Projet microcontrôleur INFO 1 2016-2017

Jeu Simon

Guide de développement.

Ce document a pour but de vous aider à développer votre code en vous indiquant les étapes à respecter.

A chaque étape validée, vous ferez une démonstration à votre encadrant pour montrer que vous avez compris le code que vous avez écrit. Vous utilisez pour cela le simulateur.

0 Récupération du projet de départ.

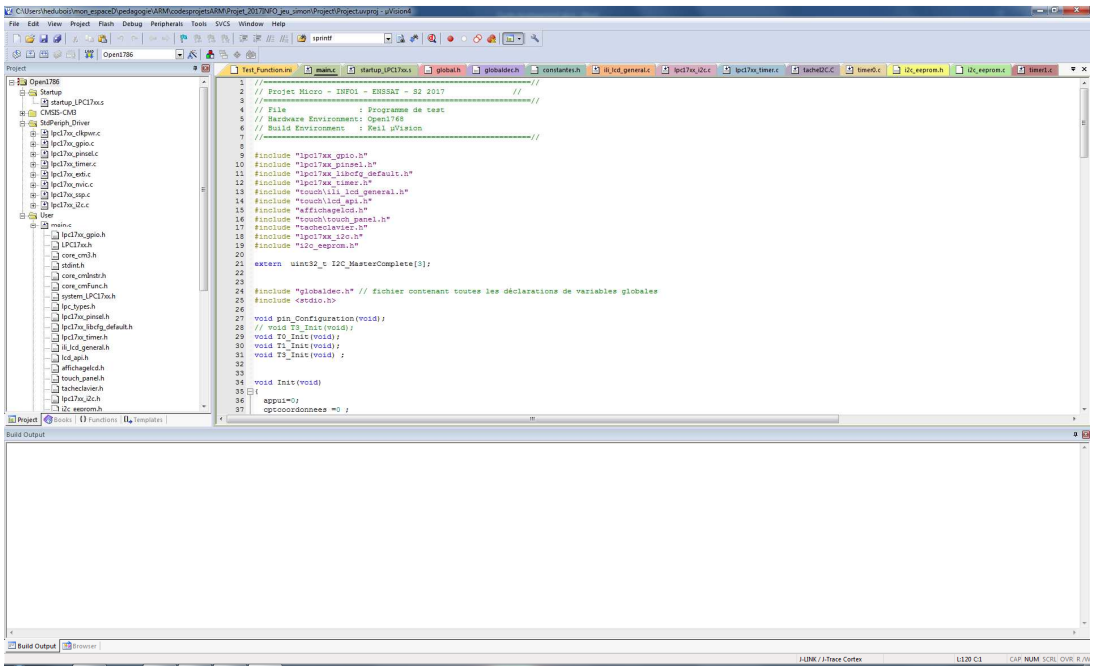
Pour pouvoir au final tourner sur la carte, le software doit être compatible avec l'architecture matérielle de la carte (choix du processeur, zones mémoires utilisées pour le code, pour les données …). Nous avons étudié cela pour vous. De plus le code de gestion de l'écran tactile vous ai donné.

C'est pourquoi vous ne devez pas partir d'un projet uvision vide, mais vous devez utiliser comme projet de départ celui à votre disposition sur ogam\ !pedagogie\pole electronique/espace 1 er année/LSI/ProjetSimon.

Copiez ce projet sur votre espace de travail personnel. Il y a 3 répertoires :

- « project » : contient le fichier .uvproj qui permet de lancer l'environnement uvision4. Contient également des fichiers de configuration du projet, des listings ou des codes issus des différents outils intégrés dans uvision4 (compilateur, éditeur de lien , debugger …)
- « libraries » : contient les bibliothèques fournies par le constructeur.
- « user » : dossier où vont se trouver tous les codes du projet.

