وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR- ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR ANNABA

جامعة باجي مختار- عنابة

Année : 2021

Faculté: Sciences de l'Ingéniorat
Département: Electronique

MEMOIRE
Présenté en vue de l'obtention du diplôme de : MASTER

Etude et implémentation d'un RTOS approprié à les solutions en langages LADDER et GRAFCET

Domaine : Sciences et Technologie
Filière : Automatique
Spécialité: Automatique et Systéme

Présenté par : Halimi Hani

Devant le JURY

Président : Fezari Mourad	**Prof . UBM Annaba**
Directeur de mémoire : *Attoui Hamza*	**MAA. UBM Annaba**
Examinateur : *Benouaret Mohamed*	**Prof. UBM Annaba**

Résumé

J. A. Stankovic définit un système temps réel comme un système dont le comportement dépend non seulement de l'exactitude du traitement effectué, mais également du temps où les résultats de ces traitements sont produits (un système temps réel n'est pas un système "qui va vite" mais un système qui satisfait à des contraintes temporelles). Un système temps réel est toujours décomposé en **tâches**, chacune ayant des fonctionnalités, des temps d'exécution et des échéances différentes. Cette décomposition transforme le système de contrôle à un système multitâche qui requiert un noyau suit une certaine politique pour ordonnancer. Un processus industriel de production est un véritable système temps réel qui nécessite des décisions exactes au bon moment, avec l'organe qui assure cette décision est l'API (Automate Programmable Industriel). Ce dernier est programmé par des concepteurs en automatiste qui utilisent des langages adaptés à l'automatisation des processus industriels comme le LADDER et le GRAFCET. La notion du temps réel et le multitâche nous guidons vers l'utilisation d'un RTOS (Real Time Operating System) avec ce dernier est un ensemble de services assure la bonne exécution du programme du contrôle. Dans ce projet PFE en essaye de faire l'étude et l'implémentation d'un RTOS avec la décomposition d'une solution en LADDER ou GRAFCET en tâches par l'intermédiaire des services du noyau.

Mots clés : RTOS, SCHEDULING POLITICS, CPU, MICROCONTROLLER, LADDER, GRAFCET.

Abstract

J. A. Stankovic defines a real-time system as a system whose behavior depends not only on the accuracy of the processing performed, but also on the time at which the results of this processing are produced (a real-time system is not a "fast" system, but a system that satisfies temporal constraints). A real-time system is always decomposed into tasks, each with different functionality, execution times and deadlines. This decomposition transforms the control system into a multitasking system that requires a core to follow a certain policy for scheduling. An industrial production process is a true real-time system that requires exact decisions at the right time, with the organ that ensures this decision is the PLC (Programmable Logic Controller). The latter is programmed by automation designers who use languages adapted to the automation of industrial processes such as LADDER and GRAFCET. The notion of real time and multitasking guides us towards the use of a RTOS (Real Time Operating System) with the latter being a set of services ensuring the proper execution of the control program. In this PFE project we try to study and implement an RTOS with the decomposition of a LADDER or GRAFCET solution into tasks through kernel services.

Key words : RTOS, SCHEDULING POLITICS, CPU, MICROCONTROLLER, LADDER, GRAFCET.

ملخص

يعرّف JA Stankovic نظام الوقت الفعلي بأنه نظام يعتمد سلوكه ليس فقط على صحة العلاج الذي يتم إجراؤه ، ولكن أيضًا على الوقت الذي يتم فيه إنتاج نتائج هذه العلاجات (نظام الوقت الفعلي ليس نظامًا "ذلك يسير بسرعة "ولكنه نظام يفي بالقيود الزمنية). يتم دائمًا تقسيم نظام الوقت الفعلي إلى مهام ، لكل منها وظائف مختلفة ، وأوقات تنفيذ ومواعيد نهائية مختلفة. يحول هذا التحلل نظام التحكم إلى نظام متعدد المهام يتطلب نواة لإتباع سياسة معينة لجدولتها. عملية الإنتاج الصناعي هي نظام في الوقت الحقيقي يتطلب قرارات دقيقة في الوقت المناسب ، مع الهيئة التي تضمن هذا القرار هي API (وحدة التحكم المنطقية القابلة للبرمجة الصناعية). تمت برمجة هذا الأخير من قبل مصممي الآليات الذين يستخدمون لغات مناسبة لآلية العمليات الصناعية مثل LADDER و GRAFCET. ترشدنا فكرة الوقت الحقيقي وتعدد المهام إلى استخدام نظام التشغيل في الوقت الفعلي (RTOS) حيث أن الأخير عبارة عن مجموعة من الخدمات التي تضمن التنفيذ السليم لبرنامج التحكم. في هذا المشروع ، نحاول دراسة وتنفيذ نظام RTOS مع تحليل حل LADDER أو GRAFCET إلى مهام من خلال خدمات kernel.

الكلمات الرئيسية : RTOS ، جدولة السياسات ، وحدة المعالجة المركزية ، وحدة التحكم الدقيقة ، سلم ، GRAFCET.

Remerciements

Avant tout, nos remerciements infinis sont adressés à « Dieu le Tout Puissant» de nous avoir donné le courage et la santé pour achever ce travail. Au moment où s'achève ce travail, permettez-nous de remercier du fond du cœur, tous ceux et toutes celles qui, pendant cette période de travail, nous a dirigé, soutenue, aidé et encouragé.

Tout d'abord nous exprimons nos respectueux remerciements à notre encadreur Mr ***Atoui Hamza*** *qu'il trouve ici l'expression de notre profonde reconnaissance tant pour avoir accordé sa confiance, sa grande disponibilité et ses précieux conseils, son aide et le temps qu'il nous a consacré pour la réalisation de ce de ce travail. Je prie Dieu de lui accorder un bon rétablissement, qu'il sera en bonne santé et avec nous le plus tôt possible 'Rabi Yechfik Chikh'.*

Nos remerciements vont aussi aux membres de jury Mr ***Benouart Mohamed*** *et Mr* ***Fezari Mourad*** *de nous avoir fait l'honneur D'accepter d'évaluer ce travail. Notre reconnaissance va également à tous les enseignants du département de ..... De l'université de Badji Mokhtar Annaba pour l'aide pendant notre formation d'étude.*

Le plus grand merci revient à nos parents qui nous ont guidé et encouragé à travailler fort et pour leur confiance qui nous a permis de donner le meilleur de nous-même afin de réaliser leur espoir mis en nous, qui n'est autre que réussite dans notre vie et nos études. Jamais on n'aura atteint nos objectifs sans eux. Que Dieu les protège et les garde en bonne santé. Enfin, nous réservons un vif remerciement à nos amies pour leur aide de, leur encouragement et amour.

MERCI À TOUS...

Dédicaces

En guise de reconnaissance à tous ceux qui m'on encouragé et aidé dans la réalisation de ce travail de recherche, je dédie ce modeste travail qui est le fruit de mes efforts

***A ma chère mère GHANIA**, pour ses précieux conseils, son soutien et sa tendresse inestimable, et sa générosité remarquable.*

***A mon cher père SEBTI**, qui a été toujours à mes côtés avec ses conseils et ses encouragements qui me donne la force et le courage d'aller toujours de l'avant dans mes études.*

*A mes chères frères **Yacine** et **Rabie***

*A ma chère sœur **Chahinez***

*A mes oncles **Bekhouche Bouraui** et **Salhi Mourad***

*A ma tante **Salhi Nadia***

*A mes chères cousins : **Walid** et **Brahim***

*Une spéciale dédicace à Mon ami **Cherif** qui compte beaucoup pour moi.*

A tous mes amis et tous qui me connaisse je vous aime

*Et surtout les familles **Halimi** et **Salhi***

Toute la promotion Master II 2020/2021.

Halimi Hani.

Liste des Tableaux

Liste des Figures

Chapitre 1 : Etat de l'art sur les systèmes temps réel et les politiques d'ordonnancement.

Chapitre 2 : Conversion du LADDER/GRAFCET à un algorithme informatique.

SOMMAIRE

SOMMAIRE

Chapitre II : Conversion du LADDER/GRAFCET à un algorithme informatique.

Introduction générale

Introduction générale :

J. A. Stankovic définit un système temps réel comme un système dont le comportement dépend non seulement de l'exactitude du traitement effectué, mais également du temps où les résultats de ces traitements sont produits (un système temps réel n'est pas un système "qui va vite" mais un système qui satisfait à des contraintes temporelles).

Un système temps réel est toujours décomposé en tâches, chacune ayant des fonctionnalités, des temps d'exécution et des échéances différentes. Cette décomposition transforme le système de contrôle à un système multitâche qui requiert un noyau suit une certaine politique pour ordonnancer. Un processus industriel de production est un véritable système temps réel qui nécessite des décisions exactes au bon moment, avec l'organe qui assure cette décision est l'API (Automate Programmable Industriel). Ce dernier est programmé par des concepteurs en automatiste qui utilisent des langages adaptés à l'automatisation des processus industriels comme le LADDER et le GRAFCET.

La notion du temps réel et le multitâche nous guidons vers l'utilisation d'un RTOS (Real Time Operating System) avec ce dernier est un ensemble de services assure la bonne exécution du programme du contrôle.

L'objectif Dans ce projet PFE est de présenter une revue en essaye de faire l'étude et l'implémentation d'un RTOS avec la décomposition d'une solution en LADDER ou GRAFCET en tâches par l'intermédiaire des services du noyau.

Pour atteindre notre objectif nous allons organiser notre travail comme suit.

Dans le premier chapitre de ce modeste mémoire nous allons présenter des différents types de système temps réel strict/ temps réel souple/ temps réel ferme, ensuite on a vu les caractéristiques du système temps réel qui sont : contraintes du temps ; correction ; embarqué ; sécurité ; concurrence ; distribué et stabilité.

Après on a traité les différentes méthodes du système temps réel (méthodes fonctionnelles structures / méthodes orientés objets / méthodes orientés composant).

On a ensuite cités le domaine d'applications des systèmes temps réel et pour finir son ordonnancement (tâches périodiques / apériodiques/ sporadiques).

Dans un deuxième chapitre nous allons présenter les l'API (Automate Programmable Industriel) et langages adaptés à l'automatisation le LADDER et le GRAFCET. Après on va faire f la conversion du ladder / grafcet en algorithme informatique que l'utilisation des systèmes embarqués à base de **µC** « La plateforme Arduino ou Atmel».

Dans un troisième, nous allons présenter les algorithmes et la réalisation des décompositions suivantes ; du Ladder vers tâches et du Grafcet vers tâches, en utilisant pendant la simulation ; l'arduino ainsi que le proteus ; De là, nous pouvons aller vers multitâche à travers et l'implémentation d'un RTOS.

Chapitre I

Etat de l'art sur les systèmes temps réel et les politiques d'ordonnancement.

I. L'état de l'Arte sur les systèmes temps réel :

I.1.Introduction :

Les systèmes informatiques temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat, autrement dit le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés. Les systèmes informatiques temps réel sont aujourd'hui présents dans de nombreux secteurs d'activités : dans l'industrie de production par exemple, au travers des systèmes de contrôle de procédé (usines, centrales nucléaires), chaîne de montage, usine chimique, etc.

Ces systèmes se diffèrent par bien des aspects, comme l'utilisation, la taille, et la criticité, mais restent néanmoins tous caractérisés par des contraintes temporelles plus ou moins strictes. Leur développement nécessite des techniques spécialisées. L'analyse des besoins doit prendre en compte l'aspect temporel qui doit être précisé et vérifié. La conception doit faire face l'architecture du système, la répartition des tâches et des processeurs, l'ordonnancement détaches…etc. Le tout est soumis à des contraintes en relation avec le temps et les ressources.

I.2. Définition :

Un système temps réel est un système (application ou ensemble d'applications) informatique dont le fonctionnement est assujetti à l'évolution dynamique d'un procédé extérieur qui lui est connecté et dont il doit contrôler le comportement. [1]

La correction d'un système temps réel dépend non seulement de la justesse des calculs mais aussi du temps auquel les résultats sont produits (contraintes temporelles).

Un système temps réel n'est pas un système « qui va vite / rapide» mais un système qui satisfait des contraintes temporelles (les contraintes de temps dépendent de l'application et de l'environnement alors que la rapidité dépend de la technologie utilisée, celle du processeur par exemple). [1]

En règle générale, les interactions d'un système temps réel sur le monde réel s'effectuent via des capteurs et des actionneurs, plutôt qu'un clavier et un écran d'ordinateur standard. Et les contraintes temporelles généralement exigent que les interactions s'accomplissent sur des

intervalles temporels prédéfinis. Il convient de noter que cela est tout à fait différent quand l'interaction s'exécute le plus rapidement possible. [1]

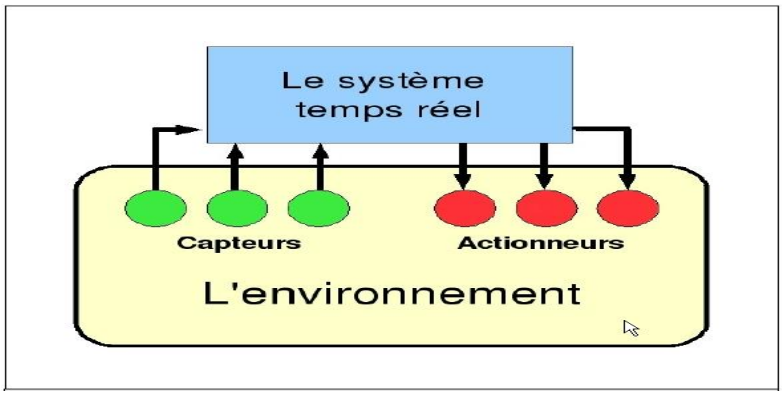


Figure1.1 : Architecture générique d'un système temps réel.

I.3. Classification des systèmes temps réel :

I.3.A-Le temps réel strict :

Le RTOS ne tolère aucun dépassement de ces contraintes, ce qui est souvent le cas lorsque de tels dépassements peuvent conduire à des situations critiques, voire catastrophiques : pilote automatique d'avion, système de surveillance de centrale nucléaire, etc. On peut considérer qu'un système temps réel strict doit respecter des limites temporelles données même dans la pire des situations d'exécution possibles. [2]

Ex : contrôle de trafic aérien, système de conduite de missile, etc.

I.3.B-Le temps réel souple :

S'accommode de dépassements des contraintes temporelles dans certaines limites au-delà desquelles le système devient inutilisable ou pénible : visioconférence, jeux en réseau, etc. Un système temps réel souple doit respecter ses limites pour une moyenne de ses

exécutions. On tolère un dépassement exceptionnel, qui pourra être compensé à court terme. [2]

Ex : projection vidéo (décalage entre le son et l'image).

I.3.C-Temps réel ferme (firme real-time) : temps réel souple avec le manquement occasionnel des échéances.

Ex. : projection vidéo (perte de quelques trames d'images).

Tableau 1.1 : Caractéristiques de systèmes temps réel souple.

Caractéristiques	Temps réel strict	Temps réel souple
Contraintes temporelles	Strict	Souple
Environnement	Monde Externe	Ordinateurs
Performance étant surchargée	Prévisible	Dégradée
Criticité	Critique	Non Critique
Redondance matérielle	Répondue	Rare
Granularité temporelle	Micro/ milli-seconde	milli-seconde/seconde
Granularité des données	Petite	Grande
taille des fichiers	Petite	Grande

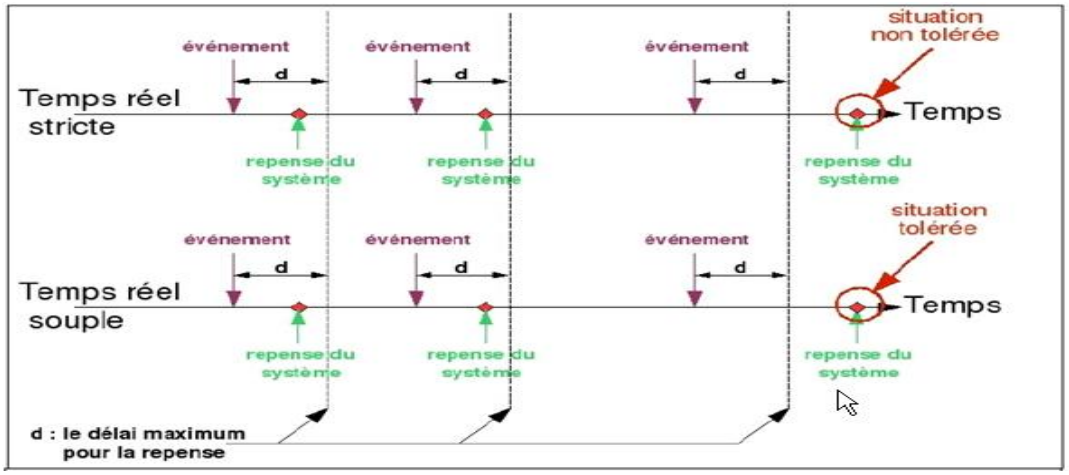


Figure 1.2 : Comparaison temps réel strict et temps souple.

I.4. Caractéristiques du temps réel :

I.4.A. Contraintes de temps :

Les contraintes de temps liées aux systèmes en temps réel signifient simplement l'intervalle de temps alloué pour la réponse du programme en cours. Ce délai signifie que la tâche doit être achevée dans cet intervalle de temps. Le système en temps réel est responsable de l'achèvement de toutes les tâches dans leur intervalle de temps.

I.4.B. La correction :

La correction est l'un des aspects les plus importants des systèmes en temps réel. Les systèmes en temps réel produisent un résultat correct dans un intervalle de temps donné. Si le résultat n'est pas obtenu dans l'intervalle de temps donné, le résultat n'est pas considéré comme correct. Dans les systèmes en temps réel, la correction du résultat consiste à obtenir un résultat correct dans le temps imparti.

I.4.C. Embarqué :

Tous les systèmes temps réel sont aujourd'hui embarqués. Un système embarqué est une combinaison de matériel et de logiciel conçue dans un but spécifique. Les systèmes en temps réel collectent les données de l'environnement et les transmettent aux autres composants du système pour traitement.

I.4.D. La sécurité :

La sécurité est nécessaire pour tout système, mais les systèmes en temps réel offrent une sécurité critique. Les systèmes en temps réel peuvent également fonctionner pendant une longue période sans défaillance. Ils se rétablissent également très rapidement lorsqu'une défaillance survient dans le système et ne causent aucun dommage aux données et aux informations.

I.4.F. Concurrence :

Les systèmes en temps réel sont concurrents, ce qui signifie qu'ils peuvent répondre à un certain nombre de processus à la fois. Il y a plusieurs tâches différentes en cours dans le système et il répond en conséquence à chaque tâche dans de courts intervalles. Cela fait des systèmes en temps réel des systèmes concurrents.

I.4.E. Distribué :

Dans divers systèmes en temps réel, tous les composants des systèmes sont connectés de manière distribuée. Les systèmes en temps réel sont connectés de telle sorte que les

différents composants se trouvent à différents endroits géographiques. Ainsi, toutes les opérations des systèmes en temps réel sont effectuées de manière distribuée.

