Identificación del comportamiento no observable de un DES emulado, a partir de un generador de lenguaje.

Rubén Darío Muñoz Chávez

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Electrónica, Instrumentación y Control.

Ingeniería en Automática Industrial

Popayán, Junio de 2017

Identificación del comportamiento no observable de un DES emulado, a partir de un generador de lenguaje.

Rubén Darío Muñoz Chávez

Director: PhD. Mariela Muñoz Añasco

Monografía presentada como requisito parcial para optar por el título de Ingeniero en Automática Industrial

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Electrónica, Instrumentación y Control.

Ingeniería en Automática Industrial

Popayán, Junio de 2017

Índice general

1 GENERALIDADES. 11

1.1 Conceptos básicos 11
1.1.1 Sistemas dinámicos de eventos discretos DES 11
1.1.2 Autómata Finito 11
1.1.3 Red de Petri 12
1.1.4 Red de Petri Interpretada 12
1.2 Estado del arte. 13
1.2.1 Métodos de Identificación en DES 13
1.2.2 Identificación del comportamiento observable 13
1.2.3 Métodos seleccionados 15
1.2.4 Identificación del comportamiento no observable 21
1.3 Análisis 23

2 SISTEMA DE EVENTOS DISCRETOS 25

2.1 Selección del proceso DES 25
2.1.1 Proceso de producción 26
2.2 Diseño y modelado del proceso 27
2.2.1 División del proceso en subsistemas 30
2.2.2 Funcionamiento y modelado 30
2.2.3 Verificación de integración entre modelo 54
2.2.4 Verificación de Propiedades estructurales 54

3 IMPLEMENTACIÓN DE LOS MODELOS EN LA TARJETA DESARROLLO 67

3.1 Integración Planta - Controlador 67
3.1.1 Procedimiento Controlador 71
3.1.2 Procedimiento Características de Planta 73
3.1.3 Procedimiento de Comportamiento de Planta 76
3.1.4 Procedimiento Integración Planta Controlador 78
3.2 Implementación de adquisición de señales 80
3.2.1 Comunicación Con Dispositivo de Desarrollo 81

4 APLICACIÓN SOFTWARE 83

4.1 Entorno de desarrollo 83
4.1.1 Reconociendo el entorno de desarrollo 85

4.1.2 Interfaz principal 86
4.1.3 Interfaz de configuración de señales 87
4.2 Espacios de la aplicación 88
4.2.1 Área de planta - controlador 88
4.2.2 Área de aislamiento de señales 89
4.2.3 Área de identificación de autómata 90
4.2.4 Área de identificación de PN 92
4.3 Generación automática de gráficos 97

5 IDENTIFICACIÓN DEL COMPORTAMIENTO OBSERVABLE Y NO OBSERVABLE EN EL CASO DE ESTUDIO 101
5.1 Identificación del comportamiento observable del sistema. 101
5.1.1 Aplicación de los algoritmos 101
5.1.2 Comparación de los resultados con las dos metodologías 106
5.2 Identificación del comportamiento no-observable 108
5.2.1 Definición del comportamiento observable y no observable . . . 108
5.2.2 Definición de prueba de detectabilidad y aplicación en el DES . 108
5.2.3 Aplicación de algoritmo de detectabilidad en caso de estudio . . 111

Bibliografía 121

Índice de figuras

1.1. Diagrama de flujo algoritmo NDFAA 16
1.2. NDFAA de ejemplo identificado [1]. 17
1.3. Diagrama de flujo algoritmo IPN 18
1.4. Modelo ejemplo IPN identificada 20
1.5. Ejemplo detector G_{det} [2]. 22

2.1. Proceso de diseño del sistema de eventos discretos. 25
2.2. Programa QFSM. 29
2.3. Diagrama etapa de suministro de materia prima 32
2.4. Diagrama de señales etapa de suministro de materia prima 33
2.5. Modelo máquina de estados etapa suministro de materiales 34
2.6. Diagrama de señales etapa de molienda 36
2.7. Modelo máquina de estados etapa molienda de material 37
2.8. Diagrama etapa de atomizado y secado 38
2.9. Diagrama de señales etapa de atomizado y secado 39
2.10. Modelo máquina de estados etapa atomizado y secado 40
2.11. Diagrama etapa de prensado 41
2.12. Diagrama de señales etapa de prensado 42
2.13. Modelo máquina de estados etapa prensado 43
2.14. Diagrama etapa de girado 44
2.15. Diagrama de señales etapa de girado 45
2.16. Modelo máquina de estados etapa girado 46
2.17. Diagrama de señales etapa de secado por luz UV 48
2.18. Modelo máquina de estados etapa secado por luz UV 49
2.19. Diagrama de señales etapa de impresión 50
2.20. Modelo máquina de Estados Etapa Impresión 51
2.21. Diagrama horno de banda transportadora 52
2.22. Diagrama de señales etapa de horneado 52
2.23. Modelo máquina de estados etapa horneado 53
2.24. PN Suministro de materia prima 55
2.25. PN Molienda de material 55
2.26. PN Atomizado y secado 56
2.27. PN Prensado 56
2.28. PN Girado 57

2.29. PN Secado por luz UV 57
2.30. PN Impresión 58
2.31. PN Horneado 58
2.32. Árbol de alcanzabilidad Suministro de materia prima 59
2.33. Árbol de alcanzabilidad Molienda de material 60
2.34. Árbol de alcanzabilidad Atomizado y secado 61
2.35. Árbol de alcanzabilidad Prensado 62
2.36. Árbol de alcanzabilidad Girado 63
2.37. Árbol de alcanzabilidad Secado por luz UV 64
2.38. Árbol de alcanzabilidad Impresión 64
2.39. Árbol de alcanzabilidad Horneado 65

3.1. Asignación de entradas en C 68
3.2. Asignación de salidas en C 69
3.3. Máquina de Estados de Ejemplo 72
3.4. Estructura del código de Controlador 73
3.5. Estructura del código para elementos tipo Banda transportadora 74
3.6. Estructura del código Para Elementos de Temperatura o Movimiento Mecánico 75
3.7. Estructura del código para Elementos de Almacenamiento 76
3.8. Estructura del código Para Tiempo de Actualización de una Etapa . . . 77
3.9. Ejemplo Estructura de Condición de Cambio Banda Transportadora . . 78
3.10. Ciclo infinito planta controlador 78
3.11. Diagrama Bloques de Sistema de Eventos Discretos Diseñado 79
3.12. Puertos de Arduino Mega 80
3.13. Tiempos de adquisición de señales 81
3.14. Trama de Datos Comunicación Serial 82

4.1. Áreas de Trabajo del IDE Visual Studio Community 85
4.2. Interfaz Principal de la aplicación 87
4.3. Interfaz Configuración de Señales 88
4.4. Proceso de aislamiento de señales 89
4.5. Estructura del script para Gráficos 97
4.6. Ejemplo PN 98

5.1. Autómata identificado subsistema molino 102
5.2. PN identificada subsistema molino 104
5.3. Ejemplo IPN detectabilidad. 110

Índice de tablas

2.1. Tabla Asignación de Señales de Salida 31

5.1. Código Autómata subsistema molino 103
5.2. Código PN subsistema molino Parte 1 105
5.3. Código PN subsistema molino Parte 2 106
5.4. Características de computadora utilizada 106
5.5. Tabla Tiempos Autómatas 107
5.6. Tabla Tiempos IPNs 107
5.7. Identificación no observable Suministro de materias primas 112
5.8. Identificación no observable Molienda de material 112
5.9. Identificación no observable Atomizado y secado 113
5.10. Identificación no observable Prensado Parte 1 114
5.11. Identificación no observable Girado 115
5.12. Identificación no observable Secado por luz UV 116
5.13. Identificación no observable Impresión 117
5.14. Identificación no observable Horneado 117

Algoritmos

4.1. Algoritmo Generador de Secuencias 90
4.2. Algoritmo Identificación Progresiva Autómata 91
4.3. Algoritmo Generador de Eventos 93
4.4. Algoritmo Identificación para PN Parte 1 94
4.5. Algoritmo Identificación para PN Parte 2 95
4.6. Algoritmo Identificación para PN Parte 3 96

5.1. Algoritmo de detectabilidad para IPN 109

Resumen

En este trabajo de grado se diseña un DES del proceso de producción de lozas cerámicas, el cual mediante una simulación y emulación de las señales en una tarjeta de desarrollo, se realiza un proceso de identificación del DES como si se tratase de un sistema de caja negra donde solo se conocen sus señales para obtener una red de Petri (IPN Paramétrica) y un autómata (NDFAA) en cada uno de los subsistemas.