Double-cliquez sur le fichier « projet.uvproj ». L'environnement uvision4 est lancé sur le projet de départ :



Prenez le temps de vous réapproprier cet environnement que vous avez déjà utilisé. Compilez le projet. Vous pourrez ainsi bénéficier de la recherche de variables ou de fonctions dans les fichiers : si vous cherchez où est déclarée une variable ou une fonction, sélectionnez son nom dans n'importe quel fichier où elle apparait et par un clic droit, demandez le « Go to definition » qui vous amènera directement dans le fichier où ce nom est déclaré. C'est très pratique quand on découvre les bibliothèques.

1 Configuration des broches du LPC1768.

Cette partie concerne tous les étudiants car elle va vous permettre un premier développement utilisant des fonctions de la bibliothèque.

Comme vous le savez (cf cours), chaque broche du composant LPC1768 peut être reliée à différents périphériques internes au microcontrôleur, ceci afin de réduire le nombre de broches et de s'adapter à l'application développée. C'est le Pin Connect Block qui est chargé de cela, configurable via les registres PINSEL. De même les broches gérées par les GPIO doivent être configurées (en entrée, en sortie,etc) C'est la première chose à faire.

Pour notre application, le « PINOUT » à respecter est le suivant :

	Broche LPC1768	Nom du Signal	Périphérique gestionnaire de la broche	Nom du Connecteur sur la carte OPEN1768
LCD écran	P2.0	D0	GPIO 2	LCD
	P2.1	D1	GPIO2	
	P2.2	D2	GPIO 2	
	P2.3	D3	GPIO 2	
	P2.4	D4	GPIO 2	
	P2.5	D5	GPIO 2	
	P2.6	D6	GPIO 2	
	P2.7	D7	GPIO 2	
	P1.25	EN	GPIO1	
	P1.24	DIR	GPIO1	
	P1.23	LE	GPIO1	
	P0.21	RD	GPIO0	
	P0.20	RS	GPIO0	
	P0.23	WR	GPIO0	
	P0.22	CS	GPIO0	
LCD tactile	P0.6	T_CS	SSP1	
	P0.9	T_DI	SSP1	
	P0.8	T_DO	SSP1	
	P0.7	T_SCK	SSP1	
	P0.19	T_IRQ	GPIO0 en IT	
Haut parleur	P1.9	son	GPIO1	ETH en haut calé sur 1
Mémoire	P0.27	SDA0	I2C0	**I2C0**
	P0.28	SCL0	I2C0	

L'initialisation des broches pour l'interface avec l'écran LCD (affichage) est faite par la procédure lcd_port_init() appelée dans la procédure lcd_Initialisation().

Pour l'interface avec l'écran pour la partie tactile, l'initialisation est faite dans la procédure touch_init().

Ces procédures s'appuient sur les bibliothèques pour programmer les broches des GPIO et le Pin Connect Block. Comme cela a été vu en amphi, ces bibliothèques utilisent des structures de configuration dans lesquelles on entre des valeurs qui servent ensuite à la programmation des registres du périphérique. Assurez-vous en analysant les codes lcd_port_init(), touch_init() et spi1_init() que vous avez compris le principe de ces structures.

En vous inspirant des codes écrits pour l'écran LCD, écrivez une procédure pin_Configuration() qui initialise les autres broches pour la gestion du haut parleur par le GPIO1 en sortie et de la mémoire par l'I2C0.

Validez en simulation votre code. On peut en effet vérifier la configuration sur l'écran accessible par le menu Peripherial/ Pin Connect Block et l'écran Peripherial/ GPIO Fast Interrupt pour le sens d'un GPIO. Faites une démo à votre encadrant en étant prêt à expliquer ce que vous avez compris, notamment l'utilisation des structures et des fonctions de la bibliothèque.

2 Configuration de l'I2C0.

Cette partie concerne les étudiants qui développent la partie de l'application qui gère la mémoire.

2.1 Compréhension de la mémoire.

Le petit module d'extension AT24/FM24 vient se connecter à la carte OPEN1768 par le connecteur nommé I2C0. Il comporte une mémoire FM24CL16 dont vous disposez de la documentation.

