Département Electronique, Energie Electrique, Automatique
1ère année de Master ISTR
Ingénierie des Systèmes Temps Réel

Commande d'une maquette de Machine à Laver Modélisation par MEF-mise en œuvre 1 parmi n en VHDL

Etudiants :
RAHMOUN Lokmane
SEGHIER Aissa

Encadrant :
P. ESTEBAN

Module : Techniques de Mise en Œuvre pour les Systèmes à Evénements Discrets commande
Année 2023/2024

Contacts : lokmane.rahmoun@univ-tsle.fr, aissa.seghier@univ-tlse3.fr

Table des matières

Table des figures

Chapitre 1

Introduction

L'application des techniques de modélisation conduit à l'obtention d'un modèle de comportement de l'application décrite par son cahier de charges. La vérification de ce modèle permet de constater son comportement se traduit par celui attendu et le décrit correctement. Des retours sur la modélisation peut être necessaires afin d'ajuster la représentation du comportement attendu.
L'étape de la mise en oeuvre va donner une réalité au modèle obtenu résultant du travail de conception et d'analyse et la commande du procédé va pouvoir être réalisée.

1.1 Objectifs

L'objectif est de développer une application écrite en C produisant un fichier écrit dans le langage cible (VHDL) matérialisant le modèle de commande du procédé (MEF) suivant la technique de mise en œuvre (codage 1 parmi n).

1.2 Contexte de la manipulation

La maquette de machine à laver présente un ensemble de boutons de commande : un bouton d'acceptation (Accept), un d'annulation (Cancel) et 3 choix de programmes (Program Selectors, Prog1 à Prog3). Un capteur détecte l'ouverture de la porte (Door Open/Close : 1=ouverte). Le moteur du tambour peut être commandé (Moteur : 1=marche) en indiquant dans quel sens il doit tourner (Sens : 0 sens horaire, 1 sens trigo). Une alarme peut être émise (Alarme), et un chiffre peut être écrit sur un afficheur 7 segments (quadruplet Aff3, Aff2, Aff1, Aff0). La maquette requiert un signal de contrôle (Validation : à maintenir à 0).
Bilan de ces signaux :

Progi,$i \in [1,3]$	Accept	Cancel	Door	
Affj, $j \in [0,3]$	Moteur	Sens	Alarme	Validation

Dans notre travail, nous ne nous intéressons pas à l'affichage ; nous concentrons uniquement sur les autres fonctionnalités : (sens,alarme,moteur,door,accept,cancel)

1.3 Structure de données du RdP benchmark

Lors de l'exécution de notre code VHDL généré dans le logiciel Quartus, nous avons rencontré des erreurs de syntaxe dues à la manière dont les activations et les désactivations des capteurs PO et Accept ont été écrites dans le fichier ndr. Pour cela, au lieu d'écrire 'Accept = '1", nous

avions écrit 'Accept', et dans le cas contraire, 'Accept = '0" était devenu 'not accept'. Après avoir identifié ce problème, nous l'avons corrigé dans le RDP suivant :

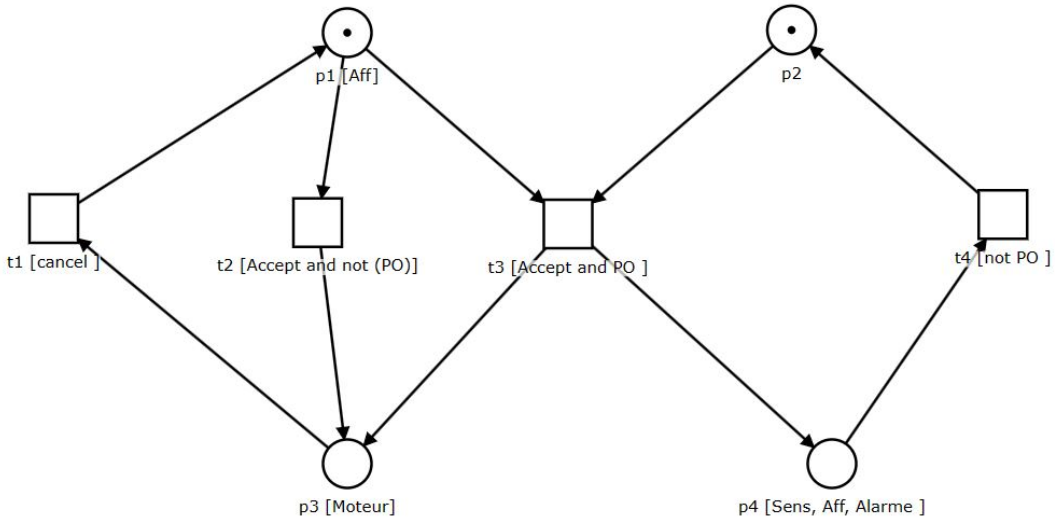


FIGURE 1.1 – Réseau de Petri de la machine à laver.

Pour élaborer notre algorithme basé sur le réseau de Petri précédent en langage C, l'utilisation de listes chaînées est nécessaire. Dans une liste chaînée, chaque élément pointe vers le suivant via un attribut appelé "suivant". Le dernier élément de la liste a un attribut "suivant" qui pointe vers NULL, indiquant ainsi la fin de la liste. Pour manipuler une liste chaînée, nous utilisons un simple pointeur vers le premier élément. Puisque chaque élément contient un pointeur vers l'élément suivant, nous pouvons parcourir tous les éléments de la liste en partant du premier. Cela offre une flexibilité dans la manipulation des données, car nous pouvons ajouter, supprimer ou modifier des éléments sans avoir besoin de réallouer un bloc de mémoire fixe.
Le fichier "types.c" contient les déclarations des différentes structures de listes chaînées telles que présentées dans la figure suivante :