I.4.H. Stabilité :

Même lorsque la charge est très lourde, les systèmes en temps réel répondent dans la contrainte de temps, c'est-à-dire que les systèmes en temps réel ne retardent pas le résultat des tâches, même lorsque plusieurs tâches se déroulent en même temps. Cela apporte la stabilité dans les systèmes en temps réel. **[3]**

I.5.Méthodes de développement de RTOS (real-time operating system) :

I.5.1.Méthodes fonctions des structures :

- SA_RT: Structured Analysis Real Time. [4]

- DARTS: Design Approach for Real-Time Systems. [5]

- SDL : Spécification and Description Langage etc. [6]

I.5.2.Méthodes Orientées Objet :

- UML: Unified Modelling Language. [7]

- HOOD : Hierarchical Object Oriented Design. [8]

I.5.3.Méthodes orientées composant :

- KOALA : technologie composant développée par Philips pour la conception de composants électroniques grand public. [9]

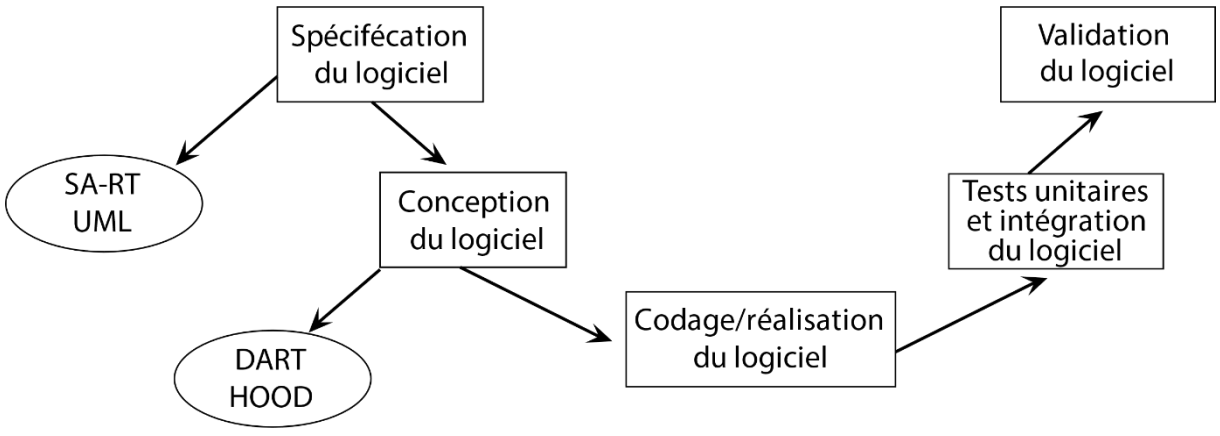


Figure 1.3 : Exemple Méthodes de développements SA_RT.

1.6. Applications du système en temps réel [10] :

Le système en temps réel a des applications dans divers domaines de la technologie. Ici, nous allons discuter des applications importantes du système en temps réel.

1.6.1. Application industrielle :

Les systèmes en temps réel jouent un rôle important dans les industries modernes. Les » systèmes sont basés sur le temps réel afin d'obtenir un rendement maximal et précis. Pour ce faire, les systèmes en temps réel sont utilisés dans la plupart des organisations industrielles. Ces systèmes permettent en quelque sorte d'obtenir de meilleures performances et une productivité élevée en moins de temps. Voici quelques exemples d'applications industrielles : Usine d'assemblage de voitures automatisée, usine chimique, etc.

1.6.2. Application dans le domaine médical :

Dans le domaine des sciences médicales, les systèmes en temps réel ont un impact énorme sur la santé et le traitement des êtres humains. Grâce à l'introduction du système en temps réel dans la science médicale, de nombreuses vies sont sauvées et le traitement de maladies complexes est devenu plus facile. Les personnes, en particulier celles liées à la médecine, se sentent désormais plus détendues grâce à ces systèmes. Voici quelques exemples d'applications de la science médicale : Robot, scanner IRM, radiothérapie, etc.

1.6.3. Applications pour les équipements périphériques :

Le système en temps réel a rendu l'impression de grandes bannières et autres choses très facile. Une fois que ces systèmes ont été utilisés, le monde de la technologie est devenu plus fort. Les équipements périphériques sont utilisés à des fins diverses. Ces systèmes sont intégrés à des micro-puces et fonctionnent avec précision afin d'obtenir la réponse souhaitée. Voici quelques exemples d'applications d'équipements périphériques : Imprimante laser, télécopieur, appareil photo numérique, etc. **[10]**

II. L'état de l'art sur d'ordonnancement :

Une tâche temps réel est l'entité de base des programmes temps réel. Ce sont ces entités qui composent le programme d'un système temps réel. Les tâches temps réel peuvent être associées à des calculs, des alarmes, des traitements d'entrée/sortie, ...

Les tâches temps réel permettent ainsi de contrôler le procédé externe via un ensemble de capteurs et d'actionneurs intégrés au procédé. Elles utilisent les primitives de l'exécutif temps réel sur lequel elles sont exécutées pour communiquer entre elles, faire des acquisitions de données (depuis des capteurs du système contrôlé) ou encore actionner des commandes sur le système contrôlé. Mais pour respecter les caractéristiques du procédé (p. ex. Désactiver à temps le contrôleur de vitesse d'une voiture dont on actionne les freins), les tâches sont soumises à des contraintes temporelles.

II.1.1. Ordonnancement :

Le concepteur d'un système temps réel définit une configuration (ou système) détaches périodiques. Elles sont soumises à des contraintes temporelles dont le respect doit être validé (vérifié) avant la mise en service du système ou durant l'exécution des tâches si leurs caractéristiques ne sont pas connues à priori.Il existe plusieurs modèles de tâches : les tâches *périodiques, apériodiques* et *sporadiques*.

Une *configuration* de tâches est définie par n (n ∈ N*) tâches.

II.2.Tâches périodiques :

Les tâches périodiques représentent les tâches récurrentes dont les activations successives sont séparées par une période constante. Il existe dans la littérature temps réel plusieurs modèles de tâches périodiques. Le plus simple mais aussi le plus fondamentales celui de LIU et LAYLAND. Leur modèle représente chaque tâche par deux paramètres [11]

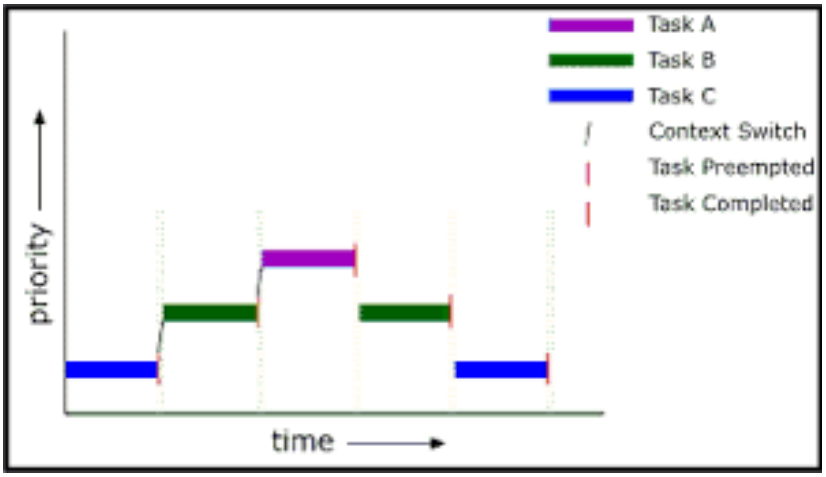


Figure 1.4 : Exemple des taches dans RTOS.

Des tâches peuvent au cours de leur exécution, partager des ressources critiques en exclusion mutuelle (c.-à-d. qu'à chaque instant, au plus une seule tâche peut utiliser la ressource). Une tâche ne peut pas utiliser une ressource occupée par une autre tâche. [12]

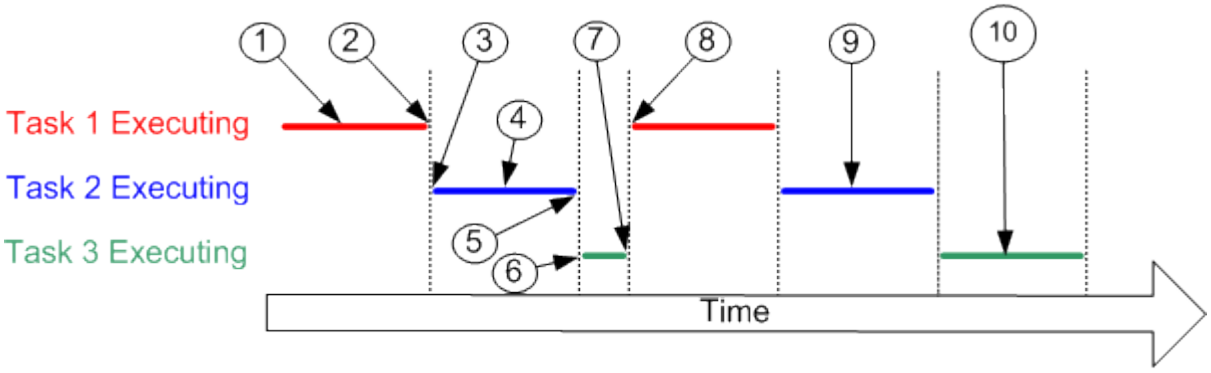


Figure 1.5 : Ordonnancement des taches dans un RTOS.

II.1.2.1. Algorithmes hors-ligne/enligne :

Les algorithmes d'ordonnancement peuvent être classés dans deux catégories, les algorithmes d'ordonnancement hors-ligne et en ligne. Les algorithmes d'ordonnancement hors-ligne construisent la séquence d'ordonnancement complète sur les bases des paramètres temporels de l'ensemble des tâches de la configuration. L'ordonnancement est construit complètement avant l'exécution de l'application. [13]

Les *algorithmes d'ordonnancement en ligne* choisissent, quant à eux, dynamiquement la prochaine requête à exécuter en fonction des paramètres des tâches prêtes à l'exécution.

Ils ne connaissent à un instant t que les tâches dont les dates d'activation sont antérieures à t. Un algorithme d'ordonnancement en ligne est *préemptif* si l'algorithme peut préempter l'exécution d'une tâche pour en choisir une autre. Dans le cas inverse, l'algorithme est *non-préemptif*. Selon le type d'attribution des priorités aux tâches, il existe trois *classes* d'algorithmes en ligne. [13]

II.2. Principaux algorithmes d'ordonnancement en ligne :

Les principaux algorithmes classiques en ligne pour l'ordonnancement de tâches périodiques. Pour chacun de ces algorithmes, le mode d'affectation des priorités aux tâches ainsi que son fonctionnement sont détaillés.

II.2.1. Rate Monotonic (RM) :

L'algorithme à priorité fixe pour les tâches, Rate Monotonic ou (RM), a été introduit par LIU et LAYLAND. [11], les priorités selon l'algorithme RM sont inversement proportionnelles aux périodes des tâches. La tâche la plus prioritaire est celle ayant la plus petite période. L'algorithme RM est optimal dans la classe des algorithmes à priorité fixe pour l'ordonnancement de tâches périodiques, indépendantes, à départ simultané au démarrage et à échéance sur requête. Si l'une de ces deux conditions est relâchée, RM n'est plus optimale. [14]

Cet algorithme à priorité fixe pour les tâches, diffère de Rate Monotonic car il base son ordonnancement non pas sur les périodes mais sur les échéances relatives.

L'algorithme DM est optimal pour l'ordonnancement de tâches périodiques, indépendantes et à départ simultané au démarrage. À noter que si les tâches sont à échéance sur requête, RM et DM sont équivalents.

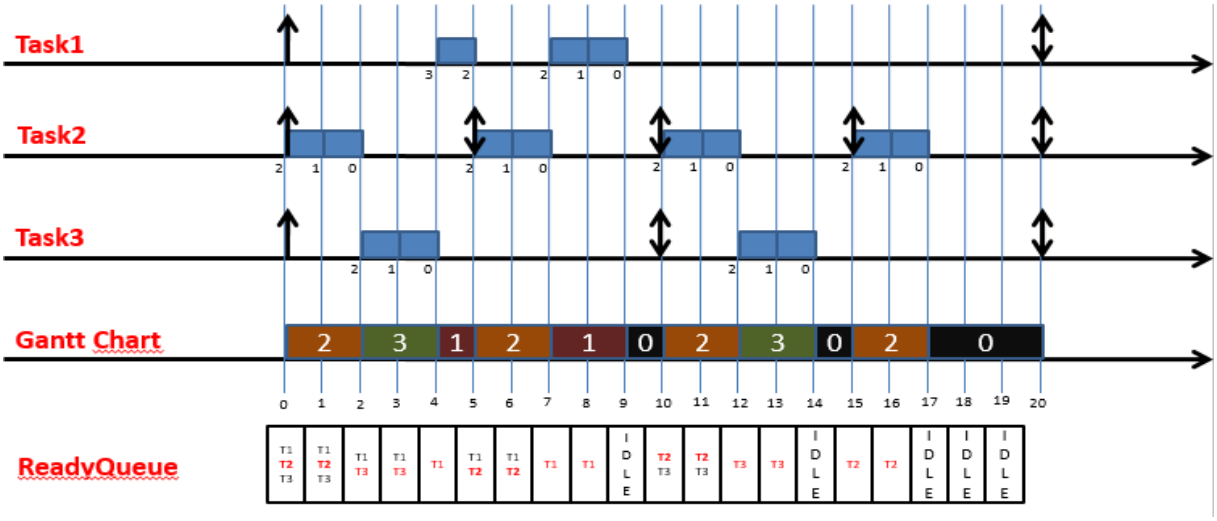


Figure 1.6 : Exemple sur le Rate Monotonic (RM)**.**

II.2.3. Earliest Deadline First (EDF):

Ear liest Deadline First ou (EDF), qui est un algorithme à priorité fixe pour les requêtes ,fut présenté par Jackson en 1955. Ear liest Deadline First attribue, à chaque instant t, la priorité la plus grande à la requête ayant la plus petite échéance absolue.

Ear liest Deadline First est optimal pour l'ordonnancement de configuration de tâches indépendantes. Par conséquent, si une configuration (avec les caractéristiques précédentes) est ordonnança blé, elle le sera par EDF. [15]

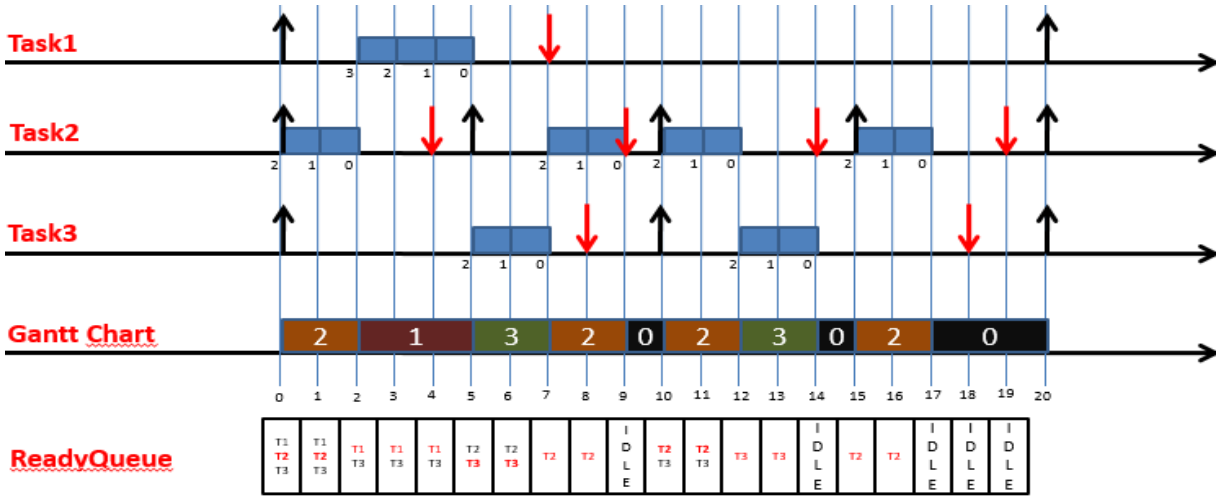


Figure 1.7: Example sur le Earliest Deadline First (EDF).

II.2.4. Least Laxity First (LLF):

Least First est un algorithme à priorité dynamique. Les priorités assignées par LLF aux requêtes sont inversement proportionnelles à la valeur de la laxité dynamique des requêtes. [16]

Least Laxity First attribue à chaque instant t la priorité la plus élevée à la requête ayant le plus faible laxité dynamique. La laxité de la kerequête de !i à uninstant t est égal à Li,k(t) = di,k −t−Ci(t) = Di(t)−Ci(t), où di,k est l'échéance absolue de la kerequête.

Least Laxity First (comme EDF), est un algorithme optimal pour l'ordonnancement de tâches indépendantes et avec des échéances relatives inférieures ou égales aux périodes. [17]

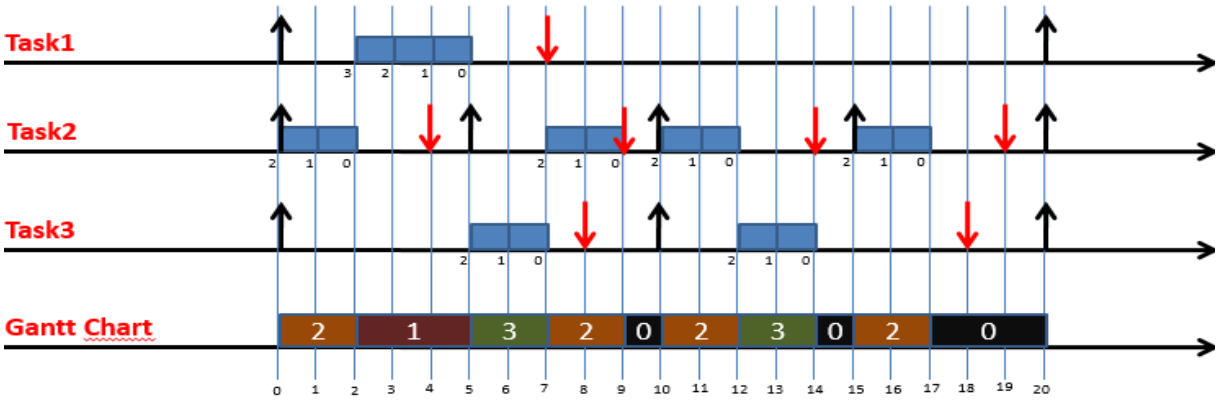


Figure 1.8 : Exemple sur le Least Laxity First (LLF).

II.3. Tâches apériodiques :

Une tâche apériodique a une date d'activation qui n'est connue qu'à l'activation et une certaine durée d'exécution. Par conséquent, deux paramètres suffisent pour représenter une tâche apériodique : = (r,C). Où r est sa date et C son temps d'exécution.

Les tâches apériodiques sont en général à contraintes souples car elles n'ont pas d'échéance avant la fin de leur exécution. Dans le cas contraire, un paramètre d'échéance d sera ajouté dans le modèle des tâches.

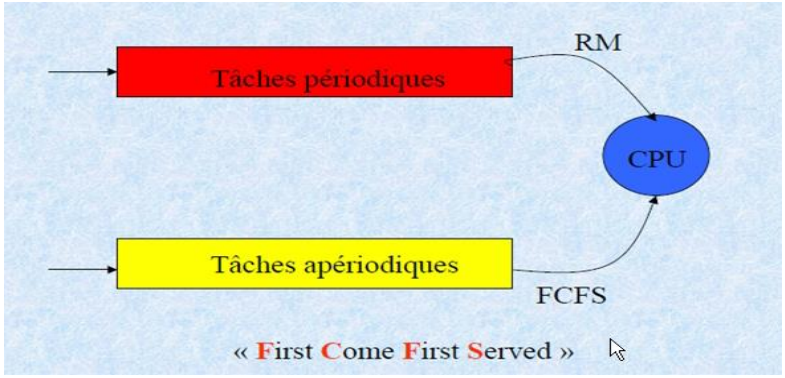


Figure 1.9 : Tâches apériodiques en arrière-plan.

II.3.1.Tâches apériodiques indépendance à contrainte souple :

II.3.1.1. First Come First Served (FCFS):

FCFS : est un algorithme se base sur l'exécution de la tâche qui arrive en premier par une **Ready Queue** en **FIFO** (First In First Out).

La plus haute priorité a donnée au tâche qui arrive en premier. Si on a plus d'une tâche **arrive** au même temps, dans ce cas, la priorité a donnée à l'ordre des tâches.

Le temps d'allocation du CPU au tâche choisi par FCFS est coopératif (la tâche prend le CPU jusqu'à la terminaison de C**)**.

Le modèle utilisé par FCFS est : **tâche (r0, C, O)**.

- **r0** : date du premier réveil.
- **C** : capacité (sa durée d'exécution).
- **O** : Ordre de tâche.

Temps de charge du noyau : Négligeable.

On n'a pas de délai critique (D = ∞).

