

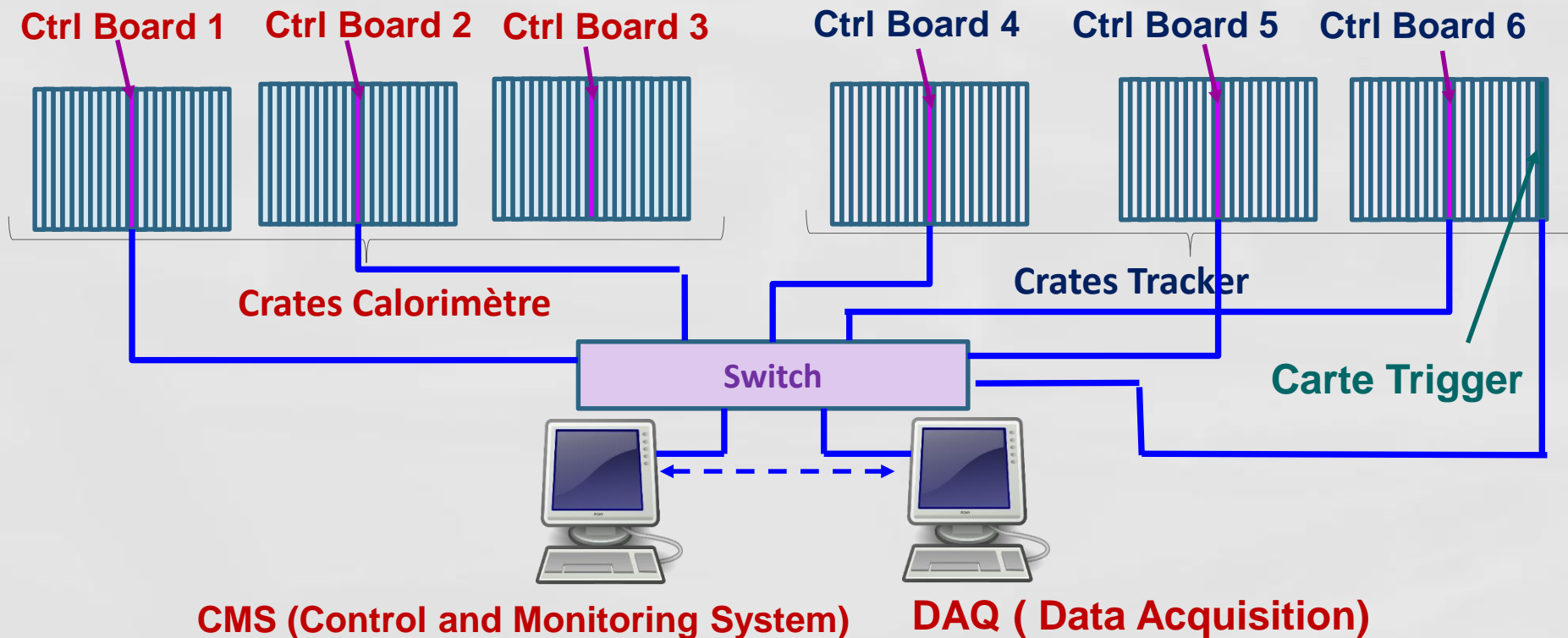
Développement d'une interface de communication Gigabit UDP sécurisée sans perte de données

D.Breton, C.Cheikali, J.Maalmi

CNRS/IN2P3/LAL Orsay

Pourquoi ce développement?

- Interface de Contrôle/DAQ pour l'expérience **SuperNemo**
- Utilisé dans d'autres systèmes maison: WaveCatcher, SAMPIC, bientôt CORTO ...
- Interface générique et 'open-source' dans la suite de **LAL-Usb -> LAL-Udp**
- Protocole maison : **LAL-Multi-Layer** adapté aux systèmes en arborescence



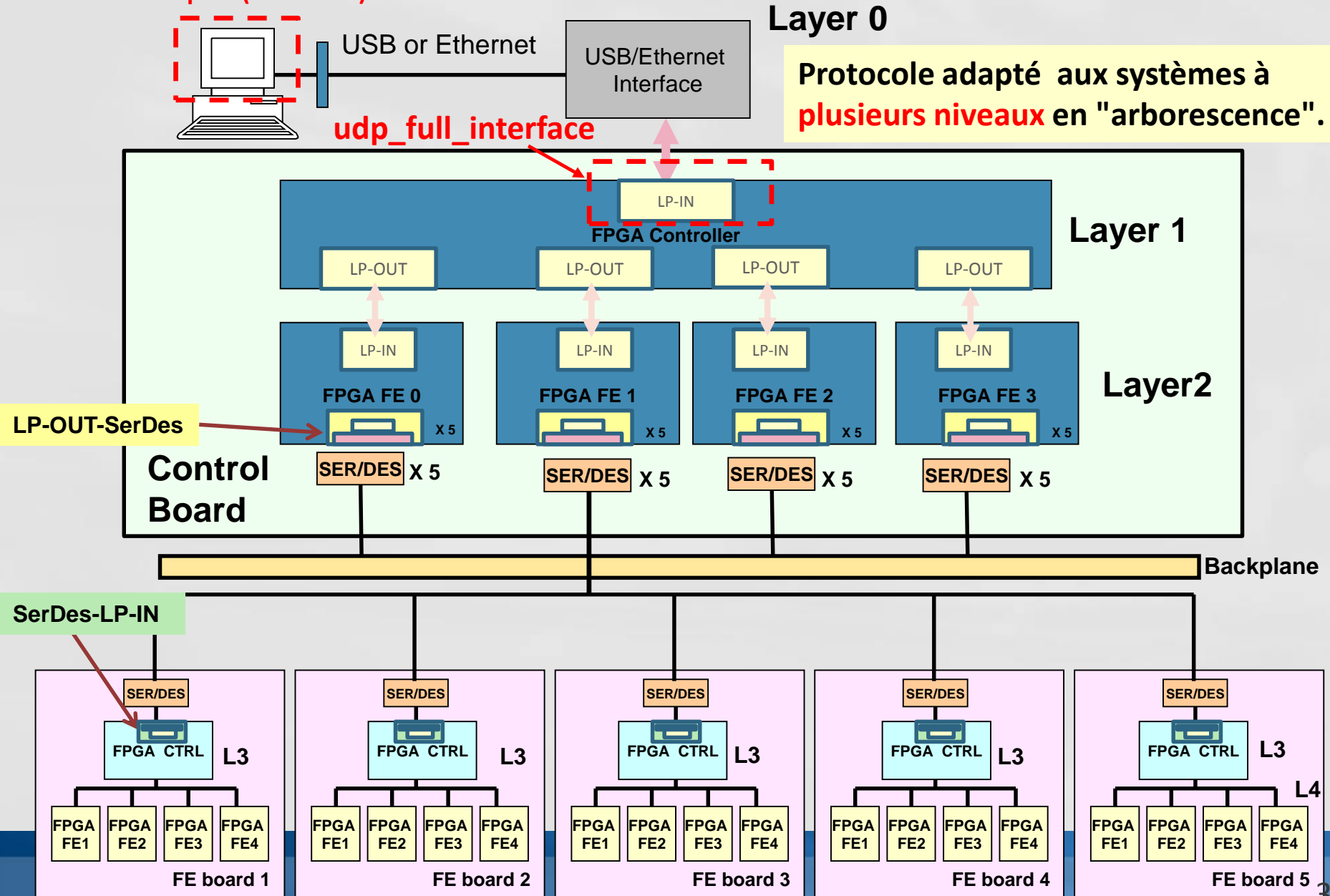
Architecture simplifiée de l'électronique de SuperNemo

Contraintes

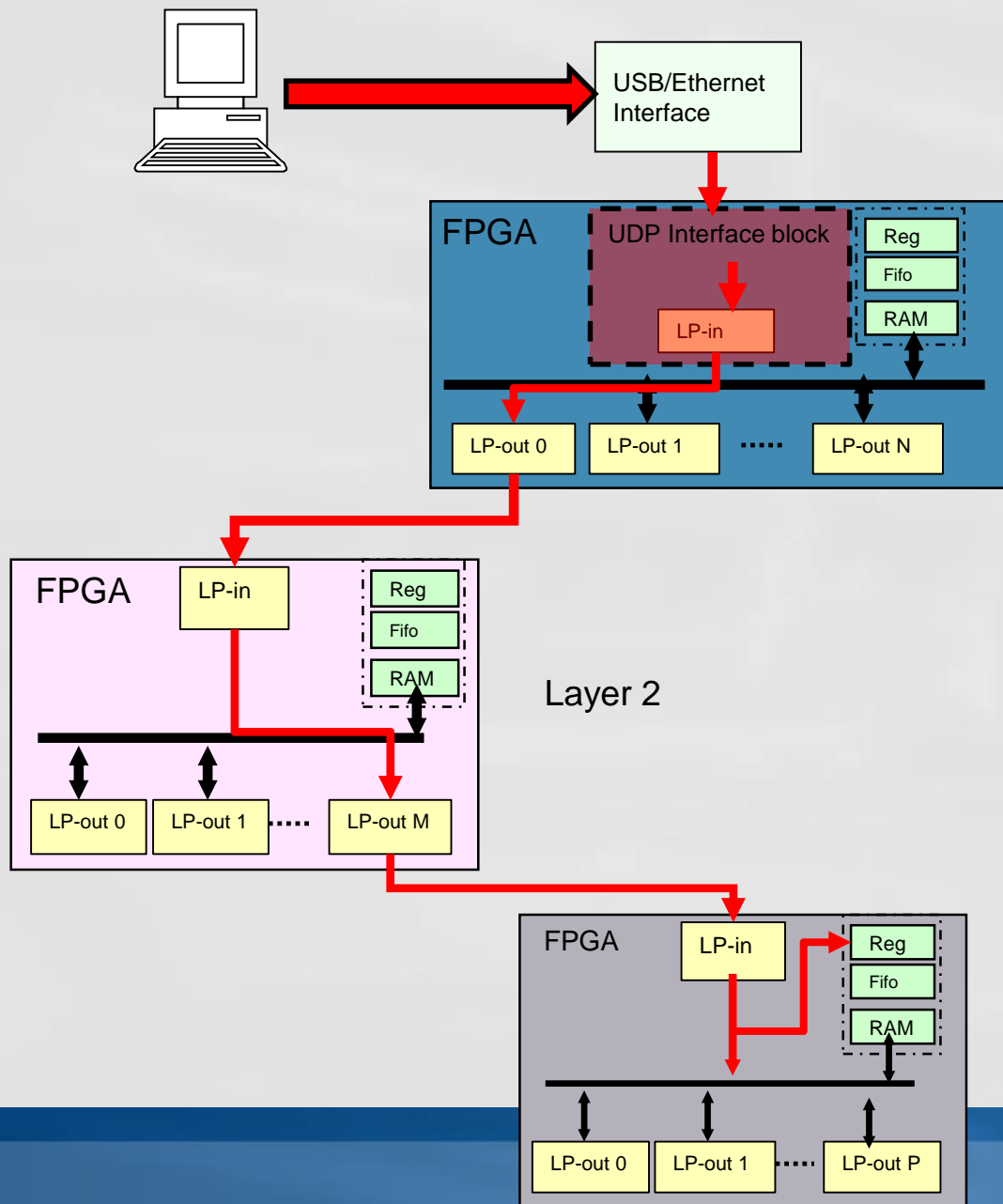
- Dans le cadre de SuperNemo:
 - Débit : $20 \text{ Mbits/lien} \times 6 = 120 \text{ Mbits max}$
 - **Aiguillage** des données de contrôle/commande et de Data vers les systèmes CMS et DAQ: donc sur des **ports différents**
 - Pas de perte de données
- Dans le cadre du développement générique:
 - Maximum de débit potentiel : s'approcher de 1Gbit/s
 - Minimum de perte de données : le débit s'adapte au Software d'Acquisition (comme l'USB)
 - Compatible avec l'interface Utilisateur LAL-Usb (**data(8 bits), subadd(7 bits), n_write, n_read, read_req, interrupt, n_wait**)
 - Simple d'utilisation : à la fois côté Hardware, Firmware et Software

