Rapport de Travaux Pratiques

Systèmes à Evènements Discrets (SED)

Alexis GIBERT
Nour GHILOUFI
1A SRI UPSSITECH

Table des matières

Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

Commande d'une maquette d'ascenseur, modélisation par machines à états finis, mise en œuvre en VHDL sur FPGA

1 But de la manipulation

Il s'agit de commander un monte-charge à quatre étages disposant des entrées et des sorties illustrés Figure 1 et répertoriés dans le Tableau 1. Comme l'illustre la Figure 1, le système de commande est connecté à deux afficheurs sept segments pouvant servir à afficher des informations au cours du fonctionnement (N° de l'état actif, etc.).

Figure 1 - Simulateur du monte-charge avec E/S (à gauche) et maquette physique réelle avec pinout du MAX7000S fourni (à droite)

	Simulateur	**Maquette physique**
Actionneurs de déplacement de la cabine	M, D	$montee, descend$
Capteurs d'étage	$et1, et2, et3, et4$	$ET[3..0]$
Poussoirs d'appel interne de la cabine	$ap1, ap2, ap3, ap4$	$AP[3..0]$
Contact d'ouverture des portes	po	$PortOuverte$
Poussoirs d'arrêt d'urgence		/
Poussoir palier	$d4, d3, d2, m3, m2, m1$	$P4d, P3d, P2d, P3m, P2m, P1m$
Afficheur 7 segment	/	$dizaine[6..0], unite[6..0]$
Horloge	/	$Horloge$
Poussoirs	$e1, e2, e3, e4$	$E[4..0]$

Tableau 1 - Correspondance des signaux simulés et physique fournis

2 Travail à réaliser

Le but : effectuer une mise en œuvre progressive d'un système de commande comportant, outre la commande des déplacements de la cabine, la mémorisation des appels palier et un système d'arrêt d'urgence.

Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

2.1 Commande de la cabine avec appels internes uniquement

2.1.1 Etapes de la commande à réaliser

1. Départ de l'étage 1
2. Appuis interne pour l'étage x souhaité
3. Déplacement de l'ascenseur vers le palier x désiré
4. Ouverture des portes
5. Fermeture des portes
6. Retour à l'étage 1

De plus, afin de pouvoir surveiller le bon fonctionnement du système de commande, on se servira des afficheurs 7 segments de la carte de contrôle pour afficher le numéro des états.

2.1.2 Entrées et sorties du système considéré

Pour cette commande nous aurons besoin des capteurs et actionneurs répertoriés dans le Tableau 2.

	Simulateur	Maquette physique
Actionneurs de déplacement de la cabine	M, D	$montee, descend$
Capteurs d'étage	$et1, et2, et3, et4$	$ET[3..0]$
Poussoirs d'appel interne de la cabine	$ap1, ap2, ap3, ap4$	$AP[3..0]$
Contact d'ouverture des portes	po	$PortOuverte$
Afficheur 7 segment	/	$dizaine[6..0], unite[6..0]$
Horloge	/	$Horloge$
Poussoir d'initialisation	/	$E(0)$

Tableau 2 - Entrées/Sorties nécessaires au fonctionnement de l'ascenceur avec les appels internes

Note : Un signal externe d'initialisation devra être ajouté pour affecter l'état initial de la/des machine(s) à états. Celui-ci sera câblé au poussoir $E(0)$ de la maquette (voir Figure 9 - Brochage réalisé sur Quartus II2.2.5).

A partir de ces signaux externes (les entrées et sorties du système) nous pouvons alors construire le diagramme FMG illustré Figure 2 ou l'on veillera à rajouter les signaux internes :

- $ap1Mem$, $ap2Mem$, $ap3Mem$, $ap4Mem$ pour garder en mémoire l'appuis du poussoir correspondant,
- $finApi$ pour mettre fin à la sauvegarde de n'importe quel bouton poussoir,
- $EtatPresent$ l'état présent de la machine à état,
- $EtatSuivant$ l'état suivant de la machine à état.

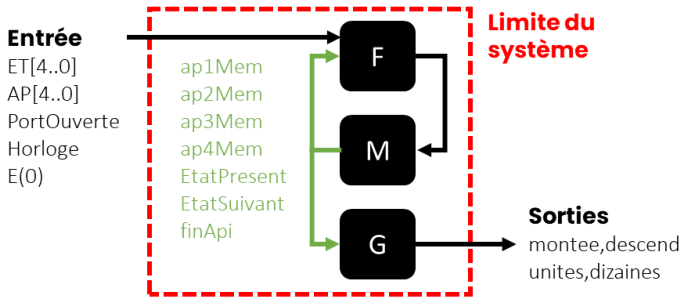


Figure 2 - Schéma bloc FMG basé sur les E/S du modèle physique pour le fonctionnement de l'ascenceur avec les appels internes

On peut alors décrire le schéma bloc FMG sous la forme d'une entité *cabine* en langage VHDL de la forme suivante :

```
ENTITY cabine IS
PORT(
                E                       : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                ET                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                AP                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                PortOuverte  : IN STD_LOGIC;
                Horloge                 : IN STD_LOGIC;
                montee,descend          : OUT STD_LOGIC;
                unites,dizaines         : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END cabine;
```

2.1.3 Modélisation et simulation de la commande

2.1.3.1 Une première modélisation autonome

Pour coller au mieux au modèle physique nous avons souhaité déporter les commandes d'automatisation de l'ouverture et la fermeture des portes dans des machines à état séparés. Ces MAE sont commandées par des signaux internes ($demandeOuverturePorteEi$ et $demandeFermeturePorteEi$) mais ne seront pas définis lors de la description du modèle physique en VHDL puisque ses actions seront réalisées manuellement.

Notons que pour la mémorisation de l'appel nous avons aussi souhaité créer une machine à état supplémentaire.

Enfin, notons que comme le simulateur n'a pas la même structure fonctionnelle que la maquette physique, il n'a donc pas le comportement souhaité lorsque on remplace les conditions fi (avec i, le numéro de l'étage) par $!oi$. Néanmoins, comme le capteur fi n'existe pas sur la maquette physique et que nous avons correctement séparé les différentes MAE, il suffira de remplacer (lors de la description du composant en VHDL) toutes les conditions $finApi$ par leurs $!oi$ (avec i, le numéro de l'étage) respectifs dans la MAE de gestion de l'ascenseur.

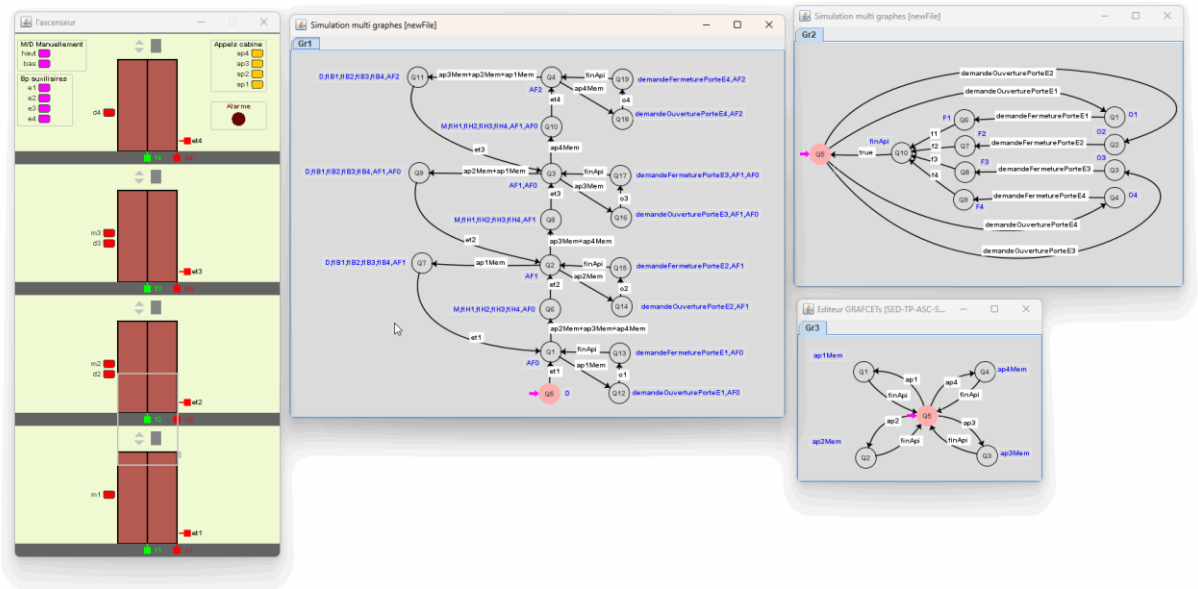


Figure 3 - Simulation du fonctionnement de l'ascenseur avec les appels internes (1ère version)

Sur la simulation ci-dessus nous avons pris le temps d'afficher les flèches et les numéros des étages à l'aide des actionneurs : $flB1, flB2, flB3, flB4, AF2, AF1$ et $AF0$ mais comme ils ne sont pas présent sur la maquette physique, il ne sera plus utile de les actionner, lors de la description VHDL.

2.1.4 Une modélisation simplifiée

Bien que ce modèle de simulation soit performant et automatisé, la MAE peut être largement simplifié si nous ne prenons pas en compte la fermeture des portes automatisée.

Effectivement, nous nous sommes rendus compte qu'un fonctionnement manuel des portes est possible (un clic sur une porte fermée provoque son ouverture et un clic sur la porte ouverte provoque sa fermeture). En d'autres termes il est possible de se limiter à deux MAE en s'approchant au maximum du modèle physique original.

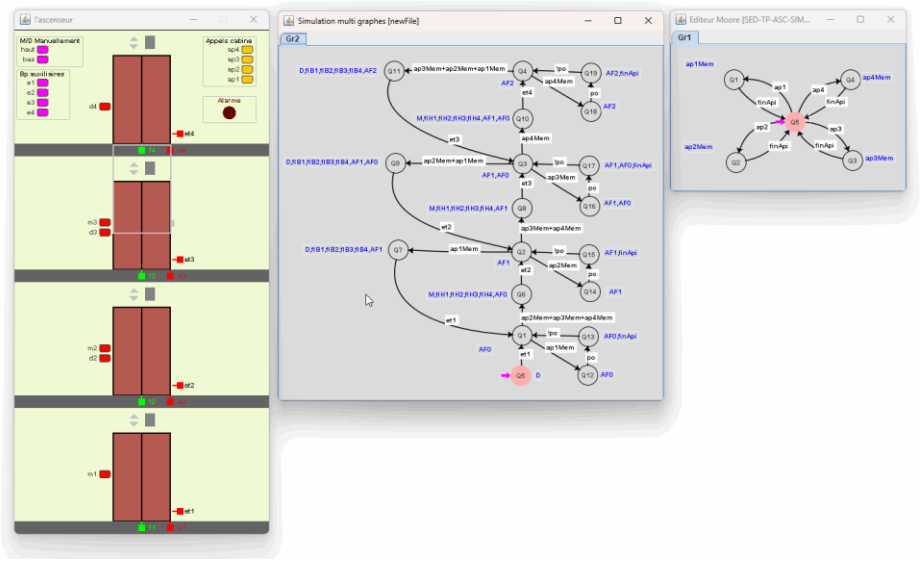


Figure 4 - Simulation du fonctionnement de l'ascenseur avec les appels internes (2ème version)

2.1.5 Description de la commande (VHDL)

Bien que la mise en œuvre de cette commande se fera plus tard (après l'intégration de la prise en compte des arrêts d'urgence), on donnera le nom $cabine$ à son entité et le nom $cabine.vhd$ à ce fichier. De plus, on veillera à bien afficher le numéro de l'état présent sur les afficheurs 7 segments pour faciliter le débogage.

Pour plus d'information sur la description du composant en VHDL, voir Annexe 1 : cabine.vhd.

La maquette réagis correctement à la commande décrite en langage VHDL.

2.1.6 Validation sur support physique

La maquette réagis correctement à la commande décrite en langage VHDL.

2.2 Prise en compte des arrêts d'urgence – Automate de sécurité

Le système d'arrêt d'urgence est sensible à deux signaux $Arret$ et $reprise$. L'appui sur $Arret$ doit figer le mouvement de la cabine en modifiant simplement les valeurs des sorties de la MAE gérant son déplacement. Le retour au mode automatique n'est alors possible que lorsque l'opérateur appuie sur reprise.

2.2.1 Modèle FMG

Parmi les poussoirs disponibles sur la carte ALTERA nous avons décidé que la fonctionnalité d'arrêt d'urgence serait sensible aux poussoirs $E(1)$ (Arrêt) et $E(2)$ (Reprise). Ainsi pour cette commande on aura besoin des capteurs et actionneurs répertoriés dans le Tableau 3.

	Simulateur	Maquette physique
Actionneurs de déplacement de la cabine	M, D	$montee, descend$
Capteurs d'étage	$et1, et2, et3, et4$	$ET[3..0]$
Poussoirs d'appel interne de la cabine	$ap1, ap2, ap3, ap4$	$AP[3..0]$
Contact d'ouverture des portes	po	$PortOuverte$
Afficheur 7 segment	$(non\ défini)$	$dizaine[6..0], unite[6..0]$
Horloge	$(non\ défini)$	$Horloge$
Poussoir d'initialisation, d'arrêt et de reprise	$(non\ défini), e1, e2$	$E(0), E(1), E(2)$

Tableau 3 - E/S nécessaires au fonctionnement de l'ascenceur avec prise en compte des arrêts d'urgence

A partir de ces signaux externes (les entrées et sorties du système) nous pouvons alors construire le diagramme FMG illustré Figure 5 où l'on veillera à rajouter respectivement les signaux externes $E(1)$ et $E(2)$ pour l'arrêt et la reprise de la commande du système ainsi qu'un signal interne s_Arret permettant de mémoriser l'arrêt du poussoir.

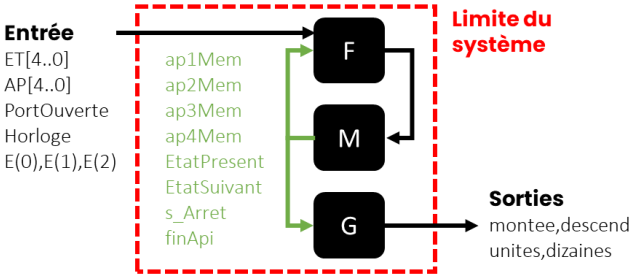


Figure 5 - Schéma bloc FMG basé sur les E/S du modèle physique pour le fonctionnement de l'ascenseur avec les appels internes et la prise en compte des arrêts d'urgence

On peut alors décrire le schéma bloc FMG sous la même forme que la première entité VHDL, soit :

```
ENTITY securite IS
PORT(
                E                       : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                ET                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                AP                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                PortOuverte  : IN STD_LOGIC;
                Horloge                 : IN STD_LOGIC;
                montee,descend          : OUT STD_LOGIC;
                unites,dizaines         : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END securite;
```

2.2.2 Modélisation et simulation de la commande

On se rend vite compte qu'il y a très peu de changement à réaliser. Il suffit de rajouter une MAE comportant 2 états permettant la mémorisation du poussoir $e1$ et de rajouter des conditions aux sorties de déplacement dans la MAE générale (sorties conditionnelles).

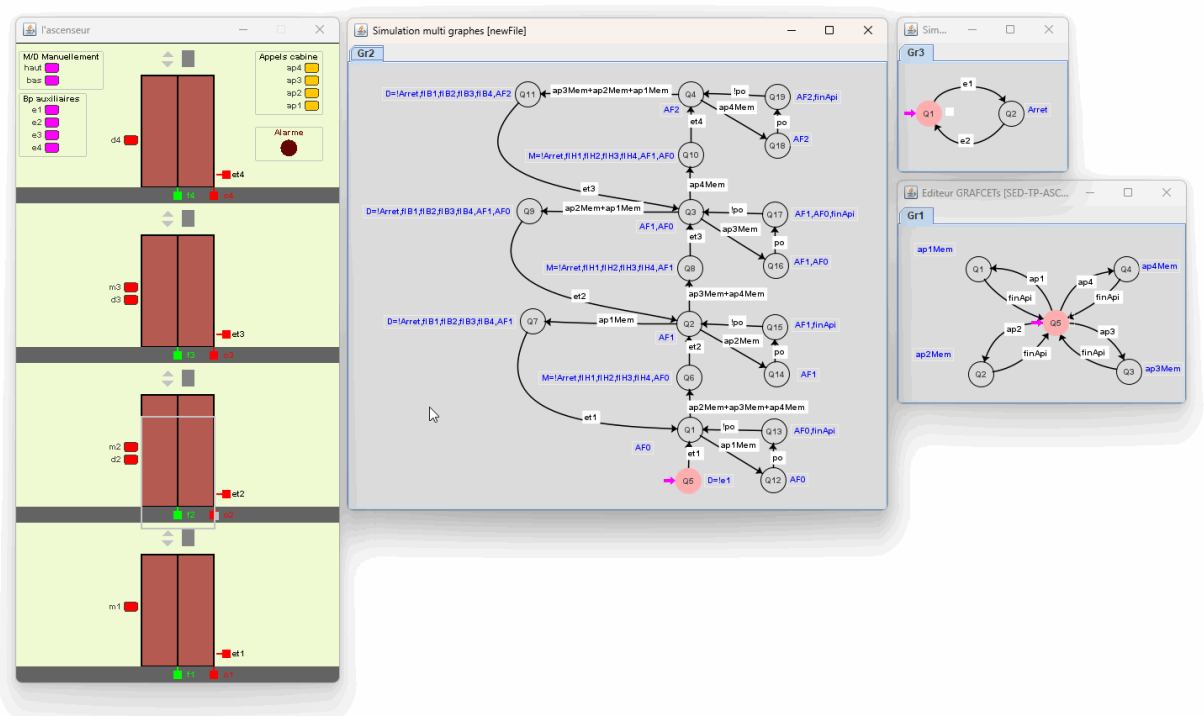


Figure 6 - Simulation du fonctionnement de l'ascenseur avec les appels internes et la prise en compte des arrêts d'urgence

2.2.3 Expressions logiques

Soit les expressions logiques correspondant à une réalisation asynchrone en rebouclage direct exposés Figure 7.

```
q11=(ap3Mem+ap2Mem+ap1Mem).Q4+!(et3).Q11
q12=ap1Mem.Q1+!(po).Q12
q13=po.Q12+!(po).Q13
q14=ap2Mem.Q2+!(po).Q14
q15=po.Q14+!(po).Q15
q16=ap3Mem.Q3+!(po).Q16
q17=po.Q16+!(po).Q17
q18=ap4Mem.Q4+!(po).Q18
q19=po.Q18+!(po).Q19

AF0=Q1+Q3+Q6+Q9+Q10+Q12+Q13+Q16+Q17
AF1=Q2+Q3+Q7+Q8+Q9+Q10+Q14+Q15+Q16+Q17
AF2=Q4+Q11+Q18+Q19
D=/e1.Q5+!Arret.Q7+!Arret.Q9+!Arret.Q11
M=!Arret.Q6+!Arret.Q8+!Arret.Q10
finApi=Q13+Q15+Q17+Q19
flB1=Q7+Q9+Q11
flB2=Q7+Q9+Q11
flB3=Q7+Q9+Q11
flB4=Q7+Q9+Q11
flH1=Q6+Q8+Q10
flH2=Q6+Q8+Q10
flH3=Q6+Q8+Q10
```

Figure 7 - Expressions symboliques de la MAE avec arrêt d'urgence

2.2.4 Description de la commande (VHDL)

On donnera le nom *securite* à son entité et le nom *securite.vhd* à ce fichier. De plus, on veillera à bien afficher le numéro de l'état présent sur les afficheurs 7 segments pour faciliter le débogage.

