Machine de Turing

ALLEGRE–COMMINGES Clément et BROUARD Romain

Table des matières

Introduction

1.1 Un peu d'histoire...

En 1928, Le mathématicien allemand David Hilbert énonce le "problème de la décision". Il se demande s'il est possible de trouver une méthode « effectivement calculable » pour décider si une proposition est démontrable. Pour résoudre ce problème, il faut caractériser ce qu'est un procédé effectivement calculable. C'est alors qu'Alan Turing, alors en thèse à Cambridge, conceptualise une machine universelle et prouve grâce à cette dernière que le problème de l'arrêt est indécidable, ce qui permet de donner une réponse négative au problème d'Hilbert pour l'arithmétique. C'est alors qu'Alan Turing introduit les concepts de programme et programmation.
Ce concept de machine universelle, que nous appellerons désormais Machine de Turing Universelle (MTU) n'est pas réalisable puisqu'il s'agit d'un objet mathématique, dont on va détailler le fonctionnement plus tard. Néanmoins, une Machine de Turing à état fini peut-être construite, et la première vit donc le jour à Bletchley Park pendant la Seconde Guerre Mondiale, où Turing lui-même et une équipe de scientifique triés sur le volet par le MI6 [1] construisirent une Machine de Turing pour casser les codes allemands générés par la machine Enigma.

1.2 Fonctionnement d'une MTU

Une MTU se compose de quatre éléments essentiels :

- un ruban de taille infinie, divisé en cases.
- une tête de lecture/écriture (qu'on appellera simplement tête de lecture, même si elle permet également d'écrire sur le ruban)
- un état interne
- une table de transition.

Une machine de Turing traite des symboles. L'ensemble des symboles est appelé Alphabet.
Le ruban permet d'accueillir des symboles qui seront lus et écrits par la tête de lecture.
La tête de lecture permet de lire et écrire un symbole. Elle peut se déplacer vers la gauche ou la droite.

Une machine de Turing fonctionne donc de la manière suivante :

1. La tête de lecture lit le symbole qu'elle pointe sur le ruban.
2. En fonction de son état et du symbole lu, la tête de lecture écrit un symbole à la place de celui qu'elle a lu précédemment.

1. Services de renseignement britanniques

3. Un déplacement est choisi en fonction de l'état de la tête de lecture et du symbole lu.
4. La tête de lecture change d'état.
5. Le ruban se déplace vers la droite ou vers la gauche selon le déplacement choisi précédemment.
6. Puis on recommence depuis le 1.

On peut donc résumer le fonctionnement comme cela : à chaque "cycle", on choisi un symbole à écrire, un nouvel état pour la tête de lecture, et un déplacement, en fonction d'un symbole lu et de l'état actuel. On peut donc définir une fonction de transition, qui va se charger de déterminer l'état futur d'une MTU en fonction de ton état courant. L'ensemble des fonctions de transition permettant de traiter un "mot" peut être représenté sous la forme d'une table de transitions ou d'un graphe de transitions.
Un mot traité est dit accepté si une Machine de Turing s'arrête en état final après l'avoir intégralement traité.
Une MTU possède forcément un nombre fini d'états, et elle a au moins deux états obligatoires : q_O l'état initial, et F un ensemble d'états d'acceptation.

Une machine de Turing est donc définie par :

- Q un ensemble fini d'états.
- q_0 un état initial tel que $q_0 \in Q$.
- F un ensemble d'états d'acceptation tel que $F \subseteq Q$.
- Γ un ensemble fini de symboles.
- Σ un ensemble fini de symboles d'entrée tel que $\Sigma \subseteq \Gamma$.
- B un symbole de ruban vide tel que $B \in \Gamma \setminus \Sigma$.
- δ une fonction de transition.

Une fonction de transition se formalise donc comme ceci :

$$\delta(q, Z) \to (p, Y, D)$$

avec q l'état de la tête de lecture, Z le symbole pointé, p le nouvel état, Y le nouveau symbole et D le déplacement.

Exemple d'une table de transition pour une Machine de Turing acceptant le langage $L = \{a^k b^k \mid k > 0\}$ avec q_0 comme état initial (représenté par une $\to$), et q_4 comme état d'acceptation (représenté par $*$) :

	symboles				
	a	b	X	Y	Blank
→ q0	(q1, X, R)			(q3, Y, R)	
q1	(q1, a, R)	(q2, Y, L)		(q1, Y, R)	
q2	(q2, a, L)		(q0, X, R)	(q2, Y, L)	
q3				(q3, Y, R)	(q4, B, R)
* q4					

Le but de ce projet va donc être de concevoir une Machine de Turing telle que définie ci-dessus.

Conception

2.1 Cahier des charges

Le cahier des charges est composé de trois parties. La partie 1 doit être traitée absolument, les parties 2 et 3 seront traitées si le temps nous le permet.

2.1.1 Partie 1

Dans un premier temps, le système conçu doit être capable de :

- exécuter un programme prédéfini (codé en dur dans le programme microcontrôleur).
- avoir un mode pas à pas pour l'exécution du programme.
- avoir un mode continu pour l'exécution du programme.
- gérer l'affichage de l'état du ruban.
- gérer l'affichage de la position de la tête de lecture.
- gérer l'affichage de la table de transition.

2.1.2 Partie 2

Dans un second temps, il faut rajouter :

- la possibilité de sélectionner un programme via un menu.
- le stockage des programmes à sélectionner.
- l'initialisation manuelle du ruban et de la position de la tête de lecture.

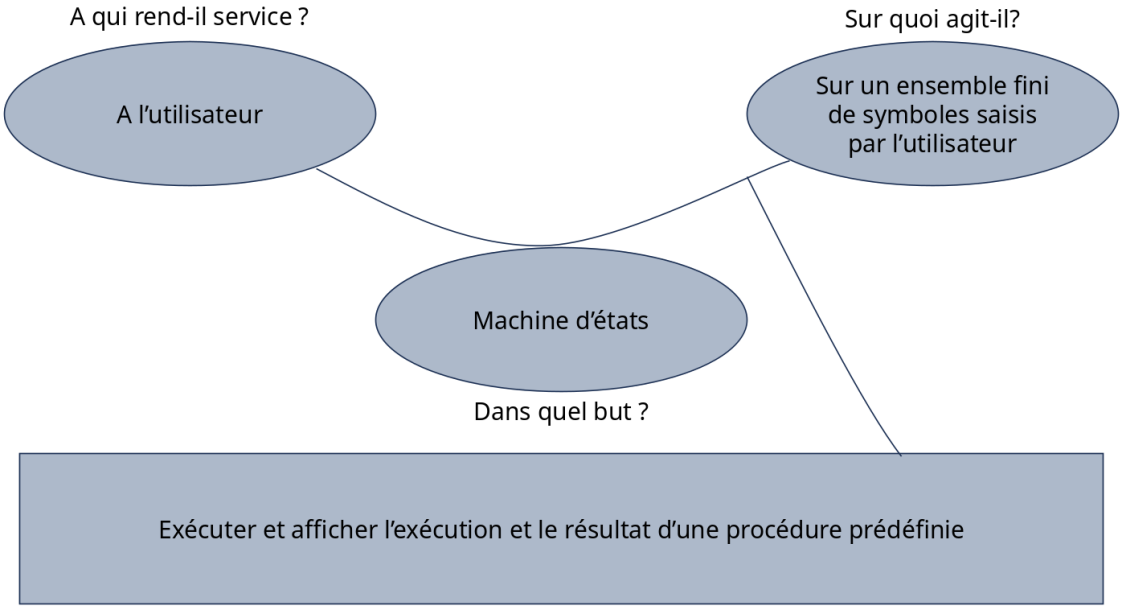
2.1.3 Partie 3

Enfin pour obtenir un système complet, il faut implémenter :

- la programmation directement sur la machine d'une table de transition.
- l'enregistrement de la table de transition programmée dans le support de stockage.
- un reset de la programmation de la ligne en cours.
- l'affichage d'une description du programme.

2.2 Définition des besoins

2.2.1 Diagramme bête à corne



2.2.2 Matrice Moscow

Must Have

- Machine de Turing capable au moins de faire l'addition de 2 nombres
- Mode continu/pas à pas pour l'exécution du programme
- Affichage de l'état du ruban
- Affichage de la position de la tête de lecture
- Gérer l'affichage de la table de transition

Should Have

- La possibilité de sélectionner un programme via un menu
- Stockage des programmes à sélectionner
- Initialisation manuelle du ruban et de la position de la tête de lecture.

Could Have

- Programmation directement sur la machine d'une table de transition
- Enregistrement de la table de transition programmée dans le support de stockage
- Reset de la programmation de la ligne en cours
- Affichage d'une description du programme

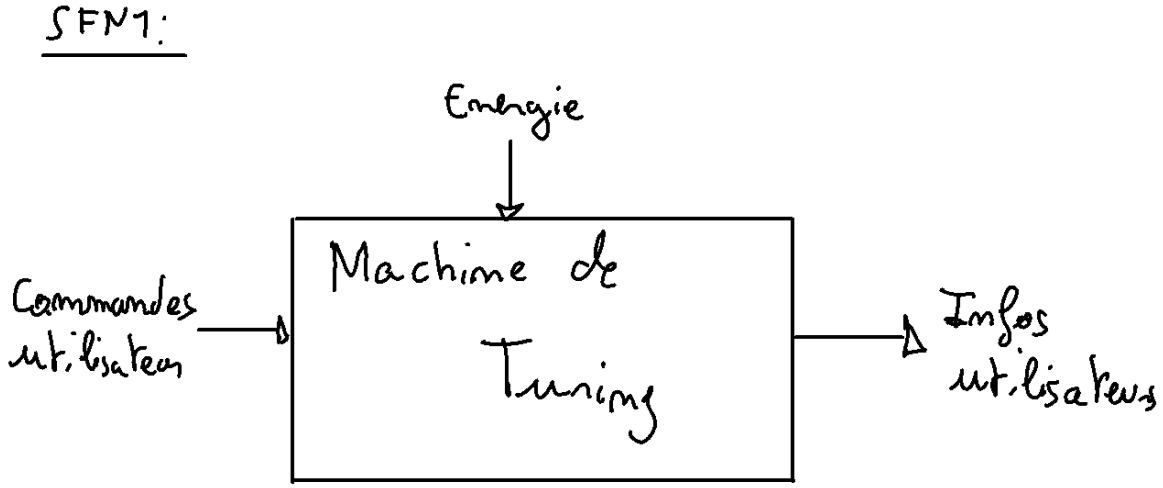
Won't Have

2.3 Conception Matérielle

Pour concevoir notre système, nous avons décidé de traiter les trois parties en même temps, ce qui nous évite de devoir repasser par une phase de conception et d'adaptation lors de la réalisation des parties 2 et 3.

