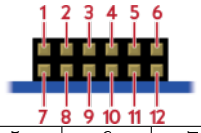
IUT de TROYES — ***DS Final XR602*** — *G. MILLON*
Département GEII — Durée : 3h00 — *ESE - S6 - P24*

Présentation du DS : On considère les exercices mis en oeuvre en TP sur la base du soft-processeur NIOS auxquels on ajoute les connaissances et expériences du module XR6.02. On souhaite mettre en oeuvre le module PMODNav regroupant 2 composants : le LPS25HB et le LSM9DS1. Le LSM9DS1 est un accelerometre/gyroscope/magnetometre 3 axes. Le LPS25HB est un barometre. La mesure de la pression atmosphérique associée à celle de la température que permet le LPS25HB, conduit si on le souhaite à une évaluation de l'altitude via un calcul.
On se propose d'exploiter le LPS25HB du module PMODNAV via le processeur NIOS.

Documentation résumée : La documentation du module PMODNAV indique les caractéristiques suivantes :

- Alimentation : VCC=3,3V
- Bus de communication : SPI avec $F_{clk} < 10MHz$, donnée au format octet transmis en MSBFirst, ClockPolarisation =1, ClockPhase = 1 , entrées Chip Select actives à l'état bas
- Brochage : vue de face des broches du module PMODNav. A l'arriere de cette vue, le circuit imprimé avec ses composants au-dessus



broche	1	2	3	4	5	6	7	8	9	10	11	12
signal	CS_A/G	MOSI	MISO	SCK	GND	VCC	INT	DRDY_M	CS_M	CS_ALT	GND	VCC

On forcera en permanence les signaux CS_A/G et CS_M à '1' pour inhiber l'accéléromètre/gyroscope/magnetometre. On réalisera ce forçage directement en VHDL. Les signaux INT et DRDY_M ne seront pas exploités.

- Registres du LPS25HB : on relève ici les seuls registres dont nous aurons besoin pour un usage initial "simplifié".

registres	whoami	config	ctrl1	ctrl2	status	press_L	press_H	temp_L	temp_H
adresse	0x0F	0x10	0x20	0x21	0x27	0x29	0x2A	0x2B	0x2C

- Usage "simplifié" du LPS25HB :
  - on peut lire le registre ***whoami*** pour vérifier que la communication en SPI est opérationnelle. La lecture de ce registre doit délivrer la valeur constante ***0xBD***.
  - On configure/paramètre le LPS25HB en écrivant la valeur ***0x80*** dans le registre ***ctrl1*** et la valeur ***0x0A*** dans le registre de ***config***. Cela conduit à un mode de mesure du LPS25HB en OneShot. Il faut donc lancer une mesure puis détecter si la mesure a été réellement effectuée, conduisant à une donnée effectivmeent disponible.
  - Le lancement d'une mesure se fait en écrivant la valeur 0x01 dans ***ctrl2***.
  - La détection d'une mesure de température disponible se fait en lisant le registre ***status***. Si le bit de ***rang 0*** du registre lu est à ***1*** (NDLR le LSB de l'octet lu) alors une donnée de mesure de température est disponible. Si le bit de rang 0 du registre lu est à 0 , alors la donnée de mesure de température n'est pas encore disponible.
  - La détection d'une mesure de pression disponible se fait en lisant le registre ***status***. Si le bit de ***rang 1*** du registre lu est à ***1*** , une donnée de mesure de pression est disponible. Si le bit de rang 0 du registre lu est à 0 (NDLR le LSB de l'octet lu) il n'y a pas encore de donnée de mesure de pression disponible.
  - Lorsque la disponibilité d'une mesure est validée, on peut lire cette donnée :
    - la lecture de la température consiste à lire les 2 registres ***temp_L*** et ***temp_H*** pour reconstituer un mot de 16 bits (NDLR : temp_H pour la partie poids fort, temp_L pour la partie poids faible). On appelle ***temp*** ce mot de 16 bits dans ce qui suit.
    - la lecture de la pression consiste à lire les 2 registres ***press_L*** et ***press_H*** pour reconstituer un mot de 16 bits avec le même principe de reconstitution que pour la température. On appelle ***press*** ce mot de 16 bits dans ce qui suit.
  - La température en degré celsius est alors calculée à partir du mot de 16 bits ainsi reconstitué. Pour cela, on force à 0 le bit MSB de ***temp***. Ensuite, la formule de calcul est $température = \frac{temp}{480} - 24.25$. L'offset de 24.25 est approximatif.