En analysant cette documentation, répondez aux questions suivantes :

- Combien de mots contient cette mémoire et quelle est la taille des mots ? Combien de bits d'adresse sont donc nécessaires pour adresser un mot ?
- L'interface de la mémoire n'est pas directement un bus d'adresse, un bus de contrôle et un bus de données, mais un bus I2C. Combien de signaux composent un bus I2C ? Comment se fait un échange sur ce bus ? Qui sera le maitre du bus dans notre cas ?
- Décrivez les trames à envoyer pour écrire un mot en mémoire, pour lire un mot en mémoire.

2.2 Initialisation du contrôleur I2C.

Le LPC1768 possède 3 contrôleurs I2C. Nous utiliserons le 0. Ce contrôleur se charge du protocole I2C pour envoyer ou recevoir des trames. Vous disposez de la description de ce contrôleur dans la documentation LPCXX_user_manual.pdf. Une bibliothèque lpc17xx_i2c.c et son lpc17xx_i2c.h associé vous sont fournis pour vous faciliter l'écriture du code.

Faites les étapes suivantes pour initialiser l'I2C :

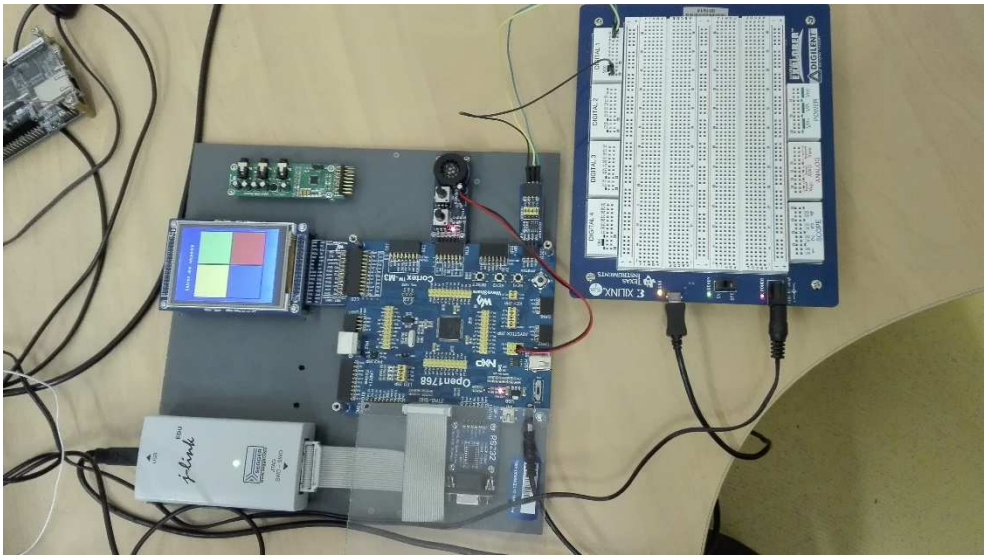
- Vérifiez si ce périphérique I2C est alimenté par défaut (Power control PCONP registre).
- Décidez de la vitesse à laquelle on va faire fonctionner ce bus (NB : il faut que ce soit compatible avec les possibilités de la mémoire). Trouvez dans le fichier lpc17xx_i2c.c la fonction qui permet de programmer cela.
- Trouvez également dans ce fichier la fonction qui permet d'autoriser des échanges I2C0
- Ecrivez une procédure init_i2c_eeprom() qui fait l'initialisation. La compiler, la tester en simulation.

2.3 Ecriture dans la mémoire.

Dans un premier temps on va utiliser le contrôleur I2C en mode polling, c'est-à-dire que l'envoi ou la réception de donnée ne seront pas gérés en interruption et monopoliseront le processeur tant que

l'envoi ou la réception ne seront pas terminée. Trouvez dans le fichier lpc17xx_i2c.c la fonction qui permet de faire une écriture avec le processeur en mode maitre sur le bus. Grace à cette fonction et à l'étude faite en 2.1, faites les étapes suivantes :

- Quels sont les paramètres à passer à cette fonction ?
- Quelles valeurs mettre sur ces paramètres pour faire l'écriture d'un mot dans la mémoire ?
- Ecrivez une procédure i2c_eeprom_write(uint16_t addr, void* data, int length) qui permet d'écrire dans la mémoire un tableau de donnée de taille length à partir de l'adresse addr.
- Compilez et vérifiez son fonctionnement en simulation. Montrez cet état d'avancement à votre encadrant
- Compilez maintenant pour tourner sur la carte (baguette magique/ onglet Debug et cocher « use jlink »). Recompilez et chargez le code sur la carte. Utilisez une carte Digilent Explorer pour visualiser le bus I2C et vérifier que la trame est correctement envoyée.



