ComBlock

USB 2.0 INTERFACE USER MANUAL

1 Scope

Most new ComBlock modules support high-speed communications with a host computer over a standard USB 2.0 connection. These ComBlock modules can be used as

(a) Ready-to-use application-specific ComBlocks, or

(b) Development platforms with user-developed code.

This manual addresses both use cases.

Users of ready-to-use application-specific ComBlocks only need to read the "Windows Device Driver Installation" section. A troubleshooting section is also available, if needed, at the end of this document.

FPGA developers should read the following sections: "Windows Device Driver Installation", "Architecture" , "USB 2.0 Device" and "FPGA/VHDL development".

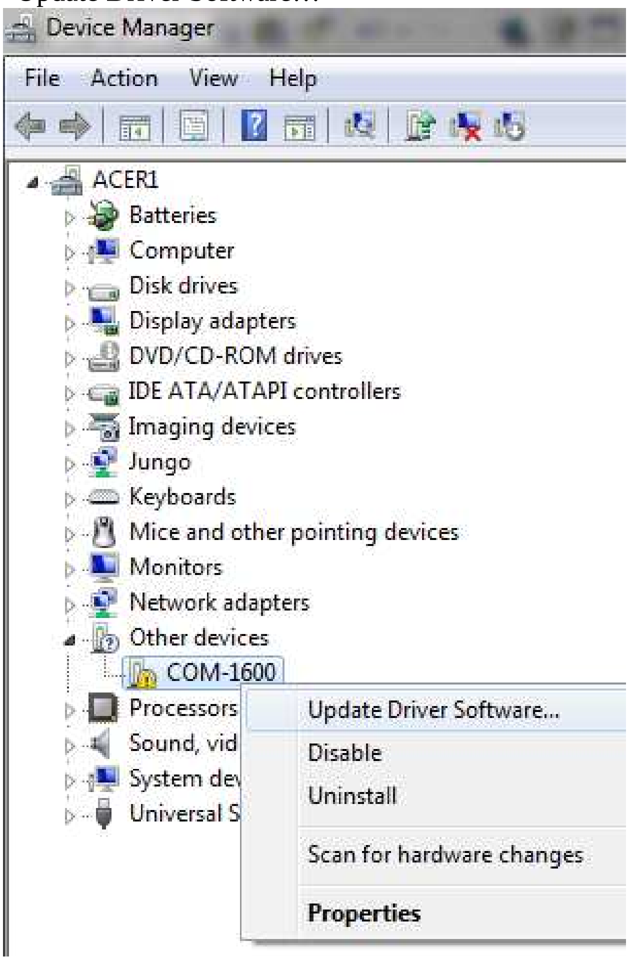
ARM developers should read the following sections: "Windows Device Driver Installation", "Architecture",

Java/C/C++ application developers on the host PC should read the following sections: : "Windows Device Driver Installation", "Architecture", "Applications".

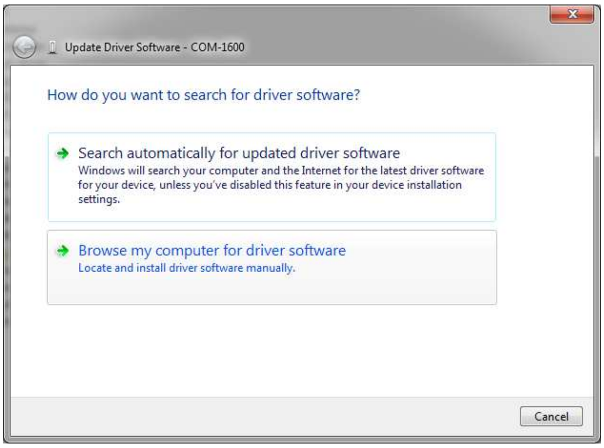
2 Windows Device Driver Installation (Windows 7)

When connecting a ComBlock for the first time, the user must install a special driver once. The step-by-step instructions are shown below for a typical Windows 7 driver installation.

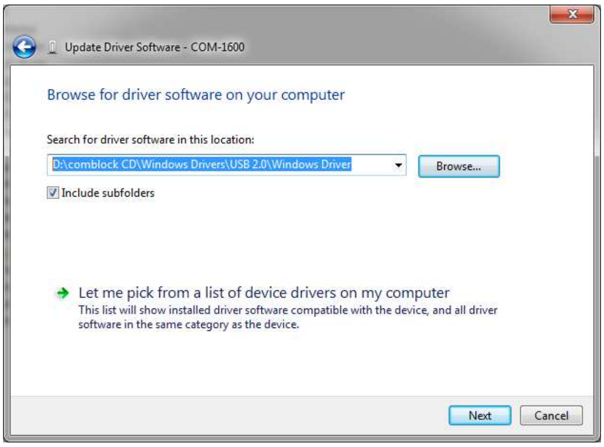
First, go to the Control Panel | Device Manager window. Plug in the USB cable to a powered ComBlock. The ComBlock appears under the "other devices" section. Right-click to select "Update Driver Software…"



MSS • 18221-A Flower Hill Way • Gaithersburg, Maryland 20879 • U.S.A.
Telephone: (240) 631-1111 Facsimile: (240) 631-1676 www.ComBlock.com