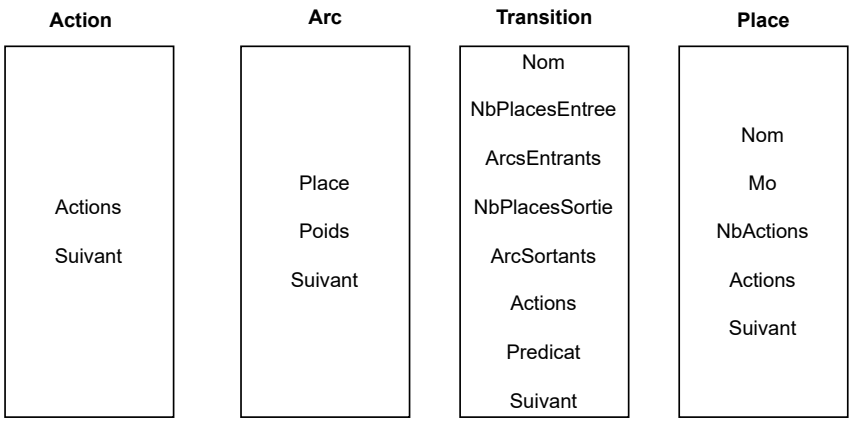


FIGURE 1.2 – Définition des structures pour le RDP.

Maintenant nous allons expliquer les éléments inclus dans la définition des structures :

PLACE

Cette structure représente une place dans un RDP. Elle contient les champs suivants :

— **NOM** : un string qui représente le nom de la place.
— **Mo** : un entier qui représente le marquage de la place.
— **NbActions** : un entier qui représente le nombre des actions associées à la place.
— **Actions** : un pointeur de type **ACTION**, qui pointe vers la première action de place (NULL s'il n'existe pas d'action associée).
— **Suivant** : un pointeur de type **PLACE**, il pointe vers la place suivante, utilisé pour construire une liste chaînée de places.

TRANSITION

Cette structure représente une transition dans un RDP. Elle contient les champs suivants :

— **Nom** : un string qui représente le nom de la transition.
— **NbPlacesEntree** : un entier qui représente le nombre de places en amont de la transition.
— **ArcsEntrants** : un pointeur de type **ARC** qui pointe vers le premier ARC de la liste chaînée des arcs entrants de la transition.
— **NbPlacesSortie** : un entier qui représente le nombre de places en aval de la transition.
— **ArcsSortants** : un pointeur de type **ARC** qui pointe vers le premier ARC de la liste chaînée des arcs sortants de la transition.
— **Actions** : un string qui représente le nom de l'action associée à la transition.
— **Predicat** : un string qui représente le nom du prédicat associé à la transition.

ARC

Cette structure représente un arc dans un RDP. Elle contient les champs suivants :

— **Place** : un string qui représente le nom de la place dont l'arc est associé.
— **Poids** : un entier qui représente le poids associé à l'arc.
— **Suivant** : un pointeur de type **ARC** qui pointe vers le prochain arc associé à la place.

ACTION

Cette structure représente une action dans un RDP. Elle contient les champs suivants :

— **Actions** : un string qui représente le nom de l'action.
— **Suivant** : un pointeur de type **Action** qui pointe vers la prochaine action associée à la place.

Toutes ces structures seront utilisées dans le code CreerStructure pour transformer la structure du RDP sous TINA en une structure manipulable en C. Dans la section qui suit, nous allons explorer cette structure pour vérifier si elle représente bien le RDP de la benchmark choisie.

En se basant sur le RDP benchmark et en suivant la logique des éléments définis dans les structures "place","transition","action" et "arc", nous pouvons schématiser le fonctionnement des listes chaînées à l'aide des deux schémas suivants.

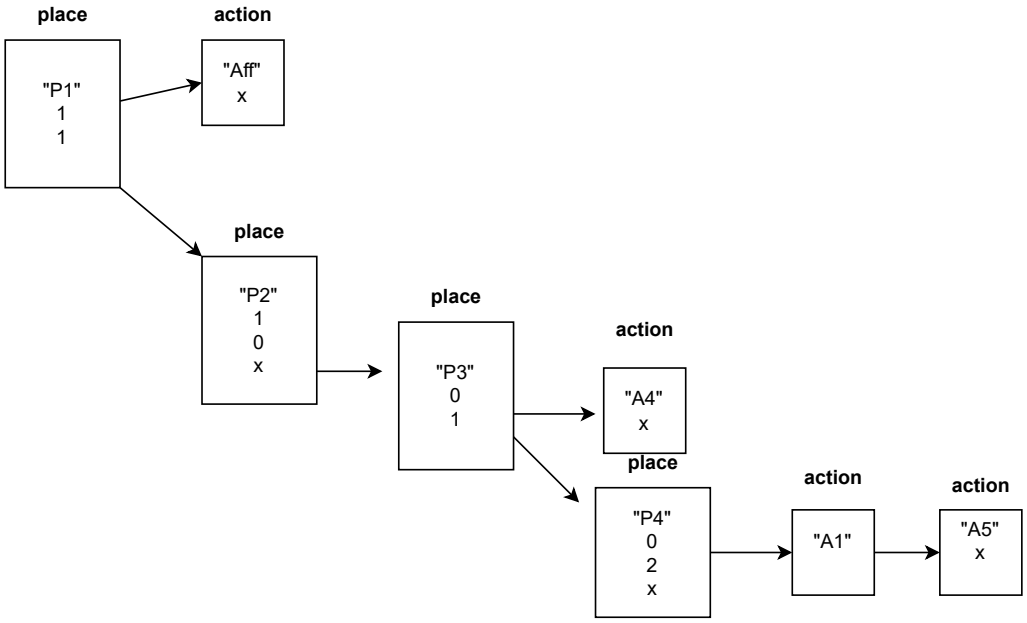


FIGURE 1.3 – Listes chaînées pour place et arc.

