Gigabit Ethernet PCS IP Core for LatticeECP2M

User's Guide

August 2007
ipug69_01.0

Introduction

The 1000BASE-X physical layer, also referred to as the Gigabit Ethernet (GbE) physical layer, consists of three major blocks, the Physical Coding Sublayer (PCS), the Physical Medium Attachment sublayer (PMA), and the Physical Medium Dependent sublayer (PMD). The LatticeECP2M™ embedded SERDES/PCS performs the PMA function, and portions of the PMD and PCS functions, including link serialization/deserialization, code-group alignment, clock tolerance compensation buffering, and 8b10b encoding/decoding. However, the embedded SERDES/PCS does not provide all necessary functions for implementing a complete GbE physical layer solution. That's where the GbE PCS IP core comes in. The IP core provides the additional functions required to fully implement the PCS functions of the GbE physical layer. These additional functions include a transmit state machine, a receive state machine, and auto-negotiation.

This document describes the IP core's operation and provides instructions for generating the core through ispLEVER® IPexpress™, including instantiating, synthesizing, and simulating the core.

Features

- Implements the transmit, receive, and auto-negotiation functions of the IEEE 802.3z specification
- 8-bit GMII Interface operating at 125 MHz
- 8-bit Code-Group Interface operating at 125 MHz
- Parallel signal interface for control and status management

Functional Description

The GbE PCS IP core converts GMII data frames into 8-bit code groups in both transmit and receive directions; and performs auto negotiation with a link partner as described in the IEEE 802.3z specification. The core's block diagram is shown in Figure 1. The following paragraphs detail the operation the IP core's main functional blocks. An example of how this IP core may be used in implementing a gigabit ethernet physical layer is shown in Figure 2.

Figure 1. GbE PCS IP Core Block Diagram

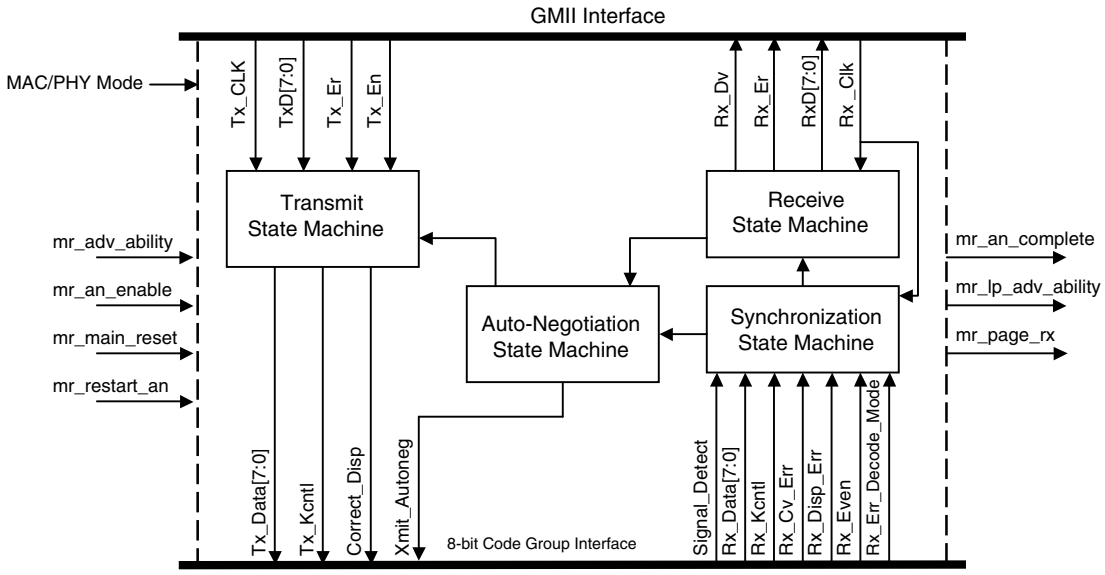
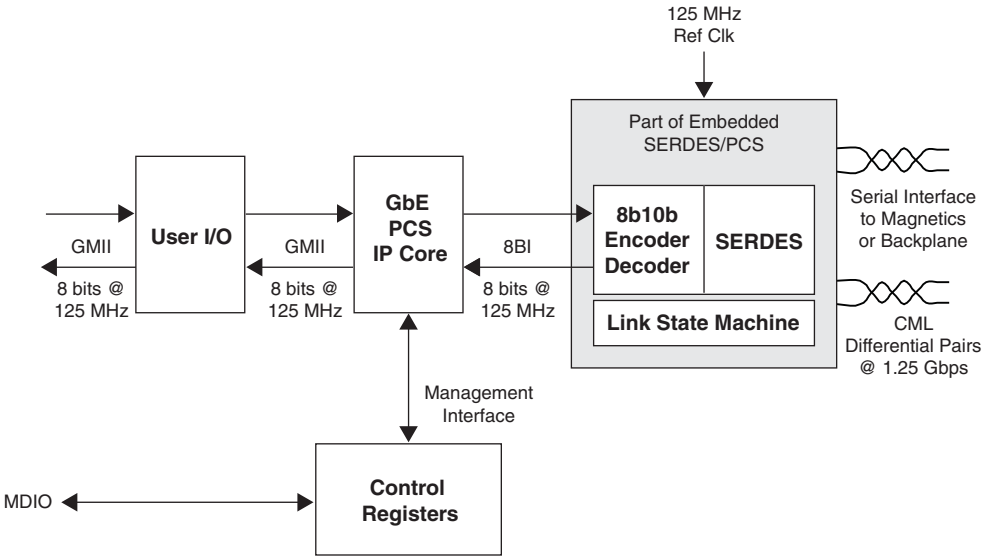