El sistema de eventos discretos diseñado es un sistema en lazo cerrado planta - controlador, el cual es de comportamiento binario, donde las señales de entrada son las respuestas o estado de los sensores y la señales de salida son las ordenes de control, el sistema de eventos discretos es implementado en una tarjeta de desarrollo arduino mega, donde gracias a otra tarjeta encargada de la adquisición de señales se logra capturar el comportamiento de dicho sistema mediante la comunicación continua entre el dispositivo de adquisición y el software diseñado.

A partir del desarrollo de una aplicación software que automatiza la identificación, se obtienen los modelos en código al igual que su respectivo grafo, adicionalmente se define una prueba de detectabilidad para los eventos no observables en los sistemas identificados.

INTRODUCCIÓN

Los sistemas pueden ser clasificados en el tiempo, según [3], como: continuos, discretos, orientados a eventos e híbridos. Un sistema de eventos discretos, SED, es un sistema dinámico que evoluciona con la aparición, a intervalos irregulares, de acontecimientos físicos, [4] y pueden ser maniobrados y/o supervisados por uno o varios controladores de eventos discretos. A nivel industrial, el controlador de eventos discretos se implementa usualmente con controladores lógicos programables (PLCs). Los SED surgen en muchos ámbitos tales como: robótica, tráfico de vehículos, logística, redes informáticas, redes eléctricas, equipos industriales, redes de comunicación, procesos de manufactura; entre otros. Estos sistemas se hacen cada vez más grandes, complejos y están en constante cambio debido al avance y desarrollo de nuevas tecnologías, métodos y procesos, por lo que se hace difícil el entendimiento de la dinámica en que funciona cada sistema. Debido a esta trascendencia se hace necesario identificar modelos que reflejen el comportamiento real de este tipo de sistemas; específicamente en entornos de automatización.

El problema de identificación en SED consiste en obtener un modelo aproximado de un sistema desconocido, expresado en un formalismo adecuado a partir del comportamiento observado representado como secuencias de eventos. Una manera de estudiar el comportamiento de los SED es a partir de su lenguaje. [1] La identificación de un modelo a partir del lenguaje del sistema, consiste en encontrar un generador que represente el lenguaje que modela el comportamiento [6]. Los formalismos para representar lenguajes son los Autómatas y las Redes de Petri (RdP). Los campos de aplicación de los modelos identificados en SED, se pueden dar para simular y comprobar características de verificación: vivacidad, alcanzabilidad, puntos muertos, entre otras; para control supervisorio o para diagnóstico de fallos, [7].

El problema de identificación de un SED en lazo cerrado (Planta - Controlador) es obtener el modelo a partir de señales de entrada - salida (E/S) observadas en la interrelación de la planta y el controlador. De acuerdo a esto, las señales de E/S deben transformarse en eventos; entonces el lenguaje del SED es una secuencia de señales de E/S.

[1] Un evento es la representación de un cambio instantáneo en alguna parte del sistema, puede caracterizarse por un valor y un instante en el que ocurre, [5]; un evento constituye una letra y el conjunto de eventos es el alfabeto; una secuencia de eventos es una palabra, [4], una secuencia de palabras constituye el lenguaje del sistema.

Como el lenguaje son secuencias observadas de señales de E/S, el modelo representa el comportamiento observable del sistema; es decir, presenta las relaciones directas entre las señales; pero no la evolución interna entre los estados sin cambios en las señales. Cuando la aplicación del modelo identificado se hace con fines de diagnóstico de fallos, una característica importante en el generador del lenguaje de E/S, es que permita identificar comportamientos no observables. Por lo tanto, se hace importante investigar sobre la manera de identificar este tipo comportamientos.

Se plantea la siguiente pregunta de investigación: ¿Cómo identificar comportamientos no observables en un generador de lenguaje de E/S, que modela el comportamiento de un SED identificado por un método ya establecido, en un caso de estudio?.

Para ello, en el presente proyecto se propone el desarrollo de un sistema de eventos discretos emulado para ser objeto de estudio y el desarrollo de una aplicación software que codifica dos métodos de identificación uno de autómatas y otro de redes de Petri, los cuales se consideran generadores de lenguajes de comportamiento. Sobre ellos se realizarán pruebas para identificar el comportamiento no observable. Para cumplir con esto, este trabajo se ha dividido en los siguientes capítulos:

En el capítulo 1 se presentan los conceptos básicos de la teoría que se maneja en el documento; así como el estado del arte del tema investigado. En el capítulo 2, se expone el diseño de un sistema de eventos discretos. En el capítulo 3 se detalla el proceso de desarrollo de un aplicativo software que tiene como función la adquisición de señales a partir de una tarjeta de desarrollo y que permite la implementación del diseño SED con el objeto de generar el lenguaje de comportamiento del SED. En el capítulo 4, se muestra la programación de los algoritmos elegidos para realizar el proceso de identificación del modelo. En el capítulo 5, se presenta el proceso de identificación del sistema, partiendo de la simulación del SED, la generación y adquisición de señales y la obtención de los modelos; así como el proceso de identificación del comportamiento no observable. En el capítulo 6 se analizan los resultados obtenidos y al final se generan las conclusiones del trabajo.

1 GENERALIDADES.

1.1. Conceptos básicos

1.1.1. Sistemas dinámicos de eventos discretos DES

En el modelado de DES, el lenguaje que lo representa, son todas las secuencias de eventos (palabras) admisibles del sistema. Un generador de lenguaje es un formalismo matemático capaz de representar el lenguaje.

Los sistemas dinámicos de eventos discretos o más ampliamente conocidos como sistemas de eventos discretos (SED o DES), satisfacen las siguientes dos propiedades [4]:

- El espacio de estado es un conjunto discreto.
- El mecanismo de transición de estado es conducido por eventos.

Definición 1. Un sistema de eventos discretos (DES) es un sistema de estado discreto, conducido por eventos, esto es, la evolución del estado depende totalmente de la ocurrencia de eventos discretos asíncronos en el tiempo.

1.1.2. Autómata Finito

Un autómata finito es un modelo matemático de una máquina de estados, que permite saber si una cadena de símbolos pertenece o no a un lenguaje definido sobre cierto alfabeto.

Un Autómata finito es un autómata que recibe secuencialmente una cadena de símbolos y cambia de estado por cada símbolo leído o también puede permanecer en el mismo estado. Al final de la lectura el estado del autómata indica si la cadena es aceptada, es decir, pertenece al lenguaje que describe la máquina. Si al final de leer todos los símbolos de entrada, la máquina está en alguno de los estados finales entonces esa cadena es aceptada, si el estado no es final entonces la cadena no pertenece al lenguaje.

