PG208
Programmation Orientée Objet / Langage C++
Projet: Couche d'abstraction pour interface matérielle

Y. Bornat A. Valade

March 12, 2025

L'objectif de ce projet est de proposer une biblbiothèque pour faciliter l'accès à une carte matérielle. C'est à dire, créer une classe (ou un ensemble de classes) qui cacheront à l'utilisateur toute la complexité des accès matériels.

Ben oui, en informatique, à chaque fois que avez eu l'impression que quelque chose est/était facile, c'est parce que quelqu'un est passé avant vous pour gérer la complexité. Maintenant, c'est votre tour.

Ce sujet comporte plusieurs parties. La première, le *comportement de base* correspond à l'objectif minimal du cours/projet et devrait être fonctionnel en deux à quatre heures. Les parties suivantes sont des compléments que vous pouvez ajouter selon votre ambition, votre niveau, vos envies, votre signe astrologique, votre couleur préférée ou la vitesse du vent... Vous ne pourrez pas *tout* faire. Privilégiez les parties qui vous permettront de montrer votre savoir-faire efficacement. Si c'est trop facile pour vous, vous perdez votre temps, si c'est trop difficile, vous prenez des risques...

Ce thème de projet n'a pas la prétention de vous montrer l'usage courant des classes dans la programmation objet. Pour cela il aurai fallu gérer une bibliothèque contenant des livres, des CDs, ... Ou alors un système de gestion de ressources humaines... Par contre, avec ce projet, vous serez contraints d'écrire de nombreuses classes et méthodes, et c'est quand même ça qui est le plus important.

1 Environement de travail

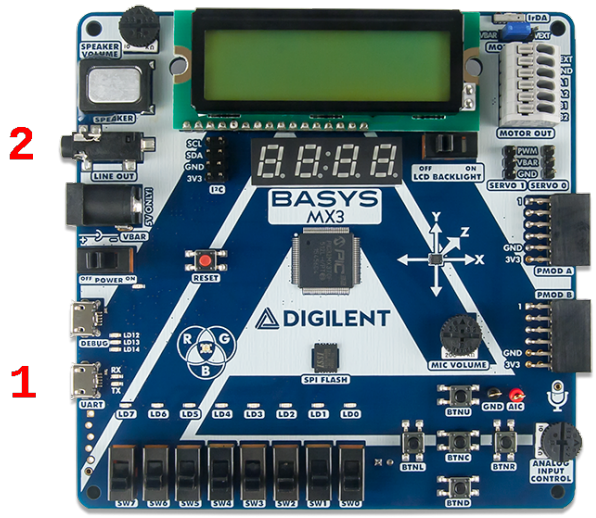

Figure 1: La carte *basys MX3*

L'accès au matériel consiste à communiquer avec la carte *BASYS MX3*. Il s'agit d'une carte de TP micro-contrôleur qui n'est plus utilisée car elle est en obsolescence et que l'environement de développement pose maintenant problème sur les machines de l'école.

La liaison avec la carte *BASYS MX3* se fait par une lisaison UART encapsulée dans une connexion USB, telle que vous l'avez déjà utilisée en TP microcontrôleur STM32 ou en TP d'architecture numérique pour communiquer avec le FPGA. Cette carte doit être connectée à l'ordinateur par un câble micro-USB. Attention à bien utiliser le connecteur idéntifié UART (celui qui est proche des switchs, référencé 1 sur la figure 1)

Les échanges se font par un protocole détaillé en section 1.2 page 5. Ce protocole permet d'écrire ou lire des valeurs dans des registres. Chaque registre correspondant à une donnée ou à une fonction précise.

Les parties 1.1.1 et 1.2 ne sont pas indispensables à la compréhension du sujet, vous pouvez les survoler lors de votre première lecture et y revenir plus tard.

1.1 Fonctionnalités disponibles

1.1.1 Registres de la carte

Le tableau 1 rassemble les registres disponibles sur la carte. L'accès précise si le registre concerné est en lecture seule, écriture seule ou lecture écriture. Une lecture sur un registre en écriture seule renverra toujours la valeur 0.