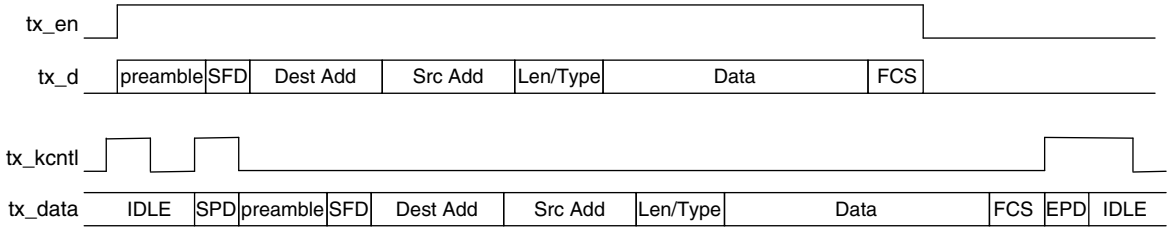
Figure 2. Typical GbE Physical Layer Implementation



Transmit State Machine

The transmit state machine implements the transmit functions described in clause 36 of the IEEE 802.3 specification. The state machine's main purpose is to convert GMII data frames into code groups. A typical timing diagram for this circuit block is shown below. Note that the state machine in this IP core does not fully implement the conversion to 10-bit code groups as specified in the 802.3 specification. Instead, partial conversion to 8-bit code groups is performed. A separate encoder (located in the LatticeECP2M embedded SERDES/PCS block) completes the full conversion to 10-bit code groups.

Figure 3. Typical Transmit Timing Diagram



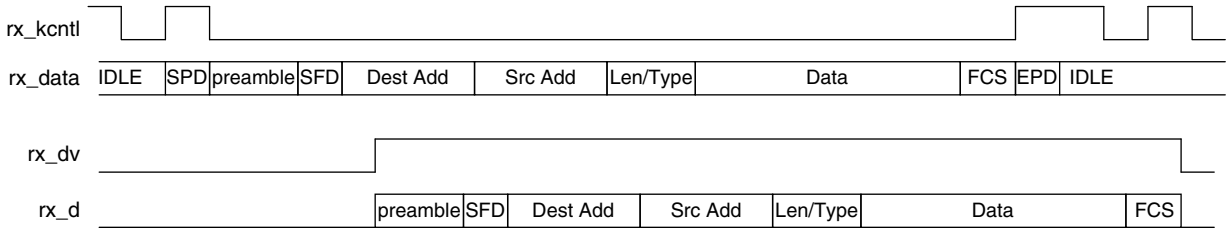
Synchronization State Machine

The synchronization state machine implements the alignment functions described in clause 36 of the IEEE 802.3 specification. The state machine's main purpose is to determine whether incoming code groups are properly aligned. Once alignment is attained, proper code groups are passed to the receive state machine. If alignment is lost for an extended period, an auto negotiation restart is triggered.

Receive State Machine

The receive state machine implements the receive functions described in clause 36 of the IEEE 802.3 specification. The state machine's main purpose is to convert code groups into GMII data frames. A typical timing diagram for this circuit block is shown below. Note that the state machine in this IP core does not fully implement the conversion from 10-bit code groups as specified in the 802.3 specification. Instead, partial conversion from 8-bit code groups is performed. A separate decoder (located in the LatticeECP2M embedded SERDES/PCS block) performs 10-bit to 8-bit conversions.

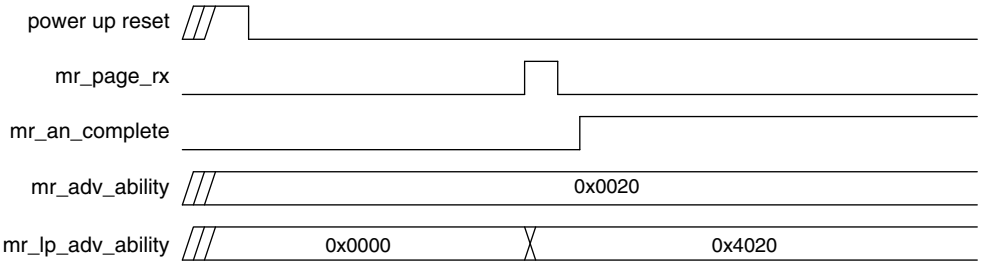
Figure 4. Typical Receive Timing Diagram



Auto-Negotiation State Machine

The auto-negotiation state machine implements the link configuration functions described in clause 37 of IEEE 802.3 specification, including checking link readiness, determining duplex mode, and negotiating flow control. A typical timing diagram is shown below.

Figure 5. Typical Auto-Negotiation Timing Diagram



Signal Descriptions

Table 1. GbE PCS IP Core Input and Output Signals

Signal Name	I/O	Description
Clock Signals		
tx_clk_125	In	**Transmit Clock** – 125 MHz clock source for transmit state machine. Incoming GMII transmit data is sampled on rising edge of this clock. Outgoing 8-bit code group transmit data is launched on the rising edge of this clock.
rx_clk_125	In	**Receive Clock** – 125 MHz clock source for receive state machine and the synchronization state machine. Incoming signals are sampled on the rising edge of the clock. Outgoing signals are launched on the rising edge of this clock.
GMII Signals		
tx_d[7:0]	In	**Transmit Data** – Incoming GMII data.
tx_en	In	**Transmit Enable** – Active high signal, asserts when incoming data is valid.
tx_er	In	**Transmit Error** – Active high signal, used to denote transmission errors and carrier extension on incoming GMII data port.
rx_d[7:0]	Out	**Receive Data** – Outgoing GMII data.
rx_dv	Out	**Receive Data Valid** – Active high signal, asserts when outgoing data is valid.
rx_er	Out	**Receive Error** – Active high signal, used to denote transmission errors and carrier extension on outgoing GMII data port.
8-Bit Code Group Signals		
tx_data[7:0]	Out	**8b Transmit Data** – 8-bit code group data after passing through transmit state machine.
tx_kcntl	Out	**8b Transmit K Control** – Denotes whether current code group is data or control. 1=control 0=data