Le Protocole LAL - Multi-Layer

Librairie : libudpML (C.Cheikali)



Le Contrôle



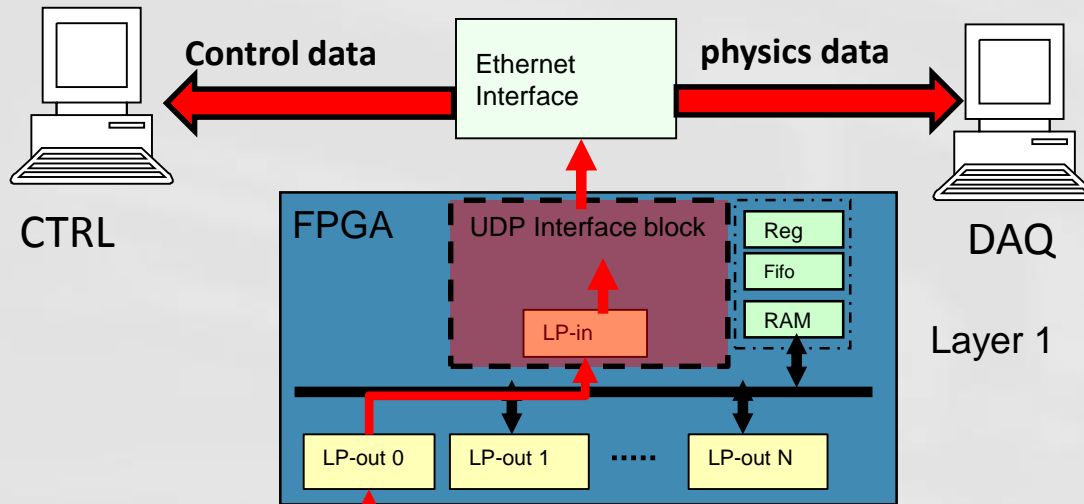
Layer 1 Protocole maison en

« **Poupées Russes** » :

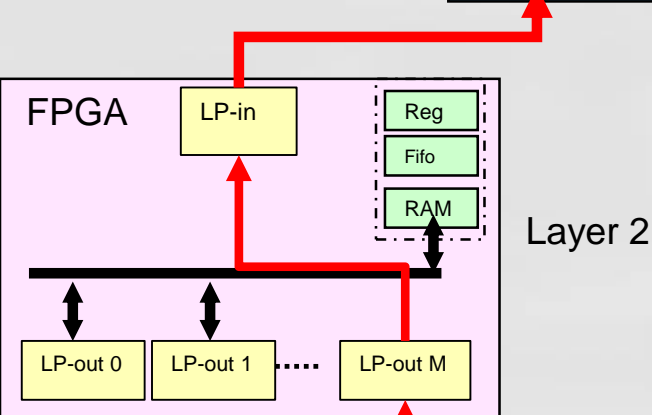
- Ici les trames sont décapsulées à chaque niveau.
- Les trames traversent chaque niveau hiérarchique de façon **transparente** jusqu'au niveau final.

Layer ...

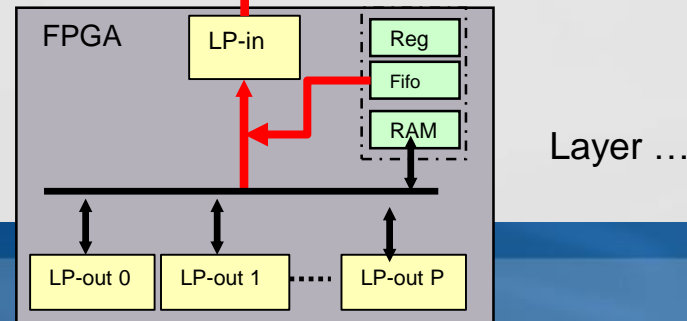
La lecture des données



Layer 1



Layer 2



Layer ...

- Protocole maison en « **Poupées Russes** » :
- Ici les trames sont encapsulées à chaque niveau.
 - Les trames de données passent d'un niveau à l'autre de façon **transparente**
 - On détecte s'il s'agit d'une trame DAQ ou CTRL à travers le header.
 - Elles sont aiguillées au niveau le plus haut vers le CTRL ou la DAQ en fonction de leur nature.

Pourquoi le Gbit Ethernet, pourquoi l'UDP?

● Pourquoi le GigaBit Ethernet ?

- Interface standard sur les PCs (1000-baseT) => **pas de hardware dédié côté DAQ + usage de cartes réseaux, switchs et câbles standards => faible coût**
- Possibilité d'atteindre le **Gigabit/s**
- Distance de communication sur cuivre >> USB => ok pour test beams par ex

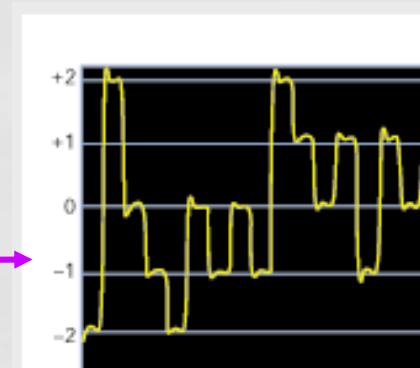
● Pourquoi l'UDP ?

- Protocole très **simple**
- L'interface (MAC) peut être implémentée dans un **FPGA à bas coût** (Ex : Cyclone III ou IV-E) en utilisant un sérialiseur externe (PHY) à 15€.
- Pas besoin de mémoire externe au FPGA ou de processeur

Mais :

- Possibilité connue de **perdre des paquets**

Attention : pas de PHY en 1000-baseT dans les FPGAs du fait du codage des signaux en PAM-5 (voir backup slides)



De quoi a-t-on besoin?



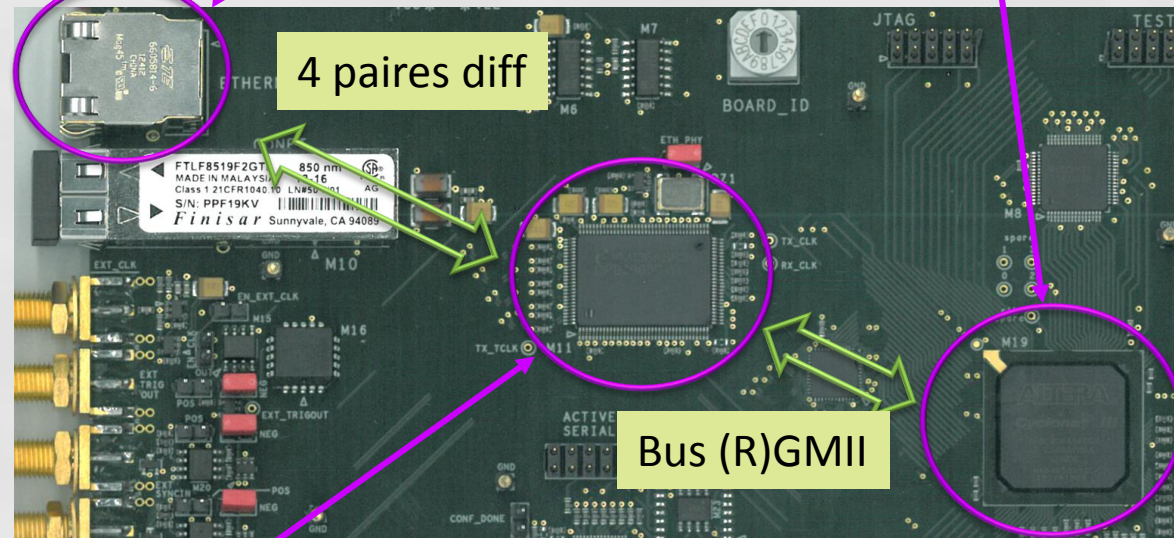
PC avec carte réseau

Câble
Ethernet
Cat5



Prise RJ45 avec
transformateurs intégrés :
TE 6605814-6

Cyclone III FPGA : EP3C40F484
Contient le **MAC**.



Couche physique (PHY) : SERDES du commerce, dédié à l'interface Ethernet : **DP83865DVH** de National Semiconductor (QFP 128 pins au pas de 0,5mm)

- 10Mbits, 100 Mbits ou 1 Gbit/s. Oscillateur dédié : **125 MHz**
- **4 paires différentielles** à **125 MHz** côté réseau, relié à une RJ45 à transformateurs intégrés (pour isolation DC HV et rejection de mode commun)
- **Bus (R)GMII** côté **FPGA** : standard pour Ethernet, **2 x (4) 8 bits de data** (Rx et Tx), 10 signaux de contrôle (dont seulement 4 indispensables)
- **Configurable au power-on par des résistances** (pas besoin d'interface de contrôle)

