MICROCONTRÔLEURS ET INTERFACES

GUIDE DE L'ÉTUDIANT
S4e – APP3

Hiver 2024 – Semaine 5 et 6

Département de génie électrique et de génie informatique

Faculté de génie

Université de Sherbrooke

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Version 2.0 Hiver 2024 – Yves Bérubé-Lauzière (YBL); cette version découle du fichier que JP Gouin avait préparé en vue de Hiver 2023 - 1 fév 2023; cette version incorpore également des éléments du guide de S Roy (en LaTeX) que S Roy a utilisé à Hiver 2023

Version 1.7 Été 2022 et Hiver 23 – JP Gouin

Version 1.6 Hiver 2022 – JP Gouin Changement du labo et des procéduraux

Version 1.5 Été 2021 – JP Gouin

Version 1.4 Hiver 2021 – JBM & JP Gouin

Version 1.3 Été 2020 – JBM révisé procéduraux et labos

Version 1.2 Février 2020 – Sébatien Roy, mise à jour

Version 1.1 Mai 2019 – Jean-Baptiste Michaud (JMB), mise à jour

Version 1.0 : Janvier 2019 – Sébastien Roy, Daniel Gaucher, inspiré en partie de matériel par Daniel Dalle,

Marc-André Tétrault et Arnaud Marchese

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

1 ACTIVITÉ PÉDAGOGIQUE ET COMPÉTENCES

GEL452 – Microcontrôleurs

1. Programmer et déployer un microcontrôleur.
2. Mettre en oeuvre et employer une méthodologie de développement de systèmes embarqués à microcontrôleur et ses applications, en utilisant des outils de développement physique et logiciel.

Description officielle : https://www.usherbrooke.ca/admission/fiches-cours/GEL452/

GEL442 – Logique programmable et interfaces

1. Modéliser et concevoir des interfaces numériques par une représentation mathématique de l'information discrète et par des méthodes de synthèse des équations booléennes.
2. Réaliser des interfaces numériques en fonction de critères de performance, évaluer la compatibilité électrique d'interfaces numériques à signaux différentiels, mesurer les performances d'échange d'informations numériques.

Description officielle : https://www.usherbrooke.ca/admission/fiches-cours/GEL442/

2 SYNTHÈSE DE L'ÉVALUATION

Tableau 1 : Synthèse de l'évaluation de l'unité

Évaluation	GEL442-1	GEL442-2	GEL452-1	GEL452-2
Validation d'APP et livrables associé	35	35	25	25
Évaluation formative théorique				
Évaluation formative pratique				
Évaluation sommative théorique/pratique	65	65	50	50
Évaluation finale	100	100	100	100
Total	200	200	175	175

3 QUALITÉS DE L'INGÉNIEUR

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant :

https://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/

Tableau 2 : Tableau des qualités de l'ingénieur

Numéro	Libellé	Touchée	Évaluée
Q01	Connaissances en génie	✓	✓
Q02	Analyse de problèmes	✓	✓
Q03	Investigation		
Q04	Conception		
Q05	Utilisation d'outils d'ingénierie	✓	✓
Q06	Travail individuel et en équipeS		
Q07	Communication		
Q08	Professionnalisme		
Q09	Impact du génie sur la société et l'environnement		
Q10	Déontologie et équité		
Q11	Économie et gestion de projets		
Q12	Apprentissage continu		

4 ÉNONCÉ DE LA PROBLÉMATIQUE

Station météorologique et sismique

Votre compagnie spécialisée dans la conception de systèmes embarqués, EmbarQ, reçoit le mandat de concevoir un prototype de station autonome de mesures météorologiques et sismiques. Le prototype aura pour cœur un microcontrôleur de type PIC32MX370F512L de la compagnie Microchip, lequel est doté d'un noyau de traitement à architecture MIPS32 et de nombreux périphériques intégrés. Pour éviter d'avoir à concevoir un circuit imprimé sur mesure pour cette première preuve de concept, votre patronne suggère de travailler à partir d'une carte de développement Basys MX3, laquelle comporte déjà plusieurs éléments intégrés nécessaires, dont un accéléromètre, un écran LCD et une mémoire FLASH.

Premièrement, un code de départ permet d'afficher l'heure (HH:MM:SS) à l'écran de la carte MX3. Ce code devra être modifié afin qu'il soit possible de configurer l'heure à l'aide des boutons au démarrage de la carte et par la suite de présenter 2 affichages contrôlés par l'état de l'interrupteur 0. Dans le but d'optimiser le développement de votre code, il est permis d'utiliser les autres interrupteurs pour afficher autre chose sur l'écran LCD, tout en gardant les affichages prescrit ci-dessous.

- Affichage 1 : Heure et valeur du potentiomètre (ligne 1) et affichage au besoin du message suivant : « Attente d'un ACK » (ligne 2).
- Affichage 2 : Les 4 valeurs de l'accéléromètre (x, y, z et module) aux 4 coins de l'écran.

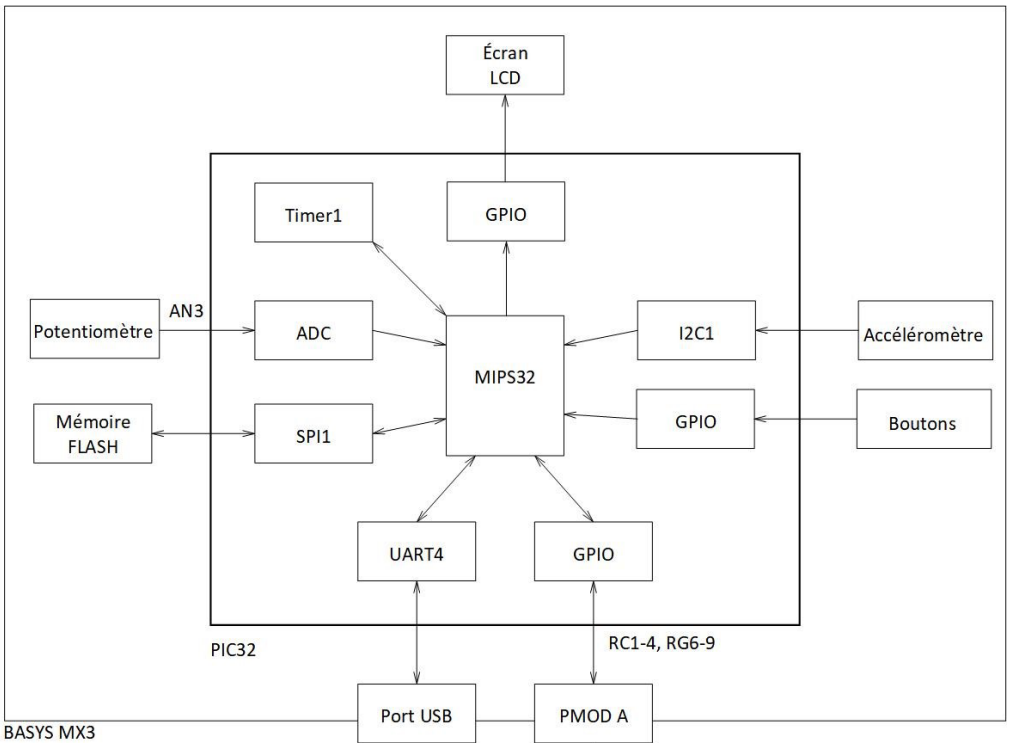


Figure 1 : Diagramme d'ensemble de la carte Basys MX3.

Pour l'instant, le prototype de station (figure 1) ne sera munie que de deux capteurs, soit (1) l'accéléromètre (déjà intégré à la carte MX3 et raccordé au microcontrôleur par une interface série I2C) et (2) un potentiomètre intégré à la carte MX3 et servant à simuler une cellule photosensible pour mesurer la luminosité. La tension analogique à la sortie de ce potentiomètre est numérisée à l'aide d'un convertisseur analogique-numérique (CAN – *analog-to-digital converter - ADC*) intégré au microcontrôleur. Le système devra maintenir une horloge interne et afficher l'heure.

Par le biais d'un compteur et d'une interruption associée, une acquisition des données des capteurs sera faite à intervalle régulier, soit à toutes les secondes. Cette même interruption servira aussi à mettre à jour l'heure affichée. Dans chaque intervalle, on calculera le module de l'accélération selon l'équation :

$$|a| = \sqrt{a_x^2 + a_y^2 + a_z^2}\,,$$

où a_x, a_y et a_z sont les données brutes fournies sur 12 bits par l'accéléromètre pour les 3 axes. Le calcul de la racine carrée sera réalisé à l'aide de l'algorithme de Babylone détaillé dans un document d'accompagnement.

