



XAPP1263 (v1.0) August 17, 2015

# AXI USB 2.0 Device: Demonstrating Performance for Bulk and Isochronous Transfers

Author: Prasad Gutti

## Summary

This application note demonstrates the performance measurement of the Xilinx universal serial bus 2.0 (USB 2.0) [Ref 1] high-speed device with an advanced microcontroller bus architecture (AMBA®) advanced eXtensible interface (AXI) for bulk and isochronous transactions. The generated test system is based on a Kintex® UltraScale All Programmable SoC. The performance is measured with two separate host drivers for bulk and isochronous transactions. This application note describes how to develop a USB system and corresponding ELF files for bulk and isochronous transactions. It is assumed that you have general knowledge of the Xilinx Vivado® Design Suite and Vivado IP Integrator [Ref 2].

## Introduction

The AXI USB 2.0 device enables USB connectivity for a design using a minimal amount of resources. This interface is suitable for USB-centric, high-performance designs, bridges, and legacy port replacement operations. The USB 2.0 protocol connects many devices over a single, half-duplex, serial bus. The AXI USB 2.0 device runs at 480 Mb/s (high speed) or at 12 Mb/s (full speed), and is designed to be plug-and-play. The host controls the bus and sends tokens to the device specifying the required action. The AXI USB 2.0 device supports up to eight endpoints, including default control endpoints that are sources or sinks of data. Endpoint 0 of the USB 2.0 device is used to enumerate the device with control transactions.

The seven user endpoints of the AXI USB 2.0 device can be configured as bulk, interrupt, or isochronous. Also, endpoints can be configured as input (to the host) or output (from the host). Each of these endpoints has two ping-pong buffers of the same size for endpoint data. The user endpoints data buffers are unidirectional and are configured by the endpoint configuration and status register of the respective endpoint. The size of the buffers can be configured from 0 to 512 bytes for bulk, 64 bytes for interrupt, and up to 1,024 bytes for isochronous endpoints.

This application note is targeted to demonstrate USB bulk and isochronous transactions performance for Kintex UltraScale devices. The following elements are used in the application:

- A MicroBlaze™ processor reference system with an AXI USB 2.0 device and DDR controller.
- Executable and linkable format (ELF) files that configure the device as bulk or isochronous endpoints.
- Host application used to measure the USB device performance.

The AXI USB 2.0 device consists of a built-in direct memory access (DMA) controller to transfer the data from endpoint buffers to the memory and vice versa. The USB 2.0 device core consists of a dedicated AXI4 slave interface and an optional AXI4 master interface. The optional master interface is configured through the device parameter Enable DMA Support. When the AXI USB 2.0 Controller is configured with DMA support, the DMA logic is included in the design. When the DMA logic is included, the user endpoint buffer location is not accessible from the AXI slave interface. When the DMA is disabled, the endpoint buffer location can be accessed by the AXI slave interface.