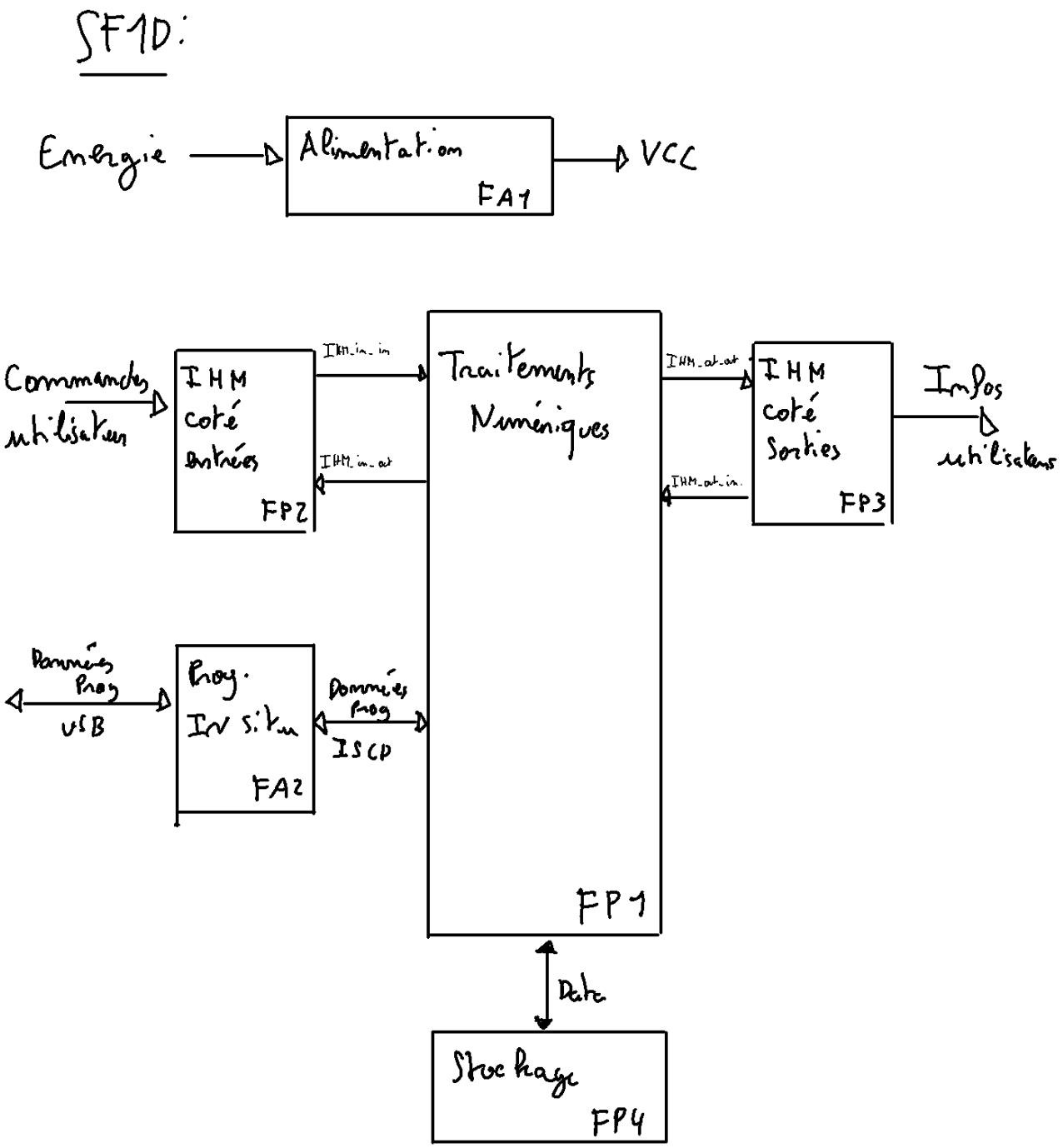
2.3.1 SFN1

Nous avons donc commencé par dessiner un Schéma fonctionnel de premier niveau, pour définir les différentes entrées et sorties de notre système :



2.3.2 SFnD

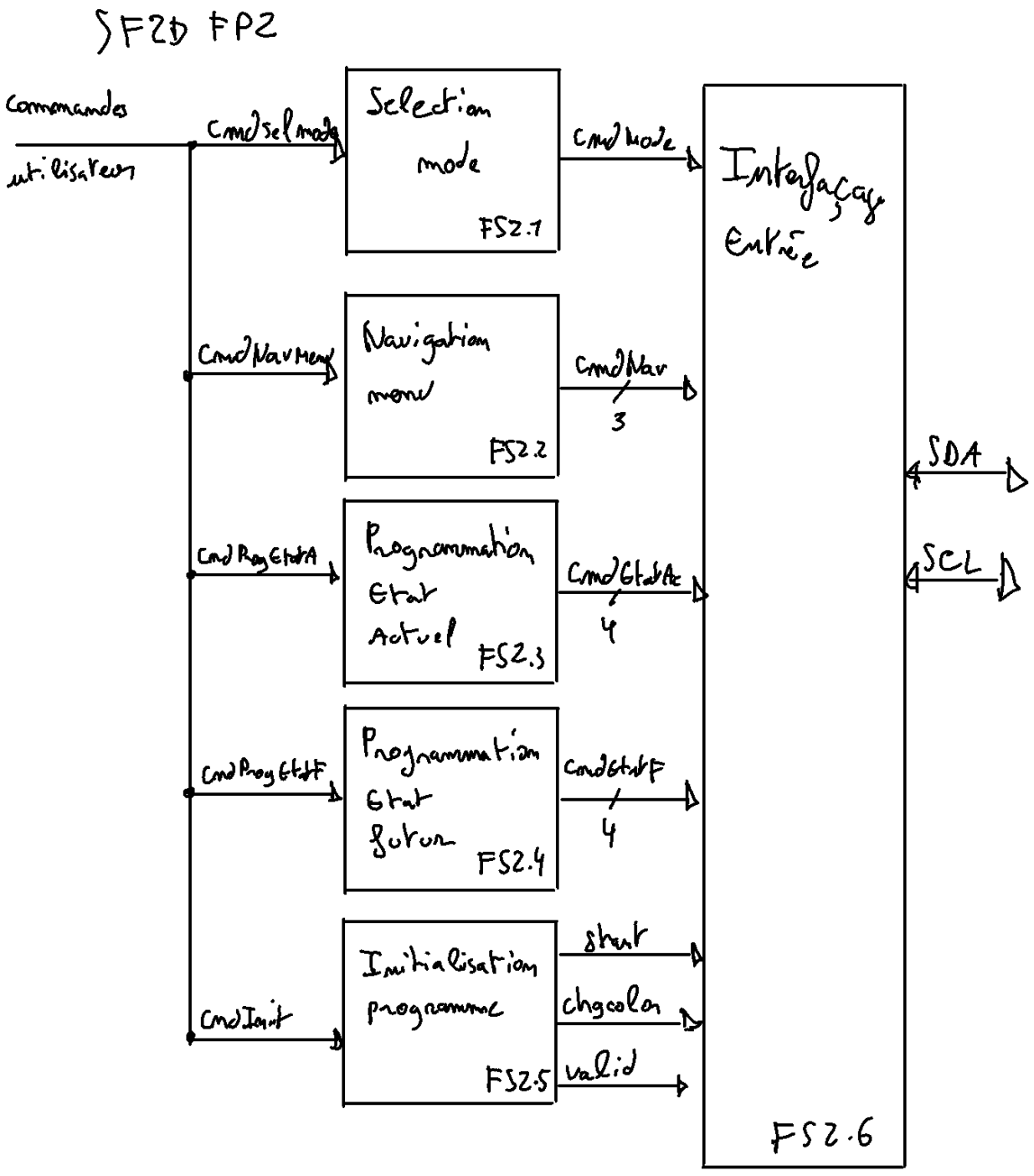
On peut désormais rentrer un peu plus dans le détail avec un SF1D :



On voit qu'on a quatre fonctions principales : une qui s'occupe de la gestion des entrées (FP2), une qui gère les sorties (FP3), une fonction de stockage pour stocker les tables de transitions (FP4), et du traitement numérique qui va piloter tout ça (FP1).
On a aussi deux fonctions annexes : la fonction alimentation qui va se charger de fournir le courant nécessaire pour que la machine puisse fonctionner, et la fonction de programmation in-situ qui va faciliter le chargement du code dans la machine et nous éviter d'avoir à sortir le micro-contrôleur (MCU) à chaque fois.

