Advanced **L**ogic **S**ynthesis for **E**lectronics
http://www.alse-fr.com

ALSE's **GEDEK**
User Guide
- Full Version -

Introduction

Firstly, thank your for using our Gedek IP !

We (A.L.S.E.) have designed this **Gigabit Data Exchange Kit** to help our customers implement data transfers between an FPGA design and one or several PCs. Our “both ends” solution is strikingly **simple to use**, **extremely compact** and allows **unbeatable transfer speeds** !

Since we wrote (and own) the entire code of this Intellectual Property, we master its quality, we can port it to any type of FPGA or ASIC, and we can deliver the source code.

This Kit has been carefully crafted, optimized, and it has already been used in many designs.

We created and packaged GEDEK to make your life as simple as possible, while allowing to achieve ultimate performance when necessary. The interfaces are very simple so that integrating GEDEK in a design is a breeze.

In many applications, the “standard, off-the-shelf GEDEK” (with a set of standard options available), as described briefly here, can be used as is. Just keep in mind that, even if your application seems to require some different functions, interface(s) or protocols, this Kit will very likely be a perfect fit after proper customization. The easiest (and probably cheapest) for a customer is clearly to define with us the exact needs of his application and let us customize this Kit to his exact needs.

Let's introduce the Gedek interface and its implementation inside your project.

Principle

This Kit is designed to implement very-high speed data exchanges between a hardware system (likely an FPGA-based application) and a host computer (PC under Windows or Linux Operating System for example) using a very standard and common Interface : the Gigabit Ethernet.

Ethernet links are both cheap, robust, and extremely common. Most if not all laptops and desktop computers come natively with an Ethernet interface. Connectors and cables are found just everywhere and many vendors offer low-cost hardware interface chips ("Phys").

The physical link can easily reach over 50 meters without extra hardware, and extending it further is easy using a standard (and cheap) switch.

As soon as it is necessary to move data between a hardware platform (FPGA) and a computer, GEDEK can be used.

GEDEK = **Data Rate** Performance + **Compactness** + **Easy Integration** + Simple **Win / *nix API**.

It is also possible to implement this communication system *between two FPGAs*. If you are interested by this variant, please contact ALSE.

Note : This Kit is not designed for use in Wide Area Networks, but only in the context of **short** and **local** links (preferably but not necessarily in a point-to-point configuration).

Typical application examples are :

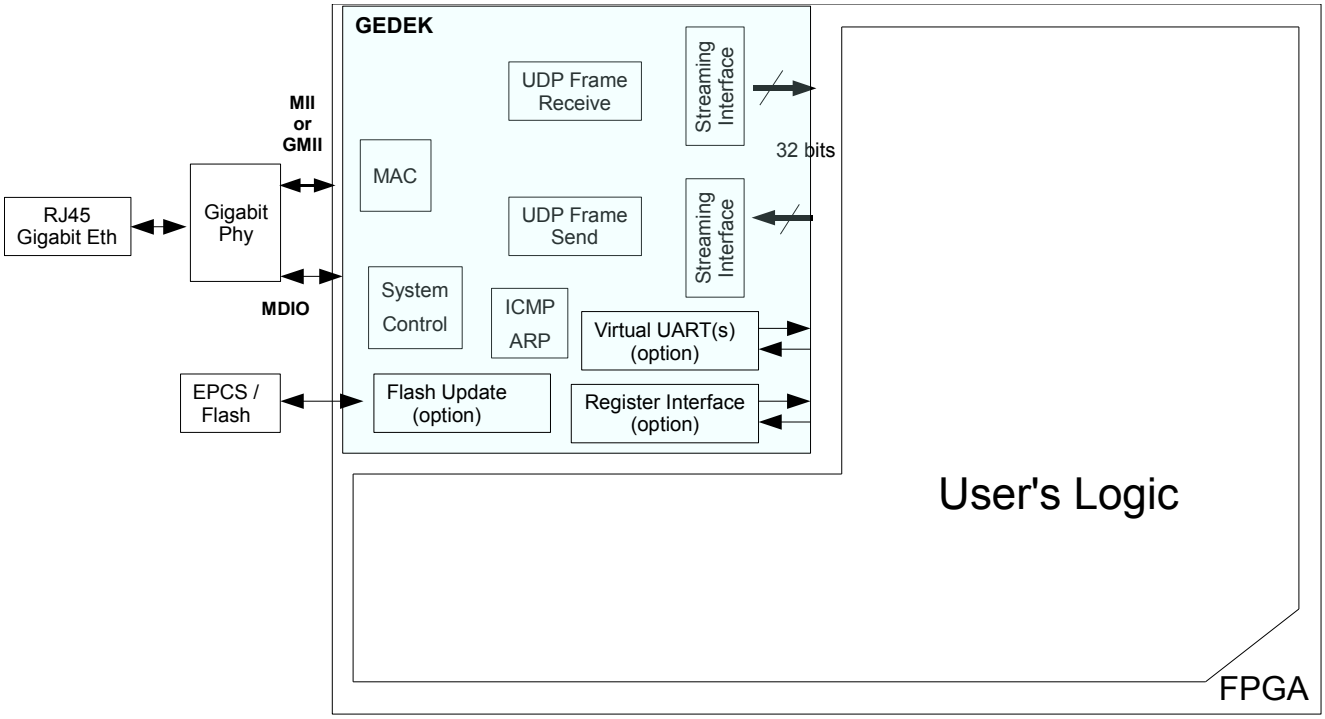
- Computer peripherals, like industrial printer
- High Speed Data Acquisition,Transfer & Pattern Generation
- Remote Data Collection,
- Video Streaming,
- Multiple Virtual UART(s) over Ethernet
- Remote FPGA reconfiguration and serial Flash update, etc...