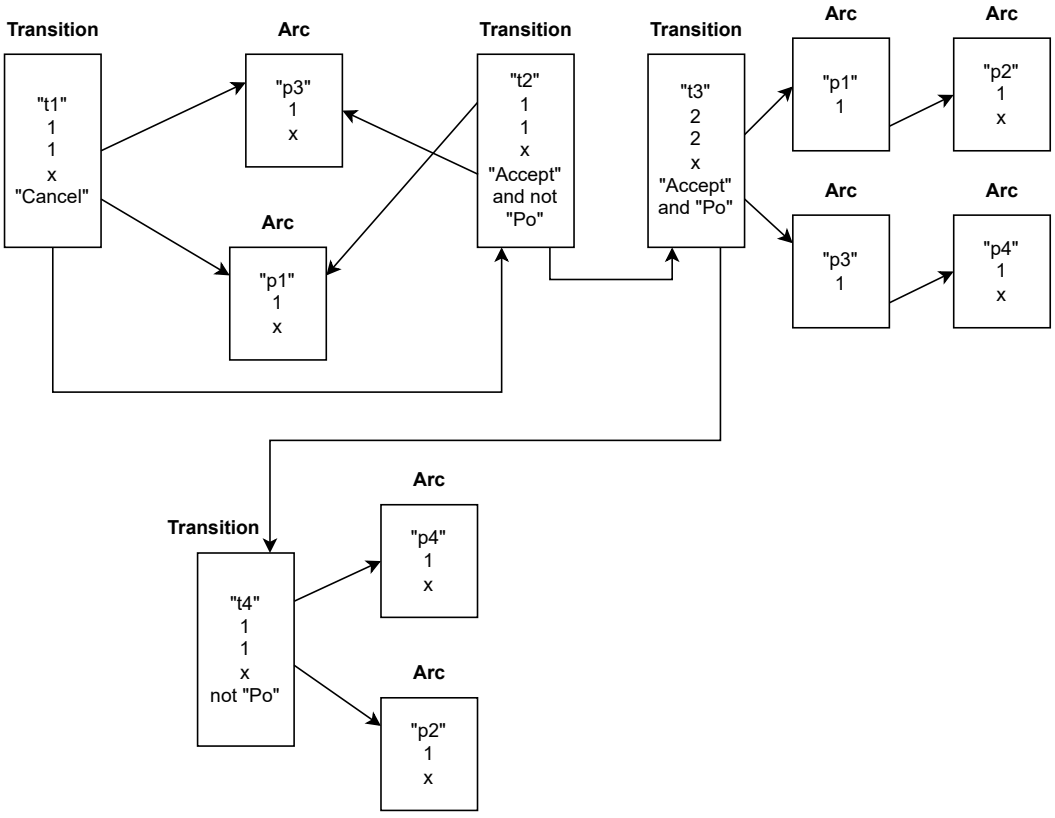


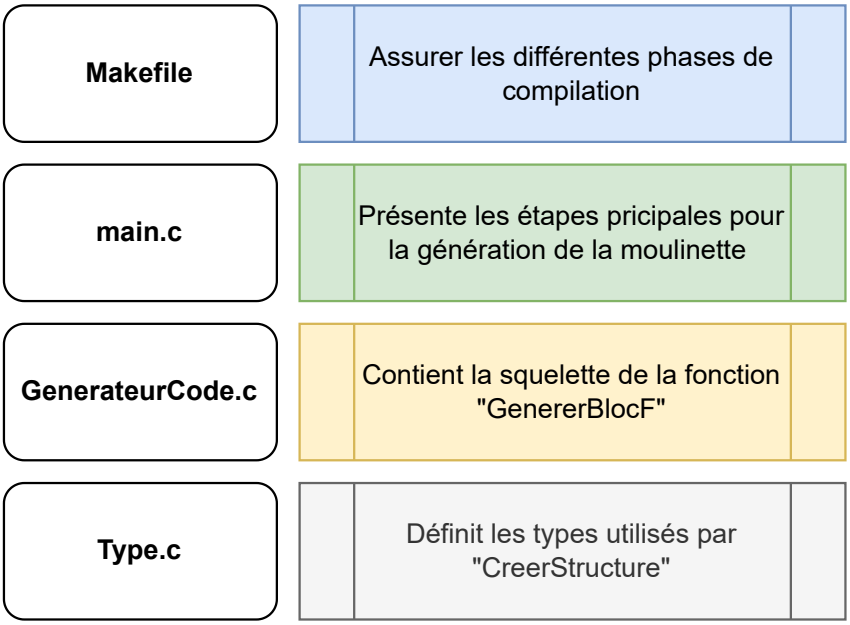
FIGURE 1.4 – Listes chaînées pour Transition et arc.

Chapitre 2

Mise en oeuvre

2.1 Ecriture du programme

Un ensemble de fichiers en langage C ainsi qu'un fichier Makefile pour développer une application permettant de générer du code, en utilisant la méthode de codage 1 parmi n.



2.2 Développement envisagé

Pour maîtriser et exploiter l'organisation de la structure de données représentant notre MEF qui a un comportement d'un Rdp nous avons développé :
— Le code de "GenererLog" qui génère un fichier texte indiquant la structure élaborée.
— La fonction "GenererBlocs" qui représente pour le bloc F l'évolution des état suivant en fonction des état précédent ainsi que des entrées.
— Ensuite on génère le bloc M dans la méme fonction "GenererBlocs" qui représente la synchronisation entre état précèdent et état suivant.
—Le bloc G décrit les actions on va l'ajouter directement dans le fichier generer en VHDL.