Pour plus d'information sur la description du composant en VHDL, voir Annexe 2 : securite.vhd.

2.2.5 Validation sur support physique

La maquette réagis correctement à la commande décrite en langage VHDL.

2.2.6 Brochage du composant matériel MAX7000s

Avant d'envoyer le VHDL au composant, on se doit d'attribuer les différents signaux d'entrée et de sortie (décrit dans l'entité) aux broches du composant matériel MAX7000s comme décrit Figure 8 sur le logiciel Quartus II comme illustré Figure 9.

Note : Sur la Figure 9, le brochage « *Fitter Location* » est le brochage par défaut, alors que le brochage « *Location* » est le brochage que nous avons affecté en adéquation avec celui présent Figure 8.

signal broche N°
écuritéPertoire de travail.
AP1.. AP4 =(6,8,9,10)
ET1.. ET4 = (15,17,20,22)
PorteOuverte = 11
Montée = 40
Descente = 36
(unites6..unites0) = (69,70,73,74,76,75,77)
(dizaines6..dizaines0) = (58,60,61,63,64,65,67)
Pour la suite de la manipulation
P4d, P3d, P2d, P3m, P2m, P1m =(5,4,18,16,21,25)
Reset, Reprise, Arrêt sont pris sur les interrupteurs
Ei :(E4 … E0) =(31 …27)

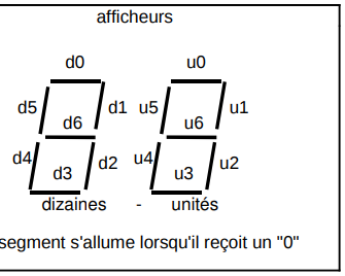
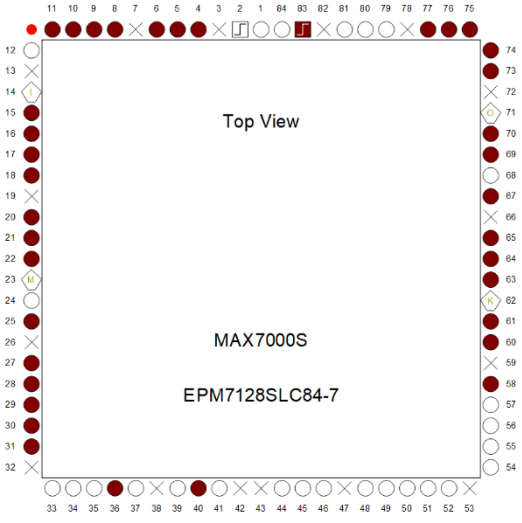


Figure 8 - Pin out (à gauche) et informations relatives aux 7 segments (à droite)



Node Name	Direction	Location	Fitter Location	Reserved
AP[3]	Input	PIN_10	PIN_10	
AP[2]	Input	PIN_9	PIN_9	
AP[1]	Input	PIN_8	PIN_8	
AP[0]	Input	PIN_6	PIN_6	
descend	Output	PIN_36	PIN_36	
dizaines[6]	Output	PIN_58	PIN_58	
dizaines[5]	Output	PIN_60	PIN_60	
dizaines[4]	Output	PIN_61	PIN_61	
dizaines[3]	Output	PIN_63	PIN_63	
dizaines[2]	Output	PIN_64	PIN_64	
dizaines[1]	Output	PIN_65	PIN_65	
dizaines[0]	Output	PIN_67	PIN_67	
E[4]	Input	PIN_31	PIN_31	
E[3]	Input	PIN_30	PIN_30	
E[2]	Input	PIN_29	PIN_29	
E[1]	Input	PIN_28	PIN_28	
E[0]	Input	PIN_27	PIN_27	
ET[3]	Input	PIN_22	PIN_22	
ET[2]	Input	PIN_20	PIN_20	
ET[1]	Input	PIN_17	PIN_17	
ET[0]	Input	PIN_15	PIN_15	
Horloge	Input	PIN_83	PIN_83	
montee	Output	PIN_40	PIN_40	
PorteOuverte	Input	PIN_11	PIN_11	
TCK	Input		PIN_62	
TDI	Input		PIN_14	
TDO	Output		PIN_71	
TMS	Input		PIN_23	
unites[6]	Output	PIN_69	PIN_69	
unites[5]	Output	PIN_70	PIN_70	
unites[4]	Output	PIN_73	PIN_73	
unites[3]	Output	PIN_74	PIN_74	
unites[2]	Output	PIN_76	PIN_76	
unites[1]	Output	PIN_75	PIN_75	
unites[0]	Output	PIN_77	PIN_77	
P1m	Unknown	PIN_25		
P2d	Unknown	PIN_18		
P2m	Unknown	PIN_21		
P3d	Unknown	PIN_4		
P3m	Unknown	PIN_16		
P4d	Unknown	PIN_5		
<<new node>>				

Figure 9 - Brochage réalisé sur Quartus II

Note : Les signaux TCK, TDI, TDO et TMS sont des signaux JTAG générés automatiquement pour le débogage. Ils ne sont donc pas à prendre en compte.

2.3 Mémorisation des appels paliers

On souhaite maintenant que les appels paliers soit pris en compte à n'importe quel moment. Un appel pour l'étage x est mémorisé dès que l'expression $Pxd + Pxm$ devient "vraie", en d'autres termes, peu importe si l'utilisateur souhaite descendre ou monter lorsqu'il appelle l'ascenseur depuis le palier, l'appel demandera à l'ascenseur de se diriger vers le palier. L'appel sera ensuite oublié dès que la porte de la cabine s'ouvre à l'étage x (voir figure suivante).

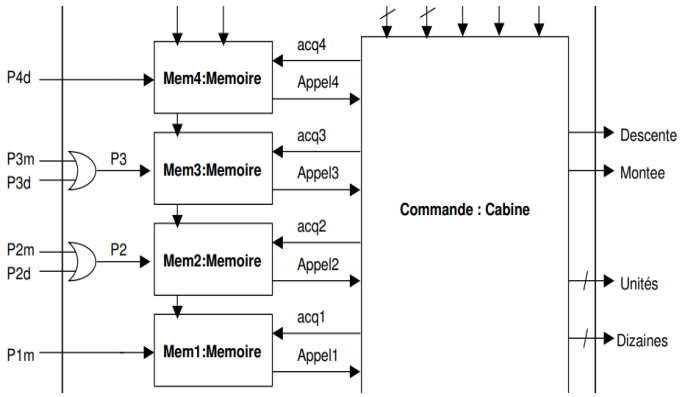


Figure 10 - Structure composant avec mémorisation

2.3.1 Modèle FMG

Pour rappel, les poussoirs paliers sont interprétés par les signaux $P4d, P3d, P2d, P3m, P2m, P1m$. Ainsi pour cette commande on aura besoin des capteurs et actionneurs répertoriés dans le Tableau 4.

	Simulateur	Maquette physique
Actionneurs de déplacement de la cabine	M, D	$montee, descend$
Capteurs d'étage	$et1, et2, et3, et4$	$ET[3..0]$
Poussoirs d'appel interne de la cabine	$ap1, ap2, ap3, ap4$	$AP[3..0]$
Contact d'ouverture des portes	po	$PortOuverte$
Poussoir palier	$d4, d3, d2, m3, m2, m1$	$P4d, P3d, P2d, P3m, P2m, P1m$
Afficheur 7 segment	$(non\ défini)$	$dizaine[6..0], unite[6..0]$
Horloge	$(non\ défini)$	$Horloge$
Poussoir d'initialisation, d'arrêt et de reprise	$(non\ défini), e1, e2$	$E(0), E(1), E(2)$

Tableau 4 - E/S nécessaires au fonctionnement de l'ascenceur incluant la mémorisation des appels

A partir de ces signaux externes (les entrées et sorties du système) nous pouvons alors construire le diagramme FMG illustré Figure 11 ou l'on veillera à remplacer le signal interne $finApi$ que l'on remplacera par les signaux $Appel1$, $Appel2$, $Appel3$, $Appel4$ correspondants aux appels des bouton poussoir interne ou externe de l'ascenseur pour chacun des 4 étages.

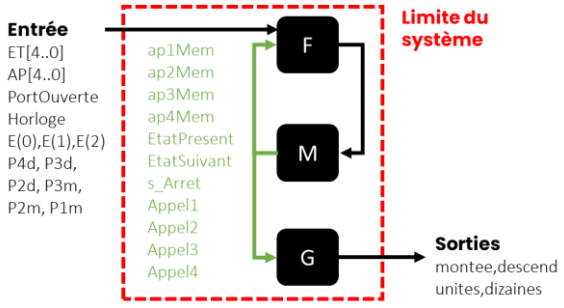


Figure 11 - Schéma bloc FMG basé sur les E/S du modèle physique pour le fonctionnement de l'ascenseur incluant les appels paliers

On peut alors décrire le schéma bloc FMG sous la forme de l'entité VHDL suivante :

```
ENTITY memorisation IS
PORT(
            E                                   : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
            ET                                  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
            AP                                  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
            P4d,P3d,P2d,P3m,P2m,P1m   : IN STD_LOGIC;
            PortOuverte               : IN STD_LOGIC;
            Horloge                             : IN STD_LOGIC;
            montee,descend                      : OUT STD_LOGIC;
            unites,dizaines                     : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END memorisation;
```

2.3.2 Modélisation et simulation de la commande

Rajouter et mémoriser les poussoirs des paliers implique de diviser la MAE de mémorisation créé plus tôt, en 4 MAE distinctes (1 MAE par étage) disposant de deux états chacun (un état « ON » et un état « OFF »). On se retrouve alors avec :

- Une MAE d'arrêt d'urgence (voir Gr1 sur Figure 12)
- Une MAE pour la commande du déplacement de l'ascenseur et des portes (voir Gr2 sur Figure 12)
- Une MAE de mémorisation des appels internes ou externes de l'étage 1 (voir Gr3 sur Figure 12)
- Une MAE de mémorisation des appels internes ou externes de l'étage 2 (voir Gr4 sur Figure 12)
- Une MAE de mémorisation des appels internes ou externes de l'étage 3 (voir Gr5 sur Figure 12)
- Une MAE de mémorisation des appels internes ou externes de l'étage 4 (voir Gr6 sur Figure 12)

Ainsi, on pourra garantir qu'à chaque fois qu'un bouton poussoir (interne ou externe) sera appelé il sera sauvegardé en mémoire.

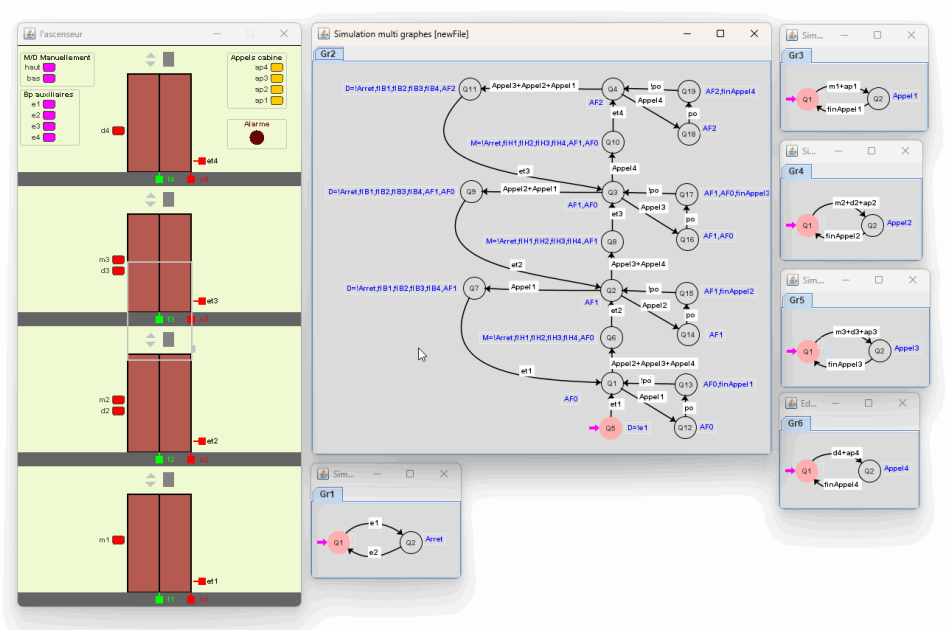


Figure 12 - Simulation du fonctionnement de l'ascenseur avec la mémorisation des appels et la prise en compte des arrêts d'urgence

2.3.3 Description de la commande (VHDL)

On donnera le nom $memorisation$ à son entité et le nom $memorisation.vhd$ à ce fichier. De plus, on veillera à bien afficher le numéro de l'état présent sur les afficheurs 7 segments pour faciliter le débogage.

Pour plus d'information sur la description du composant en VHDL, voir Annexe 3 : memorisation.vhd.

2.3.4 Validation sur support physique

La maquette réagis correctement à la commande décrite en langage VHDL.

Commande d'un banc de contrôle industriel par automate programmable TSX 57

1 But de la manipulation

Il s'agit d'effectuer la mise en œuvre d'une commande basée sur un modèle à événements discrets. Deux techniques de mise en œuvre doivent être envisagées :

1. Par description des expressions logiques des variables internes,
2. Par description directe du graphe d'état.

Cette manipulation permet l'utilisation d'un environnement de programmation industriel (UNITY de SCHNEIDER-TELEMECANIQUE) destiné à un automate programmable industriel.

2 Matériel utilisé

Le système de commande est un automate TSX57 de Télémécanique équipé d'une extension comportant 16 entrées et 8 sorties. La programmation de cet automate se fait sous l'environnement « **UNITY** ». Cet atelier permet la création, la mise au point, la documentation et l'archivage d'applications de commande. Le système commandé est constitué par une maquette de **Banc de Contrôle Industriel (BCI)** représentant un dispositif capable de réaliser des fonctions de tri, d'assemblage, d'évacuation, de contrôle des composants. Il figure une chaîne de capsulage de bouteilles. Ce système est par nature séquentiel car les pièces doivent être triées, puis assemblées et enfin contrôlées.

Il y a deux types de composants gérés par le système :

- Les « pièces hautes » en aluminium que nous appellerons les bouteilles,
- Les « pièces basses » en plastique que nous appellerons les bouchons. Le système peut être commandé soit à partir d'un PC, soit à partir d'un automate

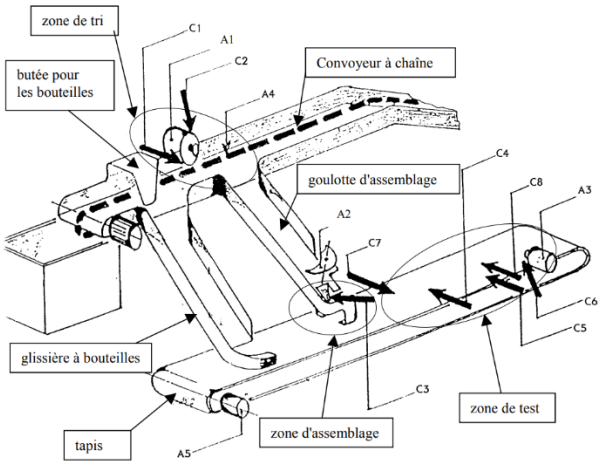


Figure 13 - Maquette physique BCI

Capteurs

- C_1 - zone de tri : détecte la présence d'une pièce,
- C_2 - zone de tri : détecte une bouteille,
- C_3 - zone d'assemblage : détecte la présence d'un bouchon dans la zone d'assemblage,
- C_4 : détecte toutes les pièces posées sur le tapis,
- C_5 : détecte toutes les pièces posées sur le tapis,
- C_6 : détecte toutes les pièces posées sur le tapis,
- C_7 : détecte une bouteille posée sur le tapis,
- C_8 : détecte un bouchon sur une bouteille.

Actionneurs

- A_1 - solénoïde de tri : permet d'éjecter les bouchons dans la goulotte d'assemblage,
- A_2 - solénoïde rotatif : positionne un par un les bouchons dans la zone d'assemblage,
- A_3 - solénoïde d'éjection : éjecte du tapis la pièce qui se trouve devant le capteur C6,
- A_4 - convoyeur à chaîne : amène les pièces séparées en zone de tri,
- A_5 - tapis : amène les bouteilles en zone d'assemblage et achemine les pièces que l'on suppose assemblées en zone de test.

2.1 Tri des pièces avec arrivée dans le bon ordre

2.1.1 Hypothèses

- Les pièces se présentent dans la zone de tri dans le bon ordre (bouchon-bouteille-bouchon-bouteille…).
- **Un délai de 1,5 seconde** doit être implémenté entre l'instant où la pièce « bouchon » est poussée dans la goulotte par l'**actionneur** d'assemblage (A_1) et l'instant d'ouverture du solénoïde rotatif (A_2) pour permettre la descente de cette pièce dans la zone d'assemblage.

2.1.2 Etapes de la commande à réaliser

- Le tapis est entrainé en continu,
- Lorsque le capteur C_1 détecte la présence d'une pièce, et le capteur C_2 ne détecte pas une bouteille, le bouchon est alors expulsé via l'actionneur A_1 dans la goulotte et un temporisateur $fintempo_1$ est lancé.
- Lorsque le temporisateur $fintempo_1$ arrive à terme et que le capteur C_3 ne détecte pas la présence d'un bouchon dans la zone d'assemblage le solénoïde rotatif A_2 et activé afin que le bouchon s'insère dans la zone d'assemblage.
- Lorsque le capteur C_3 détecte un nouveau bouchon dans la zone d'assemblage il attend un temps $fintempo_2$ avant que le solénoïde rotatif se referme (pour éviter qu'il ne se referme trop rapidement) et on attend qu'une nouvelle pièce arrive devant le capteur C_1.

2.1.3 Modélisation et simulation de la commande

Note : Initialisation de l'état à Q_1 via l'interrupteur 27 disponible via la variable $init$ (Bloc M)

La simulation a permis de mettre au jour une contrainte : le délai trop faible ($< 1{,}5s$) entre chaque pièce. Pour pallier le problème, la solution adoptée a été de dissocier la gestion du temps et la gestion de la goulotte dans des machines à états séparés. De plus, comme les capteurs C_1 et C_2 ne sont pas l'un sur l'autre mais l'un après l'autre, et comme le simulateur l'a prouvé, la condition initiale $C_1.\overline{C_2}$ ne peut pas s'appliquer correctement. A titre d'exemple le simulateur poussait même des bouteilles dans la goulotte.

