|  |
| --- |
| Polytech Paris UPMC |
| Documentation technique du projet RobLip6 |
|  |

|  |
| --- |
| Raphaël BRESSON, Charles ROGER, Zeng Zhaozong  18/04/2016 |

Matériel requis

1. Partie « Basse » :

* Carte FPGA MOJO V3
* Lidar Lite V2
* Bras articulé et base roulante
* IMU

1. Partie « Haute »

* Caméra Mobius
* Carte Firefly

### Outils logiciels requis

1. Partie « Basse »

* ISE de Xilinx avec la licence WEBPACK
* Mojo Loader de Embedded Micro

1. Partie « Haute »

* Eclipse (développement JAVA)
* Bibliothèque native RXTX (communication UART)
* Bibliothèque native OPENCV (traitement d’images)

## PARTIE « BASSE » (LOW LEVEL)

### Fonctions

* Acquisitions des capteurs
* Contrôle des actionneurs
* Réception des ordres pour les actionneurs via UART
* Envoi des valeurs des capteurs via UART

Toutes ces fonctions sont réalisées en parallèle les unes des autres(pipeline).

#### Solution FPGA avec la carte MOJO V3 (Embedded Micro)

##### Avantages de cette solution :

* Code relativement indépendant de la carte utilisée (code VHDL)
* Pins entièrement reconfigurables
* Vrai parallélisme
* Optimisations temps réel possibles
* Reconfiguration dynamique

##### Inconvénients de cette solution :

* Prix de la carte MOJO (~ 80 €)
* Développement hardware long et complexe
* Le FPGA (Spartan 6 xc6lsx9) est sous-utilisé

##### Synoptique Global

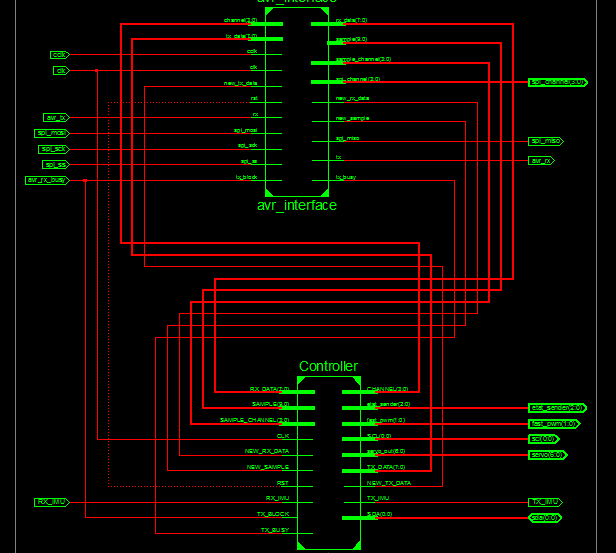
MOJO V3

CONTROLLER

* Gestion haut-niveau de la communication avec le HIGH LEVEL
* Gestion des capteurs et des actionneurs

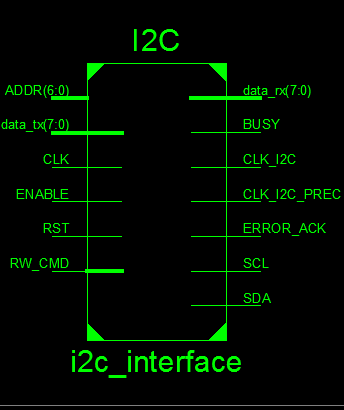
AVR\_INTERFACE

* Gestion de la bas-niveau de la communication avec le HIGH LEVEL
* Gestion bas-niveau des sorties analogiques



##### IP I2C

Il s’agit d’une IP permettant de contrôler un bus I2C en tant que maître.



Remarque : Ici on contrôle toujours SCL (cette IP est donc incompatible avec le mode I2C multi-masters). La valeur de la résistance de pull up nécessaire est de 1kOhms.

**ACK\_ERROR=’1’**

**BitNumber > 0**

**BitNumber > 0**

**WR=’1’**

**WR=’0’**

**BitNumber = 0**

**BitNumber = 0**

**BitNumber = 0**

**BitNumber > 0**

**ENABLE = 1**

**ENABLE = 0**

**RW = 1**

**ENABLE = 0**

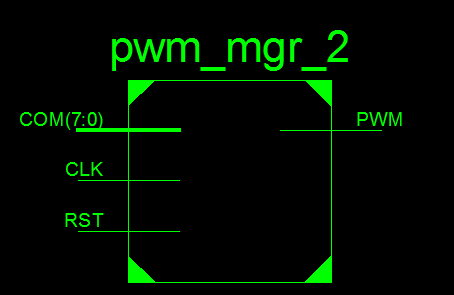
**ENABLE = 0**

RW = 0

**RW = 0**

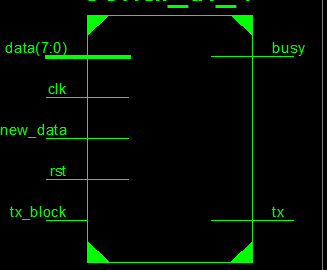
**I2C\_MASTER**

##### IP PWM



Cette IP set à générer les sorties PWM pour les servomoteurs et les roues. On contrôle la largeur de l’impulsion grâce au signal COM (COM\_LENGTH permet de régler la taille de ce bus). La variable GENERIC CMP\_LENGTH permet de régler la fréquence des impulsions (log2(50MHz/fréquence désirée)) et OFFSET permet de décaler (comprendre déphaser) l’impulsion.

