USING THE C8051FXXX ON-CHIP INTERFACE UTILITIES DLL

Relevant Devices

This application note applies to all Silicon Laboratories C8051Fxxx devices.

1. Introduction

The Interface Utilities DLL (Dynamic Link Library) provides the following functionality: download an Intel hex file to Flash; connect and disconnect to a C8051Fxxx processor; “Run” and “Halt” the microprocessor; and read and write internal, external, and code memory spaces. All this functionality is provided via a PC’s COM port or USB port and a Silicon Laboratories Debug Adapter.

When using C++, the functions described in this document must be declared as imported functions. Add the **C++ Prototype** to the header file (*.h) of the source file (*.cpp) from which the function will be called.

The procedures and guidelines presented in this document illustrate how to link the Interface Utilities DLL to a client executable. A DLL is not a standalone application. It is a library of exported functions that are linked at run-time and called by a Microsoft Windows® application. To write a client using Visual Basic please refer to Section 14 on page 17 for more information.

An example Interface Utilities application (including program source code) is provided along with the Interface Utilities DLL as a means of testing or using the Utilities DLL as a base application. The Interface Utilities application uses implicit linking, which requires the DLL to be placed in a specific directory (see Section 11 on page 16).

2. Files and Compatibility

The latest versions of the “SiUtil.dll” and “SiUtil.lib”, are available at www.silabs.com. The DLL is a Win32 MFC Regular DLL meaning it uses the Microsoft Foundation Class libraries. It can be loaded by any Win32 programming environment, and only exports ‘C’ style functions. Two versions are currently available, one in which the MFC (Microsoft Foundations Classes) library is statically linked in the DLL, and another version in which the MFC library is dynamically linked in the DLL.

The statically linked MFC version includes a copy of all the MFC library code it needs and is thus self contained. No external MFC linking is required. With the MFC library code included, the statically linked DLL will be larger in size than the dynamically linked version.

The dynamically linked MFC version is smaller in size than the static version. However, the dynamically linked DLL requires that files “MFC42.dll” and “MSVCRT.dll” be present on the target machine. This is not a problem if the client program is dynamically linked to the same version (Version 4.2) or newer of the MFC library (i.e., uses MFC as a shared library). The required MFC DLLs “MFC42.dll” and “MSVCRT.dll” are provided along with the dynamically linked MFC version of the Utility Programmer DLL. Do not replace equivalent or newer versions of these files if they are already present on the target machine.

3. Communications Functions

The following communication functions are available for use in the Interface Utilities DLL.

`Connect()`	- Connect to a target C8051Fxxx device using a Serial Adapter.
`Disconnect()`	- Disconnect from a target C8051Fxxx device using a Serial Adapter.
`ConnectUSB()`	- Connect to a target C8051Fxxx device using a USB Debug Adapter.
`DisconnectUSB()`	- Disconnect from a target C8051Fxxx device using a USB Debug Adapter.
`Connected()`	- Returns the connection state of the target C8051Fxxx device.

3.1. Connect()

Description: This function is used to connect to a target C8051Fxxx device using a Serial Adapter. Establishing a valid connection is necessary for all memory operations to succeed.

Supported Debug Adapters: Serial Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall Connect(int nComPort=1,
int nDisableDialogBoxes=0, int nECprotocol=0, int nBaudRateIndex=0);
```

Parameters:

1. *nComPort*—Target COM port to establish a connection. The default is '1'.
2. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is '0'.
3. *nECprotocol*—Connection protocol used by the target device; JTAG (0) or Silicon Laboratories 2-Wire (C2) (1). The default is '0'. The C2 interface is used with C8051F3xx derivative devices and the JTAG interface is used with C8051F0xx, C8051F1xx, and C8051F2xx derivative devices.
4. *nBaudRateIndex*—Target baud rate to establish a connection; Autobaud detection (0), 115200 (1), 57600 (2), 38400 (3), 9600 (4) or 2400 (5). The default is '0'.

Return Value: See Table 1 on page 16.

3.2. Disconnect()

Description: This function is used to disconnect from a target C8051Fxxx device using a Serial Adapter.

Supported Debug Adapters: Serial Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall Disconnect(int nComPort=1);
```

Parameters:

1. *nComPort* - COM port in use by current connection. The default is '1'.

Return Value: See Table 1 on page 16.

