Description ensemble IPbus

O. Bourrion

CNRS-IN2P3-LPSC Grenoble

22 octobre 2014

Plan

Plan

Introduction I

Original

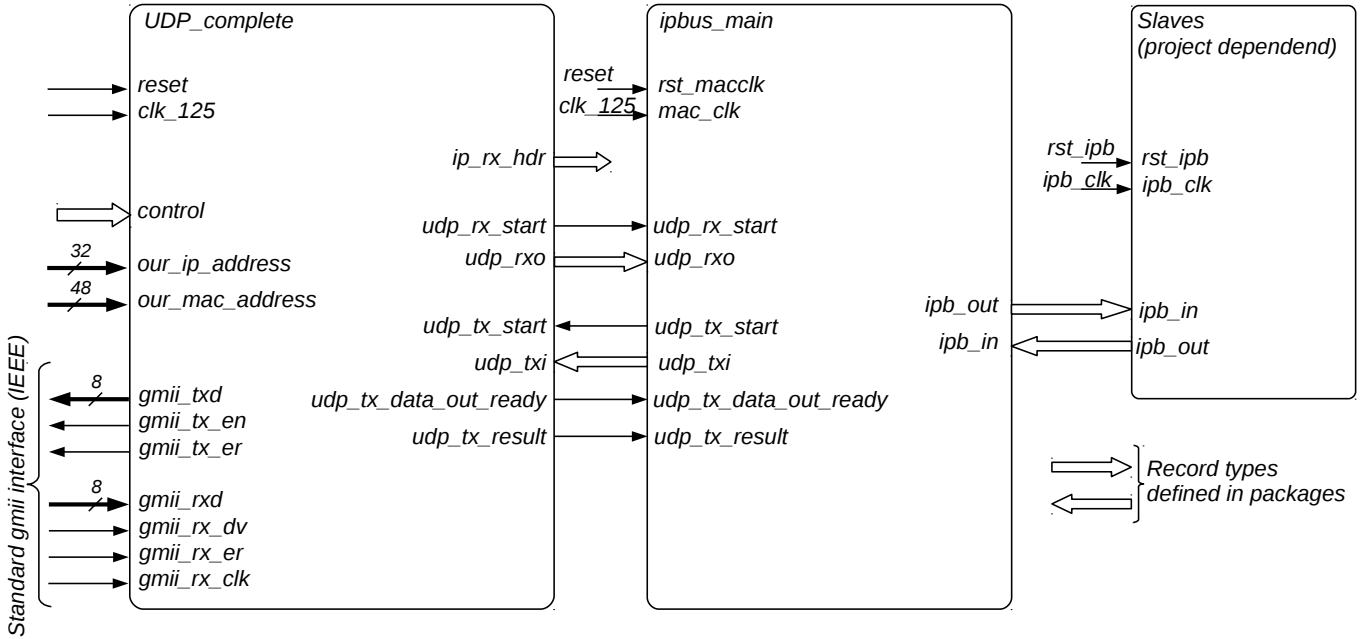
- Protocole spécifié par le CERN, voir documentation au lien suivant https://svnweb.cern.ch/trac/cactus/browser/trunk/doc
- Il est basé sur UDP avec un suivi des paquets et la possibilité de redemander un paquet perdu.
- Permet d'avoir d'un coté Ethernet et de l'autre un bus A32/D32
- Le CERN fournit un firmware et un logiciel de contrôle voir https://svnweb.cern.ch/trac/cactus/wiki
- Impossibilité de séparer la partie UDP de la partie IPBus (design imbriqués)

Introduction II

Version LPSC

- Le firmware fourni permet la simulation au niveau Ethernet
- Brique client DHCP disponible
- Suppression de la FIFO AXI de réception et de la MAC générées par coregen, remplacé par description VHDL basée sur RAM inferring
- Devient facilement portable sur une autre plateforme (ALTERA)
- Un driver logiciel sous Qt existe (multiplatforme)