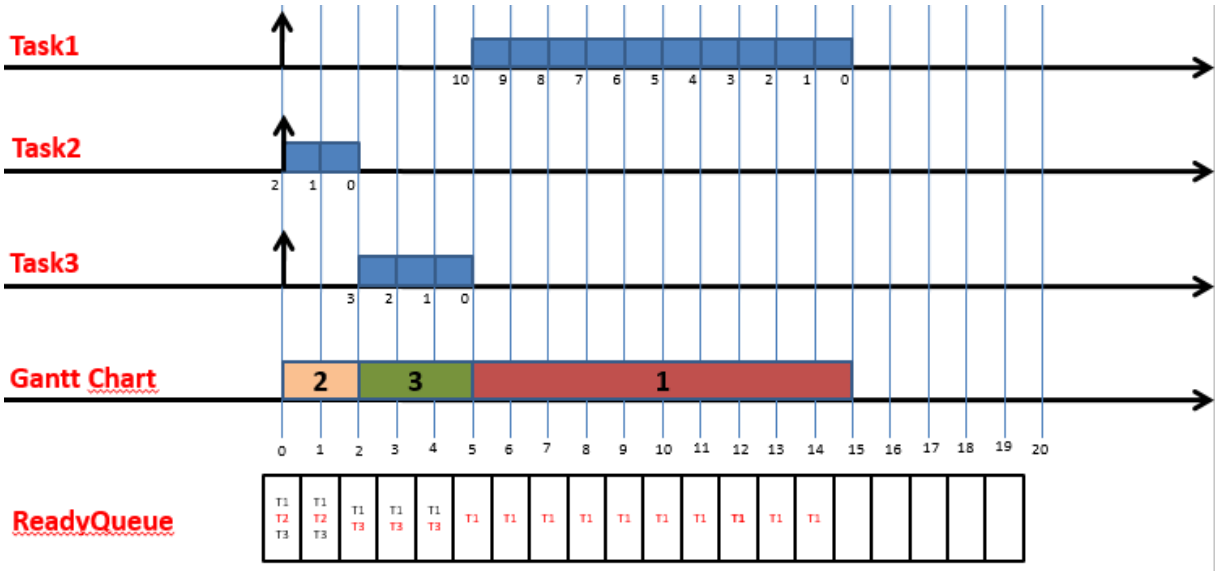


Figure 1 .10 : Exemple sur le First Come First Served (FCFS).

II.3.1.2.Shortest Job First (SJF) :

SJF : est un algorithme se base sur l'exécution de la tâche de la plus petite capacité.

Si on a plus d'une tâche **à la même capacité et arrive au même temps**, dans ce cas, la priorité a donnée à l'**ordre** des tâches.

Le temps d'allocation du CPU au tâche choisi par SJF est **coopératif (la tâche prend le CPU jusqu'à la terminaison de C)**.

Le modèle utilisé par SJF est : tâche**(r0, C, O)**.

- **r0** : date du premier réveil.
- **C** : capacité (sa durée d'exécution).
- **O** : Ordre de tâche.

Temps de charge du noyau : Négligeable.

On n'a pas de délai critique (D = ∞).

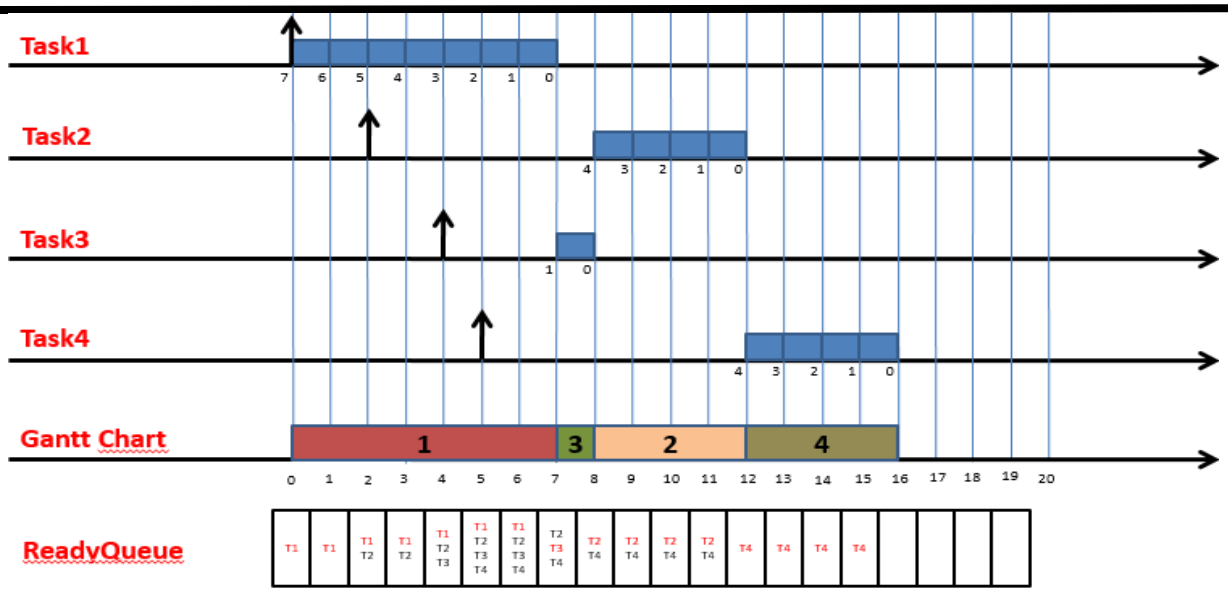


Figure 1.11 : Exemple sur le Shortest Job First (SJF).

II.3.1.3.PriorityBased (PB) :

PB : est un algorithme se base sur l'exécution de la tâche de la plus haute priorité.

La **plus haute priorité** est **fixée par l'utilisateur** selon son besoin.

Si on a plus d'une tâche **à la même priorité**, dans ce cas, la priorité a donnée à la tâche qui **arrive en premier** et si les tâches de la même priorité arrivent en même temps, la priorité a donnée à l'**ordre** des tâches.

Le temps d'allocation du CPU au tâche choisi par PB est **coopératif (la tâche prend le CPU jusqu'à la terminaison de C) ou préemptif**.

Le modèle utilisé par PB est : tâche **(r0, C, O, Pr)**.

Ou :**r0** :date du premier réveil. , **C** : capacité (sa durée d'exécution).,**O** :Ordre de tâche, **P**: Période.

- Temps de charge du noyau : Négligeable.
- On n'a pas de délai critique (D = ∞).

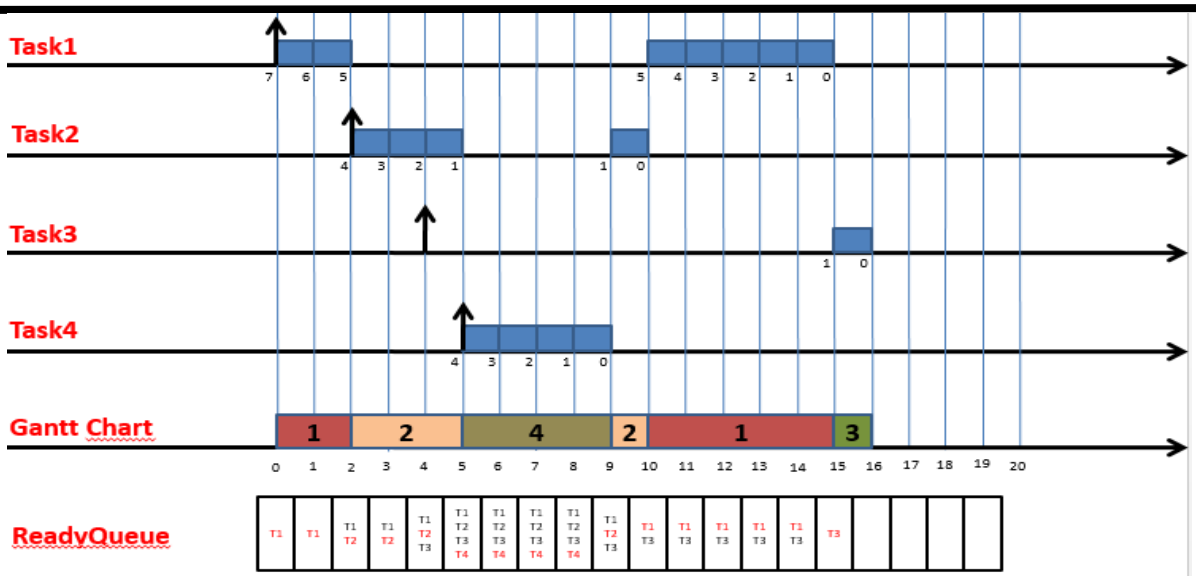


Figure 1.12 : Exemple sur le PriorityBased (PB).

II.3.1.4. Round Robin (RR) :

Round-robin (Tourniquet) est un algorithme d'ordonnancement adapté aux systèmes travaillant en temps partagés.

Une petite unité de temps, appelé **time quantum** ou **time slice**, est définie. La file d'attente (Ready Queue) est géré comme une file **circulaire**, l'ordonnanceur **parcourt** cette file et alloue un temps processeur à chacun des processus pour un intervalle de temps de l'ordre d'un quantum au maximum.

La **performance** de round-robin dépend fortement du **choix du quantum** de base.

Le RR est un algorithme d'ordonnancement **préemptif** se base sur le partage du temps entre tâches à travers une Ready Queue de type **FIFO** sans priorité associe au tâches.

Le temps d'allocation du CPU à la tâche choisie par RR est **préemptif**.

Le modèle utilisé par RR est : tâche **(r0, C, O)**.

- **r0** : date du premier réveil.
- **C** : capacité (sa durée d'exécution).
- **O**.: Ordre de tâche

Temps de charge du noyau : Négligeable.

On n'a pas de délai critique (D = ∞).

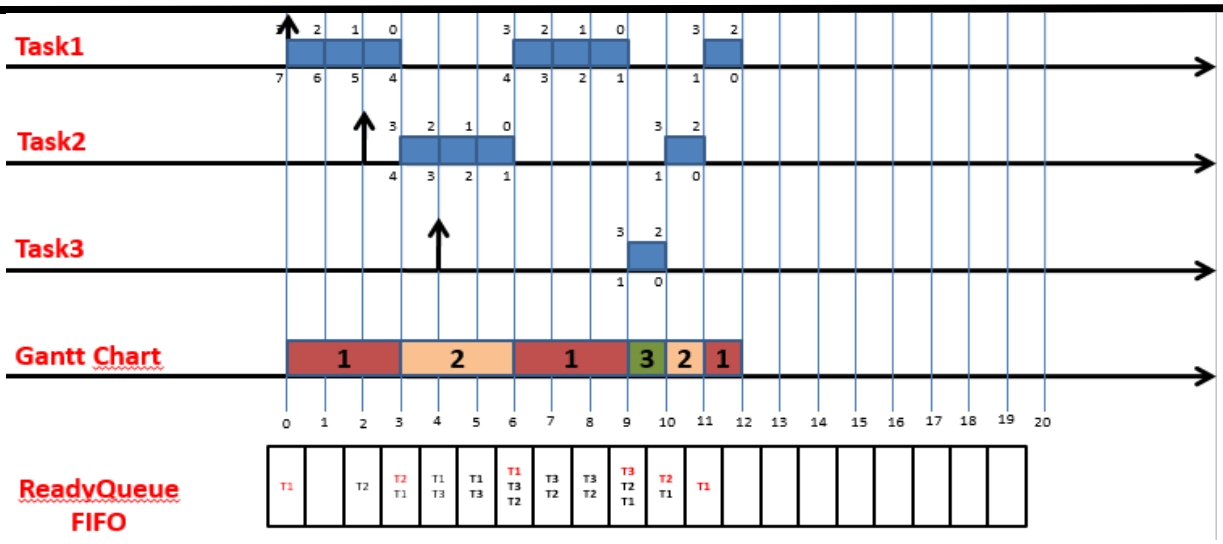


Figure 1.13 : Exemple sur le Round Robin (RR).

II.4. Tâches sporadiques :

Les tâches sporadiques sont des tâches récurrentes avec des contraintes strictes d'échéance.

Une tâche sporadique est définie par un paramètre C et une durée minimum T entre deux requêtes successives de la tâche. Le modèle peut être généralisé de la même façon que le modèle de tâche périodique. [16]

Conclusion :

Dans ce chapitre on a présenté s différents types de système temps réel strict/ temps réel souple/ temps réel ferme, ensuite on a vu les caractéristiques du système temps réel qui sont : contraintes du temps ; correction ; embarqué ; sécurité ; concurrence ; distribué et stabilité.

Après on a traité les différentes méthodes du système temps réel (méthodes fonctionnelles structures / méthodes orientés objets / méthodes orientés composant).

On a ensuite cités le domaine d'applications des systèmes temps réel et pour finir son ordonnancement (tâches périodiques / apériodiques/ sporadiques).

Chapitre II
Conversion du LADDER/GRAFCET à un algorithme informatique

II.1. Introduction

On va voir dans ce chapitre l'importance des automates ainsi que les différents types d'automates, son architecture et le principe de fonctionnement.

Nous allons aussi traiter les langages d'automatisation (ladder / Grafcet) et leurs impact sur notre domaine d'application.

II.1.1.Les automates :

L'automate programmable industriel API est aujourd'hui le constituant le plus réalisé des automatismes. On le trouve pratiquement dans tous les secteurs de l'industrie car il répond à des besoins d'adaptation et de flexibilité pour un grand nombre d'opérations.

Cette émergence est due en grande partie, à la puissance de son environnement de développement et aux larges possibilités d'interconnexions.

II.1.2. Historique

Les automates programmables industriels (API) sont apparus à la fin des années soixante aux Etats Unis, à la demande de l'industrie automobile américaine (General Motorsen leader), qui réclamait plus d'adaptabilité de leur systèmes de commande.

Les couts de l'électronique permettant alors de remplacer avantageusement les technologies actuelles. [18]

II.1.3. Définition :

API (Automate Programmable Industriel) ou en anglais PLC (Programmable Logic Controller) c'est un appareil électronique (matériel, logiciel, processus, un ensemble des machines ou un équipement industriel) destiné à la commande de processus industriels par un traitement séquentiel (Il contrôle les actionneurs grâce à un programme informatique qui traite les données d'entrée recueillies par des capteurs). Qui comporte une mémoire programmable par un utilisateur automaticien (et non informaticien) à l'aide d'un langage adapté (Le langage List, Le langage Ladder…etc) pour le stockage interne des instructions donnée pour satisfaire une objectif donnée. Automate permet de contrôler, coordonner et d'agir sur l'actionneur comme par exemple un robot, un bras manipulateur alors en peut dire API utilisé pour

automatiser des processus. L'API est structurée autour d'une unité de calcul (processeur), de cartes d'entrées-sorties, de bus de communication et de modules d'interface et de commande. [19]

II.1.3.1. Objectif de l'automatisation :

L'automatisation permet d'apporter des éléments supplémentaire à a la valeur ajoutée par le système.

- Accroitre la productivité du système (augmenter la quantité de produit).
- Améliorer la flexibilité de production.
- Améliorer la qualité du produit.
- Adaptation à des contextes particuliers.
- Adaptation à des environnements hostiles pour l'homme (milieu marin, spatial, nucléaire,…).
- Adaptation à des taches physique ou intellectuelles pénibles pour l'homme.
- Augmenter la sécurité, … etc. [20]

II.1.4. Nature des informations traitées par l'automate : [18]

- Tout ou rien (T.O.R.)
- Analogique
- Numérique

II.1.5. Structure d'un système automatisé : [21]

Un API constitué essentiellement **:** d'une unité centrale, d'un module d'entrée (ou interface d'entrées), d'un module de sortie (ou interface de sortie), d'un coupleur de communication.

- D'une console de programmation ou autre périphérique.

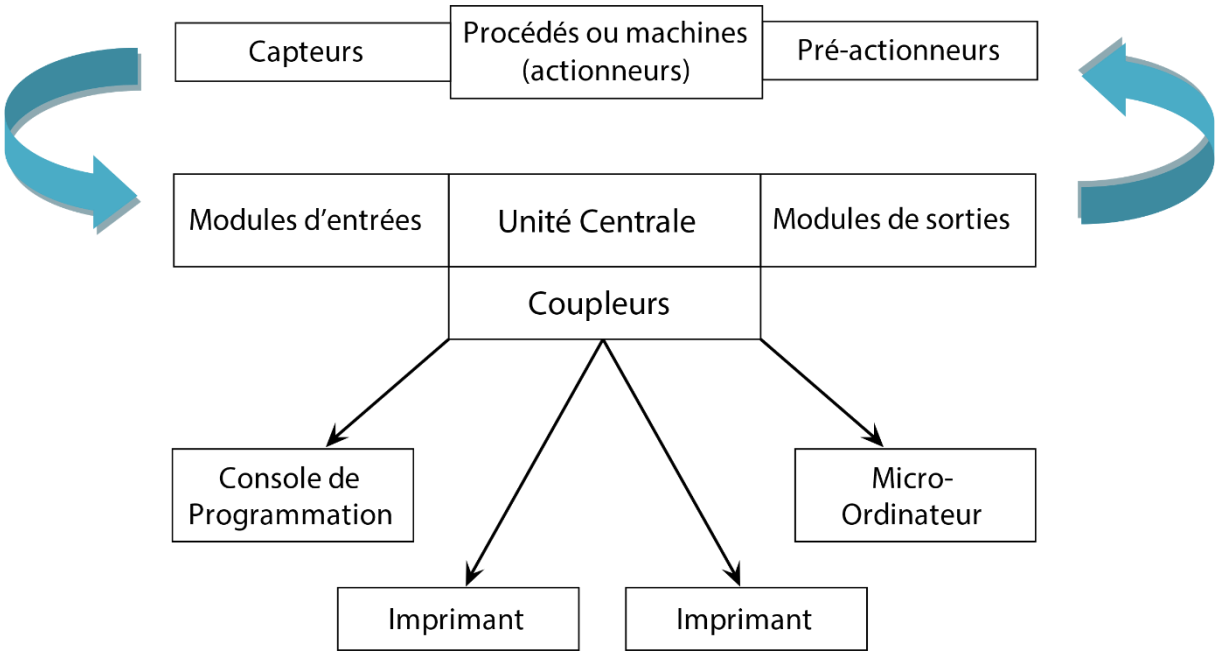


Figure 2.1 : Structure d'un système automatisé.

- **L'unité centrale (UC) :** C'est le cœur de la machine, comporte le(s) processeur(s) (unité de traitement logique ou numérique et la mémoire(s).
- **Le module des entrées (ou interface d'entrées) :** Il permette de raccorder à l'automate les différents capteurs.
- **Le module des sorties (ou interface de sortie) :** Il permette de raccorder à l'automate les différents pré-actionneurs.
- **Le Coupleur :** Ce sont des cartes électroniques qui assurent la communication entre les périphériques (Modules d'E/S ou autres) et l'unité centrale.

En général, les échanges entre l'UC et les modules d'E/S s'effectuent par l'intermédiaire d'un bus interne.

- **Les consoles :** Il existe deux types de console :

Console d'exploitation : permet le paramétrage et les relevés d'informations (modification des valeurs et visualisation).

Console de programmation, réglage et exploitation : Cette dernière effectue dans la phase de programmation : L'écriture, la modification, l'effacement et le transfert d'un programme dans la mémoire de l'automate ou dans une mémoire REPROM.

II.1.6. Différents types de l'automate programmable industriel :

Aspect extérieur : Les automates peuvent être de type compact ou modulaire.

- Automate de type compact
- Automate de type modulaire

II.1.7. Architecture interne d'un automate programmable :

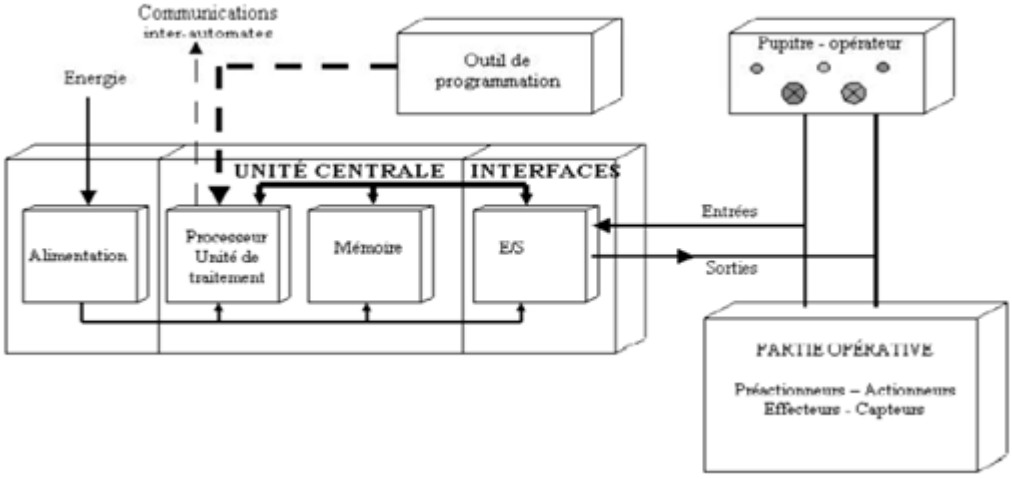


Figure 2.2 : Structure interne de l'API.

