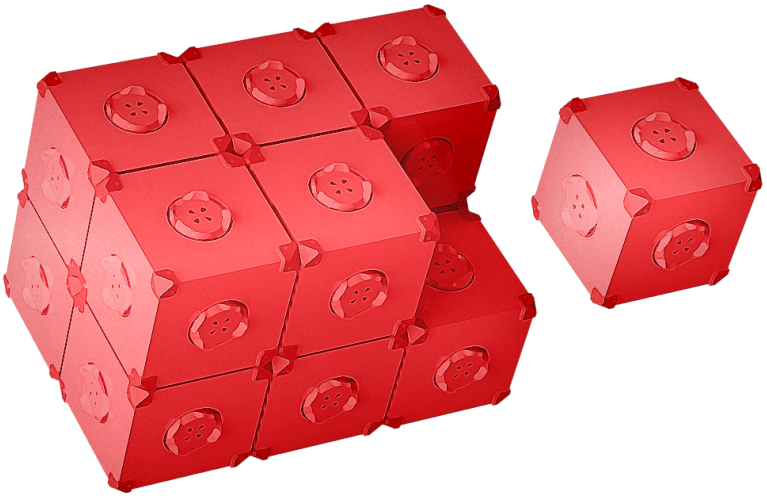
Université de Bretagne Occidentale

Rapport de Projet

Framework de simulation “SpaceWire”

Le Boudec Alan
Master 2 LSE



UFR Sciences et Techniques
Département d’Informatique
20 avenue Victor le Gorgeu
29238 Brest

Sommaire

Introduction

La plupart des personnes aiment observer le ciel mais le ciel aussi nous observe. 2 630 satellites sont actuellement en orbite autour de la Terre, lancés pour la première fois en 1967 avec un satellite expérimental, leur nombre ne cesse de s'accroître chaque année.

Comme dans beaucoup de domaines de l'informatique il est important d'avoir un système performant, robuste, fiable et configurable. Ce sont des enjeux d'autant plus importants dans le cas des satellites.
La communication est l'un de ses enjeux, c'est l'un des points cruciaux de tous types de systèmes, une bonne communication permet de bons résultats.

Cette communication doit permettre d'être rapide et efficace mais surtout fiable, puisque dans l'espace on ne peut envoyer un réparateur pour gérer d'éventuels problèmes. C'est pour répondre à cet enjeu qu'a été mise en place la communication SpaceWire, qui répond à ses exigences mais aussi qui permet une totale configuration du système. Comment fonctionne-t-il ? Comment est-il configurable ? Quel est le temps de communication ?

L'objet du projet est de tester le système SpaceWire, de comprendre son fonctionnement et de le mesurer.
Ce rapport présente le travail que j'ai effectué depuis le début de l'année scolaire. Je vais expliquer en premier lieu mes objectifs, donné par mon encadrant. Ensuite je vais montrer sur quel environnement j'ai travaillé et les différentes installations nécessaires. Par la suite, j'aborderai les documents mis à ma disposition. Puis je vais décrire les programmes pour tester le système. Enfin je commenterai les résultats et les améliorations possibles puis je terminerai sur un bilan de mon projet, la réponse à ma problématique, les objectifs réussis, les compétences acquises et l'apport du projet à mon projet professionnel.

Objectifs

Ce projet a pour mission principale de mettre en place le réseau SpaceWire. Durant le projet j'avais donc cet objectif principal comme ligne directrice entouré de différentes missions pour le bon fonctionnement de SpaceWire.

Tout d'abord je devais faire un état de l'art de l'ECSS 12-C un standard qui fournit les normes, sur SpaceWire. Ensuite j'ai fait la même chose avec les fichiers sources à utiliser pour le projet.

Après l'analyse des documents, la première grande étape était de tester le premier composant, le CODECIP. Je devais faire fonctionner le codec et par la suite faire la liaison Point-Point.
La seconde grande étape était de mettre en place le réseau complet avec des codecs et le deuxième composant le ROUTERIP.

Une fois le système testé et fonctionnement, je devais prendre des mesures des temps de communication dans le réseau avec différentes configurations.

SpaceWire Définition

Dans cette partie on va faire une synthèse du standard ECSS 12-C, sur le fonctionnement de SpaceWire. Certains points vont être parcourus sans rentrer dans les détails comme les times codes ou encore les tickets. Puisque dans le cas de mon travail on n'utilise pas certaines de ces fonctionnalités.
De plus, on expliquera aussi le code source ainsi que le fonctionnement des composants utilisés.

ECSS

SpaceWire est un réseau déployé sur des systèmes embarqués spatiaux, Il permet de connecter des instruments de mesure, des processeurs, des mémoires etc.
Dans ce réseau, la communication est assurée par des liens SpaceWire spécifiques, Data et Strobe. Sur ces liens vont transiter des paquets qui seront acheminés par des routeurs et des SpaceWire ports.

Ces paquets sont soumis à un protocole de transfert entre l'interface et la partie matérielle. Ils transitent ainsi sur le système par des liens SpaceWire et arrivent à destination grâce au routeur.

Tout d'abord on va voir comment est fait un paquet et comment il est transité, ainsi que les différentes couches du protocole d'envoi de messages. Ensuite on verra le fonctionnement d'un routeur SpaceWire.

Paquet SpaceWire

Un paquet est divisé en 3 parties, la première contient le chemin où doit aller le paquet, la deuxième contient la donnée et enfin la dernière contenant le EEP ou EOP (end of packet ou error end of packet).

figure 1.1 - Constitution d'un paquet

L'adresse peut être de 2 types soit de type "path adressing", c'est-à-dire un chemin qui contient plusieurs "portes" ou de type direct qui contient juste une porte.
Par exemple si l'on veut que notre paquet soit envoyer sur le port 4 puis 6 puis 22 on aura : {4,6,22}{Cargo}{EOP}

De plus cette “porte” peut être soit de type logique soit de type physique. Les ports physiques sont les 32 ports physiquement rattachés aux routeurs. Le type physique est un entier entre 0 et 31 et le type logique est un entier entre 32 et 254.
Un paquet est soumis à un protocole sur plusieurs couches entre l’interface et la partie matérielle. Il y a 4 couches différentes :

- Network layer
- Datalink layer
- Encoding layer
- Physical layer

C’est 4 couches sont accessibles par une interface de gestion, c’est le management layer.

Network layer

Permet de gérer les paquets , les interruptions et les times codes (signaux de synchronisation).

Datalink layer

Gère les N-Chars(les 8 bits de données) , les broadcasts(times code + interruptions)
contrôle le flot d'informations sur les liens

Encoding layer

Encode les caractères et les controls codes(EOP,EEP,ESC et FCT) en symbole puis les sérialise. Les symboles sont encodés en signal Data et Strobe. Il fait aussi le travail inverse le décodage.