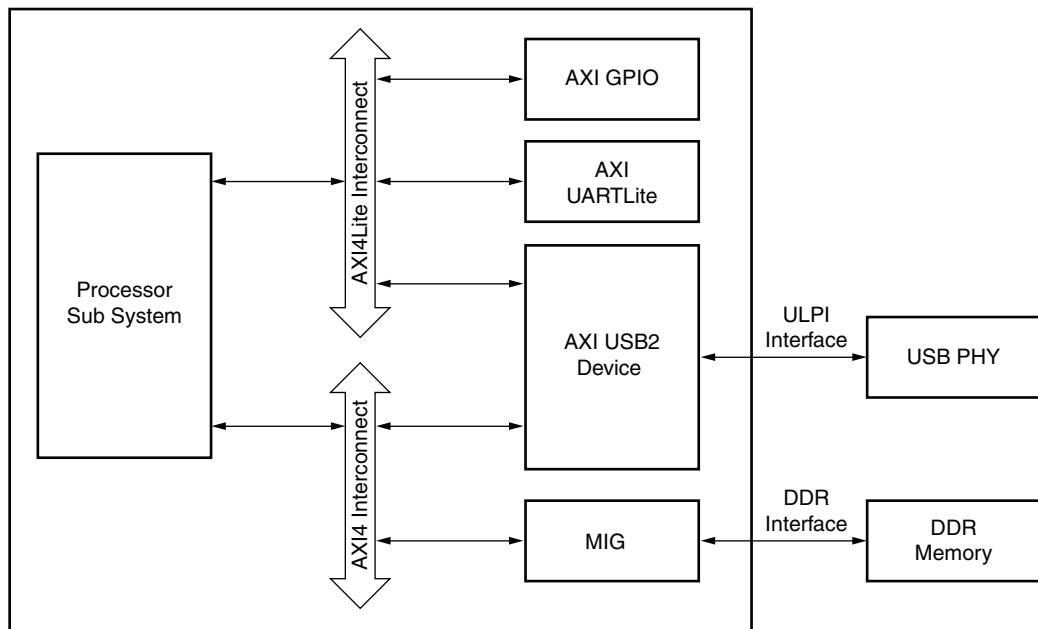
# Reference Design

## Hardware

The system is based on AXI, which is a standardized IP interface protocol based on the AMBA specification. The AXI4 mapped memory and AXI4-Lite interfaces are used in the reference design. A clock generator and a processor system reset block supplies clocks and resets throughout the system. An embedded MicroBlaze system containing I/O peripherals and processor support IP would provide High-level control of the system.

To optimize the system to balance performance and area, multiple AXI interconnect blocks are used to implement segmented or hierarchical AXI interconnect networks with each AXI interconnect block individually tuned and optimized.

A typical system for the AXI USB 2.0 device application is shown in [Figure 1](#).



X891\_01\_100912

Figure 1: Typical System for AXI USB 2.0 Device Application

### Using AXI USB 2.0 Device

The AXI USB 2.0 device IP can be connected on an AXI-based system with a 32-bit data width. The IP supports an ULPI interface on the USB PHY side. The IP can be configured in DMA mode or non-DMA mode with `Enable DMA Support` parameter. When the DMA is enabled, the AXI master interface is enabled. The AXI4 master interface must be connected to an AXI4 interconnect to which a DDR controller is connected as slave. The interrupt pin of the IP should be connected to the interrupt controller block of the processor system. The parameters used to configure the IP are:

- **Enable DMA Support:** When set to 1, a built-in DMA block is included in the design along with the AXI4 master interface. This DMA block disconnects the user endpoint buffers from the AXI slave interface.
- **Enable Un-aligned Transactions:** When set to 1, data realignment support is added to the design.
- **ULPI PHY Reset Polarity:** This parameter must be set according to the PHY requirements. When set as ACTIVE\_LOW, the ULPI reset generated by the AXI USB 2.0 device IP is an active-Low signal. When set to ACTIVE\_HIGH, the ULPI reset generated by the AXI USB 2.0 device IP is an active-High signal.
- **Enable USB Error Counters:** When set to 1, the USB link error logging counters are added to the design.
- **ENABLE HSIC PHY:** This parameter enables the HSIC PHY module and adds the HSIC interface at the I/O level. When this parameter is enabled, the ULPI interface is not available at the I/O level.
- **Connect Signaling Width:** This parameter is valid only when ENABLE HSIC PHY is set to 1. This parameter indicates the number of clock cycles for which connect signaling is sent by the HSIC PHY.

## Software

### *Mass Storage Application*

The mass storage application uses bulk transactions. It requires two endpoint buffers configured as bulk-in and bulk-out endpoints and a maximum packet size set to 512 bytes. The bulk-in endpoint is used to measure the performance for a read operation and the bulk-out endpoint is used to measure the performance for a write operation.

The bulk application initializes the system to enable the USB and UART devices, and then responds to the device enumeration commands (SET/GET descriptor) from the host. If the device receives a read request from the host after the enumeration is completed, the following process is initiated:

1. Data is written into the ping/pong buffers present in the USB device through DMA.
2. Upon receiving the IN token from the host device, the device responds with the data present in the ping/pong buffer.
3. An interrupt is sent to the processor reference system to indicate a buffer free condition for the corresponding endpoint.
4. The processor reference system detects the buffer free interrupt and allows the next data to be written into the ping/pong buffer.

This process is repeated as long as the host sends data.

The following process will initiate once the device receives a write command:

1. The host device sends an OUT token followed by a data packet.

2. Upon reception of the data, the received data is written into the ping/pong buffer.
3. An interrupt is send to the processor reference system to indicate a buffer full condition for the corresponding end-point.
4. The processor reference system reads the data and stores it into the DDR through the DMA.
5. This procedure is repeated as long as the host sends data.