– La pression atmosphérique en mm de mercure est obtenue en appliquant la formule de calcul suivante : $pression = \frac{press}{16}$.

- aspects logiciels : la mise en oeuvre du périphérique SPI du NIOS dans Platform Designer conduit à la mise à disposition d'une librarie pour pouvoir l'exploiter au sein d'un programme en langage C. Cette librairie propose une fonction unique (alt_avalon_spi_command) permettant de réaliser aussi bien les lectures, que les écritures en SPI. Les listing ci-dessous illustrent l'usage de cette fonction suivis d'un commentaire explicatif .

```
// cas écriture en SPI sur PMODNAv
alt_u8 read_length; // variable pour la lecture
alt_u8 reg=0x25; // adresse d'un registre sur lequel on veut écrire
alt_u8 value=0x01; // valeur à écrire dans le registre
alt_u8 buffer_w[2]; // tableau stockant les octets pour l'écriture
buffer_w[0]=reg; // initialisation du tableau
buffer_w[1]=value;
read_length = alt_avalon_spi_command(SPI_0_BASE,0,2,buffer_w,0,0,0); // instruction/fonction appelée
usleep(1); // petit temps d'attente de 1µs après écriture
```

Le format de la fonction alt_avalon_spi_command demande, dans cet ordre, les paramètres d'entrée suivant :

- l'adresse du périphérique SPI du point de vue du processeur NIOS : *SPI_0_BASE* . On retrouvera cela si nécessaire dans le fichier *system.h*
- le numéro du bit Slave Select à positionner : *0* . Dans la mesure où on s'est limité à 1 seul composant sur le bus SPI dans notre exercice, ce sera toujours le premier et seul signal de type slave select, qui porte donc le numéro 0 , pour notre signal CS_ALT (les autres signaux de type slave select tels que CS_M et CS_A/G ont été figés à 1 en VHDL, et ne sont donc pas pilotés par notre périphérique SPI).
- le nombre d'octets à exploiter pour l'écriture : ici *2*
- le tableau où trouver les octets à exploiter pour l'écriture : *buffer_w*
- le nombre d'octets à lire pour la lecture : ici *0* . On ne lit pas de données dans cet exemple
- le tableau où enregistrer les octets lus : ici on positionne à *0* car on ne lit pas de données dans cet exemple.
- et enfin, une valeur qu'on laissera à *0 systématiquement*.
- la variable de retour *read_length* est présente pour satisfaire le format de la fonction, mais ne sera pas exploitée puisqu'on ne fait que écrire.

Cette instruction écrit donc en SPI sur le périphérique ciblé la valeur 0x01 dans le registre 0x25 du composant.

```
// cas lecture en SPI sur PMODNav
int read_length; // nombre d'octets lus
alt_u8 reg=0x22; // adresse d'un registre
alt_u8 buffer_w[2]; // tableau stockant les octets pour l'écriture
alt_u8 buffer_r[1]; // tableau stockant les octets lus ( 1 seul ici)
buffer_w[0]=reg | 0x80; // forçage à 1 du bit MSB pour une phase de lecture du registre
read_length = alt_avalon_spi_command(SPI_0_BASE, composant,1,buffer_w,1,buffer_r,0);
```

Le format de la fonction alt_avalon_spi_command demande, dans cet ordre, les paramètres d'entrée suivant :