3.3. ConnectUSB()

Description: This function is used to connect to a target C8051Fxxx device using a USB Debug Adapter. Establishing a valid connection is necessary for all memory operations to succeed.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall ConnectUSB(
const char * sSerialNum="", int nECprotocol=0, int nPowerTarget=0,
int nDisableDialogBoxes=0);
```

Parameters:

1. *sSerialNumber*—The serial number of the USB Debug Adapter. See Section 8 for information on obtaining the serial number of each USB Debug Adapter connected. If only one USB Debug Adapter is connected, an empty string can be used. The default is an empty string.
2. *nECprotocol*—Connection protocol used by the target device; JTAG (0) or Silicon Laboratories 2-Wire (C2) (1). The default is '0'. The C2 interface is used with C8051F3xx derivative devices and the JTAG interface is used with C8051F0xx, C8051F1xx, and C8051F2xx derivative devices.
3. *nPowerTarget*—If this parameter is set to '1', the USB Debug Adapter will be configured to continue supplying power after it has been disconnected from the target device. The default is '0', configuring the adapter to discontinue supplying power when disconnected.
4. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is '0'.

Return Value: See Table 1 on page 16.

3.4. DisconnectUSB()

Description: This function is used to disconnect from a target C8051Fxxx device using a USB Debug Adapter.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall DisconnectUSB();
```

Parameters: none

Return Value: See Table 1 on page 16.

3.5. Connected()

Description: This function returns a value representing the connection state of the target C8051Fxxx.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) BOOL__stdcall Connected();
```

Parameters: none

Return Value: = 0 (not connected) or
= 1 (connected)

4. Program Interface Functions

The following program interface functions are available for use in the Interface Utilities DLL.

`Download()`	- Download a hex file to the target C8051Fxxx device.
`ISupportBanking()`	- Checks to see if banking is supported.
`GetSAFirmwareVersion()`	- Returns the Serial Adapter firmware version.
`GetUSBFirmwareVersion()`	- Returns the USB Debug Adapter firmware version.
`GetDLLVersion()`	- Returns the Utilities DLL version.
`GetDeviceName()`	- Returns the name of the target C8051Fxxx device.

4.1. Download()

Description: This function is used to download a hex file to a target C8051Fxxx device. After a successful exit from the *Download()* function, the target C8051xxx will be in a "Halt" state. If the device is left in the "Halt" state, it will not begin code execution until the device is reset by a Power-On reset or by a *SetTargetGo()* DLL function call.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:

```
extern "C" __declspec(dllimport) int__stdcall Download(
char * sDownloadFile, int nDeviceErase=0, int nDisableDialogBoxes=0,
int nDownloadScratchPadSFLE=0, int nBankSelect=-1, int nLockFlash=0);
```

Parameters:

1. *sDownloadFile*—A character pointer to the beginning of a character array (string) containing the full path and filename of the file to be downloaded.
2. *nDevice Erase*—When set to '1' performs a device erase before the download initiates. If set to '0' the part will not be erased. A device erase will erase the entire contents of the device's Flash. The default is '0'.
3. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is '0'.
4. *nDownloadScratchPadSFLE*—This parameter is only for use with devices that have a Scratchpad Flash memory block. Currently, this includes the C8051F02x, C8051F04x, C8051F06x, and C8051F12x devices. For all other devices this parameter should be left in it's default state. Set this parameter to '1' in order to download to Scratchpad memory. When accessing and downloading to Scratchpad memory the only valid address range is 0x0000 to 0x007F. The default is '0'.
5. *nBankSelect*—This parameter is only for use with C8051F12x devices. For all other devices this parameter should be left in it's default state. When using a C8051F12x derivative set this parameter to '1', '2', or '3' in order to download to a specific bank. The default is '–1'.
6. *nLockFlash*—Set this parameter to '1' to lock the Flash following the download. If Flash is locked, the DLL will no longer be able to connect to the device.

Return Value: See Table 1 on page 16.

4.2. ISupportBanking()

Description: This function checks to see if banking is supported.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall ISupportBanking(
int * nSupportedBanks);
```

Parameters:

1. *nSupportedBanks*—The *ISupportBanking()* function expects to receive a valid pointer to an integer value. The *ISupportBanking()* function will set *nSupportedBanks* to the number of banks supported on the target device, or 0 if none exist.

Return Value: See Table 1 on page 16.