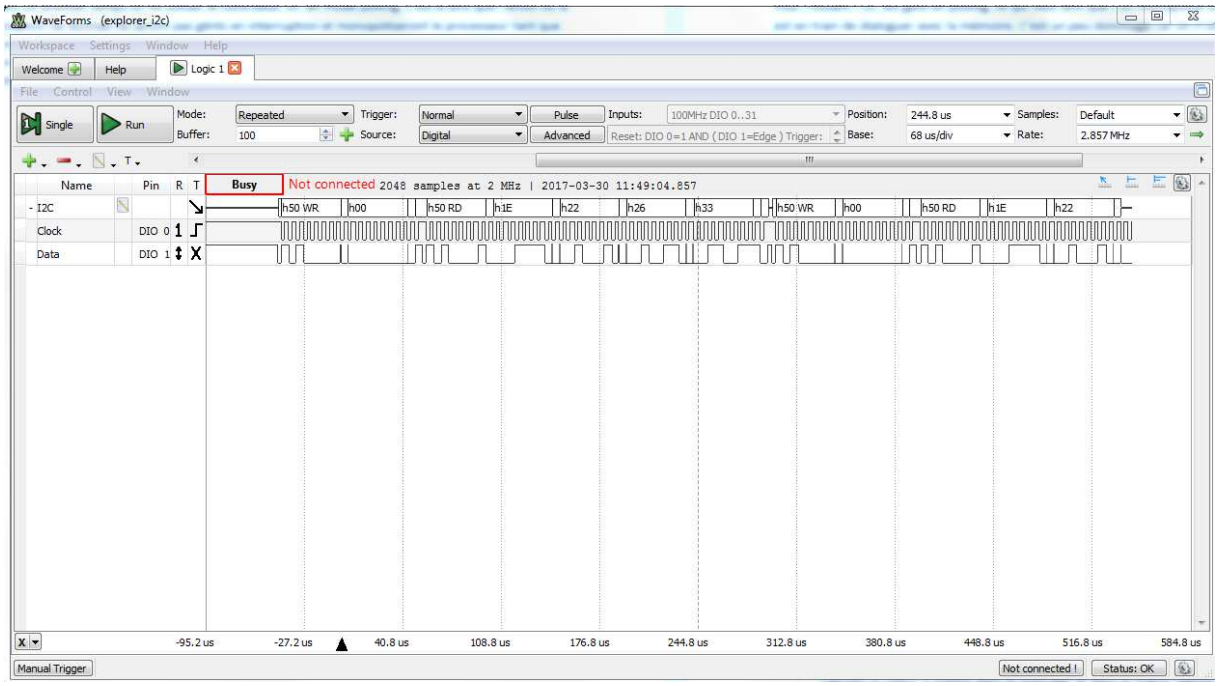


Photo de la carte équipée du module I2C relié à la carte Digilent pour visualisation des signaux.

Montrez cet état d'avancement à votre encadrant

2.4 Lecture dans la mémoire.

Procédez par étapes, comme cela vous a été montré pour l'écriture, pour écrire une fonction i2c_eeprom_read(uint16_t addr, void* data, int length) qui permet de lire dans la mémoire un bloc de donnée de taille length à l'adresse addr.

Validez en simulation, puis sur la carte. Montrez cet état d'avancement à votre encadrant.

Vos codes peuvent maintenant être intégrés dans le code général de l'application. Néanmoins une amélioration serait de fonctionner en interruption .

