip. A value of 1010b in the high nibble of the device address byte is used for the memory device. This is defined in the I²C specification. One byte of data address, A7 to A0, is transmitted before the data bytes.

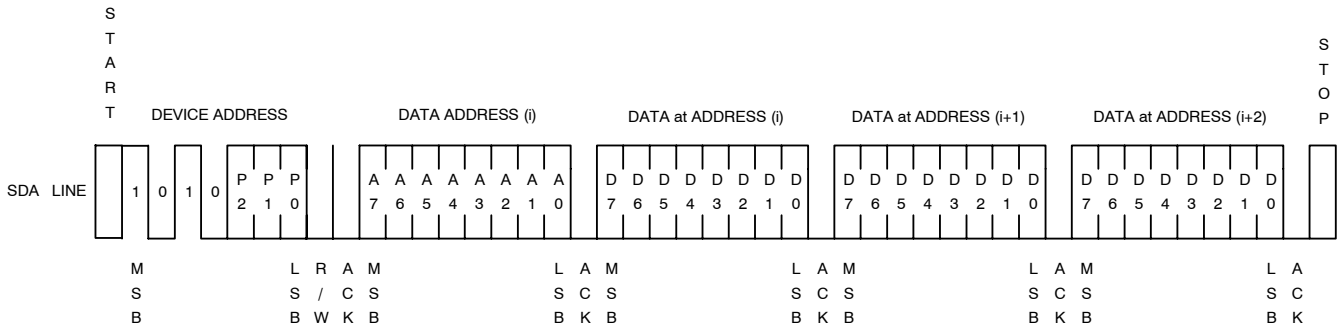

Figure 5. Writing Three Bytes of Data Starting at Address (i)

Figure 6 shows an example of a sequential read.

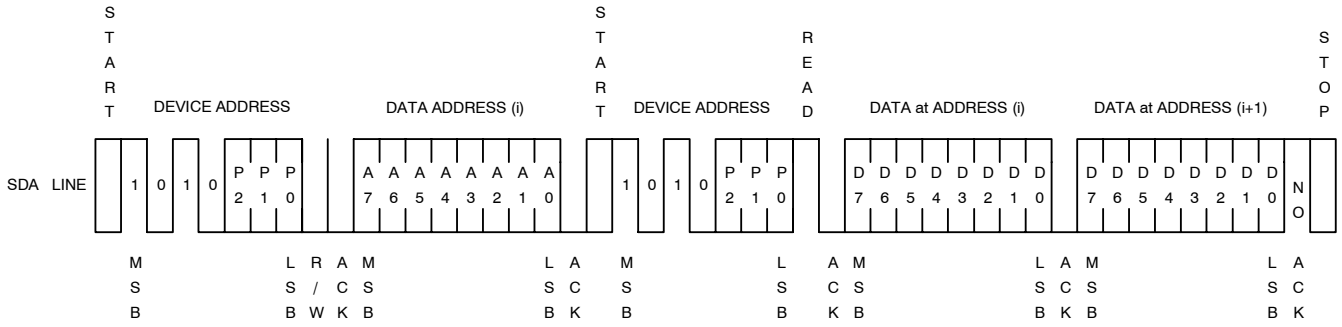

Figure 6. Reading Two Bytes of Data Starting at Address (i)

8.2 CAT3 Device

Figure 7 shows the SDA line during access of a Cat3 device. Two bytes of data are written to addresses i and i+1. Two bytes of data address, A15 to A0, are sent out before the data bytes.

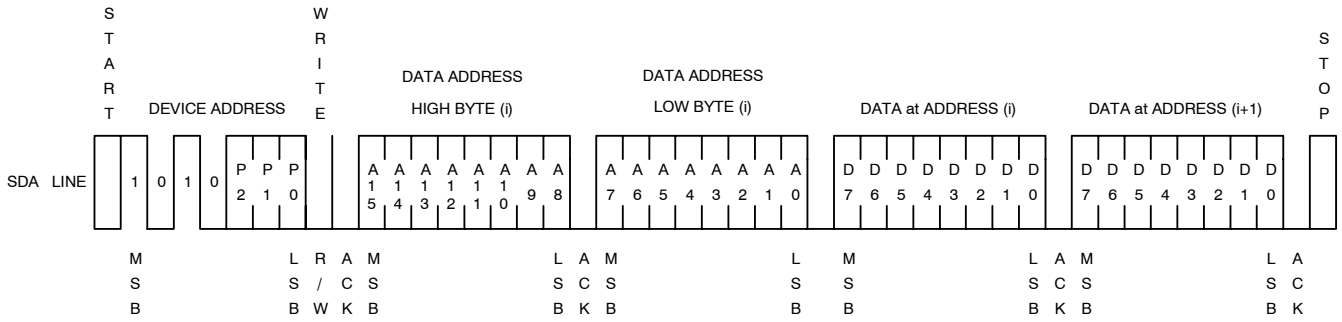

Figure 7. Writing Two Bytes of Data Starting at Address (i)

Figure 8 shows an example of a 1-byte read.

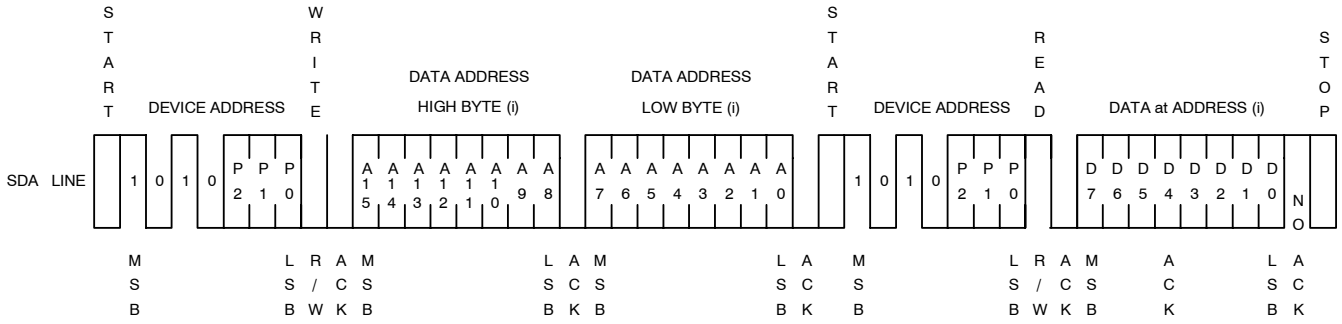

Figure 8. Reading One Byte of Data at Address (i)

References

1. *Universal Serial Bus Specification, Rev. 2.0*
2. *TUSB6250 USB 2.0 to ATA/ATAPI Bridge Controller Data Manual* (SLLS535)
3. Header generator utility (SLLC152.ZIP)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265