L'accès à la plupart des fonctions est assez intuitif. Pour les switchs ou les LEDs, chaque bit correspond à un switch ou une LED. Pour la LED *RGB*, chacune des trois valeurs code la composante de couleur. Vous trouverez ci-après quelques précisions concernant certains périphériques.

1.1.2 L'écran LCD

Pour utiliser l'écran LCD, il suffit d'écrire dans l'un des 32 registres associés. Chaque registre correspond à un caractère visible à l'écran. Ainsi les adresses `MX3ADDR_LCD_START` à `MX3ADDR_LCD_START+15` correspondent aux caractères de la ligne du haut, et les adresses `MX3ADDR_LCD_START+16` à `MX3ADDR_LCD_START+31` à la ligne du bas.

1.1.3 Registres GPR

Les registres identifiés *GPR* sont des registres généraux sans signification particulière. Il peuvent être assimilés à de la RAM.

1.1.4 La sortie audio

La carte *basys MX3* est dotée d'une sortie audio (haut parleur ou sortie jack, référencée 2 sur la figure 1) dont le niveau sonore est réglable (potentiomètre). La valeur de sortie est définie sur 8 bits par échantillon, à une fréquence de 11,025 KHz (défaut) ou de 22,05KHz. Les échantillons sont non signés. Hstoire de simplifier l'usage, il n'y a que 3 registres à manipuler (dont un registre 16 bits qui occupe donc deux adresses).

- Registre de *status* : seuls les deux bits de poids faible sont utiles pour la sortie audio. Le bit 0 permet d'activer (1) ou désactiver (0) la sortie. Le bit 1 permet de définir la vitesse d'échantillonnage à 22,0kHz (1) ou à 11,02kHz (0).
- `MX3ADDR_SND_WFIFO` est le point d'entrée d'une File d'attente (FIFO) de 64ko. Si la sortie audio est activée (bit 0 du *status* à 1), les élements de la FIFO sont consommés au rythme de l'échantillonnage sonore. Sinon, les échantillons attendent.
- `MX3ADDR_SND_ELTS_L` / `MX3ADDR_SND_ELTS_H` contiennent le nombre d'échantillons stockés dans la FIFO. Étant donné que la FIFO a une taille de 64ko, le nombre d'échantillons est sur 16 bits, et la valeur est accessible sur deux registres (`L` pour *low*, l'octet le moins significatif et `H` pour *high*, l'octet le plus significatif)

Petit détail : La communication avec la carte se fait via une liaison UART à 115 200 bps, c'est assez juste pour envoyer les données, l'utilisation du son est peu compatible avec d'autres applications.