4.3. GetSAFirmwareVersion()

Description: This function is used to retrieve the version of the Serial Adapter firmware.

Supported Debug Adapters: Serial Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall GetSAFirmwareVersion();
```

Parameters: none

Return Value: Serial Adapter Firmware version

4.4. GetUSBFirmwareVersion()

Description: This function is used to retrieve the version of the USB Debug Adapter firmware.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall GetUSBFirmwareVersion();
```

Parameters: none

Return Value: USB Debug Adapter Firmware version

4.5. GetDLLVersion()

Description: This function returns the current version of the Utilities DLL.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) char* __stdcall GetDLLVersion();
```

Parameters: none

Return Value: A string containing the Utilities DLL version

4.6. GetDeviceName()

Description: This function returns the name of the target C8051Fxxx device that is currently supported.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall GetDeviceName(
const char **psDeviceName);
```

Parameters: 1. *psDeviceName* - A pointer to a character string location where the device name will be copied.

Return Value: See Table 1 on page 16.

5. Get Memory Functions

The following functions for reading device memory locations are available for use in the Interface Utilities DLL.

`GetRAMMemory()` - Read RAM memory from a specified address.
`GetXRAMMemory()` - Read XRAM memory from a specified address.
`GetCodeMemory()` - Read Code memory from a specified address.

The “GetMemory” functions all expect to receive a pointer to an initialized unsigned char (BYTE) array of *nLength* as the first parameter. If the “GetMemory” functions complete successfully, the *ptrMem* variable will contain the requested memory.

An example is given below showing how to properly initialize an array in C++:

```
unsigned char* ptrMem;

ptrMem = new unsigned char[length]; //Assumes that length has been declared
                                    //and set elsewhere

// next populate the array with the bytes to write in memory
```

Alternatively:

```
BYTE ptrMem[10] = {0x00};  // Must initialize the array prior to passing
                           // it into the DLL
```

5.1. GetRAMMemory()

Description: This function will read the requested memory from the Internal Data Address Space. The requested RAM memory must be located in the target device’s Internal Data Address Space.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall GetRAMMemory(
BYTE * ptrMem, DWORD wStartAddress, unsigned int nLength);
```

Parameters:

1. *ptrMem*—Pointer to the receive buffer, an initialized unsigned char (BYTE) array of length *nLength*. If the function completes successfully, the *ptrMem* variable will contain the requested memory.
2. *wStartAddress*—The start address of the memory location to be referenced.
3. *nLength*—The number of bytes to read from memory.

Return Value: See Table 1 on page 16.

5.2. GetXRAMMemory()

Description: This function will read the requested memory from the External Data Address Space. The requested XRAM memory must be located in the target device's External Data Address Space. Special attention should be paid to insure proper referencing of the External Data Address Space.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall GetXRAMMemory(
BYTE * ptrMem, DWORD wStartAddress, unsigned int nLength);
```

Parameters:

1. *ptrMem*—Pointer to the receive buffer, an initialized unsigned char (BYTE) array of length *nLength*. If the function completes successfully, the *ptrMem* variable will contain the requested memory.
2. *wStartAddress*—The start address of the memory location to be referenced.
3. *nLength*—The number of bytes to read from memory.

Return Value: See Table 1 on page 16.

5.3. GetCodeMemory()

Description: This function will read the requested memory from the Program Memory Space, including the Lock Byte(s). The requested Code memory must be located in the target device's Program Memory Space. Special attention should be paid when reading from a sector that has been read locked. Reading from a sector that has been read locked will always return 0s. Also, reading from the reserved space is not allowed and will return an error.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall GetCodeMemory(
BYTE * ptrMem, DWORD wStartAddress, unsigned int nLength);
```

Parameters:

1. *ptrMem*—Pointer to the receive buffer, an initialized unsigned char (BYTE) array of length *nLength*. If the function completes successfully, the *ptrMem* variable will contain the requested memory.
2. *wStartAddress*—The start address of the memory location to be referenced.
3. *nLength*—The number of bytes to read from memory.

Return Value: See Table 1 on page 16.

6. Set Memory Functions

The following functions for writing to device memory locations are available for use in the Interface Utilities DLL.

`SetRAMMemory()` - Writes value to a specified address in RAM memory.
`SetXRAMMemory()` - Writes value to a specified address in XRAM memory.
`SetCodeMemory()` - Writes value to a specified address in code memory.

The “SetMemory” functions expect to receive a pointer to an initialized unsigned char (BYTE) array of *nLength* as the first parameter. This array should contain *nLength* number of elements initialized prior to calling into the DLL’s “SetMemory” functions. If the “SetMemory” functions complete successfully, the *ptrMem* variable will have successfully programmed the requested memory.

An example is given below showing how to properly initialize an array in C++:

