

Spécification des interfaces pilote de barre franche

1) Spécifications du circuit « gestion vérin ».

Le circuit de gestion du vérin est constitué de quatre fonctions (process) principales :

- Gestion de la PWM moteur
- Contrôle des butées de fin de course du vérin
- Gestion du convertisseur AN MCP 3201 de copie de position de barre
- Interface avec le bus Avalon du NIOS

Gestion de la PWM : elle fait appel à 2 fonctions secondaires :

- Une fonction qui fixe la fréquence de la PWM (process divide)
- Une fonction qui fixe le rapport cyclique et génère le signal PWM en sortie (process compare)

Contrôle des butées : utilise la fonction principale « controle_butées » :

- met le signal « PWM » à 0 si « angle_barre » se situe en dehors des butées « butee_g » et « butee_d » et selon le sens de rotation du moteur.
- génère les signaux « fin_course_d » et « fin_course_g »

Gestion du convertisseur AN MCP 3201 : fait appel à 5 fonctions secondaires :

- machine à état pour piloter l'adc et mémoriser la donnée « angle_barre » (process pilote_adc)
- comptage des fronts d'horloge de clk_adc (process compt_fronts)
- registre à décalage pour récupérer la donnée du convertisseur (process rec_dec)
- génération du 1MHz pour la machine à état (process gene_1M)
- génération périodique (toutes les 100ms) du signal « start_conv » (process gene_start_conv)

Interface avec le bus Avalon : fait appel à 2 fonctions secondaires (process) :

- Ecriture des data circulant sur le bus du NIOS dans des registres (écriture)
- Relecture de signaux par renvoi sur le bus du NIOS (lecture)

L'interface Avalon dispose de 6 registre tels que décrits ci-dessous :

<i>Registre</i>	<i>adresse</i>	<i>type</i>	<i>Bits concernés</i>
<i>freq</i>	<i>0</i>	<i>R/W</i>	<i>b15..b0</i>
<i>duty</i>	<i>1 (4)</i>	<i>R/W</i>	<i>b15..b0</i>
<i>Butee_g</i>	<i>2 (8)</i>	<i>R/W</i>	<i>b15..b0</i>
<i>Butee_d</i>	<i>3 (12)</i>	<i>R/W</i>	<i>b15..b0</i>
<i>config</i>	<i>4 (16)</i>	<i>R/W</i> <i>R/W</i> <i>R/W</i> <i>R</i> <i>R</i>	<i>b0 :raz_n (à 0=reset circuit)</i> <i>b1 :enable_pwm (1= pwm actif)</i> <i>b2: sens rotation (0=gauche)</i> <i>b3:fin_course_d (=1 si fin course_d)</i> <i>b4: fin_course_g (=1 si fin course_g)</i>
<i>Angle_barre</i>	<i>5 (20)</i>	<i>R</i>	<i>b11..b0</i>

Affectation des registres du circuit

Freq : fixe la fréquence de la PWM moteur (exemple : si freq = 2000 alors la fréquence de la PWM = $\text{clk}/2000$ soit 25KHz si $\text{clk}=50\text{MHz}$)

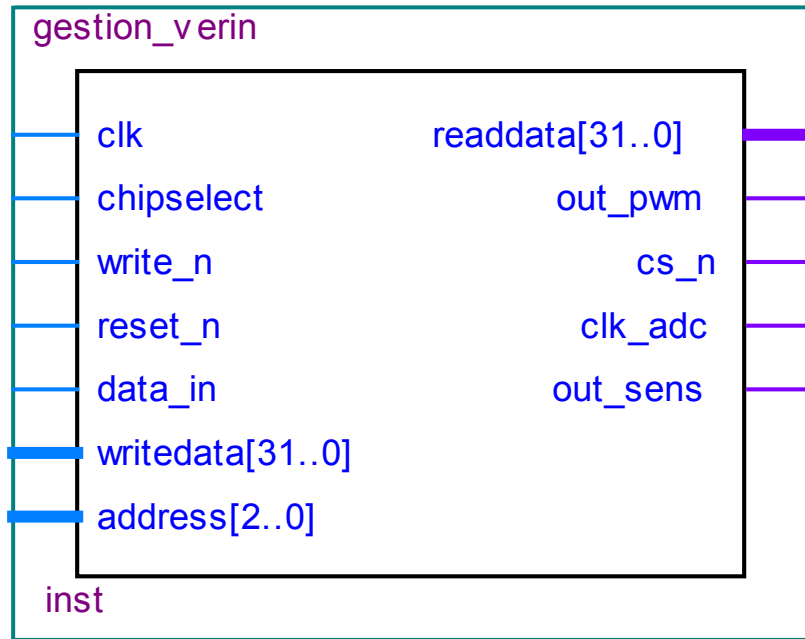
Duty : fixe le rapport cyclique (exemple si $\text{duty}=\text{freq}/2$ alors le rapport cyclique est de 50%)