On voit que les fonctions FP2 et FP3 sont encore floues, on va donc les affiner en faisant des SF2D pour qu'on se rende compte de leur fonctionnement :

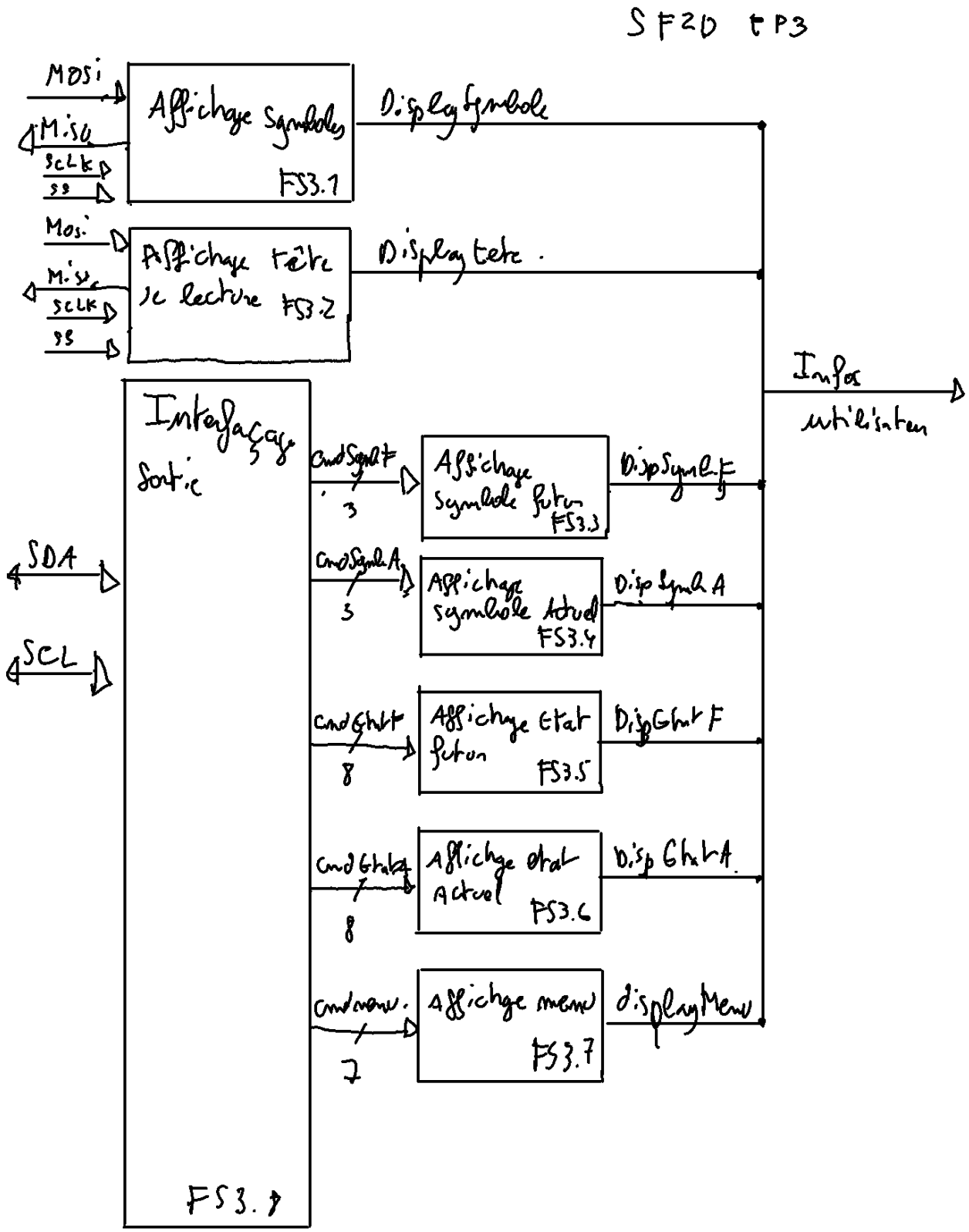
FP2 :



On voit clairement que FP2 est bien plus complexe qu'elle ne le parait, et forme un ensemble

de fonctions plus spécifiques qui vont permettre à l'utilisateur d'interagir physiquement avec la machine, dans le but de la configurer, programmer, etc.

FP3 :



De la même manière que pour FP2, FP3 est en réalité un ensemble de fonctions plus spécifiques qui vont chacune avoir un rôle bien précis dans le retour des informations utilisateur. On a donc une grande variété de fonctions d'affichage qui vont, de la même manière que pour les sous-

fonctions de FP2, être spécifiées ci-dessous.

2.3.3 Choix des technologies

Communication

Bus I²C Le nombre élevé d'entrées-sorties, du aux grand nombre de boutons poussoirs et composants d'affichage, nous a orienté sur l'utilisation d'expandeurs. Nous nous sommes orientés vers des expandeurs I²C, car l'utilisation du bus I²C nous permettait de gérer toutes nos entrées / sorties en utilisant uniquement deux broches sur le MCU (SDL et SCL). Nous avons ainsi essayé d'avoir un maximum de composants en I²C.

SPI Nous nous sommes posés la question de l'utilisation du bus SPI pour notre ruban de LEDs. Nous avions aussi besoin de choisir notre type de communication avec la carte SD, et il se trouve que le bus SPI est très utilisé pour ce type de communication.

2.3.4 Spécifications des fonctions et de leurs signaux de communication

Nous avons donc spécifié ci-dessous, pour chaque fonction, son rôle ainsi que ses signaux d'entrée et/ou de sortie. Nous avons également précisé le rôle ainsi que ce qui fait l'essence même de chaque signal (type, etc.).

Fonction	FP1	Traitements numériques
Description / rôle	Réalise tous les traitements numériques nécessaires au bon fonctionnement du système.	
Signaux d'entrée	➢ MISO	
Signaux de sortie	➢ MOSI, SCLK, SS, SCL	
Signaux E/S	➢ SDA, ISCP	

Fonction	FS2.1	Sélection du mode
Description / rôle	Fonction qui sert à sélectionner le mode d'exécution, entre pas à pas ou continu.	
Signaux d'entrée	➢ CmdSelMode	
Signaux de sortie	➢ CmdMode	
Signaux E/S	➢ NA	

Fonction	FS2.2	Navigation Menu
Description / rôle	Fonction qui sert à naviguer dans le menu.	
Signaux d'entrée	➢ CmdNavMenu	
Signaux de sortie	➢ CmdNav<0:2>	
Signaux E/S	➢ NA	

Fonction	FS2.3	Programmation Etat Actuel

Description / rôle	Fonction qui permet de programmer l'état actuel de la table de transition.
Signaux d'entrée	➢ CmdProgEtatA
Signaux de sortie	➢ CmdEtatA<0:3>
Signaux E/S	➢ NA

Fonction	FS2.4	Programmation Etat Futur
Description / rôle	Fonction qui permet de programmer l'état actuel de la table de transition.	
Signaux d'entrée	➢ CmdProgEtatF	
Signaux de sortie	➢ CmdEtatF<0:3>	
Signaux E/S	➢ NA	

Fonction	FS2.5	Initialisation Programme
Description / rôle	Fonction qui permet d'initialiser les symboles et la position de la tête de lecture.	
Signaux d'entrée	➢ CmdInit	
Signaux de sortie	➢ Start, ChgColor, Valid	
Signaux E/S	➢ NA	

Fonction	FS2.6	Interfaçage Entrée
Description / rôle		
Signaux d'entrée	➢ CmdMode, CmdNav<0:2>, CmdEtatA<0:3>, CmdEtatF<0:3>, Start, ChgColor, Valid	
Signaux de sortie	➢ SCL	
Signaux E/S	➢ SDA	

Fonction	FS3.1	Affichage Symboles

Description / rôle	Réalise l'affichage des symboles (représente le ruban).
Signaux d'entrée	➢ MOSI, SS, SCLK
Signaux de sortie	➢ MISO, DisplaySymboles
Signaux E/S	➢ NA

Fonction	FS3.2	Affichage tête de lecture
Description / rôle	Réalise l'affichage de la tête de lecture.	
Signaux d'entrée	➢ MOSI, SS, CLK	
Signaux de sortie	➢ MISO, DisplayTete	
Signaux E/S	➢ NA	

Fonction	FS3.3	Affichage Symbole Futur
Description / rôle	Affiche le symbole futur (après transition).	
Signaux d'entrée	➢ CmdSymboleF<0:2>	
Signaux de sortie	➢ DisplaySymboleF	
Signaux E/S	➢ NA	

Fonction	FS3.4	Affichage Symbole Actuel
Description / rôle	Affiche le symbole courant (avant transition).	
Signaux d'entrée	➢ CmdSymboleA<0:2>	
Signaux de sortie	➢ DisplaySylboleA	
Signaux E/S	➢ NA	

Fonction	FS3.5	Affichage Etat Futur

Description / rôle	Affichage l'état futur de la tête de lecture (après transition).
Signaux d'entrée	➢ CmdEtatF<0:7>
Signaux de sortie	➢ DisplayEtatF
Signaux E/S	➢ NA

Fonction	FS3.6	Affichage Etat Actuel
Description / rôle	Affiche l'état courant de la tête de lecture (avant transition).	
Signaux d'entrée	➢ CmdEtatA<0:7>	
Signaux de sortie	➢ DisplayEtatA	
Signaux E/S	➢ NA	

Fonction	FS3.7	Affichage Menu
Description / rôle	Affiche le menu.	
Signaux d'entrée	➢ CmdMenu<0:6>	
Signaux de sortie	➢ DisplayMenu	
Signaux E/S	➢ NA	