```
unsigned char* ptrMem;

ptrMem = new unsigned char[length]; //Assumes that length has been declared
                                    //and set elsewhere

// next populate your array with the bytes that you want to set in memory
```

Alternatively:

```
// Must initialize the array prior to calling the DLL
BYTE ptrMem[10] = {0x00, 0x14, 0xAE, 0x50, 0xAD, 0x66, 0x01, 0x05, 0x77, 0xFF};
```

6.1. SetRAMMemory()

Description: This function will write to memory in the Internal Data Address Space. The target RAM memory must be located in the target device’s Internal Data Address Space.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall SetRAMMemory(
BYTE * ptrMem, DWORD wStartAddress, unsigned int nLength);
```

Parameters:

1. *ptrMem*—Pointer to the transmit buffer, an initialized unsigned char (BYTE) array of length *nLength*. If the function completes successfully, the *ptrMem* variable will have successfully programmed the requested memory.
2. *wStartAddress*—The start address of the memory location to be referenced.
3. *nLength* —The length of memory to be read.

Return Value: See Table 1 on page 16.

6.2. SetXRAMMemory()

Description: This function will write to memory in the External Data Address Space. The target XRAM memory must be located in the target device's External Data Address Space. Special attention should be paid to insure proper referencing of the External Data Address Space.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall SetXRAMMemory(
BYTE * ptrMem, DWORD wStartAddress, unsigned int nLength);
```

Parameters:

1. *ptrMem*—Pointer to the transmit buffer, an initialized unsigned char (BYTE) array of length *nLength*. If the function completes successfully, the *ptrMem* variable will have successfully programmed the requested memory.
2. *wStartAddress*—The start address of the memory location to be referenced.
3. *nLength*—The length of memory to be read.

Return Value: See Table 1 on page 16.

6.3. SetCodeMemory()

Description: This function will write to memory in the Program Memory Space. Writing to Flash will not complete successfully if a client tries to write to a read/write/erase lock byte, the reserved area of Flash, more than one page of data at one time, or a write/erase locked sector. Also, if writing to a page that contains the read/write/erase lock bytes, a device erase will be performed (a device erase will erase the entire contents of Flash except the reserved area). Please refer to the relevant device data sheets for additional information. If *SetCodeMemory()* completes successfully, only the specified range, *wStartAddress* + *nLength*, will have successfully been written. All other values within the page will retain their initial values.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall SetCodeMemory(
BYTE * ptrMem, DWORD wStartAddress, unsigned int nLength,
int nDisableDialogs=0);
```

Parameters:

1. *ptrMem*—Pointer to the transmit buffer, an initialized unsigned char (BYTE) array of length *nLength*. If the function completes successfully, the *ptrMem* variable will have successfully programmed the requested memory.
2. *wStartAddress*—The start address of the memory location to be referenced.
3. *nLength*—The length of memory to be read.
4. *nDisableDialogs*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is '0'.

Return Value: See Table 1 on page 16.

7. Target Control Functions

The following functions for controlling a device are available for use in the Interface Utilities DLL.

`SetTargetGo()` - Puts the target C8051Fxxx device in a “Run” state.
`SetTargetHalt()` - Puts the target C8051Fxxx device in a “Halt” state.

7.1. SetTargetGo()

Description: After a successful exit from the *SetTargetGo()* function, the target C8051xxx will be in a “Run” state.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype: `extern "C" __declspec(dllimport) int__stdcall SetTargetGo();`

Parameters: none

Return Value: See Table 1 on page 16.

7.2. SetTargetHalt()

Description: After a successful exit from the *SetTargetHalt()* function, the target C8051xxx will be in a “Halt” state.

