# FPGA et ETHERNET

Patrick Nayman

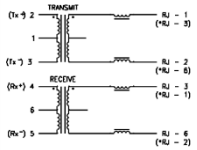
# Introduction

- Réaliser une connexion rapide entre un FPGA et un système hôte avec ETHERNET à 10-100-1000 Mbits (vitesse réelle).
- Avantages
  - Rapide
  - Possibilité d'un câble relativement long
  - Full duplex
  - Debugger gratuit TCPDUMP (linux) , WireShark (Windows)
  - Mise en œuvre relativement facile

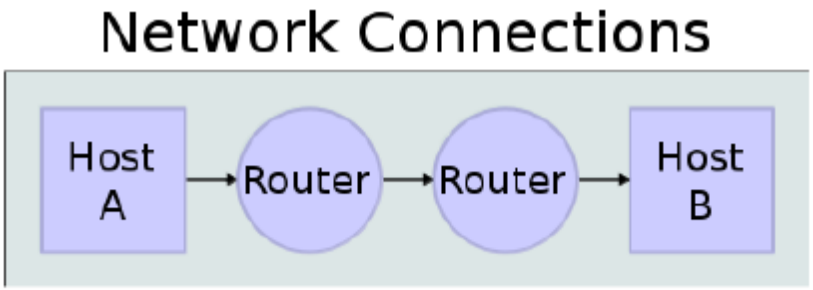
- Peu de ressources FPGA nécessaires :
  - 2200 LE (Logic Element)
  - <= 8 M9K (Memory block)

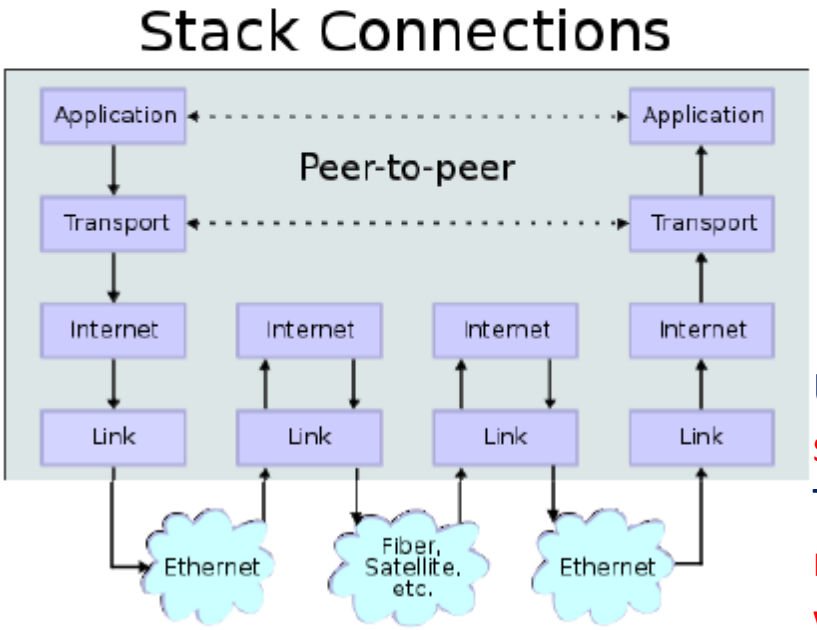
# Les Composants

- FPGA ALTERA série Cyclone (Stratix également)
- Basé sur un core GEDEK UDP (société ALSE)
  - Restriction : nécessite une liaison point à point (pas de collision)
- Nécessite 2 composants externes :
  - Ethernet PHY (sérialiseur) par exemple MARVELL 88E1119R
    - Connexion au FPGA par interface MII (Media Independent Interface) et GMII (Gigabit Media Independent Interface)
  - Non compatible avec les anciens PHY/MAC
  - Transformateur (peut être inclus dans la prise RJ45)

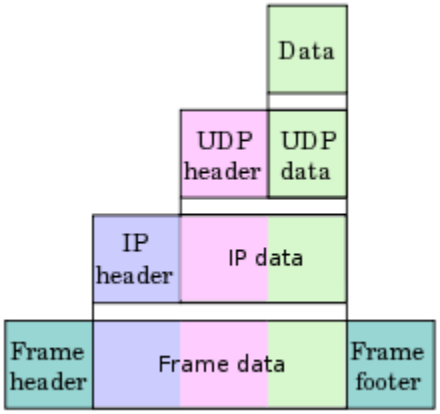


# Connexion Réseau




**Encapsulation of Application Data**




Application

Transport (TCP or UDP)