##### IP UART\_TX



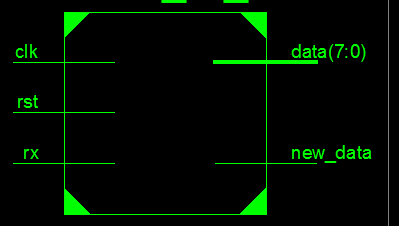
Cette entité permet d’envoyer un octet via UART (envoi séquentiel bit par bit).

Quand un nouvel octet doit être envoyé, il faut mettre NEW\_DATA à ‘1’et écrire cet octet sur le bus DATA.

BUSY à 1 signifie que l’on est déjà en train d’écrire un octet et qu’il faut attendre que cette écriture se termine.

Le débit de l’UART en bauds est contrôlé par CLK\_PER\_BIT = 50MHz/fréquence désirée et CTR\_SIZE=log2(CLK\_PER\_BIT).

##### IP UART\_RX



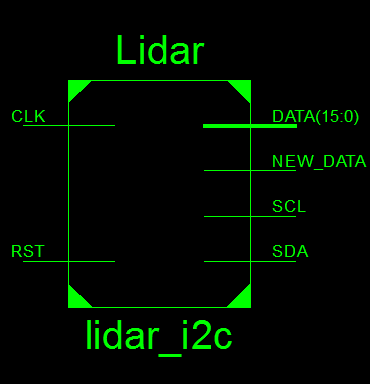
Cette entité permet de lire un octet sur UART (réception séquentielle bit par bit).

Quand un nouvel octet est lu new\_data vaut ‘1’ et on peut lire cet octet sur le bus DATA.

Busy à 1 signifie que l’on est déjà en train d’écrire un octet et qu’il faut attendre que cette écriture se termine.

Le débit de l’UART en bauds est contrôlé par CLK\_PER\_BIT = 50MHz/fréquence désirée et CTR\_SIZE=log2(CLK\_PER\_BIT).

##### IP LIDAR

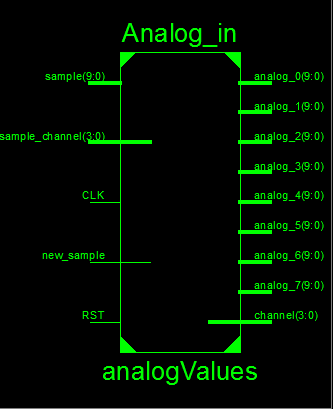


Cette entité gère le Lidar. Quand NEW\_DATA est à ‘1’ on peut lire une nouvelle valeur sur le bus DATA



Trame obtenue

##### Sorties Analogiques



Cette entité permet de récupérer les valeurs analogiques sur la carte MOJO V3.

Analog\_0 : A0

Analog\_1 : A1

Analog\_2 : A4

Analog\_3 : A5

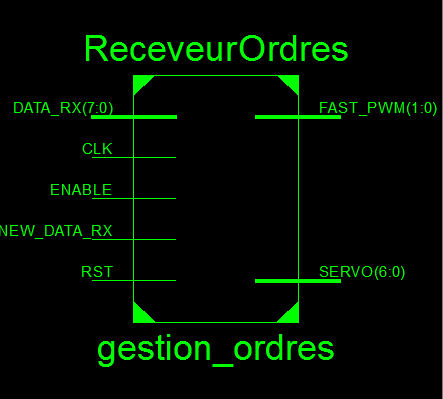
Analog\_4 : A6

Analog\_5 : A7

Analog\_6 : A8

Analog\_7 : A9

##### IP Gestion Ordres :



Cette entité reçoit les ordres du HIGH LEVEL et gère les actionneurs en conséquence.

**ReceveurOrdres**

**NEW\_RX\_DATA=1**

**NEW\_RX\_DATA=1**

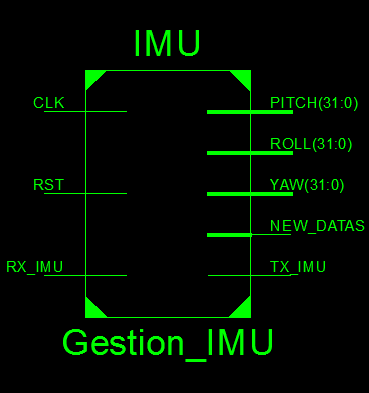
**NEW\_RX\_DATA=0**

**NEW\_RX\_DATA=0**

**ENABLE=1**

RW = 0

##### IP IMU



Cette entité gère le capteur IMU. On récupère les angles d’Euler (bus YAW, PITCH, ROLL) lorsque NEW\_DATAS vaut ‘1’.

**TX\_BUSY=1 ou NEW\_TX\_DATA=1**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt<4**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt<4**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt<4**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt<4**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt=0**

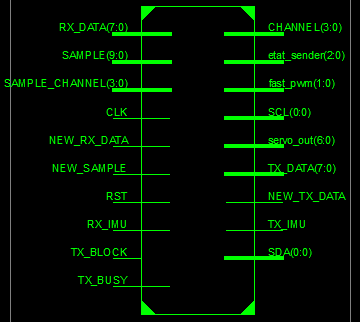
**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt=0**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et cpt=0**

RW = 0

**IMU**

##### IP Controller



Cette IP regroupe le contrôle des capteurs et des actionneurs.

**TX\_BUSY=1 ou NEW\_TX\_DATA=1 ou byte\_counter>0**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et byte\_counter=0**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et byte\_counter=0**

**TX\_BUSY=0 et NEW\_TX\_DATA=0 et byte\_counter=0**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1 ou byte\_counter>0**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1 ou byte\_counter>0**

**TX\_BUSY=0 et NEW\_TX\_DATA=0**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1**

**TX\_BUSY=0 et NEW\_TX\_DATA=0**

**TX\_BUSY=1 ou NEW\_TX\_DATA=1**

RW = 0

**CONTRÔLE**