### ***Isochronous Application***

The isochronous application is tested for Read and Write with separate applications. The isoc\_in application tests the read operation using isochronous transactions, and the isoc\_out application measures the performance of the write operation. The isoc\_in application programs the device as a microphone and uses one endpoint as an isochronous In endpoint. The isoc\_out application programs the device as a speaker and uses one endpoint as an isochronous out endpoint.

The isochronous application initializes the system to enable the USB and UART devices, and then responds to device enumeration commands (SET/GET descriptor) from the host. If the device is configured as a microphone, the following process would commence after the enumeration is completed:

1. Data is written into the ping/pong buffers existing in the USB device through the DMA.
2. Upon receiving the IN token from the host device, the device responds with the data that is existing in the ping/pong buffer.
3. An interrupt is sent to the processor reference system to indicate a buffer free condition for the corresponding endpoint.
4. The processor reference system detects the buffer free interrupt and allows the next data to be written into the ping/pong buffer.
5. This process is repeated as long as the host sends data.

The following process would trigger off if the device is configured as a speaker:

1. The host device sends an OUT token followed by a data packet.
2. Upon reception of the data, the received data is written into the ping/pong buffer.
3. An interrupt is send to the processor reference system to indicate a buffer full condition for the corresponding end-point.
4. The processor reference system reads the data and stores it into the DDR through the DMA.
5. This procedure is repeated as long as the host sends data.

---

# Requirements

## Hardware

The hardware boards and additional equipment required for this reference system include:

- Xilinx KCU105 production board
- Xilinx USB PHY FMC card with LPC connector
- JTAG platform cable USB
- One USB mini-B cable
- One micro USB cable

## Software

The installed design tool requirements for building and downloading this reference system are:

- Xilinx Vivado Design Suite 2015.1
- Xilinx Software Development Kit 2015.1
- Tera Term/putty terminal emulator for UART serial communication through a COM port.

## Host-Level Software Requirements

The host-side USB bulk performance monitor must have the following software:

- Windows XP (SP3) or higher (32-bit or 64-bit)
- .NET Framework 4

---

# Reference Design Files

You can download the reference design files from the following location:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=393319>

**Note:** Registration is required to access the reference design files.

Figure 2 shows the directory structure of the design files provided with the application note.

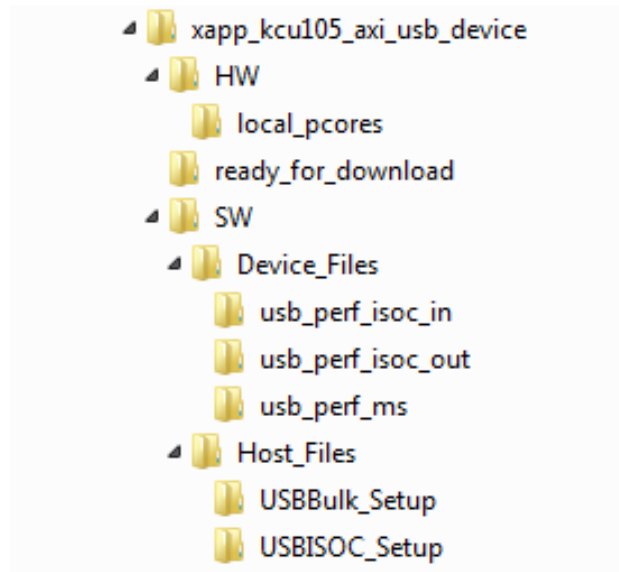


Figure 2: Directory Structures

**xapp1263\_kcu105\_axi\_usb\_device:** This is the top-level folder.

**HW:** This folder contains local\_pcores folder, all.tcl, and kcu105.xdc files.

- **local\_pcores:** This folder is empty and it can be used to include any local pcores that are being used by the system, if any.
- **all.tcl:** This file contains the Microblaze based complete hardware system for axi USB device using TCL commands.
- **kcu105.xdc:** This files contains LOC constraints required by System for AXI USB, GPIO and UART Lite IPs which in turn are required to connect to external peripherals through pins.