Physicial layer

Transmet les signaux data et strobe au support physique et récupère les signaux du support.

Management layer

Récupère les statuts de chaque couche et les monitores.

Routeur SpaceWire

un routeur SpaceWire ou “routing switch” connecte un ensemble de nœuds à l’aide de liens SpaceWire. Chaque nœud pourra communiquer avec les autres à l’aide de paquets, ainsi les nœuds seront des sources ou destinations des paquets. Le routeur va permettre cet échange grâce à des ports SpaceWire, une table de routage qui va enregistrer les chemins connus des nœuds (c’est-à-dire sur quels ports physiques ils se trouvent) et aussi un arbitre qui va organiser l'accès à la table de routage.

La table de routage est tout simplement une mémoire qui va contenir tous les chemins des nœuds. Si la demande est de savoir où envoyer le paquet, cela se fera sous la forme d'une lecture à une certaine adresse pour savoir sur quel port envoyer le paquet.

Cette demande est autorisée par l'arbitre, il va choisir qui aura accès à la table de routage et va jouer le rôle d'ordonnanceur.

Le code source

Dans le cadre du projet on utilise des codes sources disponibles sur github et écrit par Shimafuji Electric Inc. Le code est open source et est disponible à cette adresse https://github.com/shimafujigit. Le code comprend 2 composants le CODECIP et le ROUTERIP, On va détailler ensemble ces 2 parties cependant on ne verra pas les times codes, les interruptions et les tickets puisqu'on n'utilisera pas .

Tout d'abord on va travailler sur le CODECIP qui permet l'encodage et le décodage de données sur 9 bit en signaux Data et Strobe . Il se présente comme ci-dessous :

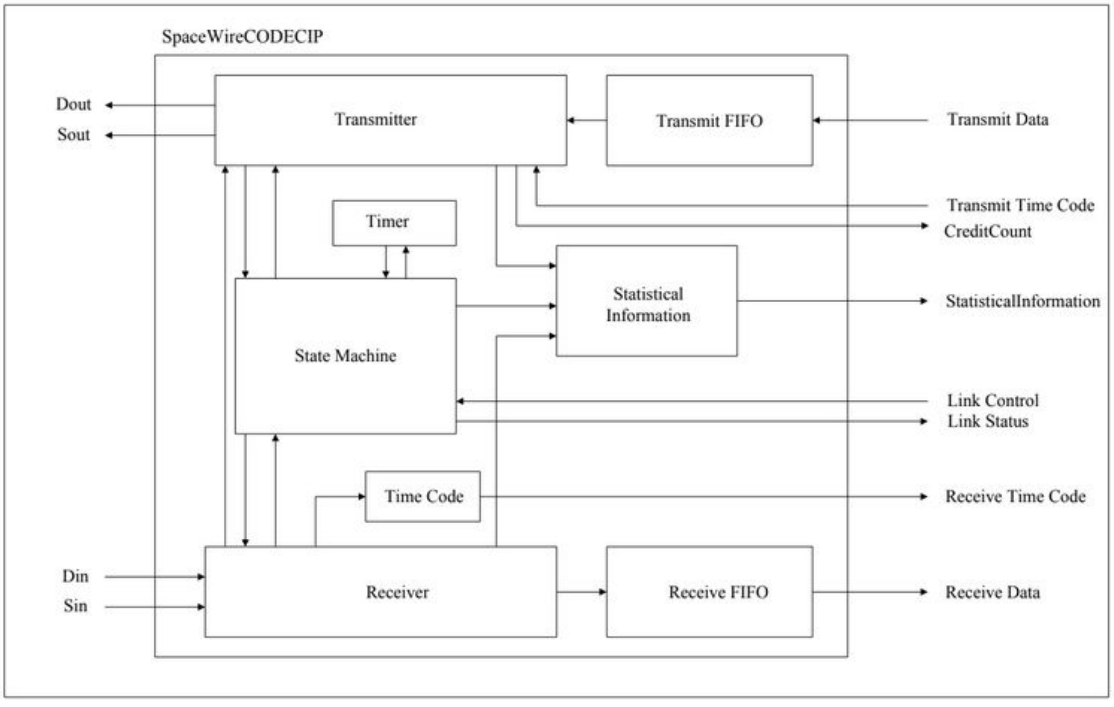


figure 1.2 - Fonctionnement du CODECIP

Il y a 3 parties importantes sur ce schéma. La partie transmit, la partie receive et la partie state machine. Statistical information permet de récupérer une bonne partie de toutes les infos de chaque bloc, mais on va travailler directement avec les informations qui sont données en sortie de bloc.

Transmit

Il va encoder les N-Char(un char étant une data de 9 bit) d'entré en signaux Data(Dout) et Strobe(Sout). Les signaux Data et Strobe sont des signaux codés sur 1 bit.

Receive

Décodage des signaux Data et Strobe en N-Char.

State Machine

Cela permet de savoir à tout moment l'état du CODEC. linkStatus nous indiquera son état et linkControl permet de configurer l'état. Le CODEC suit un diagramme d'état précis défini dans l'ECSS :

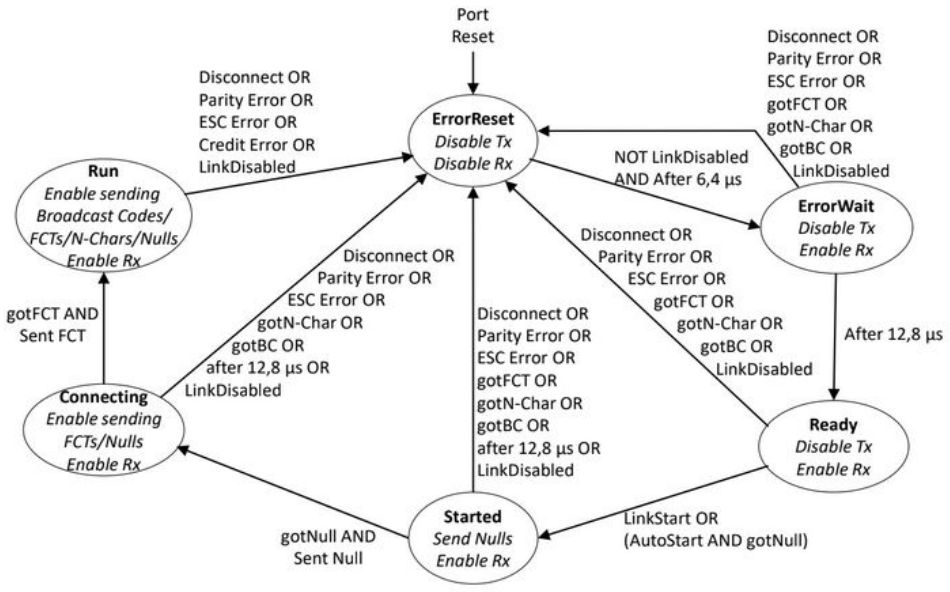


figure 1.3 - Diagramme d'état de l'initialisation d'un lien

L'état de départ est l'ErrorReset, après 6,4 us on passe dans l'état ErrorWait. Dans l'état ErroWait on va activer la réception de données.12,8 us plus tard on passe dans l'état Ready.
Si le linkStart a bien été activé on passe dans l'état Started. Dans cet état, on va échanger des messages "null". Les messages "null" sont des contrôles codes sous la forme de ESC + FCT.

