USBXPRESS™ PROGRAMMER'S GUIDE

Relevant Devices

This application note applies to the following devices:
C8051F320, C8051F321, CP2101, CP2102

1. Introduction

The Silicon Laboratories USBXpress™ Development Kit provides a complete host and device software solution for interfacing Silicon Laboratories C8051F32x and CP210x devices to the Universal Serial Bus (USB). No USB protocol or host device driver expertise is required. Instead, a simple, high-level Application Program Interface (API) for both the host software and device firmware is used to provide complete USB connectivity.

The USBXpress Development Kit includes Windows device drivers, Windows device driver installer and uninstallers, host interface function library (host API) provided in the form of a Windows Dynamic Link Library (DLL) and device firmware interface function library (C8051F32x devices only). The included device drivers and installation files support MS Windows 98SE/2000/XP.

2. Host API Functions

The host API is provided in the form of a Windows Dynamic Link Library (DLL). The host interface DLL communicates with the USB device via the provided device driver and the operating system's USB stack. The following is a list of the host API functions available:

`SI_GetNumDevices()`	- Returns the number of devices connected
`SI_GetProductString()`	- Returns a descriptor for a device
`SI_Open()`	- Opens a device and returns a handle
`SI_Close()`	- Cancels pending IO and closes a device
`SI_Read()`	- Reads a block of data from a device
`SI_Write()`	- Writes a block of data to a device
`SI_FlushBuffers()`	- Flushes the TX and RX buffers for a device
`SI_SetTimeouts()`	- Sets read and write block timeouts
`SI_GetTimeouts ()`	- Gets read and write block timeouts
`SI_CheckRXQueue()`	- Returns the number of bytes in a device's RX queue
`SI_SetBaudRate()`	- Sets the specified CP210x Baud Rate
`SI_SetBaudDivisor()`	- Sets the specified CP210x Baud Divisor Value
`SI_SetLineControl()`	- Sets the CP210x device Line Control
`SI_SetFlowControl()`	- Sets the CP210x device Flow Control
`SI_GetModemStatus()`	- Gets the CP210x device Modem Status
`SI_DeviceIOControl()`	- Interface for miscellaneous device control functions

In general, the user initiates communication with the target USB device(s) by making a call to *SI_GetNumDevices*. This call will return the number of target devices. This number is then used as a range when calling *SI_GetProductString* to build a list of device serial numbers or product description strings.

To access a device, it must first be opened by a call to *SI_Open* using an index determined from the call to *SI_GetNumDevices*. The *SI_Open* function will return a handle to the device that is used in all subsequent accesses. Data I/O is performed using the *SI_Write* and *SI_Read* functions. When I/O operations are complete, the device is closed by a call to *SI_Close*.

Additional functions are provided to flush the transmit and receive buffers (*SI_FlushBuffers*), set receive and transmit timeouts (*SI_SetTimeouts*), check the receive buffer's status (*SI_CheckRXQueue*) and miscellaneous device control (*SI_DeviceIOControl*).

For CP210x devices, functions are available to set the baud rate (*SI_SetBaudRate*); set the baud divisor (*SI_SetBaudDivision*); adjust the line control settings such as word length, stop bits and parity (*SI_SetLineControl*); set hardware handshaking, software handshaking and modem control signals (*SI_SetFlowControl*); and get modem status (*SI_GetModemStatus*).

Each of these functions are described in detail in the following sections. Type definitions and constants are defined in Appendix D.

2.1. SI_GetNumDevices

Description: This function returns the number of devices connected to the host.

Supported Devices: C8051F320/1, CP2101/2

Prototype: `SI_STATUS SI_GetNumDevices (LPDWORD NumDevices)`

Parameters:
1. NumDevices—Address of a DWORD variable that will contain the number of devices connected on return.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_NOT_FOUND or
SI_INVALID_PARAMETER