Definición 2. Un Autómata Determinístico denotado por G, es una quintupla, $G = (X, E, f, x_0, X_m)$, donde:

X: Es un conjunto de estados. E: Es un conjunto finito de eventos asociados a las transiciones. $f : XxE \to X$: Es la función de transición de estados. x_0: Es el estado inicial. $X_m \subset X$: Es el conjunto de estados marcados.

1.1.3. Red de Petri

Las redes de Petri (PN) son modelos capaces de describir el flujo de información total de un sistema con procesos concurrentes y distribuidos.

Las PN son ampliamente usadas para modelar DES, [8, **?**]. Proveen modelos compactos y capturan características importantes de los DES, como concurrencia, sincronismo, relaciones causales, recursos compartidos, etc.

Definición 3 (PN)**.** Una PN N, es un grafo bipartito representado por la tupla $N = (P, TR, Pre, Post, M_0)$ [9] donde: P es el conjunto de lugares con cardinalidad np y TR es el conjunto de transiciones con cardinalidad ntr, $Pre : P \times TR \to \mathbb{N}$, $Post : TR \times P \to \mathbb{N}$ son la matrices que contienen los arcos que conectan lugares y transiciones. La matríz de incidencia $I = Post - Pre$ es una matríz de np x ntr. La función de marcado $M : P \to \mathbb{N}$ representa el número de marcas en cada lugar, M_0 es el marcado inicial, [9, 10].

Los conjuntos de lugares previos y posteriores a una transición, se denotarán como: $\bullet tr = \{p \in P : Pre\,(p, tr) > 0\}$ $tr\bullet = \{p \in P : \ Post\,(p, tr) > 0\}$, [10].

1.1.4. Red de Petri Interpretada

Una extensión de las PN son las redes de Petri Interpretadas (IPN, por sus siglas en inglés, interpreted Petri Nets) que asocian señales de E/S al modelo, [11].

Definición 4 (Red de Petri Interpretada (IPN))**.** Una IPN es una tupla $Q = (N, \Sigma, \Phi, \lambda, \varphi)$, donde, N es una PN como la definida en Definición 3, Σ es el conjunto de símbolos de entrada. Φ es el conjunto de símbolos de salida. $\lambda : TR \to \Sigma \cup \varepsilon$ es una función de etiquetado que asigna un símbolo de entrada a cada transición. $\varphi : R(N) \to \Phi$ es una función de salida que asigna un símbolo de salida a cada marcado alcanzado.

Una $tr_r \in TR$ de una IPN está habilitada si $\forall p_q \in \bullet tr_r$, $m\,(p_q) \geq I\,(p_q, tr_r)$. Si $\lambda\,(tr_r) \neq \varepsilon$ está presente y si tr_r está habilitada entonces tr_r se dispara y se alcanza un nuevo marcado M_{k+1}, el cual se calcula mediante la ecuación de estado:

$$\begin{array}{rcl} M_{k+1} & = & M_k + I.\overrightarrow{tr_r} \\ y_k & = & \varphi\,(M_k) \end{array} \tag{1.1}$$

1.2. Estado del arte.

El problema a resolver en el presente trabajo, implica dos aspectos importantes; por un lado identificar el comportamiento observable de un DES, caso de estudio a partir de datos de señales de entrada - salida (E/S) y representarlo en un generador de lenguaje y por otro lado, proponer un método para identificar el comportamiento no-observable del DES; es decir analizar la detectabilidad del generador de lenguaje.

A continuación se presentan los resultados encontrados en estas dos temáticas.

1.2.1. Métodos de Identificación en DES

El comportamiento de un DES se compone de dos partes, la primera es el comportamiento observable, el cual es obtenido de forma directa en la salidas, dependiendo de cambios en las entradas. La segunda parte corresponde al comportamiento no observable, el cual corresponde a la evolución de los estados internos en el sistema sin notar cambios en los datos observados E/S [7].

1.2.2. Identificación del comportamiento observable

La identificación de DES se inició como un problema de inferencia gramatical, luego se propusieron varios métodos para la obtención de máquinas principalmente de Mealy y Moore [12]. El proceso de identificación se puede realizar bajo diferentes formalismos autómatas o PN, los cuales tienen como propósito realizar la obtención de modelos aproximados de un DES donde se desconoce el sistema o es poco conocido, [13].

Para el enfoque basado en autómatas se busca generar el modelo mediante el uso de un lenguaje que se represente el comportamiento del sistema en un autómata, [14]. Existen dos formas de obtener el modelo de identificación, los basados en componentes y los basados en datos. Ambos enfoques permiten construir un modelo funcional. Los basados en componentes, el modelo consta de dos partes: un modelo del sistema y un modelo de comportamiento. El modelo de comportamiento puede representar una función realizada por varios componentes individuales. De esta manera se conduce a un modelado jerárquico del sistema; el enfoque basado en datos construye el modelo de manera empírica con la información recogida durante la operación del sistema. El modelo construido se ajusta a los datos históricos, mediante un algoritmo que relaciona diferentes parámetros de ese proceso, los parámetros de salida, los parámetros de entrada y los ajustes de control, [15, 16]. Klein ([15]), propone un método off-line para la construcción de un autómata finito no determinista (NDFAA) a partir de secuencias de E/S, el uso de este enfoque ofrece ventajas, ya que no se necesita ningún conocimiento profundo sobre la estructura y los componentes del sistema. El modelo construido es el

modelo del sistema en conjunto con el controlador que opera en lazo cerrado y solo los estados observables pueden ser presentados por las salidas del sistema. Una desventaja radica en el paralelismo entre estados no se puede modelar; además su aplicación en sistemas de gran escala no es conveniente por el tamaño del modelo resultante.

En [17] se propone trabajar el sistema general en subsistemas a partir de los resultados de [15] y en [18] se presenta una mejora en el método, contemplando eventos temporizados.

El formalismo de Red de Petri (PN) es muy reconocido como un modelo adecuado para describir la estructura y el dinamismo en sistemas complejos por su particularidad de evitar la explosión de estados, [19]. Bajo el enfoque de PNs se han clasificado varios métodos de identificación, [20], los cuales se describen a continuación:

Un enfoque es la identificación progresiva, donde el método de identificación se basa en observaciones de señales de E/S, trabajando de forma on-line para obtener un modelo de una Red de Petri Interpretada (IPN) que describa el comportamiento desconocido del DES, [21, 22]. Este enfoque considera un sistema en lazo cerrado en el que cada entrada al sistema es reflejada en la salida, esto significa que si incluso, un sistema no está completamente instrumentado, la información provista por lo sensores es suficiente para detectar un cambio de estado.

Otro enfoque es la identificación de lenguaje, con programación lineal Entera (IPL), el problema consiste en determinar un conjunto de lugares de cardinalidad m, un conjunto de transiciones de cardinalidad n, así como la función de etiquetado definido como un problema de programación lineal entera. En [23] se presenta un enfoque de identificación de una IPN con IPL, además de una secuencia de eventos, se utiliza la secuencia de respuesta de salida disponible del DES para hacer la inferencia del modelo de la IPN. En esta propuesta, el conjunto de transiciones finitas obtenido de manera off-line representan el lenguaje, el cual mediante restricciones o condiciones se logra representar el lenguaje del sistema, [10, 24, 25, 26]. Muchas extensiones a este trabajo se han realizado en [27, 28]. En [29], se obtiene un sistema de red de Petri a partir del lenguaje generado; es decir, el conjunto de secuencias de transiciones que pueden ser generadas desde el marcado inicial.