Une fois réceptionné on passe dans l'état Connecting. Dans cet état on va envoyer des Fct et en recevoir pour pouvoir passer dans l'état run et envoyer des N-Chars.
Dans chaque état, si il y a une erreur on repasse dans l'état initial soit ErrorReset.

voici le CODEC avec les signaux que j'ai manipulé :

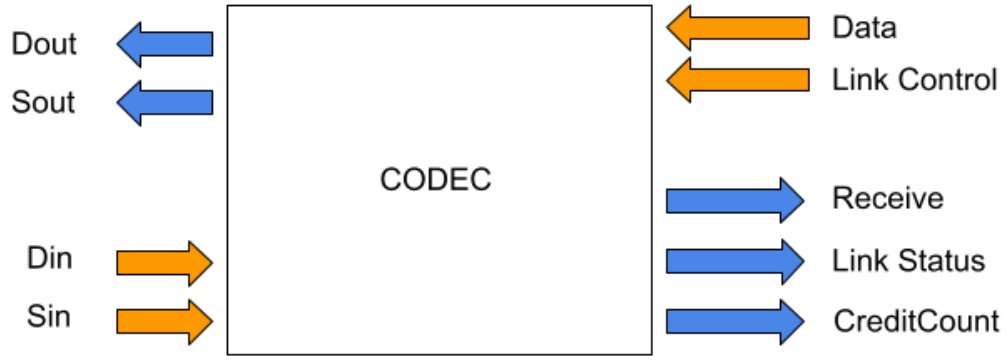


figure 1.4 - Codec simplifié

Dout pour Data out et Din pour Data in de même pour Strobe. En entrée La Data sera notre donnée codé sur 9 bits et link control nous permet d'activer la liaison.
En sortie nous avons reçu la donnée, le statut du codec et le Crédit count qu'on utilisera pas.

Si l'on regarde du côté du code :

```
entity SpaceWireCODECIP is
    port (
        clock                       : in  std_logic;
        transmitClock               : in  std_logic;
        receiveClock                : in  std_logic;
        reset                       : in  std_logic;
--
        transmitFIFOWriteEnable     : in  std_logic;
        transmitFIFODataIn          : in  std_logic_vector(8 downto 0);
        transmitFIFOFull            : out std_logic;
        transmitFIFODataCount       : out std_logic_vector(5 downto 0);
        receiveFIFOReadEnable       : in  std_logic;
        receiveFIFODataOut          : out std_logic_vector(8 downto 0);
        receiveFIFOFull             : out std_logic;
        receiveFIFOEmpty            : out std_logic;
        receiveFIFODataCount        : out std_logic_vector(5 downto 0);
--
        tickIn                      : in  std_logic;
        timeIn                      : in  std_logic_vector(5 downto 0);
        controlFlagsIn              : in  std_logic_vector(1 downto 0);
        tickOut                     : out std_logic;
        timeOut                     : out std_logic_vector(5 downto 0);
        controlFlagsOut             : out std_logic_vector(1 downto 0);
        linkStart                   : in  std_logic;
        linkDisable                 : in  std_logic;
        autoStart                   : in  std_logic;
        linkStatus                  : out std_logic_vector(15 downto 0);
        errorStatus                 : out std_logic_vector(7 downto 0);
        transmitClockDivideValue    : in  std_logic_vector(5 downto 0);
        creditCount                 : out std_logic_vector(5 downto 0);
        outstandingCount            : out std_logic_vector(5 downto 0);
--
        transmitActivity            : out std_logic;
        receiveActivity             : out std_logic;
--
        spaceWireDataOut            : out std_logic;
        spaceWireStrobeOut          : out std_logic;
        spaceWireDataIn             : in  std_logic;
        spaceWireStrobeIn           : in  std_logic;
--
        statisticalInformationClear : in  std_logic;
        statisticalInformation      : out bit32X8Array

        );
end SpaceWireCODECIP;
```

figure 1.5 - Signaux du Codec

Voici l'entité du CODEC, tout d'abord en 1 nous avons les signaux pour la clock et il y en a 3. Ces 3 signaux sont pour la clock générale, la clock de transmission et la clock de réception.

En 2 on a les signaux d'entrée et sortie de la transmission et de la réception. Dans le cas de la transmission il y a 4 signaux :

- transmitFIFOWriteEnable : demande d'écriture
- transmitFIFODataIn : la donnée
- transmitFIFOFull : Buffer d'envoi plein
- transmitFIFODataCount : compteur de N-Char

Pour la réception :

- receiveFIFOReadEnable : demande de lecture
- receiveFIFODataOut : donnée en sortie
- receiveFIFOFull : buffer de réception plein
- receiveFIFOEmpty : buffer de réception vide
- receiveFIFODataCount : compteur de réception

Dans le bloc 3 ce sont les signaux de gestion du statut du CODEC, c'est à dire :

- linkStart : permet de démarrer la connexion
- linkDisable : désactive la connexion
- autoStart : démarre la connexion mais attend à une étape intermédiaire.
  on attendre dans l'état Ready (figure 1.3) jusqu'à recevoir un message "null". Cela permet d'attendre pour une certaine condition avant de lancer la connexion
- linkStatus : statut du CODEC
- errorStatus : statut erreur du CODEC
-

Le bloc 4 contient 2 signaux transmitActivity et receiveAcitivity, ils indiquent si le CODEC est en train de transmettre une donnée ou alors de recevoir.

Enfin le bloc 5 contient les 4 signaux Data et Strobe d'entrée et sortie.