2.2. SI_GetProductString

Description: This function returns a null terminated serial number (S/N) string or product description string for the device specified by an index passed in DeviceNum. The index for the first device is 0 and the last device is the value returned by *SI_GetNumDevices* - 1.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_GetProductString (DWORD DeviceNum, LPVOID DeviceString,
DWORD Options)
```

Parameters:
1. DeviceNum—Index of the device for which the product description string or serial number string is desired.
2. DeviceString—Variable of type SI_DEVICE_STRING which will contain a NULL terminated device descriptor or serial number string on return.
3. Options—DWORD containing flags to determine if DeviceString contains a serial number string or product description string. See Appendix D for flags.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_NOT_FOUND or
SI_INVALID_PARAMETER

2.3. SI_Open

Description: Opens a device (using device number as returned by *SI_GetNumDevices*) and returns a handle which will be used for subsequent accesses.

Supported Devices: C8051F320/1, CP2101/2

Prototype: `SI_STATUS SI_Open (DWORD DeviceNum, HANDLE *Handle)`

Parameters:
1. DeviceNum—Device index. 0 for first device, 1 for 2nd, etc.
2. Handle—Pointer to a variable where the handle to the device will be stored. This handle will be used by all subsequent accesses to the device.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_NOT_FOUND or
SI_INVALID_HANDLE or
SI_INVALID_PARAMETER

2.4. SI_Close

Description: Closes an open device using the handle provided by *SI_Open*.

Supported Devices: C8051F320/1, CP2101/2

Prototype: `SI_STATUS SI_Close (HANDLE Handle)`

Parameters:
1. Handle—Handle to the device to close as returned by *SI_Open*.

Return Value: SI_STATUS = SI_SUCCESS or
SI_INVALID_HANDLE

2.5. SI_Read

Description: Reads the specified number of bytes into the specified buffer and retrieves the number of bytes read. Given valid input parameters, this function is blocking until the specified number of bytes become available or a timeout occurs (see section 2.8. *(SI_SetTimeouts)*).

Note: If timeouts are set to zero this function is not blocking and will immediately return the current number of bytes available in the device driver buffer. This may be less than the requested number of bytes (zero bytes if there is no data available) so make sure to check the read size return values in this scenario.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_Read (HANDLE Handle, LPVOID Buffer, DWORD NumBytesToRead,
DWORD *NumBytesReturned)
```

Parameters:

1. Handle—Handle to the device to read as returned by *SI_Open*.
2. Buffer—Address of a character buffer to be filled with read data.
3. NumBytesToRead—Number of bytes to read from the device into the buffer (0 - 64KBytes).
4. NumBytesReturned—Address of a DWORD which will contain the number of bytes actually read into the buffer on return.

Return Value: SI_STATUS = SI_SUCCESS or
SI_READ_ERROR or
SI_INVALID_REQUEST_LENGTH or
SI_INVALID_PARAMETER or
SI_RX_QUEUE_NOT_READY or
SI_INVALID_HANDLE

2.6. SI_Write

Description: Writes the specified number of bytes from the specified buffer to the device. Given valid parameters, this function is blocking until the write is successful, fails, or a timeout occurs. The write is successful when the device has accepted all of the data. If the write fails or a timeout occurs, SI_WRITE_ERROR is returned.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_Write (HANDLE Handle, LPVOID Buffer, DWORD NumBytesToWrite,
DWORD *NumBytesWritten)
```

Parameters:

1. Handle—Handle to the device to write as returned by *SI_Open*.
2. Buffer—Address of a character buffer of data to be sent to the device.
3. NumBytesToWrite—Number of bytes to write to the device (0 - 4096 bytes).
4. NumBytesWritten—Address of a DWORD which will contain the number of bytes actually written to the device.

Return Value: SI_STATUS = SI_SUCCESS or
SI_WRITE_ERROR or
SI_INVALID_REQUEST_LENGTH or
SI_INVALID_PARAMETER or
SI_INVALID_HANDLE

2.7. SI_FlushBuffers

Description: Flushes the receive buffer in the USB stack and the transmit buffer in the device.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_FlushBuffers (HANDLE Handle, BYTE FlushTransmit,
BYTE FlushReceive)
```

Parameters:
1. Handle—Handle to the device as returned by *SI_Open*.
2. FlushTransmit—Set to a non-zero value to flush the transmit buffer.
3. FlushReceive—Set to a non-zero value to flush the receive buffer.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_IO_FAILED or
SI_INVALID_HANDLE

2.8. SI_SetTimeouts

Description: Sets the read and write timeouts.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_SetTimeouts (DWORD ReadTimeout, DWORD WriteTimeout)
```

Parameters:
1. ReadTimeout—*SI_Read* operation timeout (in milliseconds).
2. WriteTimeout—*SI_Write* operation timeout (in milliseconds).

Return Value: SI_STATUS = SI_SUCCESS

2.9. SI_GetTimeouts

Description: Returns the current read and write timeouts.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_GetTimeouts (LPDWORD ReadTimeout, LPDWORD WriteTimeout)
```

Parameters:
1. ReadTimeout—*SI_Read* operation timeout (in milliseconds).
2. WriteTimeout—*SI_Write* operation timeout (in milliseconds).