Internet

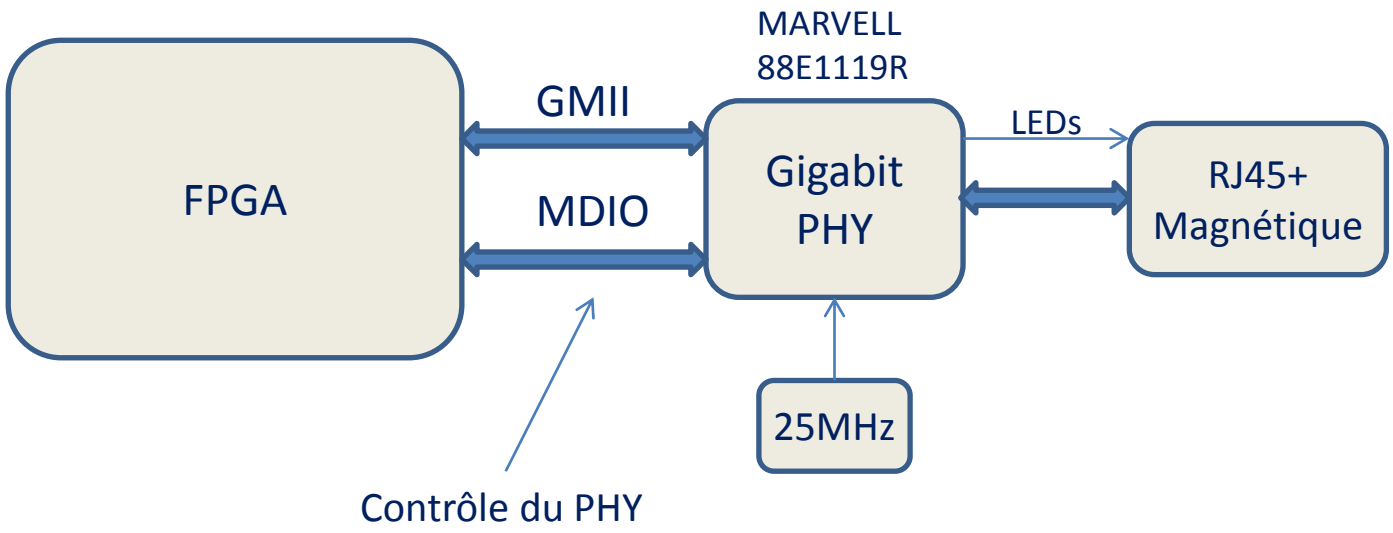
Link

UDP (User Datagram Protocol) :
simpler messaging transmissions
TCP (Transmission Control Protocol):
more complex protocol *reliable* transmission
when collisions

# Implémentation

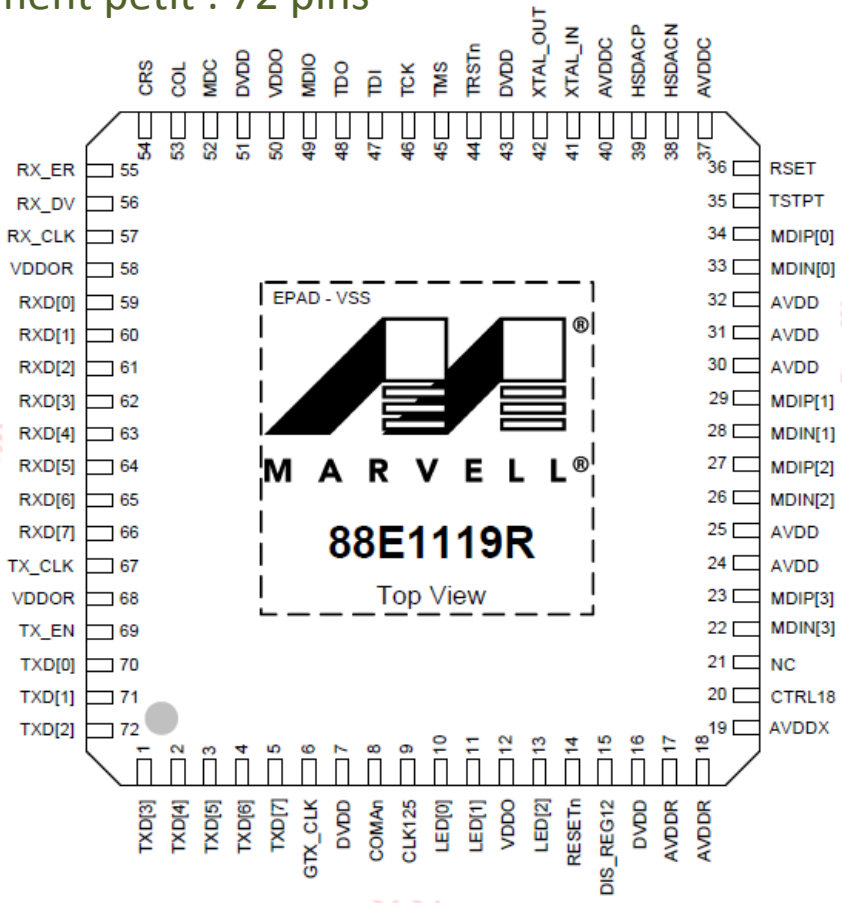
- Utilisation d'UDP (standard de transmission) à la place de TCP
  - Faibles ressources utilisées: low cost FPGA (~30€)
  - Plus rapide
- Possibilité de re-programmer le FPGA par ETHERNET
- 100Mbits or 1Gbits/s (cuivre)