Un automate programmable est constitué essentiellement de 5modules :

1. L'unité centrale : contient : un processeur et une mémoire

2. Le module d'entrée : peut être constitué de cartes d'entrées logiques ou cartes d'entrées analogiques

3. Le module de sorties : peut être constitué des cartes de sorties logiques des cartes de sortie analogiques

4. Le module d'alimentation

5. Le module de communication : Les consoles, les boîtiers de tests et les unités de dialogue en ligne.

II.1.9. Principe de fonctionnement d'un automate programmable industriel :

L'automate programmable fonctionne par déroulement cyclique du programme. Le cycle comporte trois opérations successives qui se répètent normalement comme suit :

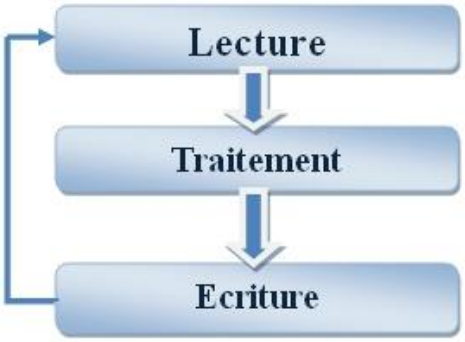


Figure 2.3 : Principe de fonctionnement d'un API.

Phase1 : Lecture (Photographie des entrées).

Phase 2 : Traitement (exécution de programme).

Phase 3 : Ecriture (mise à jour des sorties).

II.2. Les langages d'automatisme LADDER et GRAFCET :

II.2.1. Introduction :

L'écriture d'un programme consiste à créer une liste d'instructions permettant l'exécution des opérations nécessaires au fonctionnement du système. Il existe différents types de langage de programmation, nous citons : le langage GRAFCET, langage à contact (Ladder), langage LIST et langage booléen (Logigramme).

L'API traduit le langage de programmation en langage compréhensible directement par le microprocesseur. Ce langage est propre à chaque constructeur, il est lié au matériel mis-en œuvre.

II.2.2. Langage GRAFCET :

II.2.2.1GRAFCET (Graphe Fonctionnel de Commande Étapes-Transitions). Le Grafcet est un diagramme fonctionnel ; il représente par un graphe le fonctionnement de la partie opérative, donc les actions effectuées par le système. Il nous servira ensuite à décrire le fonctionnement de la partie commande, c'est-à-dire la technologie employée pour commander les actionneurs. [18]

II.2.2.2. Domaine d'application :

Utilisé industriellement, le GRAFCET est aussi enseigné dans les options techniques et l'enseignement supérieur.

Depuis les premières publications le concernant et surtout depuis la norme française NFC03-190 de 1982, cet outil a été travaillé et enrichi par le groupe systèmes logiques de l'AFCET (Association Française pour la Cybernétique Economique et Technique).

Il existe une documentation et symboles graphiques, diagramme fonctionnel "Grafcet «éditée par l'Union Technique de l'Electricité. UTE C03-190 Nov. 1990. [19]

Principe du Grafcet :

Pour visualiser le fonctionnement de l'automatisme, le GRAFCET utilise une succession alternée d'ETAPES et de TRANSITIONS.

A chaque étape correspond une ou plusieurs actions à exécuter. Une étape est soit active, soit inactive. Les actions associées à cette étape sont effectuées lorsque celle-ci est active.

Les transitions indiquent avec les LIAISONS ORIENTEES, les possibilités d'évolution entre étapes. A chaque transition est obligatoirement associée une condition logique pouvant être vraie ou fausse. Cette condition de transition est appelée RECEPTIVITE. L'évolution d'une étape à une autre ne peut s'effectuer que par le franchissement d'une transition.

Une transition ne peut être franchie, donc activer l'étape suivante que :

- si elle est validée par l'étape antérieure active.

- et que les conditions de réceptivité soient satisfaites.

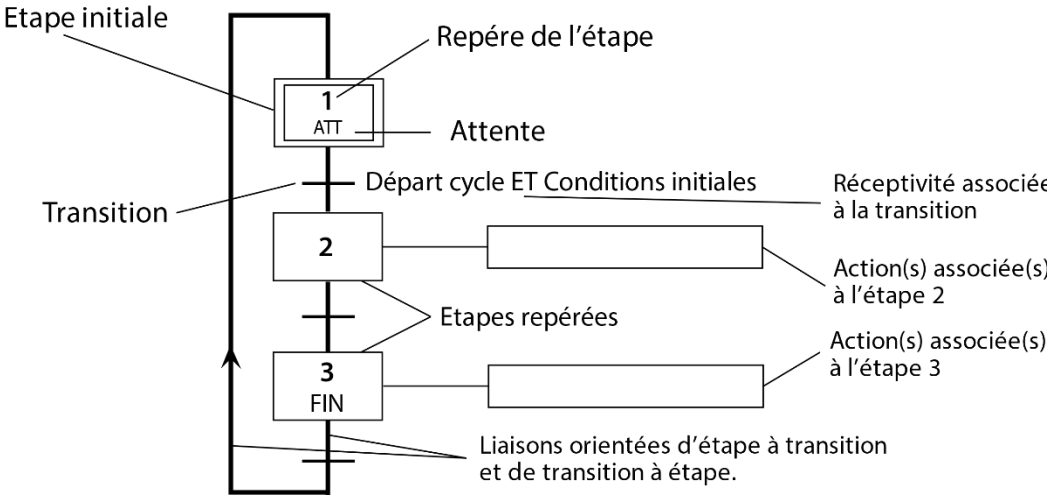


Figure 2.4 : Principe du GRAFCET.

II.2.3. Langage CONTACT (LADDER) :

Le langage à contacts (LD : Ladder Diagramme) est composé de réseaux lus les uns à la suite des autres par l'automate. Ces réseaux sont constitués de divers symboles représentant les entrées/sorties de l'automate, les opérateurs séquentiels (temporisations, compteurs, …), les opérations, ainsi que les bits systèmes internes à l'automate (ces bits permettent d'activer ou non certaines options de l'automate, telle que l'initialisation des grafcets). [18]

II.2.3.2. Les symboles utilisés :

Il existe 3 types d'élément de langage :

• les entrées (ou contact), qui permettent de lire la valeur d'une variable booléenne,

• les sorties (ou bobines) qui permettent d'écrire la valeur d'une variable booléenne,

• les blocs fonctionnels qui permettent de réaliser des fonctions avancées.

Le tableau 2.1 donne les principaux éléments (contacte et bobines) d'un réseau LD.

Tableau 2.1 : les principaux éléments (contacte et bobines) d'un réseau LD.

Object graphique	Nom
-| |-	Contact normalement ouvert
-|/|-	Contact normalement fermé
-|P|-	Contact fermé au front montant
-|N|-	Contact fermé au front descendant
-( )-	Bobine normalement ouverte
-(/)-	Bobine normalement fermée
-(S)- ou–(L)-	Bobine Latch (maintenu à 1 une fois actionné)
-(R)-ou –(U)-	Bobine Reset (remise à 0 de la bobine latch)
-(P)-	Bobine active au front montant de son entrée
-(N)-	Bobine active au front descendant de son entrée
|-<return>	Retour inconditionnel (vers le sous-programme appelant)
|-cond-<return>	Retour conditionnel
|->>Label	Saut inconditionnel
|-cond->>Label	Saut conditionnel

II.2.3.3. Structure d'un réseau de contacts :

Un réseau de contacts se compose de la manière suivante : étiquette (ou titre) +commentaire + réseau graphique (zone test + zone action).

La zone de test accueille :

- les contacts.

- les blocs fonction (temporisations, compteurs, …).

- les blocs comparaison.

La zone action accueille :

- les bobines.

- les blocs opérations [18].

II.2.3.4. Règles d'évolution d'un réseau de contacts :

La lecture d'un réseau se fait réseau connexe par réseau connexe (de haut en bas), puis de gauche à droite à l'intérieur d'un réseau connexe.

Un réseau connexe est constitué d'éléments graphiques tous reliés entre eux, mais indépendants des autres éléments graphiques du réseau.

Si l'on rencontre une liaison verticale de convergence, on évalue d'abord le sous-réseau qui lui est associé (toujours dans la même logique) avant de continuer l'évaluation du sous réseau qui l'englobe.

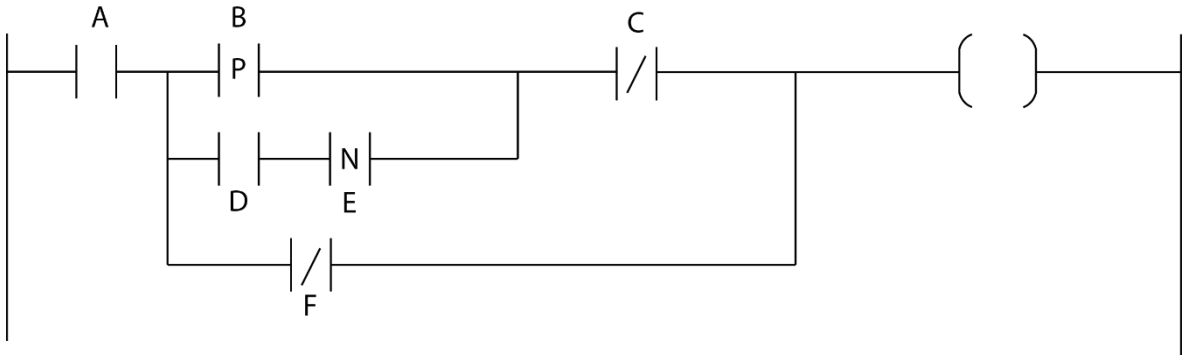


Figure 2.5 : Exemple d'un schéma Ladder.

II.3.1.Problématique :

Il est nécessaire pour l'entreprise de garantir que les automates assurent un bon fonctionnement du procédé, mais parfois les API ne méritent pas être utilisés dans des petits projets ; Citant aussi, le cout élevé des outils de ces derniers, des logiciels et de leurs licences d'utilisation. Autrement dit les programmes déployés, doivent être fiables et performants car un dysfonctionnement réduis même momentanément la production. D'autre part, une modification au niveau du programme pour définir des nouveaux besoins est aussi très couteuse. Compte tenu des difficultés inhérentes à la tâche des automates, l'aide informatique est donc indispensable.

II.3.2.Objectif et solution :

Le travail de recherche réalisé dans ce mémoire a pour but de proposer une solution qui est l'utilisation des systèmes embarqués à base de **µC** « La plateforme Arduino ou Atmel», Plus, la conversion du Ladder / Grafcet en algorithme.

II.3.3.Présentation de la plateforme Arduino : [22]

A cause de la révolution technologique à travers le hardware/software en source libre (OPEN SOURCE), le développement des applications deviennent simples et faciles à faire ; ce qui ne nous donne à la fin aucune excuse pour monter sur la vague technologique et sortir de la mentalité consommateur vers la mentalité constructeur.

II.3.4.La philosophie derrière l'ARDUINO :

1. Le matériel est « open hard » : On peut le copier, le fabriquer et le modifier librement.
2. Le logiciel est libre « open source » : On peut l'utiliser et le modifier librement.
3. Sur Internet, on trouve : un interface IDE « arduino.cc », des guides d'utilisation, des exemples et des tutoriaux et des forums.

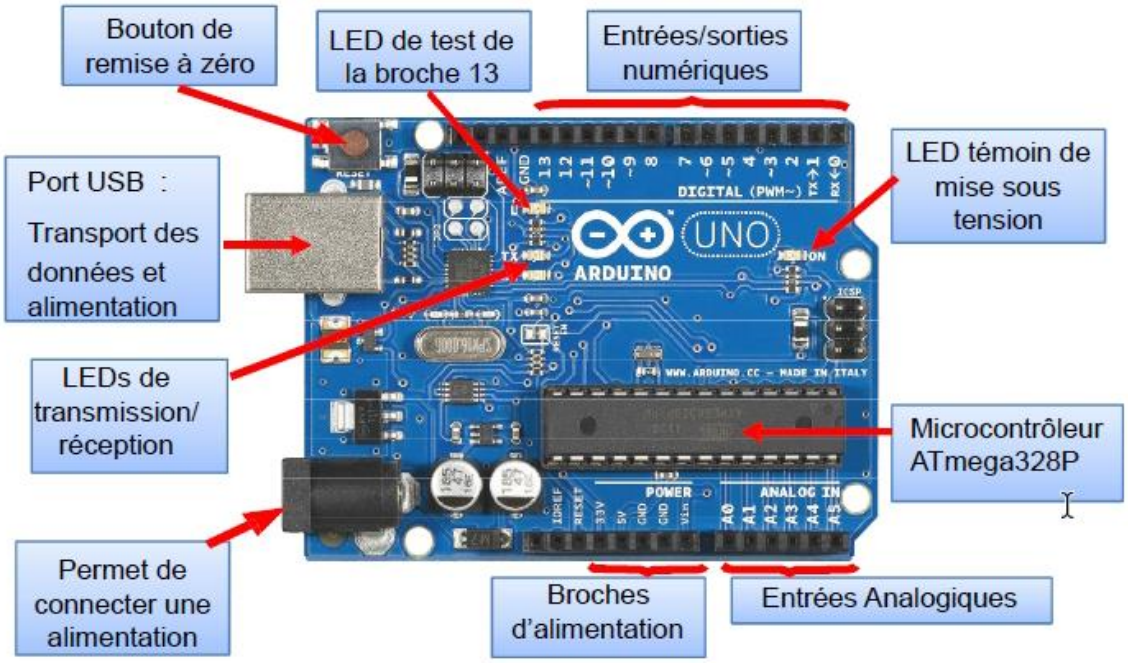


Figure 2.6 : Presentation de La carte Arduino - Uno.

II.3.5.Les principales caractéristiques de la carte ARDUINO-UNO :

• La carte "ARDUINO UNO" dispose de 14 broches (numérotées de 0 à 13) qui peuvent être configurées en "entrées digitales" ou en "sorties digitales" individuellement, susceptibles de délivrer une intensité maximale de 40 mA sous une tension égale à 5V.

• Certaines de ces broches peuvent être configurées en "sorties PWM" (Pulse Width Modulation ou modulation en largeur d'impulsion MLI) (pin3, pin5, pin6, pin9, pin10, pin11).

• Elle possède également 6 entrées analogiques (notées A0 à A5) permettant de mesurer des tensions comprises entre 0V et 5V grâce à un convertisseur ADC à 10 bits.

• Elle possède aussi un convertisseur USB-RS232TTL pour flasher le µC et pour communiquer avec le terminal.

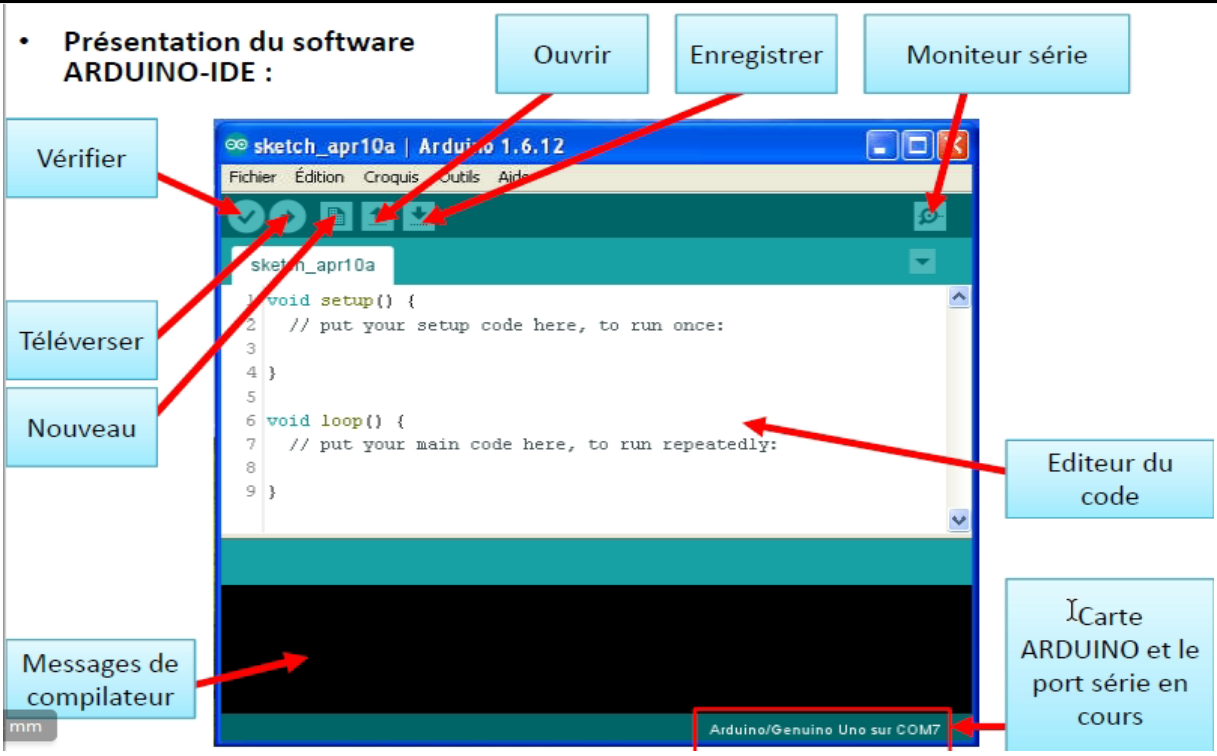


Figure 2.7 : Présentation du software Arduino IDE.

II.3.7.Explication des étapes à suivre pour l'utilisation de la carte ARDUINO-UNO :

• Cette démonstration présente la simplicité d'utilisation de la carte ARDUINO-UNO à travers un exemple qui se trouve dans la série d'exemples installés dans l'ARDUINO-IDE.

• Étapes à suivre :

1. Connecter la carte ARDUINO-UNO au PC à travers le port USB.

2. Lancer le logiciel ARDUINO-IDE.

3. Aller vers le menu « Fichier..Exemples..Basics..Blink »

4. Aller vers le menu « Outils..Type de carte..Arduinouno » pour configurer la carte qu'on va utiliser.

5. Aller vers le menu « Outils..Port série » et sélectionner le port COM de la carte Arduino UNO (configuration de la communication pour le flashage par AVRDUDE et le monitoring).

6. Cliquer sur le bouton « Téléverser ».

II.3.8.Conversion du Ladder à un algorithme informatique :

La conversion d'un Ladder à un langage informatique d'une manière plus précise doit passer par les étapes suivantes :

-Procéder à la mise en place du processus par l'avis d'un automaticien en Ladder.

-Procéder à la conversion du Ladder vers un Schéma logique.

- Noter les signaux internes sur le schéma logique pour connaitre le nombre des variables internes (mémentos).

-La conversion :

1) Relier à chaque entrée une variable.

2) Relier à chaque mémento une variable.

3) Relier à chaque sortie.

4) Faire l'algorithme suivant :

A) Initialiser les variables de sortie (l'initialisation des mémentos reste optionnelle à l'exception des bascules).

B) Boucler infiniment sur le cycle automate :

B-1) La mise à jour des sorties par le contenu des variables de sortie (**mise à jour des sorties**).

B-2) La mise à jour des variables d'entrée par les entrées de système (**lecture des entrées**).

B-3) Réalisation des différentes équations logiques en commençant par les mémentos les plus internes vers les variables de sortie (**Traitement**).

Exemple 1 : Logique Combinatoire ' XOR '

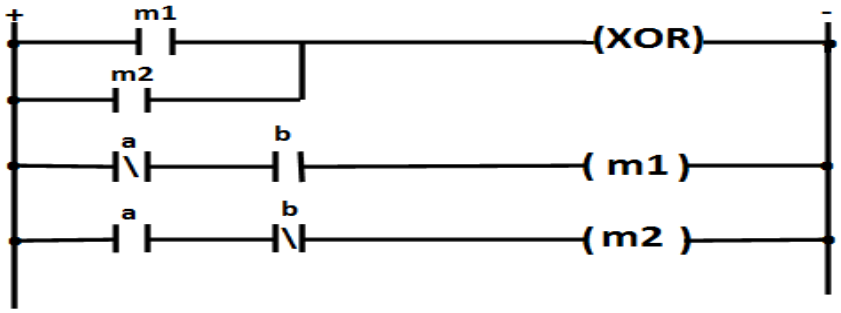


Figure 2.8 : Schéma Ladder avec memonto.