Return Value: SI_STATUS = SI_SUCCESS or
SI_INVALID_PARAMETER

2.10. SI_CheckRXQueue

Description: Returns the number of bytes in the receive queue and a status value that indicates if an overrun (SI_QUEUE_OVERRUN) has occurred and if the RX queue is ready (SI_QUEUE_READY) for reading.

Supported Devices: C8051F320/1, CP2101/2

Prototype:
```
SI_STATUS SI_CheckRXQueue (HANDLE Handle, LPDWORD NumBytesInQueue,
LPDWORD QueueStatus)
```

Parameters:
1. Handle—Handle to the device as returned by *SI_Open*.
2. NumBytesInQueue—Address of a DWORD variable that contains the number of bytes currently in the receive queue on return.
3. QueueStatus—Address of a DWORD variable that contains the SI_RX_COMPLETE flag or the SI_RX_OVERRUN flag if an RX overrun occurred.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_IO_FAILED or
SI_INVALID_HANDLE or
SI_INVALID_PARAMETER

2.11. SI_SetBaudRate

Description: Sets the Baud Rate. Refer to the device data sheet for a list of Baud Rates supported by the device.

Supported Devices: CP2101/2

Prototype:
```
SI_STATUS SI_SetBaudRate (HANDLE Handle, DWORD dwBaudRate)
```

Parameters:
1. Handle—Handle to the device as returned by *SI_Open*.
2. dwBaudRate—A DWORD value specifying the Baud Rate to set.

Return Value: SI_STATUS = SI_SUCCESS or
SI_INVALID_BAUDRATE or
SI_INVALID_HANDLE

2.12. SI_SetBaudDivisor

Description: Sets the Baud Rate directly by using a specific divisor value. This function is obsolete; use SI_SetBaudRate instead.

Supported Devices: CP2101/2

Prototype:
```
SI_STATUS SI_GetBaudDivisor (HANDLE Handle, WORD wBaudDivisor)
```

Parameters:
1. Handle—Handle to the device as returned by *SI_Open*.
2. wBaudDivisor—A WORD value specifying the Baud Divisor to set.

Return Value: SI_STATUS = SI_SUCCESS or
SI_INVALID_BAUDRATE or
SI_INVALID_HANDLE

2.13. SI_SetLineControl

Description: Adjusts the line control settings: word length, stop bits and parity. Refer to the device data sheet for valid line control settings.

Supported Devices: CP2101/2

Prototype: `SI_STATUS SI_SetLineControl (HANDLE Handle, WORD wLineControl)`

Parameters:

1. Handle—Handle to the device as returned by *SI_Open*.
2. wLineControl—A WORD variable that contains the desired line control settings. Possible input settings are:

bits 0-3	**Number of Stop bits**
0 :	1 stop bit;
1 :	1.5 stop bits;
2 :	2 stop bits

bits 4-7	**Parity**
0 :	None
1 :	Odd
2 :	Even
3 :	Mark
4 :	Space

bits 8-15	**Number of bits per word**
	5, 6, 7, or 8

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_IO_FAILED or
SI_INVALID_HANDLE or
SI_INVALID_PARAMETER

2.14. SI_SetFlowControl

Description: Adjusts the following flow control settings: set hardware handshaking, software handshaking and modem control signals. See Appendix D for pin characteristic definitions.