- l'adresse du périphérique SPI du point de vue du processeur NIOS : *SPI_0_BASE* . On retrouvera cela si nécessaire dans le fichier *system.h*
- le numéro du bit Slave Select à positionner : *0* . Dans la mesure où on s'est limité à 1 seul composant sur le bus SPI dans notre exercice, ce sera toujours le premier et seul signal de type slave select, qui porte donc le numéro 0 , pour notre signal CS_ALT (les autres signaux de type slave select tels que CS_M et CS_A/G ont été figés à 1 en VHDL, et ne sont donc pas pilotés par notre périphérique SPI).
- le nombre d'octets à exploiter pour l'écriture : ici *1 , pour désigner par 1 écriture sur le composant, l'adresse du registre que l'on souhaite lire.*

- le tableau où trouver les octets à exploiter pour l'écriture : *buffer_w*
- le nombre d'octets à lire pour la lecture : ici *1* . On ne lit qu'1 registre donc qu'1 octet
- le tableau où enregistrer les octets lus : *buffer_r*
- une valeur qu'on laissera à *0 systématiquement* .
- la variable de retour *read_length* est présente pour satisfaire le format de la fonction. Cette variable devrait contenir 1 à l'issue de l'execution de l'instrcution puisqu'on devrait avoir lu 1 octet.

Cette instruction lit donc, en SPI sur le périphérique ciblé, 1 octet : la valeur contenu dans le registre d'adresse 0x22 du composant.

Question 1 : câbler le module PMODNav à la carte DE10LITE sur des broches du connecteur HE40 en notant/relevant les correspondances bornes FPGA - signal PMODNav. On pourra éventuellement s'appuyer sur le câblage déjà réalisé dans un exercice pour le module PMODAls. ***Faire valider votre câblage par l'enseignant.***

Question 2 : Reprendre un projet quartus, issu des séances de TP, doté d'un processeur NIOS implanté avec au moins, la gestion des afficheurs 7 segments et le périphérique SPI. Par exemple, l'exercice mettant en oeuvre le module PMODALS doit convenir. Ouvrir la description du NIOS avec Platform Designer et vérifier, et adapter si nécessaire, le paramétrage du périphérique SPI pour qu'il soit conforme à la documentation du module PMODNav. On choisira un périphérique MAster, une horloge à 1,25MHz et un unique signal Slave Select à piloter. Et on règlera le protocole SPI selon la documentation du LPS25HB ci-dessus. Compiler la description du NIOS et ses périphériques dans Platform Designer. ***Faire valider par l'enseignant , la récupération de ce projet, les réglages du périphérique SPI et la compilation sous platforme designer.***

Question 3 : Apporter les modifications au projet sous quartus pour prendre en compte le câblage du module PMODNav. Cela concerne notamment les signaux SCK, MOSI, MISO, SS et les divers signaux CS du module PMODNav. Et donc par suite, il conviendra de reprendre l'outil pinplaner..., . ***Compiler la description VHDL complète sous Quartus et faire valider par l'enseignant. On pourra alors télécharger l'application pour sa partie matérielle sur la carte DE10.***

Question 4 : Démarrer Nios II Software Tools for Eclipse . Mettre en place un programme C que l'on pourra éxécuter. Eventuellement sur la base du projet récupéré, le but sera en premier lieu de piloter les afficheurs 7 segments. On réalisera une boucle infinie comptant les itérations et mettant à jour environ à chaque seconde l'affichage de l'itération sur les afficheurs 7 segments. On positionnera ce numéro d'itération sur les 2 digits de gauche parmi les 6. Les autres digit afficheront 0 dans cette partie. Compiler le programme C correspondant à ce cahier des charges et télécharger le programme pour l'executer sur la DE10 Lite. ***Faire valider par l'enseignant le fonctionnement de ce programme.***

Question 5 : Modifier le programme C pour réaliser la lecture du registre ***whoami*** et afficher la valeur qu'il renvoie sur les 2 digits de droite des afficheurs 7 segments. On pourra ajouter cela avant la boucle infinie mise en place dans le programme de la question 4. Compiler, télécharger et ***Faire vérifier et valider le bon fonctionnement par l'enseignant.***