**Ready\_for\_download:** This folder contains *design\_1\_wrapper.bit*, *isoc\_in\_perf.elf*, *isoc\_out\_perf.elf*, and *bulk\_perf.elf* files. *design\_1\_wrapper.bit* should be programmed using FPGA XMD command. The *isoc\_in\_perf.elf*, *isoc\_out\_perf.elf*, and *bulk\_perf.elf* files should be downloaded using the dow XMD command.

**SW:** This folder contains the Device\_Files and HOST\_Files folders

- **Device\_files:** This folder includes bulk\_perf, isoc\_in\_perf, and isoc\_out\_perf applications that are used by the system to check the performance of USB.
- **HOST\_files:** This file contains the applications that are to be installed on HOST PC to monitor performance graphs of USB.



# Reference Design Steps

## Hardware Setup

Figure 3 shows the Hardware setup for throughput measurement.

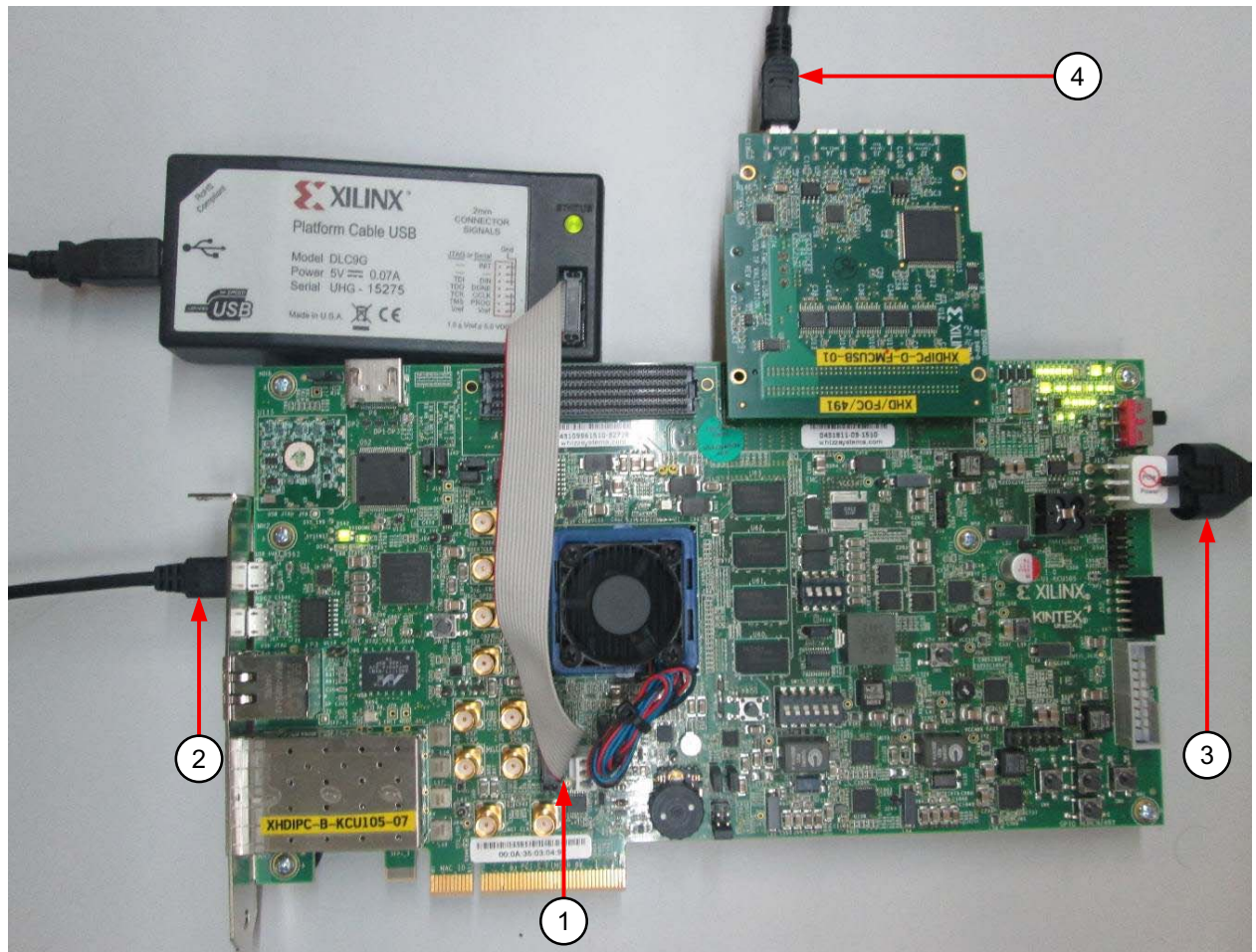


Figure 3: Hardware setup using KCU105 board

1. Connect a JTAG USB Platform cable or a USB Type A to Micro B cable from the host PC to the KCU105 board for programming bit and elf files. (In this setup, a JTAG Platform cable is used.)
2. Connect a Micro USB cable from the host PC to the USB UART port on the KCU105 for serial communication.
3. Power ON the board and then connect USB PHY FMC card on LPC port (J65) of KCU105 Board.