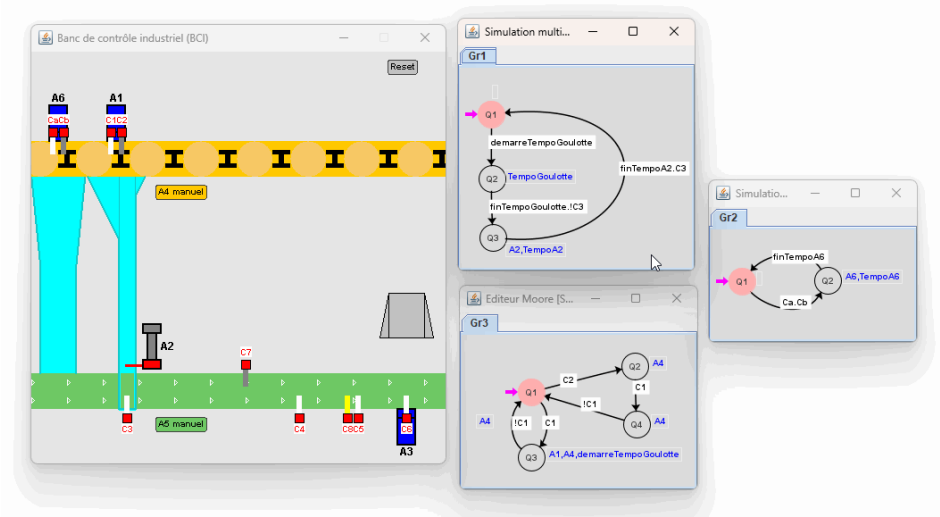


Figure 14 - Simulation du tri des pièces avec arrivée dans le bon ordre

Important : La MAE de gestion de l'actionneur $A6$ (situé dans la fenêtre $Gr2$ de la Figure 14) a été modélisée pour simuler correctement le comportement de la maquette simulée. Néanmoins par la suite nous ne la programmerons puisque cet actionneur et ces capteurs n'existent pas sur la maquette réelle.

2.1.4 Programmation de la commande simulée (langage ST)

Pour plus d'information sur la description comportement du système en langage ST, voir Annexe 4 : Gestion de la zone de tri (BCI).

Note : Le programme de la maquette physique du BCI comporte 2 machines à états fonctionnant en parallèle.

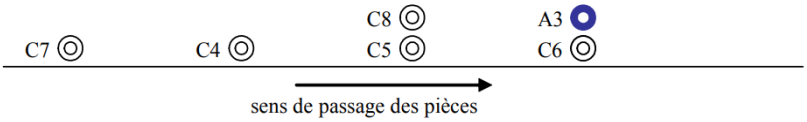
2.1.5 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage C.

2.2 Test de l'assemblage et éjection des pièces isolées

2.2.1 Hypothèses

La zone de test comporte une série de capteurs permettant de détecter le type des pièces qui y circulent (bouteille, bouchon ou assemblage). En fin de zone, un **actionneur** (A_3) permet d'éjecter une pièce quelconque hors du tapis.



Avec pour rappel :

- C_4 : détecte toutes les pièces posées sur le tapis,
- C_5 : détecte toutes les pièces posées sur le tapis,
- C_6 : détecte toutes les pièces posées sur le tapis,
- C_7 : détecte une bouteille posée sur le tapis,
- C_8 : détecte un bouchon sur une bouteille.
- A_3 - solénoïde d'éjection : éjecte du tapis la pièce qui se trouve devant le capteur C_6,
- A_5 - tapis : amène les bouteilles en zone d'assemblage et achemine les pièces que l'on suppose assemblées en zone de test.

2.2.2 Etapes de la commande à réaliser

- Le tapis est entrainé en continu à l'aide de l'actionneur A_5,
- Lorsque le capteur C_7 détecte une bouteille posée sur le tapis, alors
- Lorsque le capteur C_4 détecte une pièce, alors
- Lorsque le capteur C_8 ne détecte pas de bouchon sur une bouteille alors que le capteur C_5 détecte la présence d'une pièce (la bouteille), alors
- Lorsque le capteur C_6 détecte la pièce celle-ci sera alors expulsé via l'actionneur A_3 et lance un temporisateur pour éviter que la réponse de l'actionneur soit trop rapide.

2.2.3 Modélisation et simulation de la commande

Cette commande modélise la zone de test afin d'éjecter les pièces seules issues d'éventuels échecs du processus d'assemblage.

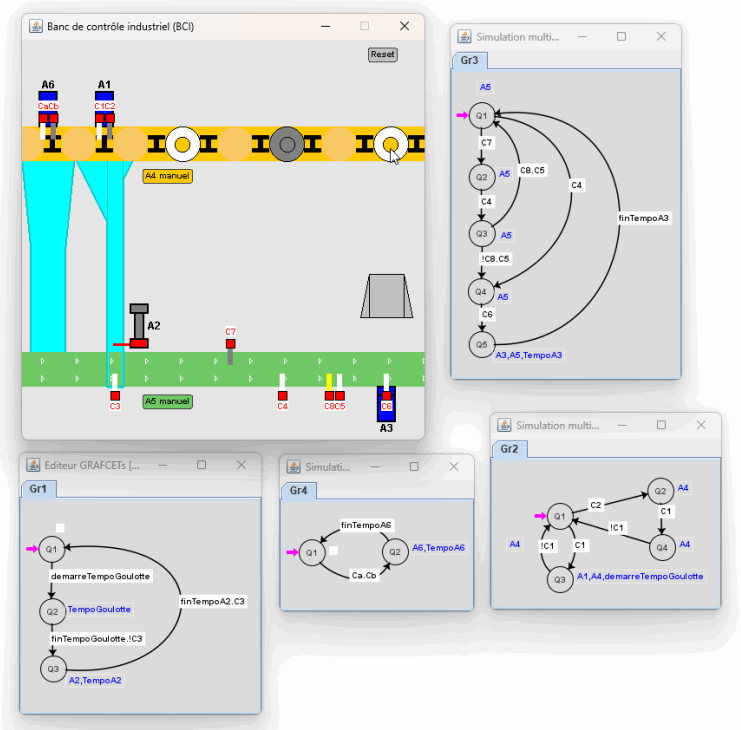


Figure 15 – Simulation de l'assemblage et de l'éjection des pièces isolées

2.2.4 Programmation de la commande (langage ST)

Pour plus d'information sur la description comportement du système en langage ST, voir Annexe 5 : .

Note : Le programme de la maquette physique du BCI comporte 3 machines à états fonctionnant en parallèle.

2.2.5 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage ST.

2.3 Fonctionnement global du système dans le cas général

2.3.1 Hypothèses

- Les pièces arrivent dans un ordre aléatoire
- La goulotte du système d'assemblage peut contenir aux plus deux bouchons. Ainsi, si trois bouchons se présentent successivement dans la zone de tri, le troisième doit être ignoré (il n'est pas poussé dans la goulotte d'assemblage) et tombe dans un stock prévu à cet effet au bout du convoyeur à chaîne.

2.3.2 Modélisation de la commande

Ne pouvant pas faire de compteur en simulation nous avons de modéliser l'ensemble des états possible.

2.3.2.1 Simulation à vide

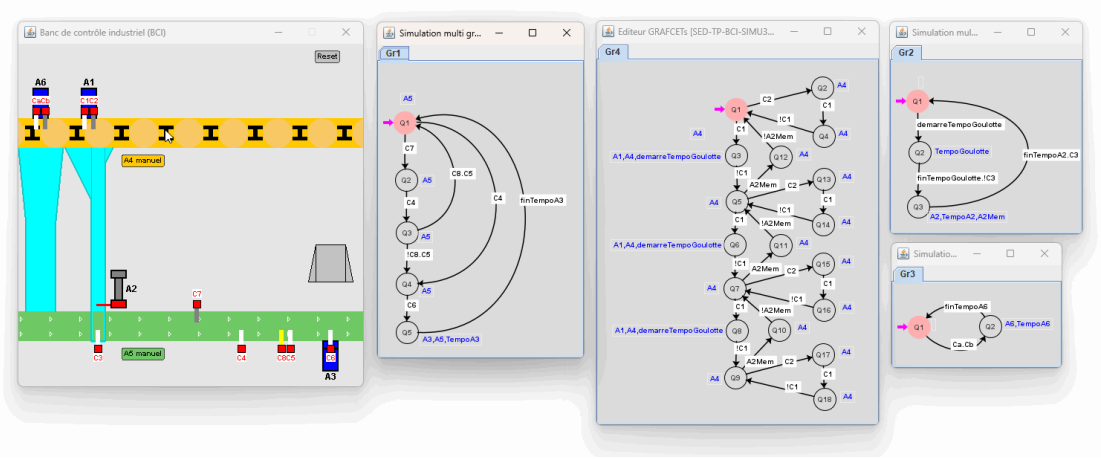


Figure 16 - Simulation du fonctionnement global du système sans bouteille

2.3.2.2 Simulation complète

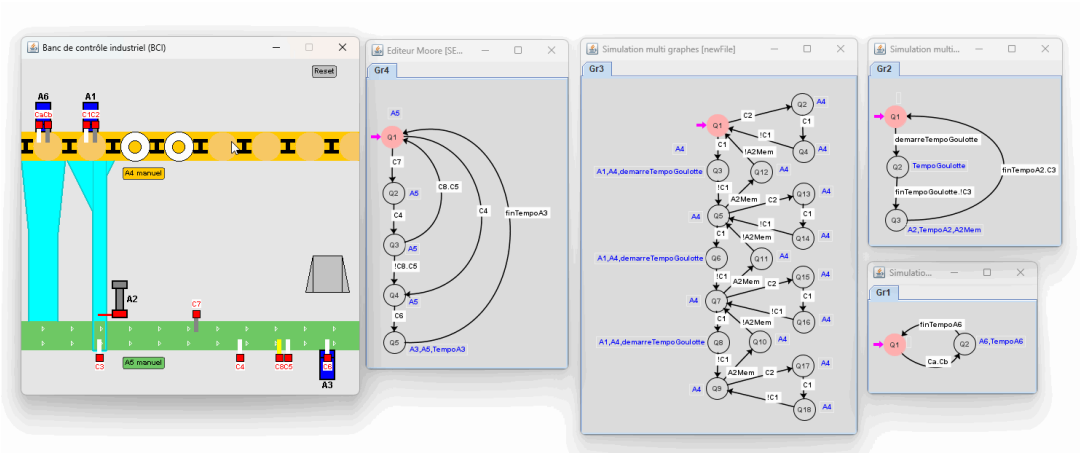


Figure 17 - Simulation du fonctionnement global du système complet

2.3.3 Programmation de la commande (langage ST)

Pour plus d'information sur la description comportement du système en langage ST, voir Annexe 6 : Application complète (BCI).

Note : Le programme comporte 3 machines à états fonctionnant en parallèle.

2.3.4 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage ST.

Système de traitement automatisé (STA) : modélisation par MAEFS et mise en œuvre en langage C

1 Objectif

L'objectif est de modéliser par graphe d'états, puis de réaliser la commande d'une maquette de traitement automatisé. Pour cela un certain nombre d'opérations élémentaires sont modélisées par des machines à états finis (MEF), puis synchronisées afin que le fonctionnement d'ensemble respecte un cahier des charges. La mise en œuvre des MEF sera logicielle, en langage C sous Linux.

2 Description de la maquette

2.1 La maquette physique

La maquette de traitement automatisé permet de simuler des fonctions de fabrication de circuit imprimés, de traitement chimique ou de nettoyage de pièces. Cette maquette est composée de 5 bacs ayant chacun au moins un emplacement pour poser le support transporté par le chariot mobile :

- Le bac de droite, dit d'entrée possède 3 emplacements
- Le bac de gauche, dit de sortie possède 2 emplacements
- Les bacs intermédiaires, dit de trempage, sont nommés $bac1$, $bac2$ et $bac3$ et sont positionnés respectivement du bac d'entrée vers le bac de sortie (de droite à gauche).

Pour déplacer les supports, un chariot se déplace horizontalement pour amener celui-ci au-dessus du bac désiré et verticalement pour prendre ou déposer le support.

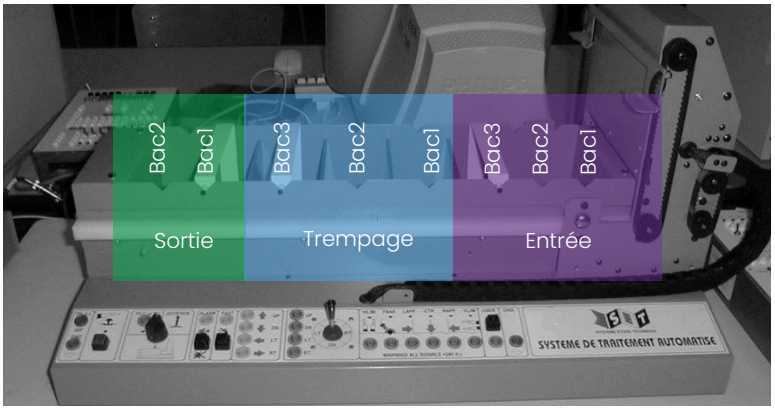


Figure 18 - Maquette physique avec les 3 bacs d'entrée (en violet), les 3 bacs de trempage (en bleu) et les 2 bacs de sortie (en vert)

2.2 La maquette simulée

Un simulateur fourni par les enseignants de l'UPSSITECH permet de vérifier le fonctionnement de la machine à état avant de la coder en C sur Linux. On notera que le nombre d'emplacement n'est pas le même que sur la maquette réelle (10 emplacements en simulation contre 8 en réalité). Ainsi, pour se rapprocher au maximum du modèle physique, nous avons donc décidé de ne pas prendre en compte les deux derniers emplacements en simulation.

Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

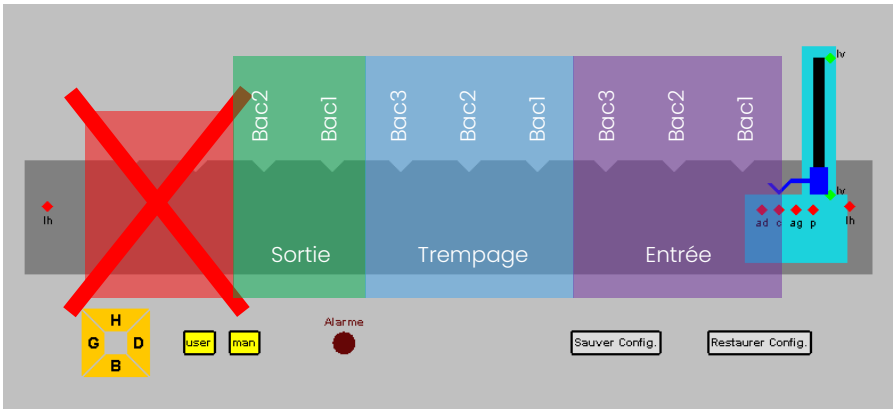


Figure 19 - Simulateur STA corrigé avec les 3 bacs d'entrée (en violet), les 3 bacs de trempage (en bleu), les 2 bacs de sortie (en vert) et les bacs qui ne seront pas pris en compte (en rouge)

Capteurs

- $LIMHOR$ et $LIMVER$ – Signaux émis par 2 capteurs de fin de course fixés sur le chariot qui définissent les limites de déplacements horizontaux et verticaux
- $APPG, CTR, APPD$ – Signaux émis par 3 capteurs de positionnement du chariot représentant respectivement l'Approche Gauche, le Centre et l'Approche Droite d'un emplacement de support
- $PRESENCE$ – Signal émis par 1 capteur de présence (du support) délivrant le signal valide lorsque $APPG$ est vrai
- $OPERATEUR$ – Signal délivré par 1 poussoir permettant une intervention de l'utilisateur

Actionneurs

- V_ACC – Pour augmenter la vitesse du chariot
- $HAUT, BAS, GAUCHE, DROITE$ – Pour déplacer le chariot
- $ALARME$ – Pour signaler la fin de course à l'utilisateur

3 Fonctionnement désiré

Le chariot est initialement à l'extrémité droite du rail ($LIMHOR$) en position basse ($LIMVER$) et sera toujours ramené à cette position à la fin du mouvement. Les supports à traiter son mis en place manuellement et doivent être traité peu importe l'emplacement du bac d'entrée sur lequel ils sont positionné. Un nouveau support n'est pris en compte que si le précédent n'est plus sur le bac d'entrée. Au fur et à mesure de l'arrivée du support (et de la disponibilité des bacs) ceux-ci sont transportés prioritairement vers le bac 1, sinon vers le bac 2 pour subir une opération de trempage du durée minimale 8 secondes. A la fin de cette opération les supports sont évacués et stockés dans un emplacement libre du bac de sortie. L'opération de déchargement des bacs est manuelle.

4 Travail demandé

On s'intéressera d'abord aux opérations élémentaires de transport d'un support :

- Du bac d'entrée vers le bac 1 et traitement dans le bac 1
- Du bac d'entrée vers le bac 2 et traitement dans le bac 2

Puis on ajoutera l'opération de détection d'une pièce (signalé par l'opérateur) et du choix du bac vers lequel transporter la pièce. On sera alors amené à mettre en œuvre des MEF synchronisées. Par la suite, on introduira les fonctions d'évacuation des bacs 1 et 2 vers le bac de sortie.

4.1 Traitement dans le bac 1 de trempage

4.1.1 Les étapes de la commande

- Déplacement à la position initiale en BAS a $DROITE$ et vérification des limites $LIMHOR$ et $LIMVER$
- Déplacement vers la $GAUCHE$ et détection d'un bac via le signal $PRESENCE$ lorsque l'on atteint l'$APPG$ de l'emplacement
- Déplacement vers la $DROITE$ afin de se positionner au centre CTR des prises du support
- Déplacement vers le $HAUT$ jusqu'à $LIMVER$ pour surélever le support
- Déplacement vers la $GAUCHE$ pour positionner le support dans le $bac1$ de la zone de trempage ($PRESENCE = 0$)
- Déplacement vers la $DROITE$ jusqu'au centre CTR
- Déplacement vers le BAS jusqu'à $LIMVER$ pour positionner le support
- Lancement du délai de 8 secondes
- Déplacement vers la $DROITE$ en vitesse accélérée V_ACC jusqu'à la position initiale

4.1.2 Modélisation et simulation de la commande

On se rend vite compte de la limitation du logiciel de simulation car il est impossible d'y mettre en place un compteur (pour compter le nombre de CTR par exemple), il va donc falloir utiliser un modèle de machine a état proche du modèle physique (la maquette) avec un nombre important d'états.

De plus, on va être obligé de déclarer une temporisation supplémentaire pour éviter que les fins de courses ne soient détectées avant l'action demandée. Par exemple si on souhaite faire monter le robot alors qu'on est en bas et que l'on touche la fin de course, on passera directement à l'état suivant, avant même que le moteur ne réagisse à la commande de montée.