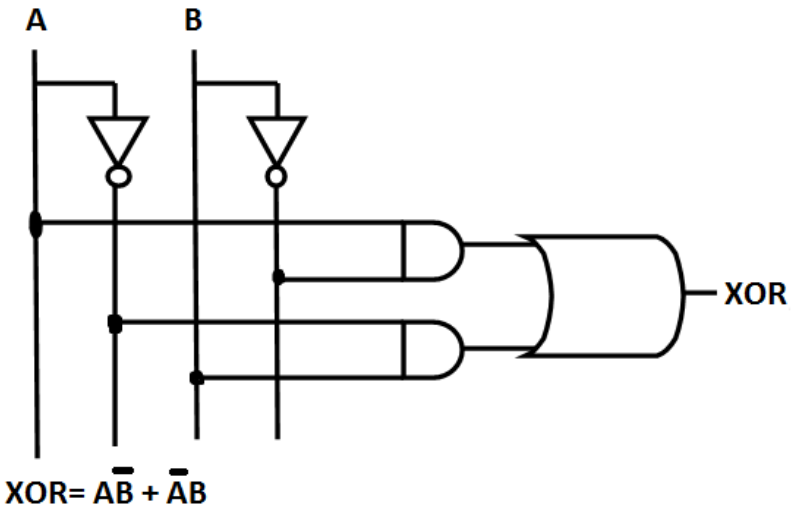


Figure 2.9 : Conversion en schéma logique.

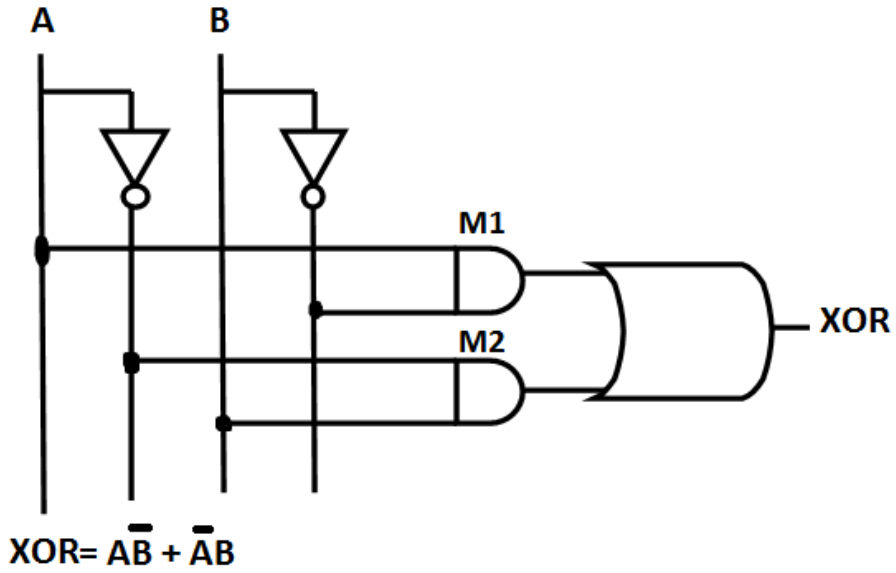


Figure 2.10 : Notation des signaux internes.

- D'une façon générale, associer à chaque signal interne et externe une variable.
- **Les variables d'entrée :**{varA, varB}.
- **Les variables internes (mementos) :** {M1, M2}.
- **Les variables de sortie :** {varX}.

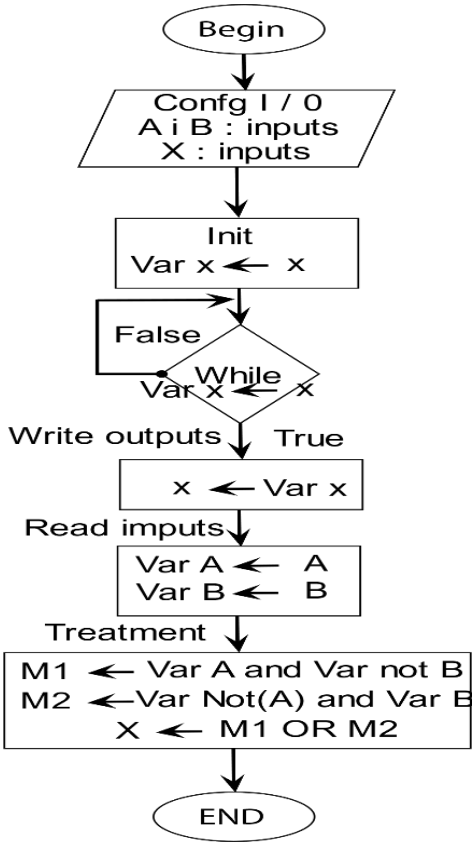


Figure 2.11: Organigramme de Ladder 'XOR'.

Algorithme :

// Initialisation
var X ←0 ;
// Cycle automate
While (true)
// La mise à jour des sorties
X←varX ;
// La mise à jour des variables d'entrée
varA←A ;varB ←B ;
// Traitement
M1← varA and varnot(B) ; M2← varnot(A) and varB
X←M1 or M2
End while
End

II.3.9.Utilisation d'un programme industriel pour traiter le problème :

Le programme suivant qui fait le perçage des pièces après fabrication :

1) Il faut un bouton START pour démarrer le cycle.

2) Il faut deux fins de course pour localiser la position de la perceuse.

3) Il fout un moteur pour déplacer verticalement la perceuse (du bas en haut ou du haut en bas) (deux signaux de commande) et un autre moteur pour tourner la mèche (un signal de commande).

4) Cahier des charges : Le système attend l'appui sur le bouton START (moteur mèche à l'arrêt), si le cas, la perceuse descend vers le bas jusqu'à la fin de course niveau bas. Puis le système donne l'ordre à la perceuse pour monter vers le haut jusqu'à la fin de couse niveau haut (cycle de perçage) (moteur mèche en marche) puis le système attend une autre fois l'appui sur le bouton START pour démarrer un autre cycle.

Donc, le système qu'on va réaliser est un système à 3 entrées et 3 sorties comme indique la figure suivante :

Figure 2.12 : Système de contrôle/commande de la perceuse.

Le LADDER : le processus de la perceuse est complètement séquentiel, donc, on va utiliser des bascules SR pour réaliser les différences étapes de la séquence de perçage.

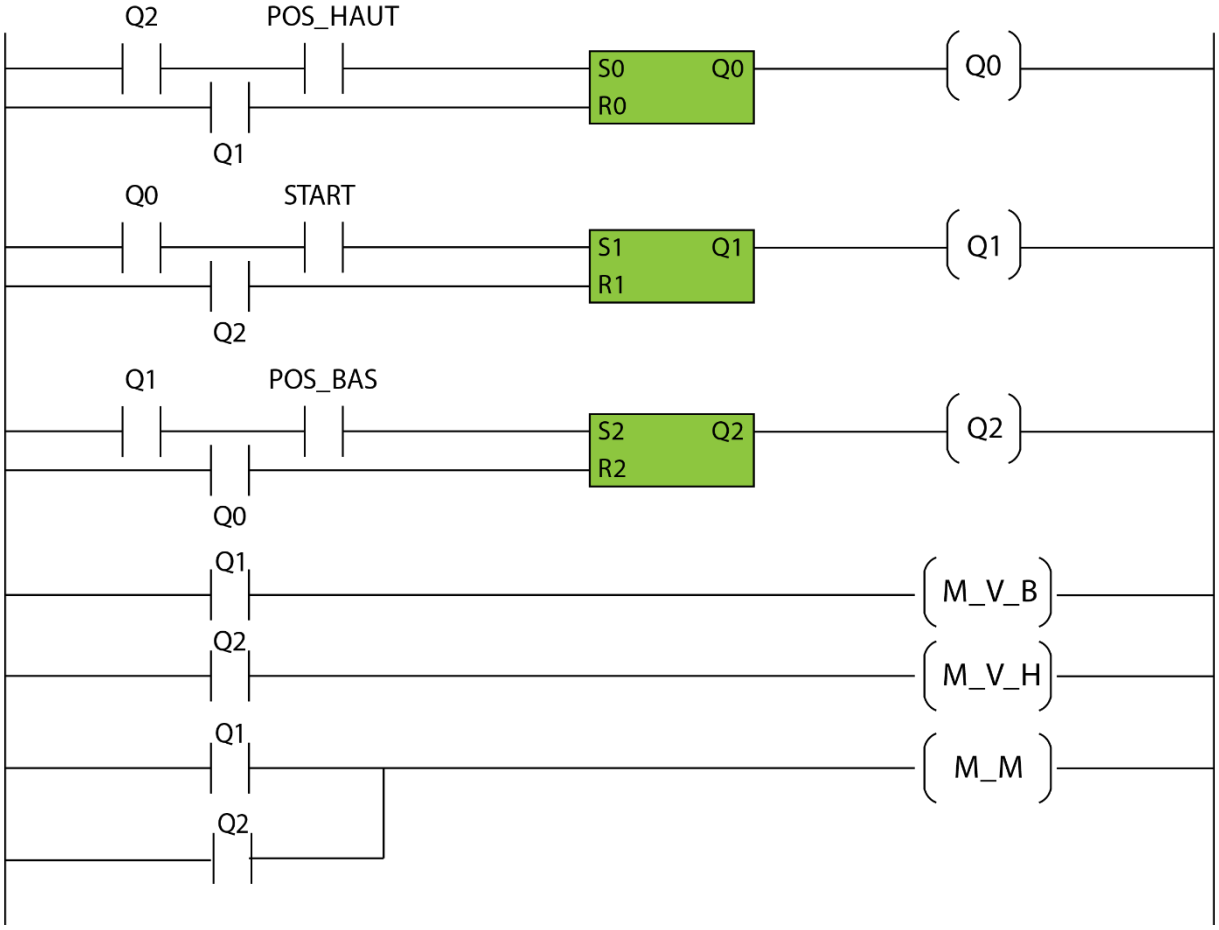


Figure 2.13 : Schéma Ladder avec memontos.

N.B : Nous avons considéré qu'au moment de démarrage de l'API, le code l'initialisation fait le SET de Q0 et le RESET du reste {Q1 et Q2}.

Exemple 2 : (Séquentielle en étapes)

-L'exemple de la perceuse (**Soyez prudent à cause de présence des bascules**):

-Conversion en schéma logique:

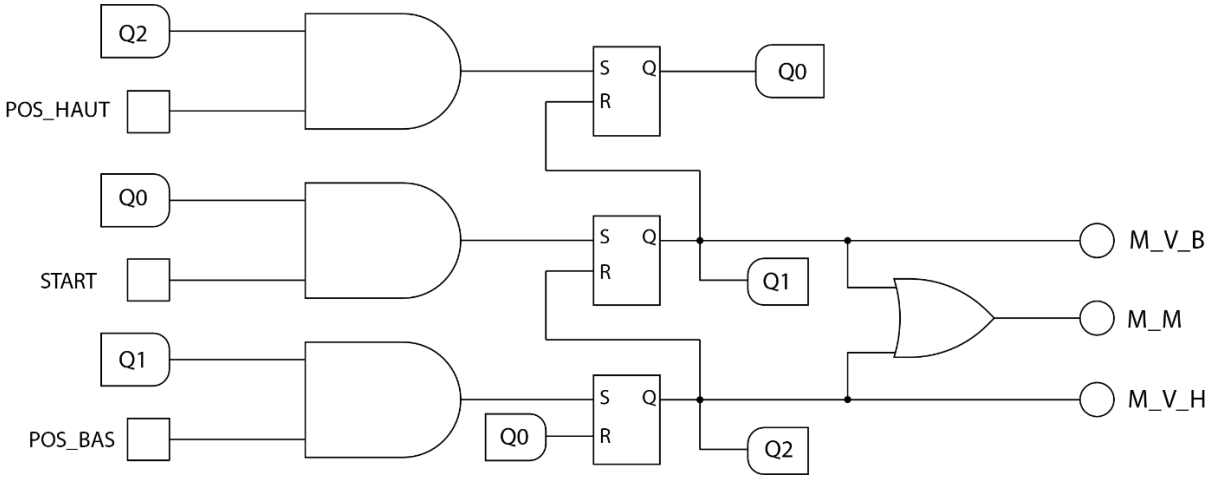


Figure 2.14 : Conversion en schéma logique.

-Notation des signaux internes :

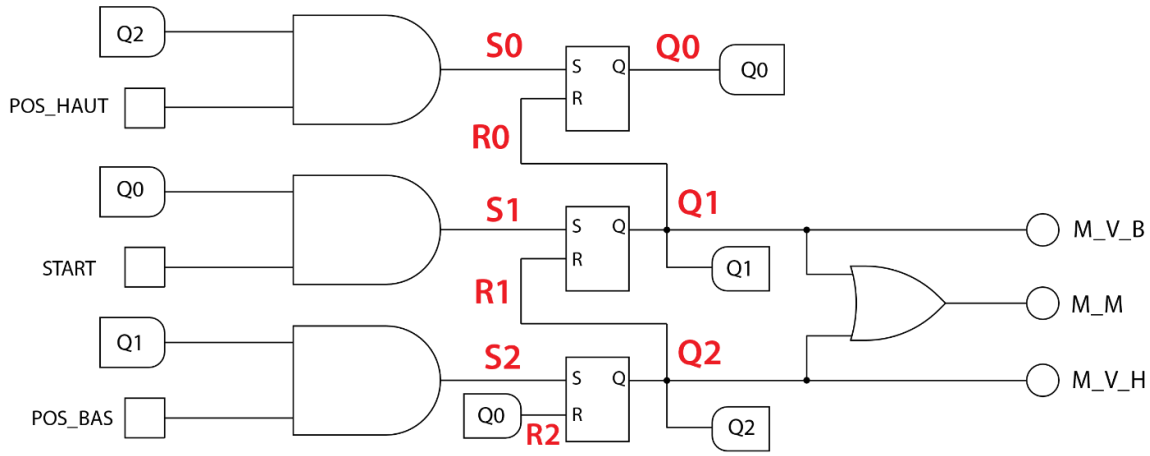


Figure 2.15 : Notation des signaux internes.

D'une façon générale, associer à chaque signal interne et externe une variable :

-**Les variables d'entrée** {varPH, varPB et varSTART}.

-**Les variables internes** (mémentos) {S0, S1, S2, R0, R1, R2, Q0, Q1 et Q2}.

-**Les variables de sortie** {varMVB, varMVHetvarMM}.

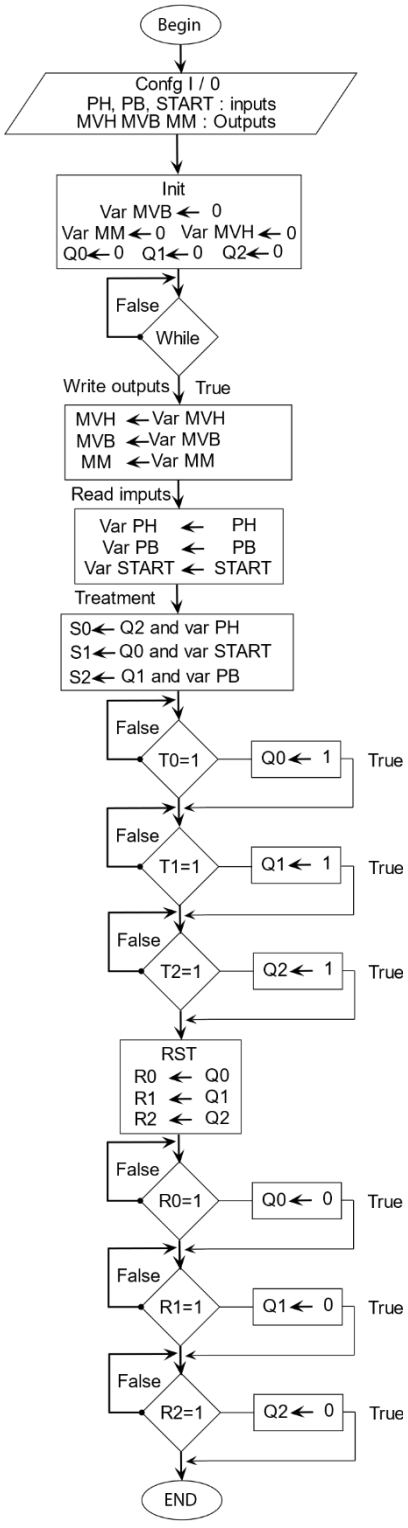


Figure 2.16 : Organigramme de Ladder ‘Perceuse’.

-Algorithme :

// Initialisation
varMVB←0;varMVH ← 0; varMM← 0;
// Faites attention à les bascules, il faut initialiser
Q0 ← 1;
Q1 ← 0;
Q2 ← 0;
// Cycle automate
While (True)
// La mise à jour des sorties
M_V_B ← varMVB;
M_V_H ← varMVH;
M_M ← varMM;
// La mise à jour des variables d'entrée
varPH←POS_HAUT;varPB ←POS_BAS; varSTART ← START;
// Le traitement (attention à les bascules, les SET puis les RESET)// les SET
S0 ← Q2 and varPH;
S1 ← Q0 and varSTART;
S2 ← Q1 and varPB;
If (S0=1) then Q0← 1; end if;
If (S1=1) then Q 1← 1; end if;
If (S2=1) then Q2← 1; end if;
// Les RESET
R0 ←Q1;
R1 ←Q2;
R2 ←Q0;
If (R0=1) thenQ0 ← 0; end if;
If(R1=1) then Q1 ← 0; end if;
If (R2=1) then Q2 ← 0; end if;
// Les variables de sortie
varMVB← Q1;varMVH ← Q2; varMM ←Q1 or Q2;
End while
End

Tout simplement, il faut utiliser la programmation modulaire, c'est d'assembler les lignes de code qui présentent une tâche bien précise comme :

-Une fonction pour initialisation,

-Une fonction pour la mise à jour des sorties,

-Une fonction pour lire les entrées du processus,

-Une fonction pour assurer le traitement.

II.3.10.Conversion du Grafcet à un algorithme informatique :

Dans notre projet : le GRAFCET du système est le suivant :

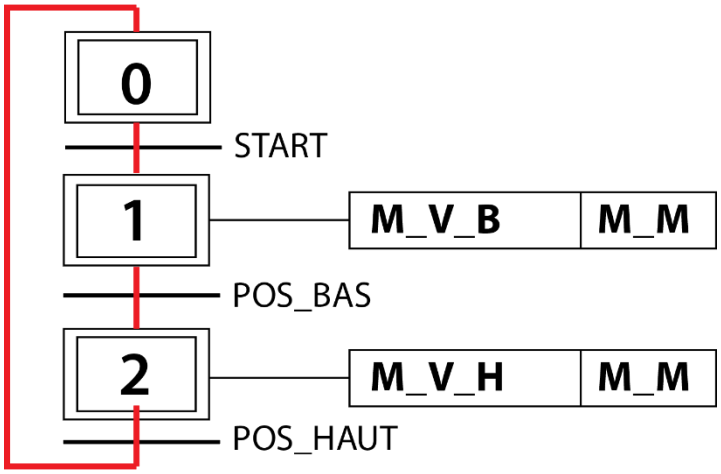


Figure 2.17 : Schéma Grafcet commande de perceuse.

La conversion d'un Ladder à un langage informatique d'une manière plus précise doit passer par :

-Procéder à la conception du processus par l'avis d'un automaticien en Grafcet.

-La conversion :

- Relier à chaque entrée une variable.
- Relier à chaque étape une variable.
- Relier à chaque transition une variable.
- Procéder à l'algorithme suivant :

1) Initialiser les étapes par marquage des étapes initiales et démarquage des étapes normales.

2) Boucler infiniment sur le cycle automate:

- La mise à jour des sorties par le contenu des étapes (**mise à jour des sorties**).
- La mise à jour des variables d'entrée par les entrées de système (**lecture des entrées**).
- Calcul des transitions par la réalisation des différentes équations logiques. (**Calcul des transitions**).
- La mise à jour des étapes sur la lumière de contenu des transitions (**mise à jour des étapes**).