Y por otro lado está la identificación de IPN paramétrica, este enfoque fue presentado en [30], el objetivo es descubrir cómo construir un modelo compacto que pueda mostrar explícitamente un comportamiento a partir del comportamiento observado del sistema, expresado como una sola secuencia de sus señales de E/S y de cómo los componentes del sistema están relacionados. Las señales de entrada son señales de los sensores y las de salida son las emitidas por un PLC hacia el actuador.

1.2.3. Métodos seleccionados

Para cumplir con las actividades y cumplir con los objetivos del presente trabajo, de generar un método para identificar el comportamiento no observable de un DES, primero se debe elegir un método ya establecido que identifique el comportamiento observable. Teniendo en cuenta el análisis anterior, se han elegido los métodos propuestos en [13] y en [1], puesto que permitirían aplicar técnicas de identificación de comportamiento no observable a partir de señales de E/S. Estos métodos se describen a continuación.

Las señales de E/S se representan en forma de vector en dos partes, una es el vector de señales de entrada (sensores) $I(t) = \left\{ \begin{array}{c} i_1 \\ i_2 \\ i_n \end{array} \right\}$ y la otra es el vector de señales de salidas (actuadores - órdenes de control) $O(t) = \left\{ \begin{array}{c} o_1 \\ o_2 \\ o_k \end{array} \right\}$ los cuales forman un vector E/S $\left\{ \begin{array}{c} I(t) \\ O(t) \end{array} \right\}$ descrito como una observación de señales E/S.

1.2.3.1. Método basado en un autómata autónomo finito no determinista NDFAA

El método de identificación de un autómata autónomo finito no determinista, descrito en [1]consiste en la obtención de un modelo de un DES en el que cada uno de sus estados corresponde a un vector señales E/S o sucesos observados del sistema, el método de identificación se describe en el diagrama de flujo de la Figura 1.1 .

Del diagrama de flujo descrito se aprecia que el método parte de existencia de secuencias de observaciones o señales, donde se verifica una secuencia a la vez, comprobando que su primer y último elemento sea el mismo, cuando la secuencia es válida se realiza el proceso de identificación que consiste en la creación de los estados y transiciones, en el que se crea un estado siempre que una observación (vector de señales E/S) es nueva, creando una transición, conectando el estado actual con el estado creado. El proceso se repite continuamente hasta terminar la secuencia para luego iniciar nuevamente la verificación de una secuencia siguiente.

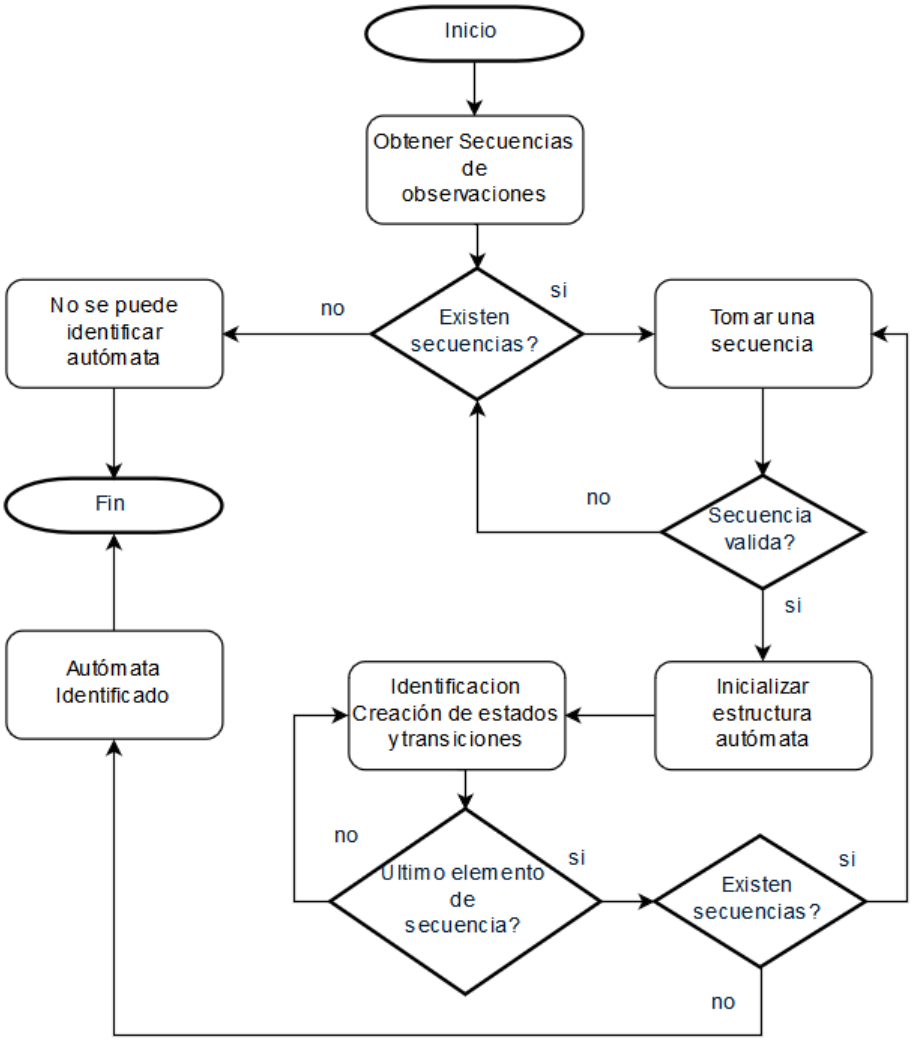

Figura 1.1: Diagrama de flujo algoritmo NDFAA

Para entender mejor el algoritmo se presenta las secuencias σ_1y σ_2 de señales E/S tomadas del capítulo 4 de [1] dondese genera una observación de tres señales con dos entradas i_1, i_2 y una salida o_1 se representa en un vector $U(t) = \left\{ \begin{array}{c} i_1 \\ i_2 \\ o_1 \end{array} \right\}$.

$$\sigma_1 = \left\{ \begin{array}{cccccc} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{array} \right\} \text{ y } \sigma_2 = \left\{ \begin{array}{cccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array} \right\}$$

De las secuencias de observaciones σ_1y σ_2 se puede apreciar que su primer y último

vector columna son iguales, esto indica que las secuencias son válidas para ser usadas en el algoritmo de identificación del NDFAA; por tanto, durante el proceso se crea un estado X_i para cada vector nuevo observado y las transiciones se crean siempre que se lee un nuevo vector de señales E/S o también cuando se lee un vector ya observado conectando entre si con el estado anterior. El autómata generado de las secuencias σ_1y σ_2 se muestra en la Figura 1.2.

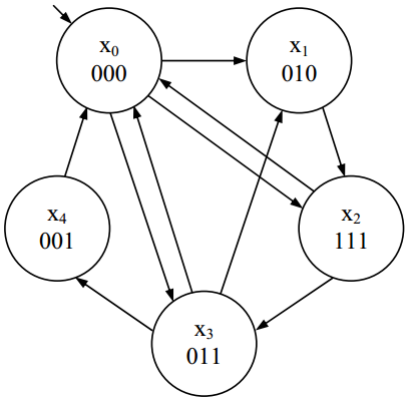

Figura 1.2: NDFAA de ejemplo identificado [1].

1.2.3.2. Método de Identificación IPN paramétrica

El método de identificación de una red de Petri interpretada (IPN) paramétrica, descrito en [30], consiste en la obtención de un modelo que describe la estructura y dinámica que tiene un DES.