Structure Générale des Trames

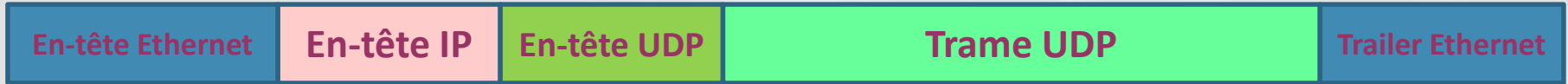
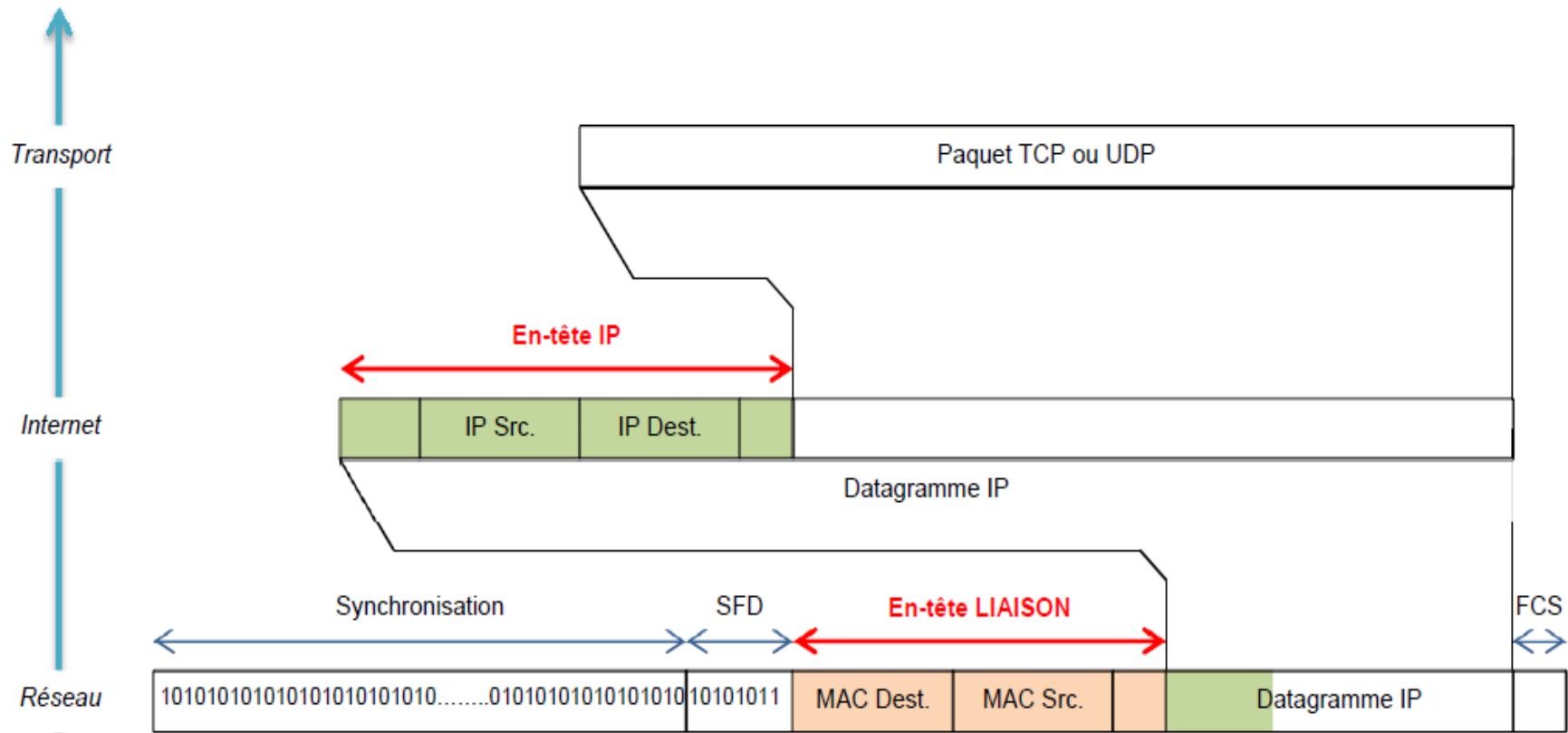
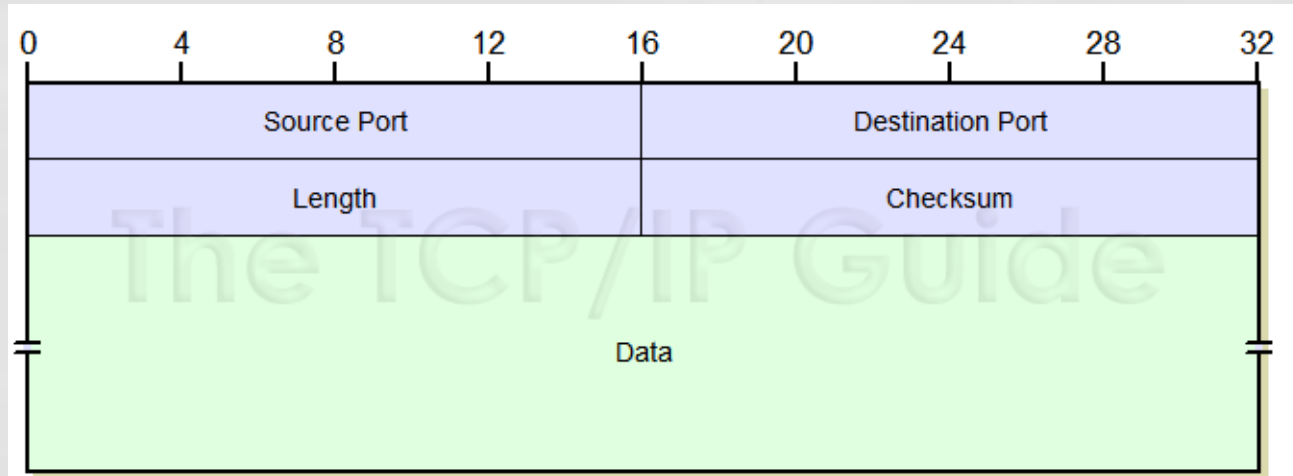


Schéma de la structure



Ce sont ces informations qui circulent sur le réseau

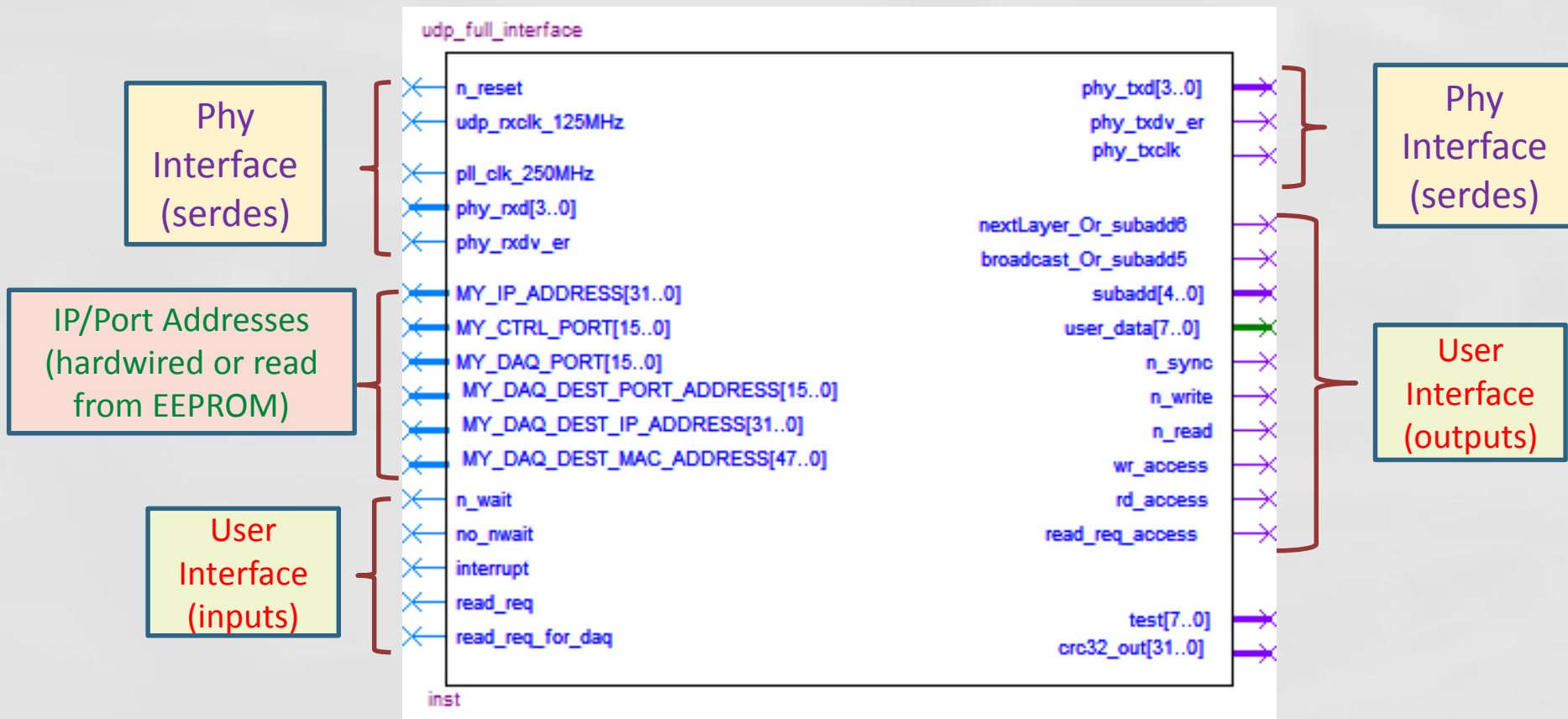
Datagramme UDP



- Le rôle de ce protocole est de permettre la transmission de données de **manière très simple** entre deux entités, chacune étant définie par **une adresse IP** et un **numéro de port**
- **Le calcul du Checksum est optionnel (= 0)**

L'interface 'UDP_Full_Interface' (MAC ou Medium Access Control + LP-IN)

L'interface côté **Utilisateur** est compatible avec l'interface utilisateur **usb_full_interface** (ML), seule l'horloge passe à **125MHz**



Le nombre d'Eléments Logiques est ~ 2200 (version basique avec ARP)

Que fait l'interface UDP?

Couche MAC:

- Décode les Trames Ethernet qui lui sont adressées (type IPv4, et UDP)
- Répond aux requêtes ARP qui lui sont adressées :

Rôle du protocole ARP

il permet de connaître **l'adresse physique (MAC address)** d'une carte réseau (ou nos cartes en l'occurrence) correspondant à une adresse IP donnée.