2.3 Le GenererLog

Afin de vérifier la cohérence de notre modèle il indispensable de passer par cette étape pour valider notre logique de raisonnement en se basant sur les listes chainées définit précédement, Cette fonction prend en entrée des structures de données représentant les places et les transitions d'un réseau de Petri, ainsi que le nom de fichier pour enregistrer un journal des opérations. À l'intérieur de la fonction, les informations sur les places sont parcourues, avec leur nom, leur marquage initial et les actions associées, tandis que les transitions sont également parcourues, affichant leur nom, leur prédicat, les arcs entrants et sortants, ainsi que les actions associées. Toutes ces informations sont écrites à la fois sur la console et dans le fichier "LeFichierLog".
Le code dévelopé pour cette partie se trouve dans l'annexe (Listing 4).
Après l'exécution de notre code, voici les résultats obtenus :

Résultat obtenu pour GenererLog
Nom de la place : p2
Marquage initial = 1
Nom de la place : p1
Marquage initial = 1
L'action faite est Aff
Nom de la place : p3
Marquage initial = 0
L'action faite est Moteur
Nom de la place : p4
Marquage initial = 0
L'action faite est Alarme
L'action faite est Aff
L'action faite est Sens
La Transition est : t1
Le Predicat est : Cancel='1';
Les Arcs entrants sont :
p3
1
Les Arcs sortants sont :
p1
1
L'action :;
La Transition est : t22
Le Predicat est : Accept='1' or Po='0';
Les Arcs entrants sont :
p1
1
Les Arcs sortants sont :
p3
1
L'action :;

La Transition est : t4
Le Predicat est : Po='0';
Les Arcs entrants sont :
p4
1

```
Les Arcs sortants sont :
p2
1
L'action : ;
La Transition est : t3
Le Predicat est : Accept='1' and Po='1' ;
Les Arcs entrants sont :
p2
1
p1
1
Les Arcs sortants sont :
p4
1
p3
1
L'action : ;
```

2.4 Interprétation

A partir du résultat obtenu on voit que notre modèle se compose de quatre places, désignées par p1, p2, p3 et p4. Les marquages initiaux diffèrent entre ces places, avec p1 et p2 initialisées à 1 tandis que p3 et p4 sont initialement vides (marquées à 0). Chaque place est associée à des actions spécifiques : p1 effectue l'action "Aff", p3 exécute "Moteur", tandis que p4 a deux actions, "Alarme" et "Aff", en plus de "Sens". Les transitions du réseau, nommées t1, t22, t4 et t3, sont gouvernées par des prédicats définis. Par exemple, la transition t1 peut se produire lorsque la condition "Cancel" est évaluée à vrai. La transition t22, quant à elle, peut être franchie si soit "Accept" est vrai, soit "Po" est faux. Chaque transition possède des arcs entrants et sortants avec des poids associés, régissant le flux de marquage entre les places. L'arc entrant de t1 depuis p3 a un poids de 1, et l'arc sortant vers p1 a également un poids de 1.

L'étape suivante consiste à developper le fichier GenerateurCode afin d'obtenir note code en langage cible (VHDL).

2.5 Le GenerateurCode

Dans cette partie on vise à générer du code VHDL à partir des informations sur les places, transitions et arcs de notre MEF qui a un comportement d'un Rdp. Ce code prend en entrée la première transition, la première place et le nom du fichier où le code VHDL sera enregistré. À l'intérieur de la fonction, le code VHDL est construit en utilisant des boucles pour parcourir les places et les transitions. Pour chaque place, les conditions pour son marquage sont établies en fonction des arcs entrants et sortants des transitions associées, ainsi que de leurs prédicats. Ensuite, le bloc M est créé pour initialiser les marquages des places, en tenant compte de la condition initiale. Enfin, le bloc G sera ajouté manuellement dans le fichier générer en langage VHDL dans le logiciel Quartus. Le code ce trouve dans l'annexe (Listing 5)

En lançant notre programme, nous avons obtenu le résultat suivant ;

```
1  /*----------------------Bloc F ----------------------------*/
2  p2_s <= (p4_p AND Po='0') OR (p2_p AND NOT(Accept='1' AND Po='1'));
3  p1_s <= (p3_p AND Cancel='1') OR (p1_p AND NOT(Accept='1' OR
       Po='0' OR Accept='1' AND Po='1'));
4  p3_s <= (p1_p AND Accept='1' OR Po='0') OR (p2_p AND p1_p AND
       Accept='1' AND Po='1') OR (p3_p AND NOT(Cancel='1'));
5  p4_s <= (p2_p AND p1_p AND Accept='1' AND Po='1') OR (p4_p AND
       NOT(Po='0'));
6
7  /* --------------------------------Bloc M------------------------ */
8  IF (init = '1') THEN
9      p2_p <= 1;
10     p1_p <= 1;
11     p3_p <= 0;
12     p4_p <= 0;
13 ELSE
14     p2_p <= p2_s;
15     p1_p <= p1_s;
16     p3_p <= p3_s;
17     p4_p <= p4_s;
18 END_IF;
19
20 /* --------------------------------Bloc G------------------------ */
```

2.6 Interprétation

Le "Bloc F" représente les conditions pour la transition d'un marquage à un autre. Chaque ligne correspond à une place, et les conditions sont établies en fonction des arcs entrants et sortants des transitions associées, ainsi que de leurs prédicats. Par exemple, pour la place p2, la condition pour son marquage p2s est une combinaison de différentes conditions impliquant les places p4 et p2, ainsi que les prédicats correspondants.

Le "Bloc M" est dédié à l'initialisation des marquages des places. Si l'état initial (init) est vrai, alors les marquages des places sont définis selon des valeurs spécifiques. Sinon, les marquages des places sont définis en fonction des marquages précédents (p2s, p1s, p3s, p4s).
Enfin, le "Bloc G" représente la partie à réaliser manuellement dans le code VHDL généré.

2.7 Implémentation

Après avoir récupéré la structure de base du code en VHDL, nous l'avons complété en ajoutant le bloc G et en corrigeant les erreurs de syntaxe pour suivre les spécifications acceptées par le logiciel Quartus.

Conclusion

Nous avons tenté d'implémenter notre code sur le prototype de la machine à laver, mais nous n'avons pas obtenu le fonctionnement souhaité. Cela est dû au fait que notre ficher "généré code" a été conçu en pensant que le fichier généré par le logiciel TINA était un Réseau de Petri avec le comportement d'une Machine à États Finis , mais malheureusement ce n'était pas le cas ; le fichier était en réalité un RDP. Nous pensons que si nous utilisons notre code avec le bon fichier, cela fonctionnera.