Table 1. GbE PCS IP Core Input and Output Signals (Continued)

Signal Name	I/O	Description
correct_disp	Out	**Corrects Disparity** – Asserted during inter-packet gaps to ensure that negative disparity IDLE ordered-sets are transmitted by the LatticeECP2M embedded SERDES /PCS. 1=correct disparity, 0=normal
xmit_autoneg	Out	**Auto-negotiation Transmitting** – This signal asserts when the IP core's auto negotiation state machine is active. The signal is used by the LatticeECP2M embedded SERDES/PCS to occasionally insert idle ordered sets into its receive path (eight ordered sets every 2048 clocks). This facilitates proper operation of the embedded clock tolerance compensation circuit. 1=autoneg is active, 0=autoneg is not active
rx_data[7:0]	In	**8b Receive Data** – 8-bit code group data presented to the receive state machine.
rx_kcntl	In	**8b Receive K Control** – Denotes whether current code group is data or control. 1=control 0=data
rx_err_decode_mode	In	**Receive Error Control Mode** – The embedded SERDES block of the LatticeECP2M FPGAs has two modes of interpreting errors, decoded and normal. In decoded mode, the three signals (rx_even, rx_cv_err, rx_disp_err) are used to decode 1-of-8 error conditions. In decoded mode, the IP core responds to the following errors: 100 = Coding Violation Error 111 = Disparity Error All other error codes are ignored by the IP core. In normal mode, the three error signals (rx_even, rx_cv_err, rx_disp_err) behave normally. The rx_err_decode_mode signal should be set high for decode mode, and low for normal mode.
rx_even	In	**Rx Even** – This signal is only used when error decoding mode is active. Otherwise, the signal should be tied low.
rx_cv_err	In	**Rx Coding Violation Error** – In normal mode, an active high signal denoting a coding violation error in the receive data path. In decode mode, used to decode 1 of 8 error conditions.
rx_disp_err	In	**Rx Disparity Error** – In normal mode, an active high signal denoting a disparity error in the receive data path. In decode mode, used to decode 1 of 8 error conditions.
signal_detect	In	**Signal Detect** – Denotes status of GbE PCS RX physical link. 1=signal is good; 0=loss of receive signal
Management Signals		
mr_adv_ability[15:0]	In	**Advertised Ability** – Configuration status transmitted by PCS during auto negotiation process.
mr_an_enable	In	**Auto Negotiation Enable** – Active high signal that enables auto negotiation state machine to function.
mr_main_reset	In	**Main Reset** – Active high signal that forces all PCS state machines to reset.
mr_restart_an	In	**Auto Negotiation Restart** – Active high signal that forces auto negotiation process to restart.
mr_an_complete	Out	**Auto Negotiation Complete** – Active high signal that indicates that the auto negotiation process is completed.
mr_lp_adv_ability[15:0]	Out	**Link Partner Advertised Ability** – Configuration status received from partner PCS entity during the auto negotiating process. The bit definitions are the same as described above for the mr_adv_ability port.
mr_page_rx	Out	**Auto Negotiation Page Received** – Active high signal that asserts while the auto negotiation state machine is in the Complete_Acknowledge state.
Miscellaneous Signals		
rst_n	In	**Reset** – Active low global reset
debug_link_timer_short	In	**Debug Link Timer Mode** – Active high signal that forces the auto negotiation link timer to run much faster than normal. This mode is provided for debug purposes (e.g.,allowing simulations to run through the auto negotiation process much faster that the normal).

Core Generation

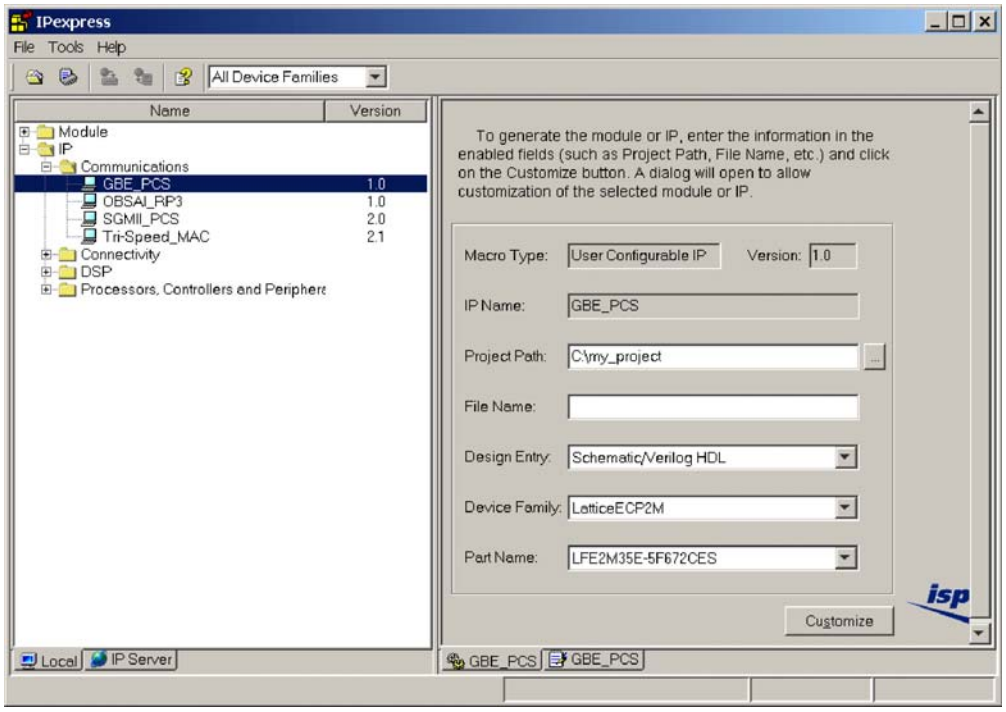
The GbE PCS IP core is available for download from the Lattice website at www.latticesemi.com. The IP files are automatically installed using ispUPDATE technology in any directory of your choosing.