Supported Devices: CP2101/2

Prototype:
```
SI_STATUS SI_SetFlowControl (HANDLE Handle, BYTE bCTS_MaskCode,
BYTE bRTS_MaskCode, BYTE bDTR_MaskCode, BYTE bDSRMaskCode,
BYTE bDCD_MaskCode, BYTE bFlowXonXoff)
```

Parameters:

1. Handle—Handle to the device as returned by *SI_Open*.
2. bCTS_MaskCode—The CTS pin characteristic must be:
   SI_STATUS_INPUT or
   SI_HANDSHAKE_LINE.
3. bRTS_MaskCod—The RTS pin characteristic must be:
   SI_HELD_ACTIVE,
   SI_HELD_INACTIVE,
   SI_FIRMWARE_CONTROLLED or
   SI_TRANSMIT_ACTIVE_SIGNAL.
4. bDTR_MaskCode—The DTR pin characteristic must be:
   SI_HELD_INACTIVE,
   SI_HELD_ACTIVE or
   SI_FIRMWARE_CONTROLLED.
5. bDSR_MaskCode—The DSR pin characteristic must be:
   SI_STATUS_INPUT or
   SI_HANDSHAKE_LINE.
6. bDCD_MaskCode—The DCD pin charactericstic must be:
   SI_STATUS_INPUT or
   SI_HANDSHAKE_LINE.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_IO_FAILED or
SI_INVALID_HANDLE or
SI_INVALID_PARAMETER

2.15. SI_GetModem Status

Description: Gets the Modem Status from the device. This includes the modem pin states.

Supported Devices: CP2101/2

Prototype: `SI_STATUS SI_GetModemStatus (HANDLE Handle, PBYTE ModemStatus)`

Parameters:

1. Handle—Handle to the device as returned by *SI_Open*.
2. lpbModemStatus—Address of a BYTE variable that contains the current states of the RS-232 modem control lines. The byte is defined as follows:

bit 0	DTR State
bit 1	RTS State
bit 4	CTS State
bit 5	DSR State
bit 6	RI State
bit 7	DCD State

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_IO_FAILED or
SI_INVALID_HANDLE or
SI_INVALID_PARAMETER

2.16. SI_DeviceIOControl

Description: Interface for any miscellaneous device control functions. A separate call to *SI_DeviceIOControl* is required for each input or output operation. A single call cannot be used to perform both an input and output operation simultaneously.

Supported Devices: C8051F320/1

Prototype:
```
SI_STATUS SI_DeviceIOControl (HANDLE Handle, DWORD IOControlCode,
LPVOID InBuffer, DWORD BytesToRead, LPVOID OutBuffer,
DWORD BytesToWrite, LPDWORD BytesSucceded)
```

Parameters:

1. Handle—Handle to the device as returned by *SI_Open*.
2. IOControlCode—Code to select control function.
3. InBuffer—Pointer to input data buffer.
4. BytesToRead—Number of bytes to be read into InBuffer.
5. OutBuffer—Pointer to output data buffer.
6. BytesToWrite—Number of bytes to write from OutBuffer.
7. BytesSucceded—Address of a DWORD variable that will contain the number of bytes read by a input operation or the number of bytes written by a output operation on return.

Return Value: SI_STATUS = SI_SUCCESS or
SI_DEVICE_IO_FAILED or
SI_INVALID_HANDLE

3. Device Interface Functions

The following set of device interface functions implements an application program interface (API) on the C8051F32x microcontroller. These functions provide a general I/O interface via the C8051F32x's USB controller without the need to manage low-level USB hardware details or protocol. The API is provided in the form of a library file pre-compiled under the Keil C51 tool chain. (Device firmware must be developed using the Keil Software C51 tool chain. The device interface functions available are:

`USB_Init()`	- Enables the USB interface
`Block_Write()`	- Writes a buffer of data to the host via the USB
`Block_Read()`	- Reads a buffer of data from the host via the USB
`Get_Interrupt_Source()`	- Indicates the reason for an API interrupt
`USB_Int_Enable()`	- Enables the API interrupts
`USB_Int_Disable()`	- Disables API interrupts
`USB_Disable()`	- Disables the USB interface
`USB_Suspend()`	- Suspend the USB interrupts

The API is used in an interrupt-driven mode. The user must provide an interrupt handler located at interrupt vector 0x10. This handler will be called upon at any USB API interrupt. Once inside this routine, a call to *Get_Interrupt_Source* is used to determine the source of the interrupt (this call also clears the pending interrupt flags).