**Note:** When you power ON an Ultrascale board with FMC card already inserted in a slot, the FMC slots are not receiving power. As a workaround, power ON the board first and then insert the FMC card in the slot. See **FMC VADJ\_1V8 Power Rail** in UG917[Ref 3] for more information.

4. Connect the J65 USB port of the FMC card to the system on which the host application is run. Figure 3 shows the board setup.

5. Start a Putty program on the host PC with the following settings:
  - Baud rate: 9600
  - Data Bits: 8
  - Parity: None
  - Stop Bits: 1
  - Flow Control: None

---

## Executing the Reference Design

This section describes how to build the reference design for both hardware and software.

First, unzip the reference design files into a local directory. You can either build the reference design which includes generating the BIT and ELF files and test on the hardware or use the pre-generated BIT and ELF files available in the reference design and test on the hardware.

Perform the following to build the reference design and test it on the hardware:

- Build the Reference Design
- Execute the Design on the Hardware

While executing the design using the pre-generated BIT and ELF files available in the reference design and test on the hardware, pay particular attention to the note mentioned in step-2

### Building the Reference Design

The following section helps you to build and generate the hardware design which includes generating the BIT and ELF files and the SDK workspace.

#### ***Creating a Vivado Design Tools Project and generating the Bitstream***

This section details the steps to start a new Vivado Design Suite 2015.1 project.

1. Open the Vivado Design Suite.
2. Open the Tcl Console in the Vivado IDE (Click **Window->Tcl Console** if you do not see it.)
3. In the Tcl Console of the Vivado IDE, change to the HW directory:

```
cd xapp1263_axi_usb_device/HW
```

4. Source the given all.tclfile

```
source all.tcl
```

5. A Project is created and output products are generated. Design is synthesized, implemented and the bitstream is generated.



Figure 4 displays the address mapping of all the IP cores with the MicroBlaze™ processor in the system.

**Note:** If you plan to use the BIT and ELF files available with the reference design, skip this section and start at executing the design on the Hardware.

Address Editor						
Cell	Slave Interface	Base Name	Offset Address	Range	High Address	
microblaze_0						
Data (32 address bits : 4G)						
lmb_bram_if_cntlr_0	SLMB	Mem	0x0000_0000	128K	0x0001_FFFF	
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF	
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF	
mig_0	CO_DDR4_S_AXI	CO_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF	
axi_intc_0	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF	
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF	
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF	
axi_usb2_device_0	S_AXI	Reg	0x4480_0000	64K	0x4480_FFFF	
Instruction (32 address bits : 4G)						
lmb_bram_if_cntlr_1	SLMB	Mem	0x0000_0000	128K	0x0001_FFFF	
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF	
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF	
mig_0	CO_DDR4_S_AXI	CO_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF	
axi_usb2_device_0						
DMA_DATA (32 address bits : 4G)						
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF	
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF	
mig_0	CO_DDR4_S_AXI	CO_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF	

Figure 4: Base and High Addresses of the IP Cores Present in IP Integrator

## Export Hardware Workspace in SDK

Use the following steps to create an ELF file for bulk transactions. These same steps can be used to create ELF files for isochronous In and Out transactions. The source files are located in `bulk_perf`, `isoc_in_perf`, and `isoc_out_perf` folders under the `SW/Device_Files` directory.

1. Once the bitstream generation is completed, open the already exported hardware workspace in SDK.
2. Open the xsdk GUI.
3. Give the workspace path as `project_1/project_1.sdk` and click **OK**.
4. Add BSP to SDK. Click **File** --> **New** --> **Board Support Package**.
5. Create a new project.
  - a. Click **File** --> **New** --> **Application project**.
  - b. Type a new project name. Ex: `bulk_perf`
  - c. Select **Use Existing** button for BSP. Click **Next**.
  - d. Select Empty project and then click **OK**.
  - e. From the Project Explorer window, select the `src` folder under `bulk_perf` project.
  - f. Right-click the `src` directory and click **Import**.
  - g. Select **General** --> **File System**, and then click **Next**.