Vue générale



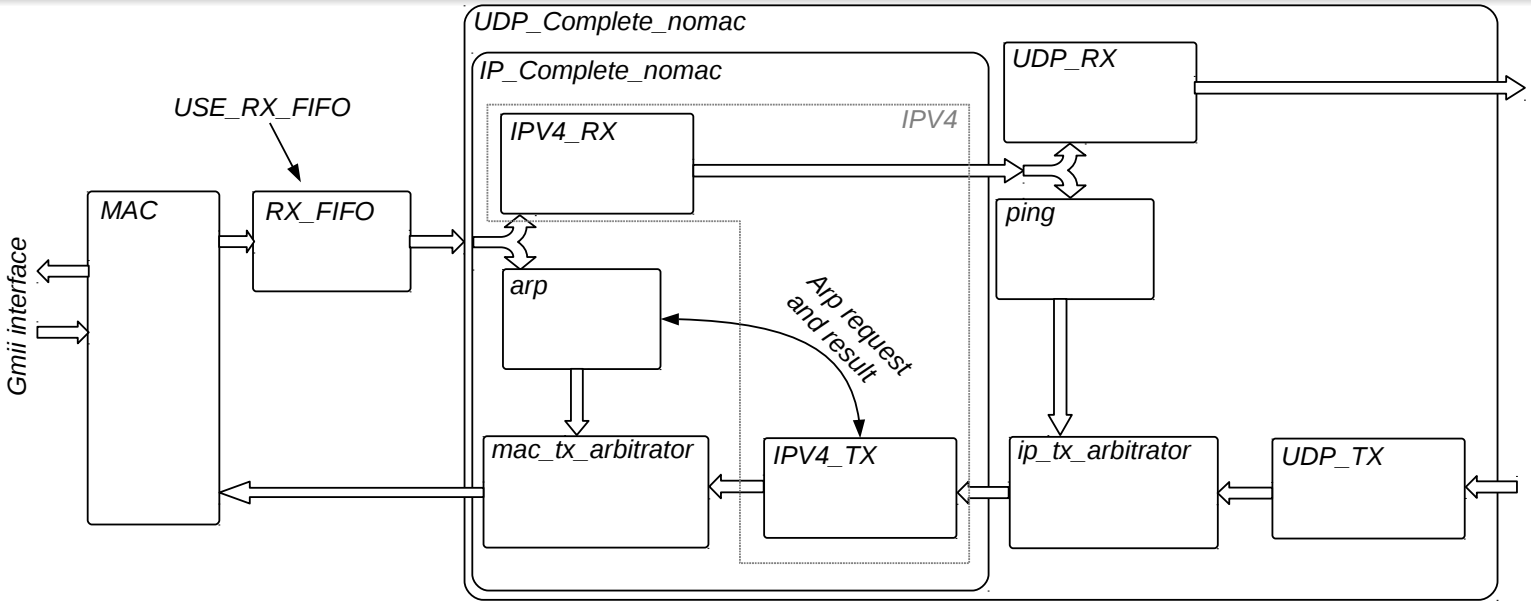
- Bloc UDP_complete et ipbus_main à paramétrer avec *generic*
- Interface vers Ethernet via gmii (norme IEEE section 2). Soit un PHY externe, soit un lien série rapide.
- Génération d'horloge/reset non représenté.
- les types *record* définis dans les différents *packages*.
- Le bloc *slaves* contient tous les éléments esclaves du projet. Il est différent pour chaque projet.