# L’Interface globale



# Le circuit Marvell 88E1119R

Circuit relativement petit : 72 pins



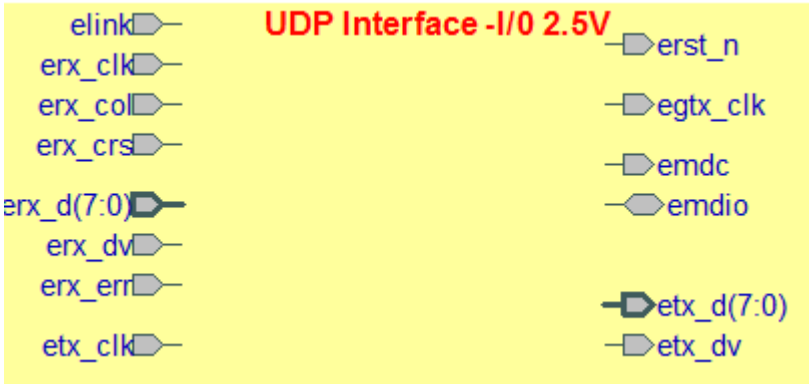
# L'Interface FPGA-Marvell

Ce qui entre et sort du FPGA

- Attention interface en 2.5V

En provenance du MARVELL

Vers le MARVELL

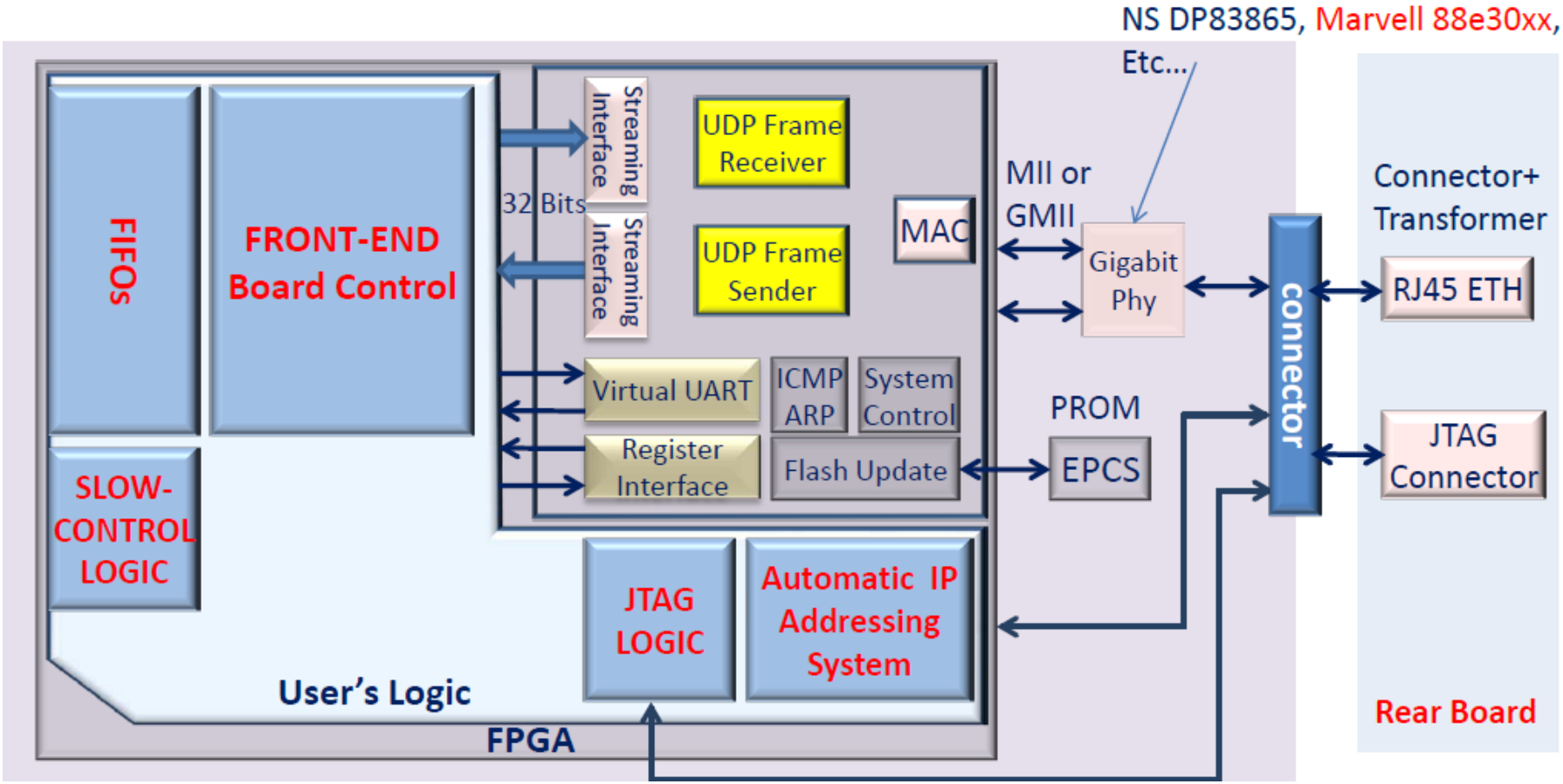



Clock reconstruite (125MHz)
Détection de collision
Détection de porteuse
Data reçue (8 ou 4 bits)
Data Valid
Erreur détectée sur le câble
Clock reçue (25MHz)

Clock pour gigabit (125MHz)
Management Interface Clock (1MHz)
Management Interface I/O Bidir
Data à envoyer (8 ou 4 bits)
Data Enable : etx_d valide

erx_d/etx_d : 8 bits en 1000Mbps
4 bits en 10-100Mbps

# Exemple de Projet




MII : Media Independant Interface
MAC : Media Access Control
UDP : User Datagram Protocol
ICMP : Internet Control Message Protocol
ARP : Address Resolution Protocol

# Les Modules disponibles dans l'IP

- Banc de registres internes (16x32 bits) (pas ETHERNET)
- Banc de registres (256x32 bits) (ETHERNET)
- UART Virtuelle (Permet d'envoyer des chaines de caractères et remplacer RS232, …CAN, I2C…
- Une interface données 32 bits
- Une interface de programmation de la PROM EPCS