En la Figura 1.3 se describe el diagrama de flujo simplificado del algoritmo suministrado.

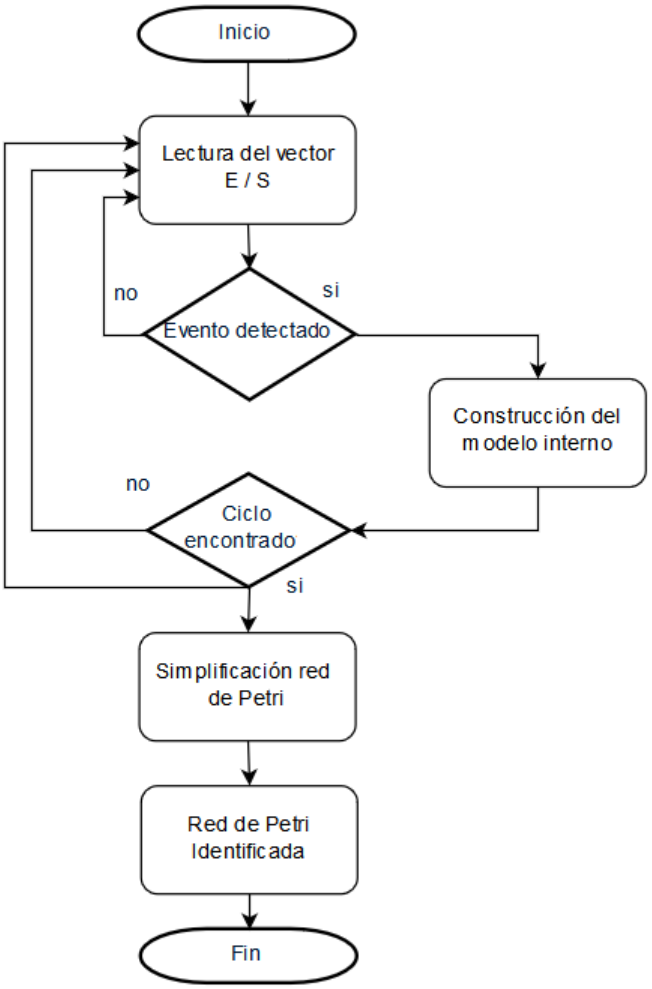

Figura 1.3: Diagrama de flujo algoritmo IPN

Del diagrama de flujo se aprecia que para la identificación de un modelo de IPN se parte de la generación de eventos con base en la diferencia entre vectores de señales E/S u observaciones consecutivas. El proceso de identificación del modelo interno es

donde se crean los lugares y la transiciones que forman el modelo, cada transición tiene asociado un lugar previo y un lugar posterior.

Cuando se realiza una lectura de E/S se almacenan tres variables, una es el vector E/S anterior, el actual y el evento generado a partir de la diferencia de las lecturas anterior y actual. Cuando se detecta un evento se realiza una verificación en la que se busca encontrar el evento en un conjunto de eventos observados, con esto se dan dos opciones, la primera es cuando el evento es nuevo, en este caso se crea una transición asociada al evento leído y un lugar asociado con la lectura E/S actual.

La segunda opción es cuando el evento leído ya ha sido observado, en este caso se busca encontrar si ya existe una transición posterior al lugar actual, tal que esté asociada al evento observado, si esto se cumple solo se actualiza el estado actual con el estado posterior a la transición encontrada. En el caso de no encontrarse ninguna transición asociada al evento observado solo puede significar dos cosas, la primera que el evento representa la creación de un ciclo interno en el modelo, y el segundo caso es que a pesar de ser un evento ya observado se realice el proceso de creación de una transición y lugar nuevo tomando dicho evento como si fuese nuevo.

Cuando ocurre un caso especial que suele estar asociado a un evento ya observado de una transición donde el vector de marcado E/S asociado a un lugar previo de una transición encontrada, no es el mismo que el vector E/S actual.

$$\sigma = \left\{ \begin{array}{cccccccccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array} \right\}$$

Un ejemplo tomando las secuencias σ en el que se tienen dos sensores y una salida, se obtiene el modelo mostrado en la Figura 1.4 donde se aprecia un modelo de mayor dimensión de elementos comparado al del autómata Figura 1.2, permitiendo observar más detalladamente la dinámica del sistema en la ocurrencia de eventos.

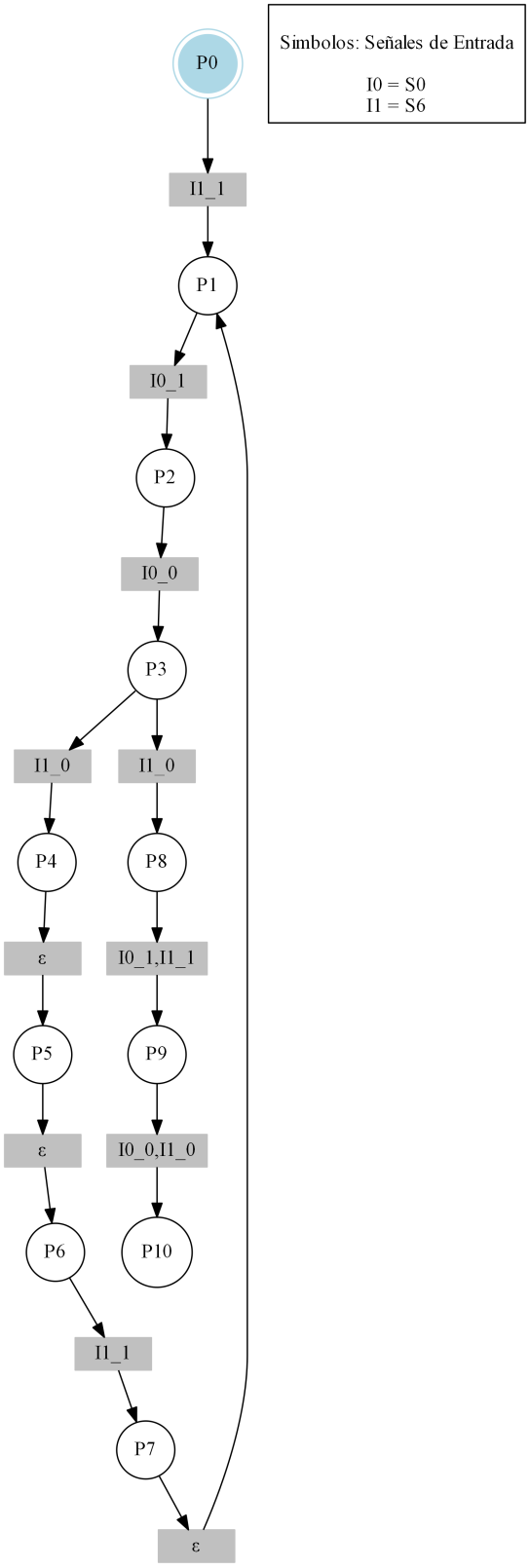

Figura 1.4: Modelo ejemplo IPN identificada

1.2.4. Identificación del comportamiento no observable

La identificación del comportamiento no-observable de un DES está relacionada con la propiedad de detectabilidad. Ésta permite a un observador estimar perfectamente el estado actual del sistema después de un número finito de símbolos observados; es decir, la detectabilidad captura la capacidad de un observador para estimar perfectamente el estado del sistema [31, 32]. El concepto de detectabilidad fue presentado por [33] el cual busca determinar los estados actuales y posteriores del sistema, basándose en una secuencia de observaciones. La observación incluye los eventos parciales y/o la observación de estados parciales.