Dec	Hex	ID	R/W	fonction
00	00	MX3ADDR_TEST	RO	Valeur de test : 75
01	01	MX3ADDR_SW	RO	Valeur des switches
02	02	MX3ADDR_LED	R/W	Valeur des LEDs
03	03			
04	04	MX3ADDR_FLASH_RFIFO	RO	Lecture Incrémentale de la mémoire Flash
05	05	MX3ADDR_FLASH_PTR_L	R/W	Index de lecture de la mémoire Flash, remis à jour si lecture dans MX3ADDR_FLASH_RFIFO. Si modifié, la nouvelle valeur n'est prise en compte qu'à l'écriture de MX3ADDR_FLASH_PTR_H
06	06	MX3ADDR_FLASH_PTR_M	R/W	
07	07	MX3ADDR_FLASH_PTR_H	R/W	
08	08	MX3ADDR_RGBLED_R	R/W	Composante rouge pour la LED RGB
09	09	MX3ADDR_RGBLED_G	R/W	Composante verte pour la LED RGB
10	0A	MX3ADDR_RGBLED_B	R/W	Composante bleue pour la LED RGB
11	0B	MX3ADDR_SND_WFIFO	WO	Écriture incrémentale des échantillons sonores
12	0C	MX3ADDR_SND_ELTS_L	RO	Nombre d'échantillons sonores présents dans la FIFO
13	0D	MX3ADDR_SND_ELTS_H	RO	
14	0E	MX3ADDR_UPTIME_L	RO	Nombre de secondes écoulées depuis la derniere initialisation de la carte
15	0F	MX3ADDR_UPTIME_H	RO	
16	10	MX3ADDR_7SEG_DEC_L	R/W	Valeur de l'afficheur à 7 segments. Ces registres activent le mode décimal.
17	11	MX3ADDR_7SEG_DEC_H	R/W	
18	12	MX3ADDR_7SEG_HEX_L	R/W	Valeur de l'afficheur à 7 segments. Ces registres activent le mode hexadécimal.
19	13	MX3ADDR_7SEG_HEX_H	R/W	
20	14	MX3ADDR_7SEG_MAP_0	R/W	Manipulation fine des afficheurs à 7 segments Chaque registre identifie un afficheur, chaque bit contrôle un segment. Ces registres activent le mode 'map'.
21	15	MX3ADDR_7SEG_MAP_1	R/W	
22	16	MX3ADDR_7SEG_MAP_2	R/W	
23	17	MX3ADDR_7SEG_MAP_3	R/W	
24	18	MX3ADDR_B_IMAT_LL	RO	Numéro d'identification individuel de la carte. Le numéro de série hexadécimal figure également sur l'étiquette au dos de la carte.
25	19	MX3ADDR_B_IMAT_LH	RO	
26	1A	MX3ADDR_B_IMAT_HL	RO	
27	1B	MX3ADDR_B_IMAT_HH	RO	
28	1C			
29	1D			
30	1E			
31	1F	MX3ADDR_STATUS	R/W	Registre de Statut de la carte. Cf tableau 3 pour les détails.
32	20	MX3ADDR_LCD_START	R/W	Caractères affichés sur l'écran LCD
...	...	-	...	
63	3F	-	R/W	
64	40	GPR	R/W	Registres génériques, non ratachés au matériel, peuvent être utilisés comme RAM
...	...		...	
255	FF	-	R/W	

Table 1: Tableau descriptif des registres accessibles sur la carte Basys MX3

1.1.5 La mémoire Flash

La carte est équipée d'une mémoire Flash de 4Mo. L'usage est assez basique : il y a un registre d'adresse sur 22 bits et un registre de lecture. Pour lire une donnée de la mémoire Flash, il suffit d'écrire l'adresse de lecture (dans la mémoire Flash) sur les registres MX3ADDR_FLASH_PTR_L, MX3ADDR_FLASH_PTR_M et MX3ADDR_FLASH_PTR_H. La valeur contenue à cette adresse ets alors accessible dans le registre MX3ADDR_FLASH_RFIFO. Une fois la valeur lue, le registre d'adresse est incrémenté, il est donc possible de lire plusieurs fois MX3ADDR_FLASH_RFIFO pour récupérer le contenu de la mémoire.

Techniquement, il n'y a pas de FIFO derrière le registre MX3ADDR_FLASH_RFIFO, contrairement à ce qu'il se passe pour la sortie sonore. Mais il porte le nom de FIFO car il se comporte de la même manière : chaque lecture consomme la donnée qui a été lue, et plusieurs lectures successives permettent de récupérer plusieurs données successives.

Lors de la modification du registre d'adresse, il est essentiel d'écrire entièrement la nouvelle adresse, car la mise à jour est effectuée à l'écriture dans le registre le plus significatif `MX3ADDR_FLASH_PTR_H`. Toute modification des deux autres registres sera annulée à la moindre lecture de `MX3ADDR_SND_WFIFO` à cause de l'incrémentation automatique.

Le contenu de la mémoire est fourni par le tableau 2. Il n'est pas possible d'écrire dans la mémoire Flash (Ou pour être plus rigoureux, le programme qui tourne sur le microcontrôleur ne le permet pas).