To implement the complete link, you need :

- An FPGA board
- A Gigabit or 100 Mbits Ethernet PHY device connected to the FPGA.
  Note that ready-to-use Gigabit Ethernet extension boards exist.
- The ALSE GEDEK Demo Kit
- A host computer with a Gigabit or 100 Mbits Ethernet connection running under Linux or Windows. Note again that using GEDEK for inter-board communication (FPGA to FPGA) is perfectly possible, please contact ALSE.

Block Diagram

This is a simplified view of the FPGA side :



The RJ45 Ethernet connector is attached (through a transformer not represented here) to a physical interface device aka “PHY”. Several such devices exist, among which NS DP83865, Marvell 88e30xx, 88e1111, Sis196, Vitesse VSC86xx, etc...

The PHY device is connected to the FPGA through industry-standard interfaces :

- for Data interface, our GEDEK kit supports both **MII** (Media Independent Interface) and **GMII** (Gigabit Media Independent Interface). Other standards (SGMII, RGMII, SFP...) are available upon request, contact ALSE.
- For Control (access to the PHY's control and status registers), we implement and support the ubiquitous **MIIM** interface.

Internal Resources

The GEDEK solution is **very compact**.
Here is the typical logic utilisation by the IP :

Device Familly	LEs / ALMs	M4K / M9K
Altera Cyclone I	about 2000	<= 10 M4K
Altera Cyclone II	about 2000	<= 10 M4K
Altera Cyclone III	about 2000	<= 8 M9K
Altera Stratix III	about 2000	<= 8 M9K

Please note that actual numbers may vary with device family, Quartus Version & GEDEK options retained.

Tested Phy

The Gedek has been validated with many PHY used in the industy like :

Vendor	Reference
National Semiconductor	DP83865
Marvell	88E30xx
Marvell	88E1111
Marvel	88E1119
Intel	LXT971A
...	...