# Banc de registres internes (16x32 bits)

- Permet de programmer (ou lire) :
  - FPGA Board MAC Address
  - Destination MAC Address
  - Destination IP Address
  - Etc.
- Accessible uniquement par les signaux (fournis par le GEDEK):

```
-- interface register
Cpu_Sel          : in STD_LOGIC;
Cpu_R            : in STD_LOGIC;
Cpu_W            : in STD_LOGIC;
Cpu_Ad           : in STD_LOGIC_VECTOR(3 downto 0);
Cpu_Wdata        : in STD_LOGIC_VECTOR(31 downto 0);
Cpu_RData        : out STD_LOGIC_VECTOR(31 downto 0);
Cpu_WaitRequest  : out STD_LOGIC;
```

# L’interface de données

```
REF_UserRxUDP_Sop  : Out    Std_logic;                    -- Flag indicating a new frame, Rx side
REF_UserRxUDP_Eop  : Out    Std_logic;                    -- Flag indicating a end of frame, Rx side
REF_UserRxUDP_Dav  : Out    Std_logic;                    -- Flag indicating that data is valid, Rx side
REF_UserRxUDP_Data: Out     Std_logic_vector(31 downto 0); -- Data, Rx side
REF_UserTxUDP_Busy : Out    Std_logic;  -- Signal indicating that the Gedek IP is ready
REF_UserTxUDP_Sop  : in     Std_logic;  -- Flag indicating a new frame, Tx side
REF_UserTxUDP_Eop  : in     Std_logic;  -- Flag indicating a end of frame, Tx side
REF_UserTxUDP_Dav  : in     Std_logic;  -- Flag indicating that data is valid, Tx side
REF_UserTxUDP_Data : in     Std_logic_vector(31 downto 0); -- Data, Tx side
REF_UserDest_Port     : in  Std_logic_vector(15 downto 0); -- PN
```

# Interface EPCS (PROM de programmation)

Pour la partie Hardware :

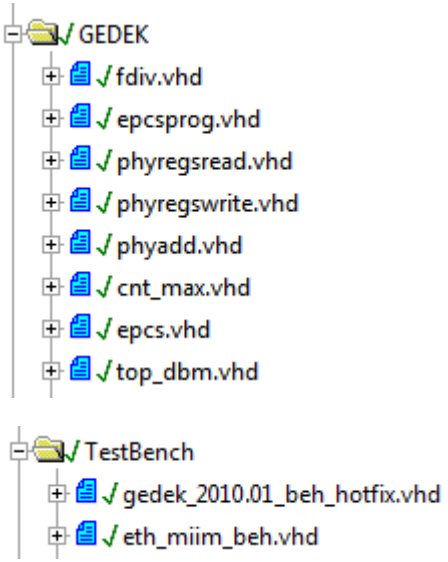
1- Dans le FPGA définir les pins suivantes et les connecter à top_dbm.vhd


2- Dans QUARTUS : Settings/Assignments/Device and Pin Options/Dual-Purpose Pins

- Data[0] Use as regular I/O
- Data[1] Use as regular I/O
- DCLK Use as regular I/O
- Flash_nCE/nCS Use as regular I/O

# Les différents VHDL pour l'IP GEDEK

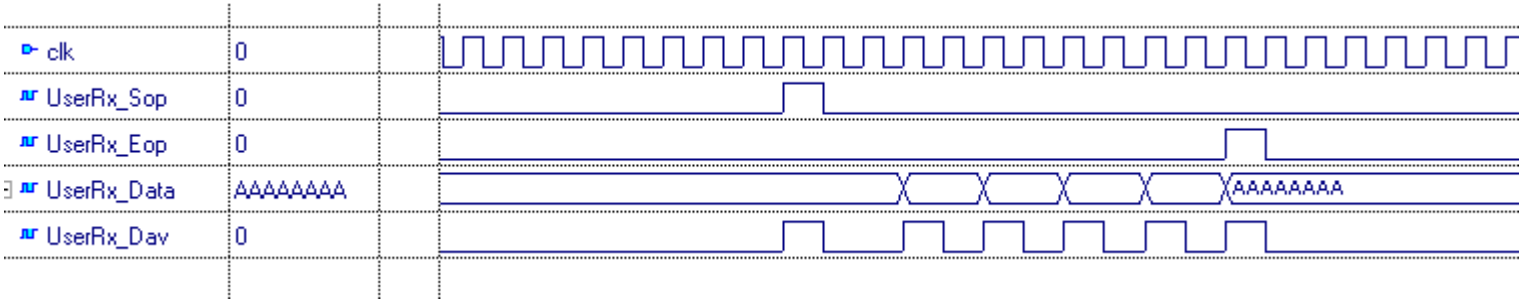
- Un ensemble de petits programmes est nécessaire :
  - top_dbm.vhd fait appel aux programmes suivants :
    - fdiv.vhd
    - epcsprog.vhd
    - phyregwrite.vhd
    - phyadd.vhd
    - cnt_max.vhd
    - epcs.vhd
    - eth_miim_beh.vhd (pour la simulation)
    - Gedek_2010.01_beh_hotfix.vhd (pour la simulation)
    - gedek_2010.01.vhd (pour la synthèse)