Fonction	FS3.8	Interfaçage sortie
Description / rôle		
Signaux d'entrée	➢ CmdSymboleF<0:2>, CmdSymboleA<0:2>, CmdEtatF<0:7>, CmdEtatA<0:7>, CmdMenu<0:6>	
Signaux de sortie	➢ SCL	
Signaux E/S	➢ SDA	

Fonction	FP4	Stockage

Description / rôle	Permet le stockage des tables de transitions.
Signaux d'entrée	➢ MOSI, SCLK, SS
Signaux de sortie	➢ MISO
Signaux E/S	➢ NA

Fonction	FA1	Alimentation
Description / rôle	Alimente le système en électricité.	
Signaux d'entrée	➢ Energie	
Signaux de sortie	➢ VCC	
Signaux E/S	➢ NA	

Fonction	FA2	Programmation In Situ
Description / rôle	Permet de charger le programme dans le MCU.	
Signaux d'entrée	➢ NA	
Signaux de sortie	➢ NA	
Signaux E/S	➢ Données USB, ISCP	

Signal	Fonctions concernées	Nature du signal (A/N/GP)	Taille entité	Grandeur et unité (U, I...)	Plage de variation - Niveaux	Excursion en fréquence	Valeur au repos	Contraintes temporelles	Conformité à une norme
	Description								
MISO	FP1, FS3.1, FS3.2, FA2	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	SPI
	Signal généré par l'esclave communiquer avec le maître.								
MOSI	FP1, FS3.1, FS3.2, FA2	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	SPI
	Signal généré par le maître pour communiquer avec l'esclave.								
SCLK	FP1, FS3.1, FS3.2, FA2	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	SPI
	Horloge transmise par le maître.								
SS	FP1,	N	1	U (V)	Niveau haut –	NA	NA	NA	SPI

Signal	Fonctions concernées	Nature du signal (A/N/GP)	Taille entité	Grandeur et unité (U, I...)	Plage de variation - Niveaux	Excursion en fréquence	Valeur au repos	Contraintes temporelles	Conformité à une norme
	Description								
	FS3.1, FS3.2, FA2				niveau bas				
	Sélection de l'esclave par le maître.								
SDA	FP1, FS2.6, FS3.8	N	1	U (V)	Niveau haut – niveau bas	NA	1	NA	I2C
	Signal de transmission des données								
SCL	FP1, FS2.6, FS3.8	N	1	U (V)	Niveau haut – niveau bas	NA	1	NA	I2C
	Signal de transmission de l'horloge								
Commandes utilisateur	FS2.1, FS2.2, FS2.3, FS2.4, FS2.5	GP	NA	U (V)	NA	NA	NA	NA	NA

Signal	Fonctions concernées	Nature du signal (A/N/GP)	Taille entité	Grandeur et unité (U, I…)	Plage de variation - Niveaux	Excursion en fréquence	Valeur au repos	Contraintes temporelles	Conformité à une norme
	Description								
	Action de l'appui sur les boutons par l'utilisateur								
CmdMode	FS2.1	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image du mode (Niveau Bas : mode pas à pas, Niveau haut : mode continu)								
CmdNav	FS2.2	N	3	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image de l'appui sur les boutons (3 signaux car 3 boutons).								
CmdEtatA	FS2.3	N	4	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image de l'appui sur les boutons (4 signaux car 4 boutons).								
CmdEtatF	FS2.4	N	4	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image de l'appui sur les boutons (3 signaux car 3 boutons).								
Start	FS2.5	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image de l'appui sur le bouton de lancement.								

Signal	Fonctions concernées	Nature du signal (A/N/GP)	Taille entité	Grandeur et unité (U, I…)	Plage de variation - Niveaux	Excursion en fréquence	Valeur au repos	Contraintes temporelles	Conformité à une norme
	Description								
ChgColor	FS2.5	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image de l'appui sur le bouton d'initialisation des symboles et de la tête de lecture.								
Valid	FS2.5	N	1	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Image de l'appui sur le bouton de validation.								
Energie	FA1	A	1	U (V)	230V CC	NA	NA	NA	NA
VCC	FA1	A	1	U (V)	5V CC 3.3V CC	NA	NA	NA	NA
ISCP	FP1, FA2	N	1	NA	Niveau haut – niveau bas	NA	0	NA	NA
	Signal de transmission des données pour la programmation in situ								
Infos Utilisateurs	FS3.1, FS3.2, FS3.3,	GP	NA	U (V)	NA	NA	NA	NA	NA

Signal	Fonctions concernées	Nature du signal (A/N/GP)	Taille entité	Grandeur et unité (U, I...)	Plage de variation - Niveaux	Excursion en fréquence	Valeur au repos	Contraintes temporelles	Conformité à une norme
	Description								
	FS3.4, FS3.5, FS3.,6, FS3.7								
	Affichage des différentes fonctions								
CmdSymboleF	FS3.3, FS3.8	N	3	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Signaux de sortie de la fonction FS3.8, qui commandent les entrés de la fonction FS3.3								
CmdSymboleA	FS3.4, FS3.8	N	3	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Signaux de sortie de la fonction FS3.8, qui commandent les entrés de la fonction FS3.4								
CmdEtatF	FS3.5, FS3.8	N	8	U (V)	0 – 3,3V	NA	NA	NA	NA
	Signaux de sortie de la fonction FS3.8, qui commandent les entrés de la fonction FS3.5								
CmdEtatA	FS3.6, FS3.8	N	8	U (V)	0 – 3,3V	NA	NA	NA	NA
	Signaux de sortie de la fonction FS3.8, qui commandent les entrés de la fonction FS3.6								

Signal	Fonctions concernées	Nature du signal (A/N/GP)	Taille entité	Grandeur et unité (U, I...)	Plage de variation - Niveaux	Excursion en fréquence	Valeur au repos	Contraintes temporelles	Conformité à une norme
	Description								
CmdMenu	FS3.7, FS3.8	N	7	U (V)	Niveau haut – niveau bas	NA	NA	NA	NA
	Signaux de sortie de la fonction FS3.8, qui commandent les entrés de la fonction FS3.7								

2.3.5 Choix des composants

Composants

Nous avions donc besoin des composants suivants :

- 2 rubans de LEDs RGB communicant en SPI
- 2 afficheurs 7 segments
- 1 écran LCD communicant en I^2C
- 2 LEDs RBG
- 2 LEDs pour indiquer le mouvement, et 1 LED d'erreur
- 15 boutons poussoirs
- 2 commutateurs à bascule
- 1 lecteur de carte micro-SD communicant en SPI
- 3 expandeurs E/S - I^2C de 16 I/O

Choix du micro contrôleur

Nous avions déjà un PIC24FJ64GA002 car c'est le micro-contrôleur que le groupe précédent utilisait. Ce micro-contrôleur s'alimente en 3,3V et possède deux modules I^2C et deux modules SPI. Comme nous utilisons des expandeurs pour toutes nous E/S, nous pouvons reprendre ce micro-contrôleur pour notre machine de Turing.

Commandes

Initialement, nous avions fait un premier choix de composants qui correspondait à nos spécificités. Toute la commande était prévue chez Farnell, mais nous avons dû nous raviser et mettre au point une autre liste à cause du délai de commande trop important.

Commande initiale Farnell

- rubans de LEDs adressables SK9822
- afficheurs 7 segments HDSM-283B
- LEDs RGB L-59EYC
- expandeurs E/S - I^2C MCP23017
- écran LCD I2C MC21605C6W-BNMLWI-V2
- boutons poussoirs ESE20C321
- commutateurs 2AS2T2A1M7RE

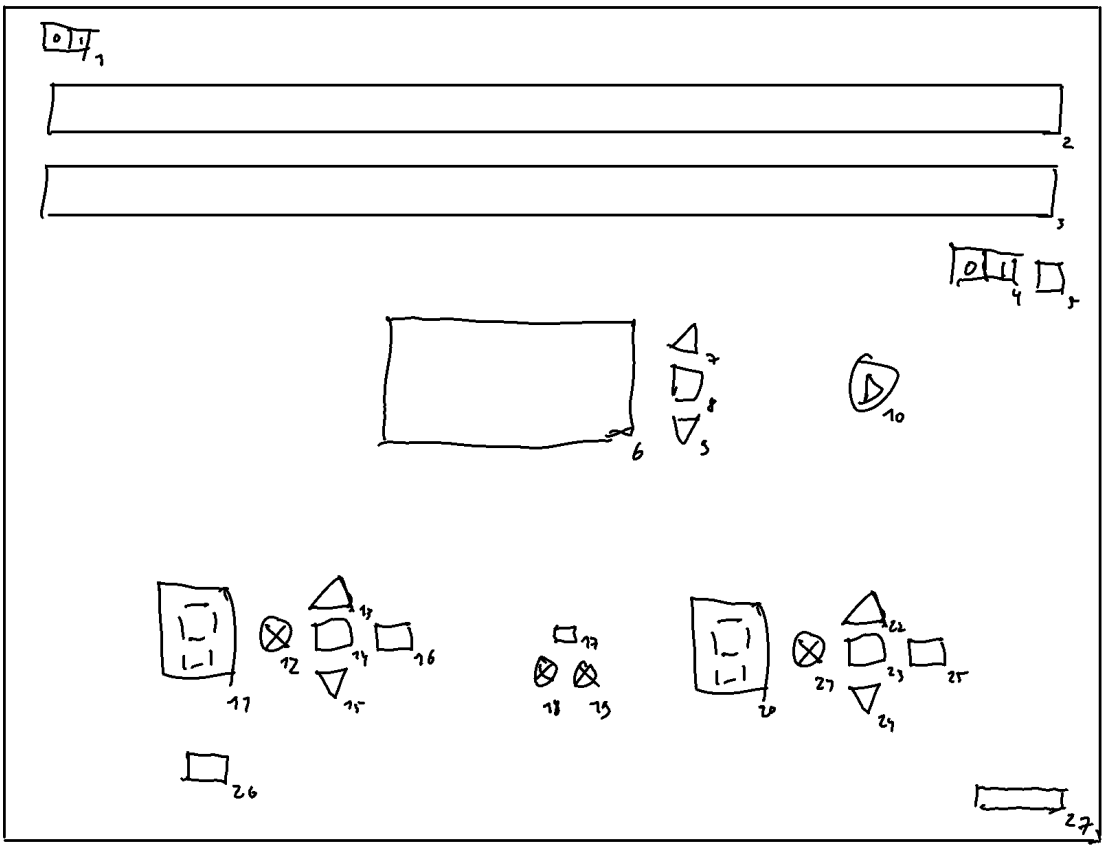
Commande réelle (RS et Amazon)