The ispLEVER IPexpress GUI window for the GbE PCS IP core is shown in Figure 6. To generate a specific IP core configuration you must specify:

- **Project Path** – Path to directory where the generated IP files will be loaded.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **Design Entry type** – Verilog HDL.
- **Device Family** – Device family to which IP is to be targeted. Only families that support the particular core are listed.
- **Part Name** – Specific targeted part within the selected Device Family.

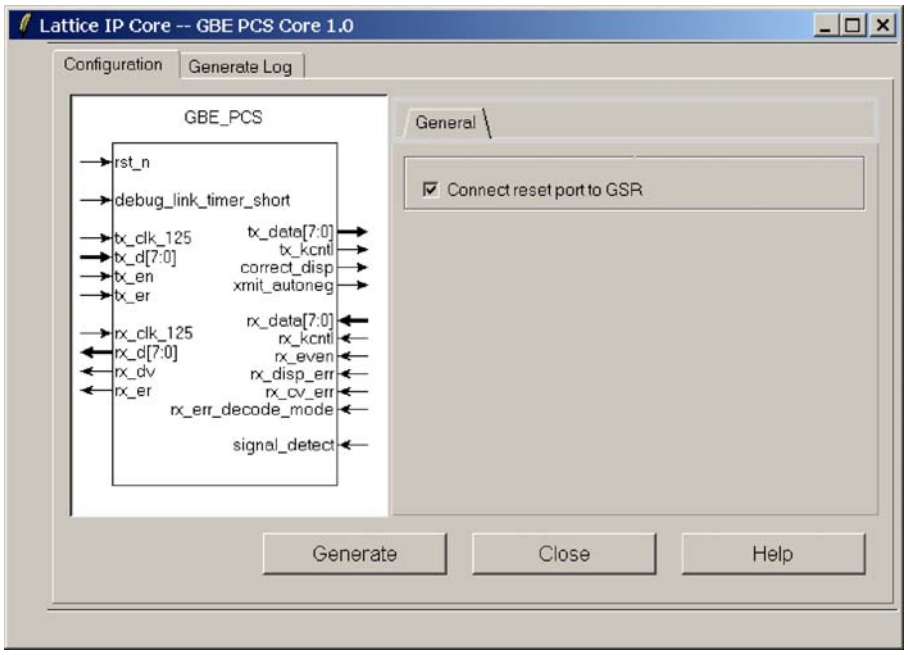
Note that if IPexpress is called from within an existing project, Project Path, Design Entry, Device Family and Part Name default to the specified project parameters. Please refer to the IPexpress on-line help for further information.

Figure 6. IPexpress Dialog Box



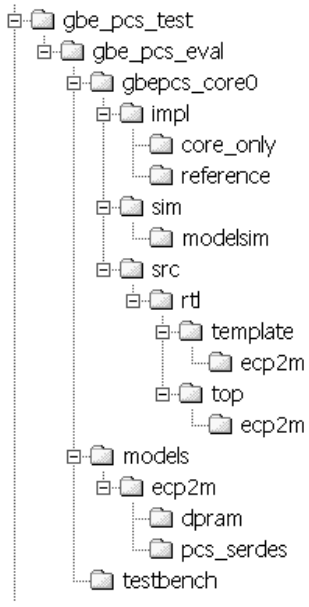
To create a custom configuration, click on the **Customize** button to display the GbE PCS IP core Configuration GUI, shown in Figure 7. From this window you may start the core generation process.

Figure 7. Configuration Dialog Box



When you click the **Generate** button, the IP core and supporting files are generated in the specified **Project Path** directory. The directory structure of the generated files is shown in Figure 8.

Figure 8. GbE PCS IP Core Generated Directory Structure



The following files are generated at the root of the "Project Path" directory (`gbe_pcs_test` in Figure 8):

- `<username>.lpc` – IP parameter file (you may modify this file if necessary)
- `<username>.ngo` – Synthesized and mapped IP core
- `<username>_bb.v` – Black box module wrapper for synthesis
- `<username>_inst.v` – Example of instantiation template to be included in customer's design
- `<username>_beh.v` – Behavioral simulation model for IP core configuration username

These are all of the files that you need to implement and verify the GbE PCS IP core in your own top-level design. The following additional files providing IP core generation status information are also generated in the "Project Path" directory:

- <username>_generate.log – ispLEVER synthesis and map log file
- <username>_gen.log – IPexpress IP generation log file

The \<gbe_pcs_eval> and subtending directories provide files supporting GbE PCS core evaluation. The \<gbe_pcs_eval> directory contains files/folders with content that is constant for all configurations of the GbE PCS. The \<username> subfolder (\gbepcs_core0 in this example) contains files/folders with content specific to the username configuration.