@ (Déc)	@ (Hex)	Taille	Description
0	0x000000	32	Texte de confirmation de lecture de la Flash
40	0x000028	7	Numéro de série de la carte en représentation ASCii Ce numéro est également disponible dans les registres `MX3ADDR_B_IMAT_XX` au format binaire
50	0x000032	1323000	Morceau musical de 60 secondes, au format signé, 8 bits par échantillon, échantillonné à 22,05kHz

Table 2: Contenu de la mémoire Flash

A partir du moment où on manipule la mémoire Flash, il faut devenir très précis dans le vocabulaire. En effet, vous serez confrontés aux adresses des registres de la carte (sur 8 bits), et aux adresses des données de la mémoire Flash (sur 22 bits). Sans compter qu'il y a aussi les adresses mémoire sur l'ordinateur. Bref, Les raccourcis de langage risquent fort de brouiller le message, plutôt que de l'expliquer.

1.1.6 Afficheur 7 segments

Il existe trois façon d'utiliser l'afficheur 7 segments.

- pour afficher une valeur décimale
- pour afficher une valeur hexadécimale
- pour afficher des symboles quelconques (mode *map*)

Dans les deux premiers modes, il suffit d'écrire une valeur non signée sur 16 bits. Selon que cette valeur est écrite aux adresses `MX3ADDR_7SEG_DEC_X` ou `MX3ADDR_7SEG_HEX_X`, cela activera le mode de représentation décimal ou hexadécimal respectivement. Mais il s'agit du même registre. Vous relirez donc la même valeur dans les deux séries de registres.

Dans le dernier mode (*map*), il est possible d'afficher n'importe quel symbol puisque chacun des quatre registres `MX3ADDR_7SEG_MAP_X` est associé à un afficheur, et que chaque bit de ces registres est associé à un segment particulier. La figure 1.1.6 indique quel segment est associé à quel bit. Écrire dans ces registres active le mode *map* automatiquement.

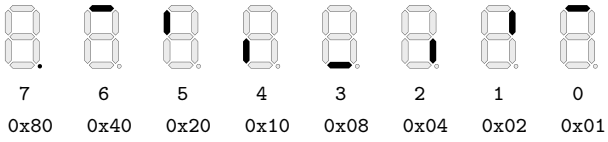

Figure 2: Répartition bit à bit des segments, la ligne spérrieure indique le numéro du bit associé au segent référencé. La ligne inférieure indique le masque correspondant

1.1.7 Registre de statut (*status*)

Parmis tous les registres du tableau 1, `MX3ADDR_STATUS` a un fonctionnement particulier car chacun de ses bits a une fonction spécifique. Le tableau 3 en précise la signification.

bit	ID	R/W	fonction
0	MX3BIT_STATUS_AUD_EN	R/W	0 : sortie son désactivée 1 : Sortie son active
1	MX3BIT_STATUS_AUD_FREQ	R/W	Fréquence d'échantillonage sonore : 0 : 22,05 kHz 1 : 11,025kHz
2-3	MX3BIT_STATUS_7SEG_MODE	R/W	Mode des afficheurs 7 segments : 00 : éteint 01 : map 10 : décimal 11 : hexadécimal
4-7			Réservés - Lus comme 0

Table 3: Tableau descriptif des bits du registre MX3ADDR_STATUS

1.2 Protocole de communication

Le protocole de communication avec la carte *BASYS MX3* consiste en une série de commandes suivies de leurs paramètres et éventuellement d'une réponse de la carte. Le tableau 4 donne la suite des commandes acceptées par la carte.

Par exemple, en envoyant successivement les valeurs 2, 2 et 255, on allume toutes les LEDs de la carte, mais il est beaucoup plus lisible d'écrire cette séquence comme MX3CMD_WR1, MX3ADDR_LED, 0xFF.

Pour chaque commande de lecture, le retour des données ne commence que quand le dernier paramètre a été envoyé.