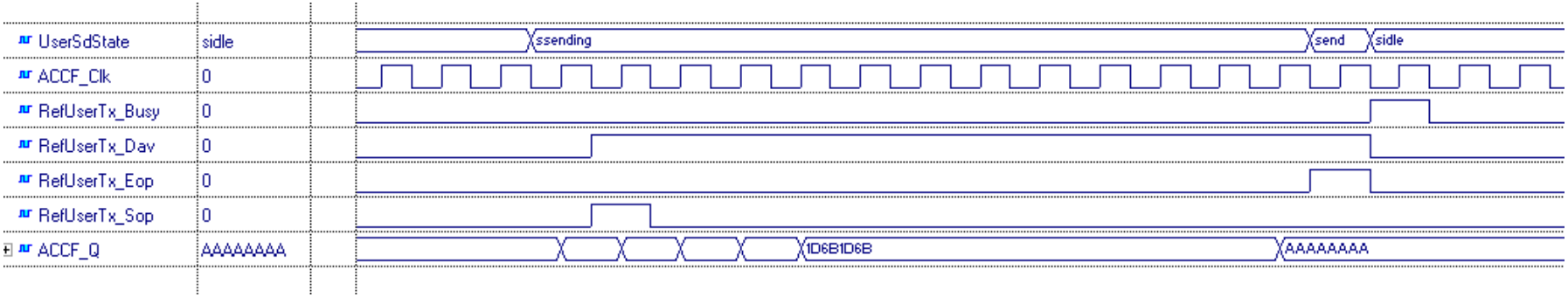
# Les Signaux

Flot : Start of Protocol, Data/Data Valid, End of Protocol

En lecture ETH vers FPGA

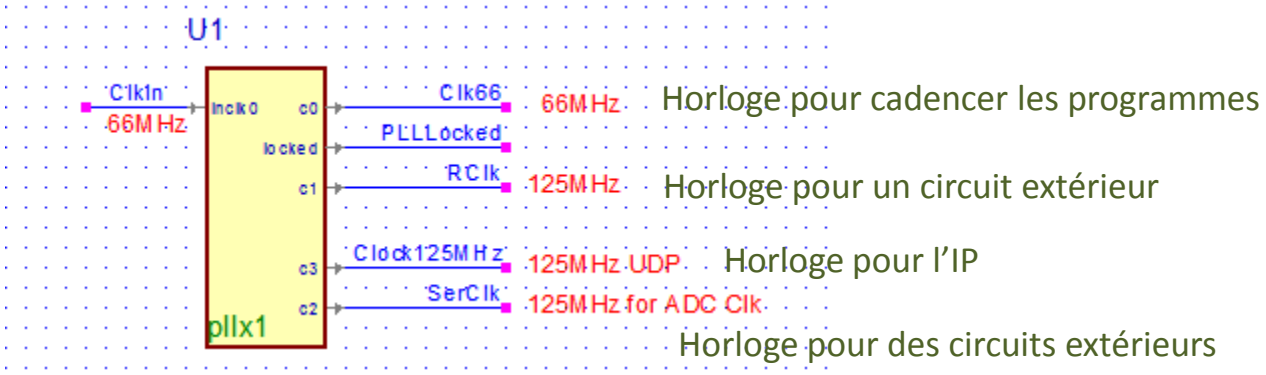



En écriture FPGA vers ETH



# Exemple : Hardware (1)

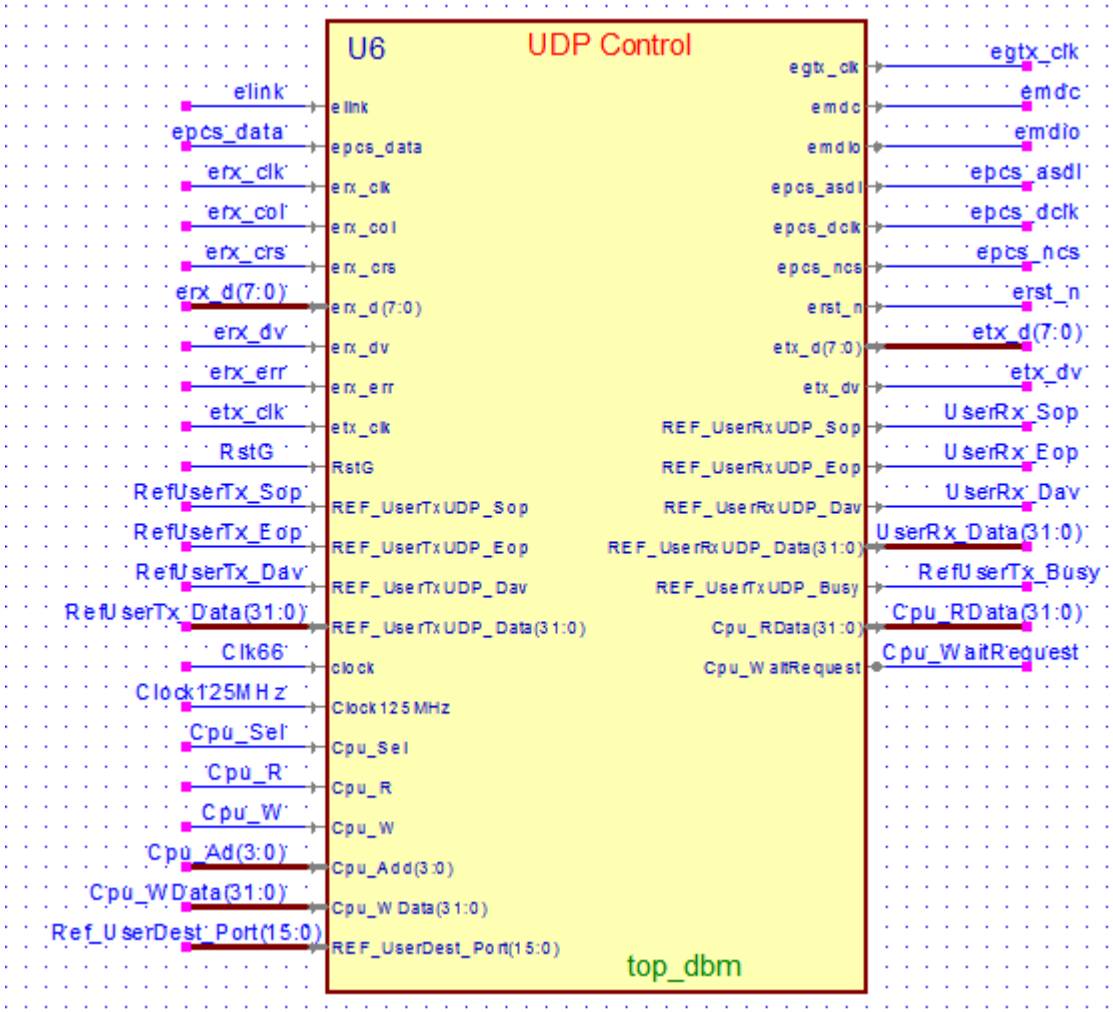
- La génération d'horloges : utiliser une PLL pour les horloges utilisateurs et pour l'IP



