Altera Interlaken IP Throughput Measurement Reference Design for Stratix Devices

Altera Corporation

February 2012

Table of Contents

1 Overview

This application note describes a reference design that demonstrates Altera Interlaken IP throughput measurement for various size packets. It also provides a design example to demonstrate integrating the Interlaken IP core in user applications. The Interlaken IP variant in this reference design is configured with 20 lanes at 6.375 Gbps, providing a real time hardware environment to measure throughput for a 100G Ethernet packet application. A companion Microsoft Excel worksheet that is provided with the reference design calculates throughput for applications that use other Interlaken IP configurations. The worksheet provides throughput and user side clock frequencies for various transceiver configurations using different numbers of lanes and different line rates.

1.1 Features

This reference design offers the following features:

- The reference design uses internal loopback, to support transceiver channels that are not available for external loopback on the demonstration board.
- The reference design includes HDL code for supporting modules that you can reuse in your own design, such as a packet generator, a packet checker, a statistics keeper, and a performance meter, in addition to the Interlaken IP core.
- The design is easily modified for applications that use different Interlaken IP core variations.
- The design includes an embedded Qsys control unit using a Nios-II microprocessor, which enables users to configure, control, and retrieve status and performance information interactively from a PC through a USB connection.
- The design provides a reconfigurable packet generator to dynamically change traffic patterns.
- The current reference design targets an Altera Stratix IV GT device.

1.2 Demonstrated Altera Technology

This reference design demonstrates the following Altera technology:

- Interlaken IP core and its throughput performance
- Stratix IV GT devices
- Qsys system integration tool
- Nios II microprocessor

This application note includes the following sections:

2 Understanding Throughput in the Interlaken Protocol

Interlaken's scalable bandwidth is achieved by its ability to run on varying numbers of lanes at varying data rates. The effective bandwidth that corresponds directly with the number of lanes and data rate per lane is called raw aggregate link bandwidth. The following equation calculates the raw aggregate link bandwidth:

$$\text{raw aggregate link bandwidth} = \text{number of lanes} \times \text{data rate per lane}$$

The raw aggregate link bandwidth for a 20Lane x 6.375Gbaud Interlaken configuration is 125 Gbps. The raw aggregate link bandwidth is not the actual rate of packet transmission. Packets are framed and encoded before being transmitted. The overhead of the framing and encoding scheme costs a small percentage of link bandwidth. This document provides a throughput calculation at each node, as shown in Figure 1.

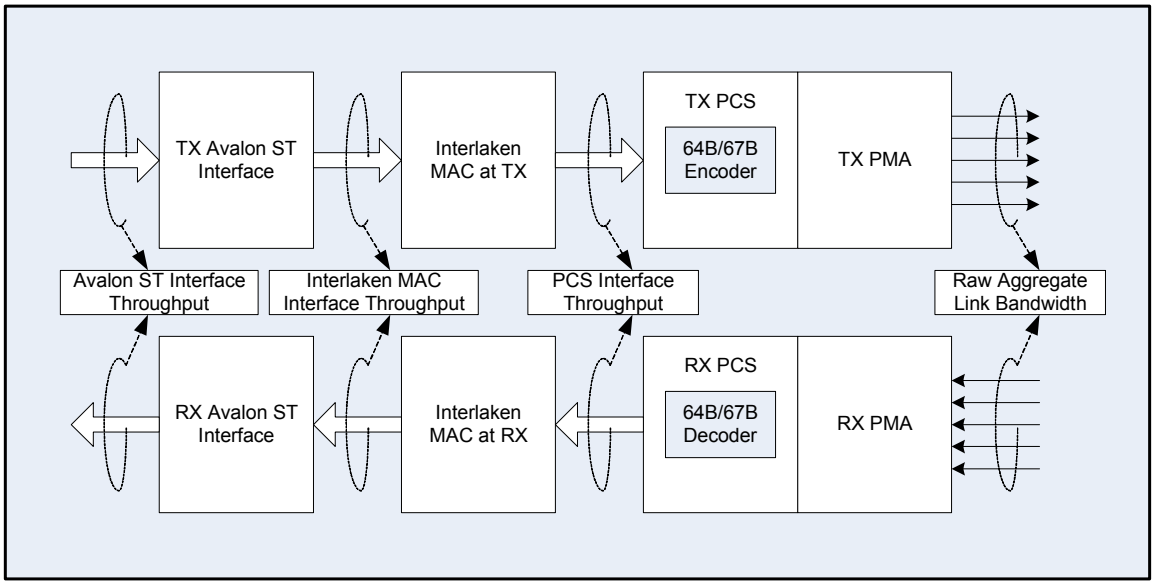

Figure 1. Interlaken Block Diagram and Interface Bandwidth Nodes

The Interlaken protocol requires 64B/67B encoding before data transmission. The encoding scheme creates an overhead of roughly 4.5% of bandwidth. The following equation calculates the maximum bandwidth for data at the PCS TX and PCS RX interfaces:

$$\text{PCS interface throughput} = \text{raw aggregate link bandwidth} \times (64/67)$$

An Interlaken MetaFrame carries the payload. The MetaFrame requires per-lane insertion of synchronization, scrambler state, skip, and diagnostic words, in addition to the payload data carried on each lane. Because the synchronization, diagnostic, and scrambler state words are sent so infrequently, they consume a minimal amount of interface bandwidth. The amount of overhead depends on the MetaFrameLength. The following equation calculates the maximum payload throughput in a MetaFrame:

$$\text{MetaFrame payload throughput} = \text{PCS interface throughput} \times (\text{MetaFrameLength} - 4)/\text{MetaFrameLength}$$

For a hypothetical 2K word data payload, the worst-case overhead, in the case of a single skip word, is (Sync word+ScramblerState+Skip+Diagnostics)/MetaFrameLength which is

$$4 \text{ words}/ 2048 = 0.20\%$$

The Interlaken MAC typically operates by sending a burst of data followed by a control word. The Interlaken protocol specification defines the BurstMax, BurstMin, and BurstShort parameters, which control how the IP core segments a packet of arbitrary size into bursts. Control words add overhead by occupying link bandwidth. The following equation calculates the maximum usable data bandwidth at the Interlaken MAC interface:

$$\text{Interlaken MAC interface throughput} = \text{MetaFrame payload throughput} \times \text{MetaFrame data burst throughput}$$

where the following equation calculates the value of the **MetaFrame data burst throughput**:

$$\text{MetaFrameDataBurst throughput} = \sum \text{DataBurstEachFrame} / \sum (\text{DataBurst} + \text{ControlWord}) \text{ EachFrame}$$

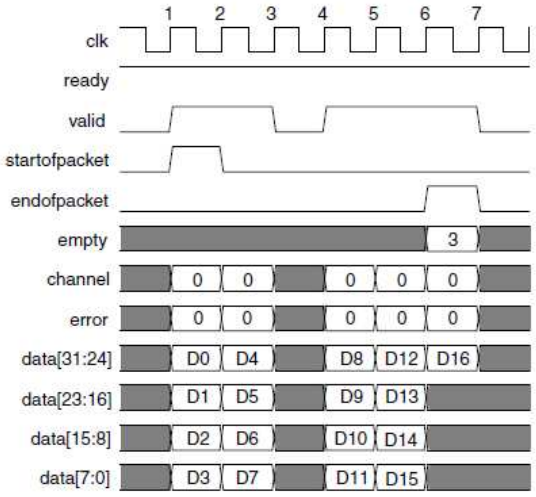

Figure 2. Avalon ST Interface for Packet Transfer

The Altera Interlaken MegaCore function that is available with the MegaCore IP library includes a user data interface that conforms to the Avalon Streaming (Avalon-ST) interface specification. This interface specification enables you to easily integrate your design with other system

components on the device using the Quartus II software. Because the Avalon-ST interface can be channelized, end-of-packet (EOP) data cycles may occur back-to-back on several channels with a very small amount of remaining packet data on each channel.. An empty signal indicates the number of bytes not carrying valid data in an EOP cycle. These invalid data bytes are not available to transmit the start bytes of the next packet.