Nom	ID	Params	Retour	Description
MX3CMD_NOP	0	-	-	Aucune action.
MX3CMD_TEST	1	-	<*data*>	Renvoie une valeur de test constante qui vaut MX3DAT_TESTCMD
MX3CMD_WR1	2	<*address*> <*data*>	-	Écrit la valeur <*data*> à l'adresse <*address*>
MX3CMD_RD1	3	<*address*>	<*data*>	Renvoie la valeur <*data*> contenue dans le regsitre à l'adresse <*address*>
MX3CMD_WR_FF	4	<*address*> <*n*> $n \times$ <*data*>	-	Écrit n valeurs successives à la même adresse <*address*>
MX3CMD_RD_FF	5	<*address*> <*n*>	$n \times$ <*data*>	Lit n valeurs successives à la même adresse <*address*>
MX3CMD_WR_MAP	6	<*address*> <*n*> $n \times$ <*data*>	-	Écrit n valeurs à plusieurs adresses successives, la première étant <*address*>
MX3CMD_RD_MAP	7	<*address*> <*n*>	$n \times$ <*data*>	Lit n valeurs à plusieurs adresses successives, la première étant <*address*>
MX3CMD_WR1MAP	8	<*address*> <*n*> <*data*>	-	Écrit n fois la valeur <*data*> aux adresses allant de <*address*> à <*address*>+n-1

Table 4: Tableau descriptif des bits du registre MX3BRD_STATUS

2 Comportement minmal

Le comportement minimal attendu est de fournir une classe MX3board qui permettra de manipuler facilement ces registres. Cette classe DOIT être fonctionnelle.

Le fichier *mx3basic.c* vous fournitt une base de départ pour communiquer avec la carte. Vous y trouverez les fonctions suivantes :

- `int board_open(char * filename)` pour ouvrir le port de communication
- `void board_d_write(int fd, unsigned char addr, unsigned char data)` pour écrire dans un registre
- `unsigned char board_d_read(int fd, unsigned char addr)` pour récupérer la valeur d'un registre
- `void board_close(int fd)` pour fermer le port de communication

Vous pouvez vérifier son bon fonctionnement en écrivant un programme de test (très) basique. Pour cela, rappelez vous que sous UNIX, pour accéder à la liaison de la carte, il faut aller dans le dossier */dev*. L'entrée pour accéder à la liaison série est *ttyUSB*XX, où XX représente un numéro, fort probablement 0 ou 1. Donc, pour tester le programme, il faut l'éxécuter avec l'argument */dev/ttyUSB0*.

2.1 Portage en C++

Reprenez le code fourni, et construisez la base de la classe `MX3board` pour gérer la carte. La classe aura au moins les propriétés suivantes :

- Le constructeur de la classe initialise (ouvre) la connexion vers la carte.
- La fermeture de la connexion se fait dans le destructeur pour garantir qu'elle ne sera pas oubliée.
- les méthodes de lecture et écriture n'ont pas besoin qu'on leur rappelle le descripteur de fichier (il est stocké en tant que propriété de la classe.)

Tant qu'on y est, séparez les codes en plusieurs fichiers :*mx3board.cpp* pour la classe elle-même, et main.cpp pour le programme de test, faites aussi un Makefile, ça vous simplifiera la vie.

ATTENTION: certains éditeurs de texte on la sale habitude de remplacer les Caractères de tabulation par des espaces. C'est pas de chance, parce que `make` a besoin de garder des caractères de tabulation.

2.2 Représentation des registres en tableau virtuel (lecture)

En surchargeant `operator[]` vous pouvez effectuer la lecture assez simplement. Si on considère que la méthode `unsigned char operator[](unsigned char addr)`, fait une lecture à l'adresse `addr`, alors le code suivant sera possible :

```
#include "mx3board"

int main(int argc, char * argv[]) {
    MX3board brd { argv[1] }
    std::cout << "Valeur des switchs : " << brd[MX3ADDR_SW] << std::endl;
    return 0;
}
```

Le souci de la surcharge de l'opérateur `[]`, c'est qu'il renvoie un `unsigned char`. Il n'est donc pas possible de faire une écriture avec la notation `brd[MX3ADDR_LED]=0x55;`. Le compilateur se plaint alors que `brd[MX3ADDR_LED]` n'est pas une *lvalue*, c'est à dire, quelque chose qu'on peut placer à gauche du signe `=`. Par opposition les *rvalue*, représentent ce qui peut se mettre à droite du signe `=`.