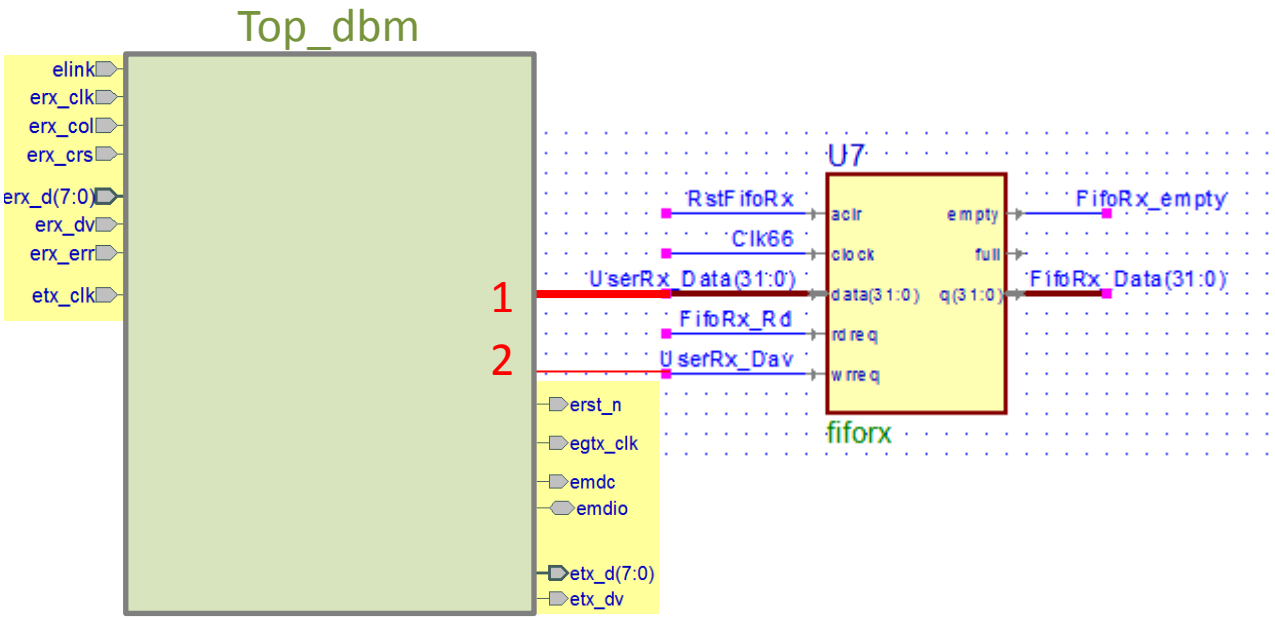

- On peut utiliser PLLLocked pour faire un Reset :

  Resetb <= not PLLLocked;