Component

The Gedek component is the following.

```
-- THIS CODE IS CONFIDENTIAL AND CANNOT BE DISTRIBUTED
-------------------------------------------------------------------
-- Copyright     : ALSE - http://alse-fr.com
-- Contact       : info@alse-fr.com
-- Nom du Projet : GEDEK
-- Nom du bloc   : Gedek
-- Description   : GEDEK Main file
-------------------------------------------------------------------
-- Auteur        : G. JOLI
-- Date          : Apr 2009
-- Version       : 2009.04
-------------------------------------------------------------------

Entity Gedek Is
  Generic (
      gMACAddress                 :       Std_Logic_Vector(47 Downto 0) := x"0007EDA1B2C4";
      gIPAddress                  :       Std_Logic_Vector(31 Downto 0) := x"C0A80112"
  );
  Port (
      -- System Signals
      Reset                       : In    Std_Logic;                      -- Asynchronous Active High Reset
      Clock                       : In    Std_Logic;                      -- Shall be >35Mhz. Typ 50 Mhz.

      -- Default MAC & IP Address
      HostDetected                : Out   Std_Logic;                      -- Asserted when connection is Up

      -- Gigabit Operation Mode
      GigaMode                    : In    Std_Logic := '1';               -- Working with dedicated feature

      ---- Register Interface
      Reg_Sel                     : In    Std_Logic;                      -- Register Interface "Chip Select"
      Reg_WaitRequest             : Out   Std_Logic;                      -- Register Interface Busy
      Reg_Address                 : In    Std_Logic_Vector (3 downto 0);  -- Register Interface Address
      Reg_Write                   : In    Std_Logic;                      -- Register Interface Write Request
      Reg_WriteData               : In    Std_Logic_Vector (31 downto 0); -- Register Interface Write Data
      Reg_Read                    : In    Std_Logic;                      -- Register Interface Read Request
      Reg_ReadData                : Out   Std_Logic_Vector (31 downto 0); -- Register Interface Read Data

      ---- UART Interface
      RxUart_Dav                  : Out   Std_Logic;                      -- Uart Rx Data Valid
      RxUart_Data                 : Out   Std_Logic_Vector(7 downto 0);   -- Uart Rx Data
      RxUart_Sop                  : Out   Std_Logic;                      -- Uart Rx Start Of Packet
      RxUart_Eop                  : Out   Std_Logic;                      -- Uart Rx End Of Packet
      --
      TxUart_Busy                 : Out   Std_Logic;                      -- Uart Tx Busy
      TxUart_Dav                  : In    Std_Logic;                      -- Uart Tx Data Valid
      TxUart_Data                 : In    Std_Logic_Vector(7 downto 0);   -- Uart Tx Data
      TxUart_Sop                  : In    Std_Logic;                      -- Uart Tx Start Of Packet
      TxUart_Eop                  : In    Std_Logic;                      -- Uart Tx End Of Packet
```

```
    ---- Stream Input
    RxUser_Dav                  : Out   Std_Logic;                      -- UDP Rx Stream Data Valid
    RxUser_Data                 : Out   Std_Logic_Vector(31 downto 0);  -- UDP Rx Stream Data
    RxUser_Sop                  : Out   Std_Logic;                      -- UDP Rx Stream Start Of Packet
    RxUser_Eop                  : Out   Std_Logic;                      -- UDP Rx Stream End Of Packet
    RxUser_SrcPort              : Out   Std_Logic_Vector(15 downto 0);  -- UDP Rx Source Port
    RxUser_DstPort              : Out   Std_Logic_Vector(15 downto 0);  -- UDP Rx Destination Port
    RxUser_Size                 : Out   Std_Logic_Vector(15 downto 0);  -- UDP Rx Packet Size
    RxUser_Checksum             : Out   Std_Logic_Vector(15 downto 0);  -- UDP Rx Packet Checksum

    ---- Stream Outputs
    TxUser_Busy                 : Out   Std_Logic;                      -- UDP Tx Stream Busy
    TxUser_Dav                  : In    Std_Logic;                      -- UDP Tx Stream Data Valid
    TxUser_Data                 : In    Std_Logic_Vector(31 downto 0);  -- UDP Tx Stream Data
    TxUser_Sop                  : In    Std_Logic;                      -- UDP Tx Stream Start Of Packet
    TxUser_Eop                  : In    Std_Logic;                      -- UDP Tx Stream End Of Packet
    TxUser_DstPort              : In    Std_Logic_Vector(15 downto 0);  -- UDP Tx Stream Destination Port

    ------------------------------------------------
    ------------------------------------------------
    -- Phy Interface
    Tx_clk                      : In    Std_Logic;                       -- Phy tx_clk
    Tx_dv                       : Out   Std_Logic;                       -- GMII Interface
    Txd                         : Out   Std_Logic_Vector(7 Downto 0);    -- GMII Interface
    --
    Rx_clk                      : In    Std_Logic;                       -- GMII Interface
    Rx_dv                       : In    Std_Logic;                       -- GMII Interface
    Rxd                         : In    Std_Logic_Vector(7 Downto 0)     -- GMII Interface
);
```

Generic Interface Description

There are 2 generics available in order to let you configuring some default value of your IP:

The Gedek default MAC Address can be configured using the gMACAddress generic
The Gedek default IP Address can be configured using the gIPAddress generic.

General Interface Description

The typical clock frequency for this IP is :

Operation Mode	Clock Frequency Required
10 Mbp/s	1 Mhz
100 Mbp/s	4 Mhz
1000 Mbp/s	35 Mhz

The IP requires an asynchonous Reset active high.

The HostDetected signal is asserted as soon as the Gedek IP has been pinged by the remote computer or FPGA.

HostDetected	Meaning
0	Gedek has not yet received the host's ping
1	Gedek has received the host's ping

The GigaMode signal is available when 100/1000Mbps MAC feature is available :

GigaMode	Operating Ethernet Speed
0	100 Mbps
1	1000 Mbps

Cpu Interface Description

The CPU interface is synchonous to the Clock signal of the interface.
This interface enables you to modify the IP Behavior.