Plan

Description du bloc UDP

- Ce bloc est paramétrable :
  - USE_RX_FIFO : A activer quand l'horloge de réception est différente de l'horloge système ;
  - IN_SIMULATION : A activer en simulation (démarrage plus rapide et désactive l'autonegociation)
  - SIMUL_DEST_DEST_NUMBER : Pour simuler numéro MAC distant et desactiver recherche ARP en simulation.
  - CLOCK_FREQ : Fréquence horloge système (125 MHz), pour régler timer
  - ARP_TIMEOUT : Temps d'attente maximum (1 min)
  - MAX_ARP_ENTRIES : nombre maximum d'adresse MAC stockées
- Cette version peut répondre à une requête ARP mais aussi en être initiatrice.
- Fonctionne sur le modèle en couche (voir diagramme)
- Le mode autonegociation est activé
- Basé sur un bloc *opencore* qui a été ajusté, voir http://opencores.org/project,udp_ip_stack

Bloc diagramme



- MAC : Gère l'accès au média et génère/vérifie les checksum
- RX_FIFO est facultative (occupe 1 BRAM), voir paramétrage
- ARP :
  - Le bloc arp répond aux requêtes si l'IP du bloc est demandée
  - Lorsqu'il répond, il stocke l'adresse MAC du demandeur dans sa table
  - Si IPv4_TX veut envoyer un paquet → demande d'adresse MAC
  - Si IP connue, réponse immédiate, sinon requête ARP sur le réseau
- arbitrators : gèrent les accès aux TX, priorité au flux de donnée
- ping : fonctionne jusqu'à la taille maxi Ethernet (1492 octets)

Limitations

MAC

- ne gère pas les collisions (nécessaire en half-duplex) ni les pauses.

Checksum

Les blocs ci-dessous ne génèrent/calculent pas de checksum pour ne pas ajouter de latence. Ce n'est pas imposé par la norme. La MAC assure déjà une bonne émission/réception.

- IPV4_RX
- UDP_RX
- UDP_TX (champ forcé à zéro $\rightarrow$ OK en IPV4)

Liste des fichiers

Couche MAC

pack_MAC.vhd, crc_gen.vhd, tri_mode_eth_mac.vhd, MAC_RX_engine.vhd, MAC_TX_engine.vhd, mac_fifo_axi4_lpsc.vhd

Couche IP

pack_ipv4_types.vhd, pack_arp_types.vhd, pack_axi.vhd, IP_complete_nomac.vhd, arp.vhd, IPv4.vhd IPv4_RX.vhd, IPv4_RX.vhd, IP_tx_arbitrator.vhd, mac_tx_arbitrator.vhd

Couche UDP

pack_simu_udp.vhd (pour la simulation !), ping.vhd, UDP_Complete.vhd, UDP_Complete_nomac.vhd, UDP_TX.vhd, UDP_RX.vhd

Interfaçage UDP (1/2)

- Prévu pour fonctionner à 125 MHz
- Bus *gmii* vers couche physique
- Signal *control* sert à effacer table arp

Bus de réception UDP

- udp_rx_start : indique une réception de trame UDP
- udp_rxo (description dans pack_ipv4_types.vhd et pack_axi.vhd) :
  - Contient les informations source et destination (IP, ports)
  - Contient la taille du paquet UDP en octets
  - Le transfert de données est au format AXI

Bus d'émission UDP

- udp_tx_start : indique une émission de trame UDP
- udp_tx_result : indique le succès/échec de l'émission de la trame
- udp_txi (description dans pack_ipv4_types.vhd et pack_axi.vhd) :
  - Contient les informations source et destination (IP, ports)
  - Contient la taille du paquet UDP en octets
  - Le transfert de données est au format AXI

Interfaçage UDP (2/2)

protocole AXI

- data : bus de données 8 bit
- data_valid : donnée disponible
- data_ready : Émission prête à accepter des données
- data_last : Dernière donnée de la trame