- rubans de LEDs adressables SK9822
- expandeurs E/S - I^2C MCP23017
- écran LCD I^2C NHD-C0220BiZ-FSW-FBW-3V3M

- LEDs RGB L-154A4SURKQBDZGW

2.3.6 Schéma représentatif

Le schéma ci-dessous nous donne une idée de ce à quoi pourrait ressembler notre machine de Turing une fois tous les composants montés :

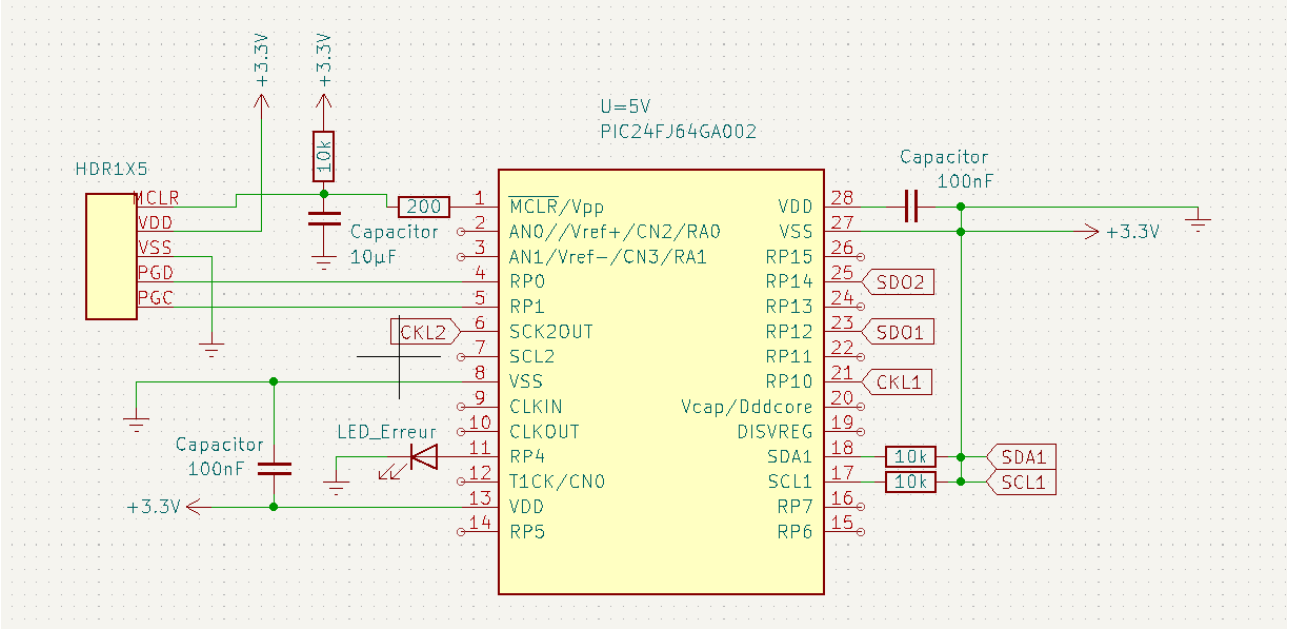


Nous avons essayé de respecter les connexions minimales recommandées de Microchip, trouvées dans la documentation de notre micro-contrôleur à la page 17 (voir Annexe 3.1). Légende :

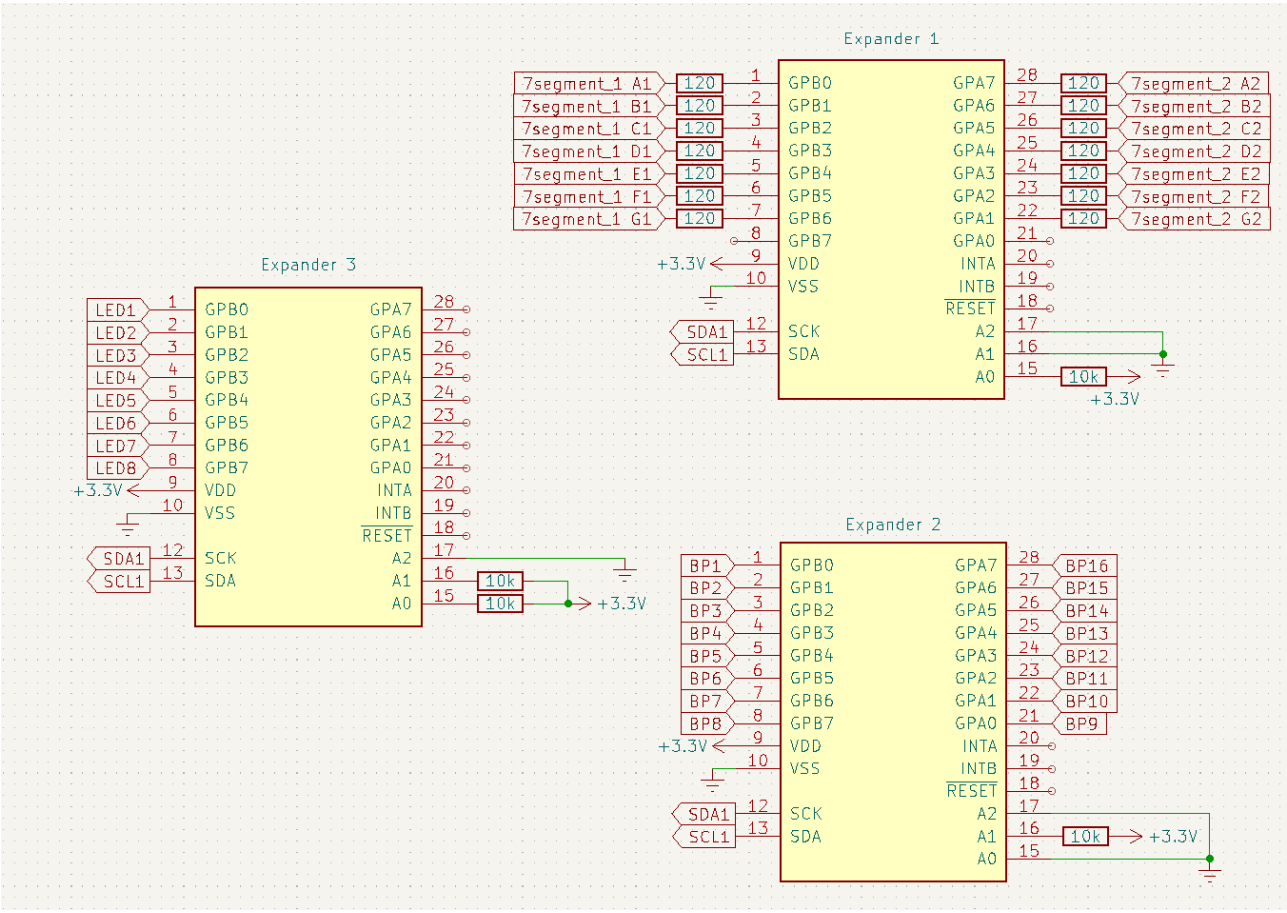
- 1 : Interrupteur d'alimentation
- 2 : Ruban de LEDs pour l'affichage des symboles
- 3 : Ruban de LEDs pour l'affichage de la tête de lecture
- 4 : Commutateur pour choisir entre le mode pas à pas et automatique
- 5 : Bouton de réinitialisation lors de l'initialisation manuelle des rubans de LEDs
- 6 : Ecran LCD pour l'affichage du menu
- 7, 8, 9 : Boutons de navigation dans le menu (monter, sélectionner, descendre)
- 10 : Bouton de lancement du programme. Sert également à passer à l'étape suivante en mode pas à pas
- 11 | 20 : Afficheurs 7 segments pour l'affichage de l'état courant | de l'état suivant
- 12 | 21 : LEDs RGB pour l'affichage du symbole courant | du symbole suivant
- 13, 14, 15, 16 | 22, 23, 24, 25 : Boutons de décrémentation, validation, incrémentation, ré-initialisation de l'état et du symbole courant | de l'état et du symbole futur
- 17 : Bouton de changement du déplacement pour l'état suivant
- 18, 19 : LED représentant le déplacement pour l'état suivant (gauche si LED 18 allumée, droite si LED 19 allumée)
- 26 : Bouton de sélection d'un état d'acceptation
- 27 : Lecteur de carte micro-SD