Butee_g et **butee_d** : réglables de 0 à 4095 elles fixent les valeurs limites que le vérin (angle_barre) ne doit pas dépasser. En dehors de ces valeurs, le moteur est automatiquement coupé.

Ces valeurs sont mises à 0 par défaut et doivent être initialisées au démarrage du système.

Config : registre de configuration du circuit. La description du rôle de chaque bit est donnée dans le tableau.

Angle_barre : il donne la valeur de l'angle de barre codée sur 12 bits (résultat de la conversion analogique numérique)..



Symbole du circuit gestion_verin seul

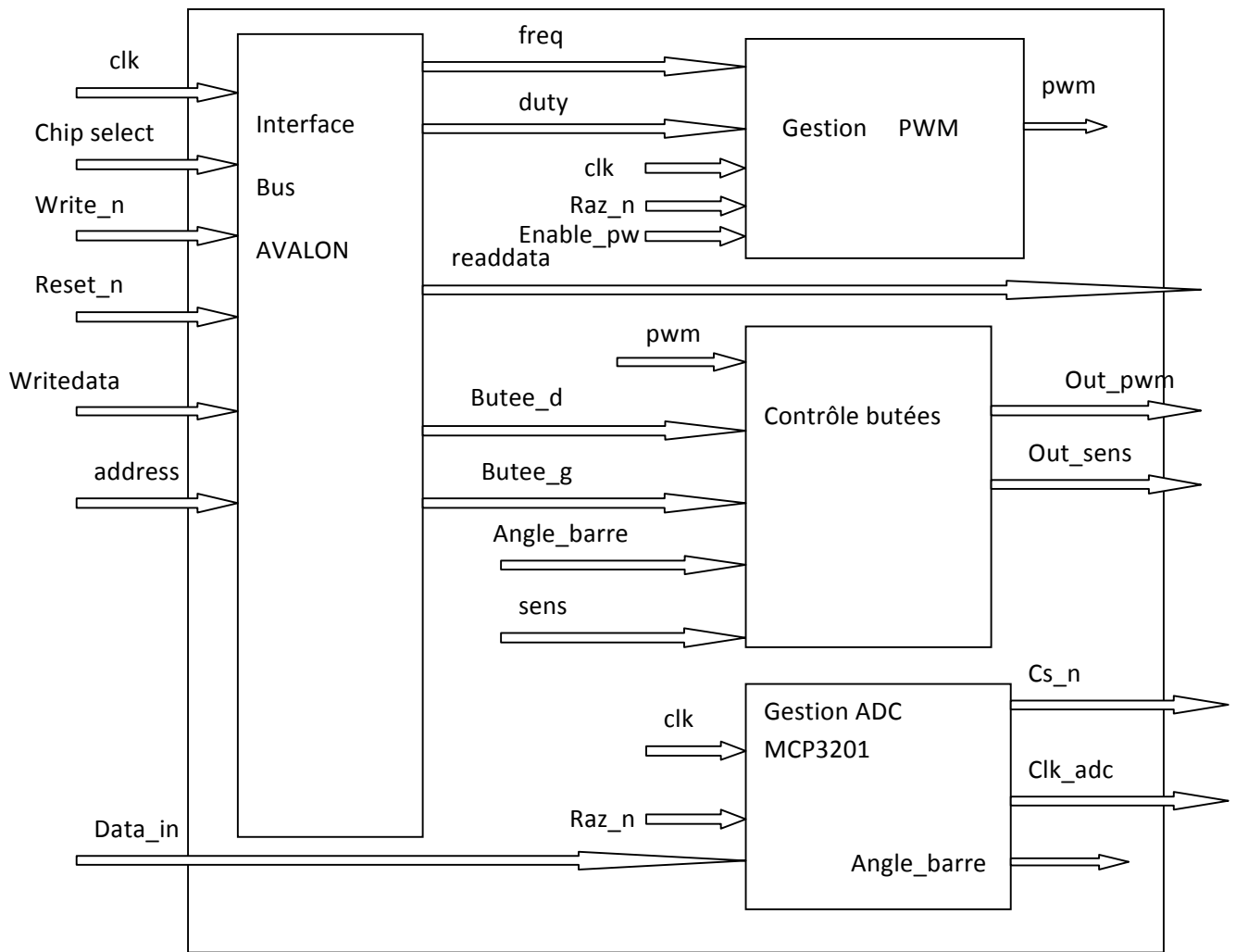


Schéma fonctionnel du circuit gestion_verin

Exemple de programme de mise en œuvre :

```

#include "alt_types.h"
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "unistd.h"
#include "stdio.h"

#define control (char *) LEADS_BASE
#define code_fonction (char *) SWITCHS_BASE
#define compas (int *) COMPAS_BASE
#define butee_d (int *) (GESTION_VERIN_0_BASE+12)
#define butee_g (int *) (GESTION_VERIN_0_BASE+8)
#define freq (int *) GESTION_VERIN_0_BASE
#define duty (int *) (GESTION_VERIN_0_BASE+4)
#define config (int *) (GESTION_VERIN_0_BASE+16)
#define angle_barre (int *) (GESTION_VERIN_0_BASE+20)

int main()
{
    unsigned int a,c,d;
    unsigned char b;

```

```

printf("Hello from Nios II!\n");
*control=(*control) | 3;//active circuits gestion_bp et gestion_compas
*butee_d=1320;
*butee_g=410;
*freq= 2000;
*duty=1500;
*config=1; // circuit gestion_verin actif, sortie pwm inactive

while (1)
{
//test bp en mode manuel seul
b=*code_fonction;