Example ISR for Firmware API

```
void USB_API_TEST_ISR(void) interrupt 16
{
   BYTE INTVAL = Get_Interrupt_Source();

   if (INTVAL & TX_COMPLETE)
   {
      Block_Write(In_Packet, 8);
   }

   if (INTVAL & RX_COMPLETE)
   {
      Block_Read(Out_Packet, 8);
   }

   if (INTVAL & DEV_CONFIGURED)
   {
      // Initialize all analog peripherals here. This interrupt
      // tells the device that it can now use as much current as
      // specified by the MaxPower descriptor.
      Init();           // Note: example command, not part of the API
   }

   if (INTVAL & DEV_SUSPEND)
   {
      // Turn off all analog peripherals
      Turn_Off_All();   // Note: example command, not part of the API

      USB_Suspend();    // This function returns once resume
                        //    signalling is present.

      // Turn all anlalog peripherals back on
      Init();           // Note: example command, not part of the API
   }
}
```

3.1. USB_Init

Description: Enables the USB interface and the use of Device Interface Functions. On return, the USB interface is configured, including the USB clock and memory. Neither the system or USB clock configurations should be modified by user software after calling the *USB_Init* function. See Appendix A for a complete list of SFRs that should not be modified after *USB_Init* is called. In addition, C8051F32x interrupts are globally enabled on the return of this function. User software should not globally disable interrupts (set EA = 0) but should enable/disable user configured interrupts individually using the interrupt's source interrupt enable flag.

This function allows the user to specify the Vendor and Product IDs as well as a Manufacturer, Product Description and Serial Number string returned as part of the device's USB descriptor during the USB enumeration (connection).

Supported Devices: C8051F320/1

Prototype:
```
void USB_Init (int VendorID, int ProductID, uchar *ManufacturerStr,
uchar *ProductStr, uchar *SerialNumberStr, byte MaxPower,
byte PwAttributes, uint bcdDevice)
```

Parameters:

1. VendorID—Sixteen-bit Vendor ID to be returned to the host's Operating System during USB enumeration. Set to zero to use the default Silicon Laboratories Vendor ID (0x10C4).
2. ProductID—Sixteen-bit Product ID to be returned to the host's Operating System during USB enumeration. Set to zero to use the default (0x0000).
3. ManufacturerStr—Pointer to a character string. (See Appendix B for formatting.) Set to the NULL pointer to use the default string "Silicon Labs".
4. ProductStr—Pointer to a character string. (See Appendix B for formatting.) Set to the NULL pointer to the default string "USB API".
5. SerialNumberStr—Pointer to a character string. Set to the NULL pointer to use the default S/N string "1".
6. MaxPower—Specifies how much bus current a device requires. Set to one half the number of milliamperes required. The maximum allowed current is 500 milliamperes.
7. PwAttributes—Set bit 6 to 1 if the device is self-powered and to 0 if it is bus-powered. Set bit 5 to 1 if the device supports the remote wakeup feature. Bits 0 through 4 must be 0 and bit 7 must be 1.
8. bcdDevice—The device’s release number in BCD (binary-coded decimal) format. In BCD the upper byte represents the integer, the next four bits are tenths, and the final four bits are hundredths. Example: 2.13 is 0213h.

Return Value: None

3.2. Block_Write

Description: Writes a buffer of data to the host via the USB. Maximum block size is 4096 bytes. Returns the number of bytes actually written. This matches the parameter NumBytes unless an error condition occurs. A zero is returned if *Block_Write* is called with NumBytes greater than 4096.

Note: *Block_Write* is a non-blocking asynchronous call, thus overflow conditions may occur if the 64KByte PC side buffer is exceeded.

Supported Devices: C8051F320/1

Prototype: `unsigned int Block_Write (uchar *Buffer, unsigned int NumBytes)`

Parameters:
1. Buffer—Pointer to a memory location of data bytes to be written.
2. NumBytes—Number of bytes to write (1 - 4096).

Return Value: Returns an unsigned 16-bit value indicating the number of bytes actually written.

3.3. Block_Read

Description: Reads a buffer of data sent from the host via the USB. Maximum block size is 64 bytes. The block of data is copied from the USB interface to the memory location pointed to by Buffer. The USB RX buffer will be emptied on return regardless of whether or not the entire buffer was read by *Block_Read*. The maximum number of bytes to read from the USB RX buffer is specified in NumBytes. The number of bytes actually read (copied to Buffer) is returned by the function. A zero is returned if there are no bytes to read.

Note: Multiple calls to *Block_Read* may be needed depending on the size of the buffer sent to *SI_Write*.