On a à notre disposition un deuxième composant le ROUTERIP SpaceWire.
Voici comment est schématisé le ROUTERIP :

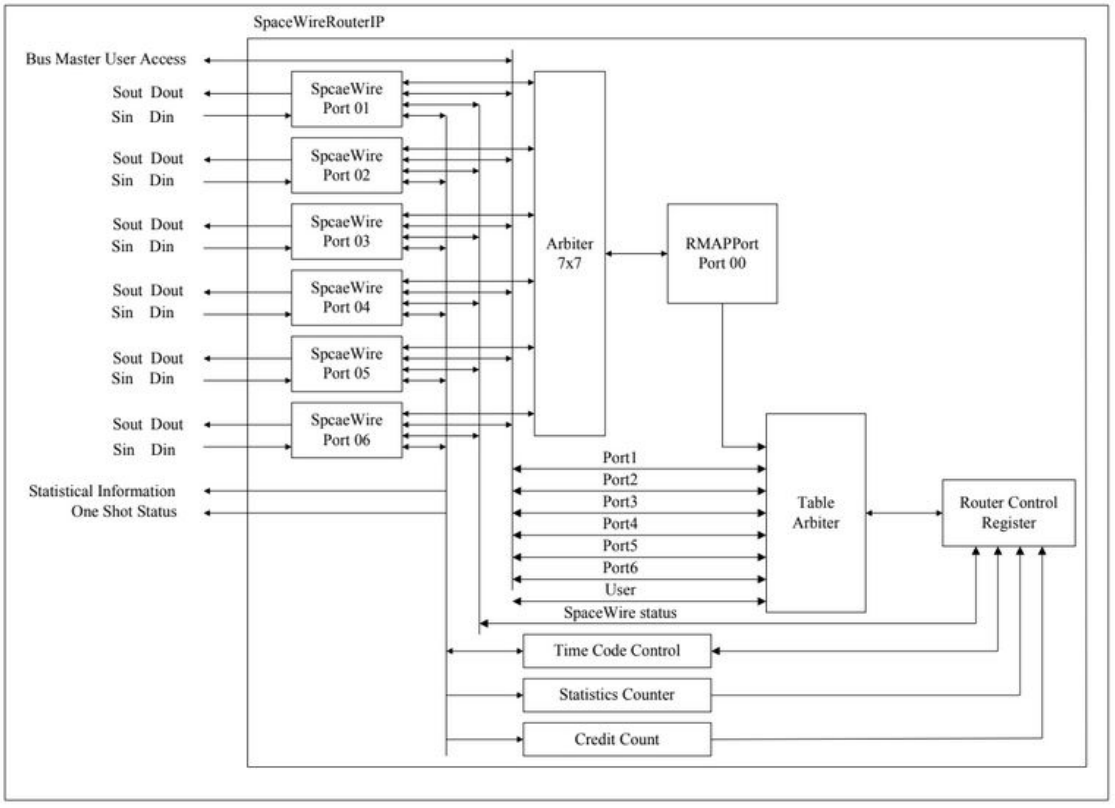


figure 1.6 - Fonctionnement du ROUTERIP

En premier lieu on peut remarquer qu'il n'y a que 6 ports physiques, ceci à été choisi par le programmeur. Cependant on peut très bien en mettre 32 le programmeur a justement écrit les modifications à apporter pour cela dans le code du programme.

En second lieu on s'aperçoit qu'il y a 2 arbitres, un arbiter 7*7 et une table arbiter. Il s'agit bien de 2 arbitres distincts avec des fonctions différentes, le premier va gérer l'accès au port de configuration et le second à la table de routage.
Le premier arbitre va permettre l'accès au port de configuration, effectivement lorsqu'un paquet va être envoyé sur le routeur il va passer par le port 0 afin d'être décodé.
C'est-à-dire qu'il va décrypter les différentes parties d'un paquet dont le header, la data mais aussi sur quel registre l'orienté.
Le port 0 est un port par défaut dans une routing switch qui permet la configuration de la table de routage par le routeur. Dans ce code, il fait aussi office de décodeur et d'aiguilleur des paquets venants des ports.

En haut du schéma il y a un Bus Master User permettant la configuration du Router Control Register par l'utilisateur. On peut aussi récupérer des informations comme le status d'un port, les crédits count, les time...etc.

On va seulement manipuler le Bus Master User Acess pour la configuration de la table de routage.

Router Control Register

Le Router Control Register est la table de routage qui va permettre aux paquets de retrouver le port physique de destination. Cette table de routage est une mémoire, un tableau de 256 lignes et de 32 colonnes. 32 colonnes puisqu'on peut au total avoir 32 ports physiques, pour rappel ici on utilise que 6 ports donc ça sera seulement les 6 premiers bits qui nous intéresserons. De plus il y a 256 lignes pour les 254 adresses possibles logiques et physiques.Les 2 lignes sont utilisées pour d'autres fonctions du programme qui n'ont pas de lien avec l'adressage.

Maintenant on va regarder comment cela fonctionne dans le code. Il y a 2 dossiers Altera et Xilinx, ce sont eux qui fournissent la mémoire pour la table de routage. Cependant ,les deux fichiers ont été générés, comme Xilinx par l'IDE du même nom.
On n'utilisera pas ces fichiers, la raison est simple: on ne peut les compiler avec ghdl. Par défaut dans le programme c'est la ram Xilinx32*256 qui est utilisé. Afin de remplacer ce composanton va créer une mémoire, ça sera une simple mémoire synchrone avec gestion d'un tableau que l'on décrira dans la prochaine section.

On s'aperçoit qu'il y a beaucoup de composants si l'on regarde bien bien plus que ce qu'il peut y avoir sur le schéma mais certains sont des composants d'autres composants, je vais donc rendre ça plus clair.

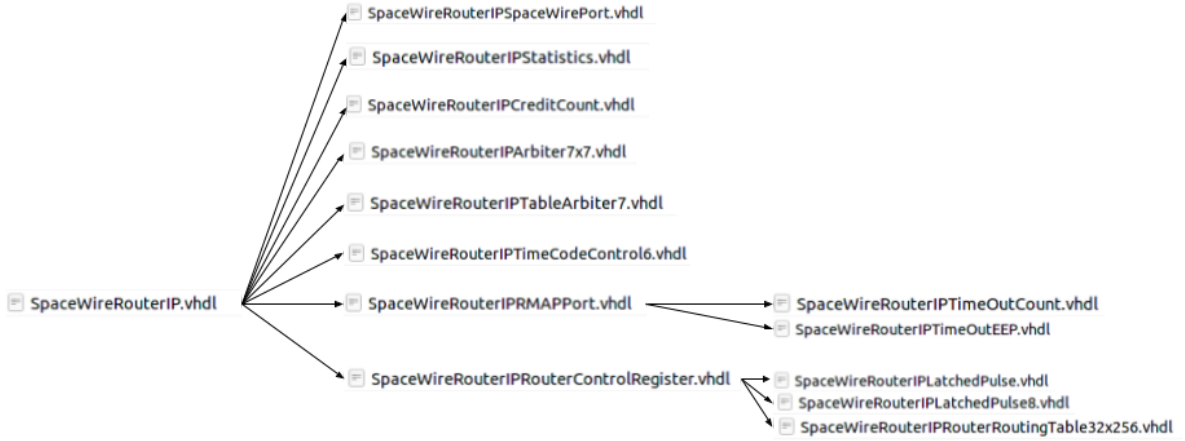


figure 1.7 - Arborescence des fichiers sources

Voici l'arborescence des fichiers, ceux qui vont nous intéresser sont le SpaceWireRouterIP et le SpaceWireRouterIPRouterRoutingTable32*256.
Le premier étant le composant principal et le second celui qui contient la table de routage.