In theory, as packet size increases, packet throughput should also increase. However, because control words are inserted per burst, the MAC throughput decreases for packets with payload size a multiple of BurstMax plus one byte, compared to the MAC throughput for packets with payload size a multiple of BurstMax. The throughput at the Avalon-ST interface is affected by packet size and interface bus width. This throughput decreases for packets with payload size a multiple of the interface bus width plus one byte, compared to the Avalon-ST interface throughput for packets with payload size a multiple of the interface bus width. For example, a 64-byte packet experiences much higher throughput than a 65-byte packet, and a 128-byte packet experiences much higher throughput than a 129-byte packet. If the packet size is a multiple of bus width, the maximally efficient throughput is obtained. If the packet size is a multiple of bus width plus one byte, the minimally efficient throughput is observed.

The reference design provides throughput measurement at the Avalon-ST interface and the benchmarking results of transmitting various packet sizes to the Altera Interlaken IP core.

3 Deliverables Included With the Reference Design

All reference design deliverables are compressed in one file. After you download the compressed file and extract the reference design files, you should observe the file directory structure shown in Figure 3.

The reference design includes the following components:

- Reference Design HDL files including the Interlaken IP core, Nios II processor, Avalon Memory-Mapped (Avalon-MM) interface, JTAG UART, packet generator, packet checker and statistics/performance monitor
- Nios II processor programming files
    - Quartus II Archives Files (.qar) for the demonstration boards and configuration files including
    - SRAM Object File (.sof)
    - SignalTap II Files (.stp)
    - Design Constraint File (.sdc)
    - Quartus Setting File (.qsf)
    - Quartus Project File (.qpf)
- Interlaken MegaCore Function Parameter Selection Worksheet for throughput and user clock frequency calculation.

- Application note that describes how to use the Interlaken MegaCore Function Parameter Selection Worksheet.
- This application note.

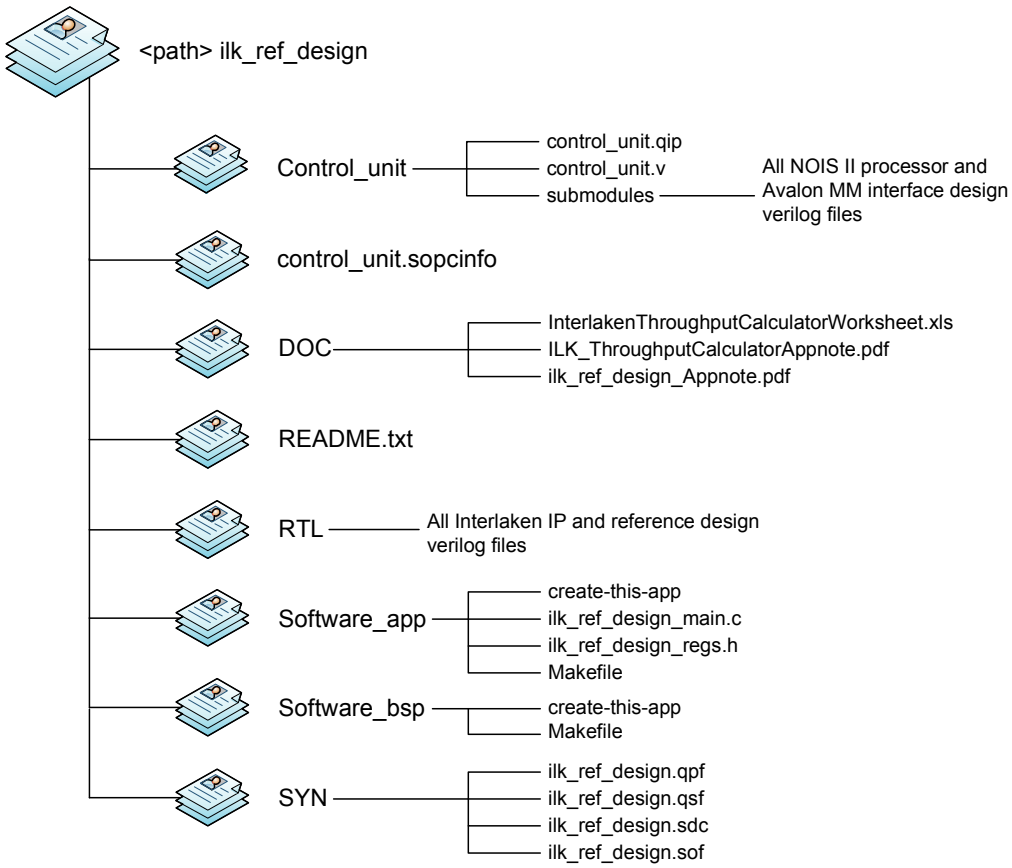

Figure 3. Reference Design Directory and File Structure

4 Reference Design System Block Functional Description

4.1 System Architecture

This reference design includes the components shown in

Figure 4:

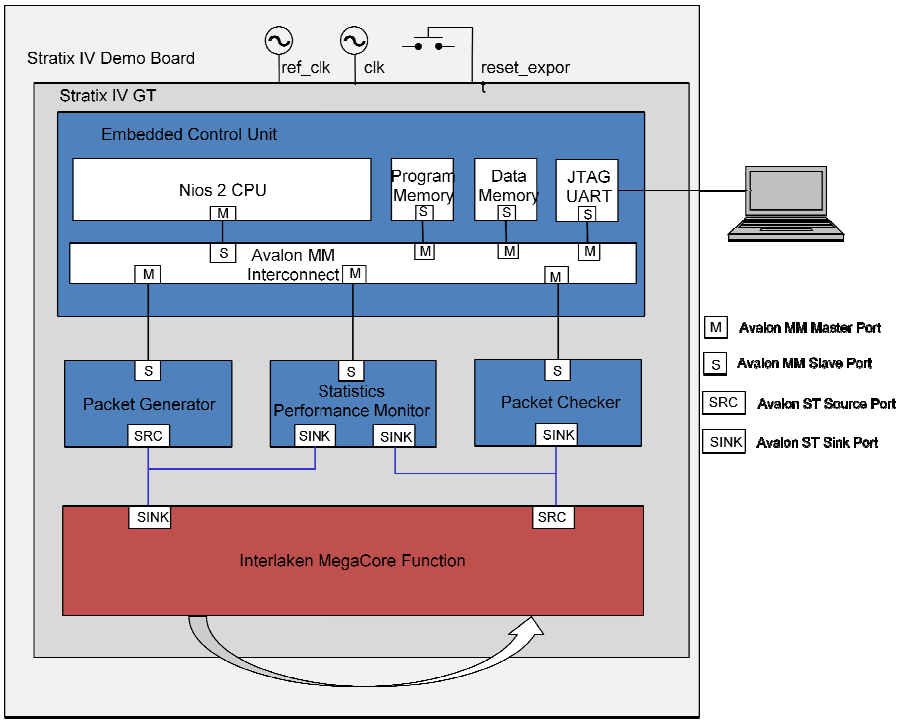

Figure 4. Reference Design System Architecture Overview

- A JTAG UART interface
- Embedded Nios II processor
- Packet Generator
- Packet Checker
- Statistics and Performance Monitor (SPM) module
- Interlaken IP Core configured in a 20 lane, 6.375 Gbps/lane configuration in internal loopback mode

4.2 JTAG UART Interface and Embedded Nios II Processor

A JTAG UART interface establishes and maintains a communication channel between a PC and the reference design. A set of user commands controls reference design execution of various actions. These commands and their definitions are listed in

Table 3 on page 15.