Supported Devices: C8051F320/1

Prototype: `BYTE Block_Read (uchar *Buffer, uchar NumBytes)`

Parameters:
1. Buffer—Pointer to a memory location of data to be written.
2. NumBytes—Number of bytes to read (1 - 64).

Return Value: Returns an unsigned 8-bit value indicating the number of bytes actually read.

3.4. Get_Interrupt_Source

Description: Returns an 8-bit value indicating the reason for the API interrupt. This function should be called at the beginning of the user's interrupt service routine to allow the USB API to determine which events have occurred. Calling *Get_Interrupt_Source* also clears any USB API interrupt pending flags.

Supported Devices: C8051F320/1

Prototype: `uchar Get_Interrupt_Source (void)`

Parameters: None

Return Value: Returns an unsigned 8-bit code indicating the reason for the API interrupt. The return values are coded as follows:

0x00		No USB API Interrupts have occurred
0x01	USB_RESET	USB Reset Interrupt has occurred
0x02	TX_COMPLETE	Transmit Complete Interrupt has occurred
0x04	RX_COMPLETE	Receive Complete Interrupt has occurred
0x08	FIFO_PURGE	Command received (and serviced) from the host to purge the USB buffers
0x10	DEVICE_OPEN	Device Instance Opened on host side
0x20	DEVICE_CLOSE	Device Instance Closed on host side
0x40	DEV_CONFIGURED	Device has entered configured state
0x80	DEV_SUSPEND	USB suspend signaling present on bus

3.5. USB_Int_Enable

Description: A call to this function enables the USB API to generate interrupts. If enabled, a USB API interrupt is generated on the following API events:

1. A USB Reset has occurred.
2. A transmit scheduled by a call to *Block_Write* has completed.
3. The RX buffer is ready to be serviced by a call to *Block_Read*.
4. A command from the host has caused the USB buffers to be flushed.
5. A Device Instance has been opened or closed by the host.

The cause of the interrupt can be determined by a call to *Get_Interrupt_Source*. If USB API interrupts are enabled, the user must provide an interrupt service routine with the entry point located at the interrupt 0x16 vector (Address = 0x0083). When this function is called, control will transfer to the interrupt 16 handler within one ms, if any interrupts are currently pending.

Supported Devices: C8051F320/1

Prototype: `void USB_Int_Enable (void)`

Parameters: None

Return Value: None

3.6. USB_Int_Disable

Description: This function disables the USB API interrupt generation.

Supported Devices: C8051F320/1

Prototype: `void USB_Int_Disable (void)`

Parameters: None

Return Value: None

3.7. USB_Disable

Description: This function disables the USB interface and the use of Device Interface Functions. On return, the USB interface is no longer available and API interrupts are turned off. Any peripherals needed by the USB interface are also turned off, minimizing power consumption.

Supported Devices: C8051F320/1

Prototype: `void USB_Disable (void)`

Parameters: None

Return Value: None

3.8. USB_Suspend

Description: This function allows devices to meet the USB suspend current specification. This function should only be called when the API suspend interrupt is received. Turn off all unneccessary user peripherals before making this function call, and turn them back on after the call returns. This routine powers down all USB peripherals and suspends the internal oscillator until USB resume signaling occurs. Once this happens, the USB peripherals are turned back on, and the function call returns to user code. All USB devices must support this feature by reducing their total power consumption to 500 µA or less.

Supported Devices: C8051F320/1

Prototype: `void USB_Suspend (void)`

Parameters: None

Return Value: None

APPENDIX A—SFRS THAT SHOULD NOT BE MODIFIED AFTER CALLING USB_INIT

The following is a list of SFRs configured by the API. These should not be altered at any time after the first call to *USB_Init*. Most of these SFRs are dedicated to the USB peripheral on the chip and should be of no concern to the programmer anyway. The only exception to this rule is the CLKSEL SFR, which is used for choosing both the system clock source and USB clock source. Care should be used to OR in the system clock desired into Bits 1-0, so as not to disturb Bits 6-4, which are the USB clock selection bits.

Off-Limits USB SFRs—USB0XCN, USB0ADR, and USB0DAT

Off-Limits Other SFRs—OSCICN, CLKMUL, CLKSEL (Only Bits 4-6 are off-limits) these three SFRs are used to configure the internal oscillator to 12 MHz, engage 4x multiplier to 48 MHz, and to use that clock for the USB core. They cannot be altered for the API to function.

