PROJET S8 - IMA4

RAPPORT - *Décodage de codeurs en quadrature de phase sur FPGA*

Benjamin LAFIT Valentin VERGEZ

Table des matières

1 Introduction

En robotique mobile mais aussi plus largement dans beaucoup de domaines techniques, il est souvent nécessaire d'obtenir des informations sur des distances parcourues.
Dans l'industrie, ça peut concerner par exemple l'avance d'un tapis roulant ou encore la rotation d'un plateau tournant.
En robotique mobile plus spécifiquement, un robot doit se repérer dans l'espace et savoir effectuer des déplacements précis. Pour ce faire, une méthode simple est de récupérer les informations de distance sur le déplacement puis de les traiter.

Pour faire ces mesures, il existent plusieurs catégories de capteurs.
Celle qui nous intéresse est celle des encodeurs et plus spécifiquement les encodeurs à signaux en quadratures de phases.
Un encodeur est un dispositif qui va retransmettre un mouvement ou une position sous la forme d'un codage particulier.

Le principe de codage à signaux en quadratures de phases sera détaillé plus tard.

1.1 Problématique

Notre projet s'inscrit dans le cadre du club Robotech Lille, club de robotique de l'école Polytech Lille, et de la Coupe de France de Robotique organisée par Planètes Sciences.

Cette Coupe de Robotique exige la conception et la réalisation de robots mobiles autonomes.
A Robotech Lille, les informations propre au déplacement sont données par ces fameux encodeurs à signaux à quadratures de phases.
Il est donc nécessaire que soit intégrée au robot un système de décodage des signaux à quadratures de phase.

Précédemment, le robot était géré par un unique micro-contrôleur, aussi en charge du décodage de ces signaux.
Cette année est née la volonté d'attribuer la gestion complète du déplacement du robot à un sous-module, facilement réutilisable et indépendant de la structure du robot. Il a été décidé, pour des raisons que nous allons voir ensuite, que ce sous-module allait être géré par un FPGA [1]. La gestion du décodage des signaux en quadrature de phase doit donc être elle aussi assumée par le FPGA, c'est à cette partie là que nous nous intéressons.

1. Field-Programmable Gate Array, un réseau de portes programmables (Voir Wikipedia : Circuits Programmables)

1.2 État de l'art

Le principe de base des encodeurs à signaux en quadrature de phases est de générer deux signaux carrés identiques, décalées de $\pm 90°$. Ce décalage contient une information importante, il permet de savoir si le codeur rotatif tourner dans le sens trigonométrique ou anti-trigonométrique.

Ces signaux carrés proviennent en fait de deux capteurs qui signale le passage ou non d'un marquage placé sur un disque tournant avec le rotor du codeur.
Si le disque ne contient d'un marquage, une période du signal indiquera que le codeur a effectué un tour complet.
L'encodeur que nous utilisons possède lui 1024 marques sur son disque, on dit alors que le codeur a une résolution de 1024 ticks par tours.

Pour interpréter ces ticks et ce décalage puis les traduire en une position, en un comptage/décomptage, on utilise traditionnellement des circuits logiques intégré dédiés à cela. On peut aussi utiliser un micro-contrôleur pour effectuer cette tâche, seulement si le codeur tourne vite, à raison de plusieurs milliers de ticks par tours, les fréquences des signaux à interpréter deviennent importantes et trop encombrantes pour le micro-contrôleur.
L'inconvénient des circuit intégrés est leur difficulté d'intégration. Pour une utilisation ponctuelle, leur volume est trop important au regard de leur fonction et il est parfois difficile de les interfacer au reste de l'application (gros bus de données). Parfois ils sont intégrés aux micro-contrôleurs, le problème devient alors leur nombre limité ou le prix.
La solution FPGA se présente comme un intermédiaire très intéressant. Le principe de fonctionnement du FPGA permet de retrouver les performances identiques aux circuits intégrés (puisqu'il s'agit simplement d'intégrer le fonctionnement de ces circuits dans le FPGA). Il permet aussi d'obtenir un système beaucoup plus complet, comme celui qu'assumerait un micro-contrôleur, mais avec une fiabilité et une robustesse inégalable.
L'inconvénient est l'aspect statique de l'application et la difficulté d'implémentation, c'est d'ailleurs pour cette raison qu'il est rare de trouver des systèmes de déplacement complet gérés ainsi.

1.3 Contenu du projet

Le sujet initial propose de réaliser un système simple sur cible FPGA permettant de :

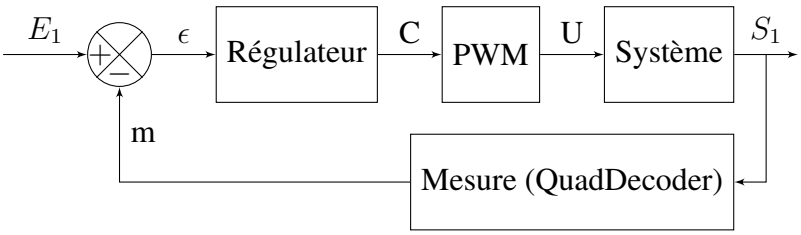
– Lire et décoder des signaux en quadrature de phase en provenance d'une ou plusieurs roues codeuses ;
– Asservir un ou plusieurs moteurs, soit pas-à-pas soit à courant continu.