Le fonctionnement de ce routeur correspond à celui décrit dans l'ECSS surtout au niveau du comportement.
Voici comment est organisé la mémoire :

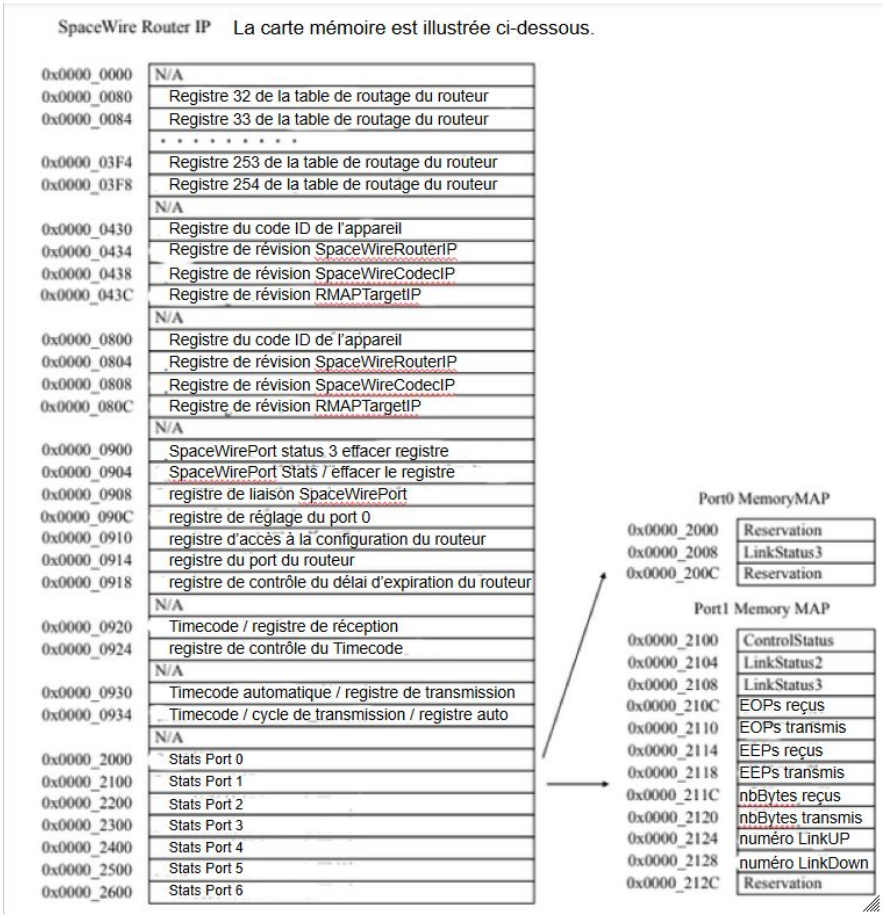


figure 1.8 - Ogranisation de la mémoire

Sur le schéma la table de routage est comprise entre 0x0000_0080 et 0x0000_03F8 ou 128 et 1016 en décimal. Ceci est l'intervalle disponible entre les adresses logiques de 32 à 254 que l'on peut retrouver par le calcul.
1016-128 = 888, sachant que comme on peut le voir on avance de 4 en 4.
donc 888/4 cela nous fait 222 valeurs disponibles, si l'on ajoute nos 32 adresses physique ça nous fait bien les 254 valeurs.

La partie table de routage est séparé du reste d'un point de vue mémoire, c'est-à-dire qu'elle est gérée de manière autonome et dans un tableau dédié à cette fonction.

Voyons tout ça par un exemple :

On veut que notre nœud 56 soit sur le port 4.
Pour cela on va vouloir écrire que l'adresse logique 56 lui soit associée l'adresse physique 4.

l'adresse pour écrire sera à : 56-32 = 24
24 * 4 = 96
128 + 96 = 224 en décimal
Soit 0x0000_0E0 en hexadécimal

A cette adresse on va mettre la valeur du bit 5 à 1 pour signifier que le port physique 4 lui est attaché (plusieurs ports peuvent y être attachés).
Il faut mettre le cinquième bit à 1 ce qui fait 16 en décimal.

Ainsi on va écrire 16 à l'adresse 224.

Maintenant que l'on sait comment fonctionne la table de routage, voyons comment la configurer avec le Bus Master User.

```
busMasterUserAddressIn      : in  std_logic_vector (31 downto 0);
busMasterUserDataOut        : out std_logic_vector (31 downto 0);
busMasterUserDataIn         : in  std_logic_vector (31 downto 0);
busMasterUserWriteEnableIn  : in  std_logic;
busMasterUserByteEnableIn   : in  std_logic_vector (3 downto 0);
busMasterUserStrobeIn       : in  std_logic;
busMasterUserRequestIn      : in  std_logic;
busMasterUserAcknowledgeOut : out std_logic
```

figure 1.9 - Signaux Bus Master User

busMasterAdressIn est l'adresse à indiquer et busMasterUserDataIn la donnée.
busMasterDataOut est la valeur de sortie à l'adresse demandée.
busMasterUserWriteEnable est la demande d'écriture ou non.
busMasterUserByteEnable est pour choisir quelle partie de la donnée à lire par bloc de 8 bits.
busMasterUserStrobe et busMasterUserRequestIn permettent de faire la demande d'accès au Router Control Register.
Si la demande est acceptée on reçoit un busMasterUserAcknowledgeOut à 1

Programmes

Avant de mettre en place le système SpaceWire on va d'abord tester le composant CODECIP et l'échange de données.
Chaque programme ci-dessous utilise les mêmes définitions d'horloges.

Point-Point Link

Un Point-Point link est lien entre deux CODEC, Ainsi on va commencer par créer un test bench avec pour composant 2 CODEC.
C'est 2 CODEC ont la même horloge et ont leur signaux Strobe et Data connecté entre eux.