2.3.7 Schémas structurels

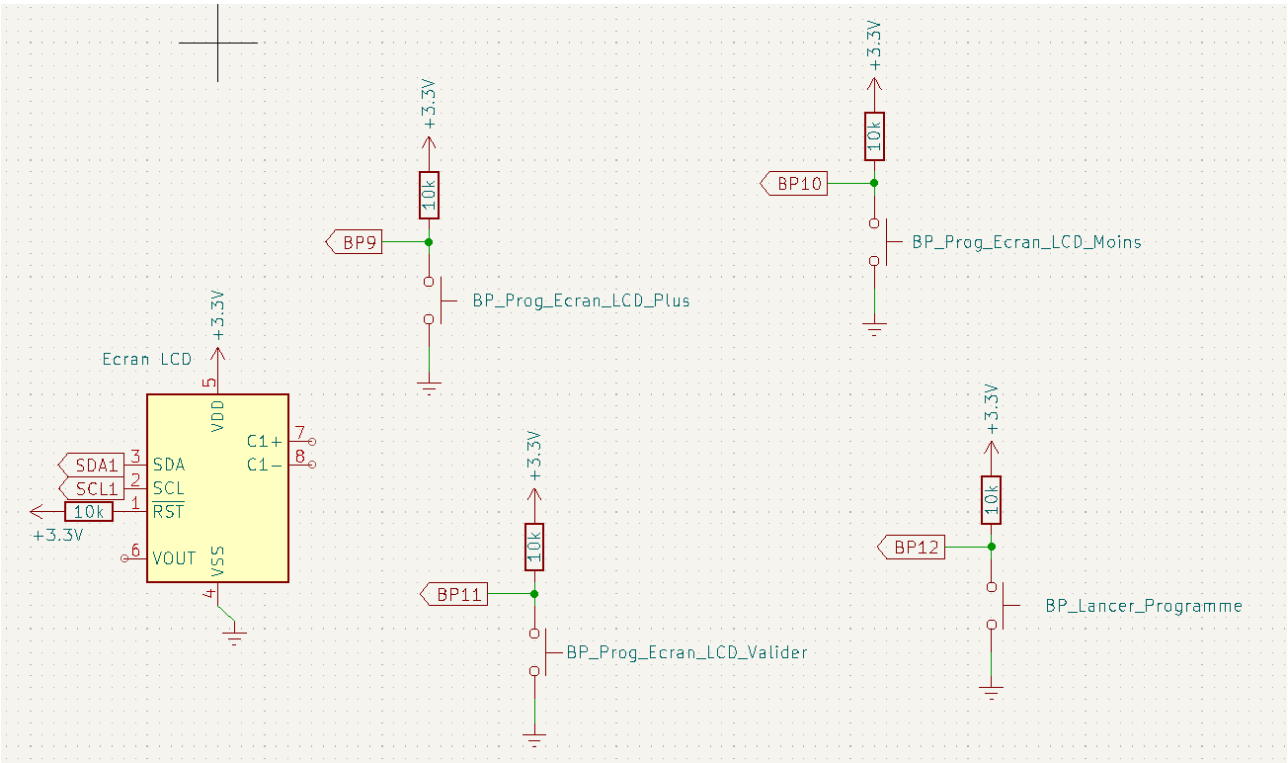
MCU



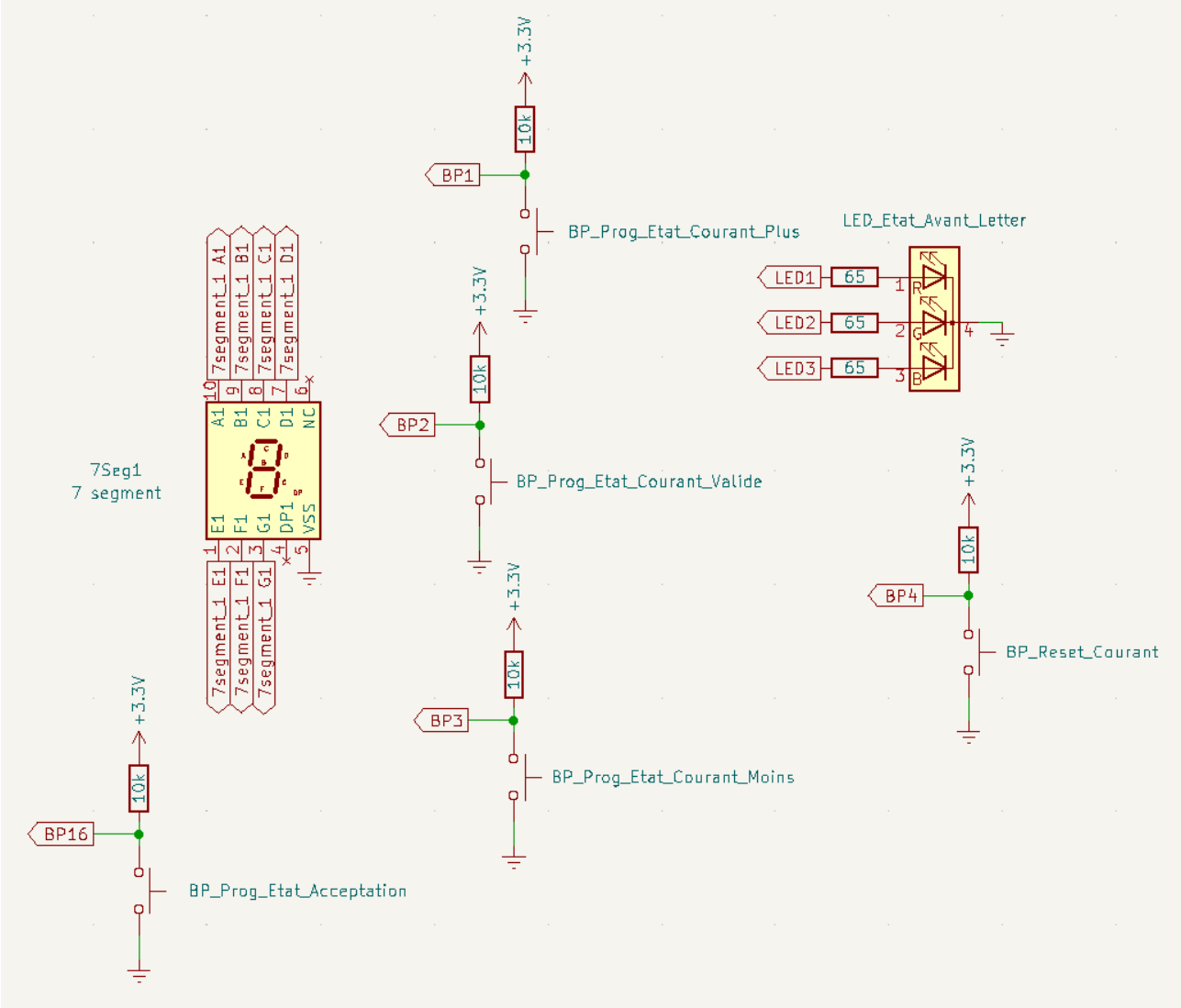
Expandeurs E/S - I²C



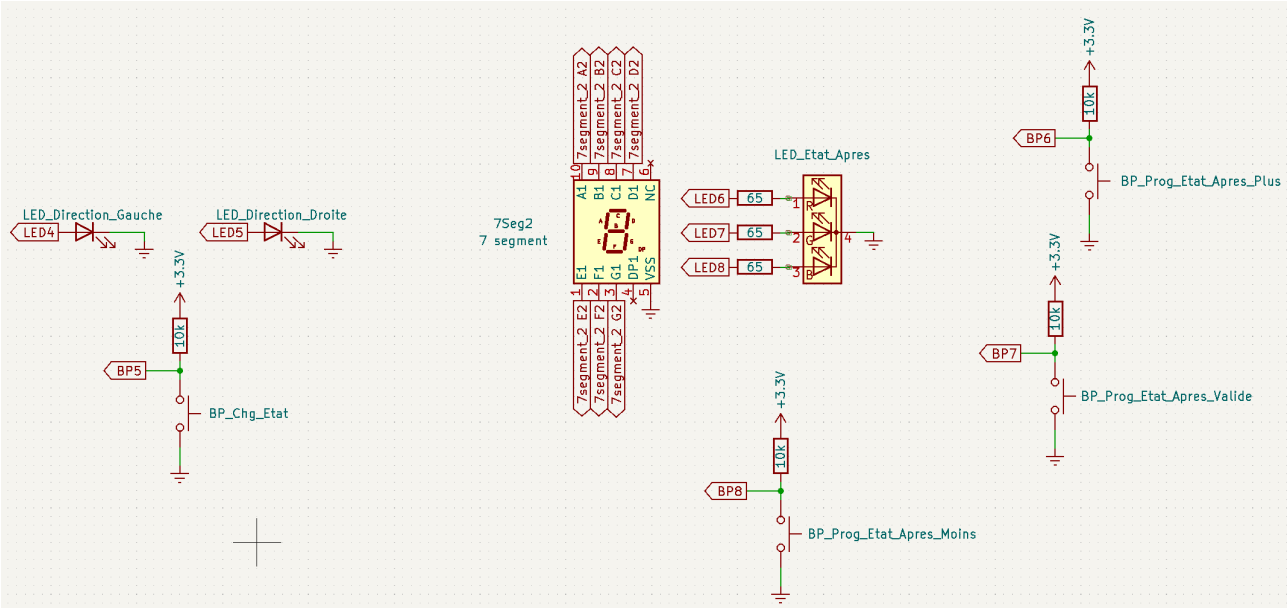
Ecran LCD


+3.3V
10k
BP9
BP_Prog_Ecran_LCD_Plus
+3.3V
10k
BP10
BP_Prog_Ecran_LCD_Moins
Ecran LCD
+3.3V
5
VDD
C1+
C1−
7
8
SDA1
SCL1
10k
+3.3V
3
2
1
SDA
SCL
RST
6
VOUT
VSS
4
+3.3V
10k
BP11
BP_Prog_Ecran_LCD_Valider
+3.3V
10k
BP12
BP_Lancer_Programme