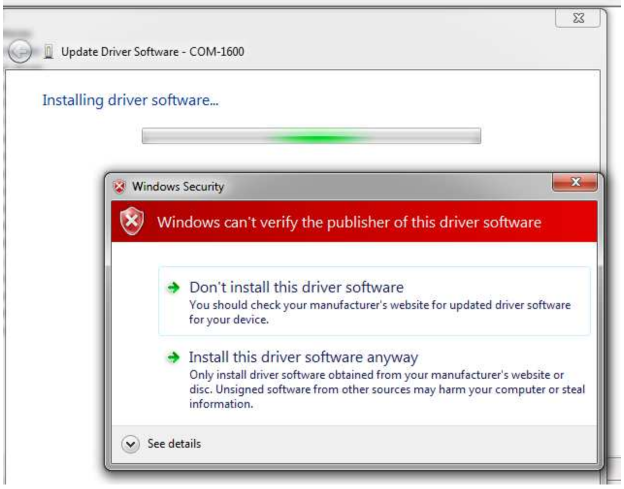


Select "Browse my computer". The next window will let the user specify the driver location for the new hardware.



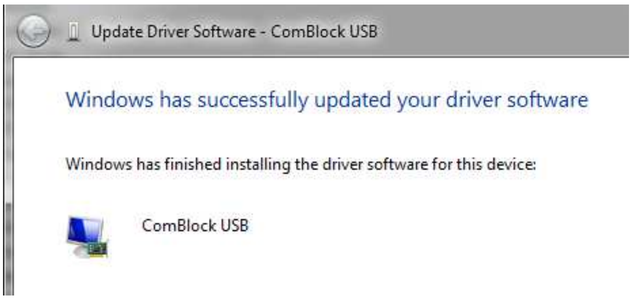
Select path to the ComBlock CD-ROM. Click on "Next"

A window may pop up to warn the user that the hardware and driver have not been tested officially for Microsoft Windows operating systems.



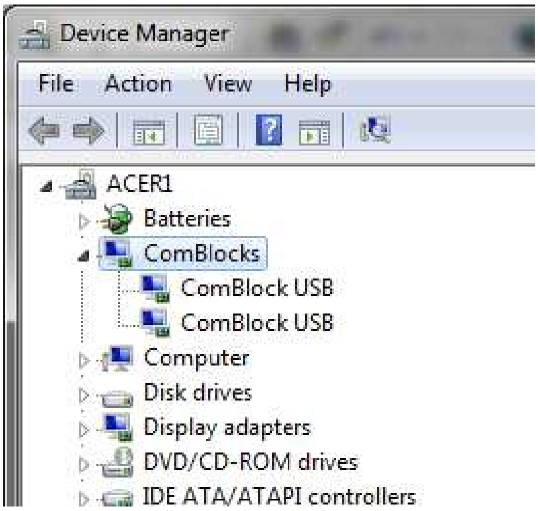
Click on "Install this driver anyway". Wait….

The last window for the New Hardware Wizard should appear, as shown below, for a successful installation.



At this point, the USB driver for ComBlock has been successfully installed and next time the ComBlock is plugged in, the system automatically finds appropriate driver. With the driver installed, the user can talk to the ComBlock, using the ready-to-use ComBlock control center or applications based on the WinUSB API (see the Applications section).

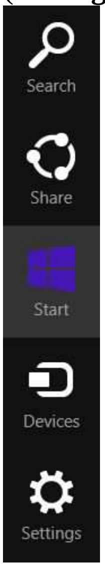
An easy way to verify the proper installation is to go to the Windows device manager (Control Panel | Device Manager). Once powered and properly connected over USB, all the ComBlocks will show up under the "ComBlocks" category as "ComBlock_USB" (as shown below for two concurrent USB connections to the same PC).



3 Windows Device Driver Installation (Windows 8 or 8.1)

The ComBlock USB driver is not signed by Microsoft. One must therefore go through a few extra steps for the operating system to allow the driver.

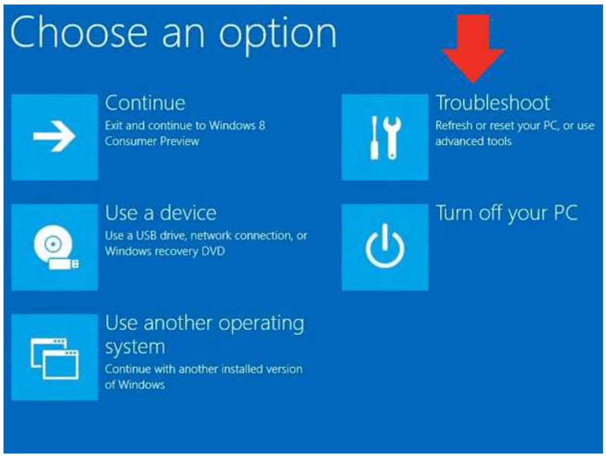
1. Hold down the Windows key on your keyboard and press the letter C to open the Charm menu, then click the gear icon (**Settings**).