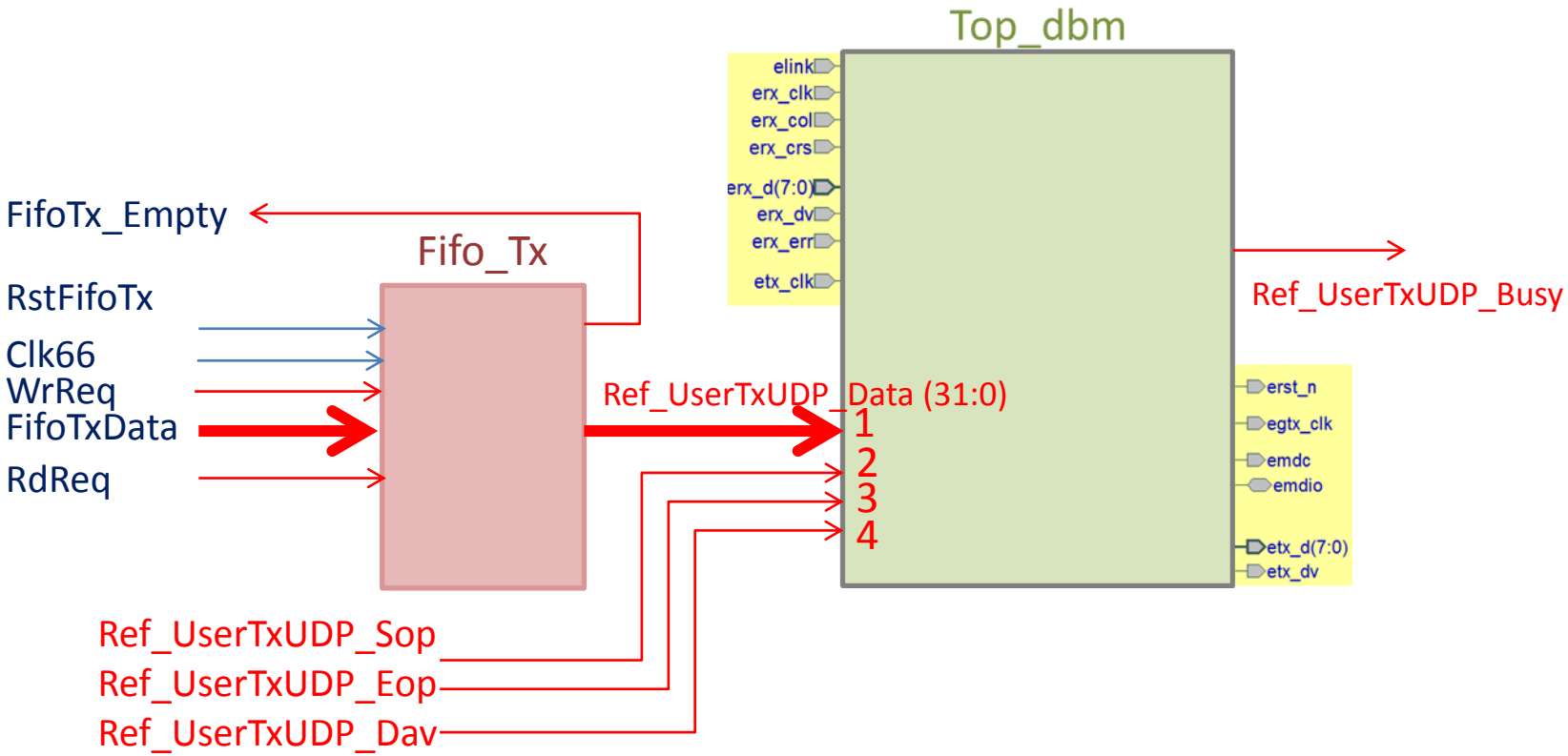
# Hardware : top_dbm.vhd (2)



# Comment se connecter ETH vers FPGA (1)



# Comment se connecter FPGA vers ETH (2)



# Comment se connecter FPGA vers ETH (3)

Exemple de machine d'états (VHDL) à utiliser :

```
when sSending =>
if RefUserTx_Busy='0'  then   -- Not Busy
    UserTx_Davi       <= '1';
    RefUserTx_Data  <= DAQF_q;
    if Phase=0 then
        RefUserTx_Sop <= '1';  -- Start of Protocol
        Phase     <= Phase+1;
    else
        RefUserTx_Sop <= '0';
        if FifoTx_Empty='1' then
            RefUserTx_Eop  <= '1';
            UserSdState     <= sEnd;
        end if;
    end if;
else
    UserTx_Davi    <= '0';
end if;
--
When sEnd =>
UserTx_Davi    <= '0';
RefUserTx_Eop <= '0';
UserSdState    <= sIDLE;
```

Etat sSending

Etat sEnd

Et en global :

RdReq <= '1' when (UserSdState=sSending) and FifoTx_Empty ='0' and RefUserTx_Busy='0'  else'0';

# Modification des adresses ETH (1)

- Changement d'un GENERIC dans top_dbm

```
iGedek : Entity work.gedek
   Generic Map (
    gFeatures       => "'HkZ+B<-Bm[$2&8TnF).3~iuGDB:?<)?v>%/nAy^e2[D(iOc67VJOr,;tKe9g&,{g" ,
    gMACAddress => x"0007EDA1B2C4",              -- Default GEDEK MAC Address is 00:07:ED:A1:B2:C4
    gIPAddress     => x"C0A80112",               -- Default GEDEK IP Address is 192.168.1.18
    Debug          => Debug
   )
```

# Modification des adresses ETH (2)

- Ecriture dans le banc de registres internes par le port CPU

| 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 | Status Register | R/W | 0x00000000 |
| 0x09 | Version Register | R | 0x<VER> |
| 0x0A | Virtual Uart Link UDP Port Number | 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 |

# Les contraintes de timing

- Le core Ethernet ne fonctionnera pas sans contraintes de timing !
- 2 cas selon la version de QUARTUS
  - Avant la version 10 : spécification des contraintes dans le fichier .qsf et selon les cas l'utilisation de TimeQuest est possible.
  - A partir de la version 10 : spécification des contraintes dans le fichier .sdc (« Synopsys Design Constraint », utilisation de TimeQuest).