You can use the JTAG UART interface to interactively issue commands and dynamically reconfigure the reference design. The Nios II processor receives user defined commands through this interface and translates them to Avalon-MM interface control register read and write

operations for various system blocks including the packet generator, the packet checker, and the statistics and performance monitor.

4.3 Packet Generator

The Packet Generator module is responsible for generating Avalon-ST packets based on characteristics you provide by programming the addressable registers. This module has the following two main interfaces:

- Control: The control interface supports access to internal registers through an Avalon-MM interface. The same clock drives the Packet Generator control interface and the Control Unit module.
- Data: The data interface is an Avalon-ST interface which drives traffic to the Interlaken IP core's transmit port. This interface – the traffic data path – is clocked with the fast system clock.

In this reference design, a Nios II embedded processor, which is in the Control Unit module, drives the Packet Generator control interface. The Nios II processor provides support for read and write operations to the address map of the Packet Generator. You use these read and write operations to program the Packet Generator.

The Packet Generator generates a very simple traffic pattern. Each word of a packet is 64 bytes wide, except the last word, which is optionally shorter than 64 bytes. In each system clock cycle, the Packet Generator can transfer a full word to the transmit port of the Interlaken IP core. The first word of a packet carries a packet_id followed by a payload of incrementing data. For example, Figure 5 shows the first three packets for a test in which the packet size is 129(decimal) bytes and the interpacket gap is one cycle.

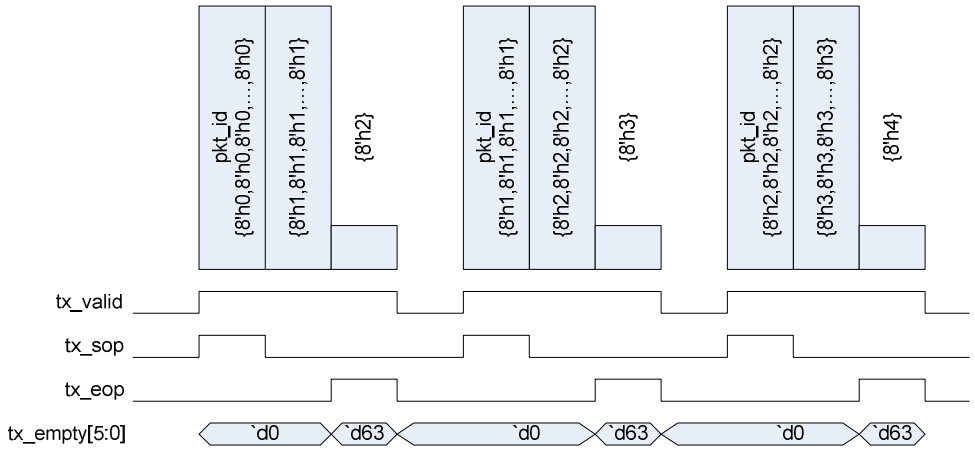

Figure 5. Reference Design System Architecture Overview

In this example, if the Packets–to-Generate register value is set to 1000, then one-thousand packets are generated. If the Send Continuous Traffic bit, bit[16] of the Control Register, is set, then the Packet Generator generates packets until that bit is cleared by the user.

The Packet Generator registers are described in Packet Generator Register Map on page 22.

4.4 Packet Checker

The Packet Checker module is responsible for checking the data integrity of packets received. This module has the following two main interfaces:

- Control: The control interface supports access to internal registers through an Avalon-MM interface. The same clock drives the Packet Checker control interface and the Control Unit module.
- Data: The data interface is an Avalon-ST interface which receives traffic from the Interlaken IP core's receive port. This interface – the traffic data path – is clocked with the fast system clock.

The Packet Checker checks for a sequential packet identifier at the begining of each packet and for an incrementing data payload in the rest of the packet. It can also detect missing start-of-packets (SOPs) and missing end-of-packets (MOPs) interface violations. All of the detected errors are captured in registers which can be addressed by the embedded Nios 2 processor via the Avalon-MM Control interface. The Packet Checker registers are described in Packet Checker Register Map on page 22.

4.5 Statistics and Performance Monitor

The Statistics and Performance Monitor (SPM) module is responsible for keeping track of performance and status statistics. This module maintains statistics about the current status reported by the Interlaken IP core and maintains traffic counters to aid in bandwidth calculations. The module snoops the TX and RX interfaces which connect the Packet Generator and the Packet Checker to the Interlaken IP core. The module keeps track of packets transmitted and received as well as bytes transmitted and received. The module also registers miscellaneous status signals from the Interlaken IP core. These registers as well as the counters are addressable and can be read from the module. Statistics and Performance Monitor Register Map on page 23 describes the address map of this module as well as the registers and their meaning.

This module has a control interface to support accesses to internal registers. The control interface is an Avalon-MM interface that is clocked with the same clock that drives the Control Unit module.

In this reference design, a Nios II embedded processor, which is in the Control Unit module, drives the SPM module control interface. The Nios II processor provides support for read and write operations to the address map of the SPM module.

The SPM module has two sets of registers. One set of register maintains the current count of statistics in the current sample window. The second set of registers shadows the first set. When the sample window expires, the current counter values are transferred from the first set of registers to the shadow counters. At this time, if the interrupt enable is programmed, the SPM module sets an interrupt bit, signaling to an external processor that a new set of statistics is ready to be read.

You must program a value, in clock cycles, for the sample window. During the sample window, current counters keep track of the statistics, and at expiration of the sample window, as described previously, the values are transferred to the shadow counters.

A C program that demonstrates the use of the SPM module is included with the reference design. The program demonstrates how to use the sample window registers, how to service the interrupt, and how to use the counter values to calculate the bandwidth sustained by the Interlaken IP core.

4.6 Interlaken IP Core

For information about the Altera Interlaken IP core, refer to the Altera Interlaken MegaCore Function User Guide.

5 Design Walkthrough with Hardware and Software Requirements

5.1 Hardware Requirements

The reference design has the following hardware requirements:

- 40G/100G Stratix IV GT SMA demonstration board USB cable
- Windows or Linux based PC

5.2 Software Requirements

The reference design has the following software requirements:

- Quartus II software version 11.1
- Altera USB-Blaster or ByteBlaster driver
- Qsys system integration tool (provided with the Quartus II software)
- Nios II Embedded Design Suite (EDS)

5.3 Setting up the 40G/100G Stratix IV GT SMA Demonstration Board

5.3.1 Demonstration Board Block Diagram

Figure 6 shows a block diagram of the 40G/100G Stratix IV GT S Demonstration board.

