Mi-projet microcontrôleur IMR1 1 2021-2022

Gestion d'une mémoire I2C

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.

Il est demandé de partager une carte OPEN1768 pour 2, un dans chaque groupe pour que vous en ayez une à chaque séance. Et quelques cartes DIGILENT Electronics Explorer pour chaque groupe. Pour réserver ces cartes, allez sur la page https://inventaire.enssat.fr/glpi/front/reservationitem.php et réservez les cartes que vous pourrez ensuite aller retirer au comptoir du service technique (2ème étage du bâtiment N). Des casiers que vous pouvez fermer avec des cadenas personnels sont disponibles au 1er étage du bâtiment N et en 101D pour vous éviter de transporter les cartes à chaque fois.

0 Récupération du projet de départ (Rappels).

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 est donné.

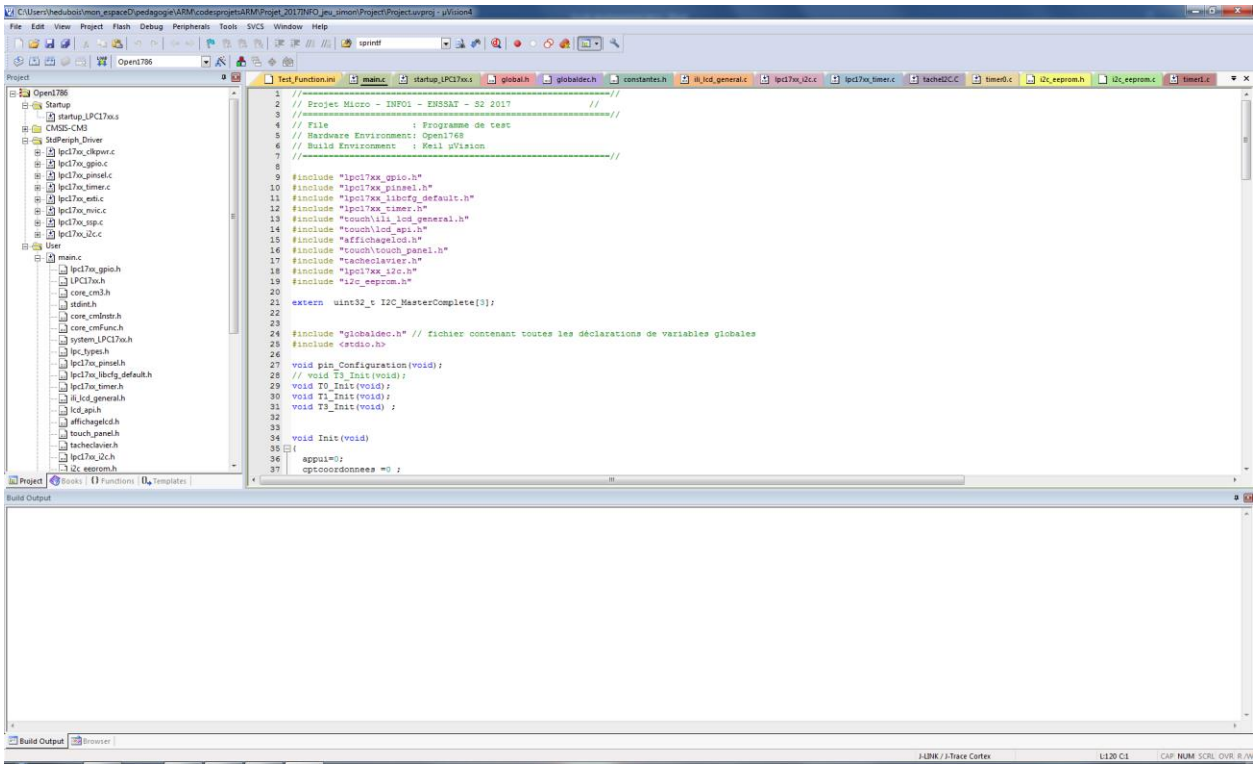
C'est pourquoi vous ne devez pas partir d'un projet uvision vide, mais vous devez utiliser comme projet de départ celui à votre disposition sur Moodle. Il contient un main et des fonctions qui permettent de gérer l'écran tactile LCD.

Copiez ce projet sur votre espace de travail personnel. Attention le bureau des PCs Enssat dans les salles peut être formaté à tout moment et ne constitue pas un espace de stockage fiable. Utilisez votre partition sur Ogam ou une clef USB.