- Construit des trames Ethernet avec le protocole UDP :
 - **Fragmentation** des données > 1500 Octets
 - **Zéro Padding** pour les trames < 46 Octets
 - **En-têtes** Ethernet, IP et UDP
 - **Checksum**

Interface Utilisateur LAL

- Englobe l'interface LP-IN qui décode le protocole LAL
- Gère l'UDP d'une façon complètement transparente pour l'utilisateur
- Permet de différencier les trames CTRL des trames DAQ et de les **aiguiller vers deux ports différents**
- **Répond automatiquement à la carte réseau qui lui adresse une requête**

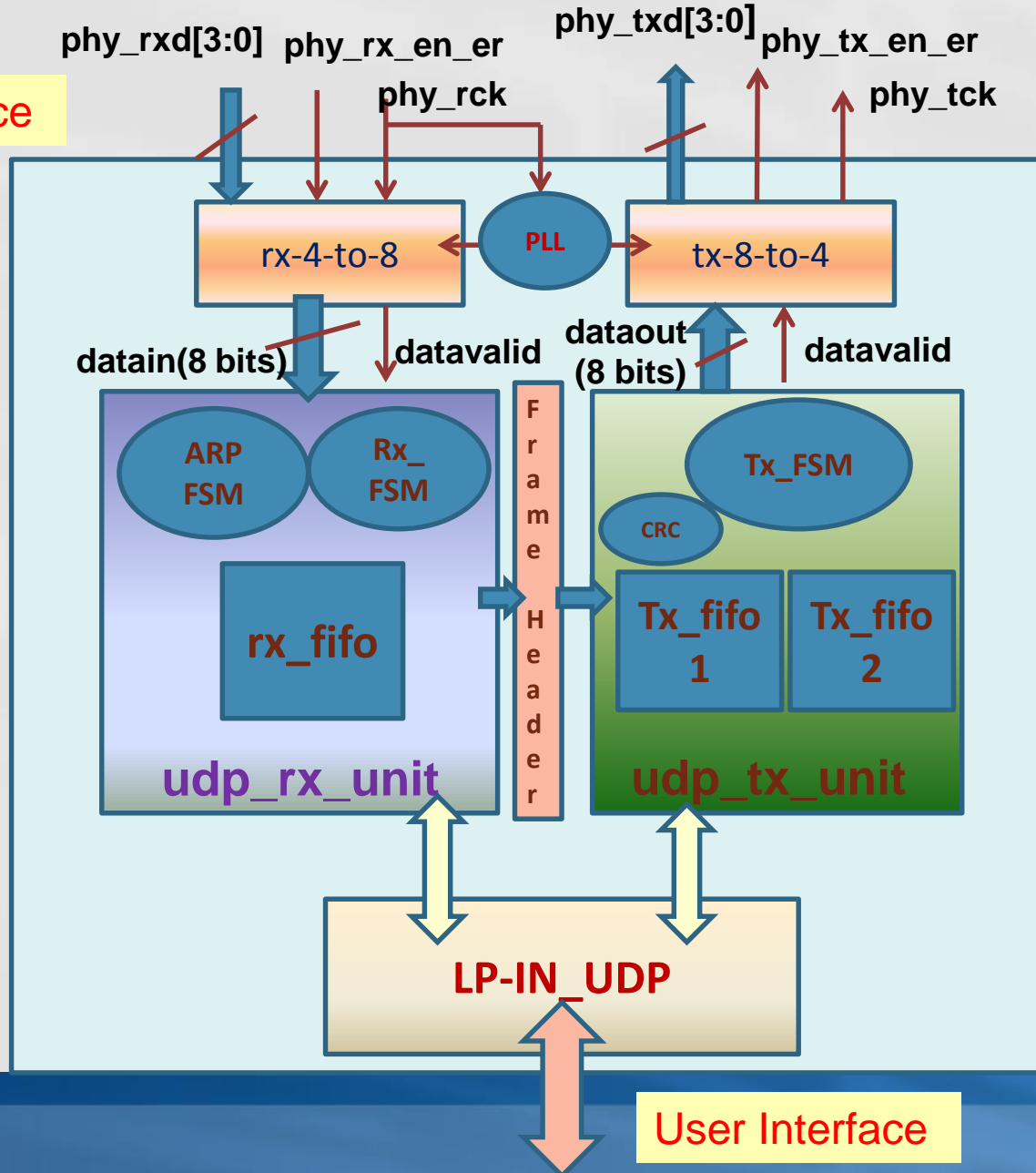
Ce que l'interface ne fait pas:

- Ne répond pas au ping
- Ne gère pas le DHCP, fonctionne uniquement sur réseau local

Un Bloc Firmware supplémentaire a été implémenté permettant de lire au **Reset** de l'interface les paramètres UDP depuis une **mémoire EEPROM** située sur la carte.

Schéma-Bloc de l'interface UDP

RGMII Interface



`phy_rck` =
`phy_tck` =
`usr_clk` =
125MHz

User Interface

Librairie UDP et Outil TestML

Une librairie « libUdp » quasi identique au niveau fonctionnalités à la librairie LALUsbML pour gérer le protocole LAL (ML) en UDP

Exemples de fonctions:

- ✓ *OpenUdpDevice(char *address, int port)*
- ✓ *UdpWrtML(int id, int *target_path_array, char sub_addr, void *buffer, int count);*
- ✓ *UdpReadML (int id, int *target_path_array, char sub_addr, void *buffer, int usercount);*

Un Outil de Test et debug :
TestML qui fonctionne en
USB et UDP

The screenshot shows the TestML application window. The interface is divided into several sections:

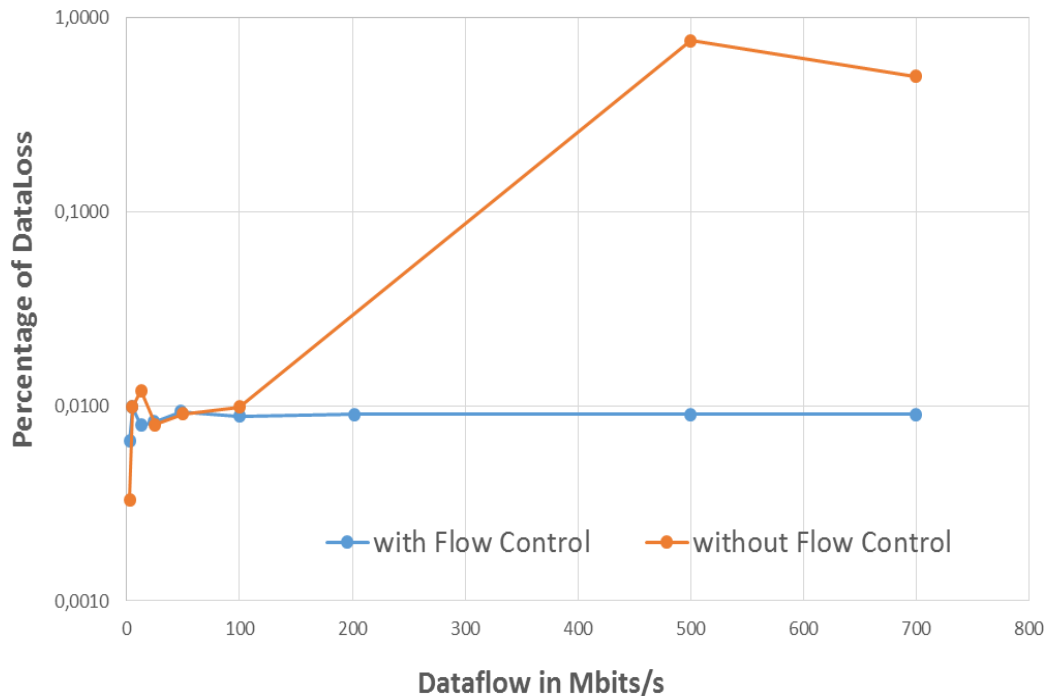
- Left Panel:** A tree view showing the network topology. It includes a 'UDP device' at IP 192.168.0.4 and several 'LPIn' (Layer 1, Layer 2, Layer 3) nodes.
- Interface type:** A dropdown menu set to 'Ethernet(UDP/IP)'. Below it, the 'UDP interface' section shows the IP address '192.168.0.4' and port '27015', with 'Open' and 'Close' buttons.
- USB device:** A dropdown menu set to 'USB-Blaster'. Below it, there are buttons for 'Scan USB bus', 'Connect and initialize', and 'Disconnect'. The 'FTDI Chip sync mode' is set to 'Asynchronous'.
- Right Panel:** A text area displaying the TestML version (156.00) and connection details for the USB-Blaster device (Serial number: 91d28408). It shows connection status, latency timer (16 ms), and device type (FT2XXBM).
- Device settings:** A section for configuring device parameters, including 'Timings (ms)' (Latency: 2, Tx timeout: 1000, Rx timeout: 1000) and 'Buffers size' (Tx buffer: 32768, Rx buffer: 32768).
- Input/output:** Fields for 'Sub-address' (0), 'Data' (0), 'Increment' (1), and 'Word count' (1).
- I/O mode:** Radio buttons for 'Read', 'Write', 'Write/Read', 'Read command', and 'Extended read'. 'Read' is selected.
- I/O data setup:** Fields for 'Word size' (8 bits), 'Byte order' (USB first), and 'Format' (LAL).
- Speed test:** A section with 'Start' and 'Stop' buttons, and fields for 'n loops' (100) and 'Duration (s)' (30).
- Options:** Checkboxes for 'Loop' (checked), 'Print' (checked), and 'Short print' (checked). 'Loop time' is set to 2 ms.
- Read request:** Fields for 'max frames' (100), 'frame size' (512), and 'Wait time ns' (1000).
- Bottom:** An 'Exit' button and a 'Target path' field with '0 1 0 -1' and 'Max layers' set to 4.

C.Cheikali - LAL

Débit et perte de données

- L'interface UDP standard fonctionne bien et atteint des débits continus jusqu'à 900 Mbits/s en fonction de la taille des paquets.
- Cependant on note bien le problème **de perte de paquets** surtout à fort débit.
- Développement d'un mode spécial où l'interface UDP ne peut envoyer qu'un **nombre limité** de trames, et attend ensuite un « **feu vert** » avant de reprendre la transmission des données.

% of DataLoss vs Dataflow in MBits/s



Tests effectués sous Windows

- sur des durées d'acquisition de **300 s**
- Ordinateur « au calme »
- Pour ralentir le débit on joue sur le délai entre les blocs de données

CONCLUSION : on ne peut pas garantir Zéro perte de données, même avec un délai très long entre les trames!