À chaque seconde, les données acquises des capteurs ainsi que le résultat du calcul du module de l'accélération seront copiés dans un tampon en mémoire FLASH afin préserver les données en cas de panne de courant. Prévoyez 8 bits pour chacun de ces champs (accélération-x, accélération-y, accélération-z, accélération-module, potentiomètre). On note que la mémoire FLASH sur la carte MX3 est branchée au microcontrôleur via une interface série de type SPI. Pour indiquer le moment de l'acquisition des données dans chaque intervalle, on mettra également en mémoire une étiquette de temps en secondes écoulées depuis l'heure configurée à l'initialisation. Cette étiquette sera codée sous forme d'un entier positif (non signé) sur 32 bits.

Après les acquisitions des données pour 16 intervalles, on considérera que le tampon en mémoire FLASH est plein et qu'il faut le transmettre à un serveur externe avant d'effacer le tampon et de recommencer le processus. Avant de transmettre, on calculera les valeurs minimum, maximum et moyenne de chaque donnée (accélération-x, accélération-y, accélération-z, accélération-module et luminosité-potentiomètre) sur les 16 intervalles. Ces valeurs calculées devront être adjointes au paquet d'information à transmettre.

Deux types de communications seront exigées : par communication série UART et par communication parallèle implantée sur le PMOD. La communication UART permettra la transmission des données par le biais d'une connexion série câblée entre la carte MX3 et un ordinateur, mettant en œuvre un périphérique UART (*Universal Asynchronous Receiver Transmitter*) sur le microcontrôleur. On structurera les données pour qu'elles soient facilement lisibles par un humain à la console. Afin de minimiser les données à l'écran, il est permis de transmettre uniquement les valeurs minimum, moyenne et maximum du module de l'accéléromètre. Cette communication permet de visualiser facilement les données, ce qui est complémentaire au débogueur de l'outil MPLAB.

La communication sur le PMOD permet le transfert des données de la carte MX3 vers une autre carte qui n'est pas encore développée. On utilisera un mécanisme simple de transfert de données plusieurs bits à la fois (en parallèle). Il faudra réserver deux lignes pour contrôler et synchroniser le transfert. Ainsi, une ligne sera réservée du côté de la carte MX3 pour indiquer que les données sont disponibles sur le port PMOD et prêtes à être lues (STROBE), tandis qu'une ligne sera réservée du côté du récepteur pour indiquer que les données ont été reçues (ACK). Cette dernière (ACK) devra être contrôlée par une onde carrée à la fréquence de votre choix provenant d'un analyseur logique (*Analog Discovery*). Il est proposé que la fréquence de cette onde carrée (ACK) soit de l'ordre de 1 kHz (on vous demande de justifier pourquoi une fréquence d'un tel ordre de grandeur). On réservera

également une ligne servant à contrôler l'intégrité des données (bit de parité). Vous devrez consulter la documentation pour choisir les broches (*pins*) disponibles sur le port PMOD de votre choix. Vous devrez justifier le choix de vos broches. Pour aider au développement des modules, vous devez proposer des diagrammes d'états de l'émetteur et du récepteur. Comme les connecteurs PMOD ne fournissent que 8 lignes de données et que 3 lignes sont réservées au protocole, il faudra manipuler les données pour les transmettre de manière efficace en mots de 4 bits.

Tel qu'indiqué à la figure 2, le début du paquet de données devra comporter une entête fixe (0101), un champ indiquant la longueur du paquet (longueur exprimée en deux mots de 4 bits) et une étiquette de temps sur 32 bits (il s'agit ici du moment du transfert des données, pas celui d'une acquisition des capteurs comme mentionné précédemment). Le paquet se terminera par une somme de contrôle (« *checksum* »), laquelle servira à vérifier l'intégrité des données transmises. La somme de contrôle consistera tout simplement en un OU-exclusif de tous les mots de 4 bits composant le paquet de données. Le format de données complet préconisé est indiqué à la figure 2.

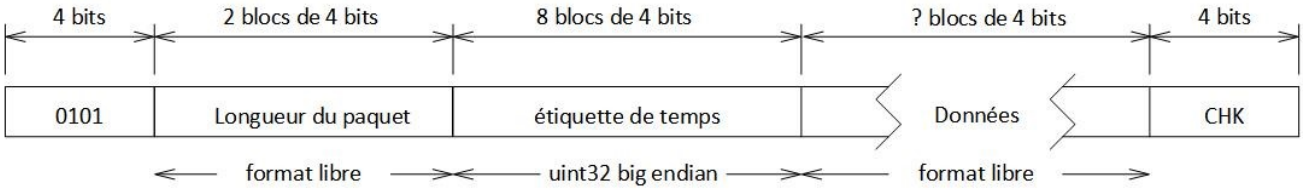


Figure 2 : Format préconisé du paquet de données.

Le développement se fera en général en C et vous pourrez vous prévaloir de bibliothèques de fonctions permettant de gérer facilement certains périphériques. Toutefois, certaines parties critiques devront être écrites en assembleur afin d'en optimiser la performance. C'est le cas du calcul du module des données de l'accéléromètre qui prendra comme paramètre d'entrée les valeurs X, Y, Z de l'accéléromètre et retournera la valeur du calcul du module. Par ailleurs, si le découpage et la mise en forme des données en vue de la transmission sur le port parallèle peut être fait en C, la gestion du port parallèle elle-même devra être faite en assembleur. Pour ces deux parties en assembleur, il sera nécessaire de faire des choix judicieux dans l'attribution des registres.

Bien que ceci ne sera pas évalué, le développement devra respecter les règles de l'art d'un processus d'assurance-qualité (notamment si vous voulez une aide de qualité), et comprendra :

- La planification de l'architecture globale de l'application embarquée sous forme de diagramme de séquence ou d'autres diagrammes UML;
- L'élaboration d'un processus d'intégration basé sur une approche MVP (*minimum viable product*);
- L'élaboration d'un plan de test incluant des tests unitaires et des tests d'intégration, ce qui permettra de cibler rapidement les problèmes lors de l'intégration et de procéder systématiquement dans votre développement et éviter de perdre du temps. Vous utiliserez un analyseur logique lorsque nécessaire.
- Vous devez remettre un rapport contenant la planification de votre intégration (MVP) incluant un plan de tests indiquant comment vous comptez tester chaque étape d'intégration ainsi que les résultats attendus.

5 CONNAISSANCES NOUVELLES

GEL452

Connaissances déclaratives (QUOI ?)

- Comprendre l'architecture d'un microprocesseur
- Distinguer entre microprocesseur et microcontrôleur
- Connaitre et pouvoir énumérer les différents types de périphériques

Connaissances procédurales (COMMENT ?)

- Utiliser le jeu d'instructions d'un microcontrôleur
- Écrire dans les registres d'un microcontrôleur à l'aide de masques
- Dans un environnement de développement :
    - Charger un programme
    - Déboguer un programme
    - Visualiser et modifier des variables
    - Visualiser les registres d'un microcontrôleur en opération lors de la mise au point d'un programme

Exploiter les périphériques internes d'un microcontrôleur

- Cadencer un microcontrôleur via des interruptions
- Écrire du code efficace et robuste
- Réaliser du calcul à virgule fixe
- Utiliser une bibliothèque

Connaissances conditionnelles (QUAND/POURQUOI ?)

- Utiliser des directives de compilation
- Protéger et restaurer le contexte lors de l'appel d'une fonction
- Utiliser une interruption

GEL442

Connaissances déclaratives (QUOI ?)

- Connaître les différents types d'interface série (I2C, SPI, one-wire, etc.)
- Connaître les différents types d'interface parallèles (convertisseurs A/N ou N/A, RAM)

Connaissances procédurales (COMMENT ?)

- Modéliser et concevoir des interfaces numériques en utilisant le langage VHDL
- Représenter mathématiquement l'information numérique discrète et optimiser des équations booléennes à l'aide de méthodes de synthèse logique
- Mesurer les performances d'échange d'information numérique

Connaissances conditionnelles (QUAND/POURQUOI ?)

- Quand utiliser un interface série ou parallèle et laquelle choisir selon l'application

6 GUIDE DE LECTURE

En contexte d'apprentissage par problème en ingénierie (APPI), les lectures à faire sont essentiellement déterminées par la formulation du problème et les connaissances nouvelles à acquérir et identifiées par le groupe lors du tutorat d'ouverture et dans le guide de l'étudiant. Il peut y avoir aussi des lectures complémentaires pour des connaissances procédurales et techniques associées et des révisions de connaissances antérieures.