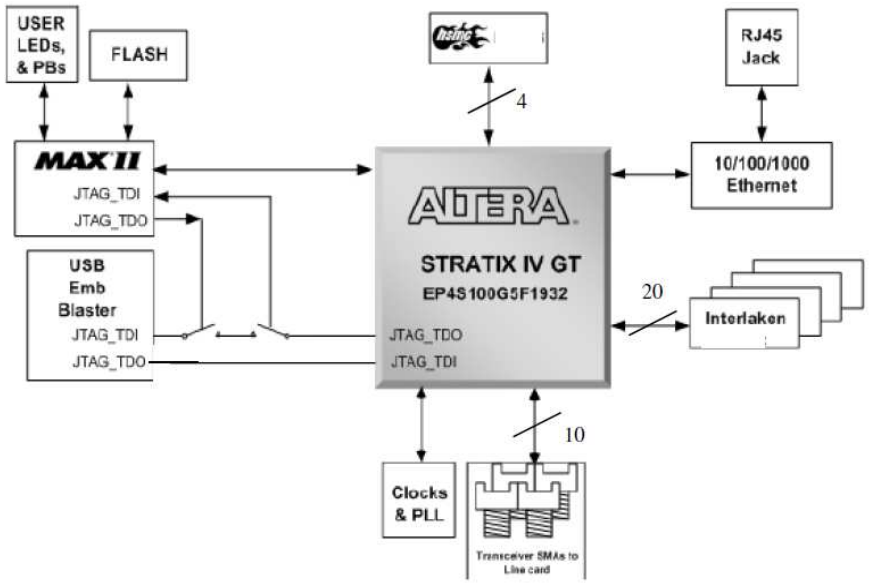

Figure 6. 40G/100G Stratix IV GT SMA Demonstration Board Block Diagram

5.3.2 Demonstration Board Overview

Figure 7 provides an overview of the 40G/100G STRATIX IV GT SMA Demonstration board features.

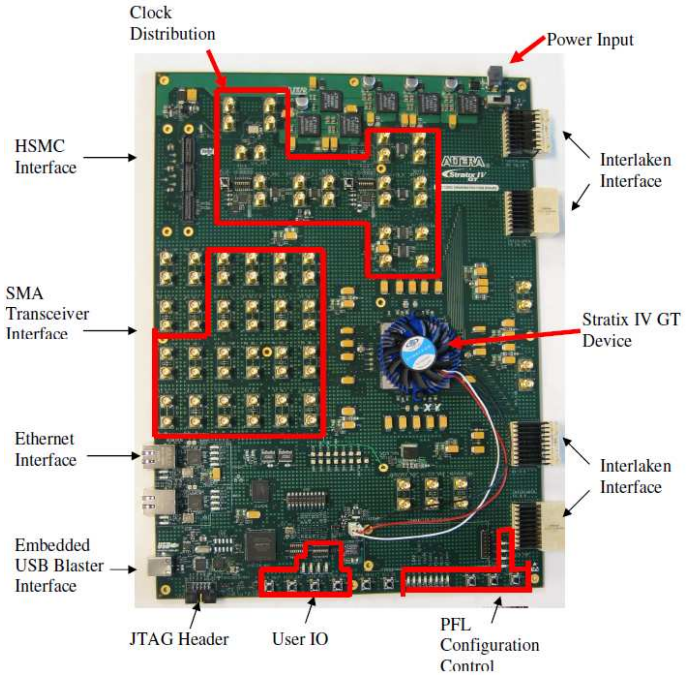

Figure 7. 40G/100G Stratix IV GT SMA Demonstration Board Features

5.3.3 Setting up PLL on Demonstration Board

There are two PLLs on demonstration board, PLLA and PLLB.

The PLLA is located near the Altera logo. The PLLA and all the clocks associated with it are distributed to the transceivers that go to the Interlaken interface. The DIP switch labeled PLLA enables the PLLA and controls the output frequency. The Table 1 below shows the DIP switch configuration for PLLA which are in the up position or set to "1". This reference design uses PLLA which CE is enabled and the frequency is 318.75MHz. PLLA provides clocks for all transceivers and Interlaken MAC.

Table 1. DIP Switch Configuration for PLLA

Fin (MHz)	CE	/	/	OD2	OD1	OD0	PR1	PR0	Fout (MHz)
25.5	1	1	1	0	0	1	1	0	318.75

NOTE: When you unzip the package, you will note that the .sdc file contains timing constraints targettting a 312.50 MHz clock and that the High Speed IO blocks, transceivers, were parameterized to target a line rate of 6250 Mbps with a reference clock of 312.50 MHz. However, the actual board uses the PLLA configured for 318.75 MHz. This is acceptable, since the positive slack achieved during compilation supports the faster 318 MHz frequency.

The PLLB is located near the HSMC interface. The PLLB and all the clocks associated with it are distributed to the transceivers that go to the SMAs and HSMC connector. The DIP switch labeled PLLB enables the PLLB and controls the output frequency. The Table 2 below shows the DIP switch configuration which are in the up position or set to "1". This reference design uses PLLB which CE is enabled and the frequency is 46.875MHz. PLLB provides clocks for Nios II processor.

Table 2. DIP Switch Configuration for PLLB

Fin (MHz)	CE	/	/	OD2	OD1	OD0	PR1	PR0	Fout (MHz)
25.5	1	1	1	1	1	1	0	1	46.875

5.4 Setting up the System Console

5.4.1 Install device drivers and Processor start up code

There is a directory called software_bsp in the reference design package. The file `create-this-bsp` in this directory is used to install necessary device drivers and cpu start up code to run software application. It only needs to be done once after the reference design package is decompressed.

```
ilk_ref_design> cd software_bsp
ilk_ref_design/software_bsp> ./create-this-bsp
```

The `create-this-bsp` will call a file `control_unit.sopcinfo` which is located one directory up in parallel to software_bsp. The following files and three directories HAL, drivers, obj, are generated after create-this-bsp is executed:

```
settings.bsp
public.mk
system.h
mem_init.mk
linker.x
linker.h
memory.gdb
alt_sys_init.c
Makefile
summary.html
libhal_bsp.a
```

The appendix A shows the sample log file generated after `create-this-bsp` is successfully executed.

5.4.2 Generate user defined commands and software environment for application

There is a directory called software_app in the reference design package. The file `create-this-app` in this directory is used to compile ilk_ref_design_main.c and ilk_ref_design_regs.h in same directory and generate Makefile. It only needs to be done once after the reference design package is de-compressed.

```
ilk_ref_design > cd software_app
ilk_ref_design/software_app> ./create-this-app
```

After create-this-app executed, obj directory and following three files are created in same directory:

```
ilk_ref_design_test.map
ilk_ref_design_test.elf
ilk_ref_design_test.objdump
```

The appendix B shows the sample log file generated after `create-this-app` is successfully executed.