L'exemple de la perceuse:

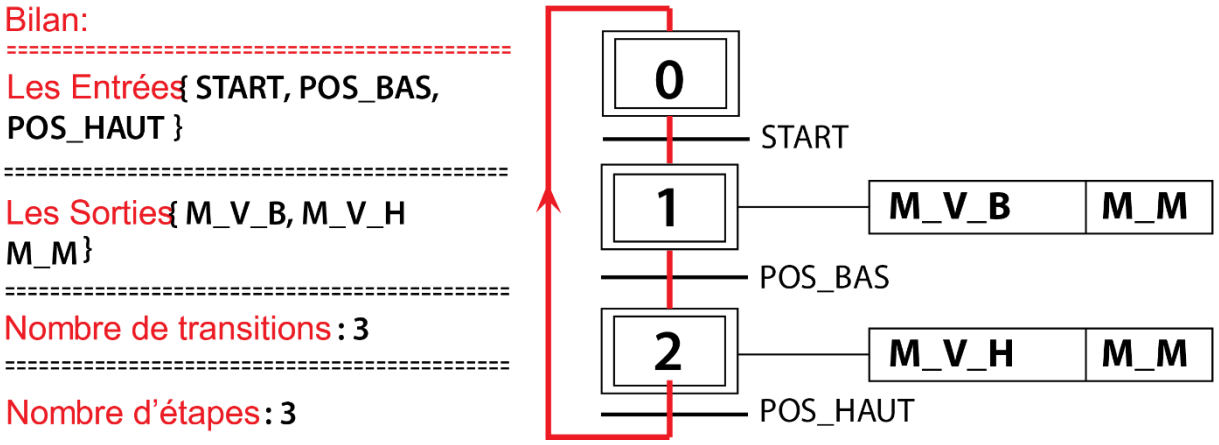


Figure 2.18 : Schéma Grafcet de la perceuse.

Les variables :

-Les variables d'entrée {varSTART, varPB et varPH}.

-Les variables des étapes {varE0, varE1 et varE2}.

-Les variables des transitions {varT0, varT1 et varT2}.

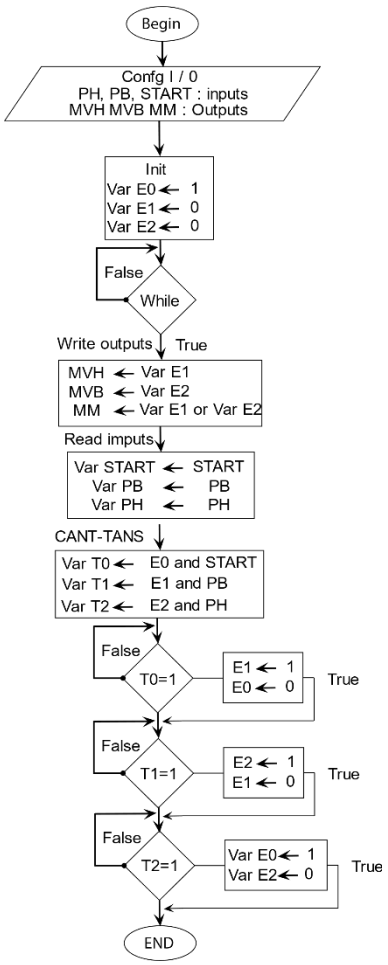


Figure 2.19 : Organigramme Grafcet de la perceuse.

Algorithme

// Initialisation des étapes
varE0 ←1;
varE1 ←0;
varE2 ←0;
// Cycle automate
While (True)
callupdateOutputs; **// la mise à jour des sorties**
callreadInputs; **// la lecture des entrées**
callcomputeTrans; **// le calcul des transitions**
callupdateEtapes; **// la mise à jour des étapes**
End While
-Le sous-programme 'updateOutputs' traitement sur sortie
M_V_H ← varE1;
M_V_B ← varE2;
M_M ← varE1 **or** varE2;
-Le sous-programme 'updateOutputs' traitement sur étape :
If (varE0 = 1) **then**
M_V_B← 0; M_V_H ←0; M_M← 0;
End If
If (varE1 = 1) **then**
M_V_B ←1; M_V_H← 0; M_M←1;
End If
If (varE2 = 1) **then**
M_V_B ←0; M_V_H ←1; M_M←1;
End If
-Le sous-programme 'readInputs':
varSTART← START;
varPB← POS_BAS;
varPH ←POS_HAUT;
-Le sous-programme 'computeTrans':
varT0← varE0 **and**varSTART;
varT1← varE1 **and** varPB;
varT2← varE2 **and** varPH;
-Le sous-programme 'updateEtapes':
If (varT0 = 1) **then**
varE1 1; varE0 ← 0;
End If
If (varT1 = 1) **then**
varE2← 1; varE1← 0;
End If
If (varT2 = 1) **then**
varE0←1; varE2←0;
End If.

Conclusion :

Nous avons vu dans ce chapitre l'impact des automates dans les différents domaines industriels ainsi que le principe de fonctionnement.

On conclue, après avoir fait la conversion du ladder / grafcet en algorithme informatique que l'utilisation des systèmes embarqués à base de **µC** « La plateforme Arduino ou Atmel», est bénéfique et moins couteuse.

Chapitre III
Décomposition d'une solution en LADDER/GRAFCET en tâches à travers une plateforme à µP OU µC

III.1.Introduction :

Lorsqu'on a un Ladder / Grafcet géant (une centaine des lignes / une centaine d'étapes) ; les fonctions de base de l'algorithme de conversion deviennent gênantes et difficiles à traiter et la mise au point devient aussi difficile à faire, pour cette raison, on a proposé une autre solution basé sur le même principe de la conversion mais en petit échelle.

Cette solution fait le découpage du Ladder / Grafcet en un ensemble de tâches, donc une conception d'un système multitâches. A travers l'exemple de la perceuse, on va expliquer comment faire la conversion en tâches

III.2. Décomposition d'une solution LADDER en tâches :

On présume que chaque ligne est une tâche ;

On fait l'algorithme suivant :

a) Initialiser les variables de sortie (l'initialisation des mémentos reste optionnelle à l'exception des bascules).

b) Boucler infiniment sur le cycle automate :

-La mise à jour des sorties par le contenu des variables de sortie (**mise à jour des sorties**).

-La mise à jour des variables d'entrée par les entrées de système (**lecture des entrées**).

-Réalisation des différentes équations logiques en commençant par les mémentos les plus internes vers les variables de sortie (**Traitement**).

On suit ces étapes dans les cas normaux, par contre, si on a des variables internes (mémentos) dans la ligne (tâches), on néglige la mise à jour des sorties.

Exemple 1 : d'un système Combinatoire: 'XOR'

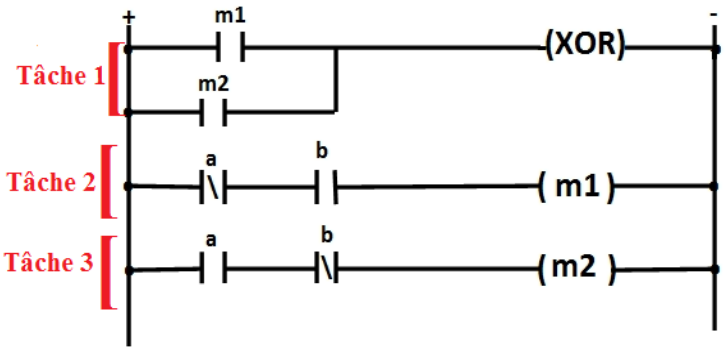


Figure 3.1 : Schéma Ladder avec tâches 'XOR'.

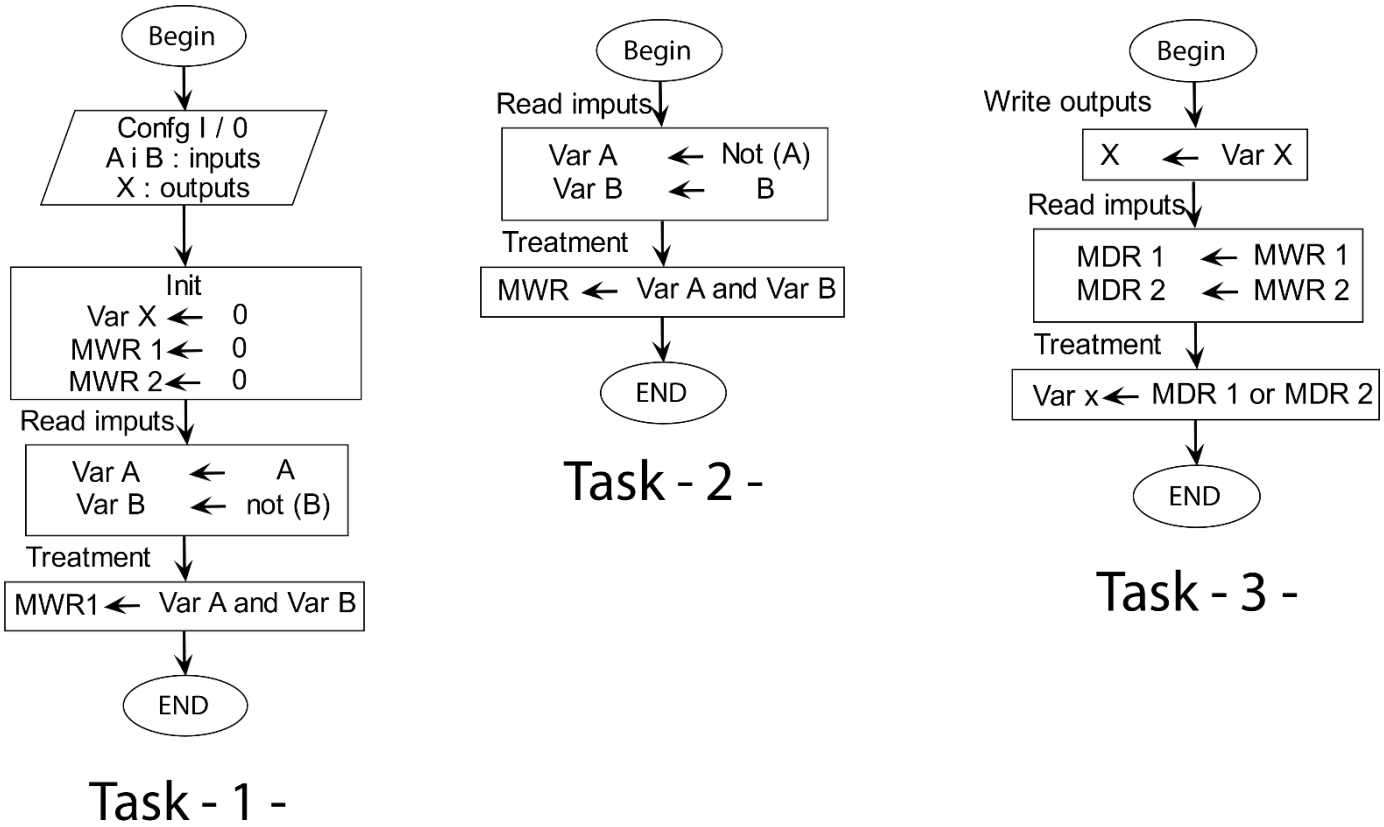


Figure 3.2 : Organigramme de Ladder avec tâches 'XOR'.

Solution de conversion :
// Initialisation
varX ← 0
MWR1 ← 0
MWR2 ← 0
// Définition des fonctions
Void Task 1() bgin
// Write outputs
// Void (because mementos)
// Read inputs
varA ←A
varB ← not(B)
// Treatment
MWR1 ← varA and varB
End
Void Task2 () bgin
// Write outputs
// Void (because mementos)
// Read inputs
varA ← not(A)
varB ← B
// Treatement
MWR2 ← varA and varB
End
Void Task3 () bgin
// Write outputs
X ← varX
// read inputs
MDR1 ← MWR1
MDR2 ← MWR2
varX ← MDR1 or MDR2
End

Exemple 2 : cas de la perceuse (séquentielle en étape)

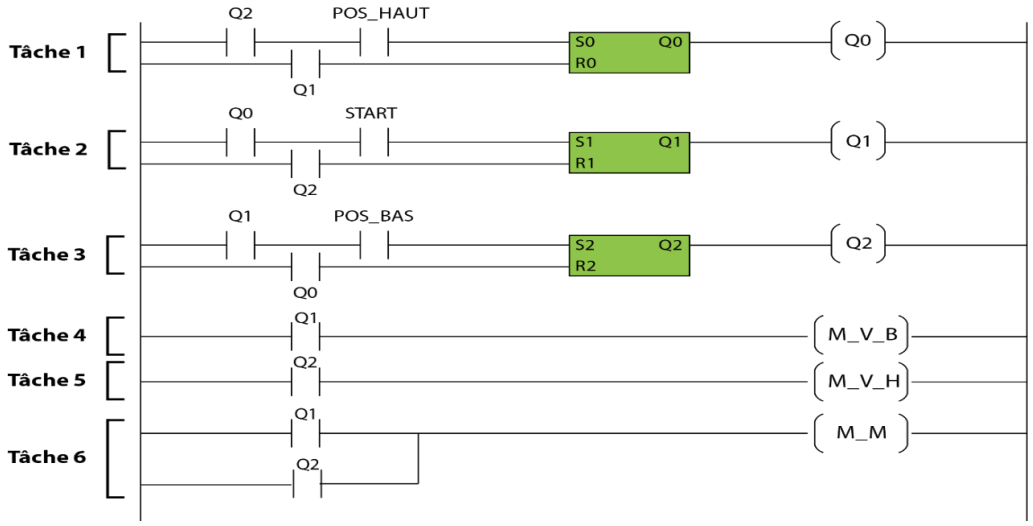


Figure 3.3 : Organigramme Ladder avec tâches de la perceuse.

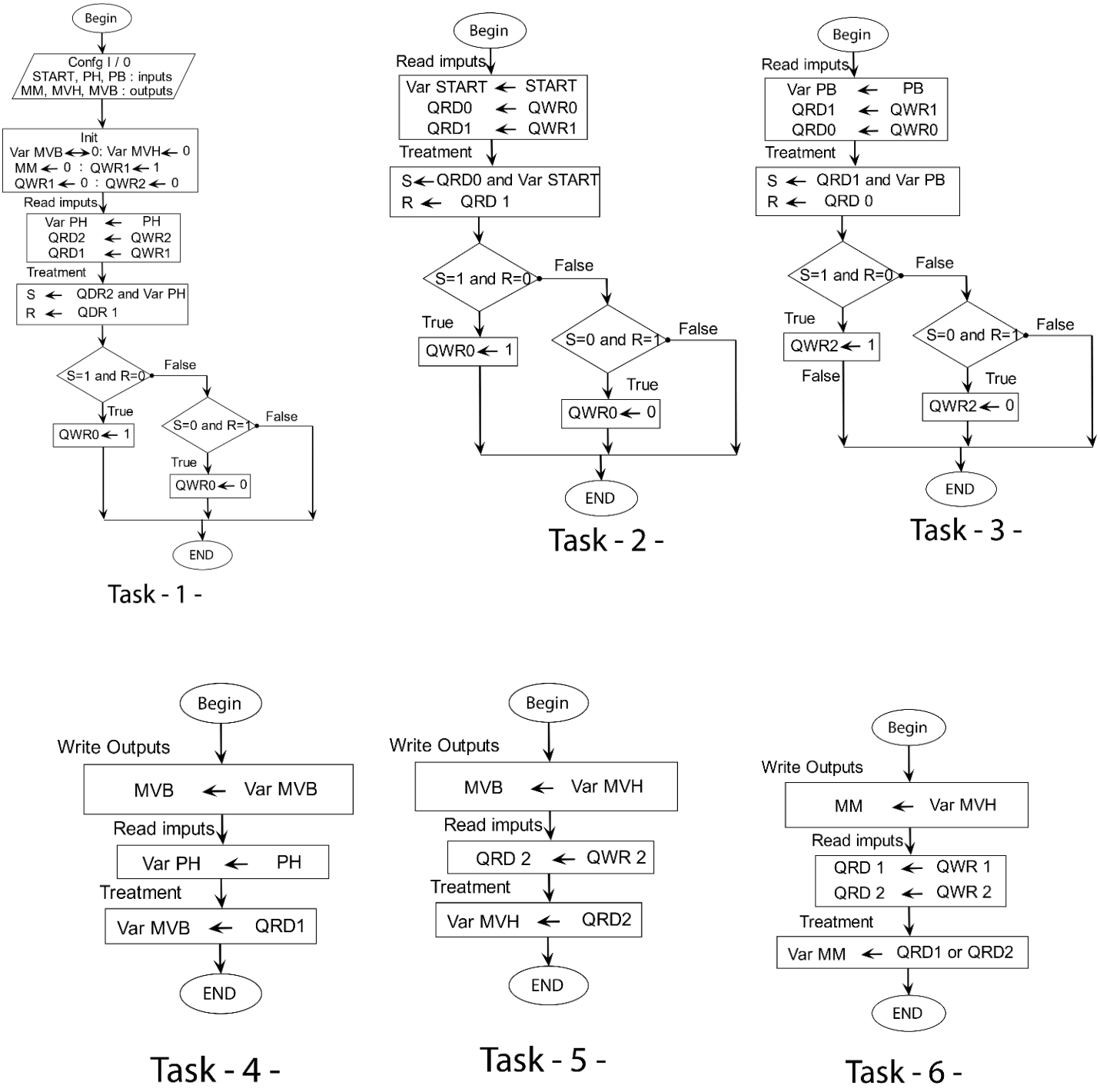


Figure 3.4 : Organigramme de Ladder avec tâches de la perceuse.

Solution de conversion :

// Initialisation
VarMVB←0
VarMVH←0
VarMM ← 0
QWR ←1
QWR ←0
QWR ←0
//Définition des fonctions
Void Task1 () bgin
// WR_OUTS
// void (because memento)
// RD_INS
varPH ← PH
QRD2 ←QWR2
QRD1 ←QWR1
// TRT
S ←QRD2 and varPH
R ←QRD1
If (S=1 and R=0) then
QWR0 ← 1
Else if (S=0 and R=1) then
QWR0← 0
End if
End
Void Task2 () bgin
// WR_OUTS
// void (because memonto)
// RD_INS
VarSTART ← START
QRD0 ←QWR0
QRD1 ←QWR2
// TRT
S← QRD0 and varSTART
R ←QRD2
If (S=1 and R=0)thene
QWR0←1
Else if (S=0 and R=1)then
QWR1←0
End if
End
Void Task3 () bgin
// WR_OUTS
// void (because memonto)
// RD_INS
varPB←PB
QRD1 ←QWR1
QRD0 ←QWR0
// TRT
S ←QRD1 and varPB

R ←QRD0
If (S=1 and R=0) then
QWR2 ←1
Else if (S=0 and R=1) then
QWR2 ←0
End if
End
Void Task4 () bgin
// **WR_OUTS**
MVB ←varMVB
// **RD_INS**
QRD1 ←QWR1
// **TRT**
varMVB← QRD1
End
Void Task5 () bgin
// **WR_OUTS**
MVH ←varMVH
// **RD_INS**
QRD2 ←QWR2
// **TRT**
varMVH ←QRD2
End
Void Task6 () bgin
// **WR_OUTS**
MM← varMM
// **RD_INS**
QRD1 ←QWR1
QRD2 ←QWR2
// **TRT**
VMM ←QRD1 or QRD2
End

III.3. Décomposition d'une solution GRAFCET vers tâches :

On présume que chaque étape et transition ou bien chaque transition et étape, sont des tâches.

Procéder à l'algorithme suivant :

1) Initialiser les étapes par marquage des étapes initiales et démarquage des étapes normales.

2) Boucler infiniment sur le cycle automate :

- La mise à jour des sorties par le contenu des étapes (**mise à jour des sorties**).
- La mise à jour des variables d'entrée par les entrées de système (**lecture des entrées**).

- Calcul des transitions par la réalisation des différentes équations logiques. (**Calcul des transitions**).
- La mise à jour des étapes sur la lumière de contenu des transitions (**mise à jour des étapes**).

On suit ces étapes dans les cas normaux, par contre, si on a pas des actions dans les étapes, on néglige la mise à jour des sorties.