Vers un UDP sans perte de paquets

- On peut perdre des paquets, même avec des débits très faibles (0.01 %)
 - Pour l'expérience SuperNemo, ceci n'est pas acceptable
- De plus, les paquets peuvent arriver dans le **désordre**:
 - numéro de trame obligatoire dans le protocole
- Unique solution : le « **Hand-Check** » avec **retransmission** des paquets en cas de perte
- Ce que l'on veut :
 - Pouvoir réordonner les paquets facilement en flux continu.
 - Minimiser l'impact sur le débit

La sécurisation de l'UDP: le Principe

Ce qu'il faut dans le **principe** :

- Au niveau Firmware :
 - **Sauvegarder** les trames pour pouvoir les **renvoyer**
 - Les trames non acquittées doivent être **renvoyées** au bout d'un certain **Timeout**, et ce autant de fois que nécessaire **jusqu'à réception de l'acquittement**.
 - Ce système est nécessaire pour les **données de DAQ** (mais pas forcément pour le contrôle)
 - Il faut avoir la **possibilité d'acquitter plusieurs trames en même temps** pour minimiser l'impact sur le débit.
- Le software (bibliothèque) doit:
 - **Acquitter** toutes les trames reçues
 - **Gérer** les trames qui seraient **dupliquées**
 - **Réordonner** les trames en flux continu

Implémentation (1/2)

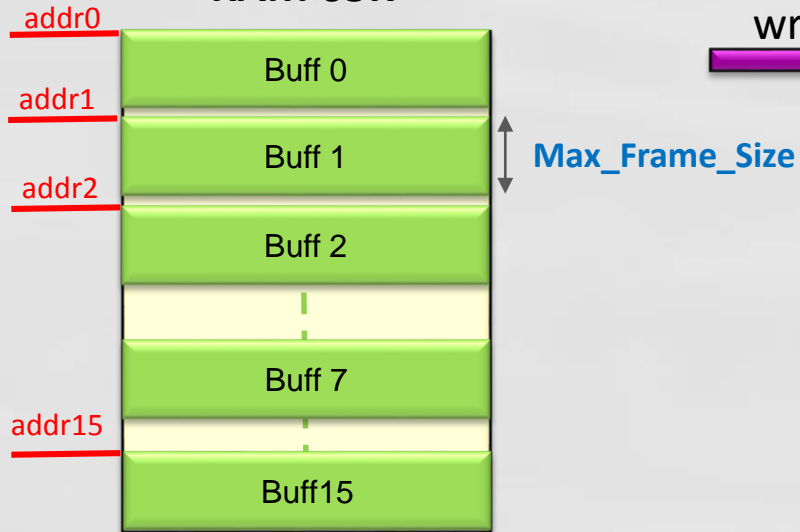
- On implémente un **bloc RAM** permettant de sauvegarder jusqu'à **16 paquets** de données envoyés (en fonction de la taille maximum des paquets)
- Chaque Trame a un **Identifiant** (32 bits) rajouté dans le protocole LAL
- Pour acquitter une trame, le software écrit dans un **registre dédié**, **l'octet de poids faible** de l'identifiant de la trame à acquitter.
- On peut ainsi **acquitter plusieurs trames** lors d'un **même accès en écriture**
- L'interface **renvoie un paquet** s'il n'a pas été acquitté au bout d'un certain **Timeout** programmable.
- **Arrêt d'envoi** des paquets si la mémoire de sauvegarde est remplie ou si un paquet d'identité N est resté non acquitté quand on atteint le paquet $N + 256$
 - ➔ Ceci va permettre d'avoir un **buffer de taille maximum bien définie** au niveau du Software pour **réordonner les données facilement** en flux-continu.

Nombre d'éléments logiques supplémentaires **~4000**

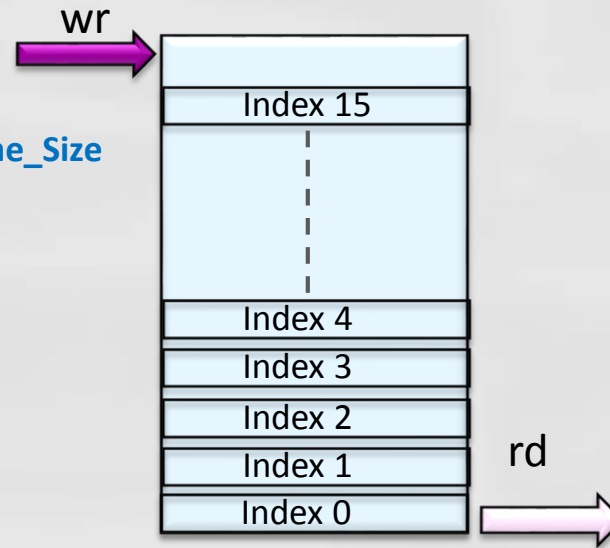
Implémentation (2/2)

Implémentation au sein du bloc LP-IN-UDP : car le mécanisme utilise l'accès à des registres à travers le protocole LAL

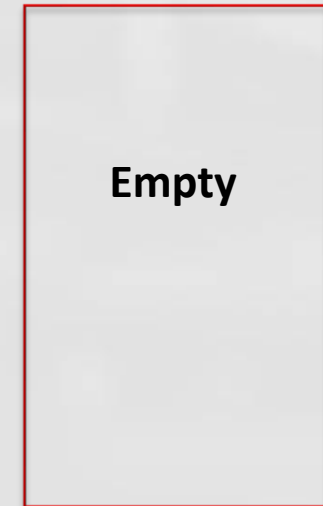
RAM 65K



Fifo_Free_Buffer_Index



Fifo_Timeout_Buffer_Index



Registres programmables pour la configuration:

- **Enable/Disable** Système d'acquittement
- **Max_Frame_Size (16 Bits)**
- **TimeOut** (pour le renvoi de paquets) (**32 Bits en pas de 8 ns**)

Registre dédié à l'acquittement

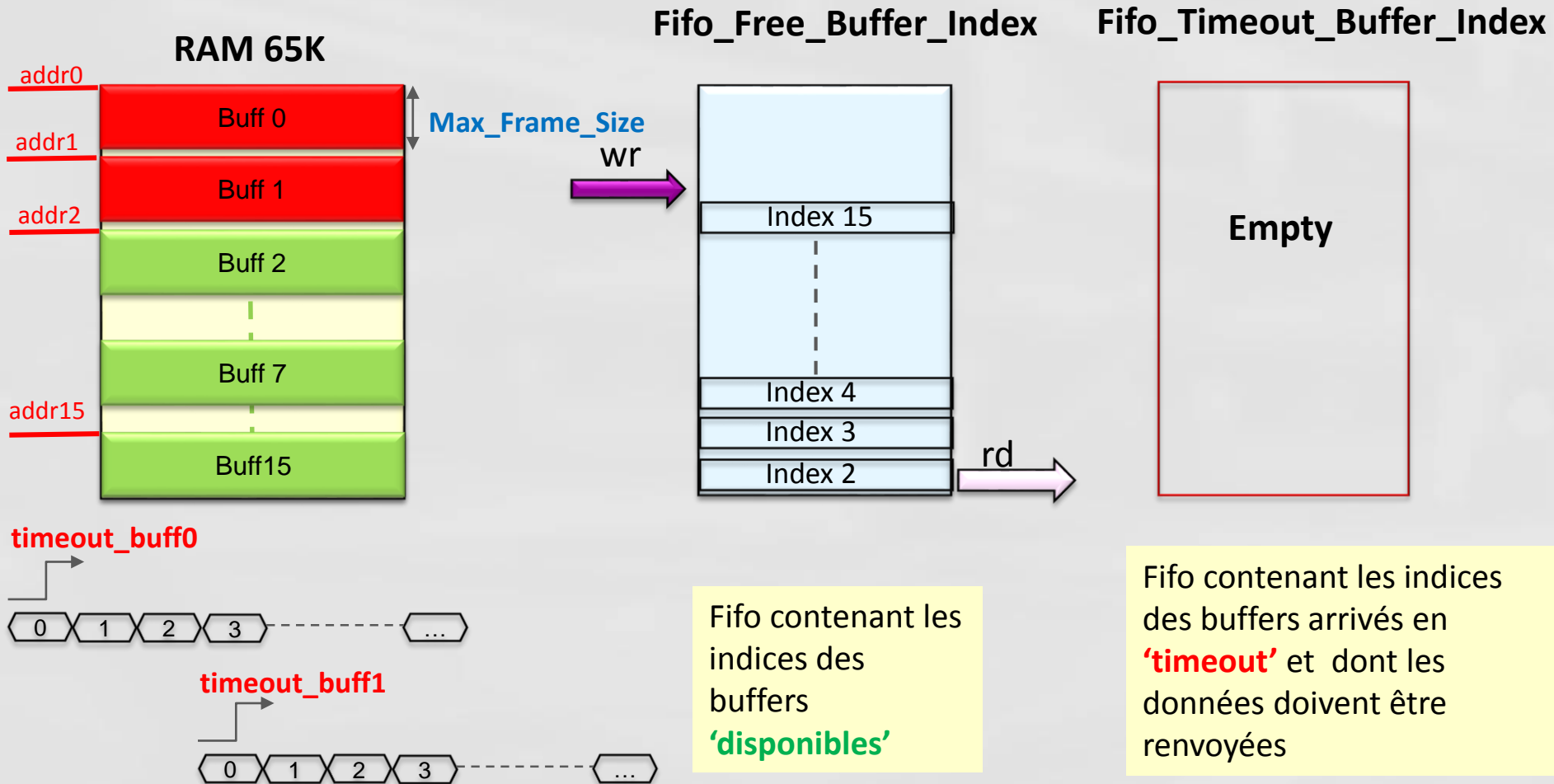
- **Frame_Ack_Reg**

Fifo contenant les indices des buffers '**disponibles**'

Fifo contenant les indices des buffers arrivés en '**timeout**' et dont les données doivent être renvoyées

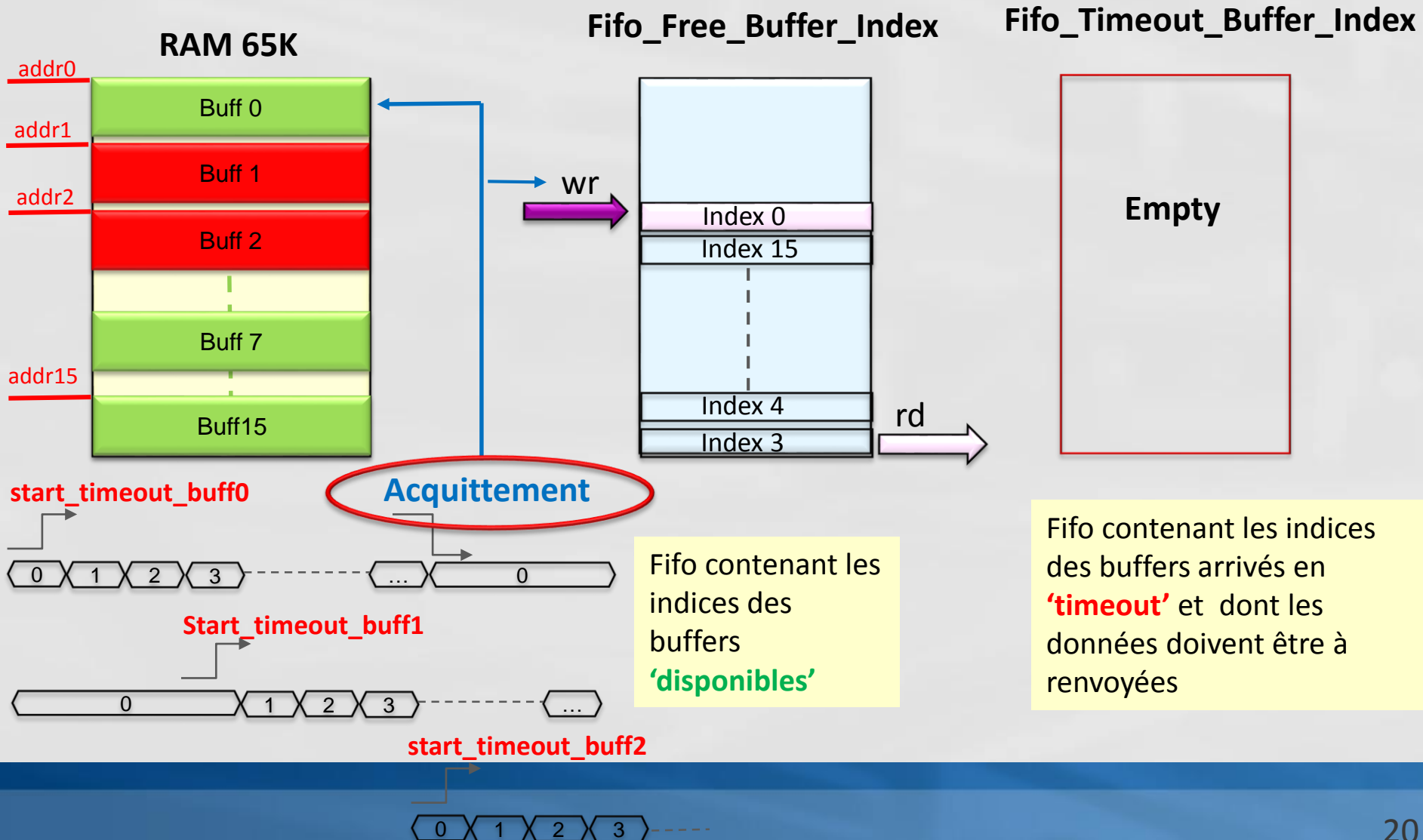
Implémentation (2/2)