Supported Debug Adapters: Serial Adapter, USB Debug Adapter

C++ Prototype: `extern "C" __declspec(dllimport) BOOL__stdcall SetTargetHalt();`

Parameters: none

Return Value: = 0 (target would not “Halt”) or
= 1 (target is now in the “Halt” state)

8. USB Debug Adapter Communication Functions

The following functions are available in the Interface Utilities DLL to query and configure a USB Debug Adapter. These functions are useful when more than one USB Debug Adapter is connected to the PC. Call *USBDebugDevices()* first to determine the number of USB Debug Adapters connected. Next, use *GetUSBDeviceSN()* to determine to Serial Number of each adapter. The Serial Number can then be used with other functions to communicate with an individual USB Debug Adapter.

`USBDebugDevices()`	- Determines how many USB Debug Adapters are present.
`GetUSBDeviceSN()`	- Obtains the serial number of the enumerated USB Debug Adapters.
`GetUSBDLLVersion()`	- Returns the version of the USB Debug Adapter driver file.

8.1. USBDebugDevices()

Description: This function will query the USB bus and identify how many USB Debug Adapter devices are present.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall USBDebugDevices(
DWORD * dwDevices);
```

Parameters:
1. *dwDevices*—A pointer to the value that will contain the number of devices that are found.

Return Value: See Table 1 on page 16.

8.2. GetUSBDeviceSN()

Description: This function obtains a Serial Number string for the USB Debug Adapter specified by an index passed in the dw*DeviceNum* parameter. The index of the first device is 0 and the index of the last device is the value returned by *USBDebugDevices()* – 1.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall GetUSBDeviceSN(
DWORD dwDeviceNum, const char ** psSerialNum);
```

Parameters:
1. *dwDeviceNum*—Index of the device for which the serial number is desired. To obtain the serial number of the first device use 0.
2. *psSerialNum*—A pointer to a character string location where the serial number will be copied.

Return Value: See Table 1 on page 16.

8.3. GetUSBDLLVersion()

Description: This function will return the version of the driver for the USB Debug Adapter.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall GetUSBDLLVersion(
const char ** pVersionString);
```

Parameters:
1. *pVersionString*—Pointer to a character string location where the version string of the USB Debug Adapter driver will be copied.

Return Value: See Table 1 on page 16.

9. Stand-Alone Functions

The following stand-alone functions are available for use in the Interface Utilities DLL. These functions do not require the use of the “Connect” or “Disconnect” functions and do not interact with any other functions.

`FLASHErase()`	- Erase the Flash program memory using a Serial Adapter.
`FLASHEraseUSB()`	- Erase the Flash program memory using a USB Debug Adapter.

9.1. FLASHErase()

Description: This function is used to erase the Flash program memory of a device using a Serial Adapter. This function must be used to “unlock” a device whose Flash read and/or write lock bytes have been written.

Supported Debug Adapters: Serial Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall FLASHErase(
int nComPort=1, int nDisableDialogBoxes=0, int nECprotocol=0);
```

Parameters:

1. *nComPort*—Target COM port to establish a connection. The default is ‘1’.
2. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is ‘0’.
3. *nECprotocol*—Connection protocol used by the target device; JTAG (0) or Silicon Laboratories 2-Wire (C2) (1). The default is ‘0’. The C2 interface is used with C8051F3xx derivative devices and the JTAG interface is used with C8051F0xx, C8051F1xx, and C8051F2xx derivative devices.

Return Value: See Table 1 on page 16.