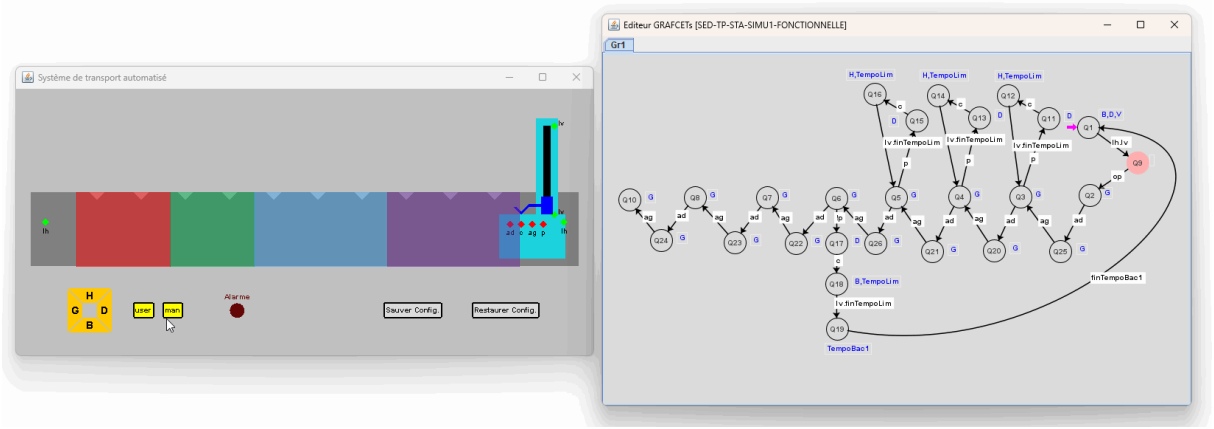


Figure 20 - Simulation de la commande de traitement dans le bac1 de trempage

4.1.3 Programmation en C

Pour plus d'informations sur le code C : voir Annexes 7 : Traitement dans le bac1 de trempage (STA)

4.1.4 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage C.

4.2 Traitement dans le bac 2 de trempage

4.2.1 Les étapes de la commande

- Déplacement à la position initiale en BAS a $DROITE$ et vérification des limites $LIMHOR$ et $LIMVER$
- Déplacement vers la $GAUCHE$ et détection d'un bac via le signal $PRESENCE$ lorsque l'on atteint l'$APPG$ de l'emplacement
- Déplacement vers la $DROITE$ afin de se positionner au centre CTR des prises du support
- Déplacement vers le $HAUT$ jusqu'à $LIMVER$ pour surélever le support
- Déplacement vers la $GAUCHE$ pour positionner le support dans le $bac2$ de la zone de trempage $(PRESENCE = 0)$
- Déplacement vers la $DROITE$ jusqu'au centre CTR
- Déplacement vers le BAS jusqu'à $LIMVER$ pour positionner le support
- Lancement du délai de 8 secondes
- Déplacement vers la $DROITE$ en vitesse accélérée V_ACC jusqu'à la position initiale

4.2.2 Modélisation et simulation de la commande

Ici il y a donc peu de changement à faire sur la machine à état réalisée précédemment. Nous avons juste déplacé les états de traitement de la pièce qui étaient préalablement sur le $bac1$ vers le $bac2$.

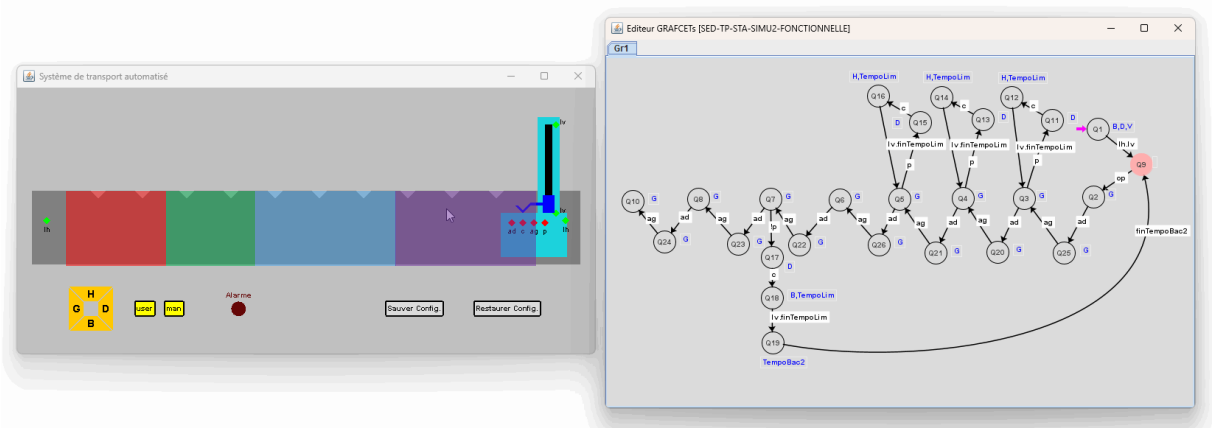


Figure 21 - Simulation de la commande de traitement dans le bac2 de trempage

4.2.3 Programmation en C

Pour plus d'informations sur le code C : voir Annexes 8 : Traitement dans le bac2 (STA).

4.2.4 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage C.

4.3 Gestion des transports pour les 2 bacs

Les deux machines à états (MAE) précédentes étant mise au point on veut maintenant :

- Détecter l'information de présence d'une pièce dans le bac d'entrée (signalé par $OPERATEUR$)
- Demander le transport et le traitement vers le $bac1$ (ou le $bac2$ si le $bac1$ est occupé)
- Pouvoir mémoriser la présence d'une nouvelle pièce dès que le transport est lancé. Si une nouvelle pièce est en place elle doit pouvoir être prise en compte sans attendre la fin du transport précédent (cette demande ne sera effective que si le système de transport est disponible, c'est-à-dire en fin de transport mais la structuration du poussoir doit pouvoir avoir lieu plus tôt).

On nous propose de modéliser ce fonctionnement à partir de 4 machines à états synchronisées.

1. **Demande de transport selon disponibilité des bacs** : cette MAE assure la structuration du poussoir $OPERATEUR$ et formule les demandes de transport vers le $bac1$ ou le $bac2$.
2. **Allocation du système de transport** : selon la disponibilité du système de transport et selon les demandes vers le $bac1$ ou le $bac2$ (et dans la partie suivante demandes de transport du $bac1$ ou du $bac2$ vers le bac de sortie) cette MAE fait le choix d'un transfert. Elle peut notamment ordonner 4 commandes de transport (2 pour cette partie).
3. **Transport et traitement** $bac1$ (renvoie disponibilité du $bac1$).
4. **Transport et traitement** $bac2$ (renvoie disponibilité du $bac2$).

Ces deux dernières MAE doivent être modifiées puisque les demandes ne viennent plus directement du poussoir $OPERATEUR$. La solution proposée consiste à envisager que sur appuis du poussoir $OPERATEUR$, une demande (fonction de la disponibilité des bacs) soit émise par la MAE **demande de transport** et soit adressée à la MAE **système de transport**. Celle-ci valide une demande et ordonne un transport (MEF Traitement $bac1$ ou MEF Traitement $bac2$).

4.3.1 Modélisation et simulation de la commande

4.3.1.1 Simulation à vide

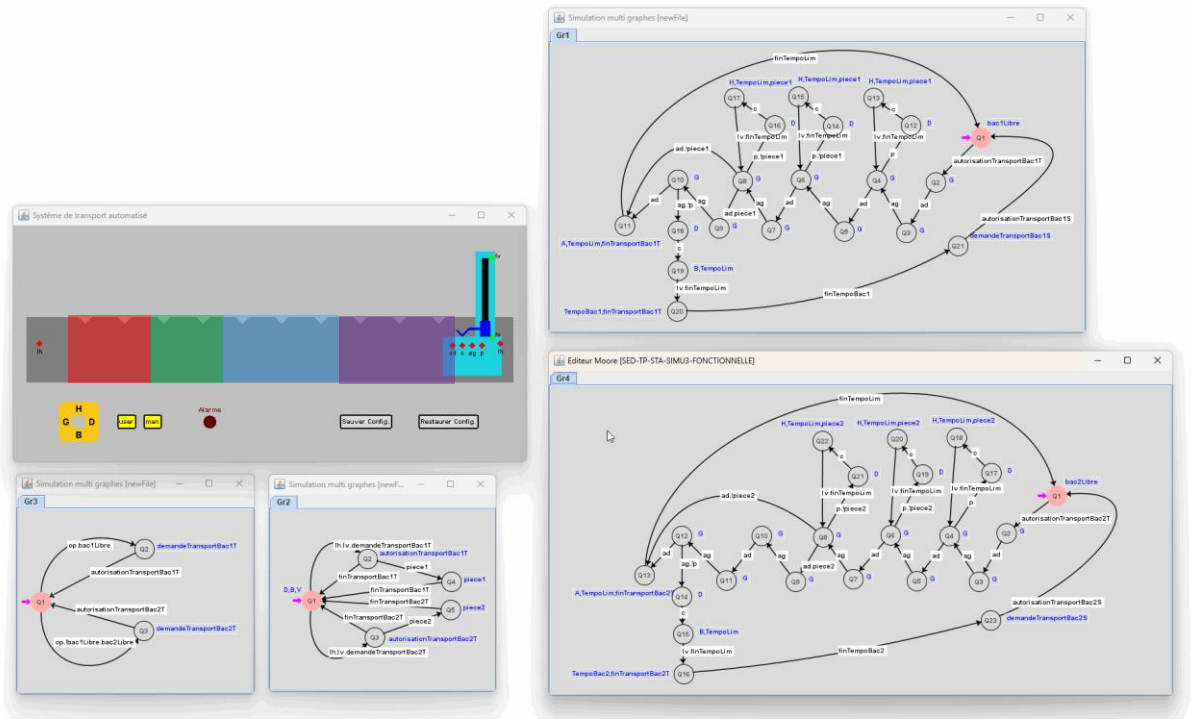


Figure 22 - Simulation à vide de la commande de gestion des transports pour les 2 bacs

Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

4.3.1.2 Simulation complète

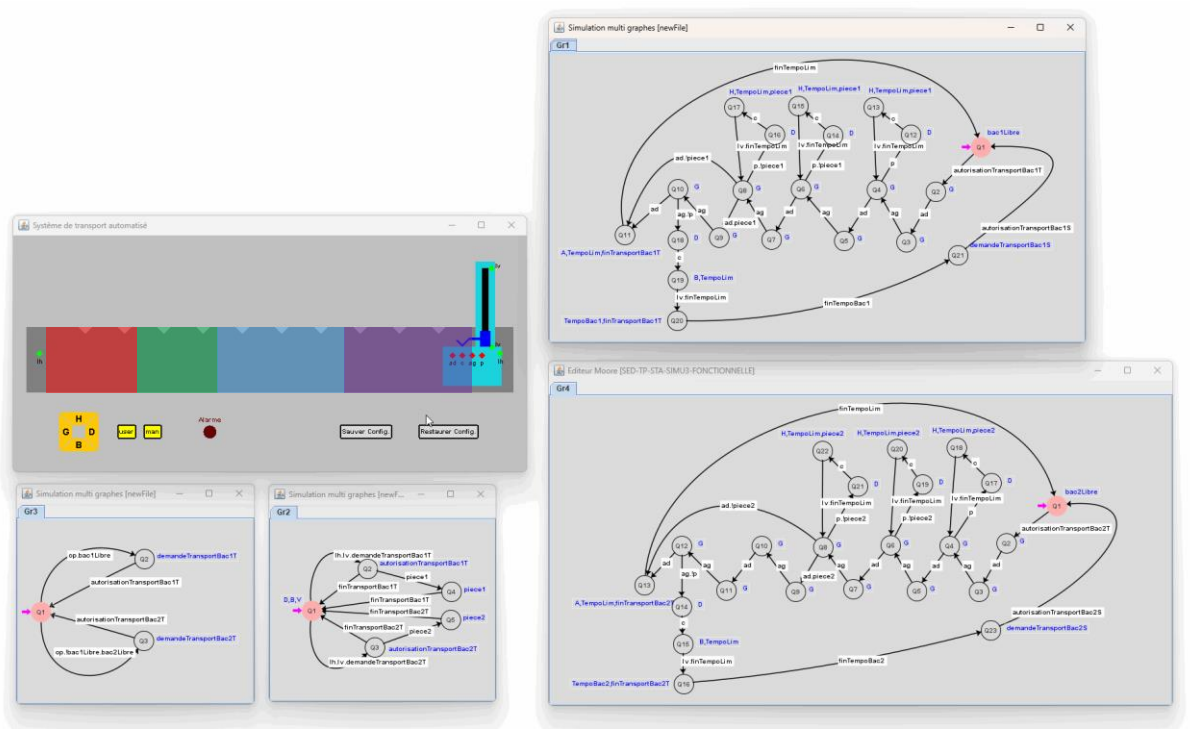


Figure 23 - Simulation chargée de la commande de gestion des transports pour les 2 bacs

4.3.2 Programmation en C

Pour plus d'informations sur le code C : voir Annexes 9 : Gestion des transports pour les 2 bacs (STA).

4.3.3 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage C.

4.4 Application complète

A la fin de la phase de trempage les MAE de traitement $bac1$ et $bac2$ demandent l'évacuation des pièces vers le bac de sortie. Ces machines à états sont fournies sous la forme de leurs fonctions F, M et G.
Le but : synchroniser ces 2 nouvelles MAE avec les 4 précédentes. En ce but, il est nécessaire de compléter la MAE système de transport et les MAE traitement $bac1$ et traitement $bac2$ qui font appel aux services rendus par les MAE d'évacuation. Les demandes d'évacuation sont adressées par ces MAE à la MAE système de transport qui validera ces demandes (dès que possible) par un signal d'autorisation. La synchronisation de ces 2 dernières MAE se fait à partir de 2 signaux « autorisation de transport » et « fin de transport », comme l'indiquent les prototypes fournis.

4.4.1 Modélisation et simulation de la commande

4.4.1.1 Simulation à vide

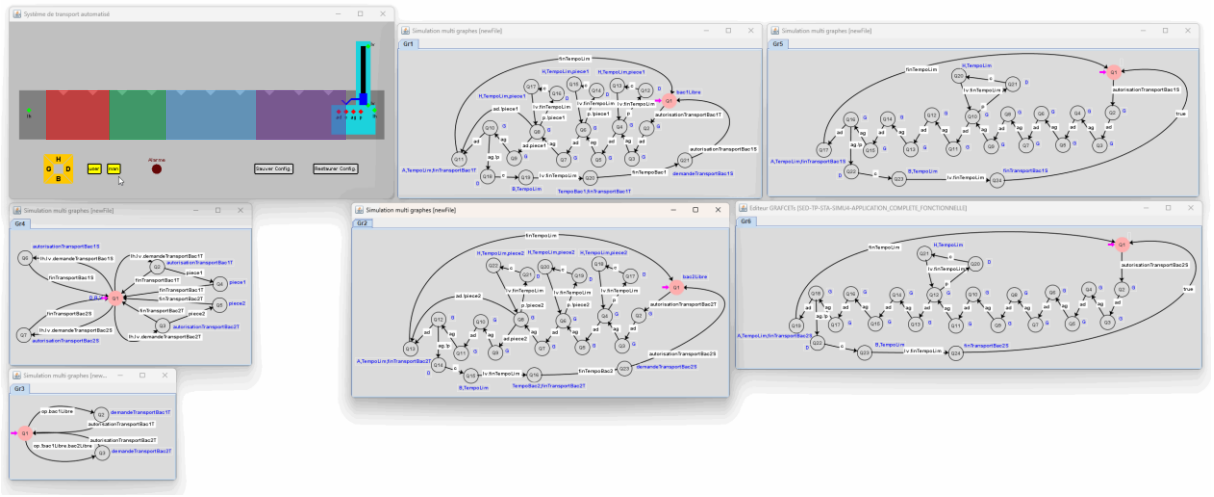


Figure 24 - Simulation à vide du fonctionnement complet

4.4.1.2 Simulation complète

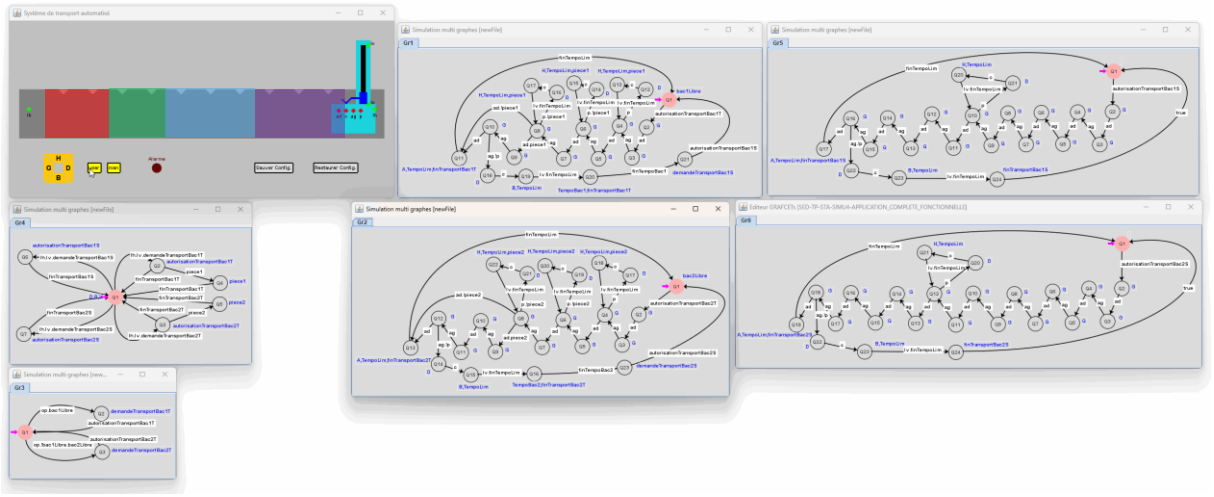


Figure 25 - Simulation chargée de la commande du fonctionnement complet

4.4.2 Programmation en C

Pour plus d'informations sur le code C : voir Annexes 10 : Application complète (STA).

4.4.3 Validation sur support physique

La maquette réagis correctement à la commande programmée en langage C.

Conclusion générale

Malgré le peu de temps en salle de TP et la charge de travail personnel importante nous somme fier d'avoir tout de même pu réaliser l'ensemble des validations demandés, nous permettant de consolider nos compétences en modélisation de machine à états et de nous initier à la modélisation de machine à états synchronisées. Nous sommes convaincus que ces heures de TP nous seront bénéfique à l'avenir quant à la réalisation de nos futurs projet personnels comme professionnels.

Nous souhaitons, tout particulièrement remercier Nahla Tabti pour ces heures passé a nous expliqués et réexpliquer le fonctionnement des maquettes dans la bonne humeur ainsi que pour ces précieux conseils.