Annexes

```
1     // 27-03-2023: T.Lasguignes: passage des d finitions dans
          types.h
2
3   #include <string.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   // 27-03-2023: T.Lasguignes: inclusions de .h au lieu de .c
7   #include "types.h"
8   #include "fonctions.h"
9
10  // 27-03-2023: T.Lasguignes: Retrait des d clarations de fonction
        au profit de fonctions.h
11
12  int main(void)
13  {
14      Transition *TransitionsRdP = NULL;
15      Place *PlacesRdP = NULL;
16
17      char LeFichierTINA[MAX_NOM] = "etiquette.ndr";
18      char LeFichierLog[MAX_NOM] = "etiquette.log";
19      char LeFichier[MAX_NOM] = "evolution_modele.txt";
20
21      printf("nom du fichier a Mouliner :");
22      scanf("%s", LeFichierTINA);
23      strcpy(LeFichierLog, LeFichierTINA);
24      strcat(LeFichierTINA, ".ndr");
25      strcat(LeFichierLog, ".log");
26
27      // ouverture du fichier issus de tina
28
29      CreerStructure(LeFichierTINA, &TransitionsRdP, &PlacesRdP);
30
31      GenererLog(PlacesRdP, TransitionsRdP, LeFichierLog);
32      GenererLog(PlacesRdP, TransitionsRdP, "LeFichierLog.log");
33      GenererBlocs(TransitionsRdP, PlacesRdP, LeFichier);
34
35      // 27-03-2023: T.Lasguignes: Ajout de la fonction de nettoyage
            de m moire (pour garder la coh rence avec les cours de C).
36      NettoyerMemoire(&TransitionsRdP, &PlacesRdP);
37
```

```
38      // 27-03-2023: T.Lasguignes: changement du type de retour: 1
            est consid r  comme potentielle faute   l'ex cution. 0
            est un succes.
39      return EXIT_SUCCESS;}
```

Listing 2.1 – main.c

```
1       // 27-03-2023: T.Lasguignes: Cr ation de fonctions.h pour
            regrouper les d clarations de fonctions (pas optimal mais
            plus propre qu'avant)
2
3   #ifndef FONCTIONS_H
4   #define FONCTION_H
5
6   #include "types.h"
7
8   // Fonctions    d velopper
9
10  void GenererLog(Place *, Transition *, char[MAX_NOM]);
11  void GenererBlocs(Transition *TransitionsRdP, Place *PlacesRdp,
        char[MAX_NOM]);
12  int CreerStructure(char[MAX_NOM], Transition **, Place **);
13
14  // Utilis es pour CreerStructure
15  void CreerUneTransition(Transition **, int, char **);
16  void CreerUnePlace(Place **, int, char **);
17  void CreerUnArc(Transition *, int, char **);
18  void NettoyerLaChaine(char *);
19  char **str_split(const char *, const char *, size_t *);
20  char **ExtraireJetons(char *, const char *, size_t *);
21
22  void NettoyerMemoire(Transition **, Place **);
23
24
25  #endif
```

Listing 2.2 – fonctions.c

```
1       // 27-03-2023: T.Lasguignes: passage en librairie => protecion
2   #ifndef TYPES_H
3   #define TYPES_H
4
5   #include <stdio.h>
6
7   // a rendre global dans le fichier types
8   #define MAX_NOM 200
9
10  // 27-03-2023: T.Lasguignes: taille maximale des lignes    lire
        dans les fichiers .ndr
11  #define MAX_LINE_LENGTH 200
12
13  typedef struct ACTION {
14      char Actions[MAX_NOM];
```

```
15      struct ACTION *Suivant;
16  } Action;
17
18  typedef struct ARC {
19      char Place[MAX_NOM];
20      int Poids;
21      struct ARC *Suivant;
22  } Arc;
23
24  typedef struct TRANSITION {
25      char Nom[MAX_NOM];
26      int NbPlacesEntree;
27      Arc *ArcsEntrants;
28      int NbPlacesSortie;
29      Arc *ArcsSortants;
30      char Actions[MAX_NOM];
31      char Predicat[MAX_NOM];
32      struct TRANSITION *Suivant;
33  } Transition;
34
35  typedef struct PLACE {
36      char Nom[MAX_NOM];
37      int Mo;
38      int NbActions;
39      Action *Actions;
40      struct PLACE *Suivant;
41  } Place;
42
43  #endif
```