9.2. FLASHEraseUSB()

Description: This function is used to erase the Flash program memory of a device using a USB Debug Adapter. This function must be used to “unlock” a device whose Flash read and/or write lock bytes have been written.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int __stdcall FLASHEraseUSB(
const char * sSerialNum, int nDisableDialogBoxes=0, int nECprotocol=0);
```

Parameters:

1. *sSerialNumber*—The serial number of the USB Debug Adapter. See Section 8 for information on obtaining the serial number of each USB Debug Adapter connected. If only one USB Debug Adapter is connected, an empty string can be used. The default is an empty string.
2. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is ‘0’.
3. *nECprotocol*—Connection protocol used by the target device; JTAG (0) or Silicon Laboratories 2-Wire (C2) (1). The default is ‘0’. The C2 interface is used with C8051F3xx derivative devices and the JTAG interface is used with C8051F0xx, C8051F1xx, and C8051F2xx derivative devices.

Return Value: See Table 1 on page 16.

10. Multi-Device JTAG Programming

Multiple Silicon Laboratories C8051Fxxx devices that use the JTAG communications protocol can be connected and programmed in a JTAG chain. Currently this includes the C8051F0xx, C8051F1xx, and C8051F2xx device families. Configure the JTAG chain as shown in Figure 1. It will be necessary to know the instruction register length of each device in the JTAG chain. The instruction register length of all Silicon Laboratories JTAG devices is 16 bits. The following functions to configure and connect to a C8051Fxxx device on a JTAG chain is available for use in the Interface Utilities DLL.

`SetJTAGDeviceAndConnect()` - Configure a connection to a C8051Fxxx device on a JTAG chain using a Serial Adapter.

`SetJTAGDeviceAndConnectUSB()` - Configure a connection to a C8051Fxxx device on a JTAG chain using a USB Debug Adapter.

Once the “SetJTAGDeviceAndConnect” function has returned successfully and a connection has been established with a Silicon Laboratories device in the JTAG chain, any of the previously mentioned Communication, Program Interface, and Memory functions may be used on the isolated device. When finished interfacing with the device, call a “Disconnect” function as usual to disconnect from the device.

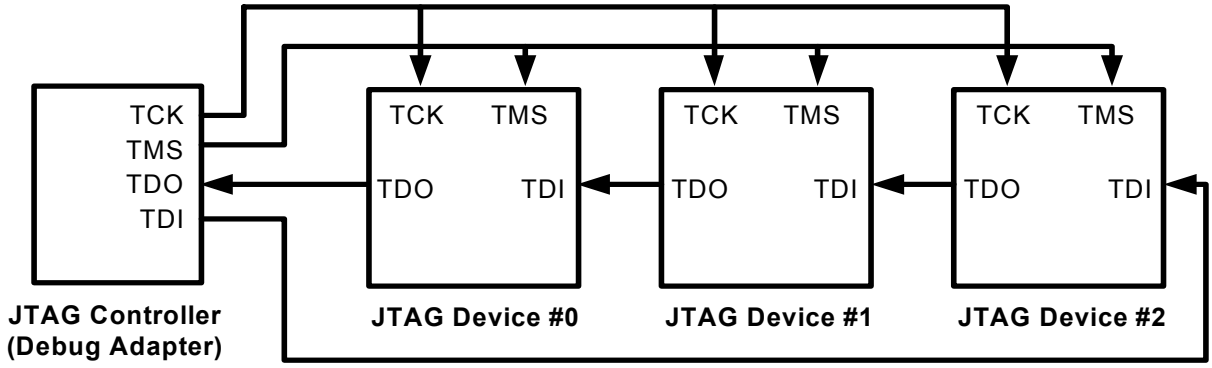

Figure 1. JTAG Chain Connection

10.1. SetJTAGDeviceAndConnect()

Description: This function is used to connect to a single target JTAG device in a JTAG chain using a Serial Adapter.

Supported Debug Adapters: Serial Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall SetJTAGDeviceAndConnect(
int nComPort=1, int nDisableDialogBoxes=0,
BYTE DevicesBeforeTarget=0, BYTE DevicesAfterTarget=0,
WORD IRBitsBeforeTarget=0, WORD IRBitsAfterTarget=0);
```

Parameters:

1. *nComPort*—Target COM port to establish a connection. The default is '1'.
2. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is '0'.
3. *DevicesBeforeTarget*—Number of devices in the JTAG chain before the target device. The default is '0'.
4. *DevicesAfterTarget*—Number of devices in the JTAG chain after the target device. The default is '0'.
5. *IRBitsBeforeTarget*—The sum of instruction register bits in the JTAG chain before the target device. The default is '0'.
6. *IRBitsAfterTarget*—The sum of instruction register bits in the JTAG chain after the target device. The default is '0'.

Following Figure 1, assuming all devices in the JTAG chain are Silicon Laboratories devices, call *SetJTAGDeviceAndConnect()* as shown in the following examples:

To access JTAG Device #0:

```
SetJTAGDeviceAndConnect(1, 0, 0, 2, 0, 32);
```

nComPort=1
nDisableDialogBoxes=0
DevicesBeforeTarget=0
DevicesAfterTarget=2
IRBitsBeforeTarget=0
IRBitsAfterTarget=32

To access JTAG Device #1:

```
SetJTAGDeviceAndConnect(1, 0, 1, 1, 16, 16);
```

nComPort=1
nDisableDialogBoxes=0
DevicesBeforeTarget=1
DevicesAfterTarget=1
IRBitsBeforeTarget=16
IRBitsAfterTarget=16

To access JTAG Device #2:

```
SetJTAGDeviceAndConnect(1, 0, 2, 0, 32, 0);
```

nComPort=1
nDisableDialogBoxes=0
DevicesBeforeTarget=2
DevicesAfterTarget=0
IRBitsBeforeTarget=32
IRBitsAfterTarget=0

Return Value: See Table 1 on page 16.