Exemple : (Perceuse)

Etape –transition

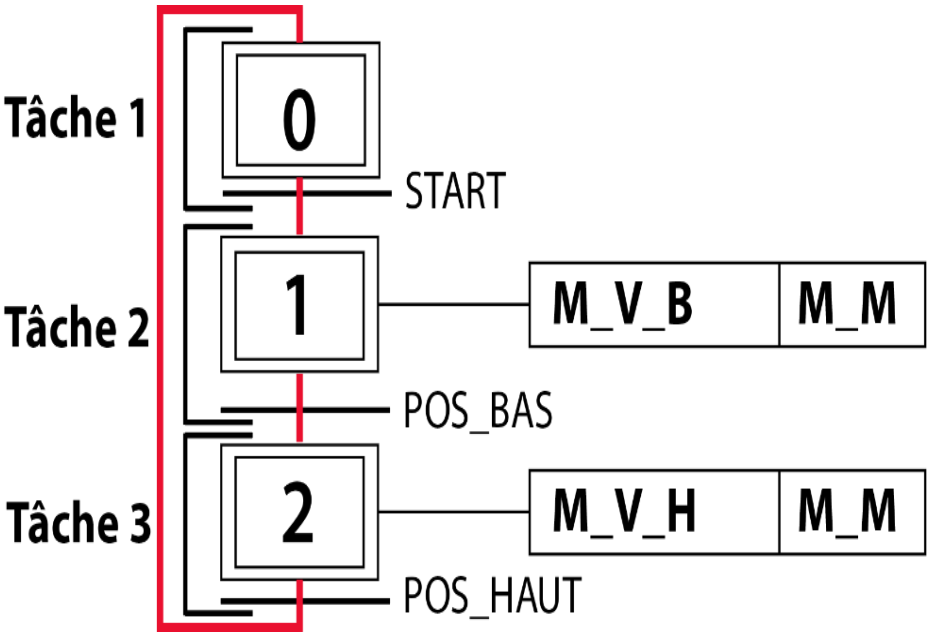


Figure 3.5 : Schéma Grafcet en tâches 'Etape-Transition'.

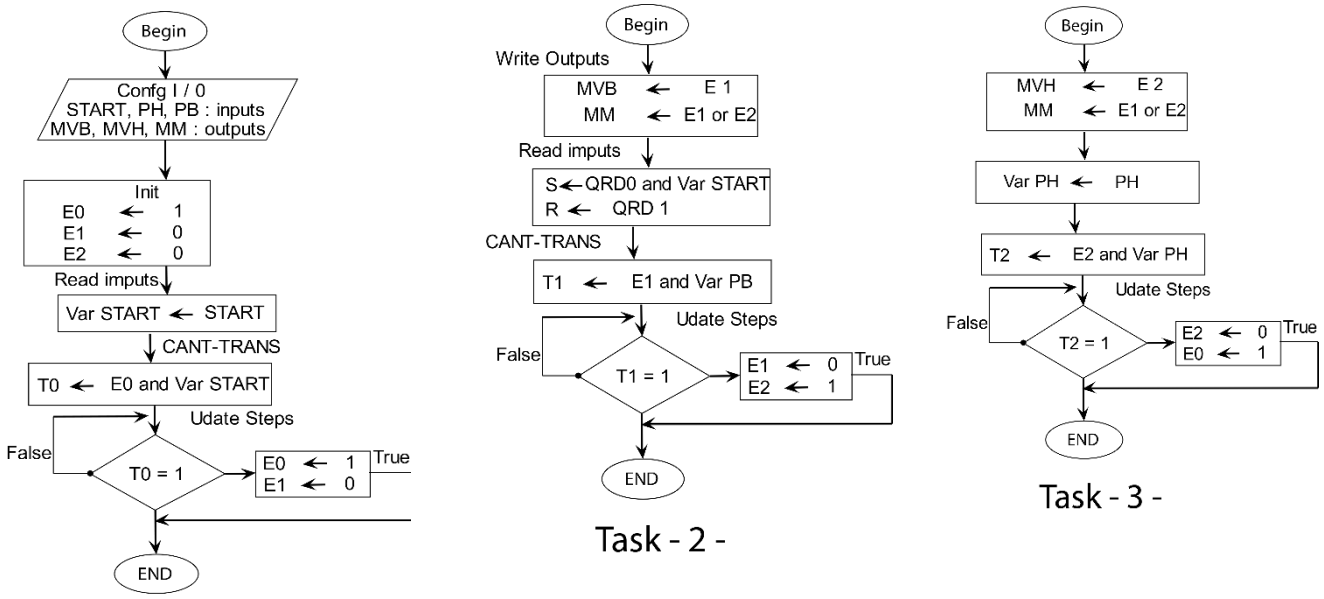


Figure 3.6 : Organigramme de Grafcet en tâches 'Etape-Transition

Solution de conversion :

// Initialisation
E0 =1 ; // **Marquage de la première étape**
E1 =0 ; // **Démarquage du reste**
E2 =0 ;
// Définition des fonctions
Void Task1 () bgin
// Write outputs
// void
// Read inputs
varSTART ←START
// CMPT-TRANS
T0← E0and varSTART
// update steps
If (T0=1) then
E0←1
E1←0
End if
End
Void Task2 () bgin
// WR-OUTS
MVB ←E1
MM ← E1 or E2
// RD_INS
varPB←PB
// CMPT-TRANS
T1←E1 and varPB
// update steps
If (T1=1)

E1←0
E2←1
End if
End
Void Task3 () bgin
// **WR-OUTS**
MVH ←E2
MM ← E1 or E2
// **RD_INS**
varPH←PH
// **CMPT-TRANS**
T2←E2 and varPH
// **update steps**
If (T2=1) then
E2←0
E0←1
End if
End

Exemple : (Perceuse)

Transition – Etape :

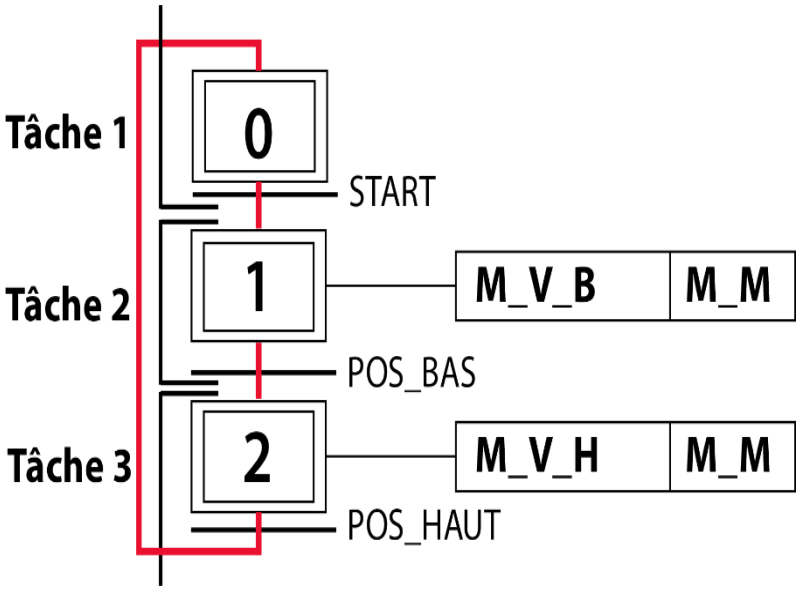


Figure 3.7 : Schéma Grafcet en tâches 'Transition-Etape'.

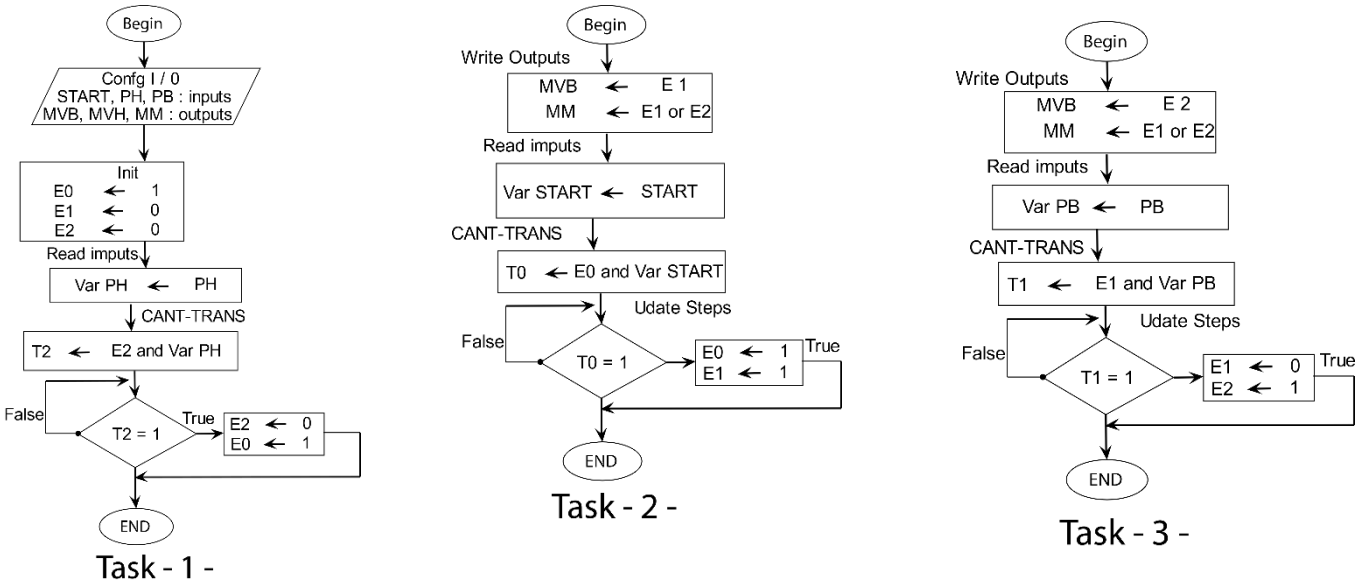


Figure 3.8: Organigramme de Grafcet en tâches 'Transition-Etape'

// Initialisation
E0 = 1; // Marquage de la première étape
E1 = 0; // Démarquage du reste
E2 = 0;
// Définition des fonctions
Void Task1 () bgin
// Write outputs
// void
// Read inputs
varPH←PH
// compute transition
T2←E2 and varPH
// update steps
If (T2=1) then
E2←0
E0←1
End if
End
Void Task2 () bgin
// WR-OUTS
MVB← E1
MM ← E1 or E2
// RD_INS
varSTART← START
// CMPT-TRANS
T0←E0 and varSTART
// update steps
If (T0=1)
E0←0

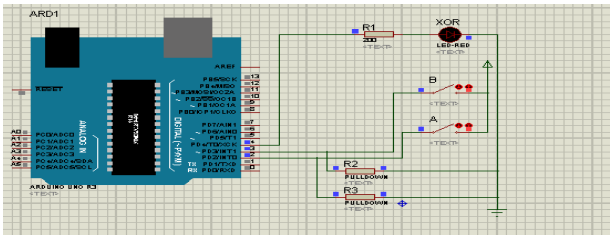
E1←1
End if
End
Void Task3 () bgin
// WR-OUTS
MVH ←E2
MM ← E1 or E2
// RD_INS
varPB PB
// CMPT-TRANS
T1←E1 and varPB
// update steps
If (T1=1)
E1←0
E2←1
End if
End

III.4.Présentation générale sur Proteus Professional :

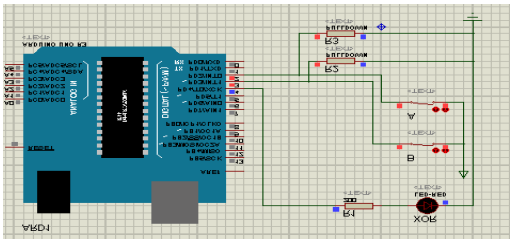
Proteus Professional est une suite logicielle destinée à l'électronique. Développé par la société L'absenter Electroniques, les logiciels incluent dans Proteus Professional permettent la CAO (Construction Assistée par Ordinateur) dans le domaine électronique. Deux logiciels Principaux composent cette suite logicielle : (ISIS, ARES, PROSPICE) et VSM. Cette suite Logicielle est très connue dans le domaine de l'électronique.

Proteus est composé de différents packages qui sont : Proteus PCB pour le circuit imprimé, Proteus VSM pour la simulation, Proteus Visual Designer/IoT Builder pour Arduino/Raspberry pour développer des projets comparables à ceux conçus avec des outils tels que Scratch et App inventer. PROTEUS permet de saisir les schémas électronique (ISIS), soit en page simple, soit en hiérarchique. L'environnement de saisie est entièrement paramétrable, si bien qu'il est possible de lui appliquer des skins (ZUKEN ou MENTOR par exemple). La saisie schématique est très simple et intuitive, PROTEUS est une des CAO les plus simples et intuitive qui existe, tout en restant un outil très puissant pour réaliser des ensembles complexes et atteindre un requis industriel conforme aux normes ISO [23].

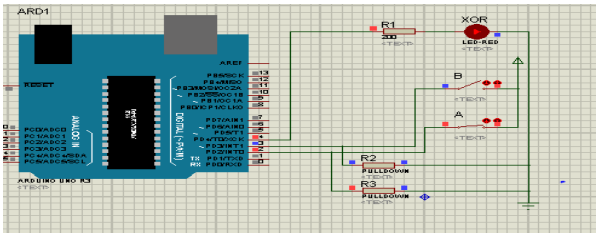
III.5.Simulation de la décomposition d'une solution LADDER vers tâches :



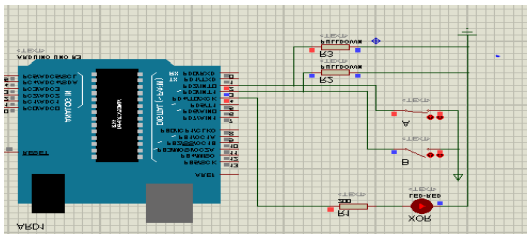
Etat 1 : A =0 ; B = 0 ; 0 XOR= 0



Etat 2 : A =1 ; B = 1 ; XOR= 0



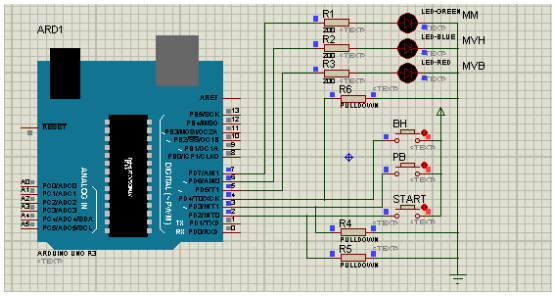
Etat 3: A =0 ; B = 1 XOR= 1



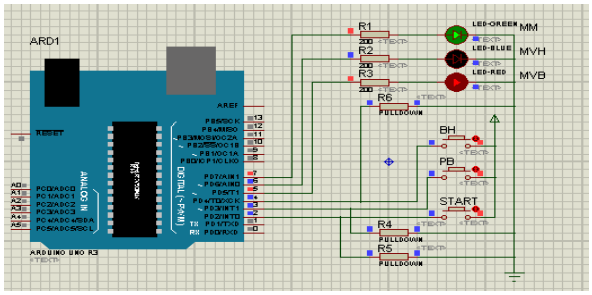
Etat 4 : A =1 ; B = 0 XOR=1

Figure 3.9 : Simulation de la fonction XOR en le proteus.

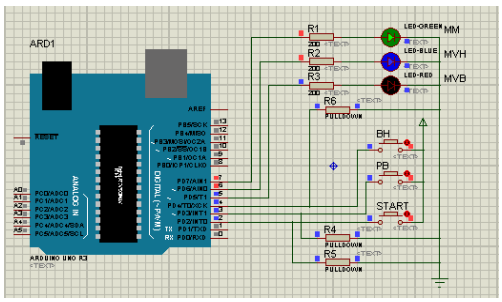
Cette figure (3.8) représente la simulation de la fonction XOR de ladders vers tâches en utilisant la plateforme proteus. Lorsque A et B sont différents la lampe s'allume, lorsque A et B sont pareils la lampe s'éteint.



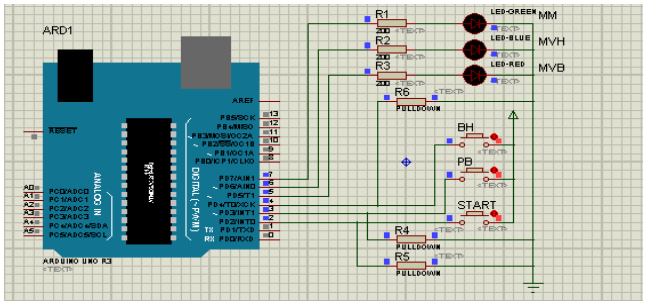
Etat 1 : la perceuse au Repos



Etat 2 : Rotation du moteur vers le bas



Etat 3 : Rotation du moteur vers le haut

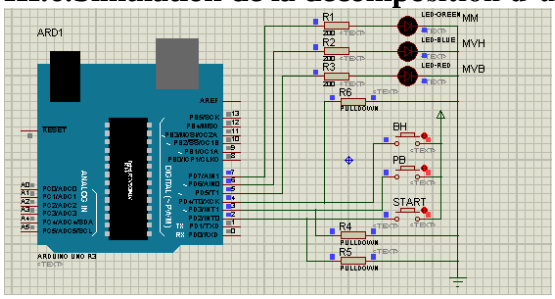


Etat 4 : Retour à l'état initiale

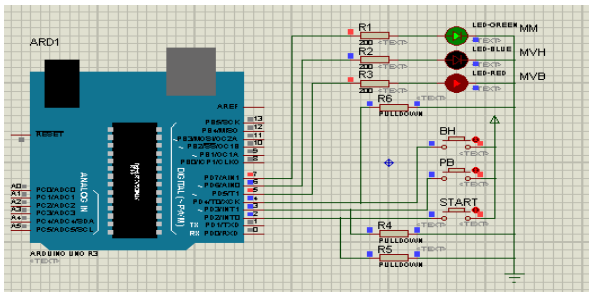
Figure 3.10 : Simulation proteus de la perceuse.

Cette figure (3.6) représente la simulation de la perceuse de Ladder vers tâches en utilisant la plateforme proteus. En appuyant sur le bouton START la perceuse commence à tourner sur elle-même et descend lorsqu'elle arrive en bas on appui sur le bouton PUSH BAS la perceuse remonte vers le haut, après on appui sur le bouton PUSH HAUT, elle revient à l'état initiale.

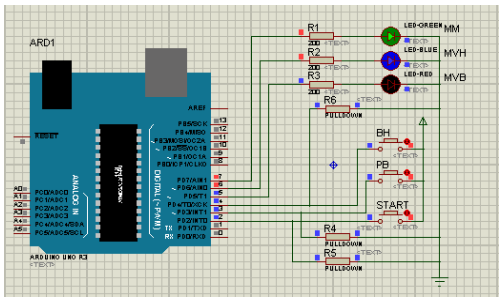
III.6.Simulation de la décomposition d'une solution GRAFCET vers tâches :



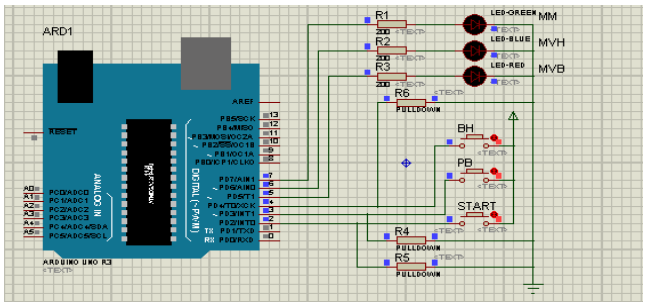
Etat 1 : la perceuse au Repos



Etat 2 : Rotation du moteur vers le bas



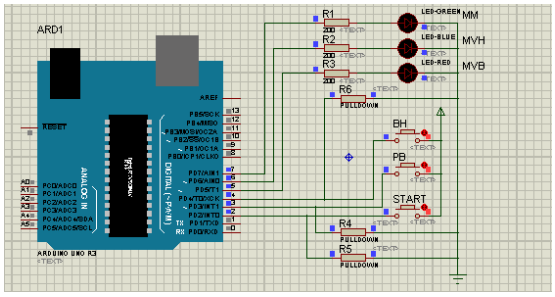
Etat 3 : Rotation du moteur vers le haut



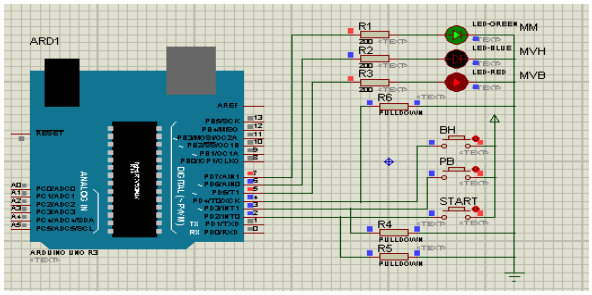
Etat 4 : Retour à l'état initiale

Figure 3.11 : Simulation de la perceuse en le proteus (étape- Transition).