printf("code_fonction= %d\n", b);
switch(b)
{
case 0: *config=1;break;
case 1: *config=7;break;
case 2: *config=3;break;
default:*config=1;
}
a=((*compas)-10)&511;
printf("compas= %d\n", a);
c=*freq;
printf("freq= %d\n", c);
d=*duty;
printf("duty= %d\n", d);
c=*butee_d;
printf("butee_d= %d\n", c);
d=*butee_g;
printf("butee_g= %d\n", d);
c=*config;
printf("config= %d\n", c);
d=*angle_barre;
printf("angle_barre= %d\n", d);
usleep(100000);
}

return 0;
}

```

2) Spécifications circuit interface commandes et indications barreur

Voir la documentation du pilote pour les modes de fonctionnement. Le mode conservateur d'allure ne sera pas implémenté.

Principe du circuit : chaque action sur les boutons poussoirs entraîne l'apparition d'un code sur la sortie code_fonction. Seuls les codes répertoriés ci-dessous seront utilisés.

```

__*****
--      module gestion des boutons poussoirs
__*****
-- entrées: BP_Babord,BP_Tribord, BP_STBY, clk_50M, raz_n
-- sorties: codeFonction, ledBabord, ledTribord,ledSTBY, out_bip
__*****

```

```

--clk_50M: horloge à 50MHz
-- raz_n: actif à 0 => initialise le circuit
-- valeurs de codeFonction:
-- =0000: pas d'action, le pilote est en veille
-- =0001: mode manuel action vérin babord
-- =0010: mode manuel action vérin tribord
-- =0011: mode pilote automatique/cap
-- =0100: incrément de 1° consigne de cap
-- =0101: incrément de 10° consigne de cap
-- =0111: décrétement de 1° consigne de cap
-- =0110: décrétement de 10° consigne de cap
__*****

```

L'interface Avalon dispose de 2 registres tels que décrits ci-dessous :