2.3 Écriture dans le tableau virtuel

Une solution, pour créer une *lvalue*, c'est de créer un classe possédant uen méthode `operator=`. Dans ce cas, la ligne `A=B;` est équivalente à `A.operator=(B)`.

Le souci, c'est qu'on veut que le résultat de `brd[MX3ADDR_SW]` puisse être utilisé comme un unsigned char, pour utiliser la valeur du registre dans des calculs. Pas de problème, pour cela, il suffit de créer une méthode `operator unsigned char` qui permettra la conversion implicite vers unsigned char.

Concrètement : créez une classe `Reg8` qui contient :

- un attribut de type `MX3board *` pour la communication
- un attribut de type `unsigned char` pour garder l'adresse du registre en mémoire
- Un constructeur qui initialise les deux attributs précédents

- Une surcharge de `operator=()` qui reçoit un `unsigned char` et qui l'envoie sur la carte (à la bonne adresse)
- Une surcharge de `operator unsigned char()` qui ne reçoit pas d'argument, mais qui lit le registre correspondant sur la carte, et le renvoie.

Modifiez le surcharcge `MX3board::operator[]()` pour qu'elle ne renvoie plus un `unsigned char`, mais un `Reg8` (initialisé à la bonne adresse).

Testez si, avec cette technique, vous êtes bien capables de manipuler les registres de la carte *basys MX3* en manipulant la classe `MX3board` comme un tableau.

Si ça ne marche pas, courage ! Si ça marche, félicitations, mais ce n'est que le début. Aucune des parties suivantes n'est obligatoire, votre projet sera constitué de celles que vous aurez choisies. Bien entendu, les parties que vous aurez choisies peuvent être combinées, et si vous avez bien pensé vos structures de données dès le départ, cela peut être avec un effort assez faible.

Les différentes extensions possibles ne sont pas triées par ordre de difficulté.

3 Extensions possibles

Vous trouverez dans cette partie toutes les possibilités qui s'offrent à vous pour compléter la structure de base, vous démarquer des vos voisins, exprimer votre talent et, peut-être, vous faire plaisir...

Pour rappel, vous ne pourrez pas tout faire, choisissez !

3.1 Manipulation des bits...

Si la carte est constituée de registres, les registres sont constitués de bits. Il est donc possible de reproduire le comportement de tableau à l'échelle du registre. Le type `bool` se prête très bien à la représentation des booléens (bizarerment :)).

Il y a besoin d'une classe `Bit` qui possède les méthodes `operator=()` et `operator bool()`, ainsi que d'un constructeur et certainement d'autres choses encore...

Il y a besoin d'ajouter à `Reg8` la méthode `operator[]()` qui reçoit un entier, et renvoie un `Bit`.

Mais maintenant, la vraie question : Est-ce que `Bit` et `Reg8` sont des classes indépendantes ? Est-ce que l'une est une sous-classe de l'autre ? Est-ce qu'elles sont toutes les deux sous-classe d'une même classe qui reste à définir ? C'est votre projet, faites votre choix...

3.2 Registres 16 bits (ou plus)

Dans la liste des registres, vous aurez remarqué que certains représentent une valeur sur 16 bits répartie sur deux registres 8 bits. Il existe des commandes pour écrire plusieurs registres d'affilée, il est donc possible de créer une classe `Reg16` qui représente des valeurs sur 16 bits.