Annexes

Annexe 1 : cabine.vhd

Description VHDL du composant MAX7000S pour la commande de l'ascenseur avec les poussoirs internes

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY cabine IS
PORT(
		E                       : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		ET                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		AP                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		PortOuverte  : IN STD_LOGIC;
		Horloge                 : IN STD_LOGIC;
		montee,descend          : OUT STD_LOGIC;
		unites,dizaines         : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END cabine;

ARCHITECTURE ar OF cabine IS

   CONSTANT zero   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000001";
   CONSTANT un     : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1001111";
   CONSTANT deux   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0010010";
   CONSTANT trois  : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000110";
   CONSTANT quatre : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1001100";
   CONSTANT cinq   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0100100";
   CONSTANT six    : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0100000";
   CONSTANT sept   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0001111";
   CONSTANT huit   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000000";
   CONSTANT neuf   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000100";
   CONSTANT erreur : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0110000";
   CONSTANT eteint : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1111111";

   TYPE M1_Etat IS (M1_Etat1, M1_Etat2, M1_Etat3, M1_Etat4, M1_Etat5, M1_Etat6, M1_Etat7, M1_Etat8, M1_Etat9, M1_Etat10, M1_Etat11, M1_Etat12,
M1_Etat13, M1_Etat14, M1_Etat15, M1_Etat16, M1_Etat17, M1_Etat18, M1_Etat19);
   TYPE M2_Etat IS (M2_Etat1,M2_Etat2,M2_Etat3,M2_Etat4,M2_Etat5);

   SIGNAL M1_EtatPresent, M1_EtatSuivant : M1_Etat;
   SIGNAL M2_EtatPresent, M2_EtatSuivant : M2_Etat;

   SIGNAL ap1Mem : STD_LOGIC;
   SIGNAL ap2Mem : STD_LOGIC;
   SIGNAL ap3Mem : STD_LOGIC;
   SIGNAL ap4Mem : STD_LOGIC;
   SIGNAL finApi : STD_LOGIC;

   BEGIN

   -- DESCRIPTION DU BLOC F
   PROCESS (AP,ET,PortOuverte,ap1Mem,ap2Mem,ap3Mem,ap4Mem)
		BEGIN
		CASE M1_EtatPresent IS
			-- traitement des etages
			WHEN M1_Etat1 => IF ap2Mem='1' OR ap3Mem='1' OR ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat6;
					ELSIF ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat12;
					END IF;
			WHEN M1_Etat2 => IF ap3Mem='1' OR ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat8;
					ELSIF ap2Mem='1' THEN M1_EtatSuivant<=M1_Etat14;
					ELSIF ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat7;
					END IF;
			WHEN M1_Etat3 => IF ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat10;
					ELSIF ap3Mem='1' THEN M1_EtatSuivant<=M1_Etat16;
					ELSIF ap2Mem='1' OR ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat9;
					END IF;
			WHEN M1_Etat4 => IF ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat18;
					ELSIF ap3Mem='1' OR ap2Mem='1' OR ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat11;
					END IF;

			--traitement des transistion entre étages
			WHEN M1_Etat5 => IF ET(0)='1' THEN M1_EtatSuivant<=M1_Etat1; END IF;
			WHEN M1_Etat6 => IF ET(1)='1' THEN M1_EtatSuivant<=M1_Etat2; END IF;
			WHEN M1_Etat7 => IF ET(0)='1' THEN M1_EtatSuivant<=M1_Etat1; END IF;
			WHEN M1_Etat8 => IF ET(2)='1' THEN M1_EtatSuivant<=M1_Etat3; END IF;
			WHEN M1_Etat9 => IF ET(1)='1' THEN M1_EtatSuivant<=M1_Etat2; END IF;
			WHEN M1_Etat10 => IF ET(3)='1' THEN M1_EtatSuivant<=M1_Etat4; END IF;
			WHEN M1_Etat11 => IF ET(2)='1' THEN M1_EtatSuivant<=M1_Etat3; END IF;

			--traitement des portes
			WHEN M1_Etat12 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat13; END IF; --etage1
			WHEN M1_Etat13 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat1; END IF;  --etage1
			WHEN M1_Etat14 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat15; END IF; --etage2
			WHEN M1_Etat15 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat2; END IF;  --etage2
			WHEN M1_Etat16 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat17; END IF; --etage3
			WHEN M1_Etat17 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat3; END IF;  --etage3
			WHEN M1_Etat18 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat19; END IF; --etage4
			WHEN M1_Etat19 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat4; END IF;  --etage4
		END CASE;
   END PROCESS;


   PROCESS (AP,finApi)
		BEGIN
		CASE M2_EtatPresent IS
			WHEN M2_Etat1 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat2 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat3 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat4 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat5 =>
				IF AP(0)='1' THEN M2_EtatSuivant<=M2_Etat1;
				ELSIF AP(1)='1' THEN M2_EtatSuivant<=M2_Etat2;
				ELSIF AP(2)='1' THEN M2_EtatSuivant<=M2_Etat3;
				ELSIF AP(3)='1' THEN M2_EtatSuivant<=M2_Etat4;
				END IF;
		END CASE;
   END PROCESS;

   -- DESCRIPTION DU BLOC M
```

```
PROCESS (Horloge,E)
 BEGIN
            -- passage à l'etat suivant
            IF ((Horloge'EVENT) AND (Horloge = '1')) THEN      M1_EtatPresent <= M1_EtatSuivant;
                                                               M2_EtatPresent <= M2_EtatSuivant;
            END IF;

            -- initialisation des etats
            IF (E(0)='1') THEN M1_EtatPresent<=M1_Etat5; M2_EtatPresent<=M2_Etat5; END IF;

END PROCESS;

-- DESCRIPTION DU BLOC G
-- sorties pour pilotage des afficheurs de la carte
with M1_EtatPresent select
            unites<=    zero        WHEN M1_Etat10,
                        un          WHEN M1_Etat1 | M1_Etat11,
                        deux        WHEN M1_Etat2 | M1_Etat12,
                        trois       WHEN M1_Etat3 | M1_Etat13,
                        quatre      WHEN M1_Etat4 | M1_Etat14,
                        cinq        WHEN M1_Etat5 | M1_Etat15,
                        six         WHEN M1_Etat6 | M1_Etat16,
                        sept        WHEN M1_Etat7 | M1_Etat17,
                        huit        WHEN M1_Etat8 | M1_Etat18,
                        neuf        WHEN M1_Etat9 | M1_Etat19,
                        erreur      WHEN OTHERS;
            dizaines<=  eteint      WHEN (M1_EtatPresent < M1_Etat10) ELSE
                        un          WHEN (M1_EtatPresent >= M1_Etat10 AND M1_EtatPresent <= M1_Etat19) ELSE
                        erreur;

-- sorties pour pilotage de la cabine
montee<='1' WHEN (M1_EtatPresent=M1_Etat6 OR M1_EtatPresent=M1_Etat8 OR M1_EtatPresent=M1_Etat10) ELSE '0';
descend<='1' WHEN (M1_EtatPresent=M1_Etat7 OR M1_EtatPresent=M1_Etat9 OR M1_EtatPresent=M1_Etat11) ELSE '0';

-- sorties pour memorisation des poussoirs
ap1Mem<='1' WHEN M2_EtatPresent=M2_Etat1          ELSE '0';
ap2Mem<='1' WHEN M2_EtatPresent=M2_Etat2          ELSE '0';
ap3Mem<='1' WHEN M2_EtatPresent=M2_Etat3          ELSE '0';
ap4Mem<='1' WHEN M2_EtatPresent=M2_Etat4          ELSE '0';

-- sortie pour mettre fin a ala memorisation du bouton poussoir interne
WITH M1_EtatPresent select
finApi      <=          '1'         WHEN M1_Etat13 | M1_Etat15 | M1_Etat17 | M1_Etat19,
                        '0'         WHEN OTHERS;


END ar;
```


Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

Annexe 2 : securite.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY securite IS
PORT(
		E                       : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		ET                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		AP                      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		PortOuverte  : IN STD_LOGIC;
		Horloge                 : IN STD_LOGIC;
		montee,descend          : OUT STD_LOGIC;
		unites,dizaines         : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END securite;

ARCHITECTURE ar OF securite IS

   CONSTANT zero   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000001";
   CONSTANT un     : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1001111";
   CONSTANT deux   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0010010";
   CONSTANT trois  : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000110";
   CONSTANT quatre : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1001100";
   CONSTANT cinq   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0100100";
   CONSTANT six    : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0100000";
   CONSTANT sept   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0001111";
   CONSTANT huit   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000000";
   CONSTANT neuf   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000100";
   CONSTANT erreur : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0110000";
   CONSTANT eteint : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1111111";

   TYPE M1_Etat IS
(M1_Etat1,M1_Etat2,M1_Etat3,M1_Etat4,M1_Etat5,M1_Etat6,M1_Etat7,M1_Etat8,M1_Etat9,M1_Etat10,M1_Etat11,M1_Etat12,M1_Etat13,M1_Etat14,M1_Etat15,M1
_Etat16,M1_Etat17,M1_Etat18,M1_Etat19);
   SIGNAL M1_EtatPresent, M1_EtatSuivant : M1_Etat;

   TYPE M2_Etat IS (M2_Etat1,M2_Etat2,M2_Etat3,M2_Etat4,M2_Etat5);
   SIGNAL M2_EtatPresent, M2_EtatSuivant : M2_Etat;

   TYPE M3_Etat IS (M3_Etat1,M3_Etat2);
   SIGNAL M3_EtatPresent, M3_EtatSuivant : M3_Etat;

   SIGNAL ap1Mem : STD_LOGIC;
   SIGNAL ap2Mem : STD_LOGIC;
   SIGNAL ap3Mem : STD_LOGIC;
   SIGNAL ap4Mem : STD_LOGIC;
   SIGNAL finApi : STD_LOGIC;
   SIGNAL s_Arret : STD_LOGIC;

   BEGIN

   -- DESCRIPTION DU BLOC F
   PROCESS (AP,ET,PortOuverte,ap1Mem,ap2Mem,ap3Mem,ap4Mem)
		BEGIN
		CASE M1_EtatPresent IS
			-- traitement des etages
			WHEN M1_Etat1 => IF ap2Mem='1' OR ap3Mem='1' OR ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat6;
					ELSIF ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat12;
					END IF;
			WHEN M1_Etat2 => IF ap3Mem='1' OR ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat8;
					ELSIF ap2Mem='1' THEN M1_EtatSuivant<=M1_Etat14;
					ELSIF ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat7;
					END IF;
			WHEN M1_Etat3 => IF ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat10;
					ELSIF ap3Mem='1' THEN M1_EtatSuivant<=M1_Etat16;
					ELSIF ap2Mem='1' OR ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat9;
					END IF;
			WHEN M1_Etat4 => IF ap4Mem='1' THEN M1_EtatSuivant<=M1_Etat18;
					ELSIF ap3Mem='1' OR ap2Mem='1' OR ap1Mem='1' THEN M1_EtatSuivant<=M1_Etat11;
					END IF;

			--traitement des transistion entre étages
			WHEN M1_Etat5 => IF ET(0)='1' THEN M1_EtatSuivant<=M1_Etat1; END IF;
			WHEN M1_Etat6 => IF ET(1)='1' THEN M1_EtatSuivant<=M1_Etat2; END IF;
			WHEN M1_Etat7 => IF ET(0)='1' THEN M1_EtatSuivant<=M1_Etat1; END IF;
			WHEN M1_Etat8 => IF ET(2)='1' THEN M1_EtatSuivant<=M1_Etat3; END IF;
			WHEN M1_Etat9 => IF ET(1)='1' THEN M1_EtatSuivant<=M1_Etat2; END IF;
			WHEN M1_Etat10 => IF ET(3)='1' THEN M1_EtatSuivant<=M1_Etat4; END IF;
			WHEN M1_Etat11 => IF ET(2)='1' THEN M1_EtatSuivant<=M1_Etat3; END IF;

			--traitement des portes
			WHEN M1_Etat12 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat13; END IF;  --etage1
			WHEN M1_Etat13 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat1; END IF;  --etage1
			WHEN M1_Etat14 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat15; END IF;  --etage2
			WHEN M1_Etat15 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat2; END IF;  --etage2
			WHEN M1_Etat16 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat17; END IF;  --etage3
			WHEN M1_Etat17 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat3; END IF;  --etage3
			WHEN M1_Etat18 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat19; END IF;  --etage4
			WHEN M1_Etat19 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat4; END IF;  --etage4
		END CASE;
   END PROCESS;


   PROCESS (AP,finApi)
		BEGIN
		CASE M2_EtatPresent IS
			WHEN M2_Etat1 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat2 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat3 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat4 => IF finApi='1' THEN M2_EtatSuivant<=M2_Etat5; END IF;
			WHEN M2_Etat5 =>
					IF AP(0)='1' THEN M2_EtatSuivant<=M2_Etat1;
					ELSIF AP(1)='1' THEN M2_EtatSuivant<=M2_Etat2;
					ELSIF AP(2)='1' THEN M2_EtatSuivant<=M2_Etat3;
					ELSIF AP(3)='1' THEN M2_EtatSuivant<=M2_Etat4;
					END IF;
		END CASE;
   END PROCESS;

   -- Process d'arret d'urgence
   PROCESS (E)
		BEGIN
		CASE M3_EtatPresent IS
			WHEN M3_Etat1 => IF E(1)='1' THEN M3_EtatSuivant<=M3_Etat2; END IF;
			WHEN M3_Etat2 => IF E(2)='1' THEN M3_EtatSuivant<=M3_Etat1; END IF;
		END CASE;
   END PROCESS;
```

```
    -- DESCRIPTION DU BLOC M
    PROCESS (Horloge,E)
     BEGIN
                -- passage à l'etat suivant
                IF ((Horloge'EVENT) AND (Horloge = '1')) THEN      M1_EtatPresent<=M1_EtatSuivant;
                                                                   M2_EtatPresent<=M2_EtatSuivant;
                                                                   M3_EtatPresent<=M3_EtatSuivant;
                END IF;

                -- initialisation des etats
                IF (E(0)='1') THEN        M1_EtatPresent<=M1_Etat5;
                                          M2_EtatPresent<=M2_Etat5;
                                          M3_EtatPresent<=M3_Etat1;
                END IF;

    END PROCESS;

    -- DESCRIPTION DU BLOC G
    -- sorties pour pilotage des afficheurs de la carte
    with M1_EtatPresent select
    unites      <=          zero        WHEN M1_Etat10,
                            un          WHEN M1_Etat1 | M1_Etat11,
                            deux        WHEN M1_Etat2 | M1_Etat12,
                            trois       WHEN M1_Etat3 | M1_Etat13,
                            quatre      WHEN M1_Etat4 | M1_Etat14,
                            cinq        WHEN M1_Etat5 | M1_Etat15,
                            six         WHEN M1_Etat6 | M1_Etat16,
                            sept        WHEN M1_Etat7 | M1_Etat17,
                            huit        WHEN M1_Etat8 | M1_Etat18,
                            neuf        WHEN M1_Etat9 | M1_Etat19,
                            erreur      WHEN OTHERS;
    dizaines    <=          eteint      WHEN (M1_EtatPresent < M1_Etat10) ELSE
                            un          WHEN (M1_EtatPresent >= M1_Etat10 AND M1_EtatPresent <= M1_Etat19)ELSE
                            erreur;

    -- sortie pour memorisation des poussoirs d'arret d'urgence
    s_Arret<='1' WHEN M3_EtatPresent=M3_Etat2          ELSE '0';

    -- sorties pour pilotage de la cabine
    montee<='1' WHEN (M1_EtatPresent=M1_Etat6 OR M1_EtatPresent=M1_Etat8 OR M1_EtatPresent=M1_Etat10) AND (s_Arret='0') ELSE '0';
    descend<='1' WHEN (M1_EtatPresent=M1_Etat7 OR M1_EtatPresent=M1_Etat9 OR M1_EtatPresent=M1_Etat11) AND (s_Arret='0') ELSE '0';

    -- sorties pour memorisation des poussoirs
    ap1Mem<='1' WHEN M2_EtatPresent=M2_Etat1           ELSE '0';
    ap2Mem<='1' WHEN M2_EtatPresent=M2_Etat2           ELSE '0';
    ap3Mem<='1' WHEN M2_EtatPresent=M2_Etat3           ELSE '0';
    ap4Mem<='1' WHEN M2_EtatPresent=M2_Etat4           ELSE '0';

    -- sortie pour mettre fin a ala memorisation du bouton poussoir interne
    WITH M1_EtatPresent select
    finApi      <=          '1'         WHEN M1_Etat13 | M1_Etat15 | M1_Etat17 | M1_Etat19,
                            '0'         WHEN OTHERS;


END ar;
```


Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

Annexe 3 : memorisation.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY memorisation IS
PORT(
		E                                   : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		ET                                  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		AP                                  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		P4d,P3d,P2d,P3m,P2m,P1m   : IN STD_LOGIC;
		PortOuverte               : IN STD_LOGIC;
		Horloge                             : IN STD_LOGIC;
		montee,descend                      : OUT STD_LOGIC;
		unites,dizaines                     : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END memorisation;

ARCHITECTURE ar OF memorisation IS

   CONSTANT zero   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000001";
   CONSTANT un     : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1001111";
   CONSTANT deux   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0010010";
   CONSTANT trois  : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000110";
   CONSTANT quatre : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1001100";
   CONSTANT cinq   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0100100";
   CONSTANT six    : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0100000";
   CONSTANT sept   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0001111";
   CONSTANT huit   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000000";
   CONSTANT neuf   : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0000100";
   CONSTANT erreur : STD_LOGIC_VECTOR (6 DOWNTO 0) := "0110000";
   CONSTANT eteint : STD_LOGIC_VECTOR (6 DOWNTO 0) := "1111111";

   TYPE M1_Etat IS
(M1_Etat1,M1_Etat2,M1_Etat3,M1_Etat4,M1_Etat5,M1_Etat6,M1_Etat7,M1_Etat8,M1_Etat9,M1_Etat10,M1_Etat11,M1_Etat12,M1_Etat13,M1_Etat14,M1_Etat15,M1
_Etat16,M1_Etat17,M1_Etat18,M1_Etat19);
   TYPE M2_Etat IS (M2_Etat1,M2_Etat2);
   TYPE M3_Etat IS (M3_Etat1,M3_Etat2);
   TYPE M4_Etat IS (M4_Etat1,M4_Etat2);
   TYPE M5_Etat IS (M5_Etat1,M5_Etat2);
   TYPE M6_Etat IS (M6_Etat1,M6_Etat2);

   SIGNAL M1_EtatPresent, M1_EtatSuivant : M1_Etat;
   SIGNAL M2_EtatPresent, M2_EtatSuivant : M2_Etat;
   SIGNAL M3_EtatPresent, M3_EtatSuivant : M3_Etat;
   SIGNAL M4_EtatPresent, M4_EtatSuivant : M4_Etat;
   SIGNAL M5_EtatPresent, M5_EtatSuivant : M5_Etat;
   SIGNAL M6_EtatPresent, M6_EtatSuivant : M6_Etat;

   SIGNAL Appel1,Appel2,Appel3,Appel4 : STD_LOGIC;
   SIGNAL finAppel1,finAppel2,finAppel3,finAppel4 : STD_LOGIC;
   SIGNAL s_Arret : STD_LOGIC;

   BEGIN

   -- DESCRIPTION DU BLOC F
   --- MAE1 pour la commande du déplacement de l'ascenseur et des portes
   PROCESS (AP,ET,PortOuverte,Appel1,Appel2,Appel3,Appel4)
		BEGIN
		CASE M1_EtatPresent IS
			-- traitement des etages
			WHEN M1_Etat1 => IF Appel2='1' OR Appel3='1' OR Appel4='1' THEN M1_EtatSuivant<=M1_Etat6;
				ELSIF Appel1='1' THEN M1_EtatSuivant<=M1_Etat12;
				END IF;
			WHEN M1_Etat2 => IF Appel3='1' OR Appel4='1' THEN M1_EtatSuivant<=M1_Etat8;
				ELSIF Appel2='1' THEN M1_EtatSuivant<=M1_Etat14;
				ELSIF Appel1='1' THEN M1_EtatSuivant<=M1_Etat7;
				END IF;
			WHEN M1_Etat3 => IF Appel4='1' THEN M1_EtatSuivant<=M1_Etat10;
				ELSIF Appel3='1' THEN M1_EtatSuivant<=M1_Etat16;
				ELSIF Appel2='1' OR Appel1='1' THEN M1_EtatSuivant<=M1_Etat9;
				END IF;
			WHEN M1_Etat4 => IF Appel4='1' THEN M1_EtatSuivant<=M1_Etat18;
				ELSIF Appel3='1' OR Appel2='1' OR Appel1='1' THEN M1_EtatSuivant<=M1_Etat11;
				END IF;