2.5 Gestion par interruption.

Pour l'instant l'I2C est géré en polling, ce qui veut dire que l'on monopolise le processeur quand on est en train de dialoguer avec la mémoire. C'est un peu dommage car on n'utilise pas le parallélisme matériel qui existe grâce au contrôleur I2C. C'est un peu comme si une personne donnait du travail à faire à une autre personne et restait ensuite à coté à attendre qu'elle ait terminé au lieu de profiter de ce temps pour aller faire autre chose …

Etudiez donc la possibilité de gérer l'I2C en mode buffer et par interruption, c'est-à-dire que lorsque l'on veut écrire ou lire, on indique le nombre de données que l'on veut écrire ou lire, elles sont gérées par l'interface I2C qui génère une interruption permettant de savoir quand le transfert complet est terminé.

Procédez comme précédemment par étape en comprenant tout d'abord, grâce à la documentation LPCXX_user_manual.pdf, à quels moments le périphérique I2C génère une interruption. En analysant ensuite le code I2C_MasterHandler(), appropriez-vous comment utiliser ce code. Développez maintenant votre code pour faire écriture et lecture dans la mémoire avec fonctionnement en interruption, en validant au fur et à mesure.

3 Génération d'une note.

Cette partie concerne les étudiants qui développent la génération d'une note sur le haut-parleur.

3.1 Etude des signaux à générer.

Pour générer un son sur le petit haut-parleur du module d'extension Speaker, il suffit de générer un signal carré sur le port P1.9 à une fréquence audible.

Décidez les notes que vous souhaitez émettre pour chaque touche de couleur. Quelles sont les fréquences associées ? Quelle précision est souhaitée sur ces fréquences pour une oreille normale ?

3.2 Utilisation d'un timer pour émettre la note.

Vous disposez d'un fichier lpc17xx_timer.c qui contient des fonctions pour la gestion d'un timer. Regardez cette bibliothèque de fonctions pour voir celles qui peuvent vous faire gagner du temps par rapport à une écriture dans les registres.

L'étude du 3.1 montre normalement que les périodes à générer sont lentes par rapport à la précision que peut avoir un timer. Dans beaucoup de cahier des charges, on peut avoir besoin de mesurer des temps à plusieurs endroits dans l'application. Plutôt que de monopoliser un timer par temps à mesurer, on va programmer un timer pour générer une interruption à un certain rythme, et on va gérer des compteurs logiciels dans cette interruption pour calculer les temps dont on a besoin. C'est que l'on appelle une base de temps.

Vous allez configurer le timer 0 pour créer une base de temps à 50 us avec une précision de 1 us. Pour cela suivez les étapes.

- Calculez la valeur à mettre dans le prescaler et dans le match register
- Ecrivez une procédure T0_Init() qui fait l'initialisation du timer 0 en mode timer, avec les bonnes valeurs dans le prescaler et le match register, avec quand ça « match », un reset et la génération d'une interruption. Cette procédure ne lancera pas le timer pour l'instant.
- Repérez dans le fichier lpc17xx_timer.c la fonction qui permet de lancer le timer.
- Ecrivez le traitant d'interruption TIMER0_IRQHandler () avec pour l'instant dedans seulement un acquittement de cette interruption.
- Dans le main programmez le contrôleur d'interruption pour autoriser l'interruption timer0 à se produire et lancez le timer.
- Compilez et simulez en mettant un point d'arrêt dans le code de l'interruption pour vérifier qu'elle se produit bien régulièrement.
- Insérez maintenant dans le code de l'interruption un compteur logiciel, c'est-à-dire une variable que vous allez incrémenter à chaque passage dans l'interruption, et qui, quand elle a atteint la valeur correspondant à la demi-période de la note souhaitée, est remise à 0 et inverse le port P1.9.
- Compilez et simulez en vérifiant que le port P1.9 s'inverse bien toutes les demi-périodes de la note. Vous pourrez pour cela visualiser le port P1.9 grâce à la fenêtre analyseur logique (sous le debbuger en mode debbug, menu View/Analysis windows/Logic Analyser) .
- Si le signal était correct en simulation, faites tourner le code sur la carte et vous devriez entendre l'émission de la note. Montrez cet état d'avancement à votre encadrant.