Ce serait cool si la classe `Reg8` avait une méthode `r16()` qui renvoie un `Reg16` attaché à la même adresse. Dans ce cas, on pourrait faire des opérations sur 16 bits comme `brd[MX3ADDR_7SEG_DEC_L].r16() =1234;` ou encore `int time = brd[MX3ADDR_UPTIME_L].r16();`

Ce serait encore plus cool, si on pouvait éviter les partenthèses. Les lignes `brd[MX3ADDR_7SEG_DEC_L]. r16=1234;` et `int time = brd[MX3ADDR_UPTIME_L].r16;` sont franchement pas mal, non ?

Certains préfère l'écriture `Reg16{brd[MX3ADDR_7SEG_DEC_L]}=1234;` et `int time = Reg16{brd[MX3ADDR_UPTIME_L ]};`, C'est pas mal non plus... à vous de choisir.

3.3 Gestion des erreurs par les exceptions

Il existe de nombreuses possibilités que les choses se passent mal lors de la communication. Pour l'instant, il est simplement possible de récupérer les problèmes avec des codes d'erreurs stockés dans des propritétés de la classe.

Implémentez une gestion des problèmes de communication (timeout, non disponibilité du matériel ...) basée sur les exceptions. Vous pouvez également générer une exception lors d'une écriture sur un registre en lecture seule, ou d'une lecture d'un registre en écriture seule.

La *gestion des problèmes* signifie, la remontée d'information, mais aussi le traitement, si on ne veut pas faire planter le programme au moindre pépin.

3.4 Utilisation hors contexte

A partir du moment ou un registre est géré par une classe à part, il est possible de déclarer des instances de `Reg8` de façon indépendante. Mais même s'il serait très intéressant d'avoir une variable de type `Reg16` qui représente directement la valeur sur les afficheurs 7-segments, ou uen variable de type `Reg8` qui représente les LEDs, il faut rester vigilant.

En effet, si un programmeur écrit `reg_LED = reg_SW;`, ce n'est pas parce qu'il cherche à ce que la variable `reg_LED` représente l'état matériel des switches. C'est parce qu'il veut copier la valeur des switches sur les LEDs.

Créez l'environnement pour faciliter l'usage de classe `Reg8` ou autres dans des variables isoles.

Petite astuce : pour pouvoir déclarer une classe sans valeur initiale, il faut lui donner une constructeur qui ne prend pas d'argument.

3.5 Flux de sortie vers l'écran LCD

Faites ce qu'il faut pour que la ligne `brd << "toto" << std::endl;` affiche *toto* sur l'écran LCD. Ce serait encore mieux s'il se comportait comme un vrai terminal, c'est à dire que chaque nouvelle ligne écrite fait défiler le texte existant d'une ligne vers le haut.

3.6 Extension : Accès multiples

Pour effectuer des écritures multiples plus efficaces, il existe la méthode simple : surcharger `operator[]` pour qu'il accepte aussi un `std::vector<unsigned char>`. Mais dans ce cas, les données sont-elles écrites à des adresses qui se suivent (map) ou toutes à la même adresse (comme pour une FIFO) ?

Le mieux serait que l'utilisateur puisse le choisir. Par exemple : `brd[MX3ADDR_SND_WFIFO].fifo = vect_data;` écrit les données dans la Fifo sonore, et `brd[MX3ADDR_LCD_START].map = vect_data;` modifie le texte sur l'écran LCD.

Proposez une technique pour les lectures multiples. Ce qui est plus compliqué, c'est que pour `vect_data = MX3BRD_FLASH_RFIFO`, l'opérateur de lecture n'a pas accès à la taille de `vect_data`.

3.7 Accès simplifié à la mémoire Flash de la carte

Après tout, ça doit bien être possible d'accéder à la mémoire Flash sous la forme `brd.Flash[42]` où 42 serait une adresse de données de la mémoire Flash.

Selon votre envie, il pourrait également y avoir une classe `Flash_ptr`, dont le comportement serait précisément celui d'un pointeur. consultez la propostion 3.9.2 pour plus d'idée en ce sens.

3.8 Accès simplifié à la sortie sonore

La sortie sonore consiste simplement en un flux de données. Il n'y a rien en C++ qui permet de gérer des flux de données ?