Listing 2.3 – types.c

```
1   // 27-03-2023: T.Lasguignes: inclusions de .h au lieu de .c
2   //GenererLog
3   #include "types.h"
4   #include <string.h>
5   #include <stdio.h>
6   #include <stdlib.h>
7
8
9   // Cette fonction genere un fichier texte a partir des
        informations fournies sur les places et les transitions
10  void GenererLog(Place *PlacesRdP, Transition *TransitionsRdP, char
        LeFichierLog[MAX_NOM]) {
11      FILE* LogC; // D claration d'un pointeur de fichier
12      Place *LaPlace = PlacesRdP; // Pointeur vers la premi re
            place dans la liste
13      Transition *LaTrans = TransitionsRdP; // Pointeur vers la
            premi re transition dans la liste
14      Arc *ArcIN=NULL; // Pointeur vers les arcs entrants d'une
            transition
15      Arc *ArcOUT=NULL ; // Pointeur vers les arcs sortants d'une
```

```
    transition
16  Action *UneAction=NULL; // Pointeur vers les actions
    associ es    une place ou une transition
17
18  // Ouverture du fichier en mode  criture
19  LogC=fopen(LeFichierLog, "w");
20  if (LogC!=NULL) {
21      // Parcours des places
22      while (LaPlace!=NULL) {
23          // Affichage des informations sur la place
24          printf("Nom de la place: %s \n", LaPlace->Nom);
25          fprintf(LogC,"Nom de la place: %s \n", LaPlace->Nom);
26          printf("Marquage initial = %d \n", LaPlace->Mo);
27          fprintf(LogC,"Marquage initial = %d \n", LaPlace->Mo);
28          printf("Ses Actions sont \n");
29          // Parcours des actions associ es    la place
30          UneAction = LaPlace->Actions;
31          for (int i=0; i<LaPlace->NbActions; i++) {
32              printf("L'action faite est %s \n",
                    UneAction->Actions);
33              fprintf(LogC,"L'action faite est %s \n",
                    UneAction->Actions);
34              UneAction = UneAction->Suivant;
35          }
36          LaPlace= LaPlace->Suivant; // Passage a la place
                suivante dans la liste
37      }
38
39      // Parcours des transitions
40      while (LaTrans!=NULL) {
41          // Affichage des informations sur la transition
42          printf("La Transition est : %s \n", LaTrans->Nom);
43          fprintf(LogC,"La Transition est : %s \n",
                LaTrans->Nom);
44          printf("Le Predicat est : %s \n" , LaTrans->Predicat);
45          fprintf(LogC,"Le Predicat est : %s ;\n" ,
                LaTrans->Predicat);
46
47          // Affichage des arcs entrants de la transition
48          printf("Les Arcs entrants sont : \n");
49          fprintf(LogC,"Les Arcs entrants sont : \n");
50          ArcIN= LaTrans->ArcsEntrants;
51          for(int i=0; i<LaTrans->NbPlacesEntree; i++) {
52              printf("%s \n", ArcIN->Place);
53              fprintf(LogC,"%s \n", ArcIN->Place);
54              printf("%d \n", ArcIN->Poids);
55              fprintf(LogC,"%d \n", ArcIN->Poids);
56              ArcIN= ArcIN->Suivant;
57          }
58
59          // Affichage des arcs sortants de la transition
```

```
60              printf("Les Arcs sortants sont : \n");
61              fprintf(LogC,"Les Arcs sortants sont : \n");
62              ArcOUT= LaTrans->ArcsSortants;
63              for(int i=0; i<LaTrans->NbPlacesSortie; i++) {
64                  printf("%s \n", ArcOUT->Place);
65                  fprintf(LogC,"%s \n", ArcOUT->Place);
66                  printf("%d \n", ArcOUT->Poids);
67                  fprintf(LogC,"%d \n", ArcOUT->Poids);
68                  ArcOUT= ArcOUT->Suivant;
69              }
70
71              // Affichage de l'action associ e    la transition
72              printf("L'action %s \n", LaTrans->Actions);
73              fprintf(LogC, "L'action : %s ;\n", LaTrans->Actions);
74
75              LaTrans= LaTrans->Suivant; // Passage    la transition
                   suivante dans la liste
76          }
77      } else {
78          printf("Error creation \n");
79      }
80
81      fclose(LogC); // Fermeture du fichier
82      printf("\nOn est dans 'GenererLog.c'\n"); // Message indiquant
           l'execution de la fonction
83  }
```