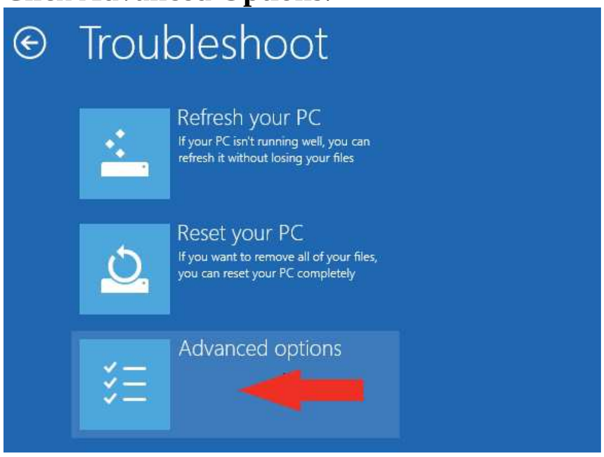
2. In Windows 8 select ‘More PC settings > General > Advanced startup > Restart now.

   In Windows 8.1 select ‘Change PC settings > Update & Recovery > Recovery

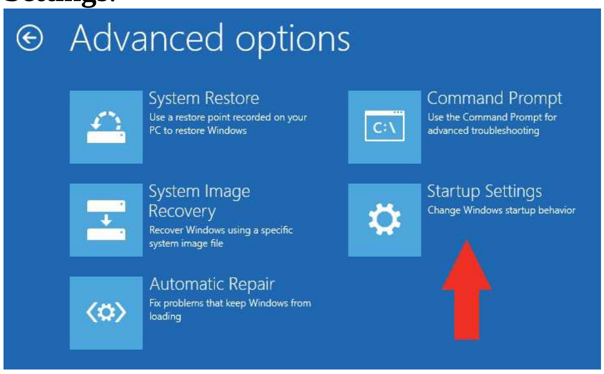
3. After restarting, click **Troubleshoot**.



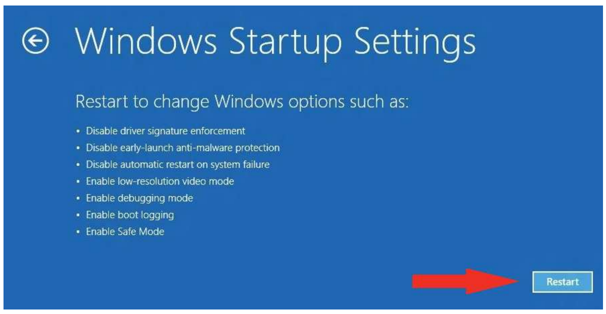
4. Click **Advanced Options**.



5. Click **Windows Startup Settings**.



6. Click **Restart**.



7. After restarting your computer a second time, choose **Disable driver signature enforcement** from the list by typing the number 7 on your keyboard.

Your computer will restart automatically.

8. After restarting, you will be able to install the ComBlock USB driver as per the instructions in section 2.

4 Linux

ComBlock does not supply any Linux driver. Nevertheless it is possible to establish communication over a standard USB cable between a PC running Linux and ComBlocks, albeit with a little more effort. The information needed to connect with a ComBlock USB device is as follows:

Vendor ID: 0x0000
Product ID: 0x0004

COM-16xx ARM:

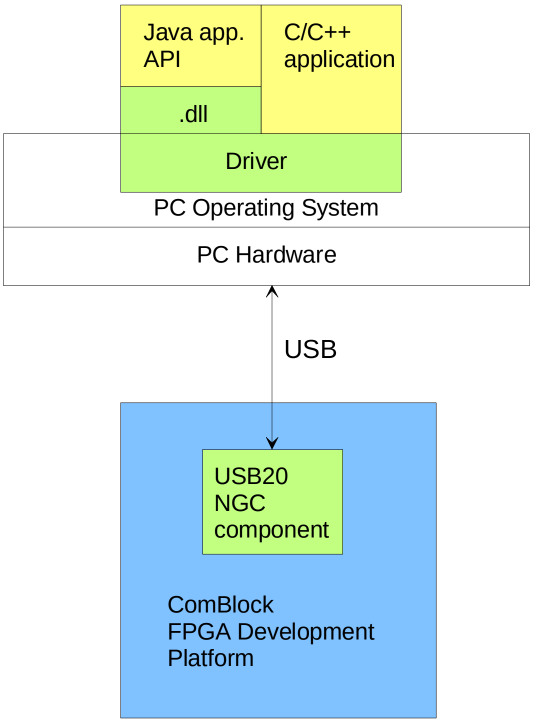
- Monitoring & Control streams
  - Bulk IN pipe: 0x82
  - Bulk OUT pipe: 0x02
- High-Speed data streams (when applicable):
  - Bulk IN pipe: 0x85
  - Bulk OUT pipe: 0x05

All others USB-equipped:

- Monitoring & Control streams
  - Bulk IN pipe: 0x81
  - Bulk OUT pipe: 0x02
- High-Speed data streams (when applicable):
  - Bulk IN pipe: 0x83
  - Bulk OUT pipe: 0x04

5 Architecture

The end-to-end communication architecture between a host computer and the ComBlock module as a USB device is illustrated below:

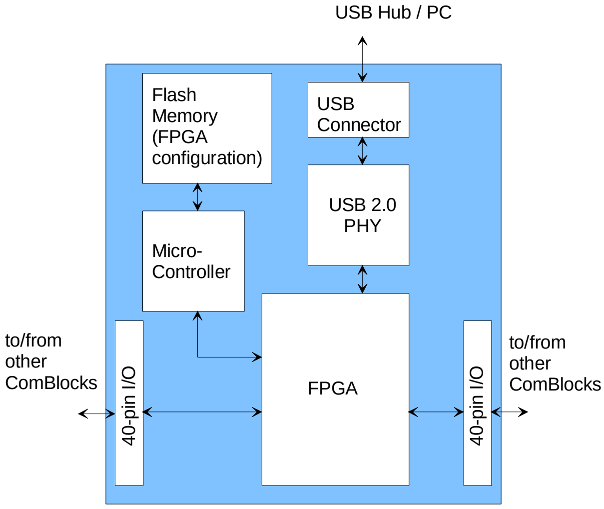


Blue: supplied hardware
Green: supplied ready-to-use software
Yellow: Source code examples

Host side (PC):
In order for a user to setup a USB 2.0 connection between the host computer and a ComBlock, the user must first create a Java or C/C++ application. The Java application calls simple methods described in the Java Application Programming Interface (API) described further in this document.

C/C++ applications can call drivers functions directly as described in the C/C++ Application described further in this document.

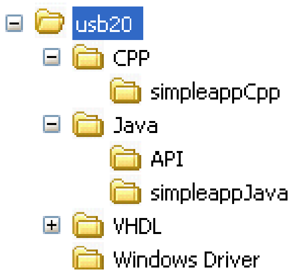
Device side (ComBlock):
On the device side, the USB connection is implemented partly within a PHY integrated circuit and partly within the FPGA as illustrated below:



Block Diagram of ComBlock as USB 2.0 Device

Supplied Components:
The **USB 2.0 software package** provides software to help users and developers create USB high-speed communication between the ComBlock platform and a host PC. The software components include the following:

- Windows device driver for XP/2000 (.sys and .inf files)
- Java API, DLL and simple application code example
- C/C++ simple application code example
- USB20 NGC component for integration within the VHDL code



The **USB 2.0 software package** is available in the ComBlock CD and can also be downloaded from www.comblock.com/download/usb20.zip

USB Capable ComBlock Platforms
The ComBlock Platforms currently capable of high-speed USB 2.0 connections are listed below:

- COM-1100
- COM-1200
- COM-1400
- COM-1600

VHDL top-level code examples (templates) for these ComBlock platforms are available from the ComBlock CD and ComBlock website (www.comblock.com/download).

6 Applications

6.1 Java API

The Java API is documented in the …\Java\API\USB.html document found in the USB 2.0 software package.

The user applications can transfer data using UsbRead and UsbWrite function calls.

The DLL (…\Java\simpleappJava\usbcpp.dll), which links the Java application to the drivers, is provided in the USB 2.0 software package.

Polling is the primary method for transferring data from the USB device to the user application (as opposed to interrupt which is not supported). Polling is achieved by attempting to read data from the USB device using the UsbRead function call. The user application can poll as frequently as it needs. If no data is present in the USB device, the UsbRead function will return 0. Otherwise, it will return the number of bytes actually read into the read buffer.

The user application supplies buffers for data transfer using the UsbRead and UsbWrite function calls. The minimum and maximum buffer sizes are:

- 1 to 4096 bytes for write
- 64 to 4096 bytes for full speed read
- 128 to 4096 bytes for high speed read

The UsbRead and UsbWrite functions return the number of bytes actually transferred, depending on flow control and the availability of data. For example, UsbRead may return 1 if only one byte was read from the USB device.

6.2 C/C++ Application

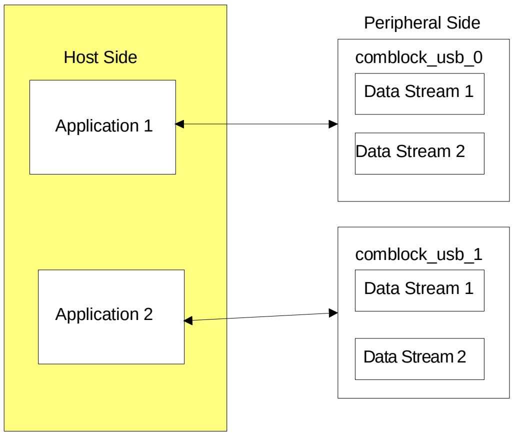
The C++ application can transfer data using the DeviceIoControl function call.

Application example can be found at ...\CPP\simpleappCpp\simpleappCpp.dsw

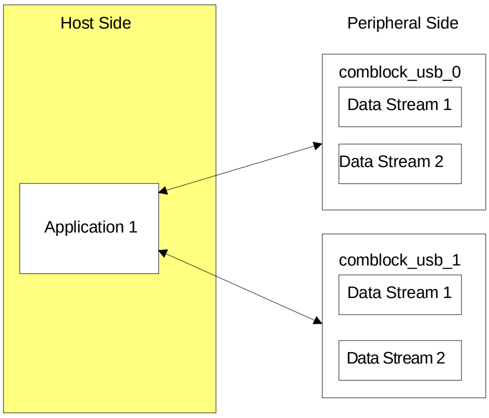
The buffer size limitations are the same as for Java.

6.3 Addressing Multiple ComBlocks

Multiple ComBlocks can be attached to a Host PC. Each ComBlock can be identified by a unique device name assigned when it is attached. The device name would be “comblock_usb_X” where X is a number starting with 0 and it depends on the order in which the ComBlock has been attached. The user applications can communicate with any of the ComBlocks exclusively by addressing them with the device name.