La recherche d'information fait partie du processus d'apprentissage. Les stratégies pour une telle recherche sont variées et particulières à chaque personne. Dans le contexte de la présente unité d'APPI sur les microcontrôleurs et les interfaces, la quantité d'information dans les documents techniques est importante. C'est un des objectifs que l'apprenant réussisse à se retrouver dans toute cette documentation; c'est un défi certain.

Le présent guide est formulé pour orienter la recherche d'information dans cette grande quantité d'information et de documents et il n'est pas exhaustif.

Dans les documents techniques, il sera utile de recourir aux explorateurs de documents et aux indexes quand ils existent pour localiser facilement l'information recherchée.

6.1 Références essentielles à consulter

6.1.1 Ouvrages de référence

1. *Computer Organization and Design, The Hardware/ Software Interface*. Elsevier / Morgan Kaufmann, 5e édition, 2014, de D. Patterson et J. Hennessy
   Disponible sous forme électronique sur le campus de l'UdeS au lien suivant : https://ebookcentral.proquest.com/lib/usherbrookemgh-ebooks/detail.action?docID=5376640
2. Concepts de base de la programmation en assembleur MIPS par J. Rossignol
   Voir page WEB de l'APP, document : ConceptsBaseProgMIPS_JR_revYBL.pdf
3. Elements of real-time systems, Digilent, March 2017
   - https://digilent.com/reference/learn/courses/unit-2/start (ou fichier PDF Digilent_ElementsRTSystems_unit_2_2017.pdf)
4. Protocole I2C
   - https://www.ti.com/lit/an/slva704/slva704.pdf
   - https://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf
5. Bus SPI
   - https://www.analog.com/media/en/technical-documentation/application-notes/AN-1248.pdf
   - https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/
6. UART
   - https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

6.1.2 Documents d'accompagnement de la problématique

- Basys MX3™ Board Reference Manual, DOC# : 410-336, Digilent, 2017

- Schémas de la carte MX3
- Basys MX3 Libpack User Guide, DOC# : 502-xxx, Digilent
- PIC32MX Family Reference Manual, DS61132B, Microchip, 2008 (une version de Digilent avec les sections en fichiers séparés est disponible sur la page WEB de l'APP)
- PIC32MX370 Data Sheet, DS60001185G, Microchip, 2017
- MPLAB XC32 C-C++ Compiler User Guide, DS50002895A, Microchip, 2019
- MPLAB XC32 Assembler, Linker and Utilities User's Guide, DS50002186A, 2013
- Carte de référence MIPS (voir page web de l'APP ou manuel de Patterson et Hennessy)
- *Programming 32-bit Microcontrollers in C : Exploring the PIC32* de L. di Jasio, disponible à la bibliothèque de l'université (QA 76.6 D545 2008)
- MMA8652FC, 3-Axis, 12-bit, Digital Accelerometer, Doc# : MMA8652FC, NXP / Freescale semiconductor, 2015

6.2 Références complémentaires

Ces références sont fournies à des fins de documentation complémentaire et ne sont pas indispensables pour résoudre la problématique.

- Ressources complémentaires du manuel de Patterson et Hennessy
- http://booksite.elsevier.com/9780124077263/?ISBN=9780124077263
- See MIPS Run de D. Sweetman, disponible à la bibliothèque de l'université (QA76.9A73S93 2007)
- Site web de la firme MIPS, http://www.mips.com

6.3 Lectures à faire

Les lectures fournies sont assez volumineuses et il vous appartient de filtrer et de cibler les contenus les plus importants pour la résolution de la problématique, ainsi que la préparation en vue du laboratoire et des procéduraux.

6.3.1 Préparation pour le procédural 1 et le laboratoire

Le procédural 1 et le laboratoire sont le même jour. Il faut donc les avoir préparés à l'avance ensemble.

Lectures obligatoires pour le **procédural 1** (dans l'ordre) :

- Livre de Patterson et Hennessy
    - Assembleur et langage machine : chapitre 2, du début du chapitre jusqu'à 2.14, **sauf :** 2.11 et la partie sur Java dans 2.12 (pages 131 et 132).
      Note : Les sections 2.15 jusqu'à la fin du chapitre sont très intéressantes, mais ne sont pas requises ici.
      Si vous avez le temps et si ça vous intéresse, vous pouvez lire en ordre de priorité les sections suivantes :
        - le début et la partie sur la compilation en C de la section 2.15 (sauter toutefois ce qui concerne Java);
        - 2.21 sur un historique des jeux d'instruction, les architectures, les langages et les compilateurs (cette section est facile à lire et est très intéressante; vous devriez la lire pour votre culture générale).

        - la section 2.16 concernant ARMv7, la section 2.17 sur les instructions x86 d'Intel (qui sont les instructions de la grande majorité des ordinateurs que nous utilisons), la section 2.18 sur ARMv8, les sections 2.19 sur les pièges à éviter et 2.20 sur les remarques pour conclure.
- Introduction à MIPS avec SPIM (lien WEB ou fichier : QtSPIM_examples.pdf). Ce document est **important** car il contient beaucoup d'infos pratiques pour la programmation.
- Concept de base de la programmation MIPS par Julien Rossignol
- Livre de Patterson et Hennessy
    - Annexe A, section A.10 (ne lire les autres sections que si vous avez le temps)
    - Arithmétique des ordinateurs : chapitre 3, sections 3.1 à 3.3
      Note : Bien qu'elle soient intéressantes, on a exclu la section 3.4 sur la division et la section 3.5 sur les points flottants pour éviter de surcharger les lectures qui sont déjà volumineuses.
    - Architecture interne d'un microprocesseur : chapitre 4, 4.1–4.5
- Aides mémoires pour MIPS (cartes de référence MIPS; voir page WEB de l'APP ou manuel de Patterson et Hennessy), fichiers :
    - MIPS32_QRCMIPS_version2_AideMemoire.pdf
    - mips-ref_MIPS_version3_AideMemoire.pdf

Lectures obligatoires pour le **laboratoire** (dans l'ordre) :

- Basys MX3™ Board Reference Manual, DOC# : 410-336, Digilent, 2017
- Schémas de la carte MX3
- Basys MX3 Libpack User Guide, DOC# : 502-xxx, Digilent
- Elements of real-time systems, Digilent, March 2017, https://digilent.com/reference/learn/courses/unit-2/start (ou fichier PDF Digilent_ElementsRTSystems_unit_2_2017.pdf)
- Interruptions : Sections 8 (Interrupts) du manuel de référence du PIC32MX (PIC32MX Family ReferenceManual) et Section 14 du guide utilisateur de MPLAB (MPLAB XC32 C-C++ Compiler User Guide)
- Temporisateurs (timers) : Section 14 (Timers) du manuel de référence PIC32MX Family ReferenceManual
- UART : Sections 21 (UART) du manuel de référence du PIC32MX (PIC32MX Family ReferenceManual)
- ADC : Sections 17 (10-bit ADC …) du manuel de référence du PIC32MX (PIC32MX Family ReferenceManual)

6.3.2 Préparation pour le procédural 2

- Livre de Patterson et Hennessy
    - Circuits logiques : Annexe B, surtout B.10 pour les machines à états finis
- Certains documents d'accompagnement de la problématique devront être consultés à titre de référence
- Protocoles I2C et SPI
    - Protocole I2C
        - https://www.ti.com/lit/an/slva704/slva704.pdf
        - https://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf
        - Sections 24 (I2C) du manuel de référence du PIC32MX (PIC32MX Family ReferenceManual)

  - Bus SPI
    - https://www.analog.com/media/en/technical-documentation/application-notes/AN-1248.pdf
    - https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/
    - Sections 23 (SPI) du manuel de référence du PIC32MX (PIC32MX Family ReferenceManual)
- Fiche technique de l'accéléromètre de la carte Basys MX3 (MMA8652FCR1)

7 LOGICIELS ET MATÉRIEL

7.1 Matériel

Il faut le matériel suivant pour faire l'unité d'APP :

- Carte Basys MX3
- Module Analog Discovery 2

7.2 Logiciels utilisés

Il faut les logiciels suivants pour faire l'unité d'APP (pour l'installation, on installe d'abord MPLAB X et ensuite en fin d'installation, l'installateur mentionnera qu'il faut installer un compilateur, on installe alors XC32) :

- Logiciel de développement MPLAB X, Microchip
- Compilateur XC32, Microchip
- Librairie de soutien Basys MX3, https://github.com/Digilent/Basys-MX3-library

7.3 Logiciels d'intérêt

- Simulateur MARS (MIPS Assembler and Runtime Simulator), https://courses.missouristate.edu/KenVollmar/MARS/, simulateur recommandé comme outil d'apprentissage de l'assembleur sans avoir à passer par MPLAB, écrit en Java donc multiplateforme.
- Simulateur SPIM, https://spimsimulator.sourceforge.net, autre simulateur de l'architecture MIPS32, disponible en versions Windows, MacOS et Linux
- Communication série : Putty, CoolTerm

8 SANTÉ ET SÉCURITÉ

8.1 Dispositions générales

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance des politiques et directives concernant la santé et la sécurité. Ces documents sont disponibles sur les sites web de l'Université de Sherbrooke, de la Faculté de génie et du département. Les principaux sont mentionnés ici et sont disponibles dans la section Santé et sécurité du site web du département: http://www.gel.usherbrooke.ca/santesecurite/.