Listing 2.4 – GenererLog

```
1   // 27-03-2023: T.Lasguignes: inclusions de .h au lieu de .c
2   #define MAX_NOM 200
3   #include "types.h" // Inclusion du fichier d'en-t te "types.h"
4   #include <string.h>
5   #include <stdio.h>
6   #include <stdlib.h>
7
8   // D finition de la fonction GenererBlocs qui g n re du code
       VHDL    partir des informations sur les places, transitions et
       arcs
9   void GenererBlocs(Transition *PremiereTransition, Place
       *PremierePlace, char LeFichierCode[MAX_NOM])
10  {
11        Place *LaPlace = PremierePlace; // Pointeur vers la
             premi re place du r seau
12      Transition *LaTrans = PremiereTransition; // Pointeur vers la
           premi re transition du r seau
13      Arc * ArcIN=NULL; // Pointeur vers un arc entrant
14      Arc * ArcOUT=NULL ; // Pointeur vers un arc sortant
15      Action * UneAction=NULL ; // Pointeur vers une action
16      Place *LaPlaceIN = PremierePlace; // Pointeur vers la premi re
           place du r seau (utilis  pour la g n ration de code)
17      Action * UneActionIN=NULL ; // Pointeur vers une action
```

```
      (utilis  pour la g n ration de code)
18    char ActionList [MAX_NOM]=""; // Liste d'actions (utilis  pour
         la g n ration de code)
19
20    FILE *vhdl=NULL; // Pointeur vers le fichier VHDL de sortie
21    int i=0; // Variable de boucle
22
23    printf("\n On est dans 'GenerateurCode.c'\n") ; // Affichage
         d'un message de suivi
24
25    // Ouverture du fichier de sortie en mode   criture
26    vhdl = fopen (LeFichierCode ,"w");
27    if (vhdl != NULL)
28    {
29
30    // Affichage d'un commentaire dans le fichier VHDL  g  n  r
31    printf("/*----------------------Bloc F
         --------------------*/\n");
32    fprintf(vhdl,"/*----------------------Bloc F
         ---------------------------*/\n");
33
34        // Boucle pour chaque place du r seau
35        while (LaPlace != NULL)
36    {
37       LaTrans = PremiereTransition; // R initialisation du
            pointeur vers la premi re transition
38       printf("\t%s_s <= " , LaPlace->Nom); // Affichage du nom de
            la place dans le code VHDL
39       fprintf(vhdl, "\t%s_s <= " , LaPlace->Nom); //   criture   du
            nom de la place dans le fichier VHDL
40       // Boucle pour chaque transition du r seau
41       while (LaTrans != NULL)
42       {
43          ArcOUT=LaTrans->ArcsSortants; // Pointeur vers les arcs
               sortants de la transition
44          // Parcours des arcs sortants de la transition
45          while(ArcOUT != NULL)
46          {
47             if (strcmp(ArcOUT->Place , LaPlace->Nom )==0)
48             {
49                ArcIN=LaTrans->ArcsEntrants; // Pointeur vers les
                     arcs entrants de la transition
50                printf("(");
51                fprintf(vhdl,"(");
52                // Parcours des arcs entrants de la transition
53                while(ArcIN != NULL)
54                {
55                   printf("%s_p AND ",ArcIN->Place); // Affichage
                        de la condition pour l'arc entrant
56                   fprintf(vhdl,"%s_p AND ",ArcIN->Place); //
                          criture   de la condition pour l'arc entrant
```

```
                    dans le fichier VHDL
57                  ArcIN=ArcIN->Suivant;   // Passage    l'arc
                    entrant suivant
58                }
59                printf("%s ) OR ",LaTrans->Predicat); // Affichage
                    du pr dicat de la transition
60                fprintf(vhdl,"%s ) OR ",LaTrans->Predicat); //
                     criture  du pr dicat de la transition dans le
                    fichier VHDL
61             }
62             ArcOUT=ArcOUT->Suivant; // Passage    l'arc sortant
                    suivant
63          }
64          LaTrans=LaTrans->Suivant;  // Passage    la transition
                    suivante
65       }
66       LaTrans = PremiereTransition; // R initialisation du
                    pointeur vers la premi re transition
67       printf("( %s_p AND NOT(" , LaPlace->Nom); // Affichage de la
                    condition pour le marquage de la place
68       fprintf(vhdl, "( %s_p AND NOT(" , LaPlace->Nom); //
                     criture  de la condition pour le marquage de la place
                    dans le fichier VHDL
69       i=1; // R initialisation de la variable de boucle
70       // Boucle pour chaque transition du r seau
71       while(LaTrans !=NULL)
72       {
73          ArcIN=LaTrans->ArcsEntrants; // Pointeur vers les arcs
                    entrants de la transition
74          // Parcours des arcs entrants de la transition
75          while(ArcIN != NULL)
76          {
77             if(strcmp(ArcIN->Place,LaPlace->Nom)==0)
78             {
79                if(i==0)
80                {
81                   printf(" OR"); // Affichage de la condition "OU"
82                   fprintf(vhdl," OR"); //   criture  de la
                       condition "OU" dans le fichier VHDL
83                   i=1; // R initialisation de la variable de
                       boucle
84                }else
85                i=0; // R initialisation de la variable de boucle
86                printf(" %s",LaTrans->Predicat); // Affichage du
                    pr dicat de la transition
87                fprintf(vhdl," %s",LaTrans->Predicat); //   criture
                    du pr dicat de la transition dans le fichier
                    VHDL
88             }
89             ArcIN=ArcIN->Suivant;   // Passage    l'arc entrant
                    suivant
```

```
90          }
91          LaTrans=LaTrans->Suivant;  // Passage    la transition
               suivante
92       }
93       printf(")) ; \n"); // Fin de la condition pour le marquage
            de la place
94       fprintf(vhdl,")) ; \n"); // Fin de la condition pour le
            marquage de la place dans le fichier VHDL
95       LaPlace=LaPlace->Suivant; // Passage    la place suivante
96    }
97
98    printf("/*------------------------------------------------------
99 ----------------------------------------*/\n");
100   printf(vhdl,"/*-----------------------------------------
101 ------------------------------------------- */\n");
102
103   LaPlace = PremierePlace; // R initialisation du pointeur vers
         la premi re place du r seau
104   printf("/* ----------------------------Bloc M
         ---------------------- */\n");
105   fprintf(vhdl,"/* -------------------------------Bloc
         M------------------------ */\n");
106
107   printf("IF (init = '1') THEN \n");
108   fprintf(vhdl,"IF (init = '1') THEN \n");
109   // Boucle pour chaque place du r seau
110   while(LaPlace != NULL)
111   {
112      printf("\t %s_p <= %d; \n", LaPlace->Nom , LaPlace->Mo); //
         Affichage de l'initialisation du marquage de la place
113      fprintf(vhdl,"\t %s_p <= %d; \n", LaPlace->Nom ,
         LaPlace->Mo); //   criture   de l'initialisation du
         marquage de la place dans le fichier VHDL
114      LaPlace= LaPlace->Suivant; // Passage    la place suivante
115   }
116   printf("ELSE \n");
117   fprintf(vhdl,"ELSE \n");
118   LaPlace = PremierePlace; // R initialisation du pointeur vers
         la premi re place du r seau
119   // Boucle pour chaque place du r seau
120   while(LaPlace != NULL)
121   {
122      printf("\t %s_p <= %s_s; \n", LaPlace->Nom , LaPlace->Nom);
         // Affichage de la mise    jour du marquage de la place
123      fprintf(vhdl,"\t %s_p <= %s_s; \n", LaPlace->Nom ,
         LaPlace->Nom); //   criture   de la mise    jour du
         marquage de la place dans le fichier VHDL
124      LaPlace= LaPlace->Suivant; // Passage    la place suivante
125   }
126   printf("END_IF; \n");
127   fprintf(vhdl,"END_IF; \n");
```

```
128
129     printf("\n") ; // Affichage d'une ligne vide
130      fprintf(vhdl,"\n"); //   criture   d'une ligne vide dans le
            fichier VHDL
131
132      } else
133      {
134         printf("Impossible d'ouvrir le fichier Code\n"); //
               Affichage d'un message d'erreur si l'ouverture du
               fichier de sortie   choue
135         exit(1); // Sortie du programme avec code d'erreur
136      }
137
138      printf("/* ----------------------------Bloc G
            ---------------------- */\n"); // Affichage d'un
            commentaire dans le fichier VHDL  g  n  r
139      fprintf(vhdl,"/* -------------------------------Bloc
            G------------------------ */\n"); //   criture   d'un
            commentaire dans le fichier VHDL  g  n  r
140
141      fclose(vhdl); // Fermeture du fichier VHDL de sortie
142  }
```

