USB20HR Nios II HAL API

User Guide

System Level Solutions, Inc. (USA)
14100 Murphy Avenue
San Martin, CA 95046
(408) 852 - 0067

http://www.slscorp.com

HAL API Version:	1.3
Document Version:	1.4
Document Date:	April 2008

IP Usage Note

The Intellectual Property (IP) core is intended solely for our clients for physical integration into their own technical products after careful examination by experienced technical personnel for its suitability for the intended purpose.

The IP was not developed for or intended for use in any specific customer application. The firmware/software of the device may have to be adapted to the specific intended modalities of use or even replaced by other firmware/software in order to ensure flawless function in the respective areas of application.

Performance data may depend on the operating environment, the area of application, the configuration, and method of control, as well as on other conditions of use; these may deviate from the technical specifications, the Design Guide specifications, or other product documentation. The actual performance characteristics can be determined only by measurements subsequent to integration.

The reference designs were tested in a reference environment for compliance with the legal requirements applicable to the reference environment.

No representation is made regarding the compliance with legal, regulatory, or other requirements in other environments. No representation can be made and no warranty can be assumed regarding the suitability of the device for a specific purpose as defined by our customers.

SLS reserves the right to make changes to the hardware or firmware or software or to the specifications without prior notice or to replace the IP with a successor model to improve performance or design of the IP. Of course, any changes to the hardware or firmware or software of any IP for which we have entered into an agreement with our customers will be made only if, and only to the extent that, such changes can reasonably be expected to be acceptable to our customers.

ug_ipusb20hr_halapi_1.4

About this Guide

Introduction

The document provides information on how to use USB20HR Nios II HAL driver API and steps to create application.

Table below shows the revision history of this user guide.

Version	Date	Description
1.4	March 2008	• Replaced USB 2.0 with USB20HR
1.3	September 2007	• Added code for the board, Modified version history & Document part no. Modified the API Function Example.
1.2	March 2007	Second publication of USB2.0 NIOS Programmer's Guide
1.1	April 2006	First Publication of the USB2.0 Nios Programmer's Guide

How to find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Use Ctrl + F to open the Find dialog box. Use Shift + Ctrl + N to open to the Go To Page dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature preview of each page, provide a link to the pages.
- Numerous links shown in Navy Blue color allow you to jump to related information.

How to Contact SLS

For the most up-to-date information about SLS products, go to the SLS worldwide website at http://www.slscorp.com. For additional information about SLS products, consult the source shown below.

Information Type	E-mail
Product literature services, SLS literature services, Non-technical customer services, Technical support.	support@slscorp.com

Typographic Conventions

The user guide uses the typographic conventions as shown below:

Visual Cue	Meaning
Bold Type with Initial Capital letters	All headings and Sub headings Titles in a document are displayed in bold type with initial capital letters; Example: **Creating a Project, Building and Managing Project.**
Bold Type with Italic Letters	All Definitions, Figure and Example Headings are displayed in Italics. Examples: **Example1-1. Read Byte from SLSUSB Device**, *Figure 2-1. Creating a New Project*
1., 2.	Numbered steps are used in a list of items, when the sequence of items is important. such as steps listed in procedure.
•	Bullets are used in a list of items when the sequence of items is not important.
	The hand points to special information that requires special attention
	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
	The warning indicates information that should be read prior to starting or continuing the procedure or processes.
	The feet direct you to more information on a particular topic.

Contents

1. HAL API Reference

The USB20HR device driver has implemented standard functions that application uses to communicate with USB 2.0 device. It is implemented on Nios II platform using ALTERA HAL Driver API. This driver is designed to support device configuration BULK IN/OUT and CONROL endpoint.

This API uses mainly four Altera HAL exported interfaces to communicate with USB 2.0 device.

1. `open()`
2. `read()`
3. `write()`
4. `close()`

For more information about open() and close() functions, please refer to the chapter 4 of Nios II Software Developer's Handbook.