# Pour simuler !

- 2 fichiers txt : input et output
  - Input définit les commandes ETHERNET
  - Output donnera le résultat de la simulation

# Exemple de input.txt

```
# Wait Delay: The provided integer is the waiting time in us
#
S 2
# Ping Frame -> Command is 'P'
# First Parameter is the IP of the ping sender (here source ip is c0.a8.01.00 = 192.168.1.0)
# Second Parameter is the target IP (here destination ip is c0.a8.01.17 = 192.168.1.23)
#
P c0a80100 c0a80117
# Data Frame -> Command is 'D'
# The 'D' command is followed by the destination port then the origin port in decimal !!
# The Frame content is provided on the same line with always 8 hexa. digits (ie 0 => 00000000)
#
# CNTRLNectarReg
#
D 04D2 4096
AAAAAAAA 00007E3E 00000001 00000002 00000003 00000004 00000005 AAAAAAAA
```

# Exemple de output.txt

0D 0A 47 45 44 45 4B 20 2D 20 44 65 66 61 75 6C 74 20 20 28 63 29 20 32 30 30 39 20 41 4C 53 45 2E 46 52

@11977.502 ns => Data Frame to 192.168.1.207:1200
BBBBBBBB 0000EE10 EDA1B2C4 C0A80112 66322E2A 66320019 C0A801CF C0A801CF BBBBBBBB

@15292.502 ns => Data Frame to 192.168.1.207:1200
AAAAAAAA 0000CCC3 C0A80112 00000001 AAAAAAAA

# Cas du script qsf

```
# -- Global Clock, Fmax & Timing constraints
set_instance_assignment -name GLOBAL_SIGNAL "GLOBAL CLOCK" -to Clock
set_global_assignment -name fmax_requirement 66MHz
set_global_assignment -name FMAX_REQUIREMENT "125 MHz" -section_id rx_clk
set_instance_assignment -name TCO_REQUIREMENT "10 ns" -from * -to gtx_clk
set_instance_assignment -name MIN_TCO_REQUIREMENT "9 ns" -from * -to gtx_clk
set_instance_assignment -name FAST_OUTPUT_REGISTER ON -to gtx_clk
set_instance_assignment -name TCO_REQUIREMENT "10 ns" -from * -to tx_*
set_instance_assignment -name FAST_OUTPUT_REGISTER ON -to tx_d*
set_instance_assignment -name FAST_OUTPUT_REGISTER ON -to tx_err
set_instance_assignment -name FAST_OUTPUT_REGISTER ON -to tx_dv
set_instance_assignment -name CLOCK_SETTINGS rx_clk -to rx_clk
set_instance_assignment -name TH_REQUIREMENT "0 ns" -from * -to rx_*
set_instance_assignment -name TSU_REQUIREMENT "3 ns" -from * -to rx_*
set_instance_assignment -name FAST_INPUT_REGISTER ON -to rx_d*
set_instance_assignment -name FAST_INPUT_REGISTER ON -to rx_err
set_instance_assignment -name FAST_INPUT_REGISTER ON -to rx_dv
```

# A propos du script sdc (TimeQuest)

- TimeQuest est un outil puissant qui demande un peu d'investissement…
  - Nécessité de contraindre toutes les E/S du design.
  - Assure un routage fonctionnel.

# Cas du script sdc

```
# Constrain MAC network-side interface clocks
create_clock -period "125 MHz" -name gedek_tx_clk  [ get_ports etx_clk  ]
create_clock -period "125 MHz" -name gedek_rx_clk  [ get_ports erx_clk  ]
create_clock -period "125 MHz" -name gedek_rx_clk_virtual
create_generated_clock -name {gedek_gtx_clk} -source [get_ports {erx_clk}]  [get_ports {egtx_clk}]
set_false_path -from [get_clocks $Clock125MHz] -to [get_ports {egtx_clk}]
# Cut the timing path betweeen unrelated clock domains
set_clock_groups -exclusive -group {gedek_gtx_clk} -group {gedek_tx_clk}  -group {gedek_rx_clk}
set_clock_groups -exclusive -group {gedek_rx_clk}  -group {gedek_gtx_clk} -group {gedek_tx_clk}
set_clock_groups -exclusive -group {gedek_tx_clk}  -group {gedek_rx_clk}  -group {gedek_gtx_clk}
set_clock_groups -exclusive -group {gedek_rx_clk}  -group [get_clocks $Clk66]
#************
# GMII - RX
#************
set_max_delay -from [get_clocks {gedek_rx_clk_virtual}] -to * 0
set_min_delay -from [get_clocks {gedek_rx_clk_virtual}] -to * 0
set_input_delay          -clock gedek_rx_clk_virtual -max [ expr 0.0 - $ETH_GMII_RX_Tsetup]  [get_ports {erx_d[*] erx_dv}]
set_input_delay          -clock gedek_rx_clk_virtual -min [ expr      $ETH_GMII_RX_Thold]   [get_ports {erx_d[*] erx_dv}]
set_input_delay -clock_fall -clock gedek_rx_clk_virtual -max [ expr 0.0 - $ETH_GMII_RX_Tsetup]  [get_ports {erx_d[*] erx_dv}] -
add_delay
set_input_delay -clock_fall -clock gedek_rx_clk_virtual -min [ expr      $ETH_GMII_RX_Thold]   [get_ports {erx_d[*] erx_dv}] -add_delay
```

Etc.

# CONCLUSIONS