10.2. SetJTAGDeviceAndConnectUSB()

Description: This function is used to connect to a single target JTAG device in a JTAG chain using a USB Debug Adapter.

Supported Debug Adapters: USB Debug Adapter

C++ Prototype:
```
extern "C" __declspec(dllimport) int__stdcall SetJTAGDeviceAndConnectUSB(
const char * sSerialNum, int nPowerTarget=0, int nDisableDialogBoxes=0,
BYTE DevicesBeforeTarget=0, BYTE DevicesAfterTarget=0,
WORD IRBitsBeforeTarget=0, WORD IRBitsAfterTarget=0);
```

Parameters:

1. *sSerialNumber*—The serial number of the USB Debug Adapter. See Section 8 for information on obtaining the serial number of each USB Debug Adapter connected. If only one USB Debug Adapter is connected, an empty string can be used. The default is an empty string.
2. *nPowerTarget*—If this parameter is set to '1', the USB Debug Adapter will be configured to continue supplying power after it has been disconnected from the target device. The default is '0', configuring the adapter to discontinue supplying power when disconnected.
3. *nDisableDialogBoxes*—Disable (1) or enable (0) dialogs boxes within the DLL. The default is '0'.
4. DevicesBeforeTarget—Number of devices in the JTAG chain before the target device. The default is '0'.
5. *DevicesAfterTarget*—Number of devices in the JTAG chain after the target device. The default is '0'.
6. IRBitsBeforeTarget—The sum of instruction register bits in the JTAG chain before the target device. The default is '0'.
7. *IRBitsAfterTarget*—The sum of instruction register bits in the JTAG chain after the target device. The default is '0'.

Following Figure 1, assuming all devices in the JTAG chain are Silicon Laboratories devices, call *SetJTAGDeviceAndConnectUSB()* as shown in the following examples:

To access JTAG Device #0:

```
SetJTAGDeviceAndConnectUSB("", 0, 0, 0, 2, 0, 32);
```

sSerialNumber=""	DevicesBeforeTarget=0
nPowerTarget=0	DevicesAfterTarget=2
nDisableDialogBoxes=0	IRBitsBeforeTarget=0
	IRBitsAfterTarget=32

To access JTAG Device #1:

```
SetJTAGDeviceAndConnectUSB("", 0, 0, 1, 1, 16, 16);
```

sSerialNumber=""	DevicesBeforeTarget=1
nPowerTarget=0	DevicesAfterTarget=1
nDisableDialogBoxes=0	IRBitsBeforeTarget=16
	IRBitsAfterTarget=16

To access JTAG Device #2:

```
SetJTAGDeviceAndConnectUSB("", 0, 0, 2, 0, 32, 0);
```

sSerialNumber=""	DevicesBeforeTarget=2
nPowerTarget=0	DevicesAfterTarget=0
nDisableDialogBoxes=0	IRBitsBeforeTarget=32
	IRBitsAfterTarget=32

Return Value: See Table 1 on page 16.

11. Linking

Unless using explicit linking, it is necessary to provide the linker with the path of the “SiUtil.lib” library file before building the client executable. In Microsoft Visual C++ this is accomplished by selecting *Settings*… from the Project menu and then the *Link* tab. In the Object/library modules box enter the full path and filename of the library file. For example; “c:\project\release\SiUtil.lib”. The library file is not needed after the client executable is built.

If the DLL is implicitly linked, the DLL must be placed in one the following directories:

1. The directory containing the EXE client file.
2. The process’s current directory.
3. The Windows System directory.
4. The Windows directory.
5. A directory listed in the PATH environment variable.

12. Test Results

On exit, the DLL will return an integer value return code. If a fatal error occurs during the DLL’s execution, the DLL will also display a message box, if display dialogs are enabled, stating the error and then exit. The return codes are listed in Table 1.