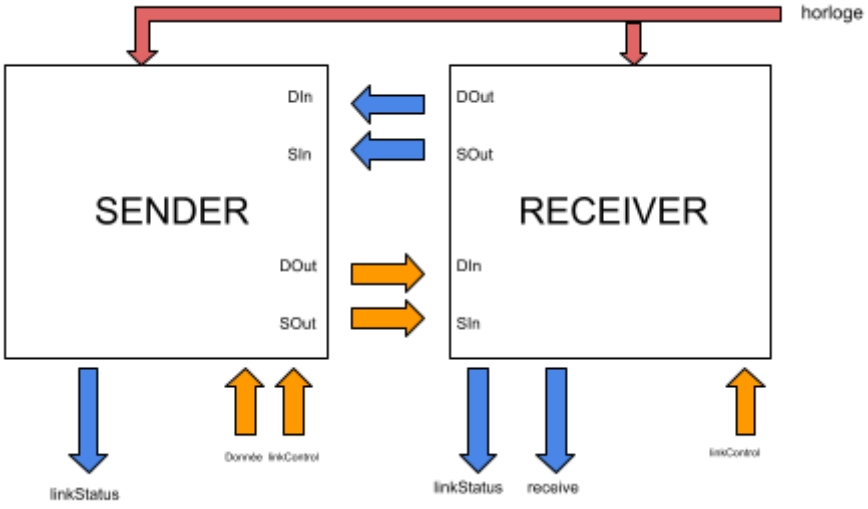


figure 2.1 - Schéma du lien Point-Point

les horloges sont réglé sur 50 mhz, 100 mhz et 166 mhz :

```
-- la clock sur 50 mhz
clocktime : process
begin
        aclock <= '0';
        wait for 20 ns;
        aclock <= '1';
        wait for 20 ns;
end process clocktime;

-- transmit clock sur 100 mhz
tclock : process
begin
        atransmitClock <= '0';
        wait for 10 ns;
        atransmitClock <= '1';
        wait for 10 ns;
end process tclock;
```

```
-- receive clock sur 166 mhz
rclock : process
begin
        areceiveClock <= '0';
        wait for 6 ns;
        areceiveClock <= '1';
        wait for 6 ns;
end process rclock;
```

figure 2.2 - Horloges

Ces chiffres sont proposés par le programmeur pour avoir un comportement du CODEC classique.

Une fois les horloges initialisées, on crée un processus qui va mettre le signal linkStart à ‘1’ des deux CODEC afin de passer dans l’état Run comme défini dans l’ECSS.

Lorsque la connexion est établie et que les deux CODEC sont dans l’état Run on peut envoyer le paquet.

```
wait until(rising_edge(aclock));
stransmitFIFODataIn <= ('0','0','1','0','0','0','1','1','1'); -- 47 héxa
stransmitFIFOWriteEnable <= '1';

-- et EOP
wait until(rising_edge(rclock));
stransmitFIFODataIn <= "100000000";

-- fin de demande de transfert
wait until(rising_edge(rclock));
stransmitFIFOWriteEnable <= '0';
```

figure 2.3 - envoie d’un paquet

Pour envoyer le paquet il faut activer la réception avec receiveFIFOReadEnable à 1 et aussi la transmission avec transmitFIFOWriteEnable comme nous montre la figure 2.3

<u>Résultats :</u>

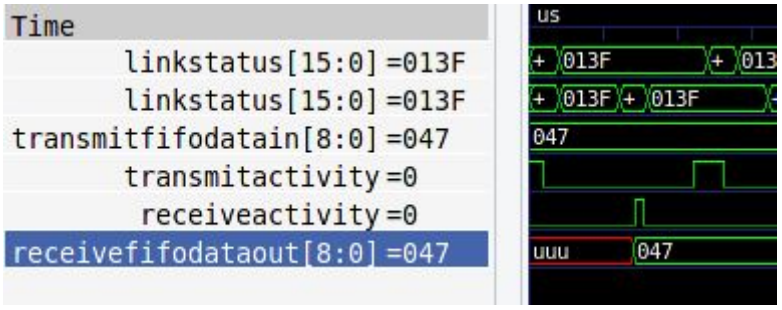


figure 2.4 - résultats Point-point

Si l’on regarde les activités de transmission et de réception ainsi que les données dans fifo on remarque que le message ici 47 est bien envoyé d’un CODEC à l’autre. Dans un échange dans un lien Point-Point il n’y a pas besoin d’envoyer l’adresse.

TestBench ROUTERIP

La mise en place du Routeur nécessite bien sûr le composant ROUTERIP mais aussi 6 CODEC qu'on va brancher sur les 6 ports physiques du Routeur.
Comme pour le Codec on initialise en premier les connexions entre les SpaceWirePorts(CODEC).

```
--======= initilisation des codec 1,2,3 4========
onelinkStart <= '1';
twolinkStart <= '1';
threelinkStart <= '1';
fourlinkStart <= '1';

tworeceiveFIFOReadEnable <= '1';
threereceiveFIFOReadEnable <= '1';
wait for 45 us;
```

figure 2.5 - initialisation SpaceWirePorts

4 ports sont activés, de plus le 2 et le 3 peuvent recevoir des paquets.
l'étape qui suit est l'envoi d'un paquet d'un port vers un autre pour cela on le découpe en 3 envoies : le chemin, la donné et le EOP.

```
--======= CONFIG 1 ==========
--envoie d'un paquet sur un port physique
--======= 1 ----> 3    ==========
--on donne l'adresse du port physique 3
wait until(rising_edge(rclock));
onetransmitFIFODataIn <= "000000011";
onetransmitFIFOWriteEnable <= '1';

-- puis la donnée 16
wait until(rising_edge(rclock));
onetransmitFIFODataIn <= "000010000";

-- et EOP
wait until(rising_edge(rclock));
onetransmitFIFODataIn <= "100000000";

-- fin de demande de transfert
wait until(rising_edge(rclock));
onetransmitFIFOWriteEnable <= '0';
```

figure 2.6 - Envoie d'un paquet sur le routeur

Le paquet est envoyé du port 1 au port physique 3. On n' a pas eu besoin de modifier la table de routage puisque c'est un port physique.
Ensuite on fait la même chose mais avec une adresse logique, donc pour cela il faut configurer la table de routage.