La detectabilidad de eventos en autómatas se centran en la caracterización de la habilidad de estimar el verdadero estado actual de un determinado DES estocástico (SDES), [33, 34, 2, 31], basándose en la probabilidad de generarse secuencias de observaciones problemáticas, permitiendo de esta manera concentrarse en el comportamiento del sistema altamente probable y caracterizar la detectabilidad del sistema dado.

En [33, 34, 2], el comportamiento no-observable es detectado haciendo uso de un DES modelado en un autómata, en el cual las transiciones están asociadas a un símbolo y los estados asociados al vector de E/S, el problema de este caso surge porque no se conoce el estado inicial, dicho comportamiento es identificado mediante la propiedad de detectabilidad, la cual es definida en cuatro tipos de detectabilidades y en algunos casos el sistema es identificado adicionando estados nuevos para evitar conflictos en las secuencias.

Detectabilidad_fuerte: Un DES es detectable si se puede determinar el estado actual y el estado subsecuente, después de un número finito de observaciones E/S para todas las trayectorias del sistema.

Detectabilidad_normal: Un DES es solo detectable si se puede determinar el estado actual y el estado subsecuente, después de un número finito de observaciones E/S para algunas de las trayectorias del sistema.

Detectabilidad_periódica_fuerte: Un DES es fuertemente detectable periódicamente, si se puede determinar el estado actual y el estado subsecuente, después de un número finito de observaciones E/S para todas de las trayectorias del sistema.

Detectabilidad_periódica_normal: Un DES es detectable periódicamente, si se puede determinar el estado actual y el estado subsecuente, después de un número finito de observaciones E/S para algunas de las trayectorias del sistema.

Para [33] la detectabilidad es realizada mediante el análisis de las secuencias de símbolos $\sigma_i = \{a, c, g, d, r\}$ conocidas de un autómata G en comportamiento normal, estas secuencias son comparadas con un autómata observado G_{obs} del cual se conocen los símbolos de las transiciones observables y no observables generando secuencias de G_{obs} solamente con los símbolos observables $\sigma_{obs,i} = \{a, c, g, d, r\}$. En algunos casos particulares durante la generación de secuencias $\sigma_{obs,i}$, ocurre que en el ciclo de la secuencia

observada los últimos dos símbolos corresponden a uno observable y luego el no observable, como el último símbolo encontrado es no observable, se produce un error en la secuencia, ya que estaría indicado que el estado actual del autómata no es el mismo estado inicial de la secuencia sino que es otro impidiendo de esta forma conocer el estado actual del sistema. El anterior error es solucionado mediante la adición de un nuevo estado observable en medio de la transición no observable, completando el símbolo faltante en la secuencia $\sigma_{obs,i}$ y logrando de esta manera hacer válida la secuencia en conflicto, esto da como ventaja el convertir un modelo con comportamiento no observable en uno observable, por otra parte la desventaja es cuando el sistema es grande y se le aumenta el tamaño adicionando nuevos estados.

En [34] la detectabilidad se desarrolla mediante un análisis probabilistico de las secuencias capturadas de un observador G_{obs}, al igual que en [33] se trabaja desde un autómata G del cual se conocen los símbolos observables y los no observables, también en cada símbolo se conoce su probabilidad de ocurrencia, donde las secuencias obtenidas de G_{obs} son comparadas con el autómata G, permitiendo determinar si se adicionan símbolos observables en algunas transiciones no observables, este tipo de detectabilidad resulta útil en el sentido que se contemplan las probabilidades de ocurrencia de las secuencias de eventos sin realizar modificaciones al tamaño del modelo detectado, sino unicamente adicionando símbolos en algunas transiciones, quedando con dos o más eventos de disparo para una misma transición, esto da la desventaja de no saber cuál es el evento que ha disparado la transición.

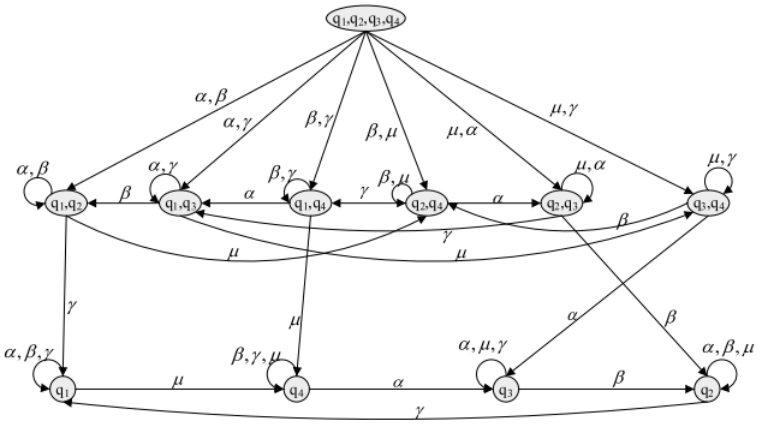

Figura 1.5: Ejemplo detector G_{det} [2].

En [2] la detectabilidad es realizada haciendo un análisis parecido al de [33], en el que se realiza un modelo de detectabilidad G_{det}, donde los estados de este modelo se componen del conjunto de estados de G que se relacionan mediante un evento. En la Figura 1.5 se muestra el ejemplo del modelo de detectabilidad suministrado en [2], el

cual solo permite determinar si G_{det} es fuertemente detectable o si es periódicamente fuerte detectable.

En [35] se realiza detectabilidad mediante el uso de un modelo de PN, en el cual las transiciones están asociadas solamente al vector de evento generado por cambios en los actuadores y los lugares asociados a los vectores generados por los sensores como lugares medibles, la detectabilidad en este caso se basa en analizar las transiciones y lugares buscando determinar, en el caso de las transiciones, si son controlables mediante la búsqueda del símbolo ε indicando que cuando el símbolo ε es encontrado se trata de una transición no controlable, para los lugares se analiza solamente vector de sensores en él se dice que un lugar P_i es medible si se encuentra un cambio en el vector de sensores.

La distinción de detectabilidad de comportamientos observables y no observables, no se ha tratado a profundidad, algunos de los métodos de identificación presentados en la sesión Subsección 1.2.2, como las propuestas de [36, 26], consideran el comportamiento observable como un modelo de PN bien conocido y el comportamiento no observable se representa como transiciones silenciosas con una etiqueta de ε. Estrada-Vargas [13], proponen un método para descubrir el comportamiento observable de una secuencia de disparo basado en el estudio de pares consecutivos de eventos de la secuencia; pero este método imposibilita encontrar dependencias a largo plazo.

1.3. Análisis

Teniendo en cuenta la revisión del estado del arte, existen dos tendencias marcadas respecto a los métodos de identificación de DES: Autómatas y PN. Cada uno de estos formalismos, tienen ventajas y desventajas en su aplicación a sistemas reales. Entre los principales enfoques de identificación de DES en lazo cerrado a partir de señales de E/S, se encuentran la generación de un autómata finito no determinista ([16, 17]) y la identificación de una IPN paramétrica [37]. Cada uno presenta una configuración de las señales diferente, lo cual genera lenguajes de diferentes estructuras, por ejemplo para un autómata autónomo no determinista con salida (NDAAO) el lenguaje obtenido consta de un conjunto de secuencias finitas representadas en palabras de un tamaño predeterminado a partir del cambio en las señales de E/S; por otra parte el lenguaje obtenido en una IPN paramétrica también está conformado por un conjunto de secuencias finitas, con la diferencia de que éstas no son formadas directamente de los cambios en las señales E/S. Estas diferencias son apreciadas en el momento del desarrollo del modelo, sobre el cual se puede apreciar dinámicas diferentes, tamaño y complejidad y en algunos casos puede llegar a ser más conveniente un enfoque sobre el otro. Por este motivo se han elegido estas dos propuestas para realizar la identificación del DES a emular.