			--traitement des transistion entre étages
			WHEN M1_Etat5 => IF ET(0)='1' THEN M1_EtatSuivant<=M1_Etat1; END IF;
			WHEN M1_Etat6 => IF ET(1)='1' THEN M1_EtatSuivant<=M1_Etat2; END IF;
			WHEN M1_Etat7 => IF ET(0)='1' THEN M1_EtatSuivant<=M1_Etat1; END IF;
			WHEN M1_Etat8 => IF ET(2)='1' THEN M1_EtatSuivant<=M1_Etat3; END IF;
			WHEN M1_Etat9 => IF ET(1)='1' THEN M1_EtatSuivant<=M1_Etat2; END IF;
			WHEN M1_Etat10 => IF ET(3)='1' THEN M1_EtatSuivant<=M1_Etat4; END IF;
			WHEN M1_Etat11 => IF ET(2)='1' THEN M1_EtatSuivant<=M1_Etat3; END IF;

			--traitement des portes
			WHEN M1_Etat12 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat13; END IF;  --etage1
			WHEN M1_Etat13 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat1; END IF;   --etage1
			WHEN M1_Etat14 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat15; END IF;  --etage2
			WHEN M1_Etat15 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat2; END IF;   --etage2
			WHEN M1_Etat16 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat17; END IF;  --etage3
			WHEN M1_Etat17 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat3; END IF;   --etage3
			WHEN M1_Etat18 => IF PortOuverte='1' THEN M1_EtatSuivant<=M1_Etat19; END IF;  --etage4
			WHEN M1_Etat19 => IF PortOuverte='0' THEN M1_EtatSuivant<=M1_Etat4; END IF;   --etage4
		END CASE;
   END PROCESS;

   --- MAE2 d'arrêt d'urgence
   PROCESS (E)
		BEGIN
		CASE M2_EtatPresent IS
			WHEN M2_Etat1 => IF E(1)='1' THEN M2_EtatSuivant<=M2_Etat2; END IF;
			WHEN M2_Etat2 => IF E(2)='1' THEN M2_EtatSuivant<=M2_Etat1; END IF;
		END CASE;
   END PROCESS;

   --- MAE3 pour la mémorisation des appels internes ou externes de l'étage 1
   PROCESS (P1m,AP,finAppel1)
		BEGIN
		CASE M3_EtatPresent IS
			WHEN M3_Etat1 => IF P1m='1' OR AP(0)='1' THEN M3_EtatSuivant<=M3_Etat2; END IF;
			WHEN M3_Etat2 => IF finAppel1='1' THEN M3_EtatSuivant<=M3_Etat1; END IF;
		END CASE;
   END PROCESS;

   --- MAE4 pour la mémorisation des appels internes ou externes de l'étage 2
   PROCESS (P2m,P2d,AP,finAppel2)
		BEGIN
```

```
                CASE M4_EtatPresent IS
                        WHEN M4_Etat1 => IF P2m='1' OR P2d='1' OR AP(1)='1' THEN M4_EtatSuivant<=M4_Etat2; END IF;
                        WHEN M4_Etat2 => IF finAppel2='1' THEN M4_EtatSuivant<=M4_Etat1; END IF;
                END CASE;
    END PROCESS;

    --- MAE5 pour la mémorisation des appels internes ou externes de l'étage 3
    PROCESS (P3m,P3d,AP,finAppel3)
                BEGIN
                CASE M5_EtatPresent IS
                        WHEN M5_Etat1 => IF P3m='1' OR P3d='1' OR AP(2)='1' THEN M5_EtatSuivant<=M5_Etat2; END IF;
                        WHEN M5_Etat2 => IF finAppel3='1' THEN M5_EtatSuivant<=M5_Etat1; END IF;
                END CASE;
    END PROCESS;

    --- MAE6 pour la mémorisation des appels internes ou externes de l'étage 4
    PROCESS (P4d,AP,finAppel4)
                BEGIN
                CASE M6_EtatPresent IS
                        WHEN M6_Etat1 => IF P4d='1' OR AP(3)='1' THEN M6_EtatSuivant<=M6_Etat2; END IF;
                        WHEN M6_Etat2 => IF finAppel3='1' THEN M6_EtatSuivant<=M6_Etat1; END IF;
                END CASE;
    END PROCESS;

    -- DESCRIPTION DU BLOC M
    PROCESS (Horloge,E)
     BEGIN
                -- passage à l'etat suivant
                IF ((Horloge'EVENT) AND (Horloge = '1')) THEN
                        M1_EtatPresent <= M1_EtatSuivant;
                        M2_EtatPresent <= M2_EtatSuivant;
                        M3_EtatPresent <= M3_EtatSuivant;
                        M4_EtatPresent <= M4_EtatSuivant;
                        M5_EtatPresent <= M5_EtatSuivant;
                        M6_EtatPresent <= M6_EtatSuivant;
                END IF;

                -- initialisation des etats
                IF (E(0)='1') THEN
                        M1_EtatPresent<=M1_Etat5;
                        M2_EtatPresent<=M2_Etat1;
                        M3_EtatPresent<=M3_Etat1;
                        M4_EtatPresent<=M4_Etat1;
                        M5_EtatPresent<=M5_Etat1;
                        M6_EtatPresent<=M6_Etat1;
                END IF;
    END PROCESS;

    -- DESCRIPTION DU BLOC G
    -- sorties pour pilotage des afficheurs de la carte
    with M1_EtatPresent select
                unites <=    zero         WHEN M1_Etat10,
                             un           WHEN M1_Etat1 | M1_Etat11,
                             deux         WHEN M1_Etat2 | M1_Etat12,
                             trois        WHEN M1_Etat3 | M1_Etat13,
                             quatre       WHEN M1_Etat4 | M1_Etat14,
                             cinq         WHEN M1_Etat5 | M1_Etat15,
                             six          WHEN M1_Etat6 | M1_Etat16,
                             sept         WHEN M1_Etat7 | M1_Etat17,
                             huit         WHEN M1_Etat8 | M1_Etat18,
                             neuf         WHEN M1_Etat9 | M1_Etat19,
                             erreur       WHEN OTHERS;
    dizaines <=  eteint      WHEN (M1_EtatPresent < M1_Etat10) ELSE
                             un           WHEN (M1_EtatPresent >= M1_Etat10 AND M1_EtatPresent <= M1_Etat19)ELSE
                             erreur;

    -- memorisation des poussoirs d'arret d'urgence
    s_Arret<='1' WHEN M2_EtatPresent=M2_Etat2          ELSE '0';

    -- sorties pour pilotage de la cabine
    montee<='1' WHEN (M1_EtatPresent=M1_Etat6 OR M1_EtatPresent=M1_Etat8 OR M1_EtatPresent=M1_Etat10) AND (s_Arret='0') ELSE '0';
    descend<='1' WHEN (M1_EtatPresent=M1_Etat7 OR M1_EtatPresent=M1_Etat9 OR M1_EtatPresent=M1_Etat11) AND (s_Arret='0') ELSE '0';

    -- MaJ memorisation des poussoirs
    Appel1<='1' WHEN M3_EtatPresent=M3_Etat2           ELSE '0';
    Appel2<='1' WHEN M4_EtatPresent=M4_Etat2           ELSE '0';
    Appel3<='1' WHEN M5_EtatPresent=M5_Etat2           ELSE '0';
    Appel4<='1' WHEN M6_EtatPresent=M6_Etat2           ELSE '0';

    -- MaJ finAppel
    finAppel1<='1' WHEN M1_EtatPresent=M1_Etat13       ELSE '0';
    finAppel2<='1' WHEN M1_EtatPresent=M1_Etat15       ELSE '0';
    finAppel3<='1' WHEN M1_EtatPresent=M1_Etat17       ELSE '0';
    finAppel4<='1' WHEN M1_EtatPresent=M1_Etat19       ELSE '0';

END ar;
```


Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

Annexe 4 : Gestion de la zone de tri (BCI)

```
(*MAE1 : Gestion du tapis et de l'actionneur A1*)
CASE etatpresent1 OF
   1: IF (C2)               THEN etatsuivant1:=2;
      ELSIF (C1)            THEN etatsuivant1:=3;                END_IF;
   2: IF (C1)               THEN etatsuivant1:=4;                END_IF;
   3: IF (not C1)           THEN etatsuivant1:=1;                END_IF;
   4: IF (not C1)           THEN etatsuivant1:=1;                END_IF;
END_CASE;

(*MAE2 : Gestion de la tempo et du solénoïde rotatif*)
CASE etatpresent2 OF
   1: IF (demareTempoGoulotte)                      THEN etatsuivant2:=2;     END_IF;
   2: IF (finTempoGoulotte and (not C3))  THEN etatsuivant2:=3;     END_IF;
   3: IF (finTempoA2 and C3)              THEN etatsuivant2:=1;     END_IF;
END_CASE;

IF(init) THEN etatsuivant1:=1; etatsuivant2:=1; END_IF;
etatpresent1:=etatsuivant1;
etatpresent2:=etatsuivant2;

demareTempoGoulotte:=(etatpresent1=3);

A1:=(etatpresent1=3);
A2:=(etatpresent2=3);
A4:=1;

TempoGoulotte(IN:=((etatpresent2=2)),PT:=Time#1.5s,Q=>finTempoGoulotte,ET=>valeurTempoGoulotte);
TempoA2(IN:=((etatpresent2=3)),PT:=Time#0.5s,Q=>finTempoA2,ET=>valeurTempoA2);
```

Annexe 5 : Gestion de la zone d'assemblage et de l'éjection (BCI)

```
(*MAE1 : Gestion du tapis et de l'actionneur A1*)
CASE etatpresent1 OF
   1: IF (C2)   THEN etatsuivant1:=2;
      ELSIF (C1)            THEN etatsuivant1:=3;                END_IF;
   2: IF (C1)   THEN etatsuivant1:=4;                            END_IF;
   3: IF (not C1)           THEN etatsuivant1:=1;                END_IF;
   4: IF (not C1)           THEN etatsuivant1:=1;                END_IF;
END_CASE;

(*MAE2 : Gestion de la tempo et du solenoïde rotatif*)
CASE etatpresent2 OF
   1: IF (demareTempoGoulotte)                      THEN etatsuivant2:=2;     END_IF;
   2: IF (finTempoGoulotte and (not C3))  THEN etatsuivant2:=3;     END_IF;
   3: IF (finTempoA2 and C3)              THEN etatsuivant2:=1;     END_IF;
END_CASE;

(*MAE3 : Gestion de l'assemblage et de l'ejection des pieces*)
CASE etatpresent3 OF
   1: IF (C7)                           THEN etatsuivant3:=2;
      ELSIF (C4)                        THEN etatsuivant3:=4;     END_IF;
   2: IF (C4)                           THEN etatsuivant3:=3;     END_IF;
   3: IF ((not C8) and C5)   THEN etatsuivant3:=4;
      ELSIF (C8 and C5)                 THEN etatsuivant3:=1;     END_IF;
   4: IF (C6)                           THEN etatsuivant3:=5;     END_IF;
   5: IF (finTempoA3)                   THEN etatsuivant3:=1;     END_IF;
END_CASE;

IF(init) THEN etatsuivant1:=1; etatsuivant2:=1; etatsuivant3:=1; END_IF;
etatpresent1:=etatsuivant1;
etatpresent2:=etatsuivant2;
etatpresent3:=etatsuivant3;

demareTempoGoulotte:=(etatpresent1=3);

A1:=(etatpresent1=3);
A2:=(etatpresent2=3);
A3:=(etatpresent3=5);
A4:=1;
A5:=1;

TempoGoulotte(IN:=((etatpresent2=2)),PT:=Time#1.5s,Q=>finTempoGoulotte,ET=>valeurTempoGoulotte);
TempoA2(IN:=((etatpresent2=3)),PT:=Time#0.5s,Q=>finTempoA2,ET=>valeurTempoA2);
TempoA3(IN:=((etatpresent3=5)),PT:=Time#0.5s,Q=>finTempoA3,ET=>valeurTempoA3);
```


Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

Annexe 6 : Application complète (BCI)

```
(*MAE1 : Gestion du tapis et de l'actionneur A1*)
CASE etatpresent1 OF
   1: IF (C2)                                    THEN etatsuivant1:=2;
      ELSIF (C1)                                 THEN etatsuivant1:=3;     END_IF;
   2: IF (C1)                                    THEN etatsuivant1:=4;     END_IF;
   3: IF (not C1)                                THEN etatsuivant1:=5;     END_IF;
   4: IF (not C1)                                THEN etatsuivant1:=1;     END_IF;
   5: IF (C1)                                    THEN etatsuivant1:=6;
      ELSIF (C2)                                 THEN etatsuivant1:=13;
      ELSIF (A2Mem)                              THEN etatsuivant1:=12;    END_IF;
   6: IF (not C1)                                THEN etatsuivant1:=7;     END_IF;
   7: IF (C1)                                    THEN etatsuivant1:=8;
      ELSIF (C2)                                 THEN etatsuivant1:=15;
      ELSIF (A2Mem)                              THEN etatsuivant1:=11;    END_IF;
   8: IF (not C1)                                THEN etatsuivant1:=8;     END_IF;
   9: IF (C1)                                    THEN etatsuivant1:=9;
      ELSIF (C2)                                 THEN etatsuivant1:=17;
      ELSIF (A2Mem)                              THEN etatsuivant1:=10;    END_IF;
   10: IF (not A2Mem)                            THEN etatsuivant1:=7;     END_IF;
   11: IF (not A2Mem)                            THEN etatsuivant1:=5;     END_IF;
   12: IF (not A2Mem)                            THEN etatsuivant1:=1;     END_IF;
   13: IF (C1)                          THEN etatsuivant1:=14;    END_IF;
   14: IF (not C1)                               THEN etatsuivant1:=5;     END_IF;
   15: IF (C1)                          THEN etatsuivant1:=16;    END_IF;
   16: IF (not C1)                               THEN etatsuivant1:=7;     END_IF;
   17: IF (C1)                          THEN etatsuivant1:=18;    END_IF;
   18: IF (not C1)                               THEN etatsuivant1:=9;     END_IF;
END_CASE;

(*MAE2 : Gestion de la tempo et du solenoïde rotatif*)
CASE etatpresent2 OF
   1: IF (demareTempoGoulotte)                   THEN etatsuivant2:=2;     END_IF;
   2: IF (finTempoGoulotte and (not C3))  THEN etatsuivant2:=3;     END_IF;
   3: IF (finTempoA2 and C3)              THEN etatsuivant2:=1;     END_IF;
END_CASE;

(*MAE3 : Gestion de l'assemblage et de l'ejection des pieces*)
CASE etatpresent3 OF
   1: IF (C7)                                    THEN etatsuivant3:=2;
      ELSIF (C4)                                 THEN etatsuivant3:=4;                END_IF;
   2: IF (C4)                                    THEN etatsuivant3:=3;                END_IF;
   3: IF ((not C8) and C5)              THEN etatsuivant3:=4;
      ELSIF (C8 and C5)                          THEN etatsuivant3:=1;                END_IF;
   4: IF (C6)                                    THEN etatsuivant3:=5;                END_IF;
   5: IF (finTempoA3)                            THEN etatsuivant3:=1;                END_IF;
END_CASE;

IF(init) THEN etatsuivant1:=1; etatsuivant2:=1; etatsuivant3:=1; END_IF;
etatpresent1:=etatsuivant1;
etatpresent2:=etatsuivant2;
etatpresent3:=etatsuivant3;

demareTempoGoulotte:=((etatpresent1=3)or(etatpresent1=6)or(etatpresent1=8));
A2Mem:=(etatpresent2=3);

A1:=((etatpresent1=3)or(etatpresent1=6)or(etatpresent1=8));
A2:=(etatpresent2=3);
A3:=(etatpresent3=5);
A4:=1;
A5:=1;

TempoGoulotte(IN:=((etatpresent2=2)),PT:=Time#1.5s,Q=>finTempoGoulotte,ET=>valeurTempoGoulotte);
TempoA2(IN:=((etatpresent2=3)),PT:=Time#0.5s,Q=>finTempoA2,ET=>valeurTempoA2);
TempoA3(IN:=((etatpresent3=5)),PT:=Time#0.5s,Q=>finTempoA3,ET=>valeurTempoA3);
```


Alexis GIBERT // Nour GHILOUFI // UPSSITECH 1A SRI

Annexes 7 : Traitement dans le bac1 de trempage (STA)

```
#include <stdio.h>    // pour printf/scanf uniquement
#include <unistd.h>   // pour usleep();
#include <sta.h>      // definition des identifiants ES et des tempos entre autres

int main(void){

  //Déclarations des variables
  int EtatPresent = 1, EtatSuivant = 1;  // les variables d'état et initialisation
  S_Temporisateur TempoBac1;             // Temporisateur du temps de trempage
  S_Temporisateur TempoLim;              // Temporisateur du temps pour éviter la détection précoce des limites
  int finTempoBac1 = 0;
  int finTempoLim = 0;
  int ag, ad, c, lh, lv, p, op;          // les entrées
  int V, H, B, G, D, A;                  // les sorties

  // initialisation des ports
  init_io();
  InitSta();

while(!stop()) /*interrompre le programme par appui sur CTRL+C*/  {
    // lecture des entrées
    ag = entree(AG);
    ad = entree(AD);
    c  = entree(C);
    lh = entree(LH);
    lv = entree(LV);
    p  = entree(P);
    op = entree(OP);

    usleep(1000); /*allongement du cycle programme*/

    /************/
    /* Bloc F  */
    /************/
    switch(EtatPresent){
      case 1 :  if (lh,lv)              EtatSuivant = 9;          break;
      case 2 :  if (ad)                 EtatSuivant = 25;         break;
      case 3 :  if (ad)                 EtatSuivant = 20;
                else if (p)             EtatSuivant = 11;         break;
      case 4 :  if (ad)                 EtatSuivant = 21;
                else if (p)             EtatSuivant = 13;         break;
      case 5 :  if (ad)                 EtatSuivant = 26;
                else if (p)             EtatSuivant = 15;         break;
      case 6 :  if (ad)                 EtatSuivant = 22;
                else if (!p)            EtatSuivant = 17;         break;
      case 7 :  if (ad)                 EtatSuivant = 23;         break;
      case 8 :  if (ad)                 EtatSuivant = 24;         break;
      case 9 :  if (op)                 EtatSuivant = 2;          break;
      case 10 :                                                                 break;
      case 11 : if (c)                  EtatSuivant = 12;         break;
      case 12 : if (lv && finTempoLim)  EtatSuivant = 3;          break;
      case 13 : if (c)                  EtatSuivant = 14;         break;
      case 14 : if (lv && finTempoLim)  EtatSuivant = 4;          break;
      case 15 : if (c)                  EtatSuivant = 16;         break;
      case 16 : if (lv && finTempoLim)  EtatSuivant = 5;          break;
      case 17 : if (c)                  EtatSuivant = 18;         break;
      case 18 : if (lv && finTempoLim)              EtatSuivant = 19;         break;
      case 19 : if (finTempoBac1)       EtatSuivant = 1;          break;
      case 20 : if (ag)                 EtatSuivant = 4;          break;
      case 21 : if (ag)                 EtatSuivant = 5;          break;
      case 22 : if (ag)                 EtatSuivant = 7;          break;
      case 23 : if (ag)                 EtatSuivant = 8;          break;
      case 24 : if (ag)                 EtatSuivant = 10;         break;
      case 25 : if (ag)                 EtatSuivant = 3;          break;
      case 26 : if (ag)                 EtatSuivant = 6;          break;
     }