On va créer une mémoire afin de faire office de ramXilinix32*256. Cette mémoire est tout simplement un tableau avec un accès synchrone aux données du tableau.

```
PROCESS (clk, we)
BEGIN
        IF (we(0)='1') THEN
                IF (clk'EVENT AND clk='1') THEN
                        mem(to_integer(unsigned(addr))) <= din;
                END IF;
        END IF;
END PROCESS;
dout <= mem(to_integer(unsigned(addr)));
```

figure 2.7 - Mémoire synchrone

On commence par écrire dans la table de routage l'adresse logique 32 pour le port 2

```
--ecriture dans la table de routage
-- mettre port 2 adresse 32
wait until(rising_edge(rclock));
rbusMasterUserWriteEnableIn <= '1';
rbusMasterUserAddressIn <= std_logic_vector(to_unsigned(128,32));
rbusMasterUserDataIn <= std_logic_vector(to_unsigned(4,32)); -- 2
rbusMasterUserByteEnableIn <= "1111";
rbusMasterUserRequestIn <= '1';
rbusMasterUserStrobeIn <= '1';
```

figure 2.8 - Écriture table de routage

On écrit 4 dans la case mémoire 128, alors on met le troisième bit à 1 soit le port 2.
Ensuite pour vérifier que l'écriture est bien faite on fait une lecture de la table de routage.

```
--lecture dans la table de routage
rbusMasterUserAddressIn <= std_logic_vector(to_unsigned(128,32)); -- 4
rbusMasterUserRequestIn <= '1';
rbusMasterUserStrobeIn <= '1';
```

figure 2.9 - Lecture table de routage

Dans le cas de la lecture ou de l'écriture il faudra attendre que le BusMasterUserAcknowledgeOut soit redescendu à 0, pour arrêter la demande de lecture ou d'écriture.

Une fois la table de routage configurée, on envoie un paquet à une adresse logique enregistré comme nous montre la figure 2.9.

```
--======= CONFIG 3 ==================
-- envoie paquet adressage logique
--======= 1 ----> 48(3)    ==========
wait until(rising_edge(rclock));
onetransmitFIFODataIn <= "000110000";
onetransmitFIFOWriteEnable <= '1';

-- puis la donnée 16
wait until(rising_edge(rclock));
onetransmitFIFODataIn <= "000001000";

-- EOP
wait until(rising_edge(rclock));
onetransmitFIFODataIn <= "100000000";

-- fin de demande de transfert
wait until(rising_edge(rclock));
onetransmitFIFOWriteEnable <= '0';
```

figure 2.9 - Envoie d'un paquet avec adressage logique

Lors de l'envoi d'un paquet avec adressage logique , l'adresse n'est pas supprimée en entête

Configuration

Une mission dans le projet était de faire en sorte que la configuration de la table de routage se fasse à partir d'une lecture d'un fichier de spécification.
Pour cela on va lire un fichier specif.txt avec les données d'adresse et de ports physiques, dans ce format :

adresse logique espace adresse physique

En vhdl la lecture d'un fichier se fait grâce à la librairie text.io. Cette librairie permet d'utiliser des primitives telles que file_open, readline ou encore read.
Dans le programme on utilise ces différentes primitives pour lire le fichier.

```
while not endfile(file_read) loop
        readline(file_read, read_ligne);
        read(read_ligne,node);
        read(read_ligne,space);
        read(read_ligne,where);

        --ecriture dans la table de routage
```

figure 2.10 - Lecture fichier spécification

Avec cette boucle on récupère la première valeur, l'adresse logique puis la seconde l'adresse physique et on fait une écriture dans la mémoire avec le bus Master User Access. La partie de l'écriture dans la table de routage est comme celle définie dans la figure 2.8.

Résultats

Dans cette partie on va tester le temps de communication entre les ports selon des configurations différentes.
Tout d'abord nous allons mesurer le temps de l'envoi d'un paquet avec un adressage physique, puis logique. Ensuite nous allons tester avec 2 ports qui envoient un paquet sur le même port avec adressage physique et logique.

Adressage Physique

La configuration est simple, c'est le programme de base que nous allons mesurer avec les mêmes valeurs. On envoie le paquet du port 1 au 3.
On va mesurer la transmission du cargo(donnée) du paquet.
on écrit le cargo à 46020 ns comme nous le montre la figure 3.1

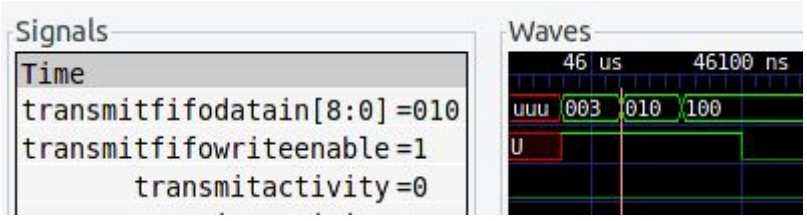


figure 3.1 - Envoie d'un paquet adressage physique

Et la figure 3.2 nous indique l'arrivée de la donnée sur le port 3 à 99500 ns.

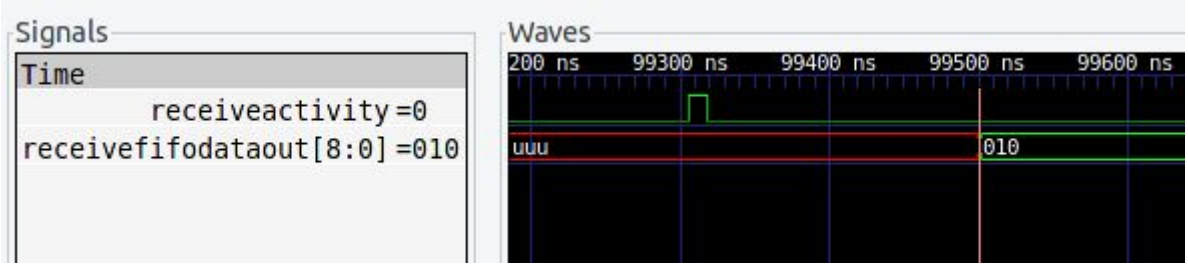


figure 3.2 - Réception d'un paquet adressage physique

On obtient un temps de 53480 ns, ceci sera notre temps de référence pour un envoi de paquet "classique" avec adressage physique.

Adressage physique avec 2 ports

On va tester d'envoyer de 2 ports différents un paquet sur le même port. Ici l'idée est de voir combien de temps le paquet va mettre pour être acheminé dans le pire des cas, donc dans celui où le port n'aura pas la main au début.