5.4.3 Embedded Nios II processor command for this reference design

Table 3 Embedded Nios II Control Unit Commands

Command	Description
H	Displays the list of commands.
load <address> <value>	Writes <value> into a register which address is <address> (in hex)
get <address>	Reads value from a register which address is <address> (in hex)
config_pkt <size> <ipg> <packets>	Specifies the packet size <size> in bytes, inter-packet gap <ipg> in cycles, and number of packets <packets> to be generated. Enter all of the arguments in decimal format. This command is equivalent to execute the following three commands sequentially: load <packet size register address> < size>

	load <inter-packet gap register address> <ipg> load <number of packets register address> < packets>
start	Triggers the packet generator to start generating packets. Use this command to send a fixed number of packets.
streaming_pkt	Triggers the packet generator to stream packets continuously.
rst_pkt_gen_ctrl	Triggers the packet generator to reset the packet generator state machine. This command is strongly recommended between two bursts of traffic (triggered either by start or streaming_pkt) and when changing packet configuration.
ilk_status	Displays the Interlaken IP core and link status. The following information is displayed: • Word lock status –20-bit bus showing the lane word lock status (the 64-bit word boundary was found) for each lane. Bit[n] indicates the status for lane n. The bit value of 1 indicates word lock is achieved on the relevant lane. • Sync lock status –20-bit bus showing lane sync lock status (the MetaFrame boundary was found) for each lane. Bit[n] indicates the status for lane n. The bit value of 1 indicates sync lock is achieved on the relevant lane. • Interlaken IP core fully locked – Message indicates whether the Interlaken IP core is fully locked. A fully locked message is shown only if all lanes are word- and sync-locked and all lane alignment is achieved. • Interlaken IP core CRC24 error status. • Interlaken IP core per lane CRC32 error status. The CRC32 error count has a residual value of 1. So the power up value is 1.
ilk_perf_walk_thru <size lower bound> <size upper bound> <size interval> <gap>	Automatically walk through and measure the performance of packets from <size lower bound> to <size upper bound> with <size interval> increment and <gap> cycles of separation between packets. Enter all of the arguments in decimal format.
clk	Sets the Interlaken IP core MAC clock period (in cyles/nanosecond). The clock information is needed for bandwidth calculation.
set_sample_window	Sets the bandwidth sample window (in cycles, in hex). This command is needed for bandwidth calculation. The default value is 0x01FFFFFF cycles.
ilk_perf	Continuously monitors and displays Interlaken performance for the current sample window. You specify the sample window duration with the set_sample_window command described in this table. After each sample-window length of time ends, the CPU is notified and collects statistics data from the packet statistics block. This process repeats itself until you enter the z command described in this table. This command displays the following information: • Sample window – The number of cycles (length of sample window) during which the performance data was collected. • TX packet count – The number of packets observed during the

	sample window at the transmit interface between the Interlaken IP core and the user application (in this reference design, the user application is the Packet Generator module). • TX byte count – The number of bytes observed during the sample window at the transmit interface between the Interlaken IP core and the user application (in this reference design, the user application is the Packet Generator module). • TX bandwidth – Interlaken transmit bandwidth during the sample window. • RX packet count – The number of packets observed during the sample window at the receive interface between Interlaken IP core and the user interface (in this reference design, the user interface is the Packet Checker module). • RX byte count – The number of bytes observed during the sample window at the receive interface between Interlaken IP core and the user interface (in this reference design, the user interface is the Packet Checker module). • RX bandwidth – Interlaken receive bandwidth during the sample window.
z	Use the z command following the ilk_perf command when you wish to stop collecting bandwidth information.

5.5 Using the Reference Design

The following lists steps to use this reference design for throughput measurement.

Step 1. Load the design (.SOF) file into FPGA.

The `ilk_ref_design.sof` in ilk_ref_design/syn directory can be loaded into FPGA through Quartus II SignalTap II Logic Analyzer or using following command in `software_app` directory

```
ilk_ref_design/software_app> nios2-configure sof ../syn/ilk_ref_design.sof
```

The appendix C shows the sample log file generated after `nios2-configure` command is successfully executed.

Step 2. Load software driver into Nios II processor instruction memory.

Use the following command to download device driver into Nios II processor.

```
ilk_ref_design/software_app> make download-elf
```

The appendix D shows the sample log file generated after `make download-elf` command successfully executed.

Step 3. Invoke Nios2-terminal

Use `nios2-terminal` command to invoke Nios2 terminal. "`ILK Reference Design>`" prompt will show after `nios2-terminal` command successfully executed.

```
ilk_ref_design/software_app> nios2-terminal
```

After Nios2-terminal prompt appears and you are ready to execute commands listed in **Table 3.**

```
ILK Reference Design > type commands listed in
Table 3 here
```

Step 4. Check Interlaken Status

It is always a good practice to check if the Interlaken is in good status after bringing up nios2-terminal. `ilk_status` is the command to check the status after "`ILK Reference Design>`" prompt**:**

```
ILK Reference Design > ilk_status
```

The following information will be shown on computer screen after `ilk_status` executed**:**

```
Command Read: ilk_status
****************************
Getting interlaken status....
****************************
Interlaken lane word lock : 0xfffff
Interlaken lane sync lock : 0xfffff
INTERLAKEN FULLY LOCKED!!
Interlaken CRC24 error count :                0
Interlaken lane 0 CRC32 error count :                1
Interlaken lane 1 CRC32 error count :                1
Interlaken lane 2 CRC32 error count :                1
Interlaken lane 3 CRC32 error count :                1
Interlaken lane 4 CRC32 error count :                1
Interlaken lane 5 CRC32 error count :                1
Interlaken lane 6 CRC32 error count :                1
Interlaken lane 7 CRC32 error count :                1
Interlaken lane 8 CRC32 error count :                1
Interlaken lane 9 CRC32 error count :                1
Interlaken lane 10 CRC32 error count :                1
Interlaken lane 11 CRC32 error count :                1
Interlaken lane 12 CRC32 error count :                1
Interlaken lane 13 CRC32 error count :                1
Interlaken lane 14 CRC32 error count :                1
Interlaken lane 15 CRC32 error count :                1
Interlaken lane 16 CRC32 error count :                1
Interlaken lane 17 CRC32 error count :                1
Interlaken lane 18 CRC32 error count :                1
Interlaken lane 19 CRC32 error count :                1
```

The CRC32 error count has a power up initial value of 1.

After Interlaken status is checked and they are good, user can run commands to measure design throughput. Below lists three examples of using this reference:

5.5.1 Example 1: Generating fixed size packets to measure throughput

Step 1, 2, 3, 4 are required before moving to below step 5, 6, 7.

Step 5. Use config_pkt command to configure packet generator.

An example of using `config_pkt` command is shown below which send five 173-byte packets with 1 cycle gap (IPG) between each packet.

```
ILK Reference Design > config_pkt 173 1 5
```

The 3rd argument 5 (number of packet) is optional if streaming packet is used for throughput measurement. The 3rd argument need to be specified if user wants to generate a fixed number of packets. The 1 cycle IPG between the packets will be maintained as long as Interlaken core doesn't back pressure. When packet generator receives back pressure from Interlaken MAC, it will delay sending packets which cause the gap bigger than specified.

Step 6. Trigger packet generator to send packets

There are two ways to let packet generator sending packets. One is to use `start` command, the other is to use `streaming_pkt` command. Below is an example of using `streaming_pkt` command.

```
ILK Reference Design > streaming_pkt
```

To stop streaming packets, use `rst_pkt_gen_ctrl` as following example.

```
ILK Reference Design > rst_pkt_gen_ctrl
```

It is always recommended to reset packet generator using `rst_pkt_gen_ctrl` before sending packet again regardless of using `start` or `streaming_pkt`. If user wants to send a fixed number of packets which the number is listed as 3rd argument of `config_pkt` in step 1, use start command as shown in below example:

```
ILK Reference Design > start
```

Step 7. Collect throughput information.

Before collecting throughput information, MAC clock period need to be specified as in nano-second. An error message will show if clock period is not specified.

```
ILK Reference Design > clk 3.2
ILK Reference Design > ilk_perf
```

The command to trigger Nios II processor to collect statistics and calculate throughput is `ilk_perf`. The processor collects the data and does calculation in a fixed interval of time which is called sample window. The default value of this sample window is 0x1ffffff. It can be changed by using

`set_sample_window` command. The recommended value of this sample window is between 0x2ffffff and 0xffff.