Genre ... `brd.sound << "fichier sonore";` ou encore `brd.sound << std::vector<unsigned char>;`. Par contre, décompresser du mp3 n'est pas dans les objectifs du projet (trop compliqué, pas assez centré sur l'objet)

Bon, il y a certainement quelques bricoles à faire autour en plus, ou pas, cela dépendra de vos choix stratégiques.

Des fichiers sonores échantillonnés à 11kHz sont disponibles sur la page liée au projet. (oui, 11kHz seulement, car la liaison UART n'est pas assez rapide poru allez plus haut:/)

3.9 (plus compliqué): Héberger des variables sur la carte

3.9.1 forme classique

Dans certains cas particulers, il est intéressant d'héberger des données dans la mémoire de la carte au lieu de les héberger dans la mémoire centrale de l'ordinateur (c'est pas exemple le cas pour le calcul sur carte graphique, mais c'est aussi une solution de debugage pour observer des valeurs en temps réel sans (trop) perturber le programme).

Créez une classe *template* `Remote<typename>` qui héberge une donnée dans la mémoire de la carte, mais qui se comporte comme cette donnée si elle avait été stockée dans la mémoire de l'ordinateur. Le type de cette donnée est déterminé par son template. L'espace occupé dans la mémoire de la carte doit être juste celui nécessaire.

Selon votre envie, le choix de l'adresse à utiliser sur la carte peut être automatique ou manuel. S'il est automatique, il vous faudra également immplémenter une méthode `free()` ou `delete()` qui indique que la valeur n'est plus utilisée est que l'espace mémoire correspondant peut être considéré comme libre.

3.9.2 Gestion en tant que pointeurs

Techniquement, la classe `Remote<>` est un pointeur qui ne dit pas son nom, et pour lequel on ne peut pas connaitre la valeur. Écrivez la classe `Remote_ptr<>` qui assume son rôle de pointeur, et qui offre toute la sémantique des pointeurs : `operator *`, `operator ->` pour accéder aux valeurs de destination, arithmétique spécifique (l'opération `++` est équivalemente à `+=sizeof(...)`), comportement en tableau (`p[n]` est équivalent à `*(p+n)`) ...

3.10 (difficile): Polling

Le protocole ne permet pas à la carte d'envoyer des évênements ou des notifications. Il faut donc faire une scrutation régulière si on veut détecter un changement sur la carte.

Créez un environnement qui permet d'éxécuter une fonction au choix de l'utilisateur dès qu'un changement a été repéré sur les switches.

Cette proposition est classée difficile car elle fait appel à des concepts de gestion des processus, qui ne seront abordés que plus tard dans le semestre.

3.11 (difficile) : Lecture Audio en arrière plan

Là, on est sur le mont everest du masochisme... Imaginez que la ligne `brd.sound << "fichier sonore";` joue votre fichier musical, mais sans bloquer l'éxécution du programme...

Comme pour la proposition précédente, on tape dans le violent et le hors sujet, mais si quelqu'un est tenté ...

4 Modalités de restitution

- Par e-mail, en pièce jointe
  - Le rapport est au format PDF, et fait moins de 1Mo
  - Le code est fourni soit en vrac (fichiers `.cpp` et `.hpp`) soit en `.zip`. Fournir aussi le Makefile. Si votre client e-mail refuse l'envoi, vous pouvez compresser à nouveau votre fichier .zip (cela crée un fichier .zip dans un fichier .zip, ça ne fait pas gagner de place, mais ça perturbe le filtre qui laissera passer vos fichiers). Le rapport n'est pas dans le fichier compressé, mais à part !
  - Le binôme est en copie, et doit vérifier que l'envoi s'est bien passé.
  - Le principe de la remise d'un document est que vous devez l'envoyer. L'envoi de lien vers un cloud quelconque n'est donc pas valable.
- Date à définir lors de la première séance

Toute remarque de votre enseignant est prioritaire sur ce sujet