Affichage et programmation de l'état courant



Affichage et programmation de l'état futur



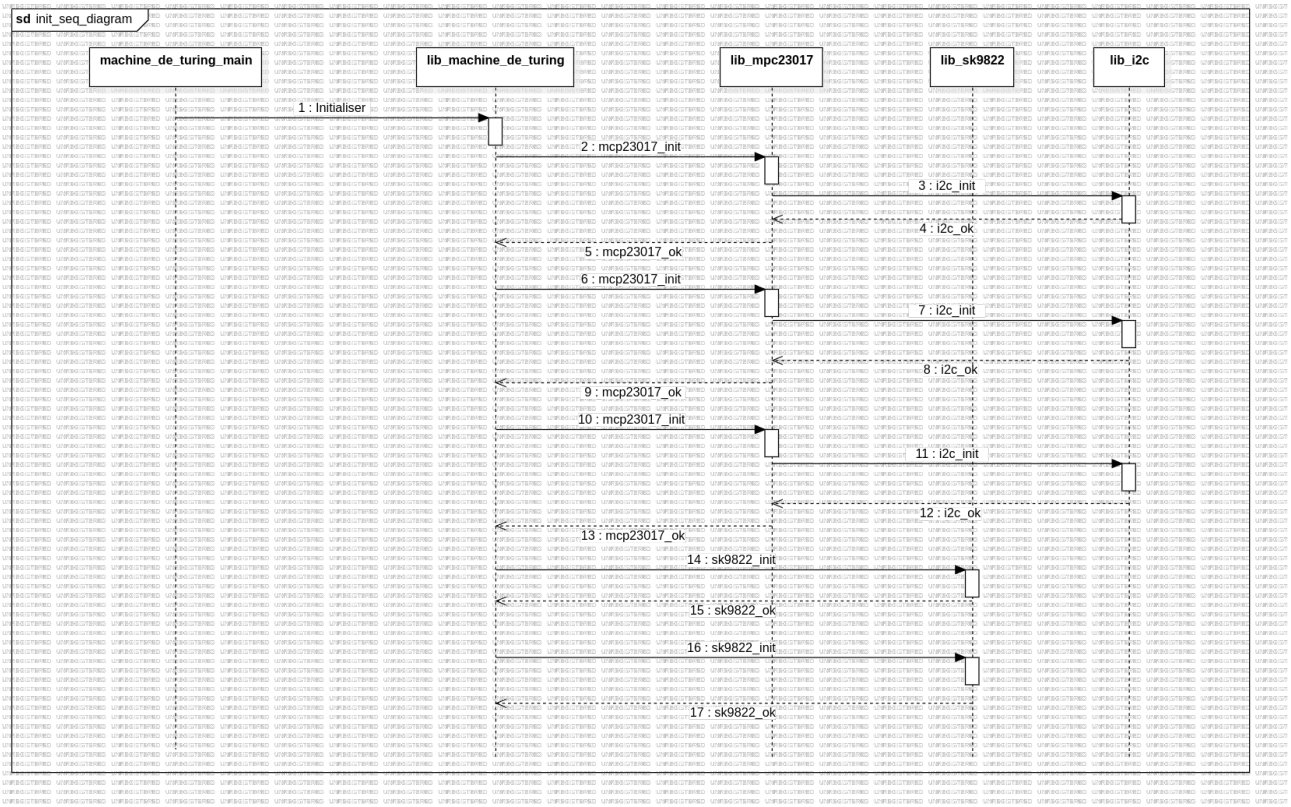
2.3.8 Tests

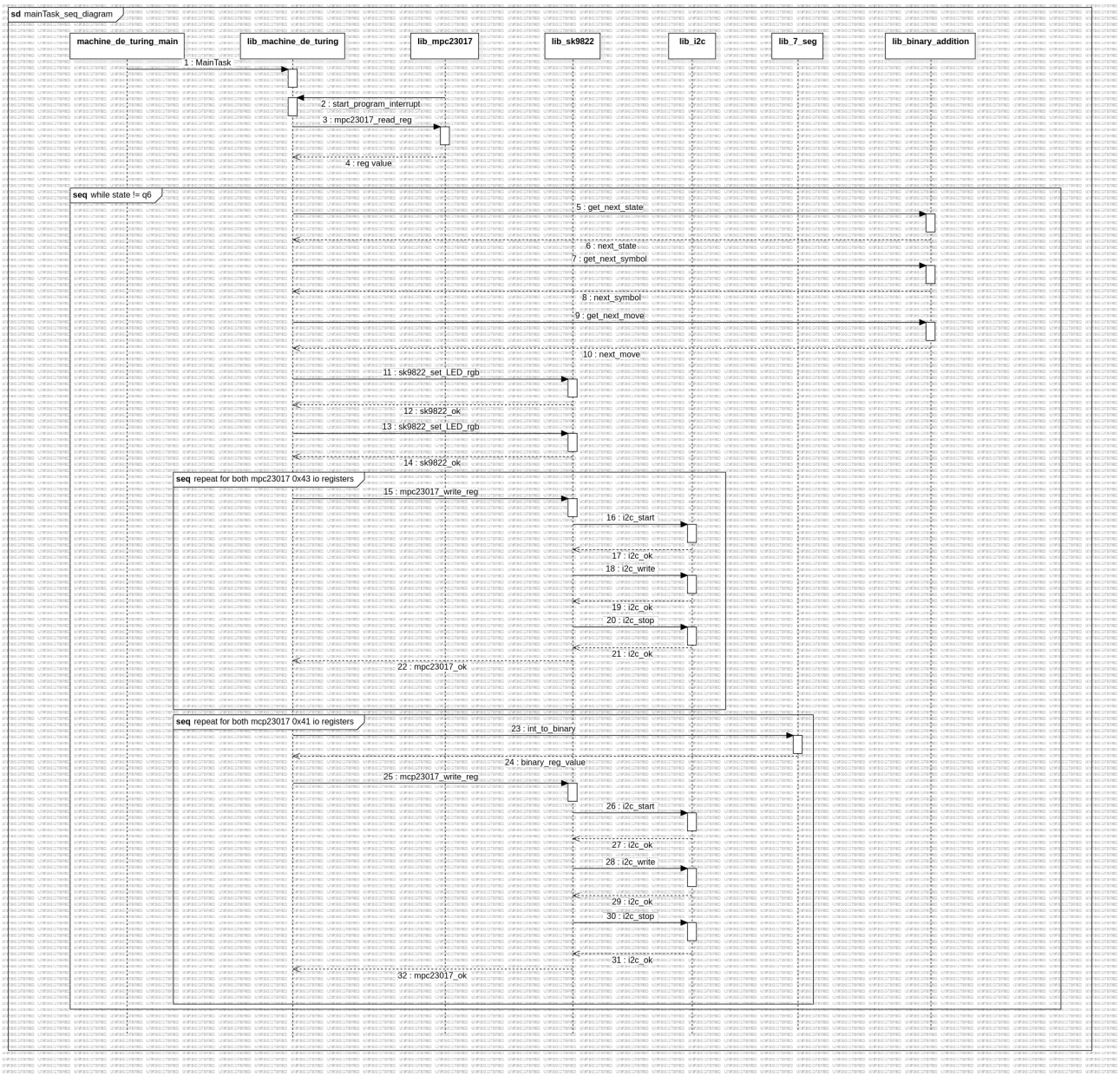
2.3.9 Conception Logicielle

Cette section ne traitera que la partie 1 du cahier des charges, les parties 2 et 3 ne pouvant être réalisées par manque de temps.

2.3.10 Diagramme de Séquences

Les deux diagrammes de séquence ci-dessous montrent respectivement la phase d'initialisation et la phase de déroulement du programme au niveau du micro-contrôleur (on ne prend pas en compte les actions de l'utilisateur).
Ce sont des diagrammes "best-case", c'est à dire qu'on ne prend pas en compte les erreurs qui pourraient survenir, et on déroule le fonctionnement du programme en supposant que tout se passe bien. Cela permet principalement de ne pas faire trop de diagrammes, ni de diagrammes trop longs.





