PJ
Protocole de communication

Jérémy Cheynet
Yann Sionneau

Année 2010

Table des matières

Première partie .

Théorie

1. Objectifs

L'objectif de notre projet est de créer un système qui permet de faire communiquer toutes sortes d'électronique embarquée entre elles. Une des conditions que nous nous sommes fixée, est de faire communiquer les appareils sur un seul et unique fil pour pouvoir, dans un second temps, faire communiquer ces appareils avec une liaison sans fils.

Les systèmes embarqués que nous comptons utiliser sont des microcontrôleurs et des FPGA [1].

Afin de faciliter la communication entre les appareils, et l'architecture de notre système, nous avons choisi d'utiliser une structure sous forme de couches d'abstractions (en s'inspirant du model OSI).

1. *FPGA* (field-programmable gate array, réseau matriciel de portes logiques programmables).

2. Spécifictions

2.1. Matériel

Notre choix étant de faire fonctionner notre système sur plusieurs système et différente architecture. Pour cela, nous avons décidé de développer notre système sur un microctontrôleur et sur un FPGA. Nous avons donc choisi, parmi la grande gamme de produits disponibles dans ces 2 catégorie, une architecture de microcontrôleur et un type de FPGA.

2.1.1. Microcontrôleur

Nous avons choisi d'implémenter notre système sur microcontrôleur car ces puces électroniques sont très répandues, très utilisées et faciles d'utilisation. En effet, utiliser un microprocesseur oblige de rajouter de la mémoire, et des modules externe, tandis qu'un microcontrôleur est autonome.

Le microcontrôleur que nous utilisons fait partie la gamme ATMega (architecture AVR) de chez ATMEL. Ce type de microcontrôleur est facilement programmable en C/C++, et se trouve pour un prix correct.

Voici les 3 types de microcontrôleurs que nous allons utiliser :

La carte arduino : Il s'agit d'une carte microcontrôleur toute prête, programmable en C/C++ avec un logiciel fourni gratuitement. Tout les programmes et toutes les shields [1] sont open-sources. Le microcontrôleur qui se trouve sur la carte est un ATMEGA168 de chez ATMEL.

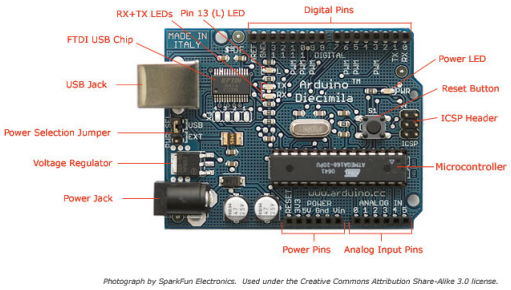


FIGURE 2.1.: Carte Arduino

L'ATMEGA324 : Il s'agit d'un microcontrôleur 40 broches de 32 entrées/sorties de chez ATMEL. La carte que nous utilisons pour l'utiliser est une carte faite maison, et programmée grâce au logiciel libre avrdude à l'aide d'un ISP programmer (système libre pour programmer la série avr de chez ATMEL). Le programme s'effectue toujours en C/C++ et se compile grâce à la toolchain GNU pour l'architecture AVR (avr-gcc, avr-objdump, avr-ld, avr-as etc ...).

L'ATTINY13 : Microcontrôleur à prix réduit de chez ATMEL. Il s'agit d'un petit microcontrôleur de 8 broches, programmable comme l'ATMEGA324. Là aussi, nous allons designer une carte pour pouvoir faire des tests de notre système.

1. Petite carte électronique que l'on peut facilement brancher sur la carte arduino

2.1.2. FPGA

Nous avons choisi d'utiliser pour nos tests un Spartan-3A 400k portes (XC3S400A) de chez Xilinx. Cette puce est présente sur la plaque de développement "AVnet Spartan-3A Evaluation Kit" en notre possession. Un port de 40 pins d'entrées/sorties (GPIOs) est disponible sur cette plaque que nous pourront utiliser pour tester le protocole, en la reliant par exemple à un Arduino. Le bitstream est synthétisé en utilisant les outils Xilinx ISE Webpack (Xst), le code source est en langage Verilog et versionné sur github. Une simulation est faite en utilisant le logiciel icarus verilog, les résultats de cette simulation sont analysés en utilisant gtkwave qui génère les chronogrammes des signaux importants.

FIGURE 2.2.: Plaque de développement "AVnet Spartan-3A Evaluation Kit"

2.2. Le BUS

Notre système fonctionne sur 1 seul et unique fil. Les appareils communiquent entre-eux en envoyant des signaux logiques sur ce fil.

Afin de ne pas avoir conflit entre les appareils qui pourront communiquer sur ce fil, nous avons choisi :
Topologie du bus : 1 maitre, plusieurs esclaves.
Protocole de communication : Questions / Réponses

2.3. Structure des couches d'abstractions

2.3.1. La couche physique

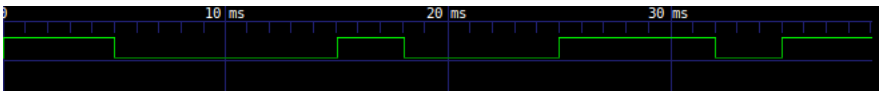
2.3.1.1. Etat logique par défaut

Par défaut, la ligne sera à un état logique haut (5V). Lorsqu'un des appareils voudra communiquer avec un autre, il devra créer un front montant pour commencer la communication (donc, passer par un état bas avant de repasser par un état haut). Ainsi, avant de communiquer, tout appareil n'ayant pas reçu d'interruption pendant un certain temps, verifiera que la "ligne" est à un état logique haut. Si ce n'est pas le cas, il attendra avant d'emmettre son signal.