All calls for read and write operation are blocking calls for USB20HR device driver. Therefore, whenever read call will be made, it will not return until the requested data is available from USB 2.0 Device. Same for the write, whenever any write call will be issued, it will try to write the data until the hardware (USB2.0 device) buffer is empty. As soon as the hardware buffer gets full, the write function will wait until the buffer empty. Once requested data is written, it will return to the requester with no of bytes sent.

Whatever data the user wants just make a pointer of that size in the application file on Nios side so that it will get the data from the IP's buffer up to the number of bytes that user defines.

The section below explains the API functions.

open()

Protocol: `int open (const char* pathname,int flags)`

Commonly called by: C/C++ programs

Include: <unistd.h>

Return: The return value is zero upon success, and -1 otherwise.

Description: The `open()` function opens a file or device and returns a file descriptor (a small, non-negative integer for use in read, write etc.). flags is one of: O_RDONLY, O_WRONLY, or O_RDWR which request opening the file read-only, write-only or read/write, respectively.

Example: An example of `open()` function is given below:

```
int h_USB;
h_USB=open(USB20HR_0_NAME,O_RDWR);
```

read()

Protocol: `int read (int h_USB,void *ptr,size_t len)`

Commonly called by: C/C++ programs

Include: <unistd.h>

Return: This function reads data from USB 2.0 device and returns no. of bytes read successfully.

Description: The `read()` function reads block of data from USB 2.0 Controller. ALTERA HAL driver will search for the USB20HR NAME and will find the function pointer associated with `read()` and call that device API. Here `sls_avalon_usb20hr_read()` will be called.

Example: An example of `read()` function is given below:

Example 1-1. Reads Data from USB 2.0 Device

```
unsigned char read_buff[700]; //application data storage
buffer.

int count=700; //number of bytes to be read.

int read_count=0;

int h_USB;   //Store SLSUSB20HR driver handle

/* Get driver handle to access device */
h_USB = open(USB20HR_0_NAME,O_RDWR);

if(h_USB<0)
{
  printf("Error : SLSUSB2.0 Device is Not Present\n");
  return 0;
}
else
{
  printf("SLSUSB2.0 Device Opened Successfully\n");
}


/* Call Read API to read data from slsusb2.0 device*/
read_count = read(h_USB, read_buff, count);

printf("No of bytes read from SLSUSB2.0 %d\n",read_count);
```

write()

Protocol: `int write (int h_USB,void *ptr,size_t len)`

Commonly called by: C/C++ programs

Include: <unistd.h>

Return: This function will writes data to USB2.0 device and returns no of bytes written successfully.

Description: The `write()` function writes a block of data to SLS USB2.0 Controller. The input argument,`ptr`, is the location from where data will be written to device and `len` is the length of data to write in bytes. ALTERA HAL driver will search for the USB20HR NAME and find the function pointer associated with `write()` and call that device API. Here `sls_avalon_usb20hr_write()` will be called.

Example: An example of `write()` function is given below:

Example 1-2. Write Data to the USB2.0 Device

```
unsigned char write_buff[512]; //application data storage buffer.

int count=512; //Stores number of bytes to write.

int write_count=0;

int h_USB;   //Stores SLSUSB2.0 driver handle

/* Get driver handle to access device */
h_USB = open(USB20HR_0_NAME,O_RDWR);

if(h_USB<0)
{
  printf("Error : SLSUSB2.0 Device is Not Present\n");
  return 0;
}
else
{
  printf("SLSUSB2.0 Device Opened Successfully\n");
}


/* Call Write API to write data to slsusb2.0 device*/
write_count = write(h_USB,write_buff,count);

printf("No of bytes are written to SLSUSB2.0 %d\n",write_count);
```

close()

Protocol: `int close(int h_USB)`

Commonly called by: C/C++ programs

Include: <unistd.h>

Return: The return value is zero upon success, and -1 otherwise.

Description: The `close()` function is standard UNIX style `close()` function, which closes the file descriptor `fd`.