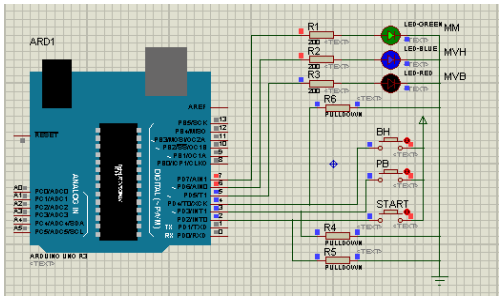
Cette figure (3.10) représente la simulation de la perceuse de Grafcet (Transition – Etape) vers tâches en utilisant la plateforme proteus. En appuyant sur le bouton START la perceuse commence à tourner sur elle-même et descend lorsqu'elle arrive en bas on appui sur le bouton PUSH BAS la perceuse remonte vers le haut, après on appui sur le bouton PUSH HAUT, elle revient à l'état initiale.



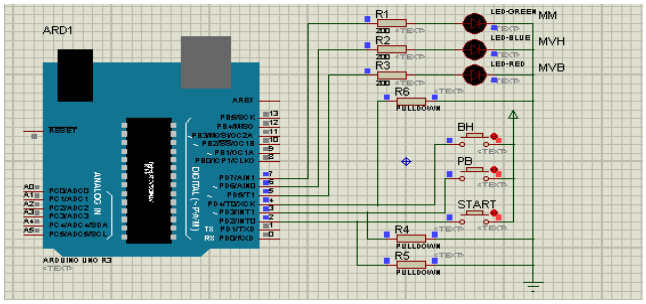
Etat 1 : la perceuse au Repos



Etat 2 : Rotation du moteur vers le bas



Etat 3 : Rotation du moteur vers le haut



Etat 4 : Retour à l'état initiale

Figure 3.12 : simulation de la perceuse en le proteus (transition –Etape).

Cette figure (3.12) représente la simulation de la perceuse Grafcet (transition -Etape) vers tâches en utilisant la plateforme proteus.

Conclusion :

Dans ce dernier chapitre, on a présenté les algorithmes et la réalisation des décompositions suivantes ; du Ladder vers tâches et du Grafcet vers tâches, en utilisant pendant la simulation ; l'arduino ainsi que le proteus ; De là, nous pouvons aller vers multitâche à travers et l'implémentation d'un RTOS.

Conclusion générale

Conclusion générale :

Dans notre projet, l'étude est basé sur la conversion du Ladder /Grafcet en multitâches en utilisant le logiciel Arduino et Proteus.

Ce mémoire ce compose en trois étapes ; état de l'art sur les systèmes temps réel et les politiques d'ordonnancement ; conversion du Ladder / Grafcet à un algorithme informatique ; décomposition d'une solution en LADDER/GRAFCET en tâches à travers une plateforme à μP OU μC . L'exécutif système temps réel est une exploitation dédié qui offres à une application système temps réel chargée de piloter un procédé , des services lui permettant de respecter sa principale contrainte ; celle du temps imparti pour s'exécuter le respect contraintes temporelle globales d'une application temps réel et répartie implique le respect , à la fois , des contraintes temporelles des tâches et de celles des messages échangés entre ces tâches .Nous avons synthétisé les différentes techniques qui permettent le placement et la migration de tâches temps réel et la communication de messages échangés entre ces tâches dans des architectures réparties.

On a traité la conversion du Ladder /Grafcet en algorithme informatique et décomposition d'une solution en LADDER/GRAFCET en tâches à travers une plateforme à μP OU μC l'Arduino et Proteus en illustration une fonction XOR en logique combinatoire et un exemple de la perceuse séquentielle en étapes.

ANNEXE

Annexe

Perceuse | Arduino 1.8.8

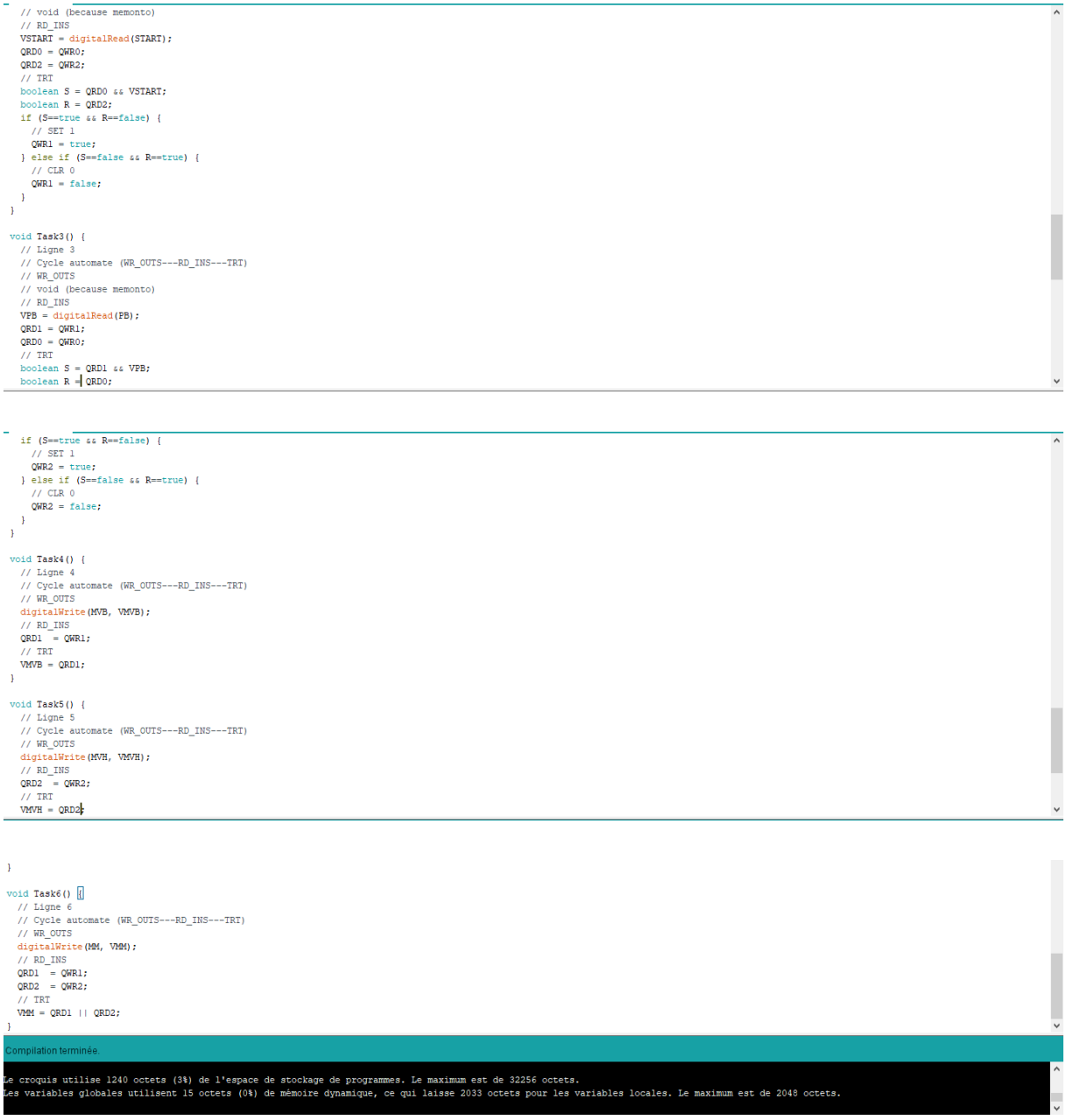
Fichier Édition Croquis Outils Aide

Perceuse §

```
#include <Mouse.h>

/*
  Description : Exemple de converstion LADDER vers Multitasks
  Exemple : automatisation d'une perceuse
*/

// Défintion des I/O
#define  START  2
#define  PB     3
#define  PH     4
#define  MVB    5
#define  MVH    6
#define  MM     7

// Déclaration des variables globales
boolean  VSTART, VPB, VPH;
boolean  VMVB, VMVH, VMM;

// Déclaration des memontos
boolean  QRD0, QRD1, QRD2;
boolean  QWR0, QWR1, QWR2;

// Prototypage des fonctions
void Task1();
void Task2();
void Task3();
void Task4();
void Task5();
```

```
void Task6();

// La fonction SETUP
void setup() {
  // Configuration des I/O
  pinMode(START, INPUT);
  pinMode(PB, INPUT);
  pinMode(PH, INPUT);
  pinMode(MVB, OUTPUT);
  pinMode(MVH, OUTPUT);
  pinMode(MM, OUTPUT);

  // Initialisation
  VMVB = false;
  VMVH = false;
  VMM = false;
  QWR0 = true; // Marquage de la première étape
  QWR1 = false; // Démarquage du reste
  QWR2 = false;
}

// La fonction LOOP
void loop() {
  // Traitement
  Task1();
  Task2();
  Task3();
  Task4();
  Task5();
```

```
  Task6();
}

// Défintion des fonctions
void Task1() {
  // Ligne 1
  // Cycle automate (WR_OUTS---RD_INS---TRT)
  // WR_OUTS
  // void (because memonto)
  // RD_INS
  VPH = digitalRead(PH);
  QRD2 = QWR2;
  QRD1 = QWR1;
  // TRT
  boolean S = QRD2 && VPH;
  boolean R = QRD1;
  if (S==true && R==false) {
    // SET 1
    QWR0 = true;
  } else if (S==false && R==true) {
    // CLR 0
    QWR0 = false;
  }
}

void Task2() {
  // Ligne 2
  // Cycle automate (WR_OUTS---RD_INS---TRT)
  // WR_OUTS
```



```
  // void (because memonto)
  // RD_INS
  VSTART = digitalRead(START);
  QRD0 = QWR0;
  QRD2 = QWR2;
  // TRT
  boolean S = QRD0 && VSTART;
  boolean R = QRD2;
  if (S==true && R==false) {
    // SET 1
    QWR1 = true;
  } else if (S==false && R==true) {
    // CLR 0
    QWR1 = false;
  }
}

void Task3() {
  // Ligne 3
  // Cycle automate (WR_OUTS---RD_INS---TRT)
  // WR_OUTS
  // void (because memonto)
  // RD_INS
  VPB = digitalRead(PB);
  QRD1 = QWR1;
  QRD0 = QWR0;
  // TRT
  boolean S = QRD1 && VPB;
  boolean R = QRD0;
  if (S==true && R==false) {
    // SET 1
    QWR2 = true;
  } else if (S==false && R==true) {
    // CLR 0
    QWR2 = false;
  }
}

void Task4() {
  // Ligne 4
  // Cycle automate (WR_OUTS---RD_INS---TRT)
  // WR_OUTS
  digitalWrite(MVB, VMVB);
  // RD_INS
  QRD1  = QWR1;
  // TRT
  VMVB = QRD1;
}

void Task5() {
  // Ligne 5
  // Cycle automate (WR_OUTS---RD_INS---TRT)
  // WR_OUTS
  digitalWrite(MVH, VMVH);
  // RD_INS
  QRD2  = QWR2;
  // TRT
  VMVH = QRD2;
}

void Task6() {
  // Ligne 6
  // Cycle automate (WR_OUTS---RD_INS---TRT)
  // WR_OUTS
  digitalWrite(MM, VMM);
  // RD_INS
  QRD1  = QWR1;
  QRD2  = QWR2;
  // TRT
  VMM = QRD1 || QRD2;
}
```

```
Compilation terminée.

Le croquis utilise 1240 octets (3%) de l'espace de stockage de programmes. Le maximum est de 32256 octets.
Les variables globales utilisent 15 octets (0%) de mémoire dynamique, ce qui laisse 2033 octets pour les variables locales. Le maximum est de 2048 octets.
```

Figure 3.9: Programme d'Arduino de la perceuse.

Cette figure (3.5) représente le programme de la perceuse du ladder vers tâches en utilisant la plateforme arduino.

Figure 3.10 : Programme d'Arduino de la fonction XOR.

Cette figure (3.7) représente le programme de la fonction XOR de ladders vers tâches en utilisant la plateforme Arduino.

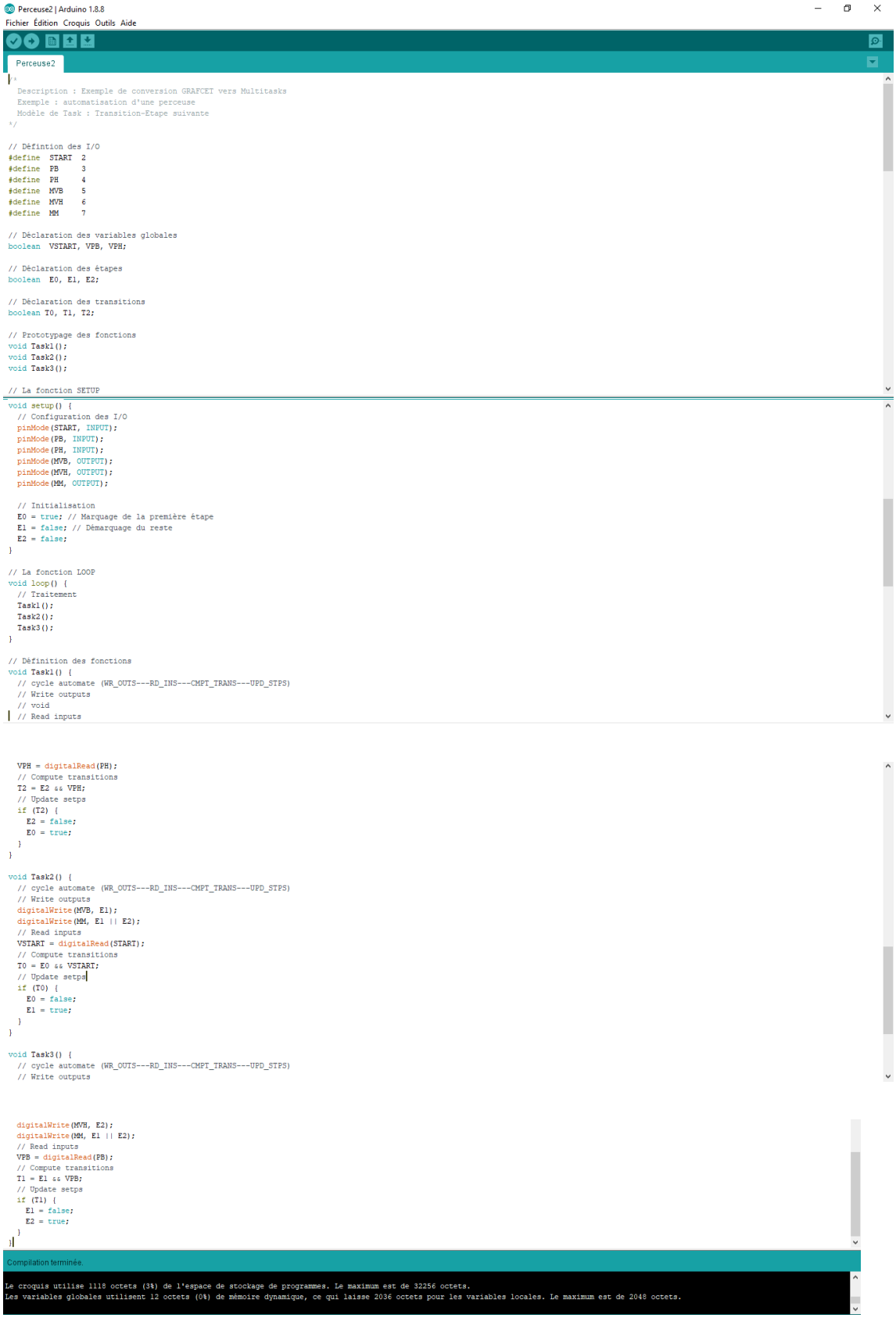


Figure 3.11 : programme d'Arduino de la perceuse.

Cette figure (3.9) représente le programme de la perceuse du Grafcet (Transition – Etape) vers tâches en utilisant la plateforme Arduino.

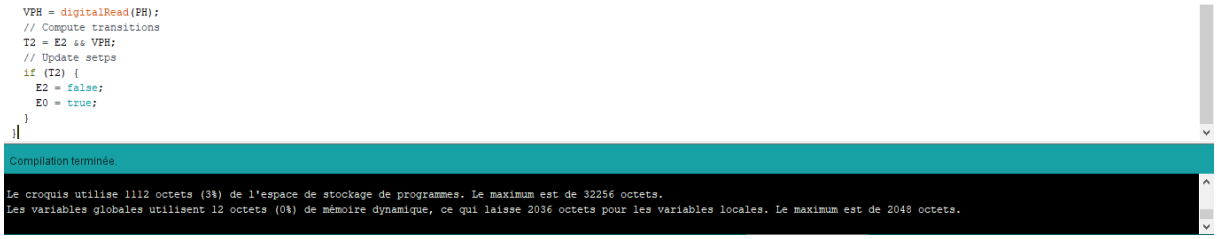


Figure 3.12 : Programme d'Arduino de la perceuse.

Cette figure (3.11) représente le programme de la perceuse du Grafcet (Etape –transition) vers tâches en utilisant la plateforme Arduino.

Références

[1] JohnStankovic, « Misconceptions about real-time computing », IEEE Computer, oct. 1988

[2] C. Duvallet, Z. Mammeri et B. Sadeg, « Analyse des protocoles de contrôle de concurrence et des propriétés ACID dans les SGBD temps réel », Revue TSI, 1999

[3] https://www.geeksforgeeks.org/characteristics-of-real-time-systems/vu le mars/2021

[4]D. J. Hatley, I. A. Pirbhai: Strategies for Real-Time System Specification; Dorset House, New York, 1987.

[5]Gomaa, Hassan, "A Software Design Method for Real-Time Systems," Communications of the ACM, Sept., 1984

[6]CCITT 1988

[7] OMG 1995

[8] HOOD Manual, CRI-CisiMatra, Toulous

[9]R. Van Ommering, F. Van der Linden & J. Kramer, "The Koala component model for consumer electronics software", IEEE Computer, Vol. 33, N° 3, pp. 78-85, March 2000

[10] https://www.geeksforgeeks.org/applications-of-real-time-system/ vu le Avril/2021.

[11] LIU C., LAYLAND J., « Scheduling algorithms for multiprogramming in a hard realtime environment », Journal of

[12] RICHARD P., COTTET F., RICHARD M., « On-line scheduling of Real-Time Distributed Computers With Complex Communication Constraints », ICECCS'2001, edited by Press, IEEE Computer, Skövde (Sweden), p. 26-34, 2001

[13] AUDSLEY N., BURNS A., DAVIS R., TINDELL K., WELLINGS A., « Fixed Priority Pre-emptive Scheduling : An Historical Perspective », Real-Time Systems, Kluwer, vol. 8, n˚2/3, p. 173-198, mars / mai 1995.

[14] [BAR 04] BARUAH S., GOOSSENS J., « Scheduling Real-time Tasks : Algorithms and Complexity », In Handbook of Scheduling : Algorithms, Models, and Performance Analysis, Joseph Y-T Leung (ed). Chapman Hall/CRC Press, 2004.

[15] DERTOUZOS M., « Control robotics: the procedural control of physical processors », IFIP Congress, p. 807

[16] MOK A., Fundamental design problems of distributed systems for the hard realtime environment, P

[17] [DER 89] DERTOUZOS M., MOK A., « Multiprocessor on-line scheduling of hard real-time tasks », IEEE Transactions on Software Engineering, vol. 15, p. 1497-1506, 1989.

[18]ALAIN GONZAGA " LES AUTOMATES PROGRAMMABLES INDUSTRIELS ".

[19] Mémoire de Master "Système de Contrôle Distribué (DCS) avec l'exploitation de l'automate programmable AC800 F (ABB) ", Université Mohamed KhiderBiskra , Juin 2012.

[20] sitelec.org/download.php?filename=cours/automates...industriels.pdf.

[21]Slim BEN SAOUD " LES AUTOMATES PROGRAMMABLES INDUSTRIELS API ".

[22] https://www.positron-libre.com/electronique/arduino/arduino.phpJuin 2021

[23] https://www.moussasoft.com/simuler-vos-projet-arduino-sur-proteus-isis juin 2021