2.3.1.2. Définition des bits

Un bit est un quantum de temps de 10ms (que nous pourrons diminuer plus tard). Il existe 4 types de bit :

- Le bit de start
- Le bit d'un état logique bas
- Le bit d'un état logique haut
- Le bit de stop



De gauche à droite : Le bit de start (10ms à l'état bas), suivi du bit représentant l'état logique bas, puis celui de l'état logique haut, et enfin, le bit de stop (front montant restant à l'état haut).

FIGURE 2.3.: Les 4 types de bits possible

Un des objectifs est de ne pas avoir de désynchronisation entres les 2 appareils qui communiquent entre eux.

En effet, supposons que tout les bits fassent 10ms. Une trame normal fait 9 bits (1octet + 1 bit de parité). Le récepteur va, à partir du premier bit reçu déclencher un timer, et regarder régulière le signal pour voir à quel état se trouve celui-ci. Cependant, si les 2 systèmes n'ont pas exactement la même fréquence, il y aura une désynchronisation des 2 appareils et une perte de bit.

Afin d'éviter cela, nous avons décidé de resynchroniser nos appareils à chaque bit. Tous les bits (état logique haut ou bas) commencent par un front montant. A partir de ce moment, le récepteur lance un timer. Au bout de 5ms (50% du temps du bit), il va regarder l'état du signal (haut ou bas) et en déduire l'état du bit. Mais au lieu de faire continuer son timer, il va le stopper, et le relancer lors de l'arriver du prochain bit par un front montant.

Pour cela, un état logique haut sera représenté par un état haut de 2/3 du temps total (soit 7ms) puis d'un état bas pendant 1/3 du temps (3ms). Il en va de même pour l'état logique bas, représenté par un état haut de 1/3 du temps et les 2/3 restant par un état bas. Ainsi, la durée du bit est de 10ms, tout les bits commencent par un front montant et finissent pas un état bas (permettant de faire un nouveau front montant).

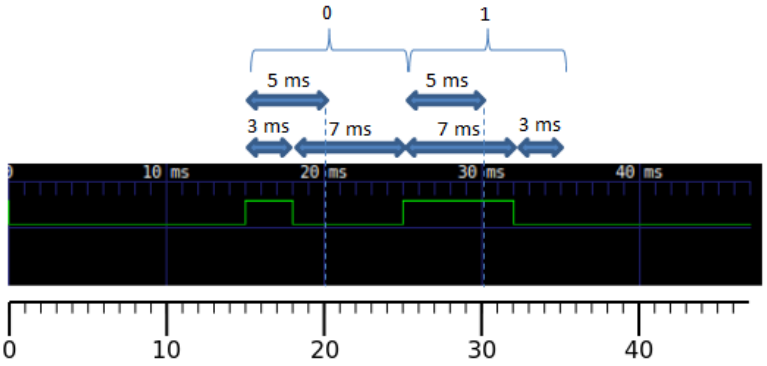


FIGURE 2.4.: Représentation des état haut et bas

2.3.1.3. Trame de données

La trame de données est constituée par une trame de 8 bits contenant un octet de données, suivi par 1 bit de parité. Cela permet d'avoir une première vérification du signal sur la couche physique. En cas d'erreur, celle-ci est transmise à la couche supérieur, qui traitera l'erreur.

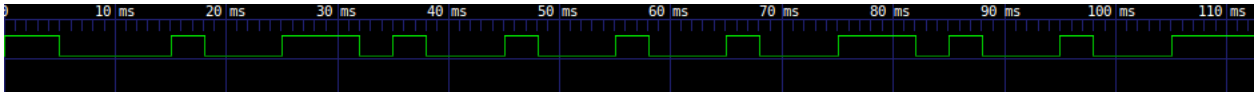


FIGURE 2.5.: Exemple d'une trame d'un octet (0x42)

2.3.2. La couche de transport

La couche de transport nous permet d'avoir un système d'adresse et de vérification du signal.

2.3.2.1. L'adressage

L'adressage se fait sur un octet. Lors de la communication entre 2 appareils, le premier transferts correspondra à l'adresse de l'appareil source. Ensuite, nous transmettrons l'adresse de destination.

Cela permet de pouvoir s'adresser à un seul et unique appareil.

2.3.2.2. La vérification

Nous avons un octet qui est réservé pour pouvoir transférer des données de vérification ainsi que la taille du paquet.

Les 4 premiers bits de cet octet servent à définir la taille de notre paquet, pour pouvoir dire au récepteur combien d'octets de données utiles sont transférés.

Les 4 derniers bits servent de checksum, pour s'assurer qu'il n'y a pas d'erreur dans le transfert des données dans le paquet.

2.3.2.3. La forme d'un paquet

Après ces 3 octets, nous transférons un certain nombres de paquets de données utiles, qui dépendra de ce que l'émetteur veut envoyer.

Voici la structure d'un paquet :

1. Un octet contenant l'adresse source
2. Un octet contenant l'adresse de destination
3. Un octet de vérification :
   - 4 bits indiquant le nombre d'octet de donnée utile à transferer
   - 4 bits de checksum sur le paquet total pour éviter d'avoir des erreurs.
4. Les octets contenant les données utiles (payload).

Adresse Source	Adresse Destination	Taille du paquet	Checksum	Octet(s) de donnée(s)

FIGURE 2.6.: Structure d'un paquet

2.3.3. La couche applicative

La couche supérieur est la couche applicative. C'est celle qui appellera la fonction d'envoie pour l'émetteur, et qui sera appelée lorsqu'une interruption arrive, signifiant l'arrivée d'un paquet, déjà traité par la couche 2.

Deuxième partie .

Pratique

3. Microcontrôleur

4. FPGA

Table des figures