La detectabilidad en DES es un tema que tiene amplias posibilidades de investigación, los resultados consultados y que se han obtenido a la fecha, no se han relacionado con la aplicación de la teoría de lenguajes; por lo tanto, en el presente proyecto se pretende definir una prueba de detectabilidad para un DES en lazo cerrado, caso de estudio, identificado a partir de señales de E/S del sistema con base en el lenguaje de las señales y a partir de las representaciones propuestas por [17] y [37].

2 SISTEMA DE EVENTOS DISCRETOS

Para el desarrollo de este trabajo se hace necesario tener un proceso emulado de un sistema de eventos discretos en lazo cerrado y simularlo bajo diferentes modos de funcionamiento, permitiendo así adquirir gran cantidad de señales de E/S. Esto con el objeto de aplicar algoritmos de identificación; ya que no se cuenta con procesos reales que cumplan con todos los requerimientos. Cómo el método de diseño del emulador, no es el punto principal de la presente investigación; el proceso de diseño se realiza con base a los puntos expuestos en la Figura 2.1

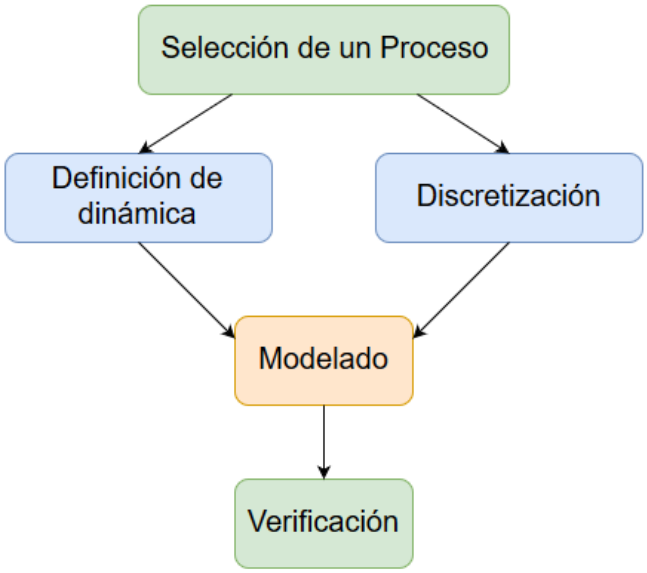

Figura 2.1: Proceso de diseño del sistema de eventos discretos.

2.1. Selección del proceso DES

Con base en diferentes fuentes de información, se decide por un sistema para la producción de baldosa cerámicas, en el cual, la mayoría de las señales se pueden discretizar y

adaptar de manera tal que entregue señales binarias de ceros y unos.

Las baldosas cerámicas son placas de poco grosor, generalmente utilizadas para revestimiento de pisos y paredes, éstas son fabricadas de arcillas y otras materias primas inorgánicas, se someten a procesos de molienda, moldeado secado y por último son cocidas a alta temperatura para que adquieran las propiedades requeridas. Son piezas cerámicas impermeables que están constituidas por un soporte cerámico, de naturaleza arcillosa, con o sin recubrimiento esmaltado. Las baldosas no esmaltadas se someten a una cocción única; las baldosas esmaltadas reciben una cubierta vitrificable entre una primera y una segunda cocción o antes de la única cocción. [38, 39]

2.1.1. Proceso de producción

El proceso de fabricación de baldosa cerámica consiste en una serie de pasos consecutivos los cuales pueden tener variaciones según el tipo de baldosa a fabricar, éstos pasos se han adaptado de [39] y se pueden resumir así:

2.1.1.1. Preparación de las materias primas

Las principales materias primas son: arcillas, feldespatos, arenas, carbonatos y caolines, estos materiales pueden variar según las propiedades que se desean en el producto final. En la industria cerámica, a las materias primas se les somete a una homogenización, esto para asegurar la uniformidad de sus características y su preparación, el proceso consiste en el mezclado de materias primas en proporciones controladas, donde pasan por una operación de molienda haciendo uso de molinos de bolas para obtener una sustancia semi-líquida uniforme llamada Barbotina, la cual luego ingresa a una operación de secado mediante columnas de atomizadores para la obtención de un fino polvo a una humedad conveniente para la operación de prensado.

2.1.1.2. Conformación y secado de la pieza

En la conformación y secado, los materiales toman forma, obteniendo una pieza en crudo del producto final, mediante el prensado de un fino polvo, resultado final de la preparación de materias primas. El prensado se realiza en seco (5-7 % de humedad), haciendo uso de prensas hidráulicas. Este procedimiento funciona conjuntamente con un mecanismo de extrusión, donde el polvo es depositado en un molde para ser seguidamente prensado.

El sistema de prensado se basa en prensas oleo dinámicas que realizan el movimiento del pistón contra la matriz, por medio de la compresión de aceite. La pieza obtenida después del prensado, es considerablemente frágil, por lo que es necesario someterla a un

secado para reducir el contenido de humedad (0,2-0,5 %), esto se logra sometiendo las piezas cerámicas a una fuerte luz ultravioleta y así entregarlas listas para las siguientes fases.

2.1.1.3. Cocción con o sin esmaltado

Dependiendo del tipo de baldosa cerámica a fabricar, se pasa o no por fases de esmaltado e impresión, de las cuales se obtienen piezas listas para ser ingresadas a un proceso de cocción, por el cual se logra obtener las propiedades y características finales deseadas.

El esmaltado consiste en la aplicación de una o varias capas de vidriado con un espesor comprendido entre 75-500 micras en total, que cubre la superficie de la pieza. Este tratamiento se realiza para conferir al producto cocido una serie de propiedades técnicas y estéticas, tales como: impermeabilidad, facilidad de limpieza, brillo, color, textura superficial, resistencia química y mecánica.

La cocción de los productos cerámicos, es una de las etapas más importantes del proceso de fabricación; ya que, de ella dependen gran parte de las características del producto cerámico: resistencia mecánica, estabilidad dimensional, resistencia a los agentes químicos, facilidad de limpieza, resistencia al fuego, etc.

Los materiales cerámicos pueden someterse a una, dos o más cocciones. Las baldosas no esmaltadas reciben una única cocción; en el caso de baldosas esmaltadas, pueden someterse a una cocción tras la aplicación del esmalte sobre las piezas crudas (proceso de mono-cocción), o someterse a una primera cocción para obtener el soporte, al que se aplica el esmalte para someterlo luego a una segunda cocción.

2.1.1.4. Clasificación y embalaje

La clasificación se realiza mediante sistemas automáticos con equipos mecánicos y visión superficial de las piezas. El resultado es un producto controlado en cuanto a su regularidad dimensional, aspecto superficial y características mecánicas y químicas. Aquellas piezas defectuosas son desechadas, y aquellas que aprueban la inspección son clasificadas según su calidad para luego ir a embalaje.

2.2. Diseño y modelado del proceso

Se considerará una planta compuesta por diferentes subsistemas y un controlador independiente por cada subsistema, los controladores pueden funcionar paralelamente en un único dispositivo de tal manera que puedan representar el comportamiento de un

PLC real, permitiendo de esta manera controlar varios sistemas aunque estos tengan o no tengan relación entre si.