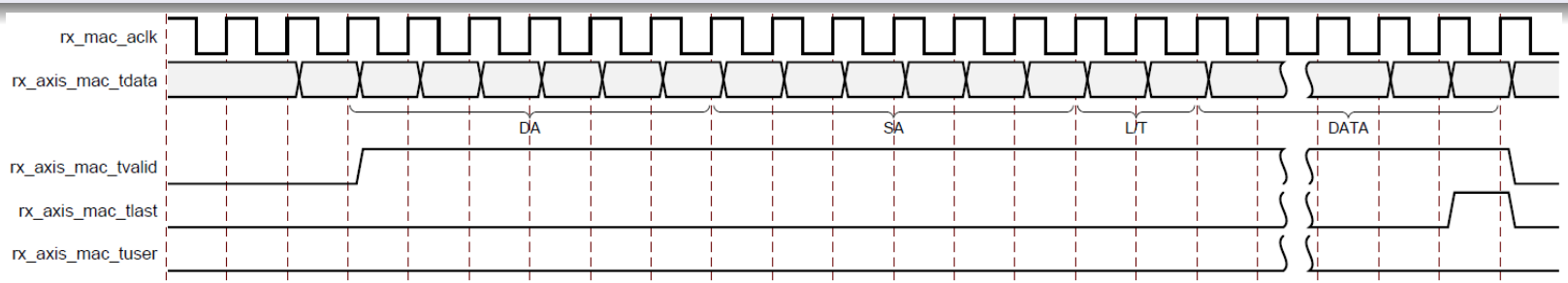


Figure 7-1: **Normal Frame Reception**

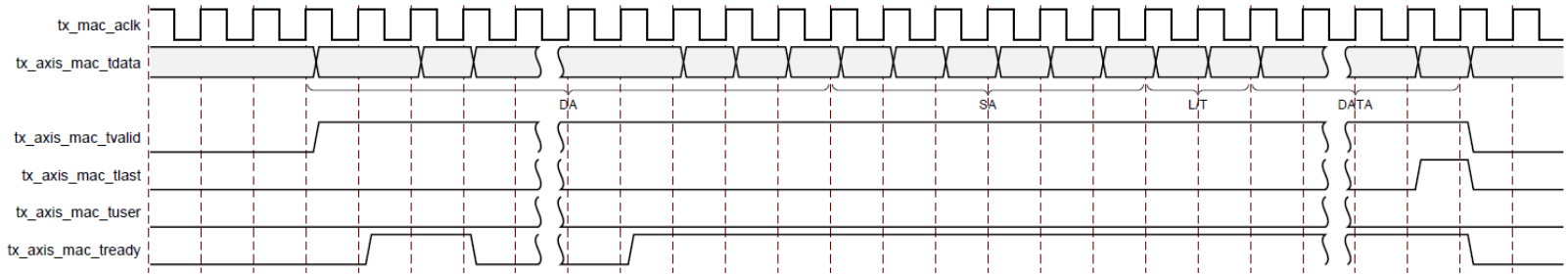


Figure 7-7: **Normal Frame Transmission Across User Interface**

Simulation

Par la couche Ethernet

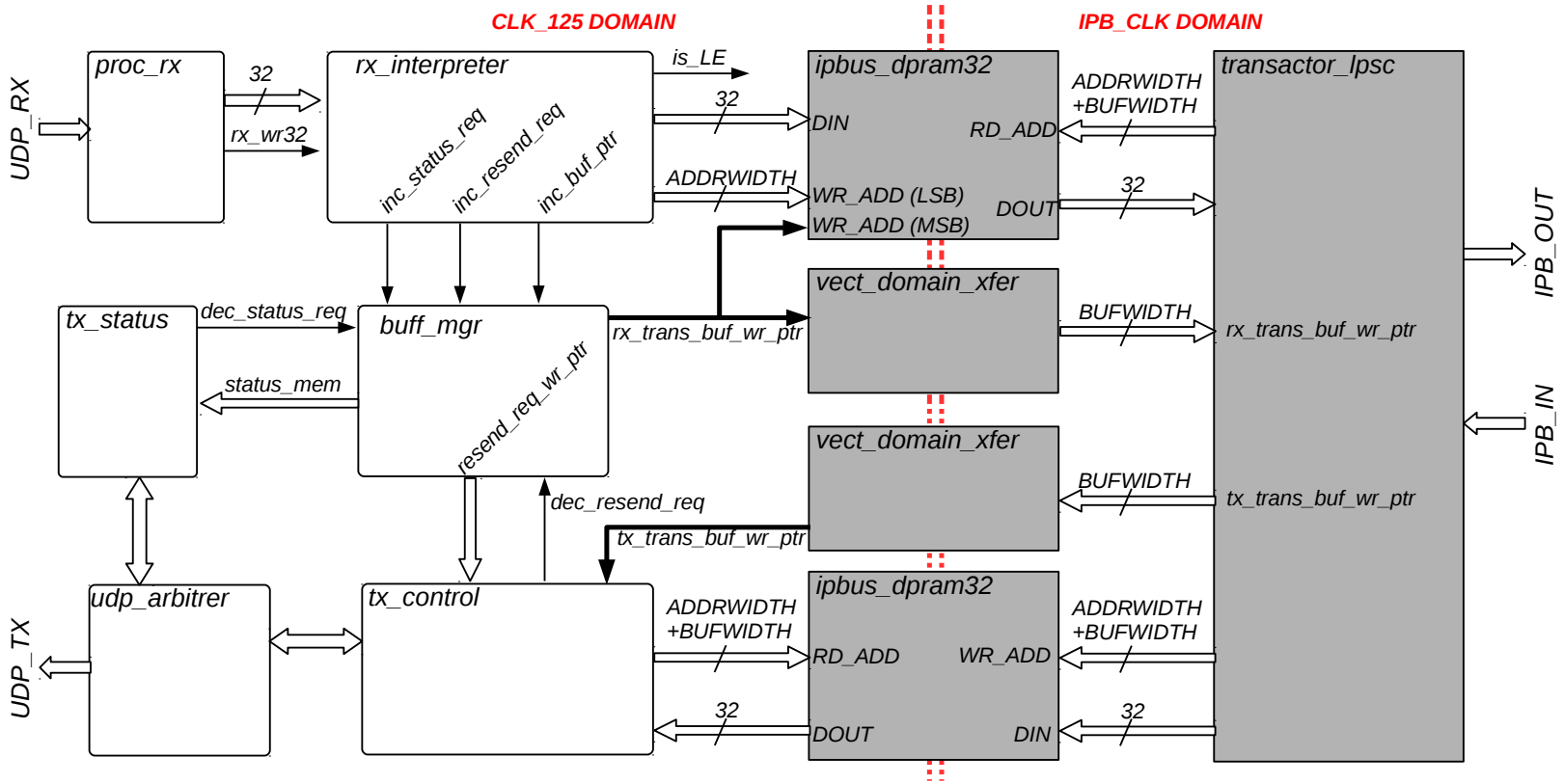
- Le testbench tb_top_lpsc_extphy permet de simuler la partie UDP.
- les données sont lues et écrites via l'interface gmii.
- Utilise des trames de référence enregistrées avec wireshark (contiennent donc les bonnes checksums) :
  - requête ARP
  - requête ping (teste aussi la couche IPv4)
  - requête IPbus Status (trame UDP donc)
  - requête IPbus contrôle (trame UDP donc)

Au niveau UDP

- Le package pack_simu_udp.vhd prend un buffer de WORD32 de taille N+1 où :
  - La case 0 contient la taille du paquet UDP en octets
  - Le reste contient la trame de taille N à transférer
- le testbench de simulation, consiste à instancier un module UDP validé (cf ci-dessus) pour l'utiliser en émetteur. C'est à dire on rentre une trame UDP, il produit une trame Ethernet.

Plan

Bloc diagramme



Paramètres *generic* :

- BUFWIDTH : nombre de buffer pour le streaming (2^{BUFWIDTH})
- ADDRWIDTH : taille maxi d'un buffer en word 32 ($2^{\text{ADDRWIDTH}}$), une taille de 512 words permet un MTU de 2040 octet
- IPBUSPORT : port UDP utilisé pour la transaction IPbus

Descriptif de chaque bloc I

proc_rx

- conversion des données reçues en 8 bit en mot de 32 bit

rx_interpreter

- Détermine si paquet IPBus (via le port adressé)
- Détermine endianness pour faire le même type de réponse
- Détermine le type de paquet IPbus (control, status, resend)

ipbus_dpram32

- Bloc mémoire d'interface entre domaine d'horloge Ethernet et IPbus
- Stocke une requête complète ainsi que son endianness
- La 1ere case mémoire stocke la longueur du paquet et son endianness
- doit être implémenté dans les blocs mémoire dédiés