Example: An example of `close()` function is given below:

```
int h_USB;
h_USB=open(USB20HR_0_NAME,O_RDWR);
...........// add required code
close (h_USB);
```

usb20hr_config()

Protocol:

```
void usb20hr_config (volatile int base,
                     volatile int irq,
                     const char *name)
```

Commonly called by: C/C++ programs

Include: <sls_avalon_usb20hr.h>

Return: –

Description: The `usb20hr_config()` is used for initializing `base`, `irq` and `name` of the device, which are defined in system.h file

Example: An example of `usb20hr_config()` function is given below:

```
usb20hr_config(USB20HR_0_BASE,
               USB20HR_0_IRQ,
               USB20HR_0_NAME)
```

usb20hr_dma_usage()

Protocol:

```
void usb20hr_dma_usage(unsigned int usage,
                       unsigned int dma_address)
```

Commonly called by: C/C++ programs

Include: <sls_avalon_usb20hr.h>

Return: –

Description: The `usb20hr_dma_usage()` is used for initializing DMA usage. It is optional function. If you don't want to use DMA then automatically it will by- pass the DMA.This function needs to be define once and can affect for all read/write transaction. There is no need to define every time before read/ write.

Example: An example usage of `usb20hr_dma_usage()` in different situation is given below:

```
//If user wants to use DMA.
usb20hr_dma_usage(1,DMA_0_BASE);

//If user has included DMA in SOPC but do not want to
use DMA
usb20hr_dma_usage(0,DMA_0_BASE)

//If user don't want to use DMA or do not want to
include in SOPC.
usb20hr_dma_usage(0,0);
```

usb20hr_connect()

Protocol: `void usb20hr_connect()`

Commonly called by: C/C++ Programs

Include: <sls_avalon_usb20hr.h>

Return: –

Description: The `usb20hr_connect()` function is used for enumerating device to host. Without this function, enumeration of device is not possible. After enumerating the device, user can perform read/write on the device.

Example: An example of `usb20hr_connect()` function is given below:

```
usb20hr_connect()
```

usb20hr_disconnect()

Protocol: `void usb20hr_disconnect()`

Commonly called by: C/C++ Programs

Include: <sls_avalon_usb20hr.h>

Return: –

Description: The `usb20hr_disconnect()` function is used to disconnect the device from the host. If user wants to perform read/write, then they have to re- enumerate the device and for that the user have to use `usb20hr_connect()` to start enumeration of device.

Example: An example of `usb20hr_disconnect()` function is given below:

```
usb20hr_disconnect()
```

2. Example Usage

The following code shows the usage of simple USB20HR HAL API to access a USB2.0 device named ***/dev/usb20hr_0***, as defined in **system.h**.

Example 2-1. Example Usage of USB20HR HAL API

```
#include "stdio.h"
#include "io.h"
#include "system.h"
#include <fcntl.h>
#include "unistd.h"
#include "sls_avalon_usb20hr.h"
#define DATA 1024

extern void usb20hr_config (volatile int ,volatile int,const char *);
extern void usb20hr_dma_usage(unsigned int usage,unsigned int dma_address);
extern void usb20hr_connect();

unsigned short incomingData[DATA];
unsigned short outgoingMesage[DATA];

int h_USB; /* usb handle */
int main()
{
  unsigned int usb_base = USB20HR_0_BASE;
  usb20hr_config(usb_base,USB20HR_0_IRQ,USB20HR_0_NAME);
  h_USB = open(USB20HR_0_NAME,O_RDWR);
  if(h_USB<0)
  {
   printf("COULDN'T open usb");
  }
  usb20hr_dma_usage(1,DMA_0_BASE);//use the dma for the USB
  usb20hr_connect();
  count= write(h_USB,outgoingMesage,DATA);
  count= read(h_USB,incomingData,DATA);
  close(h_USB);
  return 0;
}
```