Pour rappel, un projet uvision, 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. A ne pas modifier.
- « user » : dossier où vont se trouver tous les codes du projet. C'est ici que vous sauverez les codes que vous développez.

Double-cliquez sur le fichier « projet.uvproj ». L'environnement uvision5 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 pour se repérer dans les bibliothèques.

N'oubliez pas que vous avez les codes C des bibliothèques associés à chaque périphérique. Chaque fichier comporte d'abord des fonctions « privées » ; elles sont utilisées dans le même fichier et non utilisables dans un autre fichier, donc dans votre code. A la suite il y a les fonctions publiques ; leurs prototypes apparaissent dans le .h associé et vous pouvez donc les utiliser dans votre code en incluant simplement le .h associé. Chaque fonction a un entête qui décrit le rôle de la fonction et de ses paramètres. Si cela n'est pas assez clair, n'ayez pas peur d'aller regarder le code même que vous êtes capables de comprendre !

Votre code sera à organiser en plusieurs fichiers. Un pdf « Explications sur l'utilisation des fichiers .H » est disponible sur Moodle pour vous rappeler comment partager votre code en plusieurs fichiers.

1 Configuration des broches du LPC1768.

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	
	P0.9	T_DI	SSP1	
	P0.8	T_DO	SSP1	
	P0.7	T_SCK	SSP1	
	P0.19	T_IRQ	GPIO0 en IT	
Boutons poussoir	P2.10	KEY2	GPIO2	**Sur la carte mère**
	P2.11	KEY1	GPIO2	
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() qui est appelée dans la procédure lcd_Initializtion (). L'init pour le tactile est faite dans la procédure touch_init(). Vous appellerez donc ces procédures lcd_Initializtion() et touch_init() au début de votre main. Attention, ces procédures étant lentes (car pleine de tempos à respecter), vous pouvez gagner du temps en les mettant en commentaire tant que vous n'utilisez pas l'écran tactile.

De même vous pouvez gagner du temps en enlevant de votre projet les .c qui concernent la gestion de l'écran tactile tant que vous ne l'utilisez pas.

Maintenant, en vous inspirant des codes écrits en TD, écrivez une procédure pin_Configuration() qui initialise les autres broches pour la gestion de la mémoire par l'I2C0 et les boutons poussoirs.

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 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.

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 « Mémoire I2C » sur Moodle.

En analysant cette documentation, répondez aux questions suivantes :

- Combien de mots contient cette mémoire et quelle est la taille des mots ? Déduisez-en combien de bits d'adresse sont donc nécessaires pour adresser un mot ?
- L'interface de la mémoire n'est pas, comme classiquement, un bus d'adresse, un bus de contrôle et un bus de données, mais un bus I2C. Le fonctionnement de ce protocole est rappelé dans la documentation de la mémoire. Déduisez-en combien de signaux composent un bus I2C, comment se fait un échange sur ce bus. Qui sera le maitre du bus dans notre cas ? Qu'elle est l'adresse esclave I2C à utiliser pour notre mémoire ? Faites bien la différence entre l'adresse du mot auquel on souhaite accéder dans la mémoire et l'adresse esclave I2C !
- Décrivez précisément les trames à envoyer pour écrire un mot en mémoire, pour lire un mot en mémoire : combien d'octets vont être envoyés, quel est le contenu du 1er octet, du 2ème, etc …
- La bonne nouvelle est qu'il existe une bibliothèque pour gérer tout le protocole I2C et c'est ce qui va être vu maintenant.

2.2 Initialisation du contrôleur I2C.

Maintenant que vous avez compris quelles trames il va falloir utiliser pour dialoguer avec la mémoire, vous allez vous intéresser à comment le faire avec notre microcontrôleur.

Le LPC1768 possède 3 périphériques I2C. Nous utiliserons le I2C0. Ce périphérique 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. Incluez le fichier lpc17xx_i2c.c dans votre projet, et le fichier lpc17xx_i2c.h dans vos fichiers .c qui utiliseront les fonctions de cette bibliothèque.

Faites les étapes suivantes pour initialiser l'I2C :

- Tout d'abord vérifiez si ce périphérique I2C0 est alimenté par défaut (Power control PCONP registre). Si ce n'est pas le cas, trouvez comment l'alimenter.
- 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 et aussi de la carte DIGILENT qui nous servira à espionner le bus, donc pas plus de 500Khz). 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.