2.3.11 Algorithme

```
    LIB_MACHINE_DE_TURING
    VARIABLES
      mcp23017_desc_t mpc23017_7_seg
      mcp23017_desc_t mcp23017_bp
      mcp23017_desc_t mcp23017_general
      i2c_desc_t I2CModule

    ENUMERATIONS
      mcp23017_err_t
      MCP23017_OK
      MCP23017_ERROR

    mcp23017_i2c_init_type_t
      INIT_WITH_I2C1
      INIT_WITH_I2C2
      INIT_ALREADY_DONE
```

```
    working_mode_t
      MANUAL_MODE
      AUTOMATIC_MODE

    STRUCTURES
    mcp23017_desc_t
      i2c_desc_t  *pi2c
      uint8_t     i2c_Address

    mcp23017_config_t
      i2c_desc_t  *pi2c;
      mcp23017_i2c_init_type_t    initType;
      uint8_t i2c_Address;


    machine_de_turing_desc_t
      uint8_t state
      uint8_t symbol
      uint8_t next_state
      uint8_t next_symbol
      uint8_t next_move

    CONSTANTES
      MCP23017_7_SEG_ADDRESS      0100001
      MCP23017_BP_ADDRESS         0100010
      MCP23017_GENERAL_ADDRESS    0100011

      MCP23017_IOCONA_ADDRESS     0x0A
      MCP23017_IOA_ADDRESS        0x12
      MCP23017_IOB_ADDRESS        0x13

    DEBUT
      PROCEDURE Initialiser()
      VARIABLES
        mcp23017_err_t   Res;
        mcp23017_config_t    mcpCfg;
      DEBUT
        Configuration du port B en sortie

        // MPC23017 7 segs
        mcpCfg.pi2c = &I2CModule;
        Initialisation avec I2C_1
        Adresse = MCP23017_7_SEG_ADRESS;

        Res = mcp23017_init(@mpc23017_7_seg, @mcpCfg)
        SI Res != MCP23017_OK ALORS
          error_handler()
        FIN SI

        // MPC23017 BP
        mcpCfg.pi2c = &I2CModule;
        Initialisation avec I2C_1
        Adresse = MCP23017_BP_ADDRESS;

        Res = mcp23017_init(@mpc23017_bp, @mcpCfg)
        SI Res != MCP23017_OK ALORS
          error_handler()
        FIN SI

        // MPC23017 GENERAL
```

```
78        mcpCfg.pi2c = &I2CModule;
79        Initialisation avec I2C_1
80        Adresse = MCP23017_GENERAL_ADDRESS;
81
82        Res = mcp23017_init(@mpc23017_bp, @mcpCfg)
83        SI Res != MCP23017_OK ALORS
84          error_handler()
85        FIN SI
86
87      FIN Initialiser()
88
89      PROCEDURE MainTask()
90      VARIABLES
91        uint8_t working_mode
92        uint8_t next_step
93        led_color_t rw_pointer_color
94      DEBUT
95        Res = mcp23017_read_reg(@mcp23017_bp, MCP23017_IOA_ADDRESS,
    @working_mode)
96        SI Res != MCP23017_OK ALORS
97          working_mode = MANUAL_MODE
98        FIN SI
99
100       TANT QUE mtu->state != ACCEPT_STATE FAIRE
101         get_next_state(mtu->state, mtu->symbol, mtu->next_state)
102         get_next_symbol(mtu->state, mtu->symbol, mtu->next_symbol)
103         get_next_move(mtu->state, mtu->symbol, mtu->next_move)
104         set_next_move(mtu->position, mtu->next_move)
105
106         sk9822_set_LED_rgb(sk9822_1, mtu->position, mtu->symbol)
107         sk9822_set_LED_rgb(sk9822_1, mtu->position, rw_pointer_color)
108
109         RegValue = int_to_bin(mtu->state)
110         Res = mcp23017_write_reg(@mcp23017_7_seg, MCP23017_IOA_ADDRESS,
    RegValue)
111         SI Res != MCP23017_OK ALORS
112           error_handler()
113         FIN SI
114
115         RegValue = int_to_bin(mtu->next_state)
116         Res = mcp23017_write_reg(@mcp23017_7_seg, MCP23017_IOB_ADDRESS,
    RegValue)
117         SI Res != MCP23017_OK ALORS
118           error_handler()
119         FIN SI
120
121         RegValue = generate_general_reg(mtu->symbol, mtu->next_symbol, mtu
    ->next_move)
122         Res = mcp23017_write_reg(@mcp23017_general, MCP23017_IOA_ADDRESS,
    RegValue)
123         SI Res != MCP23017_OK ALORS
124           error_handler()
125         FIN SI
126
127         mtu->state = mtu->next_state
128         mtu->symbol = mtu->next_symbol
129
130         SI working_mode = MANUAL_MODE ALORS
131           TANT QUE next_step = 0 FAIRE
132           FIN TANT QUE
```

```
        SINON
          attendre(2000)
        FIN SI
      FIN TANT QUE
  FIN MainTask()

  PROCEDURE error_handler()
    allumer_led()
    TANT QUE 1 FAIRE
  FIN TANT QUE
  FIN error_handler

```

2.3.12 Code

Ci-dessous le code correspondant à l'algorithme précédent :

```c
/**
* @file    machine_de _turing_main.c
* @author   Romain BROUARD
* @date 2024/05
* @brief  main programm of the "machine de turing" project
*/

#include "lib_machine_de_turing.h"
#include "lib_sept_seg.h"

#ifndef FCY
#define FCY 400000UL
#endif

mcp23017_desc_t mcp23017_7_seg;
mcp23017_desc_t mcp23017_bp;
mcp23017_desc_t mcp23017_general;
i2c_desc_t I2CModule;

void Initialiser() {
  mcp23017_err_t   Res;
  mcp23017_config_t    mcpCfg;

  TRISB = 0x0000;
  LATB = 0;

  mcpCfg.pi2c = &I2CModule;
  mcpCfg.initType = INIT_WITH_I2C1;

  // MCP23017 7 segs
  mcpCfg.i2c_Address = MCP23017_7_SEG;
  Res = mcp23017_init(&mcp23017_7_seg,&mcpCfg);
  if (Res != MCP23017_OK) error_handler();

  // MCP23017 BP
  mcpCfg.i2c_Address = MCP23017_BP;
  Res = mcp23017_init(&mcp23017_bp,&mcpCfg);
  if (Res != MCP23017_OK) error_handler();

  // MCP23017 GENERAL
  mcpCfg.i2c_Address = MCP23017_GENERAL;
  Res = mcp23017_init(&mcp23017_general,&mcpCfg);
```

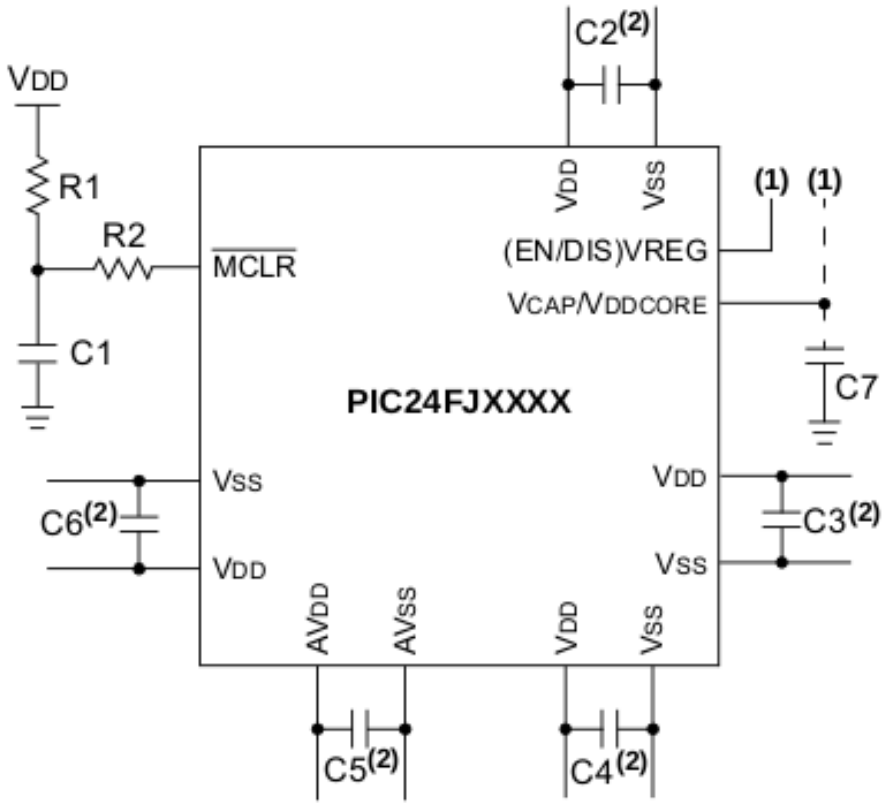
```
43    if (Res != MCP23017_OK) error_handler();
44
45
46
47  }
48
49  void MainTask() {
50    uint8_t RegAddr = 0x09;
51    uint8_t RegValue = 0b10100000;
52    uint8_t RegValue2;
53    if(mcp23017_write_reg(&mcp23017_7_seg, 0x0A, RegValue) == MCP23017_ERROR
     ) {
54      error_handler();
55    }
56    RegValue = 0b00000001;
57
58    if(mcp23017_write_reg(&mcp23017_7_seg, RegAddr, 0b00000001) ==
     MCP23017_ERROR) {
59      error_handler();
60      while(1);
61    }
62    if(mcp230173_read_reg(&mcp23017_7_seg, RegAddr, &RegValue2) ==
     MCP23017_ERROR) {
63      error_handler();
64    }
65    LATBbits.LATB15 = 0;
66    if(RegValue2 == 0b00000001) {
67      valid();
68      while(1);
69    }
70    else {
71      error_handler();
72      while(1);
73    }
74  }
75
76  void error_handler(void){
77    LATBbits.LATB15 = 1;
78    __delay_ms(500);
79  }
80
81  void valid(void) {
82    LATBbits.LATB14 = 1;
83  }
84
```

Synthèse

Nous sommes conscient d'avoir vu grand, trop grand. Aujourd'hui, nous pensons avoir une partie d'analyse plutôt solide, nous avons également du code, et nous pensons qu'avec du temps en plus, nous aurions pu arriver à une maquette, au moins de la partie 1. Nous avons manqué de chance vis-à-vis des composants, notamment du ruban de LEDs puisque celui qu'on a reçu ne marche pas.
Nous sommes fiers du travail réalisé, l'analyse est là, les composants sont là, le code est partiellement là, et nous avons pu véritablement mettre en pratique tout ce que nous avons appris en AMIIC et Programmation MCU lors de cette année.

Annexes

4.1 Connexions minimales recommandées



Key (all values are recommendations):

C1 through C6: 0.1 µF, 20V ceramic

C7: 10 µF, 6.3V or greater, tantalum or ceramic

R1: 10 kΩ

R2: 100Ω to 470Ω

4.2 Code

4.3 Sources