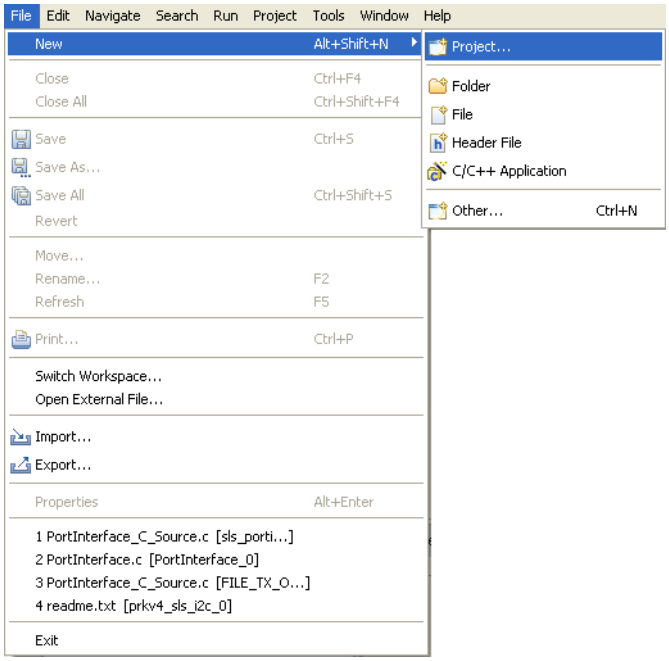
3. Designing the Project in Nios II IDE

Creating Project

The Nios II IDE provides a New Project wizard that guides you through the steps to create a new C/C++ application project.

1. To start the C/C++ application New Project wizard, choose **New (File menu)**, See Figure 3-1.

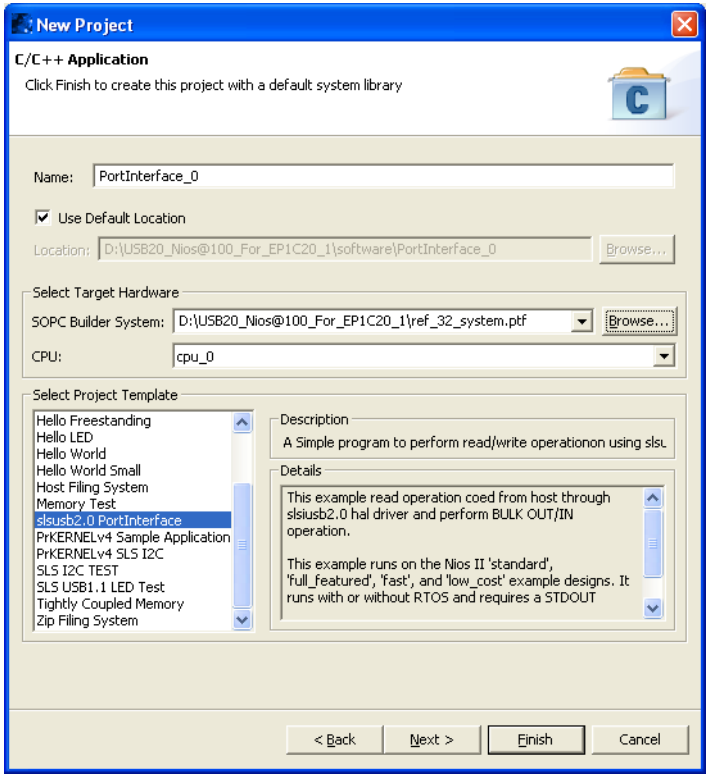
Figure 3-1. Creating New Project.



2. The C/C++ application New Project wizard prompts you to specify a name for your new project, the target hardware and template for the new project.
   - In Project templates select the "**SLSUSB2.0 PortInterface**"
   - By default the name of project will be "**PortInterface_0**". User can specify another name of the project.

- In SOPC Builder System select the **.ptf** file of your SOPC system.

Figure 3-2. Giving Name and Selecting SOPC System and Project Template



3. After you click **Finish**, the Nios II IDE creates the new project. The IDE also creates a system library project, ***_syslib**, for example **"PortInterface_0_syslib**".

Property Settings

Right click on **"PortInterface_0_syslib"**. It will open the Dialog **"Property for PortInterface_0_syslib** as shown in Figure 3-3.