APPENDIX B—FORMAT OF USER-DEFINED PRODUCT DESCRIPTION AND SERIAL NUMBER STRINGS

It is possible for the API to use strings defined and allocated in user firmware instead of the API default strings. The syntax for defining and using custom strings is:

```
unsigned char CustomString[]={number of string elements,0x03,'A',0,'B',0,'C',0...'Z',0};
```

The number of string elements = number of letters * 2 + 2, since every letter needs to be separated from the next by zeros, and USB requires that the first element be the length, and the second element is 0x03, meaning string descriptor type. This sounds harder than it is, for example:

```
//ABC Inc
unsigned char CustomString1[]={16,0x03,'A',0,'B',0,'C',0,' ',0,'I',0,'n',0,'c',0};

//Widget
unsigned char CustomString2[]={14,0x03,'W',0,'i',0,'d',0,'g',0,'e',0,'t',0};

//12345
unsigned char CustomString3[]={12,0x03,'1',0,'2',0,'3',0,'4',0,'5',0};
```

Then, if the Vendor ID and Product ID were 0xABCD and 0x1123, the call to *USB_Init* would be

```
USB_Init (0xABCD, 0x1123, CustomString1, CustomString2, CustomString3);
```

Note: It is useful to use the code keyword preceding the CustomString definitions, so that the strings are located in code space.

APPENDIX C—FIRMWARE LIBRARY MEMORY-MODEL CONCERNS

The firmware API library was created using the small memory model. Using this library in a project with a default memory model of large or compact can cause warnings to occur, depending on warning level settings. To avoid this, set the default memory model to small, and over-ride this setting by defining each function with the large compiler keyword.

APPENDIX D—TYPE DEFINITIONS (FROM C++ HEADER FILE SiUSB.H)

```
// GetProductString function flags
#define SI_RETURN_SERIAL_NUMBER        0x00
#define SI_RETURN_DESCRIPTION          0x01

// Return codes
#define  SI_SUCCESS                    0x00
#define  SI_DEVICE_NOT_FOUND           0xFF
#define  SI_INVALID_HANDLE             0x01
#define  SI_READ_ERROR                 0x02
#define  SI_RX_QUEUE_NOT_READY         0x03
#define  SI_WRITE_ERROR                0x04
#define  SI_INVALID_PARAMETER          0x06
#define  SI_INVALID_REQUEST_LENGTH     0x07
#define  SI_DEVICE_IO_FAILED           0x08
#define  SI_INVALID_BAUDRATE           0x09

// RX Queue status flags
#define  SI_RX_NO_OVERRUN              0x00
#define  SI_RX_OVERRUN                 0x01
#define  SI_RX_READY                   0x02

// Buffer size limits
#define  SI_MAX_DEVICE_STRLEN          256
#define  SI_MAX_READ_SIZE              4096*16
#define  SI_MAX_WRITE_SIZE             4096

// Type definitions
typedef  int   SI_STATUS;
typedef  char  SI_DEVICE_STRING[SI_MAX_DEVICE_STRLEN];

// Input and Output pin Characteristics
#define  SI_HELD_INACTIVE              0x00
#define  SI_HELD_ACTIVE                0x01
#define  SI_FIRMWARE_CONTROLLED        0x02
#define  SI_RECEIVE_FLOW_CONTROL       0x02
#define  SI_TRANSMIT_ACTIVE_SIGNAL     0x03

#define  SI_STATUS_INPUT               0x00
#define  SI_HANDSHAKE_LINE             0x01

//Common variable type definitions used
typedef  unsigned long     DWORD;
typedef  int               BOOL;
typedef  unsigned char     BYTE;
typedef  unsigned short    WORD;
typedef  BYTE near         *PBYTE;
typedef  DWORD near        *PDWORD;
typedef  DWORD far         *LPDWORD;
typedef  void far          *LPVOID;
```

DOCUMENT CHANGE LIST

Revision 1.4 to Revision 1.5

- Added “This function is obsolete; use SI_SetBaudRate instead” to "SI_SetBaudDivisor" on page 6.
- Removed CP2101/2 from the supported devices list in "SI_DeviceIOControl" on page 9.

CONTACT INFORMATION

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: MCUinfo@silabs.com
Internet: www.silabs.com

SILICON LABORATORIES