Listing 2.5 – GenerateurCode

```
1      Nom de la place: p2
2  Marquage initial = 1
3  Nom de la place: p1
4  Marquage initial = 1
5  L'action faite est Aff
6  Nom de la place: p3
7  Marquage initial = 0
8  L'action faite est Moteur
9  Nom de la place: p4
10 Marquage initial = 0
11 L'action faite est Alarme
12 L'action faite est Aff
13 L'action faite est Sens
14 La Transition est : t1
15 Le Predicat est : Cancel='1' ;
16 Les Arcs entrants sont :
17 p3
18 1
19 Les Arcs sortants sont :
20 p1
21 1
22 L'action :  ;
23 La Transition est : t22
24 Le Predicat est : Accept='1' or Po='0' ;
25 Les Arcs entrants sont :
26 p1
27 1
```

```
Les Arcs sortants sont :
p3
1
L'action :  ;
La Transition est : t4
Le Predicat est : Po='0' ;
Les Arcs entrants sont :
p4
1
Les Arcs sortants sont :
p2
1
L'action :  ;
La Transition est : t3
Le Predicat est : Accept='1' and Po='1' ;
Les Arcs entrants sont :
p2
1
p1
1
Les Arcs sortants sont :
p4
1
p3
1
L'action :  ;
```

Listing 2.6 – LeFichierLog

```
    library IEEE;
use ieee.std_logic_1164.all;
-- binome  :Rahmoun-Seghier
-- date TP :5/04/2024
// Programme VHDL sous Quartus
ENTITY RdP_Commande IS
   PORT(
      clk, init, Po, Accept, Cancel :  in std_logic ;
      Moteur, Sens, Alarme, Validation :  out std_logic
      ) ;
END RdP_Commande ;

ARCHITECTURE archi OF RdP_Commande IS

type  etatT is (etatT1, etatT2, etatT3, etatT4, etatT5, etatT6,
   etatT7, etatT8, etatT9) ;
signal   epT, esT : etatT ; -- temporisateur
signal   debtempo, fintempo : std_logic ;
signal   preset : std_logic_vector (2 downto 0) ;
signal   n : natural range 0 to 15 ;
-- autres signaux :
Signal p1_p, p2_p, p3_p, p4_p  : std_logic ;
Signal p1_s, p2_s, p3_s ,p4_s  : std_logic ;
```

```
23
24
25  begin
26
27  -- bloc F du temporisateur
28  process (epT, debtempo, preset)
29  begin
30     case epT is
31        when etatT1 =>
32           if ((debtempo='1') and (preset="111")) then esT <= etatT2
                ;
33        elsif ((debtempo='1') and (preset="110")) then esT <= etatT3
              ;
34        elsif ((debtempo='1') and (preset="101")) then esT <= etatT4
              ;
35        elsif ((debtempo='1') and (preset="100")) then esT <= etatT5
              ;
36        elsif ((debtempo='1') and (preset="011")) then esT <= etatT6
              ;
37        elsif ((debtempo='1') and (preset="010")) then esT <= etatT7
              ;
38        elsif ((debtempo='1') and (preset="001")) then esT <= etatT8
              ;
39        else  esT <= epT ;
40           end if ;
41        when etatT2 =>
42           if (debtempo='1') then esT <= etatT3 ;
43        elsif (debtempo='0') then esT <= etatT1 ;
44        else  esT <= epT ;
45           end if ;
46        when etatT3 =>
47           if (debtempo='1') then esT <= etatT4 ;
48        elsif (debtempo='0') then esT <= etatT1 ;
49        else  esT <= epT ;
50           end if ;
51        when etatT4 =>
52           if (debtempo='1') then esT <= etatT5 ;
53        elsif (debtempo='0') then esT <= etatT1 ;
54        else  esT <= epT ;
55           end if ;
56        when etatT5 =>
57           if (debtempo='1') then esT <= etatT6 ;
58        elsif (debtempo='0') then esT <= etatT1 ;
59        else  esT <= epT ;
60           end if ;
61        when etatT6 =>
62           if (debtempo='1') then esT <= etatT7 ;
63        elsif (debtempo='0') then esT <= etatT1 ;
64        else  esT <= epT ;
65           end if ;
66        when etatT7 =>
```

```
67          if (debtempo='1') then esT <= etatT8 ;
68       elsif (debtempo='0') then esT <= etatT1 ;
69       else  esT <= epT ;
70          end if ;
71       when etatT8 =>
72          if (debtempo='1') then esT <= etatT9 ;
73       elsif (debtempo='0') then esT <= etatT1 ;
74       else  esT <= epT ;
75          end if ;
76       when etatT9 =>
77          if (debtempo='0') then esT <= etatT1 ;
78       else  esT <= epT ;
79          end if ;
80       when others =>
81          esT <= epT ;
82    end case ;
83 end process ;
84
85 -- autre partie de la commande :
86 process ( p1_p, p2_p, p3_p, p4_p, Po, Cancel, Accept)
87 begin
88    p2_s <= (p4_p AND NOT Po) OR ( p2_p AND NOT( Accept and  Po)) ;
89    p1_s <= (p3_p AND Cancel ) OR ( p1_p AND NOT( (Accept or not
          Po) OR (Accept and Po))) ;
90    p3_s <= (p1_p AND Accept AND not Po ) OR ((p2_p AND p1_p AND
          Accept and Po ) OR ( p3_p AND NOT( Cancel))) ;
91    p4_s <= (p2_p AND p1_p AND Accept and Po ) OR ( p4_p AND NOT(
          not Po)) ;
92 end process ;
93
94 process (init, clk)
95 begin
96    if (init='1') then
97     p2_p <= '1';
98     p1_p <= '1';
99     p3_p <= '0';
100    p4_p <= '0';
101   elsif ((clk'event) and clk='1')) then
102      n <= n+1 ;
103      if (n=10) then
104         epT <= esT ;
105         n <= 0 ;
106      end if ;
107      p2_p <= p2_s;
108      p1_p <= p1_s;
109      p3_p <= p3_s;
110      p4_p <= p4_s;
111   end if ;
112 end process ;
113
114 -- signaux du temporisateur
```

```
115 preset <= "101" ; -- pour imposer une valeur constante
116 fintempo <= '1'   when (epT=etatT9)
117             else '0' ;
118 -- autres sorties :
119 -- Block G
120    Sens  <= p4_p ;
121    Moteur <= p3_p;
122 -- Aff <= (p1_p) OR (p4_p);
123    Alarme <= p4_p;
124    Validation <= '0';
125
126 END archi ;
```

Listing 2.7 – Code en VHDL