Sample communication model 1: Two user applications communicating with two ComBlocks over two USB cables.



Sample communication model 2: One user application communicating with two ComBlocks over two USB cables.

7 USB 2.0 Device

The USB20.ngc (Xilinx) FPGA component is supplied in the ComBlock CD-ROM with the

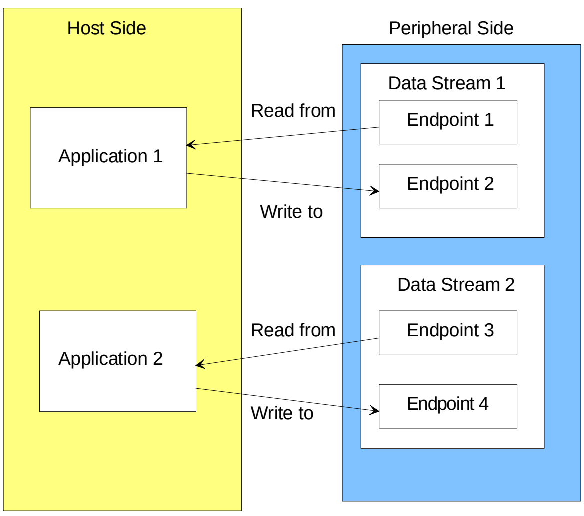
ComBlock FPGA development platforms. This component implements the USB communication protocol (Serial Interface Engine SIE) for a USB device within an FPGA.

This code implements the following:

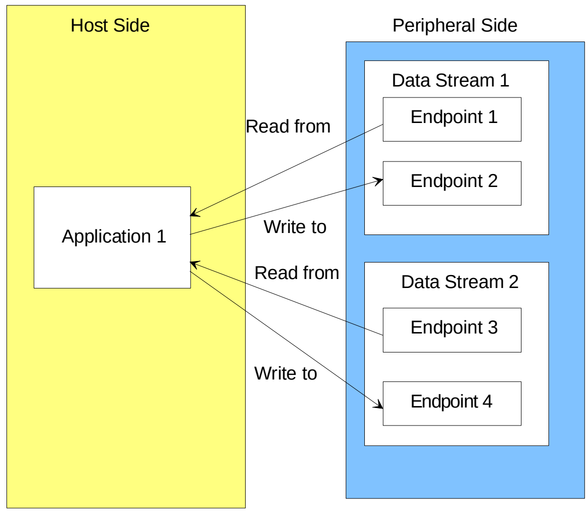
- High Speed (480 Mbits/s) and Full Speed (12 Mbits/s) data transfer. Speed selection is done automatically by auto-negotiation between the host PC and this device.
- Two independent data streams for communication between the host and the ComBlock
  - Data Stream 1 consists of endpoints 1 and 2
  - Data Stream 2 consists of endpoints 3 and 4
- Endpoints 1 and 3: can be used to read from the ComBlock
- Endpoints 2 and 4: can be used to write to the ComBlock

"Endpoint is a simplex connection that supports data flow in one direction".

The data streams are to be used in conjunction with Java or C/C++ applications. The user applications can communicate with either of the two data streams or both.



Sample communication model 3: Two user applications communicating with two independent data streams on a single ComBlock over a single USB cable.
Note: Application 1 must release the handle before Application 2 can take it and vice versa.



Sample communication model 4: One user application communicating with two independent data streams on a single ComBlock over a single USB cable.

.ngc components are supplied for the following ComBlock FPGA development platforms: COM-1200, COM-1400, COM-1600.

8 FPGA/VHDL Development

This section describes how to create a custom application that makes use of the high-speed USB 2.0 connection on ComBlock FPGA-based development platforms. Users of ready-to-use application-specific ComBlock modules can skip this section.

This section focuses exclusively on the device side of the USB connection.

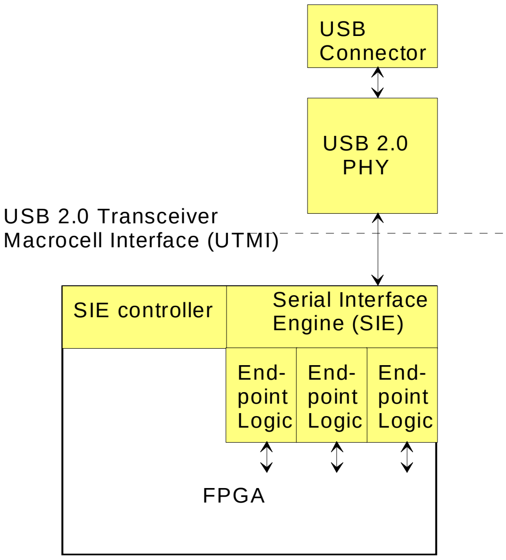
8.1 Device Architecture

8.1.1 Overview

The USB device is compliant with the USB 2.0 specification that allows for high data transfer throughputs. The hardware supports both the Full Speed (FS) mode for USB operation at 12 Mbits/s and the High Speed (HS) mode for USB operation at 480 Mbits/s. The Low Speed mode is not supported.