The \gbe_pcs_eval directory is created by IPexpress the first time the core is generated and updated each time the core is regenerated. A \<username> directory is created by IPexpress each time the core is generated and regenerated each time the core with the same file name is regenerated. A separate \<username> directory is generated for cores with different names, e.g. \<gbepcs_core1>, \<gbepcs_core2>, etc.

Instantiating the Core

The generated GbE PCS IP core package includes black-box (<username>_bb.v) and instance (<username>_inst.v) templates that can be used to instantiate the core in a top-level design. Two example RTL top-level source files are provided in \<project_dir>\gbe_pcs_eval\<username>\src\rtl\top\<technology>.

The top-level file top.v is a GbE Physical Layer Reference design (described in Appendix B). Additional files associated with the reference design are located in the directory \<project_dir>\gbe_pcs_eval\<username>\src\rtl\template\<technology>.

The top-level file top_gbe_pcs_core_only.v supports the ability to implement just the GbE PCS core by itself. This design is intended only to provide an indication of the device utilization associated with the GbE PCS IP core and should not be used as an actual implementation example.

Running Functional Simulation

The functional simulation model generated in the "Project Path" root directory (<username>_beh.v) may be instantiated in the your testbench for evaluation in the context of your application. Lattice does not provide a testbench for evaluating this IP core in isolation. However, a function simulation capability is provided in which <username>_beh.v is instantiated in an FPGA top level that implements a complete GbE Physical Layer as discussed previously and described in an appendix to this document. The top-level file supporting ModelSim® simulation is provided in \<project_dir>\gbe_pc_eval\<username>\sim\modelsim. This FPGA top is instantiated in an eval testbench provided in \<project_dir>\gbe_psc_eval\testbench.

You may run the eval simulation by doing the following:

1. Open ModelSim.

2. Under the **File** tab, select **Change Directory** and choose folder:

   <project_dir>\gbe_pcs_eval\<username>\sim\modelsim.

3. Under the **Tools** tab, select **TCL _ Execute Macro** and execute the ModelSim "**do**" script shown.

The simulation waveform results will be displayed in the ModelSim Wave window.

Synthesizing and Implementing the Core in a Top-Level Design

The GbE PCS IP core itself is synthesized and is provided in NGO format when the core is generated. You may synthesize the core in your own top-level design by instantiating the core in your top-level as described above in the "Instantiating the Core" section and then synthesizing the entire design with either Synplicity® or Precision® RTL.

As described previously, two example RTL top-level configurations supporting GbE PCS core top-level synthesis and implementation are provided in `\<project_dir>\gbe_pcs_eval\<username>\src\rtl\top\<technology>`.

The top-level file `top_gbe_pcs_core_only.v` provided in `\<project_dir>\gbe_pcs_eval\<username>\src\rtl\top` supports the ability to implement just the GBE_PCS core. This design is intended only to provide an accurate indication of the device utilization associated with the core itself and should not be used as an actual implementation example.

The top-level file `top.v` is a GbE Physical Layer Reference design `\<project_dir>\gbe_pcs_eval\<username>\src\rtl\top` supports the ability to instantiate, simulate, map, place and route the GBE_PCS IP core in a complete example design. A complete description of this design is given in an appendix to this document. Note that implementation of the reference evaluation configuration is specifically targeted to a LatticeECP2M LFE2M35E-6F672C device.

Push-button implementation of both top-level configurations is supported via the ispLEVER project files, `<username>_reference_eval.syn` and `<username>_core_only_eval.syn`. These files are located in `<project_dir>\ten_gbemac_test\ten_gbemac_eval\<username>\impl\<configuration>`.

To use these project files:

1. Select **Open Project** under the **File** tab in ispLEVER.

2. Browse to the **`\<project_dir\gbe_pcs_eval\<username>\impl`** directory and select either the **`\core_only`** or **`\reference`** directory in the **Open Project** dialog box.

3. Select and open either **`<username>_reference_eval.syn`** or **`username>_core_only_eval.syn`**. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.

4. Select the device top-level entry in the left-hand GUI window.

5. Implement the complete design via the standard ispLEVER GUI flow.

Hardware Evaluation

Lattice's IP hardware evaluation capability makes it possible to create versions of IP cores that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase on an IP license. The hardware evaluation capability may be enabled/disabled in the Properties menu of the Build Database setup in ispLEVER Project Navigator. It is enabled by default.

References

- ispLEVER Software User Manual
- ispLeverCORE™ IP Module Evaluation Tutorial available on the Lattice website at www.latticesemi.com

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
August 2007	01.0	Initial release.

Appendix A. LatticeECP2M Devices

Table 2. Performance and Resource Utilization[1]

Target Device	SLICEs	LUTs	Registers	I/Os[2]	f_{MAX} (MHz)
LFE2M35E-5F672CES	350	447	417	85	125

1. Performance and utilization characteristics are in Lattice's ispLEVER7.0 software with Synplify synthesis. When using this IP core in a different software version or a different device density or speed grade, performance may vary.
2. I/Os are for core-only top-level instantiation. The actual core does not require any primary I/O other than SERDES interface.

Ordering Part Number

The Ordering Part Number (OPN) for the GbE PCS core targeting LatticeECP2M devices is GBE-PCS-PM-U1.