    /************/
    /* Bloc M  */
    /************/
    EtatPresent = EtatSuivant;

    printf("Etat Present : \t%d\n",EtatPresent); // Affichage pour débug

    /************/
    /* Bloc G   */
    /************/
    V = ((EtatPresent==1));
    A = 0;
    B = ((EtatPresent==1)||(EtatPresent==18));
    H = ((EtatPresent==12)||(EtatPresent==14)||(EtatPresent==16));
    D = ((EtatPresent==1)||(EtatPresent==11)||(EtatPresent==13)||(EtatPresent==15)||(EtatPresent==17));
    G = ((EtatPresent>=2 && EtatPresent<=8)||(EtatPresent==10)||(EtatPresent>=20 && EtatPresent<=26));

    printf("Capteurs    :\n\tlv\tlh\tag\tc\tp\n\t%d\t%d\t%d\t%d\t%d\t%d\n",lv,lh,ag,c,p); // Affichage pour débug
    printf("Actionneurs :\n\tH\tB\tG\tD\tA\n\t%d\t%d\t%d\t%d\t%d\t%d\n\n",H,B,G,D,A); // Affichage pour débug

    //Temporisateurs
    TempoBac1.activetempo = ((EtatPresent==19));
    TempoLim.activetempo = ((EtatPresent==12)||(EtatPresent==14)||(EtatPresent==16)||(EtatPresent==18));
    finTempoBac1 = temporisateur(8000,TempoBac1);
    finTempoLim = temporisateur(1000,TempoLim);

    /*****************************/
    /* Ecriture des sorties     */
    /*****************************/
    sortie(VV,V);
    sortie(HH,H);
    sortie(BB,B);
    sortie(GG,G);
    sortie(DD,D);
    sortie(AA,A);
  }
  mzSorties(); // coupe toutes les sorties avant d'arrêter la commande
  printf("\n\n*** Arrêt provoqué par l'utilisateur (CTRL-C) ***\n\n");
  return 0;
}
```

Annexes 8 : Traitement dans le bac2 (STA)

```
#include <stdio.h>    // pour printf/scanf uniquement
#include <unistd.h>   // pour usleep();
#include <sta.h>      // definition des identifiants ES et des tempos entre autres

int main(void){
  //Déclarations des variables
  int EtatPresent = 1, EtatSuivant = 1;  // les variables d'état et initialisation
  S_Temporisateur TempoBac2;             // Temporisateur du temps de trempage
  S_Temporisateur TempoLim;              // Temporisateur du temps pour éviter la détection précoce des limites
  int finTempoBac2 = 0;
  int finTempoLim = 0;
  int ag, ad, c, lh, lv, p, op;          // les entrées
  int V, H, B, G, D, A;                  // les sorties

  // initialisation des ports
  init_io();
  InitSta();

  while(!stop()) /*interrompre le programme par appui sur CTRL+C*/ {
    // lecture des entrées
    ag = entree(AG);
    ad = entree(AD);
    c  = entree(C);
    lh = entree(LH);
    lv = entree(LV);
    p  = entree(P);
    op = entree(OP);
    //printf("Capteurs :\n\tLAG\tC\tAD\tP\tLH\tLV\tOP\n\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",AG,C,AD,P,LH,LV,OP); // Affichage pour débug

    usleep(1000); /* allongement du cycle programme */

    /************/
    /* Bloc F  */
    /************/
    switch(EtatPresent){
      case 1 :  if (lh,lv)               EtatSuivant = 9;          break;
      case 2 :  if (ad)                  EtatSuivant = 25;         break;
      case 3 :  if (ad)                  EtatSuivant = 20;
                else if (p)              EtatSuivant = 11;         break;
      case 4 :  if (ad)                  EtatSuivant = 21;
                else if (p)              EtatSuivant = 13;         break;
      case 5 :  if (ad)                  EtatSuivant = 26;
                else if (p)              EtatSuivant = 15;         break;
      case 6 :  if (ad)                  EtatSuivant = 22;         break;
      case 7 :  if (ad)                  EtatSuivant = 23;
                else if (!p)             EtatSuivant = 17;         break;
      case 8 :  if (ad)                  EtatSuivant = 24;         break;
      case 9 :  if (op)                  EtatSuivant = 2;          break;
      case 10 :                                                    break;
      case 11 : if (c)                   EtatSuivant = 12;         break;
      case 12 : if (lv && finTempoLim)   EtatSuivant = 3;          break;
      case 13 : if (c)                   EtatSuivant = 14;         break;
      case 14 : if (lv && finTempoLim)   EtatSuivant = 4;          break;
      case 15 : if (c)                   EtatSuivant = 16;         break;
      case 16 : if (lv && finTempoLim)   EtatSuivant = 5;          break;
      case 17 : if (c)                   EtatSuivant = 18;         break;
      case 18 : if (lv && finTempoLim)   EtatSuivant = 19;         break;
      case 19 : if (finTempoBac2)        EtatSuivant = 1;          break;
      case 20 : if (ag)                  EtatSuivant = 4;          break;
      case 21 : if (ag)                  EtatSuivant = 5;          break;
      case 22 : if (ag)                  EtatSuivant = 7;          break;
      case 23 : if (ag)                  EtatSuivant = 8;          break;
      case 24 : if (ag)                  EtatSuivant = 10;         break;
      case 25 : if (ag)                  EtatSuivant = 3;          break;
      case 26 : if (ag)                  EtatSuivant = 6;          break;
     }

    /************/
    /* Bloc M  */
    /************/
    EtatPresent = EtatSuivant;
    printf("Etat Present : \t%d\n",EtatPresent); // Affichage pour débug

    /************/
    /* Bloc G   */
    /************/
    V = ((EtatPresent==1));
    A = 0;
    B = ((EtatPresent==1)||(EtatPresent==18));
    H = ((EtatPresent==12)||(EtatPresent==14)||(EtatPresent==16));
    D = ((EtatPresent==1)||(EtatPresent==11)||(EtatPresent==13)||(EtatPresent==15)||(EtatPresent==17));
    G = ((EtatPresent>=2 && EtatPresent<=8)||(EtatPresent==10)||(EtatPresent>=20 && EtatPresent<=26));

    printf("Capteurs    :\n\tlv\tlh\tag\tc\tp\n\t%d\t%d\t%d\t%d\t%d\n",lv,lh,ag,c,p); // Affichage pour débug
    printf("Actionneurs :\n\tH\tB\tG\tD\tA\n\t%d\t%d\t%d\t%d\t%d\n\n",H,B,G,D,A); // Affichage pour débug

    //Temporisateurs
    TempoBac2.activetempo = ((EtatPresent==19));
    TempoLim.activetempo = ((EtatPresent==12)||(EtatPresent==14)||(EtatPresent==16)||(EtatPresent==18));
    finTempoBac2 = temporisateur(8000,TempoBac2);
    finTempoLim = temporisateur(1000,TempoLim);

    /*****************************/
    /* Ecriture des sorties     */
    /*****************************/
    sortie(VV,V);
    sortie(HH,H);
    sortie(BB,B);
    sortie(GG,G);
    sortie(DD,D);
    sortie(AA,A);
  }
  mzSorties(); // coupe toutes les sorties avant d'arrêter la commande
  printf("\n\n*** Arrêt provoqué par l'utilisateur (CTRL-C) ***\n\n");
  return 0;
}
```

Annexes 9 : Gestion des transports pour les 2 bacs (STA)

```
#include <stdio.h>    // pour printf/scanf uniquement
#include <unistd.h>   // pour usleep();
#include <sta.h>      // definition des identifiants ES et des tempos entre autres

int main(void){
  /* Déclarations variables */
  // les variables d'état et initialisation
  int M1_EtatPresent = 1, M1_EtatSuivant = 1;         //MAE 1 : Transport et traitement bac1
  int M2_EtatPresent = 1, M2_EtatSuivant = 1;         //MAE 2 : Transport et traitement bac2
  int M3_EtatPresent = 1, M3_EtatSuivant = 1;         //MAE 3 : Demande de transport selon disponiblité des bacs
  int M4_EtatPresent = 1, M4_EtatSuivant = 1;         //MAE 4 : Allocation du système de transport

  // les variables internes
  int bac1Libre = 0,                    bac2Libre = 0;                        //Informe sur l'état de l'emplacement de trempage
  int demandeTransportBac1T = 0,        demandeTransportBac2T = 0;            //Transport vers un bac de Trempage
  int demandeTransportBac1S = 0,        demandeTransportBac2S = 0;            //Transport vers un bac de Sortie
  int piece1 = 0,                       piece2 = 0;
  int autorisationTransportBac1T = 0,   autorisationTransportBac2T = 0;
  int autorisationTransportBac1S = 0,   autorisationTransportBac2S = 0;
  int finTransportBac1T = 0,            finTransportBac2T = 0;
  S_Temporisateur TempoBac1,            TempoBac2;
  S_Temporisateur TempoLim;
  int finTempoBac1 = 0,                 finTempoBac2 = 0;
  int finTempoLim = 0;
  int ag, ad, c, lh, lv, p, op;                                               // les entrées
  int V, H, B, G, D, A;                                                       // les sorties

  // initialisation des ports
  init_io();
  InitSta();

  while(!stop()) /*interrompre le programme par appui sur CTRL+C*/ {
    // lecture des entrées
    ag = entree(AG);
    ad = entree(AD);
    c  = entree(C);
    lh = entree(LH);
    lv = entree(LV);
    p  = entree(P);
    op = entree(OP);
    //printf("Capteurs :\n\tLAG\tC\tAD\tP\tLH\tLV\tOP\n\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",AG,C,AD,P,LH,LV,OP); // Affichage pour débug

    usleep(1000); /* allongement du cycle programme */

    /************/
    /* Bloc F  */
    /************/
    //MAE 1 : Transport et traitement bac1
    switch(M1_EtatPresent){
        case 1 :  if (autorisationTransportBac1T)     M1_EtatSuivant = 2;       break;
        case 2 :  if (ad)                             M1_EtatSuivant = 3;       break;
        case 3 :  if (ag)                             M1_EtatSuivant = 4;       break;
        case 4 :  if (ad)                             M1_EtatSuivant = 5;
                  else if (p)                         M1_EtatSuivant = 12;      break;
        case 5 :  if (ag)                             M1_EtatSuivant = 6;       break;
        case 6 :  if (ad)                             M1_EtatSuivant = 7;
                  else if (p)                         M1_EtatSuivant = 14;      break;
        case 7 :  if (ag)                             M1_EtatSuivant = 8;       break;
        case 8 :  if (ad)                             M1_EtatSuivant = 9;
                  else if (p)                         M1_EtatSuivant = 16;
                  else if (ad && (!piece1))           M1_EtatSuivant = 11;      break;
        case 9 :  if (ag)                             M1_EtatSuivant = 10;      break;
        case 10 : if (ad)                             M1_EtatSuivant = 11;
                  else if(ag && (!p))                 M1_EtatSuivant = 18;      break;
        case 11 : if (finTempoLim)                    M1_EtatSuivant = 1;       break;
        case 12 : if (c)                              M1_EtatSuivant = 13;      break;
        case 13 : if (lv && finTempoLim)              M1_EtatSuivant = 4;       break;
        case 14 : if (c)                              M1_EtatSuivant = 15;      break;
        case 15 : if (lv && finTempoLim)              M1_EtatSuivant = 6;       break;
        case 16 : if (c)                              M1_EtatSuivant = 17;      break;
        case 17 : if (lv && finTempoLim)              M1_EtatSuivant = 8;       break;
        case 18 : if (c)                              M1_EtatSuivant = 19;      break;
        case 19 : if (lv && finTempoLim)              M1_EtatSuivant = 20;      break;
        case 20 : if (finTempoBac1)                   M1_EtatSuivant = 21;      break;
        case 21 : if (autorisationTransportBac1S)     M1_EtatSuivant = 1;       break;
        default :                                                               break;
    }

    //MAE 2 : Transport et traitement bac2
    switch(M2_EtatPresent){
        case 1 :  if (autorisationTransportBac2T)     M2_EtatSuivant = 2;     break;
        case 2 :  if (ad)                             M2_EtatSuivant = 3;     break;
        case 3 :  if (ag)                             M2_EtatSuivant = 4;     break;
        case 4 :  if (ad)                             M2_EtatSuivant = 5;
                  else if (p)                         M2_EtatSuivant = 12;    break;
        case 5 :  if (ag)                             M2_EtatSuivant = 6;     break;
        case 6 :  if (ad)                             M2_EtatSuivant = 7;
                  else if (p)                         M2_EtatSuivant = 14;    break;
        case 7 :  if (ag)                             M2_EtatSuivant = 8;     break;
        case 8 :  if (ad)                             M2_EtatSuivant = 9;
                  else if (p)                         M2_EtatSuivant = 16;
                  else if (ad && (!piece2))           M2_EtatSuivant = 11;    break;
        case 9 :  if (ag)                             M2_EtatSuivant = 10;    break;
        case 10 : if (ad)                             M2_EtatSuivant = 11;
                  else if(ag && (!p))                 M2_EtatSuivant = 18;    break;
        case 11 : if (ag)                             M2_EtatSuivant = 12;    break;
        case 12 : if (ad)                             M2_EtatSuivant = 13;
                  else if(ag && (!p))                 M2_EtatSuivant = 14;    break;
        case 13 : if (finTempoLim)                    M2_EtatSuivant = 1;     break;
        case 14 : if (c)                              M2_EtatSuivant = 15;    break;
        case 15 : if (lv && finTempoLim)              M2_EtatSuivant = 16;    break;
        case 16 : if (finTempoBac2)                   M2_EtatSuivant = 23;    break;
        case 17 : if (c)                              M2_EtatSuivant = 18;    break;
        case 18 : if (lv && finTempoLim)              M2_EtatSuivant = 4;     break;
        case 19 : if (c)                              M2_EtatSuivant = 20;    break;
        case 20 : if (lv && finTempoLim)              M2_EtatSuivant = 6;     break;
        case 21 : if (c)                              M2_EtatSuivant = 22;    break;
        case 22 : if (lv && finTempoLim)              M2_EtatSuivant = 8;     break;
        case 23 : if (autorisationTransportBac2S)     M2_EtatSuivant = 1;     break;
        default :                                                             break;
    }
```

```
    //MAE 3 : Demande de transport selon disponiblité des bacs
    switch(M3_EtatPresent){
        case 1 : if (op && bac1Libre)                M3_EtatSuivant = 2;
                 else if (op && (!bac1Libre))        M3_EtatSuivant = 3;       break;
        case 2 : if (autorisationTransportBac1T)     M3_EtatSuivant = 1;       break;
        case 3 : if (autorisationTransportBac2T)     M3_EtatSuivant = 1;       break;
        default :                                                              break;
    }

    //MAE 4 : Allocation du système de transport
    switch(M4_EtatPresent){
        case 1 : if (lh && lv && demandeTransportBac1T)          M3_EtatSuivant = 2;
                 else if (lh && lv && demandeTransportBac2T)     M3_EtatSuivant = 3;       break;
        case 2 : if (piece1)                                     M3_EtatSuivant = 4;
                 else if (finTransportBac1T)                     M3_EtatSuivant = 1;       break;
        case 3 : if (piece2)                                     M3_EtatSuivant = 5;
                 else if (finTransportBac2T)                     M3_EtatSuivant = 1;       break;
        case 4 : if (finTransportBac1T)                          M3_EtatSuivant = 1;       break;
        case 5 : if (finTransportBac2T)                          M3_EtatSuivant = 1;       break;
        default :                                                                          break;
    }

    /************/
    /* Bloc M  */
    /************/

    M1_EtatPresent = M1_EtatSuivant;    //MAE 1 : Transport et traitement bac1
    M2_EtatPresent = M2_EtatSuivant;    //MAE 2 : Transport et traitement bac2
    M3_EtatPresent = M3_EtatSuivant;    //MAE 3 : Demande de transport selon disponiblité des bacs
    M4_EtatPresent = M4_EtatSuivant;    //MAE 4 : Allocation du système de transport

    printf("M1_EtatPresent : %d\t M2_EtatPresent : %d\t M3_EtatPresent : %d\t M4_EtatPresent : %d\t
",M1_EtatPresent,M2_EtatPresent,M3_EtatPresent,M4_EtatPresent); // Affichage pour débug

    /************/
    /* Bloc G   */
    /************/

    //MAE 1 : Transport et traitement bac1
    A = ((M1_EtatPresent==11));
    B = ((M1_EtatPresent==19));
    H = ((M1_EtatPresent==13)||(M1_EtatPresent==15)||(M1_EtatPresent==17));
    D = ((M1_EtatPresent==12)||(M1_EtatPresent==14)||(M1_EtatPresent==16)||(M1_EtatPresent==18));
    G = ((M1_EtatPresent>=2)&&(M1_EtatPresent<=10));
    bac1Libre = ((M1_EtatPresent==1));
    finTransportBac1T = ((M1_EtatPresent==20)||(M1_EtatPresent==11));
    demandeTransportBac1S = ((M1_EtatPresent==21));

    //MAE 2 : Transport et traitement bac2
    A = ((M2_EtatPresent==13));
    B = ((M2_EtatPresent==15));
    H = ((M2_EtatPresent==18)||(M2_EtatPresent==20)||(M2_EtatPresent==22));
    D = ((M2_EtatPresent==17)||(M2_EtatPresent==19)||(M2_EtatPresent==21)||(M2_EtatPresent==14));
    G = ((M2_EtatPresent>=2)&&(M2_EtatPresent<=12));
    bac2Libre = ((M2_EtatPresent==1));
    finTransportBac2T = ((M2_EtatPresent==16)||(M2_EtatPresent==13));
    demandeTransportBac2S = ((M2_EtatPresent==23));

    //MAE 3 : Demande de transport selon disponiblité des bacs
    demandeTransportBac1T = ((M3_EtatPresent==2));
    demandeTransportBac2T = ((M3_EtatPresent==3));

    //MAE 4 : Allocation du système de transport
    D = ((M4_EtatPresent==1));
    B = ((M4_EtatPresent==1));
    V = ((M4_EtatPresent==1));
    autorisationTransportBac1T = ((M4_EtatPresent==2));
    autorisationTransportBac2T = ((M4_EtatPresent==3));
    piece1 = ((M4_EtatPresent==4));
    piece2 = ((M4_EtatPresent==5));