8.1.2 USB device implementation

The USB device implementation is divided into two sections: a very high-speed physical layer, mostly analog processing (USB transceiver macrocell), and a lower speed digital section comprised of the serial interface engine (SIE), the SIE controller, and the end-point logic. The physical layer is implemented by a specific PHY integrated circuit (SMSC GT3200) whereas the digital processing is implemented within the FPGA.



The interface between the FPGA and the USB 2.0 PHY transceiver is a standard as described by the "USB 2.0 Transceiver Macrocell Interface (UTMI) Specifications", Version 1.05 3/29/2001 found at www.usb.org

The ComBlock is a self-powered device and does not draw power from the USB device.

The USB PHY (SMSC GT3200) interfaces exclusively with the USB20 component. No other interface signaling is needed.

Data is exchanged between the USB20 component and the application through a 16Kbit dual-port (elastic) buffer in each direction. Hence the application-processing clock (CLK_P) can be selected independently of the USB20 60 MHz clock USB_CLK60G. [Note: application clock must be faster].

8.2 USB20 NGC Component

A NGC component encapsulating the USB serial interface engine (SIE) is provided as part of the ComBlock VHDL code template. The SIE works in conjunction with the Windows drivers to establish a virtual channel between the ComBlock and a host computer.

8.2.1 User Interface

The component is described primarily by its interface definition:

```
entity USB20 is
  port (

--// Clocks Resets
ASYNC_RESET: in std_logic;
USB_CLK60G: in std_logic;
 -- reference clock. 60 MHz.
 -- Supplied by the SMSC GT3200 IC (CLK0UT)
 -- Generally not used outside of this component.
 -- Global clock (BUFG MUST be instantiated outside).
CLK_P: in std_logic;
 -- Main processing or I/O clock used outside of this component.
 -- All application interface signals are synchronous with CLK_P
 -- Key assumption: CLK_P is slightly faster than USB_CLK60G/2.
 -- Other key assumption: CLK_P < 4 * CLK60G

--// USB PHY interface (SMSC GT3200 IC)
-- Direct connection between the USB20 component and the USB
-- PHY. Synchronous with USB_CLK60G clock
USB_RESET: out std_logic;
USB_DATABUS16_8: out std_logic;
USB_SUSPENDN: out std_logic;
USB_XCVRSELECT: out std_logic;
USB_TERMSELECT: out std_logic;
USB_OPMODE: out std_logic_vector(1 downto 0);
 -- operational mode
 -- 00 = normal operation
 -- 01 = non-driving (all terminations removed)
 -- 10 = disable bit stuffing and NRZI encoding (unused)
 -- 11 = reserved (unused)
USB_LINESTATE: in std_logic_vector(1 downto 0);
USB_TXVALID: out std_logic;
USB_TXREADY: in std_logic;
USB_VALIDH: inout std_logic;
 -- VALIDH is not used in 8-bit mode
USB_RXVALID: in std_logic;
USB_RXACTIVE: in std_logic;
USB_RXERROR: in std_logic;
USB_DATA_IN: in std_logic_vector(7 downto 0);
USB_DATA_OUT: out std_logic_vector(7 downto 0);
 -- time critical. User should add OFFSET OUT constraints in the
   constraint editor.
USB_VBUS_SENSE: in std_logic;

--// Data Stream 1
-- Synchronous with CLK_P clock
DATA1_OUT: out std_logic_vector(7 downto 0);
DATA1_OUT_SAMPLE_CLK: out std_logic;
 -- read DATA1_OUT at rising edge of CLK_P when
 -- DATA1_OUT_SAMPLE_CLK = '1'
DATA1_OUT_BUFFER_EMPTY: out std_logic;
DATA1_OUT_SAMPLE_CLK_REQ: in std_logic;
 -- requests data. If no data is available in the buffer, the
 -- DATA1_OUT_SAMPLE_CLK will stay low.
 -- (flow control)

DATA1_IN: in std_logic_vector(7 downto 0);
DATA1_IN_SAMPLE_CLK: in std_logic;
 -- read DATA1_IN at rising edge of CLK_P when
 -- DATA1_IN_SAMPLE_CLK = '1'
DATA1_IN_SAMPLE_CLK_REQ: out std_logic;
 -- requests data when the input elastic buffer is less than half full.
 -- (flow control)

--// Data Stream 2
-- Synchronous with CLK_P clock
DATA2_OUT: out std_logic_vector(7 downto 0);
DATA2_OUT_SAMPLE_CLK: out std_logic;
```

```
  -- read DATA2_OUT at rising edge of CLK_P when
  -- DATA2_OUT_SAMPLE_CLK = '1'
DATA2_OUT_BUFFER_EMPTY: out std_logic;
DATA2_OUT_SAMPLE_CLK_REQ: in std_logic;
  -- requests data. If no data is available in the buffer, the
  -- DATA2_OUT_SAMPLE_CLK will stay low.
  -- (flow control)
DATA2_IN: in std_logic_vector(7 downto 0);
DATA2_IN_SAMPLE_CLK: in std_logic;
  -- read DATA2_IN at rising edge of CLK_P when
  -- DATA2_IN_SAMPLE_CLK = '1'
DATA2_IN_SAMPLE_CLK_REQ: out std_logic;
  -- requests data when the input elastic buffer is less than half full.
  -- (flow control)

--// Test Points
  USB_TP: out std_logic_vector(10 downto 1)
  -- bit 1: speed after auto-negotiation with host PC: '1' if high-speed.
  -- bit 2: speed after auto-negotiation with host PC: '1' if full-speed.
  -- bit 3: valid SETUP message (PID valid, CRC5 valid).
  -- SETUP is the first message from the host PC to this USB device
  -- bit 4: Host asks to read the descriptor table.
  -- bit 5: data stream 2 input, elastic buffer write pointer LSb
  -- (address bit 0)
  -- bit 6: data stream 2 input, elastic buffer read pointer LSb
  -- (address bit 0)
  -- bit 7: data stream 2 input, elastic buffer write pointer MSb
  -- (address bit 10)
  -- bit 8: data stream 2 input, elastic buffer read pointer MSb
  -- (address bit 10) Useful in checking flow control
  -- bit 9: data stream 2 output, elastic buffer write pointer MSb
  -- (address bit 10)
  -- bit 10: data stream 2 output, elastic buffer read pointer MSb
  --(address bit 10) Useful in checking flow control

-- Other useful test points available at the component interface:
-- VBUS_SENSE. Goes high upon cable being plugged in at both ends
-- USB_RXERROR: USB PHY detects receive errors
-- USB_CLK60G: 60 MHz reference clock from the USB PHY through
-- global buffer. Useful in checking input and output signal timing.
-- USB_RXVALID from PHY (useful in checking the input timing
w.r.t. USB_CLK60G.
);
end entity;
```