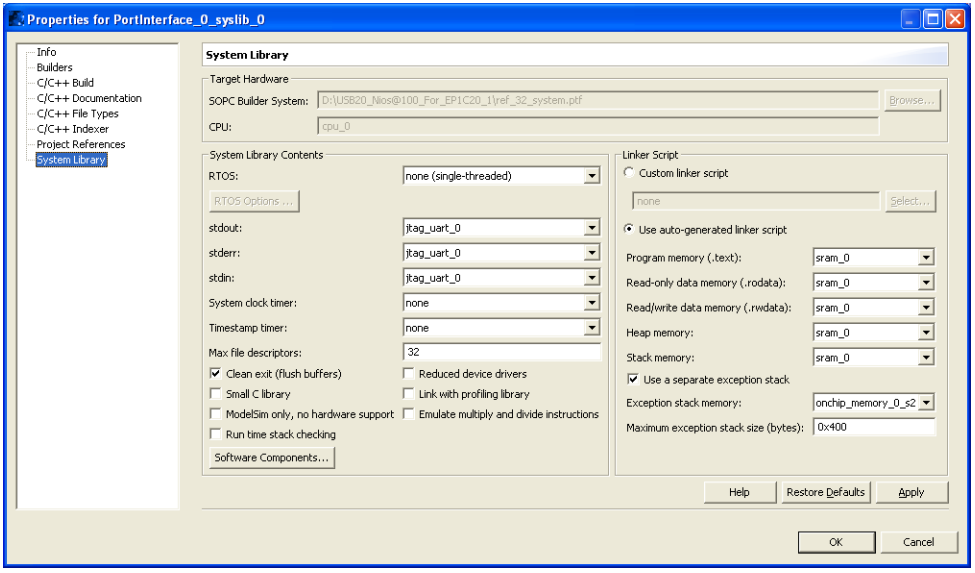
System Library Settings

1. Select **"System Library"** from the left view of the dialog.
   - Set the following values in the Linker Script See Figure 3-3.
     – **Program Memory(.text)**: sram_0

– **Read-only data memory**: sram_0
– **Read/Write data memory**: sram_0
– **Heap Memory**: sram_0
– **Stack Memory**: sram_0

- Now check the **“Use a separate exception stack“**check box. And set the following property
  – **Exception stack memory:** onchip_memory_0_s2
  – **Maximum exception Stack size (byte):** 0x400

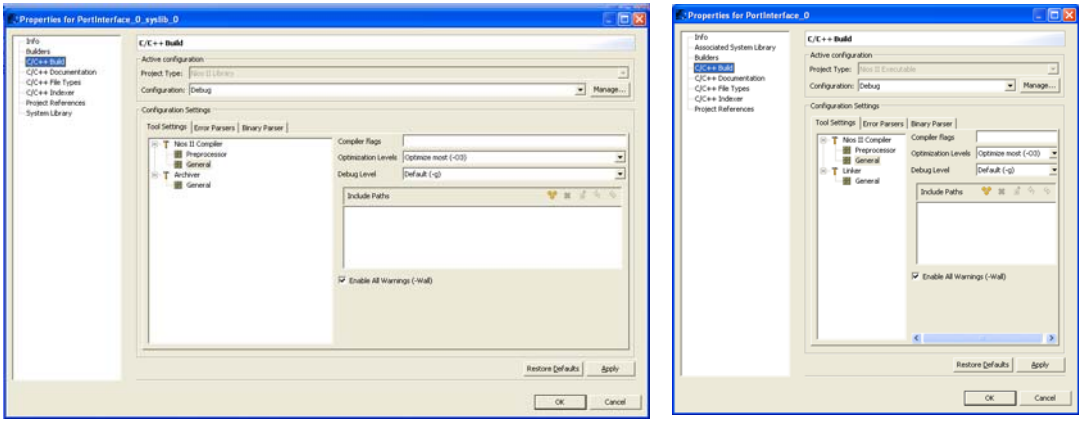
Figure 3-3. Portinterface_0_syslib settings - System Library



C/C++ Build Settings

1. Select the **“C/C++ Build”** to set optimization level of library from left view of the dialog.
2. Click on **“Tool Setting”** tab under the **Configuration Settings**.
   - Select the **“General”** under “**NIOS II Compiler**” tree.
     – Set **Optimization level:** -Optimize most(-O3).
3. Click on **Apply** and then **Ok** button.