Executing `ilk_perf` will output recursive display of throughput information as shown below. To exit from `ilk_perf`, enter command "z".

```
..........
..........

Sample 235 :

Sample Window ......................            4095 cycles
Operation clock ....................        3.200000 ns

TX Packets .........................            2704 packets
        TX Byte Count ..............          173056 bytes
        TX Bandwidth ...............      105.650795 Gbps
        Expected packet size .......              64 bytes
        Observed packet size .......       64.000000 bytes

RX Packets .........................            2704 packets
        RX Byte Count ..............          173056 bytes
        RX Bandwidth ...............      105.650795 Gbps
        Expected packet size .......              64 bytes
        Observed packet size .......       64.000000 bytes


Sample 236 :

Sample Window ......................            4095 cycles
Operation clock ....................        3.200000 ns

TX Packets .........................            2736 packets
        TX Byte Count ..............          175104 bytes
        TX Bandwidth ...............      106.901100 Gbps
        Expected packet size .......              64 bytes
        Observed packet size .......       64.000000 bytes

RX Packets .........................            2720 packets
        RX Byte Count ..............          174080 bytes
        RX Bandwidth ...............      106.275948 Gbps
        Expected packet size .......              64 bytes
        Observed packet size .......       64.000000 bytes

..........
..........
```

If Interlaken reference design suddenly loses sync, lock or has data CRC error while `ilk_perf` is still executing, the above bandwidth information will still show but with additional error statements.

5.5.2 Example 2: Automatically measure throughput over a range of packet size

Step 1, 2, 3, 4 are required before executing below ilk_perf_walk_thru command. The clock period need to be specified before executing ilk_perf_walk_thru command.

```
ILK Reference Design > clk 3.2
```

```
ILK Reference Design > ilk_perf_walk_thru 64 67 1 0
```

The above `ilk_perf_walk_thru` example is to collect throughputs for packet sizes ranging from 64 to 67 (64, 65, 66, 67). It will stream 64-byte packets back to back first without gap and collect throughput information for 10 iterations. Then it will do same thing to 65-byte packets, then 66-byte packets and 67-byte packets. The ilk_perf_walk_thru command contains `streaming_pkt` and `rst_pkt_gen_ctrl` and `ilk_perf` functions. The contained `ilk_perf` outputs 4 sets of throughput data for packet size from 64 to 67, 10 throughput sampling data for each packet size. Appendix E is attached at the end of this App note to show the output of this example.

5.5.3 Example 3. Using command file to collect throughput data into a file.

This reference design allows user to write all commands into a script file and execute it without bringing up Nios2-terminal. Output throughput data for various packet sizes can also be collected into a file.

Open a command script file named `perf_walk_through.cmds` using `vi` in `software_app` directory as shown below:

```
Design_restored/software_app> vi perf_walk_through.cmds
```

Write below two lines of command into `perf_walk_through.cmds` and save the file.

```
clk 3.2
ilk_perf_walk_thru 64 67 1 0
```

Run the `perf_walk_through.cmds` in regular shell under software_app directory without invoking nios2-terminal as shown below:

```
ilk_ref_design/software_app>nios2-terminal <perf_walk_through.cmds> outputfile
```

The appendix E shows the output file format.

6 Measured Bandwidth Throughput

To obtain more results of the throughput performance based on this variation of the Interlaken core used in this reference design, please make the request using the following email address.

interlaken_support@altera.com

7 System Register Map System Register Map

The system console commands like “load” and “get” allow user directly access registers inside Packet Generator, Packet Checker and Statics and Performance monitor. The address argument for “load” and “get” commands are system base address for each functional block plus the offset for each register.

7.1 System Base Address

The base addresses for the functional blocks are:

- Packet Generator Base Address: 0x61400
- Packet Checker Base Address: 0x61000
- Statistics and Performance Monitor Base Address: 0x00000

7.2 Packet Generator Register Map

The following table describes the Packet Generator module internal registers address offset.

Address	Description	Bit Definitions
0x00	Packet Generator control register	[31:17] – Unused; [16] – Send continuous traffic; [15:1] – Unused; [0] – Start packet generation;
0x04	Packet size	[31:16] – Unused; [15:0] – Packet size specified as number of bytes;
0x08	Packets-to-generate	[31:24] – Unused; [23:0] – Number of packets to generate;
0x0C	Interpacket gap	[31:8] – Unused; [7:0] – Number of idle cycles between packets;

7.3 Packet Checker Register Map

The following table describes the Packet Checker module internal registers address offset.

Address	Description	Bit Definitions
0x00	Packet Checker control register	[31:17] – Unused [16] – Soft clear [15:3] – Unused [4] – RX_ready [3:1] – Unused [0] – Enable checker
0x07	Missing SOP	[31:0] – Number of SOP missing from RX received packets
0x08	Missing EOP	[31:0] – Number of EOP missing from RX received packets
0x09	Packet word error	[31:0] – Number of packet word errors from RX received packets
0x0A	Packet ID error	[31:0] – Number of Packet ID errors from RX received packets

7.4 Statistics and Performance Monitor Register Map

The following table describes the Statistics and Performance Monitor module internal registers address offset.

Address	Description	Bit Definitions
0x00	Control register	[31:1] – Unused [0] – Soft clear
0x04	Sample window	[31:0] – Width of the Sample Window
0x08	Interrupt	[31:1] – Unused; [0] Interrupt Bit
0x0C	Interrupt enable	[31:1] – Unused; [0] Interrupt Enable
0x10	Time counter	[31:0] – Current cycle counter, used internally to determine when the sample window has expired
0x14	Misc status	[31:25] – Unused; [24] – tx_status_hungry [23:21] – Unused; [20] – tx_status_overflow [19:17] – Unused; [16] – tx_status_underflow [15:13] – Unused; [12] – rx_status_fully_locked [11:9] – Unused; [8] – rx_status_overflow [7:5] – Unused; [4] – rx_status_all_word_locked [3:1] – Unused; [0] – rx_status_all_sync_locked
0x18	RX Status Lock Time	[31:0] – locked time since acquiring fully locked condition
0x1C	RX Status Error Count	[31:16] – Unused [15:0] – Number of CRC24 errors reported by Interlaken IP core
0x20	RX Status Per Lane Word Lock	[31:20] – Unused [N] – Lane N Word Locked, for N in [19:0]
0x24	RX Status Per Lane Sync Lock	[31:20] – Unused [N] – Lane N Sync Locked, for N in [19:0]
0x28	TX Avalon Error Count (1)	[31:0] – Number of times that tx_chX_dataout_error has been observed
0x2C	RX Avalon Error Count (1)	[31:0] – Number of times that rx_chX_dataout_error has been observed
0x30	TX Packet Count (1)	[31:0] – Number of packets transmitted in one sample window
0x34	TX Packet Byte Count (1)	[31:0] – Number of bytes transmitted in one sample window
0x38	RX Packet Count (1)	[31:0] – Number of packets received in one sample window
0x3C	RX Packet Byte Count (1)	[31:0] – Number of bytes received in one sample window
0x40	Lane 0 CRC32 Errors	[31:8] – Unused; [7:0] Lane 0 CRC32 Error Count
0x44	Lane 1 CRC32 Errors	[31:8] – Unused; [7:0] Lane 1 CRC32 Error Count
0x48	Lane 2 CRC32 Errors	[31:8] – Unused;[7:0] Lane 2 CRC32 Error Count
0x4C	Lane 3 CRC32 Errors	[31:8] – Unused; [7:0] Lane 3 CRC32 Error Count
0x50	Lane 4 CRC32 Errors	[31:8] – Unused; [7:0] Lane 4 CRC32 Error Count