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 terminés. 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 notre mémoire ? NB : attention le buffer contenant les données doit être déclaré en global.
- 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 mémoire addr. NB : le type « void » a été utilisé pour vous laisser décider du type le plus approprié ; à vous de le changer. Rappel : ne pas confondre adresse esclave I2C et adresse où on veut écrire dans la mémoire.
- Compilez et vérifiez son fonctionnement en simulation, en vérifiant notamment les contenus des variables. 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. Vous disposez sur Moodle d'un fichier « Fichier pour la carte espion digilent » déjà bien configuré. Utilisez-le et vérifiez attentivement que la trame est envoyée est correcte.

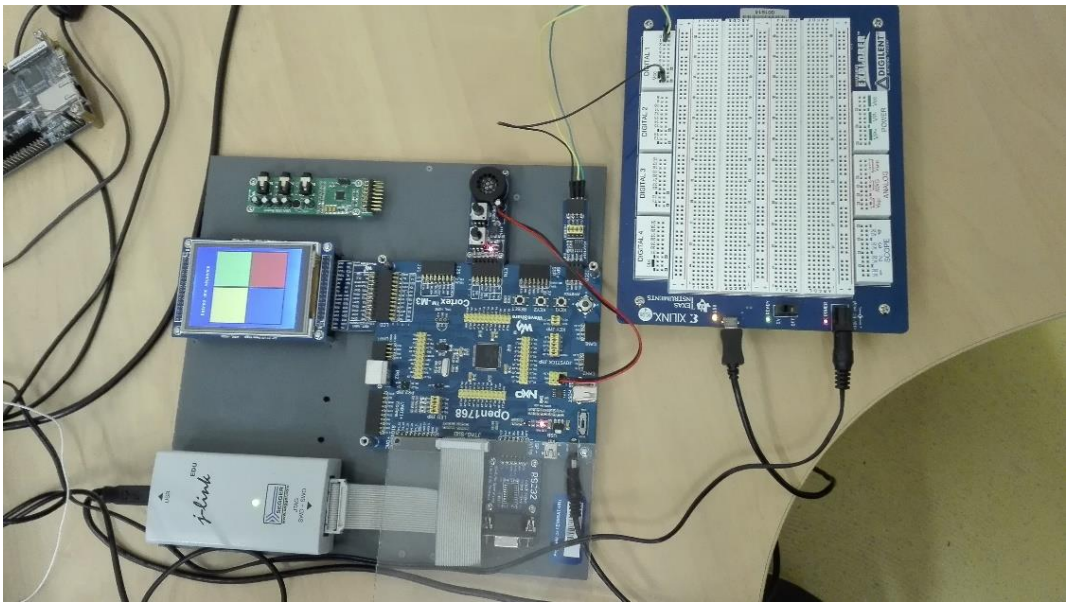


Figure 1 connexion du module avec la mémoire et de la carte Digilent au module I2C