- Politique 2500-004 : Politique de santé et sécurité en milieu de travail et d'études
- Directive 2600-04 : Directive relative à la santé et à la sécurité en milieu de travail et d'études
- Sécurité en laboratoire et atelier au département de génie électrique et de génie informatique

9 SOMMAIRE DES ACTIVITÉS

Semaine 1

- Première rencontre de tutorat
- Étude personnelle et exercices
- Atelier d'introduction
- Formation à la pratique procédurale 1
- Formation à la pratique en laboratoire
- Formation à la pratique procédurale 2

Semaine 2

- Étude personnelle et exercices
- Présentation orale de la solution
- Validation pratique de la solution
- Deuxième rencontre de tutorat
- Évaluation formative théorique et pratique
- Évaluation sommative théorique et pratique

10 PRODUCTION À REMETTRE

- La résolution de la problématique se fait en équipe de 2.
- La présentation et le code doivent être remis avant 9h la journée de la validation.
- Pénalités de 10% par journée de retard
- Productions originales, sinon pénalité

Planification MVP et tests d'intégration

Décrire la planification de votre intégration (MVP) incluant un plan de tests indiquant comment vous comptez tester chaque étape d'intégration ainsi que les résultats attendus.

- Maximum 2 pages et sous forme d'un tableau.
- Chaque ligne du tableau doit indiquer une fonction à intégrer et au minimum un test / résultat attendu.
- Remis avant le procédural 2 afin d'être commenté en grand groupe à la fin du procédural 2.

Présentation

Vous devez présenter votre solution pour la problématique sous forme de présentation orale d'une durée de 10 minutes. Le tuteur posera des questions afin de valider votre compréhension. Voici les sujets que vous devez aborder durant votre présentation.

- Diagrammes UML de la solution sur le PIC32 :
    - Diagramme d'activité UML de la fonction principale (*main*) pour gérer les interruptions à chaque 1 ms.
    - Diagramme d'activité UML assembleur du calcul du module de l'accéléromètre.
    - Structure et stratégie de gestion du tampon en mémoire FLASH pour préparer les données à la transmission sur l'interface parallèle.
    - Diagramme d'activité UML assembleur pour la gestion du port parallèle du côté MX3, graphe de machine à états et justification des choix des broches / ports de communication.
- Graphique de la machine à états pour l'envoi et la réception du port parallèle (PMOD).
- Format précis du paquet de données tel que stocké dans la mémoire Flash.
- Plan de validation pour l'intégration des différents périphériques.

Validation

L'objectif ici est de démontrer que vous avez maîtrisé et complété le mandat selon les spécifications fournies.

Nous chercherons, en vous posant quelques questions, à valider à la fois votre démarche de conception et votre maîtrise des outils (la carte MX3, MPLAB, la programmation en C, l'usage de la librairie pour communiquer avec les périphériques et la programmation en assembleur). Chaque étudiant doit pouvoir répondre aux questions individuellement. Votre cahier de laboratoire doit être mis à disposition pour consultation afin d'évaluer votre démarche.

Vous devrez présenter les éléments suivants :

- L'interface permettant d'ajuster l'heure;
- La capture des données de l'accéléromètre (bus I2C à l'analyseur logique);
- Le calcul de la racine en assembleur;

- La conversion du signal du potentiomètre;
- Le bon fonctionnement de la console (via l'interface UART);
- Le paquet à transmettre en mémoire (dans le débogueur MPLAB ou à la console);
- Un paquet transmis sur le PMOD JA (utilisez l'analyseur logique et un bouton de la MX3 pour générer le ACK en lieu et place du slave).

Dépôt

Votre planification, votre présentation et vos codes doivent être déposés sur le serveur de dépôt dans les dossiers respectifs. Assurez-vous que votre archive porte le nom *cip1-cip2.zip*. L'absence de dépôt entraîne la note 0, peu importe le résultat de l'oral/validation. Tout retard est assorti d'une pénalité.

11 ÉVALUATIONS

11.1 Production à remettre

L'évaluation des productions à remettre portera sur les compétences figurant dans la description des activités pédagogiques. La pondération des différents éléments est indiquée au tableau suivant. L'évaluation est directement liée aux livrables demandés à la section 10 et le tableau suivant y réfère à l'aide d'une courte description.

Tableau 3 : Sommaire de l'évaluation de la présentation et de la validation

Éléments demandés	GEL442-1	GEL442-2	GEL452-1	GEL452-2
Présentation				
UML : Flash, PMOD	15			
UML : main, racine				15
Machine d'état communication PMOD	10			
Création du format des paquets Flash et PMOD				5
Retour sur votre plan d'intégration	10			5
Validation				
Code assembleur racine			10	
Code assembleur PMOD		15		
Code C pour la gestion des données à transmettre			5	
Fonctionnement UART, potentiomètre, accéléromètre		10		
Ajustement de l'heure			10	
Lecture du PMOD et de I2C sur Analog Discovery		10		
Total	35	35	25	25

Quant à la qualité de la communication technique, elle ne sera pas évaluée de façon sommative, mais si votre présentation est fautive sur le plan de la qualité de la communication et de la présentation, il vous sera retourné et vous devrez le reprendre pour être noté.

11.2 Évaluation sommative de l'unité

L'évaluation de l'unité d'APP3 est un examen écrit (théorique) et sur MPLAB (pratique) qui porte sur tous les éléments de compétences de l'unité. L'évaluation se fait avec la documentation sur ordinateur. L'évaluation pratique porte sur GEL442 et GEL452 tels que vus pendant l'unité et elle se fait sans documentation.

11.3 Évaluation sommative finale

L'évaluation finale est un examen écrit (théorique) et sur MPLAB (pratique) qui porte sur tous les éléments de compétences de l'unité. L'évaluation se fait avec la documentation sur ordinateur. L'évaluation pratique porte sur GEL442 et GEL452 tels que vus pendant l'unité et elle se fait sans documentation.

12 POLITIQUES ET RÈGLEMENTS

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance des politiques, règlements et normes d'agrément suivants.

Règlements et politiques de l'Université de Sherbrooke

- Règlement des études
  https://www.usherbrooke.ca/registraire/

Règlements facultaires

- Règlement facultaire d'évaluation des apprentissages / Programmes de baccalauréat
- Règlement facultaire sur la reconnaissance des acquis

Norme d'agrément

- Informations pour les étudiants au premier cycle :
  https://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg
- Informations sur l'agrément :
  https://engineerscanada.ca/fr/agrement/a-propos-de-l-agrement

Si vous êtes en situation de handicap, assurez-vous d'avoir communiqué avec le Programme d'intégration des étudiantes et étudiants en situation de handicap à l'adresse de courriel prog.integration@usherbrooke.ca.

13 INTÉGRITÉ, PLAGIAT ET AUTRES DÉLITS

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance de la déclaration d'intégrité relative au plagiat :
https://www.usherbrooke.ca/ssf/antiplagiat/jenseigne/declaration-dintegrite/

14 ATELIER D'INTRODUCTION

Buts de l'activité

- Apprendre les différents concepts importants dans l'utilisation d'un microprocesseur
- Se familiariser avec la carte Basys MX3 incorporant un microcontrôleur PIC32
- Se familiariser avec l'outil de développement MPLAB X de Microchip pour programmer un PIC32
    - Connaître le flot de compilation
    - Créer un projet MPLAB X
    - Inclure ou créer les fichiers .c, .h, … pour un projet de programmation à l'aide de MPLAB X
    - Compiler un projet et téléverser l'exécutable sur la carte Basys MX3
- Analyser et modifier un programme en C
- Utiliser la librairie de fonctions en C (LibPack) pour la carte Basys MX3.
- Utiliser les outils de simulation et de débogage
- Utiliser l'instrument de mesure Analog Discovery 2 (AD2)

L'atelier comportera des exemples simples nécessitant un peu de travail pratique en cours de séance.