    //Temporisateurs
    TempoBac1.activetempo = ((M1_EtatPresent==20));
    TempoBac2.activetempo = ((M2_EtatPresent==16));
    TempoLim.activetempo = ((M1_EtatPresent==11)||(M1_EtatPresent==13)||(M1_EtatPresent==15)||(M1_EtatPresent==17)||(M1_EtatPresent==19)
                           ||(M2_EtatPresent==13)||(M2_EtatPresent==18)||(M2_EtatPresent==20)||(M2_EtatPresent==22)||(M2_EtatPresent==15));
    finTempoBac1 = temporisateur(8000,TempoBac1);
    finTempoBac2 = temporisateur(8000,TempoBac2);
    finTempoLim = temporisateur(1000,TempoLim);

    printf("Capteurs    :\n\tlv\tlh\tag\tc\tp\n\t%d\t%d\t%d\t%d\t%d\n",lv,lh,ag,c,p); // Affichage pour débug
    printf("Actionneurs :\n\tH\tB\tG\tD\tA\n\t%d\t%d\t%d\t%d\t%d\n\n",H,B,G,D,A); // Affichage pour débug

    /*****************************/
    /* Ecriture des sorties     */
    /*****************************/
    sortie(VV,V);
    sortie(HH,H);
    sortie(BB,B);
    sortie(GG,G);
    sortie(DD,D);
    sortie(AA,A);
  }
  mzSorties(); // coupe toutes les sorties avant d'arrêter la commande
  printf("\n\n*** Arrêt provoqué par l'utilisateur (CTRL-C) ***\n\n");
  return 0;
}
```

Annexes 10 : Application complète (STA)

```
#include <stdio.h>    // pour printf/scanf uniquement
#include <unistd.h>   // pour usleep();
#include <sta.h>      // definition des identifiants ES et des tempos entre autres

int main(void) {
  /* Déclarations variables */
  // les variables d'état et initialisation
  int M1_EtatPresent = 1;
  int M1_EtatSuivant = 1;
  int M2_EtatPresent = 1;
  int M2_EtatSuivant = 1;
  int M3_EtatPresent = 1;
  int M3_EtatSuivant = 1;
  int M4_EtatPresent = 1;
  int M4_EtatSuivant = 1;
  int M5_EtatPresent = 1;
  int M5_EtatSuivant = 1;
  int M6_EtatPresent = 1;
  int M6_EtatSuivant = 1;

  // les variables internes
  int bac1Libre = 0;
  int bac2Libre = 0;
  int demandeTransportBac1T = 0;
  int demandeTransportBac2T = 0;
  int demandeTransportBac1S = 0;
  int demandeTransportBac2S = 0;
  int piece1 = 0;
  int piece2 = 0;
  int autorisationTransportBac1T = 0;
  int autorisationTransportBac2T = 0;
  int autorisationTransportBac1S = 0;
  int autorisationTransportBac2S = 0;
  int finTransportBac1T = 0;
  int finTransportBac2T = 0;
  int finTransportBac1S = 0;
  int finTransportBac2S = 0;
  S_Temporisateur TempoBac1;
  S_Temporisateur TempoBac2;
  S_Temporisateur TempoLim;
  int finTempoBac1 = 0;
  int finTempoBac2 = 0;
  int finTempoLim = 0;

  int ag, ad, c, lh, lv, p, op;          // les entrées
  int V, H, B, G, D, A;                  // les sorties

  // initialisation des ports
  init_io();
  InitSta();

  while(!stop()) /*interrompre le programme par appui sur CTRL+C*/ {
    // lecture des entrées
    ag = entree(AG);
    ad = entree(AD);
    c  = entree(C);
    lh = entree(LH);
    lv = entree(LV);
    p  = entree(P);
    op = entree(OP);
    //printf("Capteurs :\n\tLAG\tC\tAD\tP\tLH\tLV\tOP\n\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",AG,C,AD,P,LH,LV,OP); // Affichage pour débug

    usleep(1000); /* allongement du cycle programme */

    /************/
    /* Bloc F  */
    /************/
    //MAE 1 : Transport et traitement bac1
    switch(M1_EtatPresent){
        case 1 :  if (autorisationTransportBac1T)     M1_EtatSuivant = 2;       break;
        case 2 :  if (ad)                             M1_EtatSuivant = 3;       break;
        case 3 :  if (ag)                             M1_EtatSuivant = 4;       break;
        case 4 :  if (ad)                             M1_EtatSuivant = 5;
                  else if (p)                         M1_EtatSuivant = 12;      break;
        case 5 :  if (ag)                             M1_EtatSuivant = 6;       break;
        case 6 :  if (ad)                             M1_EtatSuivant = 7;
                  else if (p)                         M1_EtatSuivant = 14;      break;
        case 7 :  if (ag)                             M1_EtatSuivant = 8;       break;
        case 8 :  if (ad)                             M1_EtatSuivant = 9;
                  else if (p)                         M1_EtatSuivant = 16;
                  else if (ad && (!piece1))           M1_EtatSuivant = 11;      break;
        case 9 :  if (ag)                             M1_EtatSuivant = 10;      break;
        case 10 : if (ad)                             M1_EtatSuivant = 11;
                  else if(ag && (!p))                 M1_EtatSuivant = 18;      break;
        case 11 : if (finTempoLim)                    M1_EtatSuivant = 1;       break;
        case 12 : if (c)                              M1_EtatSuivant = 13;      break;
        case 13 : if (lv && finTempoLim)              M1_EtatSuivant = 4;       break;
        case 14 : if (c)                              M1_EtatSuivant = 15;      break;
        case 15 : if (lv && finTempoLim)              M1_EtatSuivant = 6;       break;
        case 16 : if (c)                              M1_EtatSuivant = 17;      break;
        case 17 : if (lv && finTempoLim)              M1_EtatSuivant = 8;       break;
        case 18 : if (c)                              M1_EtatSuivant = 19;      break;
        case 19 : if (lv && finTempoLim)              M1_EtatSuivant = 20;      break;
        case 20 : if (finTempoBac1)                   M1_EtatSuivant = 21;      break;
        case 21 : if (autorisationTransportBac1S)     M1_EtatSuivant = 1;       break;
        default :                                                               break;
    }

    //MAE 2 : Transport et traitement bac2
    switch(M2_EtatPresent){
        case 1 :  if (autorisationTransportBac2T)     M2_EtatSuivant = 2;       break;
        case 2 :  if (ad)                             M2_EtatSuivant = 3;       break;
        case 3 :  if (ag)                             M2_EtatSuivant = 4;       break;
        case 4 :  if (ad)                             M2_EtatSuivant = 5;
                  else if (p)                         M2_EtatSuivant = 12;      break;
        case 5 :  if (ag)                             M2_EtatSuivant = 6;       break;
        case 6 :  if (ad)                             M2_EtatSuivant = 7;
                  else if (p)                         M2_EtatSuivant = 14;      break;
        case 7 :  if (ag)                             M2_EtatSuivant = 8;       break;
        case 8 :  if (ad)                             M2_EtatSuivant = 9;
                  else if (p)                         M2_EtatSuivant = 16;
                  else if (ad && (!piece2))           M2_EtatSuivant = 11;      break;
        case 9 :  if (ag)                             M2_EtatSuivant = 10;      break;
        case 10 : if (ad)                             M2_EtatSuivant = 11;
                  else if(ag && (!p))                 M2_EtatSuivant = 18;      break;
        case 11 : if (ag)                             M2_EtatSuivant = 12;      break;
```

```
        case 12 : if (ad)                           M2_EtatSuivant = 13;
                  else if(ag && (!p))               M2_EtatSuivant = 14;      break;
        case 13 : if (finTempoLim)                  M2_EtatSuivant = 1;       break;
        case 14 : if (c)                            M2_EtatSuivant = 15;      break;
        case 15 : if (lv && finTempoLim)            M2_EtatSuivant = 16;      break;
        case 16 : if (finTempoBac2)                 M2_EtatSuivant = 23;      break;
        case 17 : if (c)                            M2_EtatSuivant = 18;      break;
        case 18 : if (lv && finTempoLim)            M2_EtatSuivant = 4;       break;
        case 19 : if (c)                            M2_EtatSuivant = 20;      break;
        case 20 : if (lv && finTempoLim)            M2_EtatSuivant = 6;       break;
        case 21 : if (c)                            M2_EtatSuivant = 22;      break;
        case 22 : if (lv && finTempoLim)            M2_EtatSuivant = 8;       break;
        case 23 : if (autorisationTransportBac2S)   M2_EtatSuivant = 1;       break;
        default :                                                             break;
    }

    //MAE 3 : Demande de transport selon disponiblité des bacs
    switch(M3_EtatPresent){
        case 1 : if (op && bac1Libre)               M3_EtatSuivant = 2;
                 else if (op && (!bac1Libre))       M3_EtatSuivant = 3;       break;
        case 2 : if (autorisationTransportBac1T)    M3_EtatSuivant = 1;       break;
        case 3 : if (autorisationTransportBac2T)    M3_EtatSuivant = 1;       break;
        default :                                                             break;
    }

    //MAE 4 : Allocation du système de transport
    switch(M4_EtatPresent){
        case 1 : if (lh && lv && demandeTransportBac1T)     M3_EtatSuivant = 2;
                 else if (lh && lv && demandeTransportBac2T) M3_EtatSuivant = 3; break;
        case 2 : if (piece1)                                M3_EtatSuivant = 4;
                 else if (finTransportBac1T)                M3_EtatSuivant = 1; break;
        case 3 : if (piece2)                                M3_EtatSuivant = 5;
                 else if (finTransportBac2T)                M3_EtatSuivant = 1; break;
        case 4 : if (finTransportBac1T)                     M3_EtatSuivant = 1; break;
        case 5 : if (finTransportBac2T)                     M3_EtatSuivant = 1; break;
        default :                                                               break;
    }

    //MAE 5 : Transport vers bac1 de sortie
    switch(M5_EtatPresent){
        case 1 :  if (autorisationTransportBac1S)   M5_EtatPresent = 2;     break;
        case 2 :  if (ad)                           M5_EtatPresent = 3;     break;
        case 3 :  if (ag)                           M5_EtatPresent = 4;     break;
        case 4 :  if (ad)                           M5_EtatPresent = 5;     break;
        case 5 :  if (ag)                           M5_EtatPresent = 6;     break;
        case 6 :  if (ad)                           M5_EtatPresent = 7;     break;
        case 7 :  if (ag)                           M5_EtatPresent = 8;     break;
        case 8 :  if (ad)                           M5_EtatPresent = 9;     break;
        case 9 :  if (ag)                           M5_EtatPresent = 10;    break;
        case 10 :  if (ad)                          M5_EtatPresent = 11;
                  else if (p)                       M5_EtatPresent = 21;    break;
        case 11 :  if (ag)                          M5_EtatPresent = 12;    break;
        case 12 :  if (ad)                          M5_EtatPresent = 13;    break;
        case 13 :  if (ag)                          M5_EtatPresent = 14;    break;
        case 14 :  if (ad)                          M5_EtatPresent = 15;    break;
        case 15 :  if (ag)                          M5_EtatPresent = 16;    break;
        case 16 :  if (ad)                          M5_EtatPresent = 17;
                   else if (ag && (!p))             M5_EtatPresent = 22;    break;
        case 17 :  if (ag)                          M5_EtatPresent = 18;    break;
        case 18 :  if (ad)                          M5_EtatPresent = 19;    break;
        case 19 :  if (ag)                          M5_EtatPresent = 1;     break;
        case 20 :  if (lv && finTempoLim)           M5_EtatPresent = 10;    break;
        case 21 :  if (c)                           M5_EtatPresent = 20;    break;
        case 22 :  if (c)                           M5_EtatPresent = 23;    break;
        case 23 :  if (lv && finTempoLim)           M5_EtatPresent = 24;    break;
        case 24 :                                   M5_EtatPresent = 1;     break;
    }

    //MAE 6 : Transport vers bac2 de sortie
    switch(M6_EtatPresent){
        case 1 :  if (autorisationTransportBac2S)   M6_EtatPresent = 2;     break;
        case 2 :  if (ad)                           M6_EtatPresent = 3;     break;
        case 3 :  if (ag)                           M6_EtatPresent = 4;     break;
        case 4 :  if (ad)                           M6_EtatPresent = 5;     break;
        case 5 :  if (ag)                           M6_EtatPresent = 6;     break;
        case 6 :  if (ad)                           M6_EtatPresent = 7;     break;
        case 7 :  if (ag)                           M6_EtatPresent = 8;     break;
        case 8 :  if (ad)                           M6_EtatPresent = 9;     break;
        case 9 :  if (ag)                           M6_EtatPresent = 10;    break;
        case 10 :  if (ad)                          M6_EtatPresent = 11;    break;
        case 11 :  if (ag)                          M6_EtatPresent = 12;    break;
        case 12 :  if (ad)                          M6_EtatPresent = 13;
                  else if (p)                       M6_EtatPresent = 20;    break;
        case 13 :  if (ag)                          M6_EtatPresent = 14;    break;
        case 14 :  if (ad)                          M6_EtatPresent = 15;    break;
        case 15 :  if (ag)                          M6_EtatPresent = 16;    break;
        case 16 :  if (ad)                          M6_EtatPresent = 17;    break;
        case 17 :  if (ag)                          M6_EtatPresent = 18;    break;
        case 18 :  if (ad)                          M6_EtatPresent = 19;
                  else if (ag && (!p))              M6_EtatPresent = 22;    break;
        case 19 :  if (ag)                          M6_EtatPresent = 1;     break;
        case 20 :  if (c)                           M6_EtatPresent = 21;    break;
        case 21 :  if (lv && finTempoLim)           M6_EtatPresent = 12;    break;
        case 22 :  if (c)                           M6_EtatPresent = 23;    break;
        case 23 :  if (lv && finTempoLim)           M6_EtatPresent = 24;    break;
        case 24 :                                   M6_EtatPresent = 1;     break;
    }

    /************/
    /* Bloc M  */
    /************/

    M1_EtatPresent = M1_EtatSuivant;    //MAE 1 : Transport et traitement bac1
    M2_EtatPresent = M2_EtatSuivant;    //MAE 2 : Transport et traitement bac2
    M3_EtatPresent = M3_EtatSuivant;    //MAE 3 : Demande de transport selon disponiblité des bacs
    M4_EtatPresent = M4_EtatSuivant;    //MAE 4 : Allocation du système de transport
    M5_EtatPresent = M5_EtatSuivant;    //MAE 5 : Transport vers bac1 de sortie
    M6_EtatPresent = M6_EtatSuivant;    //MAE 6 : Transport vers bac2 de sortie

    printf("M1_EtatPresent : %d\t M2_EtatPresent : %d\t M3_EtatPresent : %d\t M4_EtatPresent : %d\t
",M1_EtatPresent,M2_EtatPresent,M3_EtatPresent,M4_EtatPresent); // Affichage pour débug
```

```
    /************/
    /* Bloc G   */
    /************/

    //MAE 1 : Transport et traitement bac1
    A = ((M1_EtatPresent==11));
    B = ((M1_EtatPresent==19));
    H = ((M1_EtatPresent==13)||(M1_EtatPresent==15)||(M1_EtatPresent==17));
    D = ((M1_EtatPresent==12)||(M1_EtatPresent==14)||(M1_EtatPresent==16)||(M1_EtatPresent==18));
    G = ((M1_EtatPresent>=2)&&(M1_EtatPresent<=10));
    bac1Libre = ((M1_EtatPresent==1));
    finTransportBac1T = ((M1_EtatPresent==20)||(M1_EtatPresent==11));
    demandeTransportBac1S = ((M1_EtatPresent==21));

    //MAE 2 : Transport et traitement bac2
    A = ((M2_EtatPresent==13));
    B = ((M2_EtatPresent==15));
    H = ((M2_EtatPresent==18)||(M2_EtatPresent==20)||(M2_EtatPresent==22));
    D = ((M2_EtatPresent==17)||(M2_EtatPresent==19)||(M2_EtatPresent==21)||(M2_EtatPresent==14));
    G = ((M2_EtatPresent>=2)&&(M2_EtatPresent<=12));
    bac2Libre = ((M2_EtatPresent==1));
    finTransportBac2T = ((M2_EtatPresent==16)||(M2_EtatPresent==13));
    demandeTransportBac2S = ((M2_EtatPresent==23));

    //MAE 3 : Demande de transport selon disponiblité des bacs
    demandeTransportBac1T = ((M3_EtatPresent==2));
    demandeTransportBac2T = ((M3_EtatPresent==3));

    //MAE 4 : Allocation du système de transport
    D = ((M4_EtatPresent==1));
    B = ((M4_EtatPresent==1));
    V = ((M4_EtatPresent==1));
    autorisationTransportBac1T = ((M4_EtatPresent==2));
    autorisationTransportBac2T = ((M4_EtatPresent==3));
    piece1 = ((M4_EtatPresent==4));
    piece2 = ((M4_EtatPresent==5));
    autorisationTransportBac1S = ((M4_EtatPresent==6));
    autorisationTransportBac2S = ((M4_EtatPresent==7));

    //MAE 5 : Transport vers bac1 de sortie
    A = ((M5_EtatPresent==17));
    B = ((M5_EtatPresent==23));
    H = ((M5_EtatPresent==20));
    D = ((M5_EtatPresent==21)||(M5_EtatPresent==22));
    G = ((M5_EtatPresent>=2)&&(M5_EtatPresent<=16));
    finTransportBac1S = ((M5_EtatPresent==17)||(M5_EtatPresent==24));

    //MAE 6 : Transport vers bac2 de sortie
    A = ((M6_EtatPresent==19));
    B = ((M6_EtatPresent==23));
    H = ((M6_EtatPresent==21));
    D = ((M6_EtatPresent==20)||(M6_EtatPresent==22));
    G = ((M6_EtatPresent>=2)&&(M6_EtatPresent<=18));
    finTransportBac2S = ((M6_EtatPresent==19)||(M6_EtatPresent==24));

    //Temporisateurs
    TempoBac1.activetempo = ((M1_EtatPresent==20));
    TempoBac2.activetempo = ((M2_EtatPresent==16));
    TempoLim.activetempo = ((M1_EtatPresent==11)||(M1_EtatPresent==13)||(M1_EtatPresent==15)||(M1_EtatPresent==17)||(M1_EtatPresent==19)
                           ||(M2_EtatPresent==13)||(M2_EtatPresent==18)||(M2_EtatPresent==20)||(M2_EtatPresent==22)||(M2_EtatPresent==15)
                           ||(M5_EtatPresent==20)||(M5_EtatPresent==17)||(M5_EtatPresent==23)
                           ||(M6_EtatPresent==21)||(M6_EtatPresent==19)||(M6_EtatPresent==23));
    finTempoBac1 = temporisateur(8000,TempoBac1);
    finTempoBac2 = temporisateur(8000,TempoBac2);
    finTempoLim = temporisateur(1000,TempoLim);

    printf("Capteurs    :\n\tlv\tlh\tag\tc\tp\n\t%d\t%d\t%d\t%d\t%d\t%d\n",lv,lh,ag,c,p); // Affichage pour débug
    printf("Actionneurs :\n\tH\tB\tG\tD\tA\n\t%d\t%d\t%d\t%d\t%d\t%d\n\n",H,B,G,D,A); // Affichage pour débug

    /*****************************/
    /* Ecriture des sorties      */
    /*****************************/
    sortie(VV,V);
    sortie(HH,H);
    sortie(BB,B);
    sortie(GG,G);
    sortie(DD,D);
    sortie(AA,A);
  }

  mzSorties(); // coupe toutes les sorties avant d'arrêter la commande
  printf("\n\n*** Arrêt provoqué par l'utilisateur (CTRL-C) ***\n\n");
  return 0;
}
```