Figure 3-4. Portinterface_0_syslib Settings-C/C+ Build

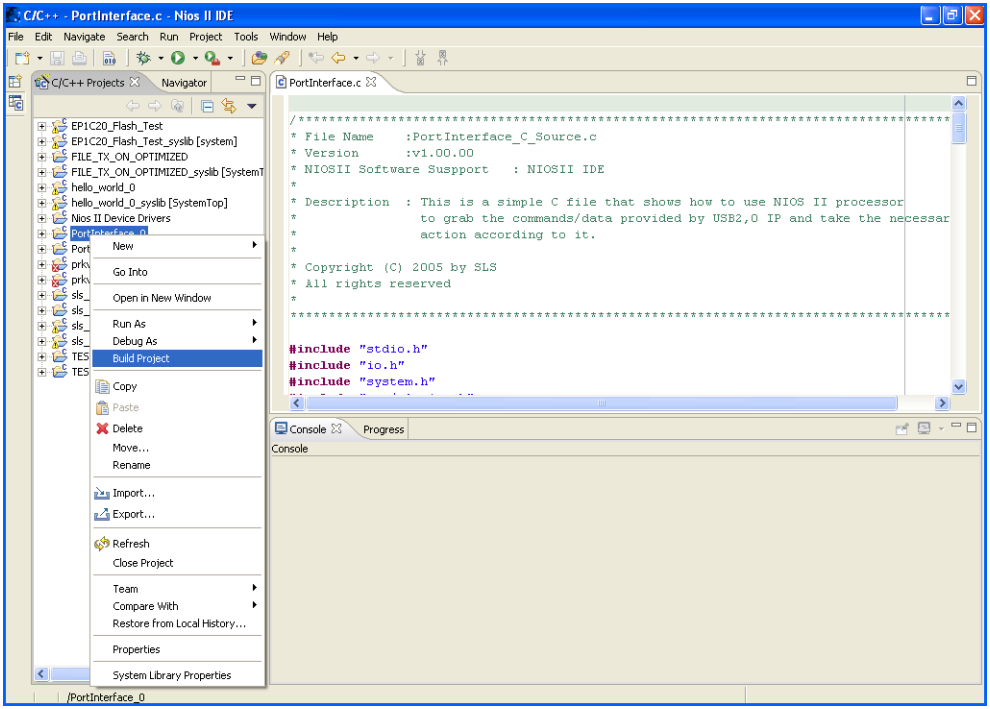


Building and Managing the Project

Right-clicking on any resource (a file, folder, or project) opens a context sensitive menu with operations, you can perform on the resource. Right-clicking is usually the quickest way to find the operation you need, though operations are also available in menus and toolbars.