Table 1. Return Codes

Return Code	Error	Status	Possible Causes
–3	Flash Write Error	Failed	Invalid page write, writing to the reserved area of Flash, etc.
–2	Target State Failure	Failed	Target not in Halt State
–1	Target State Failure	Failed	Target not Connected
0	No error	Success	Function call completed Successfully
1	File Name or path Error	Failed	Invalid path and/or file doesn’t exist
2	COM Port Error	Failed	Cannot establish a connection with the selected COM port
3	Download Sequence Error	Failed	Invalid number of bytes, requested memory operation does not exist
4	Reset Sequence Error	Failed	The target device failed to execute a reset sequence; verify that a valid connection still exist
5	Device Erase Error	Failed	The target device failed to execute an Erase sequence; verify Write/Erase Lock Bytes; verify that a valid connection still exists
7	Error Closing COM Port	Failed	Could not establish a connection with the target to Close the COM port; verify that a connection exists
8	Invalid Parameter	Failed	Invalid parameter[s] passed into the DLL
9	USB Debug Adapter DLL Missing	Failed	The USB Debug Adapter DLL, USBHID.dll, was not found
10	USB Debug Adapter Error	Failed	Cannot establish a connection with the selected USB Debug Adapter

13. Known Limitations

Upon invocation of the DLL in the debug mode of a client process, dialog messaging may not function properly. Dialog messaging provides a client with a way to get instant information that may not otherwise be available. Dialog messaging supports a progress indicator that provides a client with information on the progress of memory operations that may take time to complete. Calling into the DLL with a client (in debug version) may cause the DLL to misinterpret the correct window handle used to display the dialog boxes. The recommended course of action is to set all functions that have a *nDisableDialogBoxes* parameter to '1' before calling DLL functions. All *nDisableDialogBoxes* parameters default to '0'. This problem will not occur in the release mode of a client process.

14. Visual Basic Information

When writing a Visual Basic client it is important to note that the Interface Utilities DLL is written using Visual C++. Thus, you must take into account variable type differences between the two languages. Specifically, the VC++ boolean type and the VB boolean type are incompatible. In VC++ TRUE = 1 and FALSE = 0, whereas in VB TRUE = –1 and FALSE = 0. To resolve this issue you must use an integer instead of a boolean when writing your VB client and send an integer value 1 for TRUE and integer value 0 for FALSE.

Document Change List

Revision 2.2 to Revision 2.3

- Instructions for declaring functions moved to Section 1.
- Reference to the SetJTAGDeviceAndConnect() function moved from Section 9 to Section 10.
- Replaced incorrect function descriptions in the Section 9 function list.

Revision 2.3 to Revision 2.4

- "Supported Debug Adapter" field added to each function definition.
- The following new functions added to support the USB Debug Adapter: ConnectUSB(), DisconnectUSB(), USBDebugDevices(), GetUSBDeviceSN(), GetHIDDLLVersion(), FLASHEraseUSB(), GetUSBFirmwareVersion(), SetJTAGDeviceAndConnectUSB()
- Section 4, "Program Interface Functions" split into 2 sections, creating Section 7, "Target Control Functions".
- Section 8, "USB Debug Adapter Communication Functions" added.
- Added Return Codes 9 and 10 to Table 1, "Return Codes".

Revision 2.4 to Revision 2.5

- Pointer format in prototypes corrected for consistency.
- `Std::string` types of variables changed to `const char * variables`.
- Section 8.3, "GetUSBDLLVersion()" function name changed from GetHIDDLLVersion to GetUSBDLLVersion.
- Section 10.2, "SetJTAGDeviceAndConnectUSB()" prototype corrected.

Revision 2.5 to Revision 2.6

- New parameter *nLockFlash* added to Download() function, Section 4.1.
- GetDLLVersion() function returns a "char*" value instead of an "int" value, Section 4.5.

Revision 2.6 to Revision 2.7

- Added description for new *GetDeviceName()* function as Section 4.6 on page 5.
- Moved function descriptions for *GetSAFirmwareVersion()*, *GetUSBFirmwareVersion()*, and *GetDLLVersion()* to Section 4, "Program Interface Functions".

Revision 2.7 to Revision 2.8

- Updated Section 4.6 on page 5.
- Updated Section 8.2 on page 11.
  - Replaced "dllexport" with "dllimport".

NOTES:

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