Implémentation au sein du bloc LP-IN-UDP : car le mécanisme utilise l'accès à des registres à travers le protocole LAL



Implémentation (2/2)

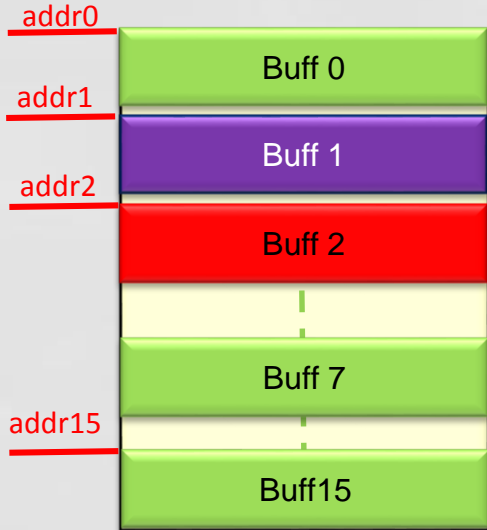
Implémentation au sein du bloc LP-IN-UDP : car le mécanisme utilise l'accès à des registres à travers le protocole LAL



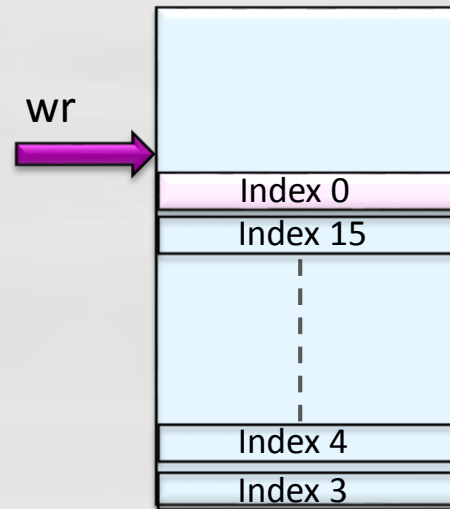
Implémentation (2/2)

Implémentation au sein du bloc LP-IN-UDP : car le mécanisme utilise l'accès à des registres à travers le protocole LAL

RAM 65K



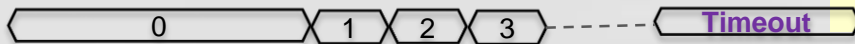
Fifo_Free_Buffer_Index



Fifo_Timeout_Buffer_Index



Start_timeout_buff1



Fifo contenant les indices des buffers 'disponibles'

Fifo contenant les indices des buffers arrivés en 'timeout' et dont les données doivent être renvoyées

start_timeout_buff2



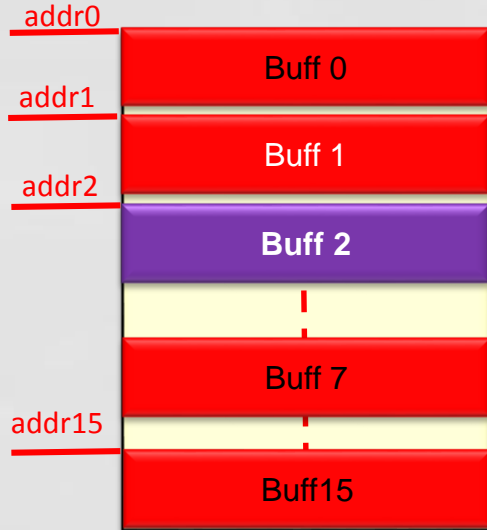
Implémentation (2/2)



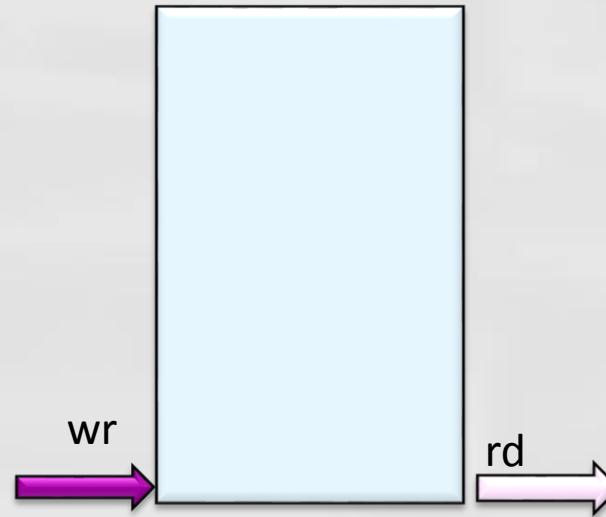
Deux possibilités pour l'arrêt d'envoi de nouvelles trames :

1. Mémoire de Sauvegarde remplie

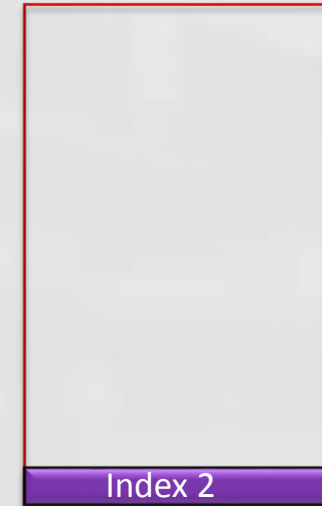
RAM 65K




Fifo_Free_Buffer_Index



Fifo_Timeout_Buffer_Index



 On arrête
d'envoyer de
nouvelles trames!

Fifo contenant les
indices des
buffers
'disponibles'

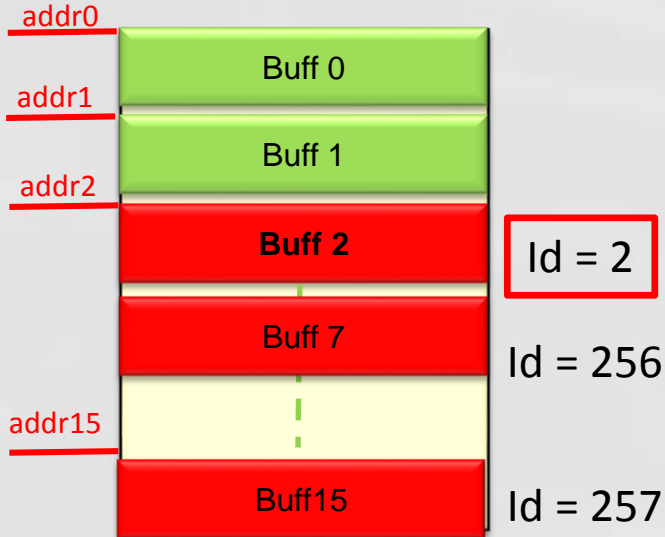
Fifo contenant les indices
des buffers arrivés en
'timeout' et dont les
données doivent être
renvoyées

Implémentation (2/2)

Deux possibilités pour l'arrêt d'envoi de nouvelles trames :

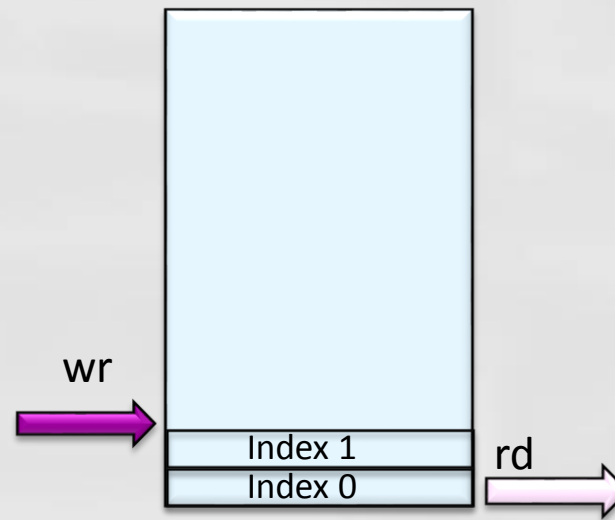
2. Si la plus vieille trame non acquittée est **N** et on arrive à la trame **N + 256**

RAM 65K



Fifo_Free_Buffer_Index

Fifo_Timeout_Buffer_Index



STOP 258 % 256 = 2 !!