Descriptif de chaque bloc II

vect_domain_xfer

- Module d'interface d'un bus entre 2 domaines d'horloge
- Utilisé pour passé le numéro de buffer pointé vers/depuis le transactor_lpsc
- Le passage de domaine se fait en code gray

transactor_lpsc

- C'est le module qui effectue les transaction IPbus en décodant la trame reçue
- Tous les types de transaction prévu (à l'exception de ceux touchant l'espace de configuration) sont supportés
- Ce module commence à travailler si :
  - Le pointeur de buffer local est différent du pointeur fourni par buff_mgr
  - Le numéro de paquet est nul ou celui attendu

Descriptif de chaque bloc III

tx_control

- Prend le contenu d'un buffer prêt à être envoyé (dans un ipbus_dpram32) et le transfert vers l'UDP
- ou si une rémission est demandée va chercher si un paquet avec le numéro correspondant existe encore et le transfert le cas échéant

buff_mgr

- Mémorise les différents type de requête décodées par rx_interpreter
- Décrémente les requêtes à chaque traitement réussi

tx_status

- Sur requête de status renvoie l'état de la cible comme spécifié dans la norme à l'exception des "traffic history", champs laissé à zéro

Descriptif de chaque bloc IV

tx_status

- Sur requête de status renvoie l'état de la cible comme spécifié dans la norme à l'exception des "traffic history", champs laissé à zéro

udp_arbitrer

- Arbitre l'accès à l'émetteur UDP, priorité est donnée aux données

Limitations

- Ne gère qu'un seul client (IP du demandeur non stocké dans les buffers permettant le streaming) à la fois

Protocole IPBus I

- Protocole adresse/donnée 32 bit (lecture/écriture)
- Signal de *strobe* pour valider la transaction
- Signal *ack* pour finir le cycle $\rightarrow$ possibilité de ralentir les accès (jusqu'à 255 clk)
- possibilité pour l'esclave de signal une erreur via *err*
- Durée minimum d'un cycle IPBus : 2 ipb_clk
- Pour une même transaction, pas de temps mort pour changer d'adresse
- Au changement de transaction, 1 cycle perdu
- Au changement de paquet, plusieurs cycles perdus

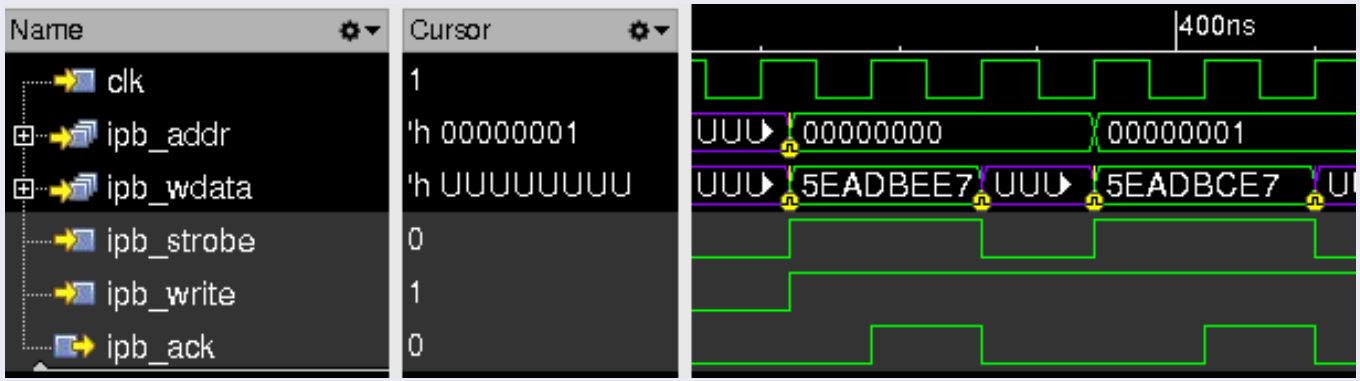
Protocole IPBus II

Déclaration bus (dans ipbus_package.vhd)