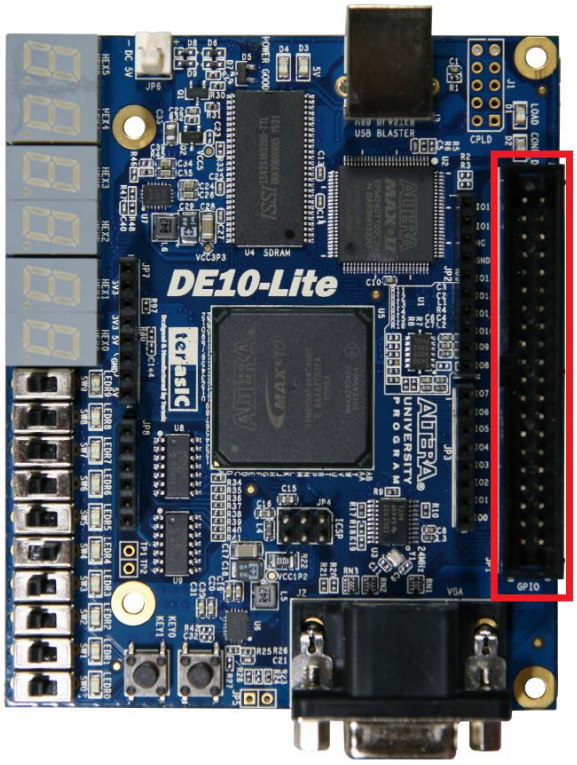
Question 6 : Modifier à nouveau le progamme au sein de la boucle infinie pour y ajouter la lecture de la temperature. On suivra les informations de la documentation pour lancer une mesure, vérifier/attendre la disponibilité de celle-ci, récupérer les 2 octets et calculer la valeur de la temperature puis l'afficher. La vérificaton de la disponibilité pourra consister à reboucler la lecture du registre status jusqu'à 1000 fois en plaçant une petite temporisation pour attendre que la mesure

Question 7 : Ajouter la mesure le pression au programme et gérer l'affichage de celle-ci en remplacement de la température selon le positionnement d'un interrupteur.

3. 5 Using 2x20 GPIO Expansion Headers

The board has one 40-pin expansion headers. Each header has 36 user pins connected directly to the MAX 10 FPGA. It also comes with DC +5V (VCC5), DC +3.3V (VCC3P3), and two GND pins. Both 5V and 3.3V can provide a total of 5W power.

Figure 3-18 shows the related schematics. Table 3-7 shows the pin assignment of GPIO headers.



GPIO (JP1)

PIN_V10	GPIO_[0]	1		2	GPIO_[1]	PIN_W10
PIN_V9	GPIO_[2]	3		4	GPIO_[3]	PIN_W9
PIN_V8	GPIO_[4]	5		6	GPIO_[5]	PIN_W8
PIN_V7	GPIO_[6]	7		8	GPIO_[7]	PIN_W7
PIN_W6	GPIO_[8]	9		10	GPIO_[9]	PIN_V5
	5V	11		12	GND	
PIN_W5	GPIO_[10]	13		14	GPIO_[11]	PIN_AA15
PIN_AA14	GPIO_[12]	15		16	GPIO_[13]	PIN_W13
PIN_W12	GPIO_[14]	17		18	GPIO_[15]	PIN_AB13
PIN_AB12	GPIO_[16]	19		20	GPIO_[17]	PIN_Y11
PIN_AB11	GPIO_[18]	21		22	GPIO_[19]	PIN_W11
PIN_AB10	GPIO_[20]	23		24	GPIO_[21]	PIN_AA10
PIN_AA9	GPIO_[22]	25		26	GPIO_[23]	PIN_Y8
PIN_AA8	GPIO_[24]	27		28	GPIO_[25]	PIN_Y7
	3.3V	29		30	GND	
PIN_AA7	GPIO_[26]	31		32	GPIO_[27]	PIN_Y6
PIN_AA6	GPIO_[28]	33		34	GPIO_[29]	PIN_Y5
PIN_AA5	GPIO_[30]	35		36	GPIO_[31]	PIN_Y4
PIN_AB3	GPIO_[32]	37		38	GPIO_[33]	PIN_Y3
PIN_AB2	GPIO_[34]	39		40	GPIO_[35]	PIN_AA2

Figure 3-18 I/O distribution of the expansion headers