Exemple 1 – Création d'un projet et allumer une DEL

Cet exemple permet une première familiarisation avec la programmation de la carte Basys MX3.

a) Créer un nouveau projet exemple1.X
b) Prendre l'archive exemple1_etudiant.zip disponible sur la page WEB de l'APP, la décompresser à un endroit de votre choix; cette archive contient les fichiers de code de départ main.c et config_bits.h qui permettent d'allumer une DEL.
c) Intégrer au projet exemple1.X les fichiers main.c et config_bits.h
d) Compiler le projet et téléverser l'exécutable sur la carte MX3.
e) La DEL 2 doit s'allumer.

Questions

1. Comment déterminer où sont branchées les différentes DEL ?
2. Placer un breakpoint et utiliser le débogueur et la surveillance (Watches) des SFR pour regarder l'état des broches (*pins*) à chacune des instructions.

Exemple 2 – Allumer une DEL avec la librairie led.h dans la librairie LibPack

Cet exemple montre qu'on peut faire la même chose qu'à l'exercice précédent, mais cette fois à l'aide de fonctions de la librairie LibPack de Digilent. Cette librairie est disponible sur la page WEB de l'APP.

a) Créer un nouveau projet exemple2.X.
b) Prendre l'archive exemple2_etudiant.zip disponible sur la page WEB de l'APP, la décompresser à un endroit de votre choix (l'archive contient 5 fichiers dont des fichiers de la librairie LibPack de Digilent pour accéder aux différents périphériques de la carte Basys MX3, ici les DEL).
c) Intégrer les fichiers de l'archive au projet exemple2.X (on peut simplement copier les fichier de l'archive dans le répertoire exemple2.X et intéger les fichiers dans ce répertoire au projet, c'est moins long qu'aller

les chercher dans un autre répertoire; l'outil d'exploration des fichiers de MPLAB X est peu performant, on dirait une ancienne version d'explorateur de fichier de Windows !)

d) Compiler le projet et téléverser l'exécutable sur la carte MX3.
e) La DEL 2 doit s'allumer.

Questions

1. Prendre le temps d'analyser le code et aller lire les fonctions de la librairie led.h et led.c.
2. Modifier le code pour faire allumer d'autres DEL.
3. À l'aide des outils de débogage, afficher le port A, utiliser le débogueur et la fonction « Step Into » et le Call Stack pour suivre le flot d'appel des fonctions.

Exemple 3 – Faire clignoter une DEL

a) Créer un nouveau projet exemple3.X.
b) Prendre l'archive exemple3_etudiant.zip disponible sur la page WEB de l'APP, la décompresser à un endroit de votre choix (l'archive contient 5 fichiers dont des fichiers de la librairie LibPack de Digilent pour accéder aux différents périphériques de la carte Basys MX3, ici les DEL).
c) Intégrer les fichiers de l'archive au projet exemple3.X.
d) Compiler le projet et téléverser l'exécutable sur la carte MX3.
e) La DEL 2 doit être allumée en permanence et la DEL 4 doit clignoter.

Questions

1. Mettre un breakpoint conditionnel lorsque le compteur dans le code a une valeur prédéfinie, p.ex. 8 (ne pas mettre une trop grande valeur, car ça peut être long !).

Tutoriel sur la carte Analog Discovery 2

L'Analog Discovery 2 (AD2) peut servir à la fois d'oscilloscope, de générateur de signal ou d'analyseur logique (logic analyser).
L'AD2 permet de décoder l'état d'un bus (ex. les données lues 0xF8) ou un protocole de communication (ex. I2C).

Le document du tutoriel sur l'AD2 est disponible sur la page WEB de l'APP, fichier :
AnalogDiscovery2_Tutoriel_DG_xxxx.docx

a) Prendre l'archive Tutoriel_AD2.X.zip disponible sur la page WEB de l'APP, la décompresser à un endroit de votre choix. Le fichier décompressé devrait être est un projet MPLAB X nommé Tutoriel_AD2.X
b) Ouvrir le projet fourni (on ne vous demande pas de créer un projet ici comme dans les exemples précédents; on saute cette étape ici).
c) Prendre le temps d'analyser le code de la fonction myTimerISR() dans le fichier main.c.
d) Compiler le projet et téléverser l'exécutable sur la carte MX3.
e) La DEL 0 doit clignoter et une broche du PMOD B doit générer une onde carrée de 1 kHz.
f) Avec l'AD2 vous devez observer la tension sur la broche du PMOD B. Suivre ce qui est indiqué dans le document du tutoriel pour l'utilisation de l'AD2. Dans ce qui est indiqué dans le guide, on utilisera l'AD2 en mode oscilloscope et en mode d'analyseur logique.

15 PRATIQUE PROCÉDURALE 1

Buts de l'activité

- Connaitre les principales caractéristiques des processeurs RISC en général et de l'architecture MIPS en particulier.
- Se familiariser avec le langage assembleur

- Structure d'un programme
- Éléments de base : instructions, opérandes, indentation, commentaires, directives, étiquettes
- Directives au compilateur

  - Se familiariser avec le langage assembleur de l'architecture MIPS

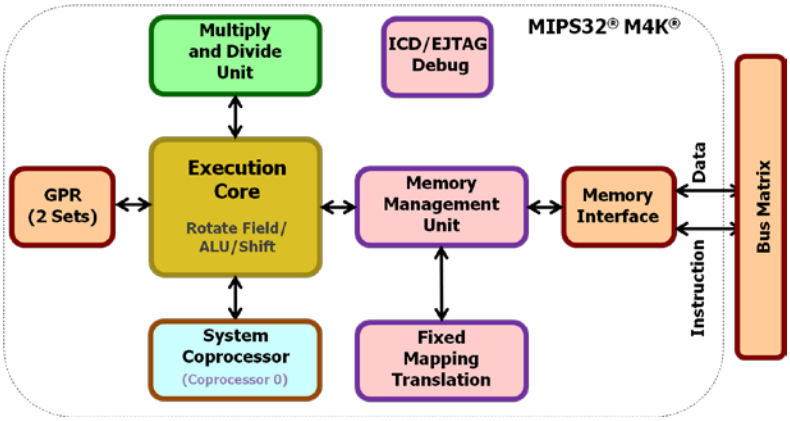
- Instructions natives et pseudo-instructions
- Comprendre l'utilisation des registres.
- Classes d'instructions

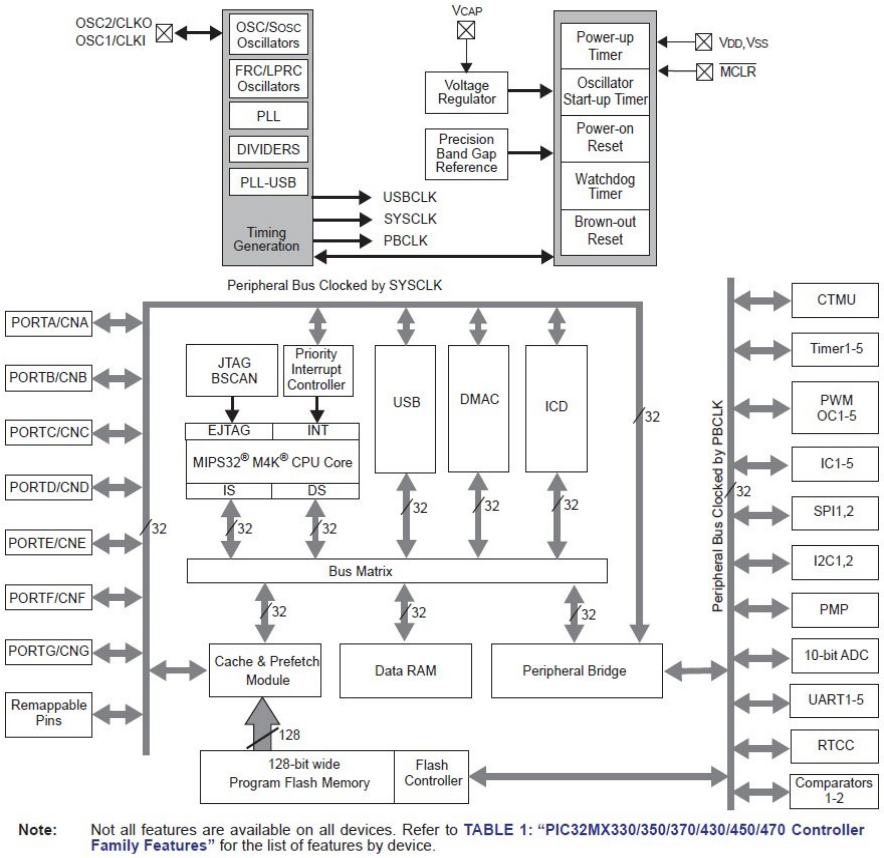
  - Connaître le flot de compilation.