- h. Browse to the **SW/Device\_Files/bulk\_perf** directory and click **OK**.
- i. Select and import the following SW files and click **Finish**.
  - xusb\_cp9.c
  - xusb\_cp9.h
  - xusb\_storage.c
  - xusb\_storage.h
  - xusb\_types.h
- j. Now, an elf file will be generated.

The above procedure is applicable to generate ELF files for `isoc_in_perf` and `isoc_out_perf` applications.

## Installing Host Side Driver

This section describes how to install the host side driver.

Separate applications are provided for bulk and isochronous transaction performance measurement. For the isochronous transactions performance measurements, go to the `SW/Host_Files/USBISOC_Setup` directory. For bulk transaction performance measurements, go to the `SW/Host_Files/USBBulk_Setup/` directory. Install both the applications separately whenever you initiate a particular performance measurement.

Double-click the Xilinx USB Perfmon Setup x86.msi file for a 32-bit machine or Xilinx USB Perfmon Setup x64.msi for a 64-bit machine. Follow the instructions to complete the installation.

**Note:** For 64-bit Windows 7 operating system, after starting the PC and when Windows starts loading, keep pressing the F8 key to open the Advanced Boot Options. In the Advanced Boot Options prompt, disable the Disable Driver Signature Enforcement by setting it to No.

**Note:** Ensure that the board is configured with a \*.bit file and that the \*.elf file is loaded before installing the drivers.

If the application does not work, load the MicroBlaze processor application again. It might need to be loaded several times until it works (maximum 2-3 times.)

## Executing the Design on the Hardware

Use the following steps to execute the bitstream and elf files on the hardware setup.

1. Connect the JTAG cable and USB-UART cable to the board.
2. Create `test` folder.



---

**IMPORTANT:** Use the `ready_for_download` folder instead of creating a `test` folder to access the pre-generated BIT and ELF files available in the Reference Design. Skip the steps 3,4 and 5 and start from Step-6.

---

- Copy the bit file from the following location to the `test` folder:

```
project_1/project_1.runs/impl_1/design_1_wrapper.bit
```

- Copy the elf files from the following location to the `test` folder:

```
project_1/project_1.sdk/bulk_perf/Debug/bulk_perf.elf,
project_1/project_1.sdk/isoc_in/Debug/isoc_in.elf,
project_1/project_1.sdk/isoc_out/Debug/isoc_out.elf
```

- In the command prompt, type the following command to open the `test` folder:

```
cd test
```

- Type `xmd` in the command prompt to start the Xilinx Microprocessor Debugger (XMD).

- Configure the FPGA with `design_1_wrapper.bit` through a JTAG cable using the following command at the XMD prompt:

```
fpga -f design_1_wrapper.bit
```

- Connect to the processor running on the FPGA using the following command at the XMD prompt:

```
connect mb mdm
```

- Reset and stop the FPGA using the following commands at the XMD prompt:

```
rst
stop
```

- Open the hyperTerminal and configure it to 9600 baud with default configuration to observe the results. Ensure that the UART cable is connected to the board and the PC.

### ***Configuring the Board as Bulk USB:***

- Download the elf file into memory (block RAM or DDR) and execute the software on board.

```
stop; rst; dow bulk_perf.elf; run
```

- If USB enumerated is not displayed, run the previous commands again.

### ***Configuring the Board as Isoc USB***

- Type the following command on the XMD terminal for Isoc-in:

```
rst;stop;dow isoc_in.elf;run
```

The hardware is ready to be used as an In-mode isochronous device.