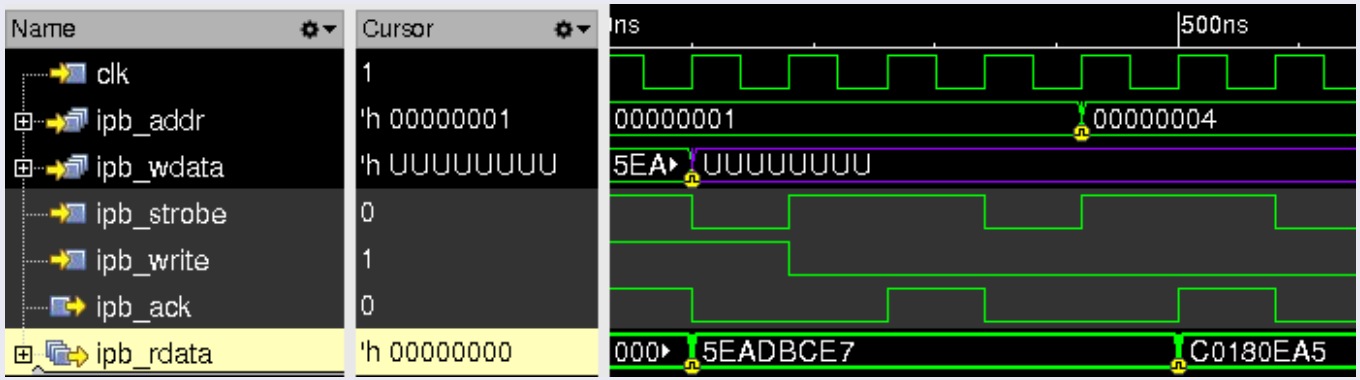
```
type ipb_wbus is
record
    ipb_addr: std_logic_vector(31 downto 0);
    ipb_wdata: std_logic_vector(31 downto 0);
    ipb_strobe: std_logic;
    ipb_write: std_logic;
end record;

type ipb_rbus is
record
  ipb_rdata: std_logic_vector(31 downto 0);
  ipb_ack: std_logic;
  ipb_err: std_logic;
end record;
```

Ecriture simple

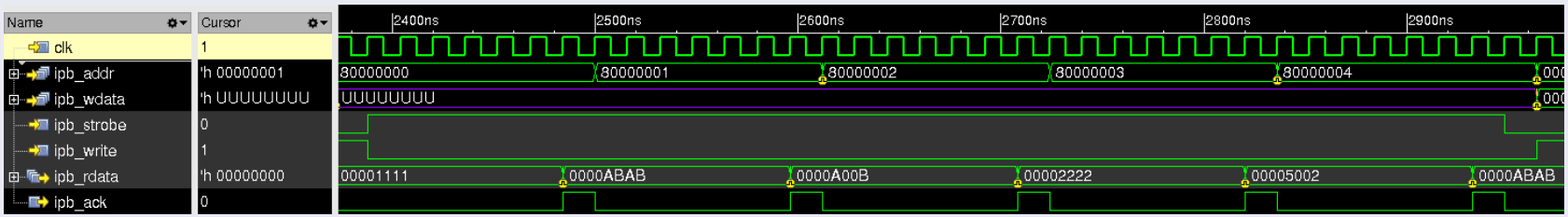


Lecture simple



Protocole IPBus IV

Lecture multiple, ralentie par ack



Interconnexion esclaves

- Créer une matrice de ipb_wbus et ipb_rbus
- Utiliser ipbus_fabric.vhd pour décoder le bus issu du transactor_lpsc.vhd pour faire les bus à envoyer vers chaque esclave
- le décodage se fait en 2 phases :
  - La partie haute pour une sélection de zone dans ipbus_fabric.vhd
  - la partie basse (le niveau registre) dans chaque esclave (voir esclaves exemples)
- Le décodage poids fort est réglé dans le package/fichier ipbus_addr_decode.vhd
- Pour alléger le plus possible le décodage tester le moins de bit possible