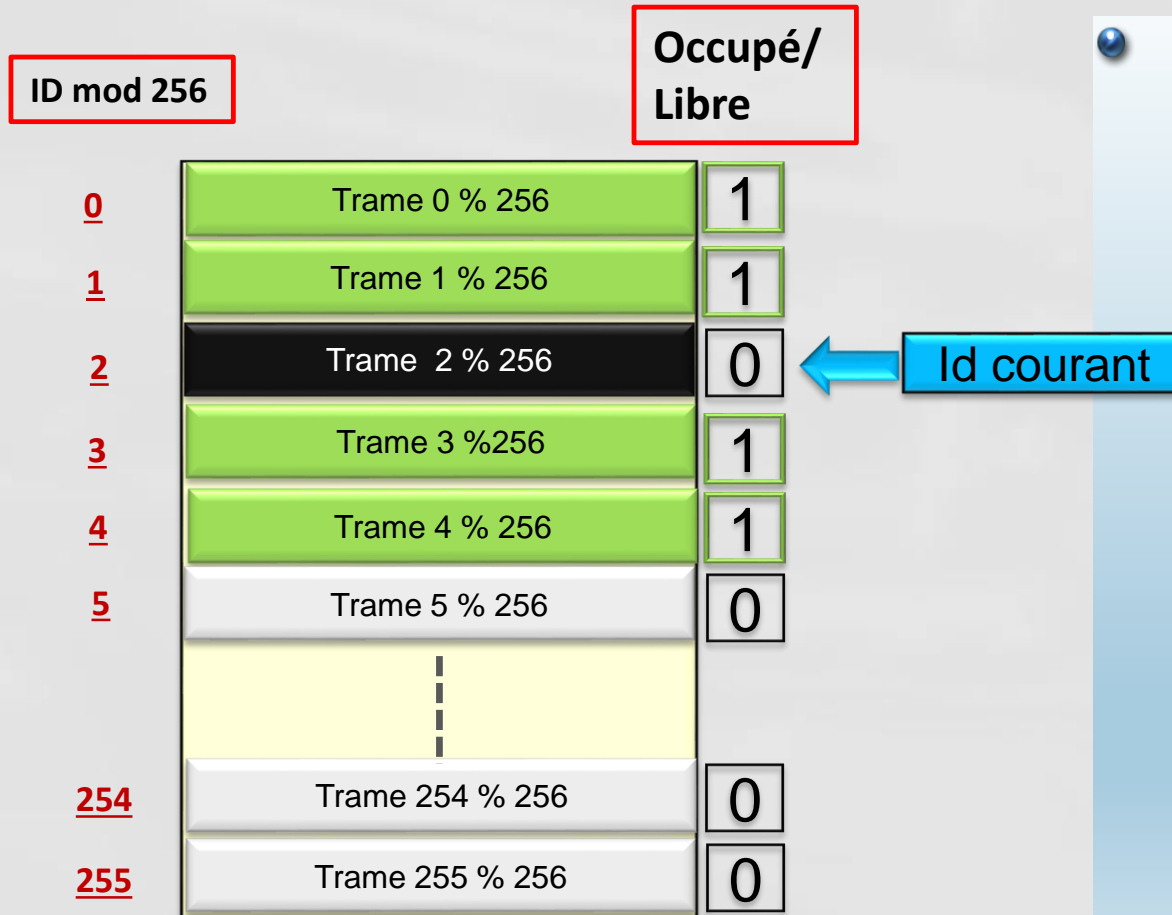
On arrête d'envoyer de nouvelles trames!

Fifo contenant les indices des buffers '**disponibles**'

Fifo contenant les indices des buffers arrivés en '**timeout**' et dont les données doivent être renvoyées

Comment réordonne-t-on les trames?

- Au niveau Software (**intégrée dans la librairie libudp**)



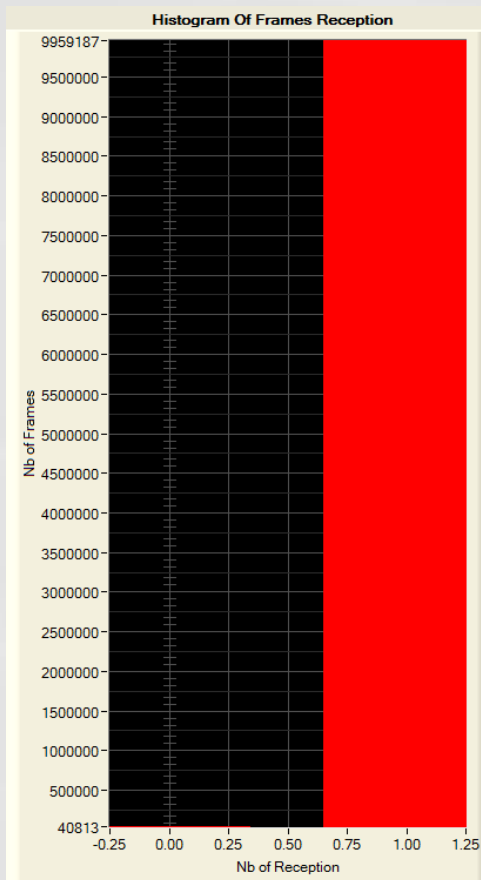
- La fonction de lecture sécurisée :

- **Acquitte** toutes les trames reçues
- Les trames **dupliquées** ne sont pas ré-écrites dans le buffer.
- Chaque trame est sauvegardée dans le buffer en fonction de son **id % 256**
- Ne « donne » en sortie que les **trames ordonnées**

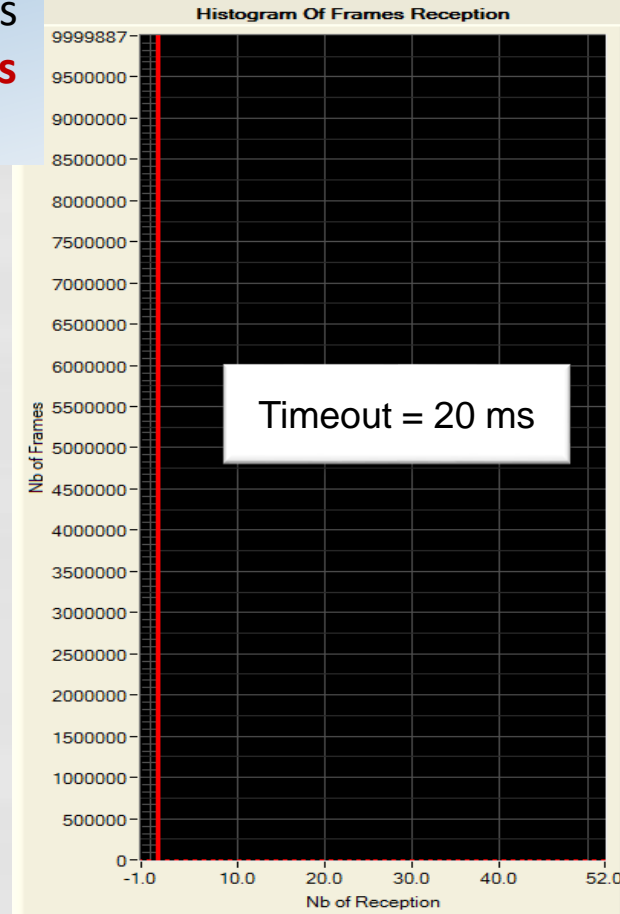
Mémoire Tampon Software
Taille = 256 x Taille Max d'une trame attendue

Tests et Résultats (1/2)

- **10 millions de trames** envoyées
- Taille des paquets : **3200 Octets**
- Délai entre trames **100 ns**

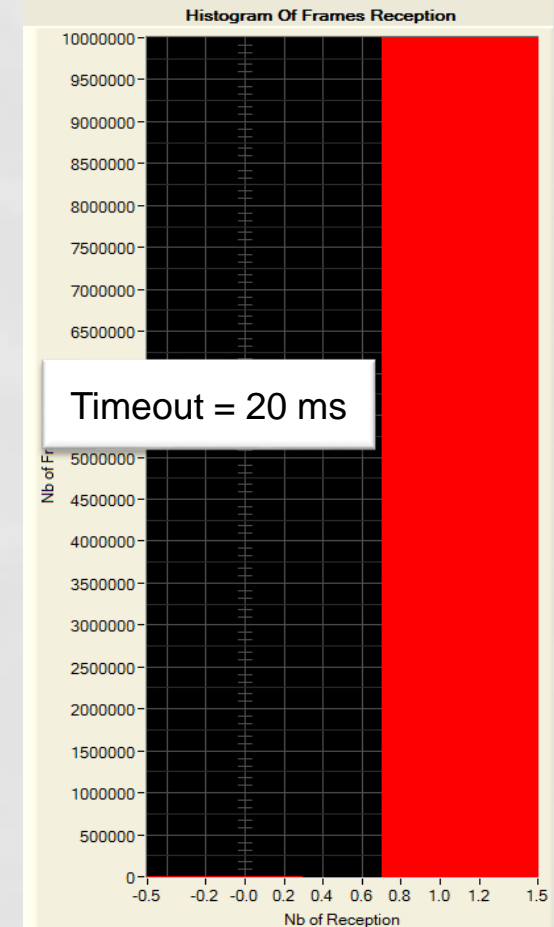


Cas 1 : Pas de 'Hand Check'
Vitesse Moyenne ~ **700 Mbits/s**
Trames Perdus ~ 40000 ~ 0,4%



Cas 2: 'Hand Check' ON

- On **ne réordonne pas** les trames et **on ne gère pas** les trames **dupliquées**.
- Vitesse moyenne = **680 Mbits/s**
- Trames reçues jusqu'à 52 fois (PC en pause pendant 1 s??) !

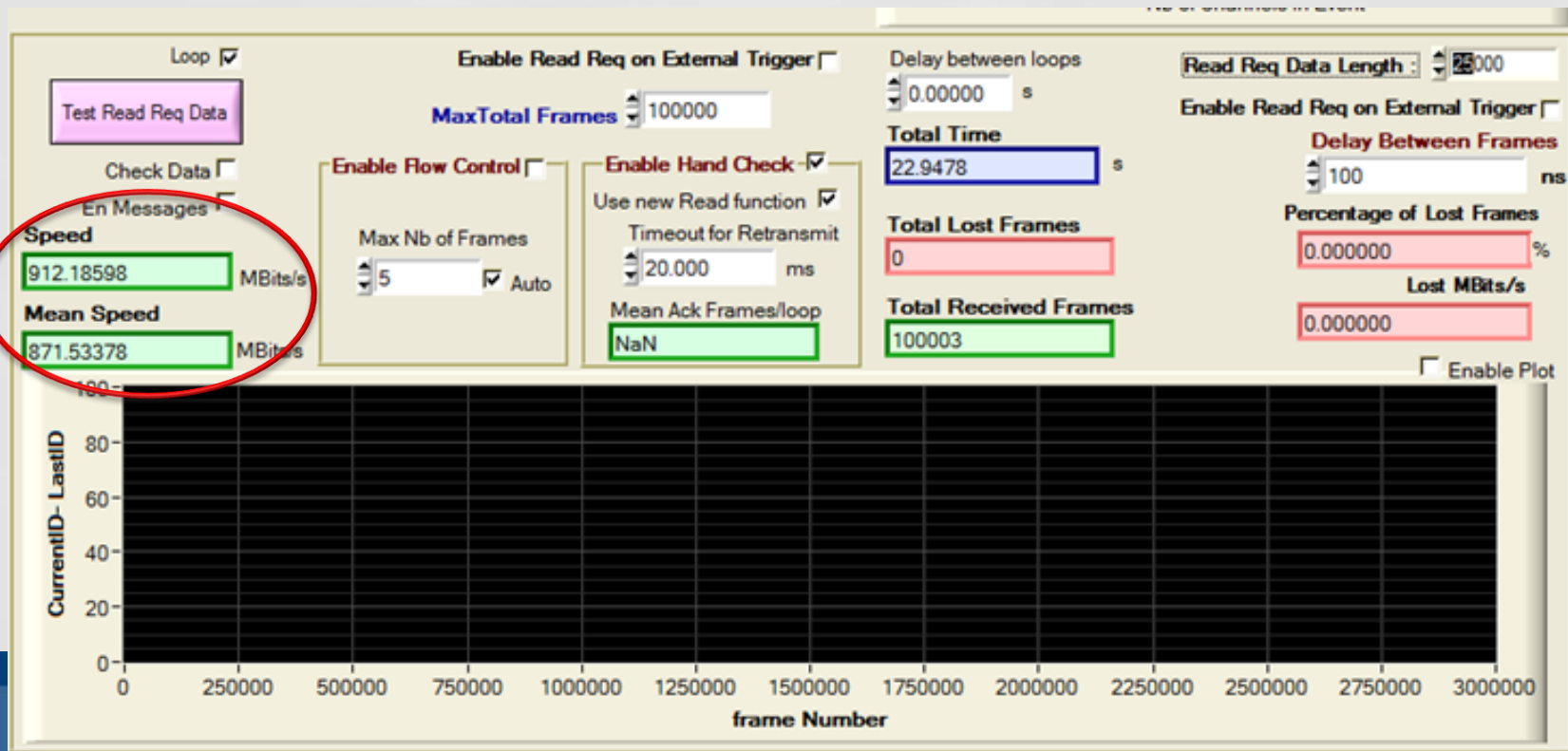


Cas 3: 'Hand Check' ON

- On utilise la fonction qui ordonne les paquets et gère les trames dupliquées
- Vitesse moyenne = **680 Mbits/s**