You can use the IPexpress software tool to help generate new configurations of this IP core. IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and ispLEVER help system. For more information on the ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

Appendix B. GbE Physical Layer Reference Design

Introduction

This appendix describes the operation, simulation, and implementation of a GbE Physical Layer design, using Lattice's GbE PCS IP Core. The reference design utilizes two channels, one generates and monitors simplified ethernet frames, the other loops back all received ethernet frames. The two channels can be externally connected through the SERDES physical links, thereby establishing a demonstration of the interoperability between two GbE physical layers. Another application is connecting the reference design loopback channel to an external GbE traffic source (e.g. a Smartbits Test Generator), thereby demonstrating interoperability with the external traffic source.

Functional Description

The reference design block diagram is shown in Figure 9. The major blocks include two GbE PCS IP cores, the LatticeECP2M embedded SERDES/ PCS, a frame driver, a frame receiver, and control logic.

The Driver Channel

The driver channel is shown in the lower part of Figure 9, and is comprised of a frame driver, a frame monitor, and GbE PCS IP core, part of an embedded SERDES/PCS, and some control registers.

The transmit side of the channel begins at the frame driver, where a single 512-byte gigabit ethernet frame is repeatedly transmitted. The frame enters the transmit side of the GbE PCS IP core, where it is converted into 8-bit code groups. Next the frame enters the transmit portion of an embedded SERDES/PCS channel where 8b10b encoding and 10-bit-to-1-bit serialization occurs. The frame leaves the FPGA over the external 1.25Gbps SERDES physical link.

The receive side of the channel begins at the external SERDES input port. Clock recovery is performed, the data is deserialized, and an 802.3z synchronization state machine aligns the data stream to comma characters, and 10B8B decoding is performed. Next the frame enters the receive portion of the GbE PCS IP core, where 8-bit code groups are converted to GMII frames. Then the frame arrives at the frame monitor. If the payload data matches the frame driver data pattern, a "pass" signal is asserted. If the data pattern check fails, a "fail" signal is asserted.

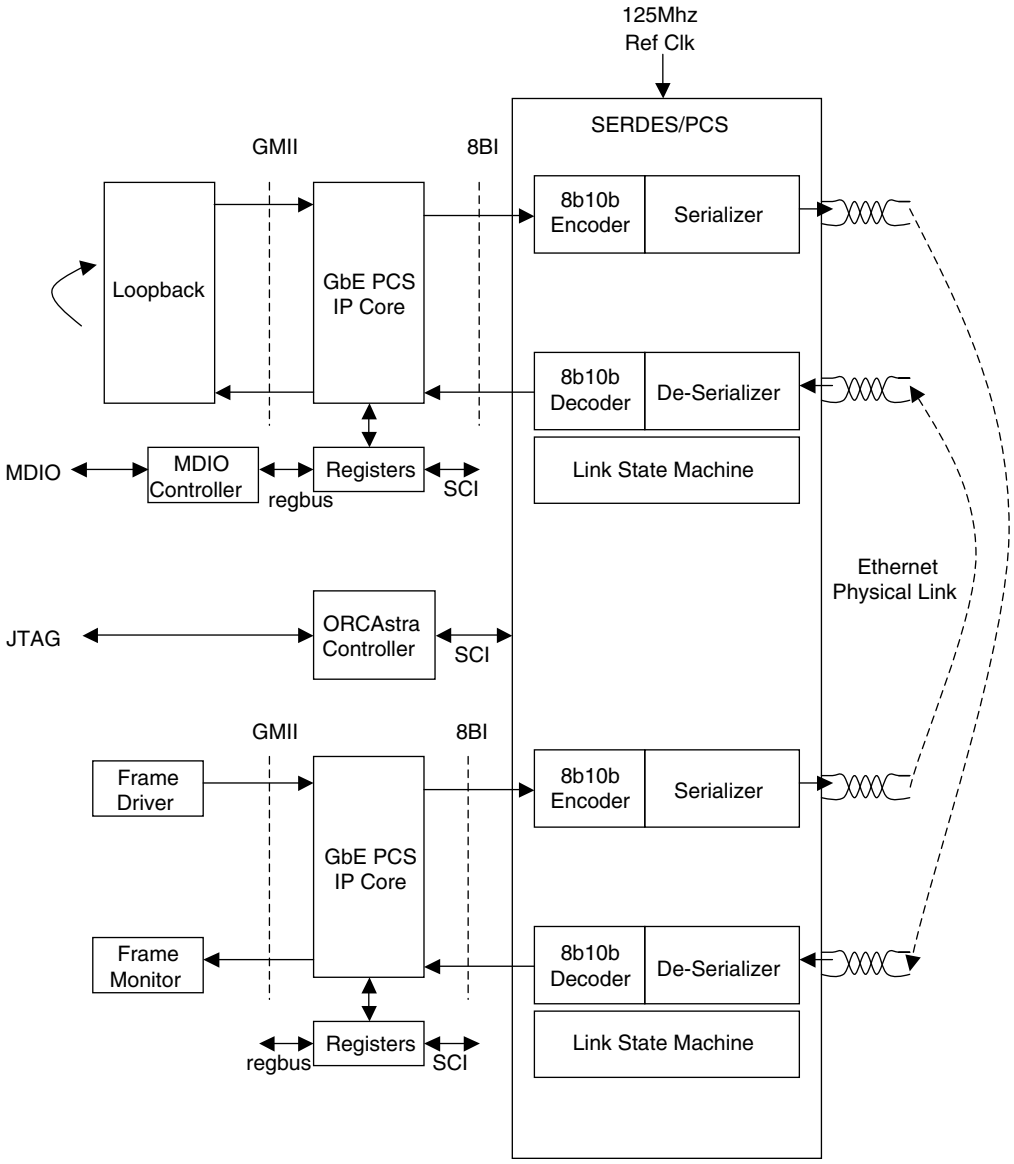
The Loopback Channel

The driver channel is shown in the upper part of Figure 9. It is similar to the driver channel except that it does not contain a frame driver or frame monitor. Instead this channel utilizes a parallel loopback block. All GMII frames from the receive path of the GbE PCS IP core are looped back to the transmit path of the GbE PCS IP core.