Exemple de table d'adressage

```
if    std_match(addr, "1-------------------------------") then
    sel := 3; -- flash / base 0x80000000 / mask 0x80000000
elsif std_match(addr, "0-----------------00----0000----") then
    sel := 0; -- ctrl_reg / base 0x00000000 / mask 0x800030f0
elsif std_match(addr, "0-----------------00----0001----") then
    sel := 1; -- thres / base 0x00000010 / mask 0x800030f0
elsif std_match(addr, "0-----------------01------------") then
    sel := 2; -- ram / base 0x00001000 / mask 0x80003100
elsif std_match(addr, "0-----------------00----0010----") then
    sel := 4; -- cfdthres / base 0x00000020 / mask 0x800030f0


elsif std_match(addr, "0-----------------10------------") then
    sel := 11; -- adc_serial / base 0x00002000 / mask 0x80003100
else
    sel := 99
```

Liste des fichiers

IPbus maître

ipbus_package.vhd, transactor_lpsc.vhd, vect_domain_xfer.vhd, ipbus_dpram32.vhd, ipbus_main.vhd

IPbus esclave

ipbus_addr_decode.vhd, ipbus_fabric.vhd
Et vos fichiers !

Simulation IPbus : pour simuler tout type de transactions

pack_simu_transactor_lpsc.vhd

Simulation

- Possibilité de simuler sans la couche Ethernet pour accélérer temps de simulation
- Il faut créer un composant contenant tous les esclaves et le décodeur d'adresse
- le composant *top* utilisant les esclaves ne doit ajouter que la couche UDP et l'interface Ethernet
- Utiliser les procédure du package de simulation pour faire les transactions désirées

Plan

Synthèse et PAR

Synthèse

- Aller dans le répertoire *synthesis*
- Fichiers de paramétrage *.prj pour *synplify_premier*
  exemple *synplify_premier -batch top_ipbus_lpsc_basex.prj*
- La netlist EDIF générée est utilisée en entrée de ISE

Placement routage

- Dans le répertoire PAR, en fonction de la cible choisie
- Faire source *PAR_FPGA.sh &* et suivre l'avancement avec *tail -f route.log*

Plan

Logiciel

Une bibliothèque logicielle Qt existe, elle peut-être compilée dans votre projet ou liée si vous créez un *.so ou un *.dll

Méthodes publiques de la classe TIPbusClient.h

```
void connectClient(QString addStr, int port);
void read_std(int add, quint32 *buff,int size);
void read_ni(int add, quint32 *buff,int size);
void rmw_bit(int add, quint32 *buff,quint32 ANDterm,quint32 ORterm);
void rmw_sum(int add, quint32 *buff,quint32 A);
void write_std(int add, quint32 *buff,int size);
void write_ni(int add, quint32 *buff,int size);
void setTimeout(int val);
void dispatch();
```

- Une méthode de connection à la cible
- 6 méthodes pour créer des transactions
- Une méthode pour forcer l'envoi d'un paquet IPBus même si il n'est pas plein

Plan

Documents utiles

- LogiCORE IP Tri-Mode Ethernet MAC user guide (ug777)
- Opencore de la MAC : http://opencores.org/project,ethernet_tri_mode
- IEEE standard for Ethernet section one (802.3-2012) : Spécification de la couche MAC
- IEEE standard for Ethernet section two (802.3-2012) : Spécification de l'interface gmii
- Opencore pour la couche UDP : http://opencores.org/project,udp_ip_stack
- Dépot SVN https://lpsc-secure.in2p3.fr/svn/akido/Ethernet/ (login :guest, passwd : guest)
- Obtention d'une plage d'adresse MAC par le CNRS http://www.mrct.cnrs.fr/spip.php?article21