3.3 Emission de notes différentes.

Vous avez maintenant les fonctions qui permettent d'émettre une note à une fréquence donnée. Enrichissez votre code pour pouvoir émettre et arrêter d'émettre une note dont la fréquence est variable. Attention, comme le timer0 sert de base de temps pouvant être utilisée pour d'autres mesures de temps, il ne faut pas l'arrêter … L'arrêt de l'émission d'une note doit donc être fait par logiciel et pas en arrêtant ce timer .

4 Ecran tactile LCD.

Cette partie concerne les étudiants qui ont travaillé sur le main et l'écran LCD.

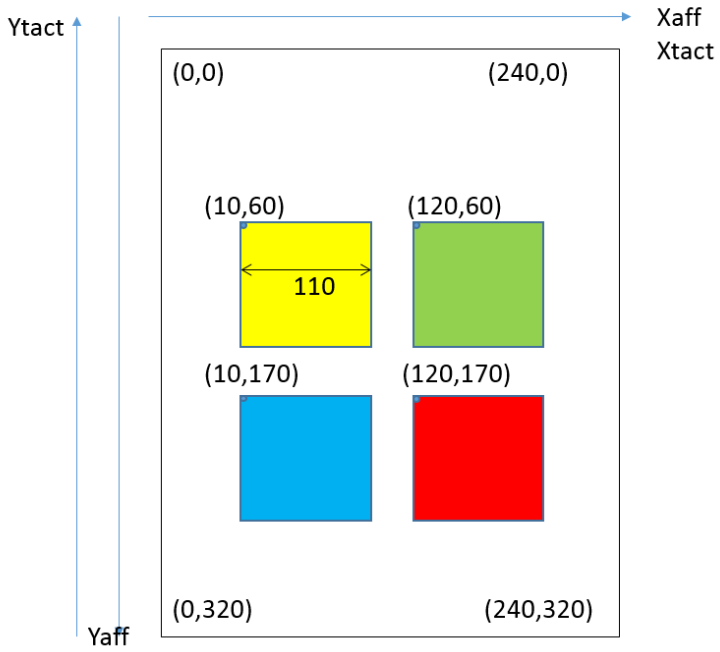
L'écran tactile est assez compliqué à gérer et il a été choisi cette année de vous donner les codes qui le gèrent pour que vous vous consacriez aux autres parties. Il vous faut néanmoins utiliser correctement ces codes. Si vous avez le temps, vous avez toutes les documentations pour les comprendre dans le détail, mais ce n'est pas demandé.

Sachez simplement que l'affichage est géré grâce une interface de 15 signaux (ceux donnés dans le tableau du pinout), qui sont gérés par les GPIO et doivent respecter certains timings précis. Le coté écran tactile est lui géré via une interface SPI et par le port P0 .19 qui est à 0 quand l'écran est touché.

4.1 LCD Affichage.

L'initialisation est réalisée par la fonction lcd_Initializtion() qui vous est fournie. Elle contient des tempos qui doivent être respectées et est à exécuter une seule fois, au début du main. Après cette

initialisation, l'écran est en mode portrait avec le repère de coordonnées donné sur le schéma ci-dessous.



On peut écrire des chaines de caractères à partir d'un point par la fonction LCD_write_english_string , dessiner un rectangle d'une certaine couleur par la fonction dessiner_rect, comme cela l'a été fait pour afficher l'écran d'accueil.

Attention ces fonctions utilisent des attentes de temps bloquantes. Elles ne doivent donc pas être utilisées sous interruption. De même on n'utilisera ces fonctions que dans une seule tache logicielle pour éviter des conflits d'accès à cette ressource écran.