Exercice 1 Architecture de processeur

a) Qu'est-ce qui distingue un microprocesseur, un microcontrôleur, un FPGA et un système embarqué?
b) Qu'est-ce qui distingue l'architecture de Harvard et celle de von Neumann ? Quels sont les avantages respectifs ?
c) Quelles caractéristiques devraient posséder un microprocesseur ou un microcontrôleur destiné à des applications de type "traitement du signal"? On parle alors d'un processeur de signal numérique (*Digital Signal Processor* ou *DSP*).
d) Quelles sont les caractéristiques des architectures de types RISC?
e) Déclinez les catégories d'instructions du jeu d'instruction de base de l'architecture MIPS32.
f) Quels aspects des architectures RISC permettent de réaliser des processeurs dont la circuiterie est plus simple?
g) Quelles conséquences cette réduction de complexité matérielle entraîne-t-elle sur d'autres parties du système?

Exercice 2 Architectures de processeurs



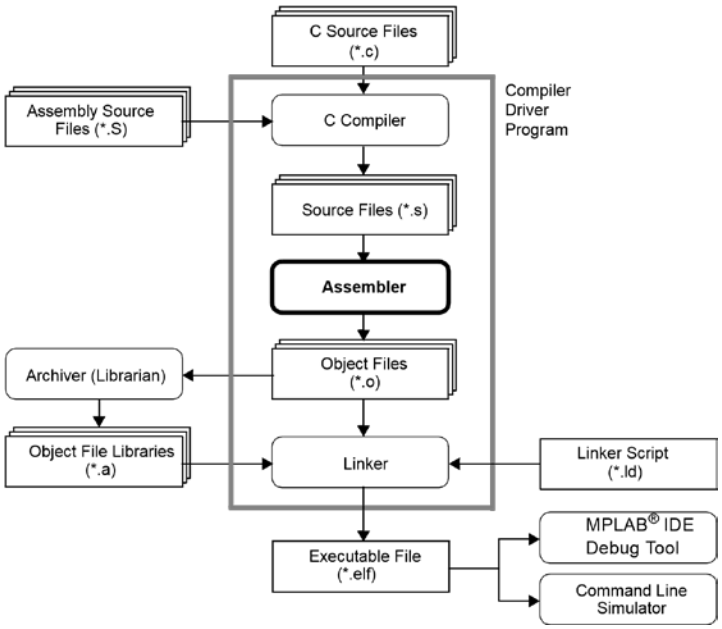


Note: Not all features are available on all devices. Refer to **TABLE 1: "PIC32MX330/350/370/430/450/470 Controller Family Features"** for the list of features by device.

a) Identifiez les principaux éléments du coeur de traitement M4K dans l'architecture du PIC32 et les liens entre eux.
b) Représentez à l'aide d'un diagramme en bloc de haut niveau le chemin de données (*datapath*) du coeur de traitement M4K.
c) Qu'est-ce qui distingue l'architecture de Harvard et celle de von Neumann?
d) Quels sont les 5 étages du pipeline de coeur M4K et comment opèrent-ils en fonctionnement normal?
e) Que se passe-t-il si une instruction a besoin du résultat de l'instruction précédente?
f) Pourquoi y a-t-il plusieurs niveaux de cache dans les microprocesseurs à haute performance? À quoi sert le cache dans l'architecture PIC32?
g) En incluant le coeur M4K, identifiez les principaux modules de l'architecture PIC32.

Exercice 3 Flot de compilation

Commenter le rôle de chacune des étapes d'une compilation à partir de la figure suivante.



Exercice 4 Instructions, adressage et exécution

a) Quelle est la différence entre le code assembleur et le langage machine?
b) Quels sont les 2 types de commandes dans un programme en assembleur?
c) Un programme en assembleur comporte au moins 2 zones. Quelles sont-elles et à quoi servent-elles?
d) Quelle est la différence entre une variable en C et un registre en assembleur?
e) Quels sont les 4 modes d'adressage de l'architecture MIPS?
f) Qu'est-ce qu'une pseudo instruction? Identifier les pseudo-instructions dans le code assembleur de l'exercice 5.
g) Que produira l'assembleur avec la pseudo-instruction ble $t0, $t1, etiquette ?
h) Qu'est-ce qu'une directive (.text, .globl, .main, .data, …)
i) Il existe 3 formats d'instructions en langage machine dans l'architecture MIPS pour ce qui est de la manière et l'ordre de déclinaison des opérandes de source et de destination.
    a. Donnez la spécification de ces 3 formats.
    b. Déterminez le registre de destination dans chacune des instructions suivantes :
        i. lw $s1, (10)s2
        ii. add $t1, $t2,$t3
        iii. addi $t2, $t1, 10
j) Expliquer ce que font les instructions suivantes :
    a. lw $s1, 0($t1)
    b. add $t1, $t2, $t3
    c. addi $t2, $t1, 10
    d. ble $t0, $t1, boucle
    e. mult $t0, $t1
    f. sb $s1, 10($a0)

Exercice 5 Analyse d'un programme en assembleur

Soit le programme C suivant qui calcule la somme des 15 premiers entiers positifs en stockant les résultats intermédiaires dans un tableau :

```c
int A[16];

int main() {
        int n, s;               /* n est l'index et s est la somme */
        s=0;                    /* initialisation de la somme */
        for (n=1; n < 16; n++) {
                s=s+n;          /* ajout de l'index à la somme */
                A[n]=s;         /* résultat intermédiaire stocké */
        }
}
```

Voici un programme effectuant la même tâche, en langage assembleur MIPS :

```
.data 0x10010000
Ici:    .space 16

.text 0x400000

.globl main

main:
        li $t0, 0
        li $t1, 0
        sb $t1, Ici($0)
        li $t0, 1
boucle:
        add $t1, $t1, $t0
        sb $t1, Ici($t0)
        addi $t0, $t0, 1
        blt $t0, 16, boucle

        li $v0, 10
        syscall
```

a) Identifiez les segments de données et d'instructions de ce programme.
b) Expliquez le rôle des éléments suivants : .data, .byte, .text, .globl, main:, boucle:, Ici:
c) Quels éléments du programme assembleur correspondent aux variables en C (n, s et le tableau A)?
d) Expliquez le rôle et le fonctionnement de la commande syscall?
e) À quelle adresse mémoire sont chargées les instructions de ce programme?
f) Quels mécanismes peuvent être utilisés en assembleur pour représenter une variable?
g) Ce programme utilise l'instruction **li** (load immediate) pour charger une constante dans un registre. Si la constante avait été récupérée d'une adresse en mémoire, on aurait pu utiliser la commande **lb** (load byte) pour charger un octet. Quelles sont les différences entre les instructions **lb**, **lbu** et **lw**?
h) Quelle est la dimension en octets de ce programme?
i) Faire un ordinogramme et donner le pseudocode de ce programme.

Exercice 6 Traduire un programme C en assembleur

Soit le code C suivant :

```
1       int valeurs[12]={4, 20, 6, 2, 1, 1, 10, 15, 8, 36, 5, 7};
2       int bornes[2]={2, 5};
3
4       int main() {
5               int n, s;               /* n est l'index et s est la somme */
6               s=0;                    /* initialisation de la somme */
7               n=bornes[0];            /* initialisation de l'index */
8               while (n <= bornes[1]) {
9                       s=s+valeurs[n]; /* ajout de la nième valeur à la
                                           somme */
10                      n=n+1;          /* incrémentation de l'index */
11              }
12      }
```

a) Quelle devrait être la valeur de s à la fin de l'exécution ?
b) Proposez un programme assembleur équivalent.