<i>Registre</i>	<i>adresse</i>	<i>type</i>	<i>Bits concernés</i>
<i>config</i>	<i>0</i>	<i>R/W</i>	<i>b0=raz_n</i>
<i>Code</i>	<i>1 (4)</i>	<i>R/W</i>	<i>b3..b0= Code_Fonction</i>

3) Spécifications circuit acquisition vitesse vent

```

*****
-- module gestion_anemometre
__*****
--entrées:
--clk_50M : hologe 50MHz
--raz_n: rest actif à 0 => initialise le circuit
--in_freq_anemometre: signal de fréquence variable de 0 à 250 HZ
--continu : si=0 mode monocoup, si=1 mode continu
-- en mode continu la donnée est rafraîchie toute les secondes
--start_stop: en monocoup si=1 démarre une acquisition, si =0
-- remet à 0 le signal data_valid
__*****
-- sorties:
-- data_valid: =1 lorsque une mesure est valide
-- est remis à 0 quand start_stop passe à 0
-- data_anemometre : vitesse vent codée sur 8 bits
__*****

```

L'interface Avalon dispose de 2 registres tels que décrits ci-dessous :

<i>Registre</i>	<i>adresse</i>	<i>type</i>	<i>Bits concernés</i>
<i>config</i>	<i>0</i>	<i>R/W</i>	<i>b2=Start/Stop, b1=continu, b0=raz_n</i>
<i>Code</i>	<i>1 (4)</i>	<i>R/W</i>	<i>b9=valid, b7..b0= data_anemometre</i>

4) Spécifications circuit interface NMEA (RS232)

Mode émission :

Envoi d'une trame à 4800 bauds, 1 start, 1 stop, pas de parité et constituée de quatre octets codés ASCII : SCDU

S : signifie le début du message

C DU: centaines, dizaines et unités de degrés correspondant à l'angle de barre.

La trame est émise avec une périodicité d'une seconde (à tester).

Un signal start/stop=1 démarre la transmission, =0 arrête la transmission et revient au repos.

Un signal fin_transmit=1 indique que la transmission est terminée.

Un signal raz_n=0 inhibe le circuit.

L'interface Avalon dispose de 5 registres tels que décrits ci-dessous :

<i>Registre</i>	<i>adresse</i>	<i>type</i>	<i>Bits concernés</i>
<i>config</i>	<i>0</i>	<i>R/W</i>	<i>b2=fin_transmit, b1=Start/Stop, b0=raz_n</i>
<i>synchro</i>	<i>1 (4)</i>	<i>R/W</i>	<i>b7..b0=code ASCII carac. synchro</i>
<i>centaine</i>	<i>2 (8)</i>	<i>R/W</i>	<i>b7..b0=code ASCII centaine</i>
<i>dizaine</i>	<i>3 (12)</i>	<i>R/W</i>	<i>b7..b0=code ASCII dizaine</i>
<i>unité</i>	<i>4 (16)</i>	<i>R/W</i>	<i>b7..b0=code ASCII unité</i>

Mode réception:

Réception d'une trame à 4800 bauds, 1 start, 1 stop, pas de parité, ayant le même format que l'émission (SCDU).

Un signal mode=1 fait fonctionner le circuit en continu, =0 le récepteur dépend de l'état du signal start/stop.

Le signal start/stop=1 active la réception, =0 arrête la réception et revient au repos.

Un signal data_valid=1 indique que la réception est terminée.

Un signal raz_n=0 inhibe le circuit.

L'interface Avalon associée dispose de 5 registres tels que décrits ci-dessous :

<i>Registre</i>	<i>adresse</i>	<i>type</i>	<i>Bits concernés</i>
<i>config</i>	<i>0</i>	<i>R/W</i>	<i>b2=mode, b1=Start/Stop, b0=raz_n</i>
<i>synchro</i>	<i>1 (4)</i>	<i>R/W</i>	<i>b7..b0=code ASCII 1er caractère</i>
<i>centaine</i>	<i>2 (8)</i>	<i>R/W</i>	<i>b7..b0=code ASCII 2^{ème} caractère</i>
<i>dizaine</i>	<i>3 (12)</i>	<i>R/W</i>	<i>b7..b0=code ASCII 3^{ème} caractère</i>
<i>unité</i>	<i>4 (16)</i>	<i>R/W</i>	<i>b7..b0=code ASCII 4^{ème} caractère</i>