Signal	Mode	Width	Description
Cpu_Sel	In	1	Register Interface ChipSelect
Cpu_RnW	In	1	Register Read (0) or Write (1) request.
Cpu_Add	In	4	Register address
Cpu_WData	In	32	Register value to be written @Cpu_Add
Cpu_RData	Out	32	Register value read @Cpu_Add
Cpu_WaitRequest	Out	1	When asserted, the register interface cannot accept any command

Register Mapping

Address	Functionality	Access	Reset Value
0x00	FPGA Board MAC Address (32 LSB)	R/W	Generic Dependent
0x01	FPGA Board IP Address	R/W	Generic Dependent
0x02	Destination MAC Address (32 LSB)	R/W	0x66322E2A
0x03	Destination MAC Address (16 MSB)	R/W	0x00000019
0x04	Destination IP Address	R/W	0xC0A801CF
0x05	Reserved	N/A	N/A
0x06	Reserved	N/A	N/A
0x07	Reserved	N/A	N/A
0x08	Reserved	N/A	N/A
0x09	Reserved	N/A	N/A
0x0A	Virtual Uart Link UDP Port Number *(When Available)*	R/W	0x00000017
0x0B	Reserved	N/A	N/A
0x0C	Reserved	N/A	N/A
0x0D	Reserved	N/A	N/A
0x0E	Reserved	N/A	N/A
0x0F	Reserved	N/A	N/A

Virtual Uart Interface Description *(When option available)*

The Virtual Uart reception interface is synchonous to the Clock signal.

This interface present all the Virtual frame received by the Gedek.

The availables signals are :

Signal	Mode	Width	Description
RxUart_Dav	Out	1	RxUart_Data is Valid
RxUart_Data	Out	8	Received Uart Data
RxUart_Sop	Out	1	First data in the Uart frame
RxUart_Eop	Out	1	Last data in the Uart frame

The Virtual Uart emission interface is synchonous to the Clock signal.

All the datas provided to this interface will be sent to the remote host by the Gedek on the virtual uart link.

The availables signals are :

Signal	Mode	Width	Description
TxUart_Busy	Out	1	TxUart Interface is not ready. Do not assert the Dav signal.
TxUart_Dav	In	1	RxUart_Data is Valid
TxUart_Data	In	8	Received Uart Data
TxUart_Sop	In	1	First data in the Uart frame
TxUart_Eop	In	1	Last data in the Uart frame

Please note that the TxUart_Busy signal will be asserted a few clock cycle before the gedek internal fifo will be full. It is made in order to make the feeder easier to design, however consider that as soon as this signal is asserted, you have to stop asserting the TxUart_Dav signal.

Streaming Interface Description

The data streaming reception is synchonous to the Clock signal.

This interface present all the UDP frames incomming to the Gedek.

The availables signals are :

Signal	Mode	Width	Description
RxUser_Dav	Out	1	RxUser_Data is Valid
RxUser_Data	Out	32	Received Data
RxUser_Sop	Out	1	First data in the frame
RxUser_Eop	Out	1	Last data in the frame
RxUser_SrcPort	Out	16	Frame source port
RxUser_DstPort	Out	16	Frame destination port
RxUser_Size	Out	16	Frame size
RxUser_Checksum	Out	16	Frame checksum

The data streaming emission is synchonous to the Clock signal.

All the datas provided to this interface will be sent to the remote host by the Gedek.

The availables signals are :

Signal	Mode	Width	Description
TxUser_Busy	Out	1	Tx Interface is not ready. Do not assert the Dav signal.
TxUser_Dav	In	1	RxUser_Data is Valid
TxUser_Data	In	32	Received Data
TxUser_Sop	In	1	First data in the frame
TxUser_Eop	In	1	Last data in the frame
TxUser_DstPort	In	16	Frame destination port

Please note that the TxUser_Busy signal will be asserted a few clock cycle before the gedek internal fifo will be full. It is made in order to make the feeder easier to design, however consider that as soon as this signal is asserted, you have to stop asserting the TxUser_Dav signal.

MII/GMII Interface Description

The signals of this interface have to be connected directly to the FPGA pins connected to the ethernet Phy.

Signal	Mode	Width	Description
tx_clk	In	1	Phy Tx clock: -Internal 125 gtx_clock signal in 1000 Mbps mode -FPGA input in 100Mbps
tx_dv	Out	1	Phy Tx Data Valid
txd	Out	8 or 4	Phy Tx Data : -8 bits in 1000 Mbps mode -4 bits in 100 Mbps mode
rx_clk	In	1	Phy Rx Clock
rx_dv	In	1	Phy Rx Data Valid
rxd	In	8 or 4	Phy Rx Data : -8 bits in 1000 Mbps mode -4 bits in 100 Mbps mode

You can also have the MIIM Phy Interface with the following signals

Signal	Mode	Width	Description
eth_mdc	Out	1	Serial Clock. About 1 Mhz
eth_mdio	InOut	1	Serial data in/out for Phy configuration
eth_reset_n	Out	1	Phy reset signal

=> In case of any PHY communication problems, do not hesitate to contact us !

Further informations

If you need some more information, please refer to the reference design code and documentation.

If you cannot find your answer, please contact ALSE at info@alse-fr.com,