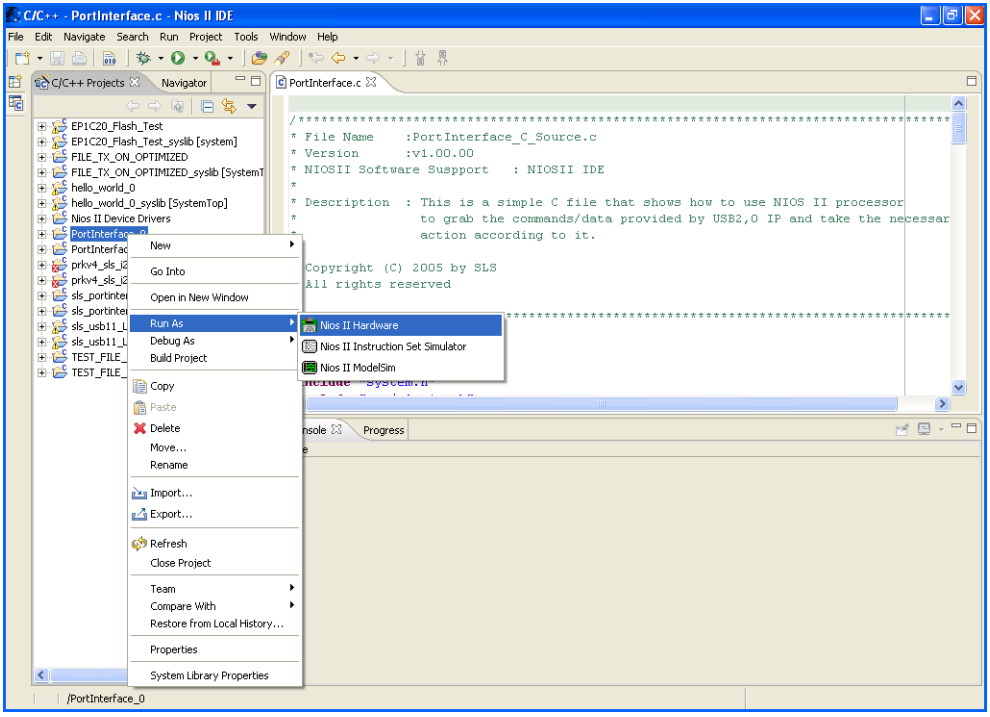
1. To compile a project, right-click the project in the C/C++ Projects view, and choose **Build Project.** as shown in Figure 3-5.

Figure 3-5. Build Project



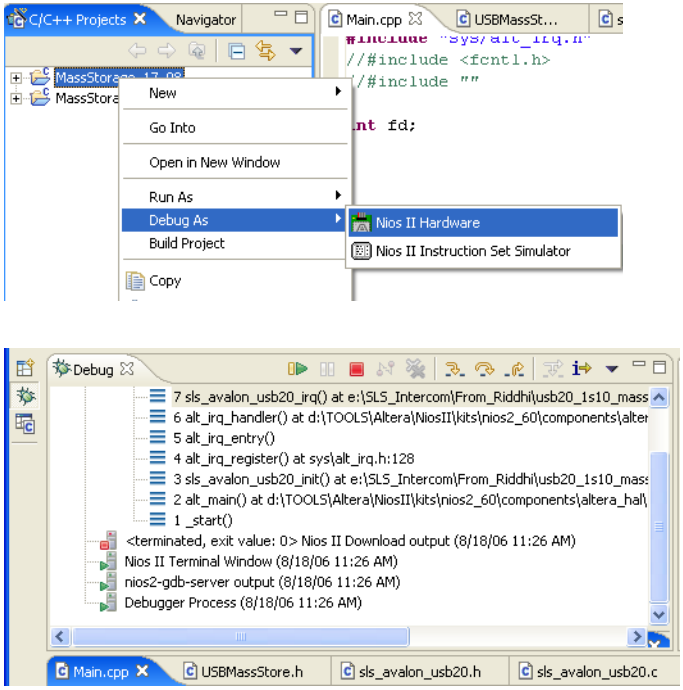
2. Run and debug operations are available by right-clicking the project. The Nios II IDE allows you to run or debug the project on a target board.
   - To run the program on a target board, choose **Run As > Nios II Hardware**. See Figure 3-6.

Figure 3-6. Run Project



- To debug the program, choose **Debug As >Nios II Hardware**. See Figure 3-7.

Figure 3-7. Debug Project



Annexure A: Bulk Out Configuration

Current configuration of HAL driver is **1**Kbytes of buffer depth, double buffering supported and Max payload size is **512 Bytes**.

☞ For 512 Bytes of buffer depth, there would be some minor change in HAL driver. Please contact us to make changes in HAL API.

If user has an idea about how to develop the application, like suppose he/she wants to use streaming in 256Bytes, 1KB, 2KB, 3KB, 800 Bytes etc. then they can use this configuration as it is.

Suppose they want to develop their own protocol and if data size of each packet is less then pay load size (here 512 Bytes) then also, they can use this configuration.

Now if user's protocol varies in combination of size; for e.g. For a particular device design, if protocol is less then pay load size or equal to pay load size or greater then payload size, then contact us for any such changes.