Amusez-vous à modifier le texte affiché, l'endroit où il est affiché, la couleur des rectangles pour vérifier que vous avez compris comment faire un affichage. Ne vous amusez pas trop longtemps …

4.2 LCD tactile.

L'initialisation du côté tactile de l'écran est faite par la procédure touch_init() qui vous est fournie. Elle doit être faite une seule fois, au début du main , juste après l'initialisation de l'affichage. Une fois faite, l'écran signalera un appui en mettant le port P0 .19 à 0 tant que l'appui est en cours, puis remettra P0.19 à 1 quand l'appui est fini. La procédure touch_read() permettra de faire un dialogue SPI pour aller récupérer dans les variables globales touch_x et touch_y les coordonnées de l'appui. Les coordonnées pour les différents zones d'appui seront à étalonner car elles peuvent légèrement changer en fonction de l'écran.

A titre d'exemple, l'étalonnage sur un écran a donné :

- Zone jaune Xtact entre 600 et 2000, Ytact entre 2000 et 3000
- Zone verte Xtact entre 2100 et 3600, Ytact entre 2000 et 3000
- Zone bleue Xtact entre 600 et 2000, Ytact entre 700 et 1800
- Zone rouge Xtact entre 2100 et 3600, Ytact entre 700 et 1800

Attention cette procédure touch_read() utilise des attentes bloquantes. Elle ne doit donc pas être utilisée sous interruption. . De même on n'utilisera ces fonctions que dans une seule tache logicielle pour éviter des conflits d'accès à cette ressource tactile.

Le principe sera de scruter ce port P0.19 « de temps temps » grâce à une base de temps. Le « de temps en temps » doit être cohérent avec la rapidité à laquelle un utilisateur peut appuyer sur l'écran. Il semble raisonnable de scruter toutes les 100 ms.

Pour réaliser cette scrutation, vous allez donc créer une base de temps à 20ms grâce au timer1. Suivez les mêmes étapes que pour la création de la base de temps de gestion des notes. Validez de la même façon que précédemment qu'elle fonctionne bien.

Complétez ensuite le code de l'interruption de cette base de temps pour y réaliser la scrutation du port P0.19 et armer un drapeau flagtacheclavier quand il y a eu un appui. Vérifiez en simulation que tout cela fonctionne correctement. Montrez cet état d'avancement à votre encadrant.

On peut maintenant utiliser le drapeau pour déclencher la lecture des coordonnées de l'appui grâce à la procédure touch_read(). Comme elle ne doit pas être appelée sous interruption, vous allez utiliser la méthode de programmation donnée dans l'amphi, à savoir réveiller une tache logicielle dans le main quand le drapeau flagtacheclavier aura été mis à 1.

Mettez en place ce mécanisme pour afficher quelque part sur l'écran les coordonnées de l'appui.

5 Emission de la note liée à la zone d'appui et sauvegarde des appuis successifs.

Cette étape concerne tous les étudiants car il va s'agir de structurer l'ensemble de l'application.

Vous disposez maintenant de tout ce qu'il faut pour être capable de détecter un appui et sa position.

Il est donc temps de mettre en place l'application du jeu en lui-même. Vous allez pour cela réfléchir à comment enchainer les traitements grâce à la méthode expliquée en amphi sur la programmation structurée. Décrivez sur papier grâce à cette méthode une tache logicielle tache_clavier qui saura gérer le fonctionnement suivant :

- Ne rien faire si l'écran n'est pas appuyé
- Emettre une note correspondant à la bonne couleur tant qu'on appuie
- mémoriser dans un tableau la succession des couleurs appuyées
- arrêter ce mode quand il s'est passé plus de 4 secondes sans appui.

Montrez cette modélisation à votre encadrant.

6 Construction de l'application complète.

Vous avez maintenant tout pour réfléchir à la structure de l'application complète : quelles sont les taches logicielles ? Comment « avancent-elles » ? Quelles sont les taches matérielles ? Comment faire le lien entre l'appui sur l'écran et la sauvegarde en mémoire i2c ? Etc …
A vous de spécifier tout cela et de le coder, étape par étape, en validant chaque avancée en simulation et sur la carte, jusqu'à obtenir le jeu complet. Validez auprès de votre encadrant à chaque fois que vous atteignez une étape intéressante. Il sera très content si il peut vraiment utiliser le jeu à la fin du projet ☺ !