Clock Distribution

All timing in the reference design is derived from `ref_clk` a 125 MHz primary input to the FPGA. This clock is fed to the embedded SERDES/PCS where it is phase locked and used to time all outgoing SERDES channels, used to reference clock recovery of all incoming SERDES channels, and used to source timing for all of the FPGA soft logic, including the transmit and receive paths of the GbE PCS IP cores.

Figure 9. Block Diagram GbE Physical Layer Reference Design



IP Core Registers

A set of registers are implemented for each of the GbE PCS IP cores. These registers provide the management control functions discussed in IEEE 802.3 clauses 22 and 37. The registers are most commonly associated with managing auto-negotiation. The registers can be assessed by an external MDIO interface that conforms to the SMI protocol in IEEE 802.3 clause 22 or the registers can be accessed by an external JTAG interface that conforms to the Lattice ORCAstra protocol. The external `pin enable_smi` selects which register control method is used.

Table 3 shows the register set for one of the IP cores. Both IP cores have identical register sets. When using SMI, the two cores are distinguished by using different port IDs. When using ORCAstra, the two cores are distinguished by different memory address mappings.

Table 3. GbE PCS IP Core Management Registers

Address	Access Mode	Register Name	Register Bits															
			D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0x0	R/W	Control		—	—	mr_main_reset	—	—	mr_restart_an	—	—	—	—	—	—	—	—	—
0x1	R	Status	0	0	0	0	0	0	0	0	0	0	mr_an-complete	0	0	0	0	0
0x2	—	N/A	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
0x3	—	N/A	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
0x4	R/W	Auto-Negotiation Advertisement	mr_adv_ability[15:0]															
0x5	R	Auto-Negotiation Link Partner	mr_lp_adv_ability[15:0]															
0x6	R	Auto-Negotiation Expansion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	mr_page_rx	0

LatticeECP2M Embedded SERDES/PCS Registers

The embedded SERDES/PCS has a large register set for managing control and status. These registers are automatically configured for proper operation during FPGA configuration by means of an auto-configuration file called `pcs_serdes.txt`. If you want the registers to maintain their auto-configured states, you do not need to manually access the embedded SERDES/PCS registers. However, if you want to change register settings, or monitor the status registers, then you must manually access the registers. This reference design employs an external JTAG interface controlled by Lattice ORCAstra software for accessing the embedded SERDES/PCS registers. Table 4 shows the address mapping. Note that you may also access the GbE PCS IP core management registers through the ORCAstra interface. Please consult technical note TN1124, *LatticeECP2M SERDES/PCS Usage Guide,* for details on the embedded SERDES/PCS registers.

Table 4. ORCAstra Register Memory Map

ORCAstra Address	Register Description
0x000 - 0x03F	Embedded SERDES/PCS – Channel 0 Registers
0x400 - 0x07F	Embedded SERDES/PCS – Channel 1 Registers
0x800 - 0x0BF	Embedded SERDES/PCS – Channel 2 Registers
0xC00 - 0x0FF	Embedded SERDES/PCS – Channel 3 Registers
0x100 - 0x13F	Embedded SERDES/PCS – Quad Registers
0x800 / 0x810	IP Core 0 / IP Core 1 – `control reg [7:0]`
0x801 / 0x811	IP Core 0 / IP Core 1 – `control reg [15:8]`
0x802 / 0x812	IP Core 0 / IP Core 1 – `status reg [7:0]`
0x803 / 0x813	IP Core 0 / IP Core 1 – `status reg [15:8]`
0x808 / 0x818	IP Core 0 / IP Core 1 – `mr_adv_ability [7:0]`
0x809 / 0x819	IP Core 0 / IP Core 1 – `mr_adv_ability [15:8]`
0x80A / 0x81A	IP Core 0 / IP Core 1 – `mr_lp_adv_ability [7:0]`
0x80B / 0x81B	IP Core 0 / IP Core 1 – `mr_lp_adv_ability [15:8]`
0x80C / 0x81C	IP Core 0 / IP Core 1 – `mr_an_expansion [7:0]`
0x80D / 0x81D	IP Core 0 / IP Core 1 – `mr_an_expansion [15:8]`
0x820	Soft FPGA Logic ID Version Register 0 (Data = 0xAA)
0x821	Soft FPGA Logic ID Version Register 1 (Data = 0x55)
0x820	Soft FPGA Logic Control Register (bit D0 enables frame driver)

Pinouts

Table shows the primary pinouts for this reference design. Note that when you choose the 672-pin package when generating your GbE PCS IP core with Lattice IPexpress, pin-number preferences will be generated. These pin numbers correspond to the pinouts of the Lattice demo board for ECP2M, and are listed in Table . If you choose a different package, IPexpress will not generate a pin-out preference file, and the actual pin numbers for your design will be chosen during place and route. In this case, the pin numbers shown in Table are not applicable.