- Type the following command on the XMD terminal for Isoc-out:

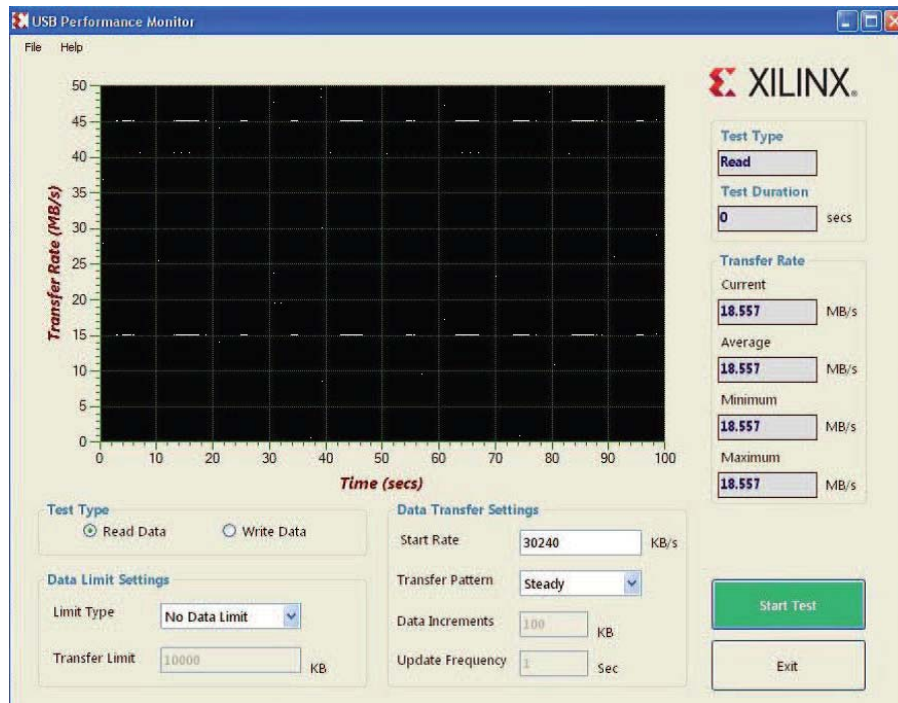
```
rst;stop;dow isoc_out.elf;run
```

- When the device is ready, you can execute the application with `UsbPerform.exe`. After the full configuration is completed, carefully remove all the USB ports except the one that is connected to the FMC card. Execute `UsbPerform.exe`.

**Note:** The start rate should not be less than 1024K and the rate/graph can vary based on the USB hub transactions. Normally, it should not vary more than 10%. However, it can change significantly if a new USB device is plugged into the system.

## Results

When you execute the `UsbPerfmon.exe` after the driver is installed and the system is detected, the USB Performance Monitor as shown in [Figure 5](#) is displayed.



*Figure 5: USB Performance Monitor*

You can change the transfer rates and type of test through the GUI for the performance measurement software. The options that you can change are provided at the bottom of the display. The outputs are on the right side of the display. You can set the following parameters:

- **Test Type:** The test can be carried out in Read or Write mode. The throughput measurement for Read and Write transactions can be measured separately.
- **Data Limit Settings:** The maximum data to be transmitted per transaction can be constrained through this tab.
- **Data Transfer Settings:** Use this tab to change the transfer rate settings. The Start Rate option indicates the starting transfer rate from the host. Change pattern of the transfer rate can be set through the transfer pattern setting option.

## Performance Measurement for Bulk Transactions

Performance can be measured for Read and Write operations separately. For a Write operation, the performance measurement is determined for the following GUI parameters:

- Start Rate = 40,024 KB/s
- Transfer Pattern = Steady

With these settings, a maximum performance of was 18.005 obtained with the AXI USB 2.0 device as shown in [Figure 6](#).



**Figure 6: Performance Measurement for Bulk Transaction (40024 KB/s start rate)**

As shown in [Figure 6](#), the current reading displays the performance measured at that instance (for one second) in the transfer rate section. The average reading shows the average of the performance numbers calculated until that point of time. The minimum reading shows the minimum value of the performance number measured over the time. The maximum reading shows the maximum value of the performance number measured until that second of the time.