0x54	Lane 5 CRC32 Errors	[31:8] – Unused; [7:0] Lane 5 CRC32 Error Count
0x58	Lane 6 CRC32 Errors	[31:8] – Unused; [7:0] Lane 6 CRC32 Error Count
0x5C	Lane 7 CRC32 Errors	[31:8] – Unused; [7:0] Lane 7 CRC32 Error Count
0x60	Lane 8 CRC32 Errors	[31:8] – Unused; [7:0] Lane 8 CRC32 Error Count
0x64	Lane 9 CRC32 Errors	[31:8] – Unused; [7:0] Lane 9 CRC32 Error Count
0x68	Lane 10 CRC32 Errors	[31:8] – Unused; [7:0] Lane 10 CRC32 Error Count
0x6C	Lane 11 CRC32 Errors	[31:8] – Unused; [7:0] Lane 11 CRC32 Error Count
0x70	Lane 12 CRC32 Errors	[31:8] – Unused; [7:0] Lane 12 CRC32 Error Count
0x74	Lane 13 CRC32 Errors	[31:8] – Unused; [7:0] Lane 13 CRC32 Error Count
0x78	Lane 14 CRC32 Errors	[31:8] – Unused; [7:0] Lane 14 CRC32 Error Count
0x7C	Lane 15 CRC32 Errors	[31:8] – Unused; [7:0] Lane 15 CRC32 Error Count
0x80	Lane 16 CRC32 Errors	[31:8] – Unused; [7:0] Lane 16 CRC32 Error Count
0x84	Lane 17 CRC32 Errors	[31:8] – Unused; [7:0] Lane 17 CRC32 Error Count
0x88	Lane 18 CRC32 Errors	[31:8] – Unused; [7:0] Lane 18 CRC32 Error Count
0x8C	Lane 19 CRC32 Errors	[31:8] – Unused; [7:0] Lane 19 CRC32 Error Count

(1) These counters are maintained per sample window instance.

8 Appendix

Appendix A

After executing create-this-bsp, the screen will show log similar to the following and should say BSP build complete.

```
create-this-bsp: Running "nios2-bsp hal . /data/pochu/ipapps/ilkn_rd_s4gx/design  "
nios2-bsp: Using /tools/acds/11.1acs/0.13/linux64/nios2eds/sdk2/bin/bsp-set-defaults.tcl to
set system-dependent settings.
nios2-bsp: Updating existing BSP because ./settings.bsp exists.
nios2-bsp: Using SOPC design file
/data/pochu/ipapps/ilkn_rd_s4gx/design/control_unit.sopcinfo found in
/data/pochu/ipapps/ilkn_rd_s4gx/design
nios2-bsp: Running "nios2-bsp-update-settings --settings ./settings.bsp --bsp-dir .  --sopc
/data/pochu/ipapps/ilkn_rd_s4gx/design/control_unit.sopcinfo --script
/tools/acds/11.1acs/0.13/linux64/nios2eds/sdk2/bin/bsp-set-defaults.tcl  "
INFO: Updating BSP settings...
INFO: nios2-bsp-update-settings --settings ./settings.bsp --bsp-dir .  --sopc
/data/pochu/ipapps/ilkn_rd_s4gx/design/control_unit.sopcinfo --script
/tools/acds/11.1acs/0.13/linux64/nios2eds/sdk2/bin/bsp-set-defaults.tcl
INFO: Initializing BSP components...

INFO: Finished initializing BSP components. Total time taken = 3 seconds
INFO: Searching for BSP components with category: os_software_element
INFO: Initializing SOPC project local software IP
INFO: [Info] <b>/data/pochu/ipapps/ilkn_rd_s4gx/design/*</b> matched 135 files in 0.01
seconds
INFO: [Info] <b>/data/pochu/ipapps/ilkn_rd_s4gx/design/*/*_sw.tcl</b> matched 0 files in 0.01
seconds
INFO: [Info] <b>/data/pochu/ipapps/ilkn_rd_s4gx/design/ip/**/*_sw.tcl</b> matched 0 files in
0.00 seconds
INFO: [Info] <b>/data/pochu/ipapps/ilkn_rd_s4gx/ip/**/*_sw.tcl</b> matched 0 files in 0.00
seconds
INFO: Finished initializing SOPC project local software IP. Total time taken = 2 seconds
INFO: Searching for BSP components with category: driver_element
INFO: Searching for BSP components with category: software_package_element
INFO: No memory regions found in BSP Settings File. Default memory regions generated.
INFO: Loading drivers from ensemble report.
INFO: Finished loading drivers from ensemble report.
INFO: Tcl message: "STDIO character device is jtag_uart_0"
INFO: Tcl message: "No system timer device"
INFO: Tcl message: "Default instruction linker sections mapped to program_0"
INFO: Tcl message: "Default data linker sections mapped to mem_0"
INFO: Tcl message: "No bootloader located at the reset address."
INFO: Tcl message: "Application ELF allowed to contain code at the reset address."
INFO: Tcl message: "The alt_load() facility is enabled."
INFO: Tcl message: "The .rodata section is copied into RAM by alt_load()."
```

```
INFO: Tcl message: "The .rwdata section is copied into RAM by alt_load()."
INFO: Default memory regions will not be persisted in BSP Settings File.
INFO: Generated file "/data/pochu/project/ilk_ref_design_syn_2/software_bsp/settings.bsp"
INFO: BSP settings updated
INFO: Generating BSP files...
INFO: Generating BSP files in "/data/pochu/project/ilk_ref_design_syn_2/software_bsp"
INFO: Default memory regions will not be persisted in BSP Settings File.
INFO: Generated file "/data/pochu/project/ilk_ref_design_syn_2/software_bsp/settings.bsp"
INFO: Generated file "/data/pochu/project/ilk_ref_design_syn_2/software_bsp/summary.html"
INFO: Generated file "/data/pochu/project/ilk_ref_design_syn_2/software_bsp/mem_init.mk"
INFO: Generated file "/data/pochu/project/ilk_ref_design_syn_2/software_bsp/public.mk"
INFO: Generated file "/data/pochu/project/ilk_ref_design_syn_2/software_bsp/Makefile"
INFO: Finished generating BSP files. Total time taken = 2 seconds
INFO: BSP files generated in "/data/pochu/project/ilk_ref_design_syn_2/software_bsp"
create-this-bsp: Running make
[BSP build complete]
```

Appendix B.

After executing create-this-bsp, the screen will show log similar to the following and should say build complete.

```
create-this-app: Running "nios2-app-generate-makefile --bsp-dir ../software_bsp --elf-name
ilk_ref_design_test.elf --set OBJDUMP_INCLUDE_SOURCE 1 --src-rdir . --set APP_INCLUDE_DIRS ."
INFO: nios2-app-generate-makefile --bsp-dir ../software_bsp --elf-name
ilk_ref_design_test.elf --set OBJDUMP_INCLUDE_SOURCE 1 --src-rdir . --set APP_INCLUDE_DIRS .
INFO: Generating application makefile
SEVERE: ./Makefile (Permission denied)
SEVERE: nios2-app-generate-makefile failed.
nios2-app-generate-makefile failed
create-this-app: Running "make"
Info: Building ../software_bsp/
make --no-print-directory -C ../software_bsp/
[BSP build complete]
Info: Creating ilk_ref_design_test.objdump
nios2-elf-objdump --disassemble --syms --all-header --source ilk_ref_design_test.elf
>ilk_ref_design_test.objdump
[ilk_ref_design_test build complete]

To download and run the application:
    1. Make sure the board is connected to the system.
    2. Run 'nios2-configure-sof <SOF_FILE_PATH>' to configure the FPGA with the hardware
design.
    3. If you have a stdio device, run 'nios2-terminal' in a different shell.
    4. Run 'make download-elf' from the application directory.

To debug the application:
    Import the project into Nios II IDE.  Refer to Nios II IDE Documentation for more
information.
```

Appendix C.