Exercice 7 Manipulation de bits et masquage

Les opérations logiques sur les bits, y compris les décalages, sont les seules façons de manipuler les bits individuellement à l'intérieur d'un registre. L'une des manipulations les plus courantes consiste à appliquer un "masque" par le biais d'une opération logique "ET." Le masquage permet de forcer certains bits sélectionnés à zéro.

a) Donnez le résultat des opérations logiques suivantes
    a. 0101 AND 0011
    b. 0101 OR 0011
    c. 0101 XOR 0011
b) Étant donné la valeur suivante dans un registre : 01100110, proposez une opération logique
    a. pour forcer à un le bit 3 (en sachant que le bit 0 est le bit de poids faible à la droite par convention)
    b. pour modifier l'état des bits 5 et 6
    c. pour modifier l'état des bits 4 et 5.
c) En supposant que le registre $t8 d'un processeur MIPS contient a priori la valeur hexadécimale FFFF000A. Pouvez-vous proposer une instruction pour modifier le contenu de $t8 de manière à obtenir 8FFF0008 ? Écrivez un programme assembleur complet, qui charge au préalable la valeur dans $t8, puis applique le masquage.
d) Si on ne connait pas à l'avance la valeur contenue dans le registre $t8, proposez une instruction logique en assembleur MIPS pour
    a. forcer les bits 10, 14 et 15 à '1'
    b. forcer les bits 21, 12 et 8 à '0'
    c. faire basculer l'état des bits 13, 12, 4 et 3.
e) Pour modifier les états des broches 0 (mettre à 1) et 4 (mettre à 0) du PMOD A. Quelles sont les étapes pour faire ceci en code assembleur. Indice : Vous devez utiliser les fonctions lecture et l'écriture en plus des fonctions de logique.

16 PRATIQUE EN LABORATOIRE

Buts de l'activité

- Se familiariser avec le langage assembleur

- Structure d'un programme
- Éléments de base : instructions, opérandes, indentation, commentaires, directives, étiquettes
- Directives au compilateur

- Se familiariser avec le langage assembleur de l'architecture MIPS

- Instructions natives et pseudo-instructions
- Classes d'instructions

Exercice 1 Conception d'un programme en assembleur

Écrivez un programme en assembleur qui détermine la nombre d'interrupteurs dans la banque SW0–SW7 qui sont à la position « ON » et qui allume un bloc de DEL dans la banque LD0–LD7 dont la longueur correspond à ce nombre. Comme il y a au total 8 interrupteurs et que les DEL et les interrupteurs sont indexés à partir de 0, on allumera toutes les DEL (jusqu'à LD7) si les 8 interrupteurs sont actifs, et seulement LD0 si un seul interrupteur est actif. P.ex., si 4 interrupteurs sont à « ON » (peu importe lesquels), alors les DEL 0 à 3 seront allumées; si aucun interrupteur n'est actif, alors toutes les DEL doivent être éteintes.

a) Dans le code de départ du fichier main.c du projet Labo1.X, compléter la fonction fct_C() qui prend pour argument le tableau de l'état des interrupteurs valueSW et retourne l'indice de la DEL à allumer noLED. Le corps de la fonction doit comporter une boucle qui compte le nombre d'interrupteurs actifs. La fonction fct_C() qui doit avoir les spécifications suivantes :

Entrée
Le tableau de l'état des interrupteurs
Fonction
Une boucle qui compte le nombre d'interrupteur à « ON ».
Sortie
Le numéro le plus élevé de la DEL à allumer

b) Modifier votre code en complétant le code en assembleur dans le fichier fct_S.s qui remplacera la fonction fct_C() et qui sera appelée par main.c au lieu de la fonction en fct_C(). Il faut prévoir l'utilisation correcte des registres dans la fonction en assmbleur pour les entrées et la sortie de la fonction (par référence ou par valeur).

Exercice 2 Utilisation d'une interruption et configuration d'un temporisateur (minuterie ou *timer*)

Le but de cet exercice est de modifier le code de départ dans le fichier main.c du projet Labo_Exercice2_Interrupt_timer.X afin que la DEL change d'état à toutes les secondes. On procèdera par étapes.

a) Commencer par modifier la fonction d'initialisation qui configure la fréquence des interruptions pour que cette dernière soit de 1 kHz (à toutes les millisecondes). Pour ce faire :
   - réviser la mécanique d'interruptions de l'architecture MIPS32 et les éléments de configuration logicielle associés
   - optimiser le diviseur de PBCLK, le prescaler du timer et la valeur du registre PR en fonction de l'horloge principale SYSCLK

b) Changer le code (sans changer sa fonction) pour que le compteur soit maintenant mis à jour dans la routine de service d'interruption __ISR de manière à ce que la variable globale Flag_1s de type static volatile int soit mise à '1' une fois par seconde.

c) Ajouter une fonction en C, puis en assembleur pour générer une onde carrée de 500 Hz sur le PMOD B, puis valider la fréquence avec le module Analog Discovery.

d) En utilisant les fonctions de la bibliothèque adc.h, modifier le code en C pour que la période en millisecondes de l'onde carrée soit égale à la valeur du potentiomètre, puis valider la fréquence avec le module Analog Discovery.

Exercice 3 Console UART

a) Brancher le port USB de la carte MX3 à l'un des ports USB de votre ordinateur. Utiliser un émulateur de terminal (comme PuTTY, Multiway, CoolTerm ou autre — voir « Autres logiciels d'intérêt » sur la page WWEB de l'APP) et configurer correctement le port série (voir section 8.3 de Lab 4a: Universal Asynchronous Receiver / Transmitter) pour établir la communication. Lancer le code fourni (projet Labo_Exercice3_UART.X) et vérifier que vous pouvez communiquer en tapant un message quelconque sur la console et en appuyant sur <ENTER>. La carte MX3 devrait par défaut vous retourner votre message.

b) Est-ce que la fréquence de PBCLK choisie à l'exercice précédent convient au baudrate demandé ? Quelle serait la valeur du registre BRG dans ce cas ?

c) Modifier le code pour reconnaître une séquence préprogrammée à l'avance et répondre de manière unique. Par exemple, taper "Bonjour" à la console pourrait provoquer la réponse "Au revoir" de la part de la carte MX3.

d) Modifier le code pour que la carte réponde à la commande "Allume n" où n est un chiffre de 0 à 7 en allumant la DEL correspondante (LDn).

Exercice 4 Horloge

a) À partir du projet Labo_Exercice4_Horloge.X, étudier le fonctionnement du code main.c, faites-le tourner sur la carte et vérifier que vous avez bien une horloge qui fonctionne.

b) Modifier le code pour qu'il soit possible de "figer" l'heure à l'écran avec un bouton, puis de la relancer en appuyant à nouveau sur le bouton. L'heure continue d'être mise à jour à l'interne en toute circonstance, de sorte que l'heure correcte est affichée lorsque l'horloge est relancée, peu importe la durée pendant laquelle l'affichage était figé.

17 PRATIQUE PROCÉDURALE 2

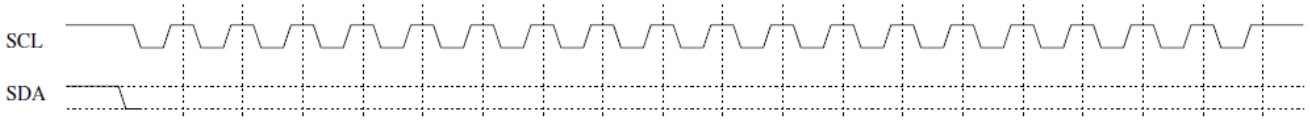
Buts de l'activité

- Comprendre comment vérifier la compatibilité d'interfaces électriques entre deux dispositifs.
- Comprendre le fonctionnement et l'utilité d'un tampon de mémoire dans un mécanisme de communication numérique (interface).
- Se familiariser avec les mécanismes de communication et d'interfaçage asynchrones entre deux dispositifs indépendants, y compris :
    - la modélisation des comportements de part et d'autre par des graphes de machines à états;
    - la différence entre une réalisation sous forme de circuits sur FPGA et sous forme de logiciel sur microcontrôleur;
    - les mécanismes de coordination entre transmetteur et récepteur.
- Savoir identifier et optimiser les fonctions combinatoires qui interviennent dans les interfaces et les mécanismes d'accès à la mémoire.
- Se familiariser avec les protocoles I2C et SPI.