La cible visée est un FPGA de chez Xilinx (carte de développement LX9 à base de Spartan 3 [2]).
En parallèle, une carte d'interface de puissance doit être réalisée, c'est l'interface entre FPGA et moteurs.

Les objectifs du sujet sont atteints. Dans ce document sera expliqué le principe du décodage de signaux utilisé, l'asservissement, la génération de signaux logique de commande du moteur, la carte de puissance d'interfaçage FPGA - Moteur ainsi qu'un élément supplémentaire.
En effet, comme la finalité de ce projet est d'être utilisé dans un sous-module complet, nous avons commencé à implémenter une interface série avec une mémoire interne au FPGA. L'objectif est d'avoir un module fonctionnel, complètement configurable et pilotable par liaison série. Ainsi, les utilisateurs futures n'auront qu'à effectuer les branchements du module à leur robot et à le commander avec leurs autres modules électroniques, via liaison série. Ils n'auront pas à se soucier du fonctionnement interne, il s'agira d'une boîte noire.

Toutefois, comme ce sous-module peut-être sujet à des améliorations, les codes sources sont bien sûr fournis aux utilisateurs et doivent permettre une mise à jour facile de l'application.
C'est en tout cas pour ce soucis d'utilisation facile que nous avons choisis ce système de pilotage série.

2. Le Spartan 3 est un modèle de FPGA de la marque Xilinx


E_1
ϵ
Régulateur
C
PWM
U
Système
S_1
m
Mesure (QuadDecoder)

2 Interface série

Dans le but d'établir une communication entre le Raspberry (partie intelligente) et le FPGA nous avons adopter un mode de communication série. Ce protocole étant relativement simple à coder et à mettre en place, c'est pourquoi nous avons décider de l'utiliser au sein du robot pour l'échange de données.

2.1 Reception

Ce programme agit sur chaque front montant de l'horloge. Son but sera de permettre la réception des données à 9600 bauds(debit paramétrable par une variable générique). A chaque front montant de l'horloge,nous incrémentons un compteur de 1 jusqu'à compter la période d'un bit (5208 dans le cas d'une transmission à 9600 bauds avec un horloge de 50MHz). Lorsque ce chiffre est atteint nous récupérons la valeur du bit Rx et la stockons dans la variable data_next par décalage à chaque fois. De cette manière nous aurons, dans data_next, la valeur des bits du message mais dans le sens inverse. A chaque récupération de bit, la variable i est incrémentée de afin de connaître la fin de réception de la donnée. Au premier front d'horloge (bit de start à 1), le compteur est initialisé à sa moitié afin que la récupération des données se fasse au milieu de la période du bit pour être sur d'avoir la bonne valeur. Lorsque i=nombre bit de données + 2 (le nombre de bit de données est paramétrable par une variable générique), tous les bits du messages ont été enregistrés. Nous écrivons donc la donnée ainsi reçue sur le registre de sortie de manière inversée (oData(0)=Data_next(8) ...).

2.2 Emission

Ce programme, sur chaque front montant va commencer la transmission si le bit iEnableTransmit est à l'état chaud. A ce moment là on va incrémenter un compteur qui permettra de faire une communication à un debit donné. C'est à dire que pour une communication à 9600 baud par exemple, on doit laisser le bit de data à son état pendant 5208 coups d'horloge avec un horloge à 50MHz.

2.3 Module de gestion

3 Interface PWM

Le module de PWM permet de contrôler un moteur grâce à une modulation par largeur d'impulsion. Ce module prends en entrée un valeur sur 8bits permettant de régler la valeur du rapport cyclique de la PWM.

4 Améliorations possibles

5 Conclusion

Annexes

Listing 1 – QuadDecoder.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity QuadDecoder is
7  --Configurable quadrature counter width; default is 32 bits
8     generic (register_width : integer:= 32 );
9     port (clk: in STD_LOGIC;
10          A : in STD_LOGIC;
11          B : in STD_LOGIC;
12          reset : in STD_LOGIC;
13          -- Sortie
14          Bd : out STD_LOGIC;
15          Ad : out STD_LOGIC;
16          sEn : out STD_LOGIC;
17          sUp : out STD_lOGIC;
18          count : out STD_LOGIC_VECTOR(register_width-1 downto 0));
19 end entity QuadDecoder;
20
21 -- Architecture
22 architecture behavioral of QuadDecoder is
23
24 -- Signals
25 signal A_delayed : std_logic ;
26 signal B_delayed : std_logic ;
27 signal enable : STD_LOGIC;
28 signal up : STD_lOGIC;
29 signal count_register : STD_LOGIC_VECTOR(register_width-1 downto 0);
30
31 begin
32
33 process (clk) begin
34   if (clk'event and clk='1') then
35     up <= A XOR B_delayed;
36     enable <= A XOR B XOR A_delayed XOR B_delayed;
37     A_delayed <= A;
38     B_delayed <= B;
39
40     if (reset='0') then
41         if (enable='1') then
42             if (up='1') then
43                 count_register <= count_register+1;
```

```
44              else
45                  count_register <= count_register-1;
46              end if;
47          end if;
48      else
49          count_register <= (others =>'0');
50      end if;
51    end if;
52 end process;
53
54 Ad <= A_delayed;
55 Bd <= B_delayed;
56 count <= count_register;
57 sEn <= enable;
58 sUp <= up;
59
60 end architecture behavioral;
```