8.2.2 USB Device Descriptors

Several data structures (descriptors) are stored in non-volatile memory within the ComBlock. They are read by the host computer operating system upon attaching the ComBlock to the host USB port.

The NGC USB20 component includes the standard descriptors listed below. The user cannot modify them. The descriptors below may be of use for software developers who want to develop a driver for the host computer. Readers intending to use the supplied Windows driver can skip this section.

Device Descriptor		
Offset	***Data (hex)***	***Description and interpretation***
0	12	Size of this descriptor in bytes
1	01	DEVICE descriptor type
2	00	USB specification release 2.00 (High-speed capable device)
3	02	
4	FF	Vendor-specific class (not registered with USB-IF)
5	FF	Vendor-specific subclass class (not registered with USB-IF)
6	FF	Vendor specific protocol class (not registered with USB-IF)
7	40	Maximum packet size for endpoint zero (64 when operating at high-speed)
8	00	Vendor ID
9	00	
10	04	Product ID
11	00	
12	01	Device release number 1.01
13	01	
14	01	Index of string descriptor describing manufacturer
15	02	Index of string descriptor describing product
16	00	Index of string descriptor describing the device's serial number. (No string)
17	01	Number of possible configurations at the current operating speed

Device Qualifier Descriptor		
Offset	***Data (hex)***	***Description and interpretation***
0	0A	Size of this descriptor in bytes
1	06	Device qualifier type
2	00	USB specification release 2.00 (High-speed capable device)
3	02	
4	FF	Vendor-specific class (not registered with USB-IF)
5	FF	Vendor-specific subclass class (not registered with USB-IF)
6	FF	Vendor specific protocol class (not registered with USB-IF)
7	08	Maximum packet size for endpoint zero for other speed (8 when operating at high-speed)
8	00	Number of other-speed configurations
9	00	Reserved for future use.

Configuration Descriptor		
Offset	***Data (hex)***	***Description and interpretation***
0	09	Size of this descriptor in bytes
1	02	CONFIGURATION descriptor type
2	2E	Total length of data returned for this configuration (this configuration + one interface descriptor + four endpoints)
3	00	
4	01	Number of interfaces supported by this configuration
5	01	Configuration number
6	00	Index of string descriptor describing this configuration (no string)
7	D6	Self-powered.

8	00	Does not use power from the USB bus.

Other_Speed_Configuration Descriptor		
Offset	***Data (hex)***	***Description and interpretation***
0	09	Size of this descriptor in bytes
1	07	Other_Speed_Configuration descriptor type
2	2E	Total length of data returned for this configuration
3	00	
4	01	Number of interfaces supported by this configuration
5	01	Configuration number
6	00	Index of string descriptor describing this configuration (no string)
7	D6	Self-powered.
8	00	Does not use power from the USB bus.

Interface Descriptor 0		
Offset	***Data (hex)***	***Description and interpretation***
0	09	Size of this descriptor in bytes
1	04	INTERFACE descriptor type
2	00	Number of this interface
3	00	Alternate settings
4	04	Number of endpoints (excluding endpoint 0 default control pipe)
5	FF	Interface class code
6	FF	Interface subclass code
7	FF	Interface protocol
8	00	Index of string descriptor

Endpoint Descriptor 1 (device to host direction)		
Offset	***Data (hex)***	***Description and interpretation***
0	07	Size of this descriptor in bytes
1	05	ENDPOINT descriptor type
2	81 or 82	IN, EP1, most ComBlocks COM-16xx
3	02	Attribute: Bulk, data endpoint
4	40	Maximum packet size: 64
5	00	
6	00	No polling in this direction

Endpoint Descriptor 2 (host to device direction)		
Offset	***Data (hex)***	***Description and interpretation***
0	07	Size of this descriptor in bytes
1	05	ENDPOINT descriptor type
2	02	OUT, EP2
3	02	Attribute: bulk, data endpoint
4	40	Maximum packet size: 64
5	00	
6	00	No polling in this direction