Table 5. Reference Design Pinouts

Signal Name	I/O	Pin Number	Description
ref_clk	In	N23	Reference Clock – 125 MHz clock source
rst_n	In	M7	Reset – Active low global reset
hdinp0	In	URC_SQ_HDINP0	Inbound SERDES P – Channel 0
hdinn0	In	URC_SQ_HDINN0	Inbound SERDES N – Channel 0
hdoutp0	Out	URC_SQ_HDOUTP0	Outbound SERDES P – Channel 0
hdoutn0	Out	URC_SQ_HDOUTN0	Outbound SERDES N – Channel 0
hdinp1	In	URC_SQ_HDINP1	Inbound SERDES P – Channel 1
hdinn1	In	URC_SQ_HDINN1	Inbound SERDES N – Channel 1
hdoutp1	Out	URC_SQ_HDOUTP1	Outbound SERDES P – Channel 1
hdoutn1	Out	URC_SQ_HDOUTN1	Outbound SERDES N – Channel 1
mdc	In	Note 1	MDIO Clock (0 to 50 MHz)
mdio	In/Out	Note 1	MDIO Bidirectional Data Signal
port_id_0[4:0]	In	Note 1	MDIO Port ID for IP Core 0
port_id_1[4:0]	In	Note 1	MDIO Port ID for IP Core 1
tck	In	TCK	JTAG Clock Source (0 to 50 MHz)
tdi	in	TDI	JTAG Data Input
tdo	Out	TDO	JTAG Data Output
tms	In	TMS	JTAG Test Mode Select
chan_0_activity_LED	Out	U6	Blue LED flashes when RX_DV on Chan 0 toggles
chan_0_autoneg_complete_LED	Out	U5	Green LED solid on when Chan0 autoneg completes OK
mon_activity_LED	Out	V2	Blue LED flashes when RX_DV on Chan 1 toggles
mon_autoneg_complete_LED	Out	W2	Green LED solid on when Chan1 autoneg completes OK
mon_pass_LED	Out	V1	Yellow LED solid on when Chan1 monitor data pattern OK
mon_fail_LED	Out	U2	Red LED solid on when Chan 1 monitor data pattern fails
enable_smi	In	Note 1	When high, IP core registers are controlled by MDIO. When low, IP core registers are controlled by ORCAstra
debug_link_timer_short_0	In	Note 1	When high, autoneg link timer on IP core0 is 2µsec (debug mode). When low, link timer on IP core0 is 10mec (normal).
debug_link_timer_short_1	In	Note 1	When high, autoneg link timer on IP core1 is 2µsec (debug mode). When low, link timer on IP core1 is 10mec (normal)

1. Location preferences are not specified for these pins. See your implementation results for actual pin locations.

Simulation

To run the reference design simulation, do the following:

1. Open ModelSim.

2. Under the **File** tab, select **Change Directory** and choose folder

   \<project_dir>\gbe_pcs_eval\<username>\sim\modelsim.

3. Under the **Tools** tab, select **TCL _ Execute Macro** and execute the ModelSim "**do**" script shown.

The simulation waveform results will be displayed in the ModelSim Wave window.

Implementation

To synthesize/map/place/route the reference design:

1. Select **Open Project** under the **File** tab in ispLEVER.

2. Browse to **\<project_dir\gbe_pcs_eval\<username>\impl\reference** in the **Open Project** dialog box.

3. Select and open **<username>_reference_eval.syn**. A this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.

4. Select the device top-level entry in the left-hand GUI window.

5. Implement the complete design via the standard ispLEVER GUI flow.

Note that the implementation flow described here is only validated for use with the Synplicity synthesizer. The reason for this is that some of the timing constraints are listed in the "physical" format. The names of these physical preferences change when a different synthesizer is used. It is possible to use the Precision RTL synthesizer with this reference design; however, you will have to rename the existing physical preferences into names compatible with the Precision RTL synthesis output.

Reference Design Performance and Utilization

Table 6. Performance and Resource Utilization[1]

Target Device	SLICEs	LUTs	Registers	EBRs	SERDES/PCS	I/Os	f_{MAX} (MHz)
LFE2M35E-5F672CES	1250	1720	1321	2	1	31	125

1. Performance and utilization characteristics are in Lattice's ispLEVER 7.0 software with Synplify synthesis. When using this IP core in a different software version or a different device density or speed grade, performance may vary.

References

- Technical Note 1124, *LatticeECP2M SERDES/PCS Usage Guide*
- *LatticeECP2/M Family Data Sheet*

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for Programmable Logic IC Development Tools ***category:***

Click to view products by Lattice ***manufacturer:***

Other Similar products are found below :

HLDC-DDR3-A DK-DEV-5SGXEA7N EK-10M50F484 EPXA4F672C2 EPXA4F672C3 EPXA4F672C1 K0161 LCMXO256C-S-EVN 12GSDIFMCCD SFP+X4FMCCD 88980182 P0582 HW-PWAC-2600317 DEV-17514 LCMXO3D-9400HC-B-EVN P0671 DK-K7-CONN-G P0467 LCMXO2-1200ZE-P1-EVN LCMXO2-4000HE-DSIB-EVN DK-DEV-4SGX530N LCMXO3L-SMA-EVN P006-006-2 EK-U1-VCU108-G A2F500-DEV-KIT-2 LCMXO3LF-9400C-ASC-B-EVN 471-014 EVAL-TPG-ZYNQ3 SL001 80-001005 P0466-EDU EK-10CL025U256 P0496 P0493 DK-SOC-10AS066S-A DK-DEV-10CX220-A 80-001002 iCE40UP5K-MDP-EVN RXCA10S066PF34-IDK00A ALTHYDRAC5GX ALTNITROC5GX ALTBERYLLC5GX 471-015 LFE3-17EA-USB3-EVN 1553 4332 Hinj HinjKit SNOMAKRKIT SnoMakrR10