5) Spécifications circuit interface acquisition cap

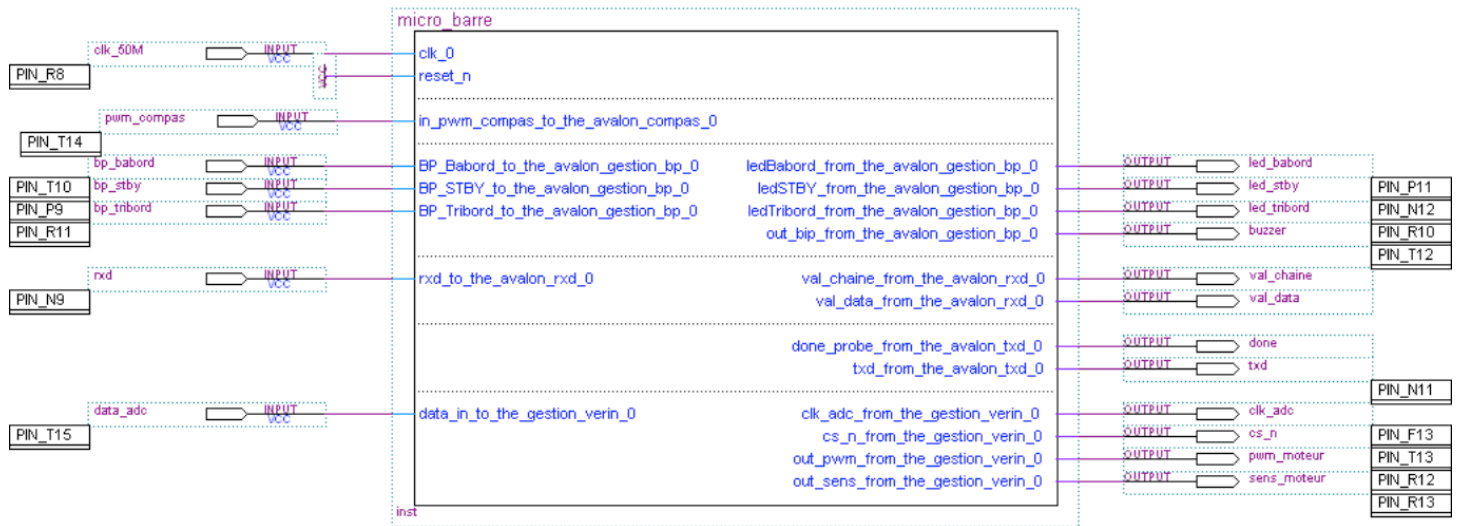
```

_*****
-- module gestion_compas pour boussole CMPS03 ou CMPS10
_*****
--entrées:
--clk_50M : hologe 50MHz
--raz_n: reset actif à 0 => initialise le circuit
--in_pwm_compas: signal PWM de la boussole, durée varie de 1ms à 36,9ms
--continu : si=0 mode monocoup, si=1 mode continu
-- en mode continu la donnée est rafraîchie toute les secondes
--start_stop: en monocoup si=1 démarre une acquisition, si =0
-- remet à 0 le signal data_valid
_*****
-- sorties:
-- data_valid: =1 lorsque une mesure est valide
-- est remis à 0 quand start_stop passe à 0
-- out_1s : signal de contrôle du top seconde (normalement pas utilisé)
-- data_compas : valeur du cap réel exprimé en degré codé sur 9 bits
_*****

```

L'interface Avalon dispose de 2 registres tels que décrits ci-dessous :

<i>Registre</i>	<i>adresse</i>	<i>type</i>	<i>Bits concernés</i>
<i>config</i>	<i>0</i>	<i>R/W</i>	<i>b2=Start/Stop, b1=continu, b0=raz_n</i>
<i>Compas</i>	<i>1 (4)</i>	<i>R/W</i>	<i>b9=valid, b8..b0= data_compas</i>



Symbole du SOPC complet version carte DE0 Nano