Endpoint Descriptor 3 (device to host direction)		
Offset	***Data (hex)***	***Description and interpretation***
0	07	Size of this descriptor in bytes
1	05	ENDPOINT descriptor type
2	83, or 85	IN, EP3, most ComBlocks COM-16xx
3	02	Attribute: Bulk, data endpoint
4	40	Maximum packet size: 64
5	00	
6	00	No polling in this direction

Endpoint Descriptor 4 (host to device direction)		
Offset	***Data (hex)***	***Description and interpretation***
0	07	Size of this descriptor in bytes
1	05	ENDPOINT descriptor type
2	04	OUT, EP4
3	02	Attribute: bulk, data endpoint
4	40	Maximum packet size: 64
5	00	
6	00	No polling in this direction

8.2.3 Constraint File

Timing of the60 MHz interface between the NGC component and the USB PHY is critical. The following constraints should be added in the .ucf constraint file (using the PACE editor for example) to ensure proper timing:

```
NET "USB_CLK60" TNM_NET = "USB_CLK60";
TIMESPEC "TS_USB_CLK60" = PERIOD "USB_CLK60" 16 ns HIGH 50 %;
 # 60 MHz clock period is 16.666 ns

NET "USB_TXVALID" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<0>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<1>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<2>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<3>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<4>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<5>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<6>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
NET "USB_DATA_OUT<7>" OFFSET = OUT 10 ns AFTER "USB_CLK60"  ;
 # requested output delay for the DATA_OUT bus and USB_TXVALID is 10ns. 11 ns is generally acceptable.

NET "USB_DATA_OUT<0>"  LOC = "FPGA pin number" | DRIVE = 24  | SLEW = FAST ;
NET "USB_DATA_OUT<1>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
NET "USB_DATA_OUT<2>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
```

```
NET "USB_DATA_OUT<3>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
NET "USB_DATA_OUT<4>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
NET "USB_DATA_OUT<5>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
NET "USB_DATA_OUT<6>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
NET "USB_DATA_OUT<7>"  LOC = " FPGA pin number " | DRIVE = 24  | SLEW = FAST ;
 # Increase the output drive for DATA_OUT to minimize the output delay.
```

8.2.4 Synthesis Statistics

The FPGA size occupied by the USB20 component is as follows (and percentage utilization in the case of a Xilinx Virtex-2 1000 FPGA):

```
Logic Utilization:
Number of Slice Flip Flops: 620 out of 10,240 6%
Number of 4 input LUTs: 1,351 out of 10,240 13%
Logic Distribution:
Number of occupied Slices: 854 out of 5,120   16%
Number of Slices containing only related logic: 854 out of 854 100%
Number of Slices containing unrelated logic:  0 out of 854    0%
Total Number 4 input LUTs: 1,449 out of 10,240   14%
Number used as logic: 1,351
Number used as a route-thru: 98
Number of Block RAMs: 5 out of      40   12%

Total equivalent gate count for design:  342,354
```

9 Troubleshooting help

In case of any problems encountered during the communication setup please try the following:

- Check the version number of the driver to be 3.0 or above (Go to - Control Panel -> System -> Hardware -> Device Manager -> Other Devices -> comblock_usb -> Driver)
- Make sure the cable is not too long typically around not more than 5 feet.
- The cable has to be USB 2.0 compliant

During FPGA integration, the following test points can be of some help in debugging a non-responsive USB connection:

a) VBUS_SENSE goes high when a cable connects the ComBlock module and a computer. This low-tech test point is simply based on the detection of +5V on the cable.

b) The outcome of the speed auto-negotiation is shown on USB_TP(1) ('1' if high-speed) or USB_TP(2) ('1' if full-speed).

c) Following speed auto-negotiation, activity on test point USB_TP(3) indicates that error-free data packet are received over the USB connection by the ComBlock.

d) The host computer then tries to read the descriptor tables to identify which driver to load. This is visible by activity on test point USB_TP(4).

e) Some read failures are detected by the PHY and flagged by the USB_RXERROR signal.

f) Failure of the host to read the descriptor table (and thus inability to load the proper driver) could be traced to excessive delay when the 8-bit output data is transferred from the FPGA to the USB PHY. Probe DATA (7:0) and compare with the 60 MHz reference clock at the PHY. The DATA signal should be stable 8 ns before the rising edge of the 60 MHz reference clock. If so, timing constraints should be adjusted in the constraint file.

g) Flow control issues between the user VHDL code and the USB connection can be traced by looking at the most significant address bits of the elastic buffers embedded within the USB component. See test points USB_TP(10:5). When properly working, the most significant address bits on the read side and write side of the elastic buffer should move in unison (i.e. the read pointer never passes the write pointer).

Throughput:

The USB 2.0 device sustained (average) throughput was measured using one-way data transfer benchmarks as shown below:

Throughput test conditions	***Throughput***
High speed. Host computer: Intel Pentium 4 2.8 GHz. C runtime application, no hard disk data transfers. No other application running.	86 Mbits/s (either direction)
Full speed. Host computer: AMD Duron processor 850 MHz. C runtime application, no hard disk data transfers. No other applications running.	6.5 Mbits/s (either direction)