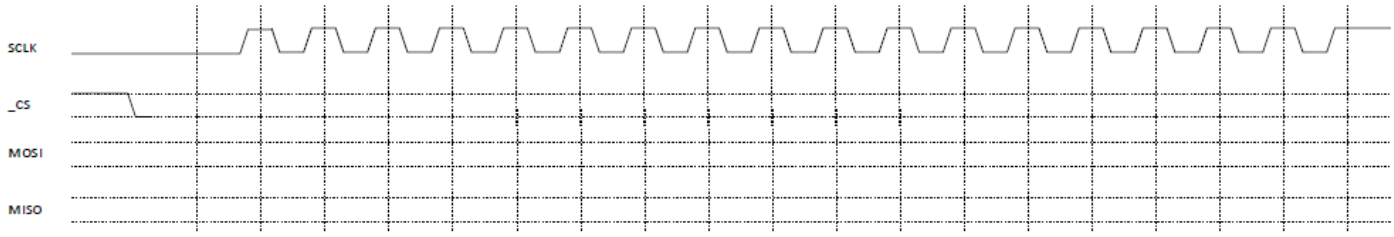
Exercice 1 Protocoles I2C et SPI

On souhaite envoyer un octet à un esclave via le protocole I2C ou le protocole SPI. Les paramètres sont les suivants : Adresse de l'esclave : 0x32 et données à transmettre : 0x8A

a) Quelle est la séquence précise d'opérations à réaliser pour effectuer ce transfert selon le protocole I2C?
b) Complétez le chronogramme ci-dessous pour illustrer le transfert complet, en annotant au besoin.



c) En quoi la séquence d'opérations diffère-t-elle si on souhaite effectuer un transfert équivalent via le protocole SPI?
d) Complétez le chronogramme correspondant au transfert de la donnée 0x8A vers le registre 0x02 de l'esclave.



Exercice 2 Configuration d'un périphérique (ADC)

On veut utiliser l'ADC du PIC32MX pour échantillonner le potentiomètre de la carte MX3. Déterminer où trouver dans la documentation l'information utile à chacune des étapes, soit :

a) Comprendre l'architecture et le fonctionnement du périphérique.
b) Comprendre les branchements physiques.

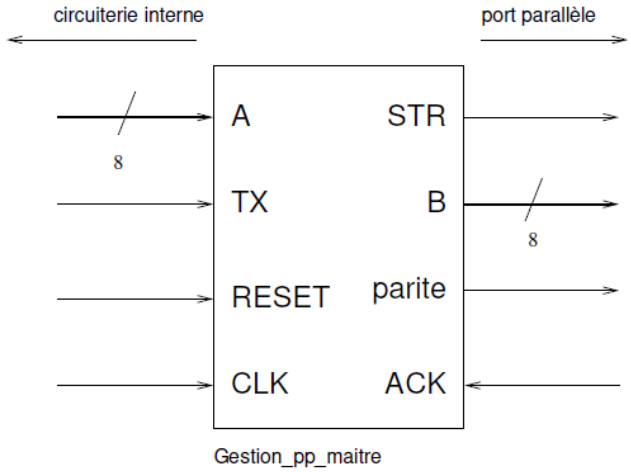
c) Comprendre comment configurer les pattes appropriées.
d) Comprendre comment configurer les registres des périphériques en fonction des spécifications et de l'application, en particulier l'horloge, le format de sortie et la commande de début de conversion.
e) Comprendre l'interface de la librairie fournie par Digilent par rapport aux éléments précédents.

Exercice 3 Compatibilité électrique des interfaces

a) Le connecteur PMOD A de la carte MX3 est raccordé aux lignes RC1–4 et RG6–9 du PIC32. D'après la fiche signalétique (datasheet) du PIC32 et les schémas de la carte MX3, si ces lignes sont configurées en sortie, quelles sont les plages de tension qui représentent le niveau '0' et le niveau '1'? Assumez que le courant en sortie est inférieur à 9 mA pour le niveau '0' et supérieur à -7 mA pour le niveau '1'.
b) Si les lignes sont configurées en entrée, quelles sont les plages de tension admissibles pour représenter un niveau '0' et un niveau '1'?
c) Les ports du PIC32 sont-ils compatibles avec un périphérique qui répond au standard LVTTL? LVCMOS?

Exercice 4 Port parallèle

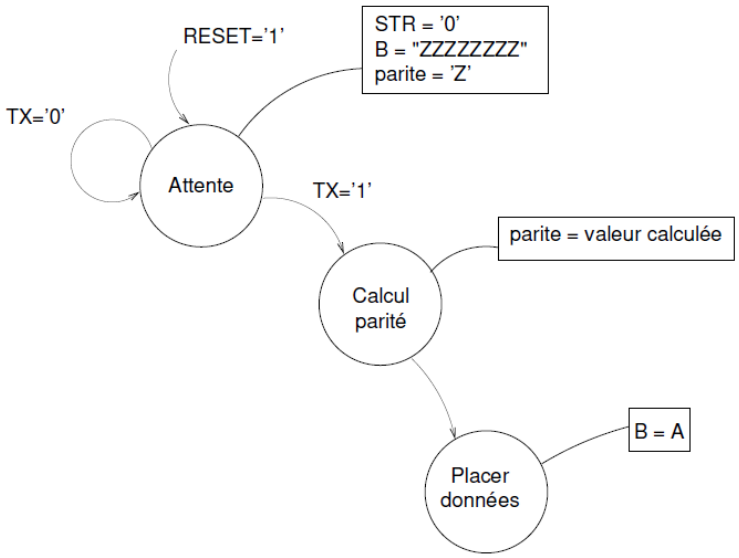
On conçoit un port parallèle comportant 8 bits de données pour permettre à deux systèmes électroniques distincts de communiquer à haut débit. En plus des 8 lignes de données, on aura 3 lignes réservées à la gestion de la communication, soit la ligne STR ("strobe"), ACK ("acknowledge") et PAR (parité). Ce port parallèle n'est conçu que pour fonctionner dans un seul sens, c'est-à-dire que le système 1 (ou maître) transmet des données au système 2 (ou esclave).



Imaginons que la logique de commande du port parallèle du côté maître est réalisée sous forme de circuit logique. Cela pourrait prendre la forme du module illustré à la figure précédente. Pour un premier prototype de protocole, on détermine les comportements suivants :

- Lorsque la commande TX est à '1', le module prend les données en A et les place sur le port parallèle en B.
- Lorsque le maître a placé 8 bits à transmettre (avec un 9e bit de parité approprié) sur les lignes en sortie, il envoie une impulsion d'une durée d'un coup d'horloge sur la ligne STR pour signifier à l'esclave que les données sont prêtes à être lues.
- Lorsque l'esclave voit l'impulsion sur STR, il récupère les données, valide le bit de parité, et indique que la lecture est complétée en envoyant une impulsion d'un coup d'horloge sur la ligne ACK. Ceci signifie donc que l'esclave est prêt à recevoir de nouvelles données.

- Pour simplifier l'architecture, rien pour l'instant n'indique à la logique interne qu'un ACK est reçu, ou au contraire qu'un ACK n'est jamais venu.



a) La figure précédente donne un graphe partiel de la machine à états pour le module de gestion du côté maître. Complétez le graphe en observant les règles suivantes :
   - Les conditions sont associées aux transitions et les sorties aux états eux-mêmes.
   - Si la condition sur une branche est vraie, cette transition est empruntée à la prochaine montée d'horloge.
   - Une branche sans condition est toujours empruntée à la prochaine montée d'horloge.
   - Lorsqu'une sortie est activée, elle conserve sa valeur jusqu'à ce que celle-ci soit modifiée.

b) Selon vous, est-ce important d'initialiser les sorties B et parité à 'Z'? Expliquez.

c) Pourrait-on accélérer les choses et faire B <= A et STR <= '1' au même état dans le graphe?

d) Développez le graphe de machine à états correspondant.

e) Tel que spécifié au début de la question, ce port parallèle fonctionnerait si les deux systèmes étaient alimentés par la même horloge et étaient parfaitement synchronisés. Si l'esclave a une fréquence d'horloge f_2 qui diffère de celle du maître f_1 selon l'équation

$$f_2 = C f_1,$$
$$f2 = Cf1,$$

où C est un facteur quelconque, pendant combien de coups d'horloge pensez-vous qu'il faudrait maintenir la ligne STR à 1 pour être sûr que son état soit détecté par l'esclave? Incluez un facteur de sécurité dans votre réponse d'au moins 5% de la fréquence d'horloge de l'esclave.

f) Pouvez-vous imaginer une variante de ce protocole, avec les mêmes lignes et les mêmes rôles, qui peut fonctionner de manière fiable même si on ne connait pas la fréquence d'horloge de l'esclave et même si il y a des délais sporadiques dans le système? Que devrait-on changer pour cela?

g) Un processus tel que décrit par les graphes de machines à états que vous avez produits peut-il être implémenté directement en assembleur sur le microcontrôleur? Est-ce que le comportement serait exactement le même?