Tests et Résultats (2/2)

- Avec des trames de de taille proche de 32kOctets on approche le gigabit par seconde (**900 Mbit/s**)
- Au-delà, à cause de la taille de la RAM tampon dans le FPGA (65KOctets) : il n'y a plus qu'un seul buffer disponible → l'acquittement et l'envoi d'une nouvelle trame sont donc sérialisés
 - Par ex, pour des trames de 40kOctets, on passe de **900 à 560 Mbits/s**
 - Pour des trames de **50kOctets** on tourne à **690 Mbits/s**



Conclusion

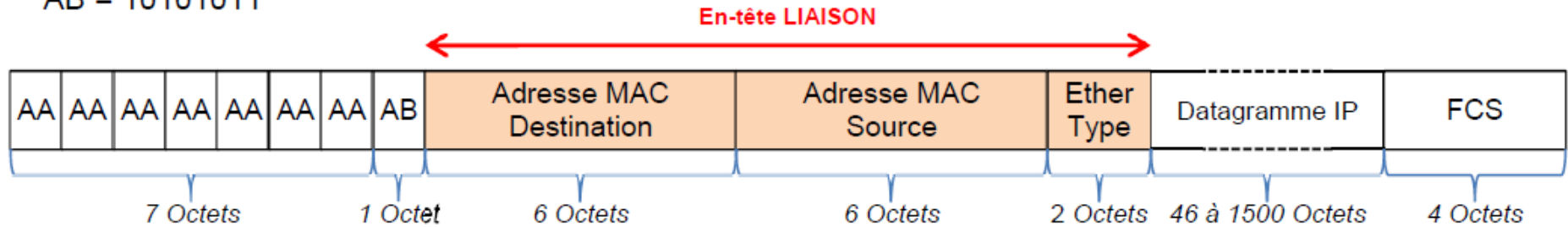
- L'interface UDP a été développée dans le cadre de l'expérience SuperNemo et pour d'autres projets maison : elle est déjà utilisée en test beam au CERN.
- Il s'agit d'une interface qui gère les contraintes réseau minimum nécessaires
- L'interface de base standard peut être contenue dans moins de 2200 éléments logiques dans un FPGA bas coût (type Cyclone III ou IV-E) + des blocs mémoires (1 Fifo 4096 Octets + 2 Fifos 1500 Octets)
 - On peut gérer à minima le flux en mettant du délai entre les trames
- L'interface **UDP sécurisée** utilise en tout ~ **6200 éléments logiques** + Un bloc mémoire **65kOctets** (modifiable) + les Fifos ci-dessus → entièrement écrite en Verilog + instantiation PLL, Fifo et RAM d'Altera.
- L'interface côté utilisateur est très simple :
 - **(data (8 bits), subadd(7 bits) n_read, n_write, n_wait, read_req, interrupt)**
- Une **bibliothèque dédiée** a été développée qui permet une utilisation très facile d'accès.
- L'interface sécurisée a été testée sur un run de 13h en acquisition réelle de données de la carte Calorimètre de SuperNemo sans perte de donnée.
- D'autres tests vont être effectués dans le cadre de SuperNemo : histogramme des timestamps des trames, tests avec plusieurs cartes connectées à travers le même switch, etc...
- Possibilité de **sécuriser la partie CTRL** d'une manière très simple par envoi d'un acquittement.

Backup Slides

Trame Ethernet II

AA = 10101010

AB = 10101011



Préambule : (7 octets) Permet la synchronisation des horloges de transmission. Il s'agit d'une suite de 1 et de 0 soit 7 octets à la valeur 0xAA

SFD : (1 octets) "Starting Frame Delimiter". Il s'agit d'un octet à la valeur 0xAB. Il doit être reçu en entier pour valider le début de la trame.

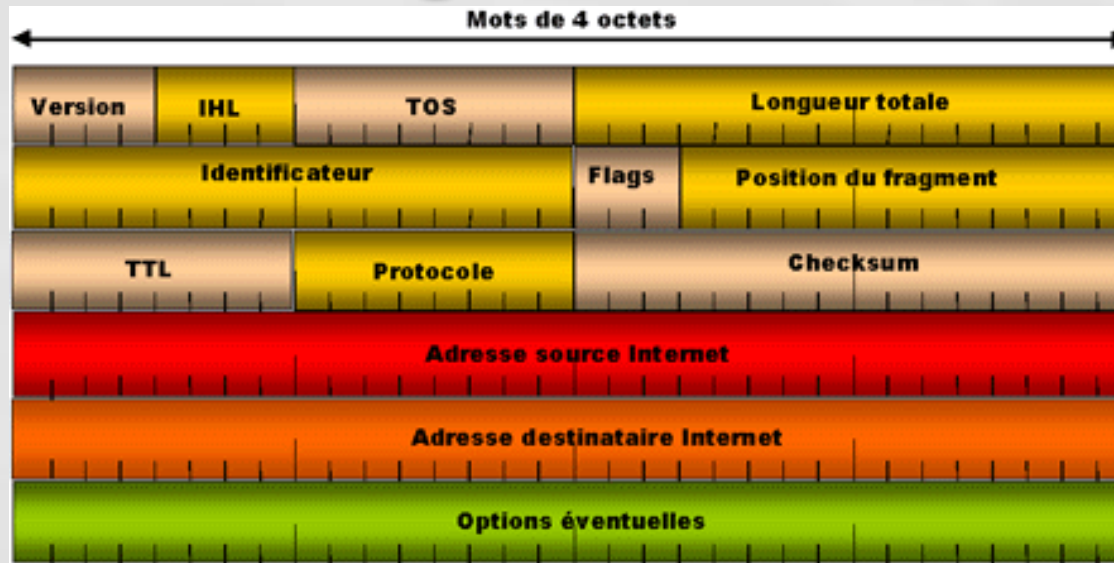
En-tête : (14 octets) - Adresse MAC du destinataire (6 octets)
 - Adresse MAC de l'émetteur (6 octets)
 - EtherType (Type de protocole) (2 octets)

Exemples de valeurs du champ EtherType →

FCS : (4 octets) Frame Check Sequence.
 Ensemble d'octets permettant de vérifier que la réception s'est effectuée sans erreur.

EtherType	Protocole
0x0800	IPv4
0x0806	ARP
0x809B	AppleTalk
0x8035	RARP
0x86DD	IPv6

Datagramme IP



Longueur totale du Header IP = 20 Octets si pas d'options

Longueur totale : (16 bits) Longueur du datagramme entier y compris en-tête et données mesurée en octets.

Identificateur : (16 bits) Valeur assignée par l'émetteur pour identifier les fragments d'un même datagramme.

Flags : (3 bits) Commutateurs de contrôle :

- Bit 0 Réserve, doit être laissé à 0
- Bit 1 (DF - Don't fragment) 0= Fragmenté 1= Non fragmenté
- Bit 2 (MF – More Fragment) 0= Dernier fragment 1= Fragment

OFFSET : (13 bits) Décalage du premier octet du fragment par rapport au datagramme complet non fragmenté. Cette position est mesurée en blocs de 8 octets (64 bits).

Durée de vie : (8 bits) Temps en secondes pendant lequel le datagramme doit rester dans le réseau.

Protocole : (8 bits) Protocole porté par le datagramme (au-dessus de la couche IP): ICMP, TCP, UDP etc..

Comment passer 1Gbit/s en half-duplex à 125 MHz sur 4 paires (1000_baseT) ???

Contrairement aux 10 et 100 Mbits/s, le Gbit Ethernet 1000-baseT (pour « Twin » => paires différentielles) n'utilise pas des niveaux logiques mais 5 niveaux analogiques pour le codage des données sur chaque paire (250 Mbits/s par paire).

⇒ c'est pourquoi un **hardware spécifique** est nécessaire (DACs et buffers différentiels pour l'émission et ADCs pour la réception)

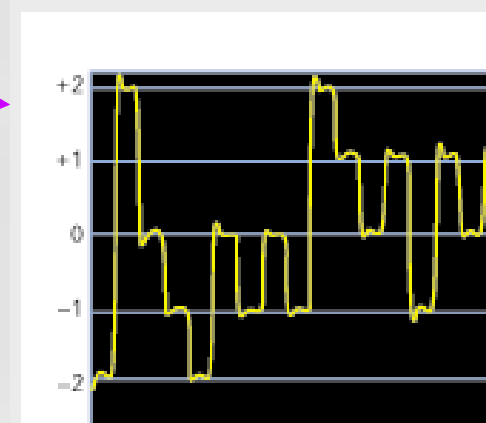
⇒ Ce codage est appelé « PAM5 » :

2 bits (4 valeurs) plus 1 code de correction d'erreur (FEC)

(pour info : 10 Gbits => PAM17 !)

⇒ **Avantage** : il rend le réseau 100 Mbits/s utilisable pour le Gbit/s car la densité spectrale du signal est comparable !

⇒ Conséquence : complexité (cachée) et phase « d'accrochage » ...



⇒ Pour cette raison, les FPGAs n'incluent pas d'interface 1000-baseT, uniquement des 1000-baseX (SERDES 1 Gbit/s) dédiées aux fibres optiques => besoin d'un PHY 1000-baseT externe pour communiquer directement avec un switch ou un PC !

ARP: Protocole de Résolution d'Adresse

Rôle très important:

il permet de connaître l'**adresse physique (MAC address)** d'une carte réseau (ou nos cartes en l'occurrence) correspondant à une adresse IP donnée.

La machine émettrice envoie **une requête à tout le monde** (adresse broadcast : FF:FF:FF:FF:FF:FF). La machine/Carte qui **reconnaît son adresse IP** peut renvoyer une réponse ARP en utilisant l'adresse source inscrite dans la requête comme adresse de destination.

Exemple d'une trame ARP interceptée par WireShark:

15 5... IntelCor_3f:7d:75 Broadcast ARP 42 Who has 192.168.0.254? Tell 192.168.0.5