After executing nios2-config-sof, the screen will show log similar to the following.

```
Info: *******************************************************************
Info: Running Quartus II 64-Bit Programmer
Info: Command: quartus_pgm --no_banner --mode=jtag -o p;./../ilk_ref_design.sof
Info (213046): Using programming cable "USB-Blaster on SJ-POCHU-HP [USB-0]"
Info (213011): Using programming file ./../ilk_ref_design.sof with checksum 0x15CC2D8C for
device EP4S100G5F45C2ES1@1
Info (209060): Started Programmer operation at Wed Jan 25 11:37:24 2012
Info (209016): Configuring device index 1
Info (209017): Device 1 contains JTAG ID code 0x024030DD
Info (209007): Configuration succeeded -- 1 device(s) configured
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Wed Jan 25 11:38:01 2012
Info: Quartus II 64-Bit Programmer was successful. 0 errors, 0 warnings
    Info: Peak virtual memory: 1064 megabytes
    Info: Processing ended: Wed Jan 25 11:38:01 2012
    Info: Elapsed time: 00:00:45
    Info: Total CPU time (on all processors): 00:00:08
```

Appendix D.

After executing make download-elf, the screen will show log similar to the following.

```
Info: Building ../software_bsp/
make --no-print-directory -C ../software_bsp/
[BSP build complete]
Info: Downloading ilk_ref_design_test.elf
Using cable "USB-Blaster on SJ-POCHU-HP [USB-0]", device 1, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 93KB in 1.6s (58.1KB/s)
Verified OK
Starting processor at address 0x000401B4
```

Appendix E.

Output of ilk_perf_walk_thru command.

```
##########################################
INTERLAKEN PERFORMANCE WALK THROUGH

from 64 bytes to 67 bytes with 1 byte(s) interval
and 0 cycles of gap between packets.
##########################################

##########################################
Measuring Performance for 64 byte packets...
##########################################

Sample 0 :

Sample Window ......................            4095 cycles
Operation clock ....................        3.200000 ns

TX Packets .........................            2704 packets
       TX Byte Count ..............          173056 bytes
       TX Bandwidth ...............      105.650795 Gbps
        Expected packet size .......             64 bytes
        Observed packet size .......       64.000000 bytes

RX Packets .........................            2711 packets
       RX Byte Count ..............          173504 bytes
       RX Bandwidth ...............      105.924301 Gbps
        Expected packet size .......             64 bytes
        Observed packet size .......       64.000000 bytes

....................
.................... (similar information for sample 1 to sample 9, omitted)


Sample 9 :

Sample Window ......................            4095 cycles
Operation clock ....................        3.200000 ns

TX Packets .........................            2710 packets
       TX Byte Count ..............          173440 bytes
       TX Bandwidth ...............      105.885223 Gbps
        Expected packet size .......             64 bytes
        Observed packet size .......       64.000000 bytes

RX Packets .........................            2707 packets
       RX Byte Count ..............          173248 bytes
       RX Bandwidth ...............      105.768013 Gbps
```

```
        Expected packet size .......              64 bytes
        Observed packet size .......       64.000000 bytes



##############################################
Measuring Performance for 65 byte packets...
##############################################

Sample 0 :

Sample Window ......................            4095 cycles
Operation clock ...................         3.200000 ns

TX Packets .........................            2048 packets
       TX Byte Count ..............          133056 bytes
       TX Bandwidth ...............       81.230766 Gbps
        Expected packet size .......              65 bytes
        Observed packet size .......       64.968750 bytes

RX Packets .........................             2047 packets
        RX Byte Count ..............          133056 bytes
        RX Bandwidth ...............       81.230766 Gbps
         Expected packet size .......              65 bytes
         Observed packet size .......       65.000488 bytes


.....................
..... (similar information for sample 1 to sample 9, omitted)


Sample 9 :

Sample Window ......................             4095 cycles
Operation clock ....................          3.200000 ns

TX Packets .........................             2048 packets
        TX Byte Count ..............          133056 bytes
        TX Bandwidth ...............       81.230766 Gbps
         Expected packet size .......              65 bytes
         Observed packet size .......       64.968750 bytes

RX Packets .........................             2048 packets
        RX Byte Count ..............          133119 bytes
        RX Bandwidth ...............       81.269234 Gbps
         Expected packet size .......              65 bytes
         Observed packet size .......       64.999512 bytes


##############################################
Measuring Performance for 66 byte packets...
```

```
###########################################

Sample 0 :

Sample Window ......................            4095 cycles
Operation clock ....................         3.200000 ns

TX Packets .........................            2048 packets
        TX Byte Count ..............          135104 bytes
        TX Bandwidth ...............       82.481071 Gbps
         Expected packet size .......              66 bytes
         Observed packet size .......       65.968750 bytes

RX Packets .........................            2048 packets
        RX Byte Count ..............          135166 bytes
        RX Bandwidth ...............       82.518929 Gbps
         Expected packet size .......              66 bytes
         Observed packet size .......       65.999023 bytes


.....................
..... (similar information for sample 1 to sample 9, omitted)


Sample 9 :

Sample Window ......................            4095 cycles
Operation clock ....................         3.200000 ns

TX Packets .........................            2048 packets
        TX Byte Count ..............          135104 bytes
        TX Bandwidth ...............       82.481071 Gbps
         Expected packet size .......              66 bytes
         Observed packet size .......       65.968750 bytes

RX Packets .........................            2048 packets
        RX Byte Count ..............          135166 bytes
        RX Bandwidth ...............       82.518929 Gbps
         Expected packet size .......              66 bytes
         Observed packet size .......       65.999023 bytes



###########################################
Measuring Performance for 67 byte packets...
###########################################

Sample 0 :

Sample Window ......................            4095 cycles
Operation clock ....................         3.200000 ns

TX Packets .........................            2048 packets
        TX Byte Count ..............          137152 bytes
        TX Bandwidth ...............       83.731377 Gbps
         Expected packet size .......              67 bytes
```

```
        Observed packet size .......       66.968750 bytes

RX Packets .........................           2047 packets
        RX Byte Count ..............         137152 bytes
        RX Bandwidth ...............       83.731377 Gbps
        Expected packet size .......             67 bytes
        Observed packet size .......       67.001465 bytes


.....
..... (similar information for sample 1 to sample 9, omitted)


Sample 9 :

Sample Window ......................           4095 cycles
Operation clock ....................        3.200000 ns

TX Packets .........................           2048 packets
        TX Byte Count ..............         137152 bytes
        TX Bandwidth ...............       83.731377 Gbps
        Expected packet size .......             67 bytes
        Observed packet size .......       66.968750 bytes

RX Packets .........................           2048 packets
        RX Byte Count ..............         137213 bytes
        RX Bandwidth ...............       83.768623 Gbps
        Expected packet size .......             67 bytes
        Observed packet size .......       66.998535 bytes
```