Comme pour la configuration simple on reçoit le premier message à 99500 ns, mais le second paquet n'arrive qu'a 104 900 ns comme nous montre la figure 3.3 :

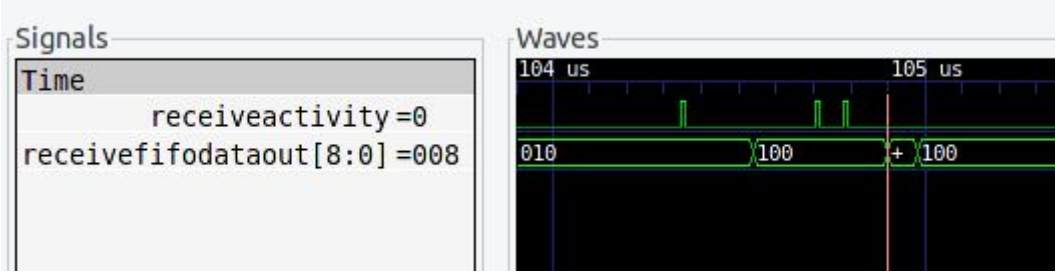


figure 3.3 - Réception de deux paquets avec adressage physique

Adressage logique

Ce test est exactement le même que celui avec un adressage physique à la différence que le paquet est envoyé par adressage logique.
l'envoi du cargo à le même temps d'envoie : 46020 ns,la figure 3.4 nous montre la réception à 99 540 ns.

figure 3.4 - Réception paquet adressage logique

on a une augmentation de 40 ns de plus dans le cas de l'adressage logique.

Adressage logique avec 2 ports

Le test est l'envoi de 2 paquets par adressage logique sur un port.
Le premier cargo est reçu au même temps qu'un envoi de paquet classique. Le second cargo va arriver plus tard à 105 140 ns, voir la figure 3.5 .

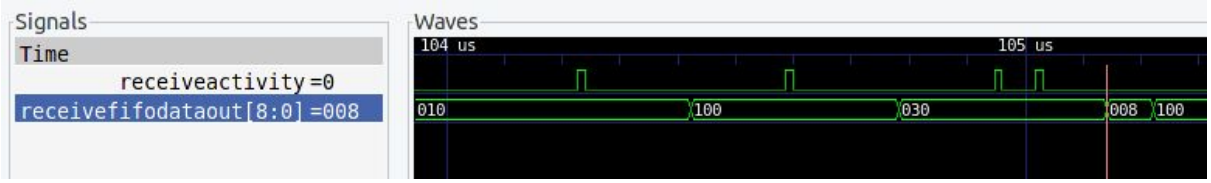


figure 3.5 - Réception paquet 2 ports avec adressage logique

Améliorations

Durant ce projet on travaille exclusivement avec ghdl et gtkwave en ligne de commande, cela implique qu'on crée une mémoire afin de remplacer celle de Xilinx32*256. En effet, comme cela est mentionné précédemment le programme à été généré par l'ide Xilinx. Il serait intéressant de refaire toute la partie de test sur le temps de communication avec Xilinx pour comparer les résultats.

Dans les programmes on ne fait pas appel aux signaux de times codes, d'interruptions ou encore aux tickets. Ces signaux sont optionnels pour le fonctionnement de SpaceWire mais cela peut peut-être avoir un impact sur le temps de communication.
La suite logique des programmes serait après le bon fonctionnement du système de mettre en place ces options.

Le test du routeur n'est pas vraiment optimal dans le sens ou il n'est pas testé dans un environnement complet. Dans le test bench du système SpaceWire avec le routeur on a qu'un routeur et 6 ports. On pourrait placer plusieurs routeurs et plusieurs ports afin de faire un réseau plus complexe comme c'est le cas sur un satellite.

Conclusion

Pour conclure, le projet était de comprendre, tester et mesurer un réseau de communication spatiale,SpaceWire. Pour mener à bien cette mission j'ai effectué tout d'abord une veille technologique sur le standard ECSS 12-C. Par la suite j'ai fait la même chose mais sur le code sources mise à disposition sur github par l'entreprise Shimafujigit.
Une fois acquis la connaissance du réseau et du code, j'ai testé le fonctionnement du programme avec ghdl. Tout d'abord le premier composant, le codec puis le routeur et le codec en même temps.
Le programme fonctionnant j'ai ainsi pu faire des mesures sur l'échange des paquets à travers un routeur entre les ports. Ces mesures nous ont montré que les échanges sont très rapides même s' il y peut y avoir de légère attente si plusieurs paquets arrivent sur le même port.

Ce projet a permis le test du fonctionnement du code sources et aussi les temps d'échanges de paquets. Cependant le réseau de tests que j'ai fait prend en compte qu'un routeur et plusieurs codecs. Il serait intéressant de voir les échanges dans un réseau plus complexe avec plusieurs routeurs et codecs. De plus il y a d'autres fonctionnalités dans le réseau que j'ai n'ai pas traité.

Durant ce projet j'ai pu développer mes compétences avec le langage vhdl, mais aussi mes connaissances dans le domaine de réseaux de communication sur les systèmes embarqués spatiaux.
Une de mes grandes passions est l'astronomie et l'un de mes souhaits serait de travailler dans ce domaine grâce à l'informatique et ce projet m'a réconforté dans cette idée.

Annexes

Environnement et Installation

La totalité du projet s'est faite sous linux ubuntu sur la dernière version.
Les différents programmes utilisés dans ce projet sont écrits en VHDL, ainsi pour la phase de compilation on utilise l'outil ghdl.
Cet outil permet de compiler des fichiers vhdl en ligne de commande, ceci était une des conditions de développement. Il existe un IDE qui propose le même service mais de manière intégrée, Xillinx.

Installation de ghdl :

```
sudo apt-get install ghdl
```

Utilisation de ghdl :

```
ghdl -a -fexplicit --ieee=synopsys fichier.vhdl
ghdl -e -fexplicit --ieee=synopsys entiter
ghdl -r -fexplicit --ieee=synopsys entiter --vcd=test.vcd --stop-time=50us
```

analyse du fichier.vhdl avec -a
elaboration de l'entité avec -e
run de l'entité avec -r que l'on va écrire dans un fichier test.vcd pour un temps de 50 us.

Afin de pouvoir simuler les programmes on utilise l'outil gtkwave, qui comme ghdl s'utilise aussi en ligne de commande.

Installation de gtkwave :

```
sudo apt-get install gtkwave
```

Utilisation gtkwave :

```
gtkwave test_codec.vcd
```

On lance gtkwave avec le fichier vdc créé avec ghdl -r.

Pour la simplification de la compilation on utilise un fichier bash qui réunit toutes les commandes de compilation des programmes, souvent appelé go.sh. Ceci étant dit ghdl à une option qui permet de générer un makefile.

Lors du lancement du programme il y aura souvent des Warnings dû à des signaux non initialisés ou qui ont des valeurs inconnues, ceci est tout à fait normal puisqu'on ne travaille pas avec tous les concepts du réseau SpaceWire.