[Figure 7](#) displays the results for these GUI parameter settings:

- Start Rate = 1,024 KB/s
- Transfer Pattern = Fixed Increment
- Data increments = 1,000 KB
- Update frequency = 1 sec (frequency at which data rate is increased)

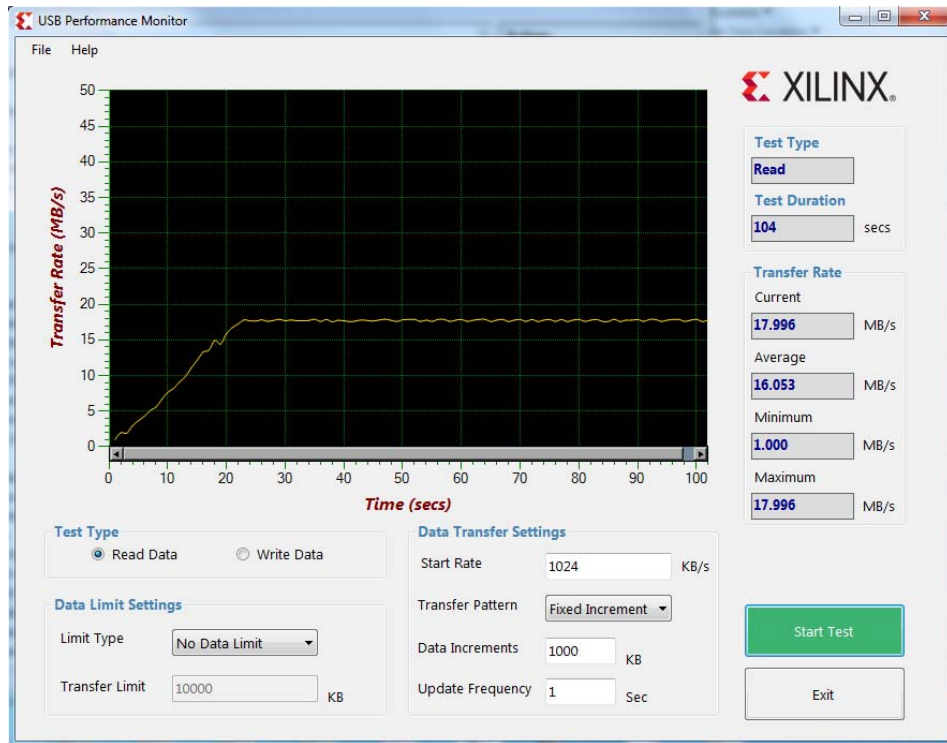


Figure 7: Performance Measurement for Bulk Transaction (1024 KB/s)

**Note:** The performance numbers vary depending on the number of other USB devices connected to the system, and the number of packets the host can initiate in a micro frame.

## Performance Measurement for Isochronous Transactions

Figure 8 displays the performance measurement for an isochronous Out transaction with a starting data rate of 1,024 KB/s and fixed increment pattern. The maximum throughput achieved is 23.067 MB/s and the theoretical maximum throughput is 24 MB/s.



Figure 8: Performance Measurement for Isochronous Out Transaction

Figure 9 displays the performance measurement for an isochronous In transaction with a starting data rate of 1,024 KB/s and fixed increment pattern. The maximum throughput achieved is 22.906 MB/s and the theoretical maximum throughput is 24 MB/s.



Figure 9: Performance Measurement for Isochronous In Transaction



## Reference Design Details

The reference design files can be downloaded from the following location:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=393319>

Registration is required to access the reference design files.

### AXI-USB Reference Design

Table 1 shows the reference design checklist for this application note.

Table 1: Reference Design Matrix

Parameter	Description
<b>General</b>	
Developer name	Prasad Gutti
Target devices	Kintex UltraScale FPGA XCKU040-FFVA1156-2-E
Source code provided	Yes
Source code format	VHDL/Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	Reference designs provided for SDK and cores generated from Vivado IP catalog.
<b>Simulation</b>	
Functional simulation performed	N/A
Timing simulation performed	N/A
Test bench used for functional and timing simulations	N/A
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
<b>Implementation</b>	
Synthesis software tools/versions used	Vivado Design Suite: System Edition 2015.1
Implementation software tools/versions used	Vivado Design Suite: System Edition 2015.1
Static timing analysis performed	Yes (Passing timing in PAR/TRCE)
<b>Hardware Verification</b>	
Hardware verified	Yes
Hardware platform used for verification	Kintex UltraScale FPGA KCU105 evaluation kit

---

## References

This section lists the references used in this document.

1. LogiCORE IP AXI Universal Serial Bus (USB) 2.0 Device v5.0 Product Guide ([PG137](#))
2. Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator ([UG994](#))
3. KCU105 Board User Guide([UG917](#))

---

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/17/2015	1.0	Initial Xilinx release.

---

## Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

© Copyright 2011—2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.