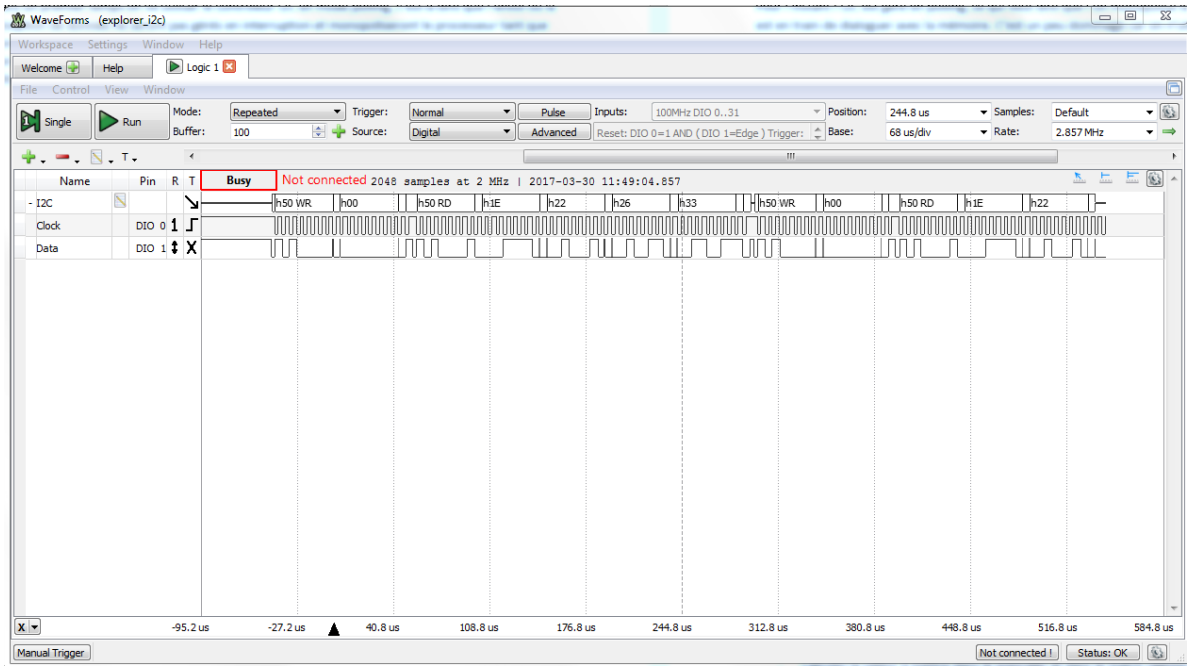


Figure 2 Visulation de la trame I2C par l'outil Waveform si tout va bien!

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.

3 Construction d'une application complète.

Vous avez maintenant tout pour réfléchir à la structure d'une application complète plus complexe.
Vous devez inventer une application qui respecte les contraintes suivantes :

- Un timer sert de base de temps pour générer une interruption toutes les 10 ms avec une précision de 500 us. Ce timer vous sert par exemple à scruter l'écran tactile.
- Votre application devra utiliser la mémoire I2C en lecture et en écriture
- Votre application pourra utiliser l'écran tactile (des bibliothèques sont dispos pour cela)
- Votre application devra être construite petit à petit, en validant à chaque étape jusque sur la carte

A partir de votre cahier des charges, identifiez les taches matérielles, les taches logicielles. Ecrivez le code petit à petit en utilisant les procédures ou fonctions déjà développées. Montrez à votre encadrant à chaque fois que vous atteignez une étape intéressante.

Attention vous devez être capable d'expliquer chaque ligne que vous avez écrite, en lien avec le microcontrôleur et en utilisant le simulateur!

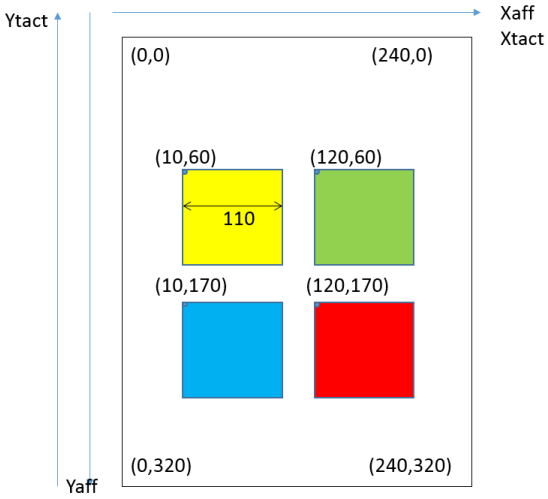
4 Pour utiliser l'écran pour afficher et également en tactile, voici quelques explications :

L'écran tactile est assez compliqué à gérer et il vous est donné 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 (mais c'est bien d'être curieux …).

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 (Xaff, Yaff) système de coordonnées pour l'affichage, (Xtact, Ytact) système de coordonnées pour le tactile.



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.

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, à tracer des rectangles de couleur pour vérifier que vous avez compris comment faire un affichage. Ne vous amusez pas trop longtemps …

NB : certaines cartes ont des écrans avec un contrôleur légèrement différent qui peine quelque fois à s'intialiser ; l'écran reste alors blanc malgré l'initialisation. Si vous êtes dans ce cas, chargez votre code sur la carte ; appuyez sur le bouton « reset » de la carte ; le programme est exécuté « à froid » et cela résout généralement le problème (c'est-à-dire que l'affichage se fait et que tant que vous n'éteignez pas la carte, vous pouvez maintenant utiliser le debbug sans problème).

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 en 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 pouvez utiliser la base de temps à 10ms en vous inspirant de celle écrite pour faire clignoter la led. 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.

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.

Utilisez ce mécanisme pour votre application.