Para obtener un modelo de comportamiento en cada subsistema, se realizaron las siguientes actividades:

1. División del proceso en subsistemas.

2. Descripción del funcionamiento de cada subsistema (comportamiento del controlador).

3. Definición de estados del controlador en cada subsistema.

4. Identificación y selección de señales de E/S de comportamiento binario, (sensores / actuadores), relacionadas a cada subsistema.

5. Representación gráfica de cada subsistema, con base en las señales de E/S.

Para el diseño y modelado del proceso existen varios métodos, en los modelos más usados se encuentran las máquinas de estados, los autómatas programables, las redes Petri (PN) y Sequencial Function Chart (SFC) o también conocido como Grafcet el cual fue establecido en el estándar IEC848 en 1988.

Teniendo en cuenta las características requeridas, se escoge modelar mediante el uso de máquinas de estados por su simplicidad y su capacidad de ser fácilmente exportables a diferentes lenguajes de programación. Las máquinas de estados son un caso particular, debido a que comparten características de los autómatas y las PN, con base en los modelos planteados en [30, 1] para posteriormente realizar una verificación estructural mediante la transformación de la máquina de estados en su equivalente PN.

Las máquinas de estados modelan una aproximación del comportamiento del controlador en lazo cerrado, tomando como ejemplo [40] donde diseñan en forma directa el controlador de una PN, sin modelar la planta.

Las características que estas máquinas de estados tienen son:

- Vectores de entrada para disparar las transiciones (salidas de la planta).

- Vectores de salida para generar cambios en la planta asociados a los estados (salidas del controlador).

- Se ignoran los vectores de entrada intermedios (cambios en estradas - sin cambios en salidas vistos desde la planta).

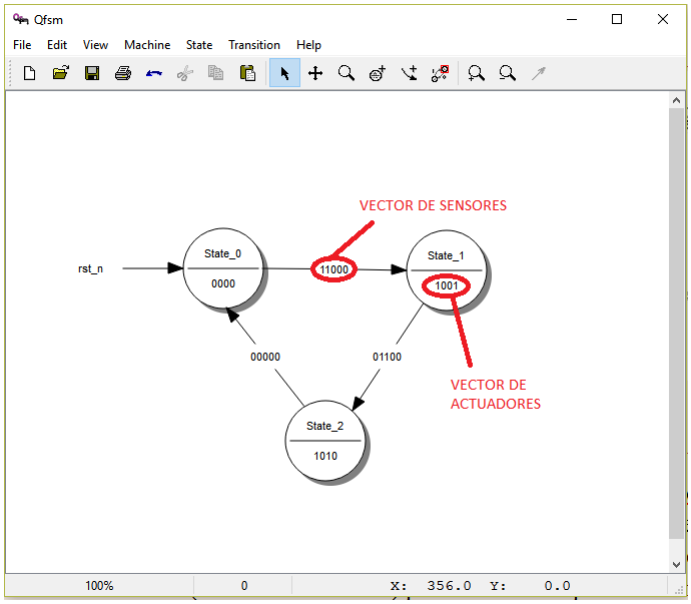

Figura 2.2: Programa QFSM.

La representación de señales se hace mediante una aplicación de software libre llamada QFSM [41] que es una herramienta gráfica para diseño y simulación de máquinas de estados finitas Figura 2.2, a partir de este programa se prueban y se diseñan los modelos de comportamiento para cada subsistema, teniendo en cuenta que los estados, al igual que las transiciones poseen su propio vector asociado.

Las señales de los vectores asociados a entradas (señales de sensores) pueden tener 3 tipos de valores, “0”, “1”, “X”; donde la “X” indica que la señal de sensor no se tiene en cuenta para disparar la transición.

Los vectores asociados a salidas (señales de control a los actuadores) corresponden a los estados, las señales de estos vectores poseen unicamente dos valores “0” o “1”, los estados se representan a partir de la orden de control generada por un controlador ante una entrada especifica de sensores, nótese que solo existen transiciones para los estados iniciales y finales de las señales de sensores y que se ignora los cambios de las señales de sensores que ocurren durante una operación (cambios intermedios que no disparan las transiciones), por ejemplo supongase que se tiene un sistema de cinco señales de entrada y cuatro señales de salida Figura 2.2, donde se tiene el estado_1 como estado actual del sistema y el controlador mantiene la orden de salida (actuadores) “1001” ante un vector de entrada (sensores) “11000”. Mientras se mantiene la orden de salida “1001” se observa que el vector de entrada evoluciona de “11000” a “11100” y luego a “01100”, donde el vector de entrada “11100” es un vector de entrada intermedio el cual no dispara la transición para cambiar de estado.

Cuando el vector de entrada evoluciona a el vector “01100” la condición de la transición

se cumple, obteniendo como respuesta del controlador del sistema el vector de salida "1010" evolucionado así el estado actual de estado_1 a estado_2.

2.2.1. División del proceso en subsistemas

El proceso escogido de producción de baldosas de cerámica se organizó de tal manera que se obtiene las siguientes ocho etapas, que definen los subsistemas:

- Suministro de Materias Prima
- Molienda
- Atomizado y Secado
- Prensado
- Girado
- Secado por Luz UV
- Impresión
- Horneado

Para la integración de las etapas se hace necesario la introducción de elementos intermedios como tanques de almacenamiento y bandas transportadoras sobre los cuales se encuentran algunos sensores, con el fin de lograr la relación entre etapas consecutivas. Para los tanques de almacenamiento la colocación de sensores, indican la presencia de material y así dar inicio al subsistema siguiente, en las bandas transportadoras los sensores representarían la presencia de material en las mismas, ya sea indicando materia prima o presencia de piezas cerámicas en un segmento de banda durante el proceso de producción.

2.2.2. Funcionamiento y modelado

Para el diseño de cada modelo se realiza un proceso iterativo en el cual se representa la máquina de estados sin tener en cuenta señales de sensores de otras etapas, usando solo las señales de los elementos que componen el subsistema o sistema objetivo, de esta manera se logra obtener el comportamiento aislado, el cual es probado de forma manual. Cuando se han verificado las máquinas de estados se procede a determinar qué señales de sensores de etapas anteriores y posteriores se deben añadir a los modelos (sensores de elementos intermedios) para obtener el modelo definitivo, logrando de esta manera la integración entre subsistemas.

Teniendo en cuenta esto para el caso de estudio se organizan la señales de E/S que se necesitan en el modelado y éstas se describen en la Tabla 2.1.

Tipo De Señal	Nombre		Tipo De Señal	Nombre
Salida	Banda1		Entrada	S0
Salida	Compuerta_Izq		Entrada	S1
Salida	Compuerta_Der		Entrada	S2
Salida	Luz_Material		Entrada	S3
Salida	Molino		Entrada	S4
Salida	Bomba1		Entrada	S5
Salida	Valvula1		Entrada	S6
Salida	Quemador1		Entrada	S7
Salida	Ventilador		Entrada	S8
Salida	Banda2		Entrada	S9
Salida	Bomba2		Entrada	S10
Salida	Banda3		Entrada	S11
Salida	Prensa_Arriba		Entrada	S12
Salida	Prensa_Abajo		Entrada	S13
Salida	Paleta_Inicio		Entrada	S14
Salida	Paleta_Fin		Entrada	S15
Salida	Valvula2		Entrada	S16
Salida	Aspiradora		Entrada	S17
Salida	Banda5		Entrada	S18
Salida	Gira_Izq		Entrada	S19
Salida	Gira_Der		Entrada	S20
Salida	Banda4_Iqz		Entrada	S21
Salida	Banda4_Der		Entrada	S22
Salida	Lampara		Entrada	S23
Salida	Banda6		Entrada	S24
Salida	Bomba3		Entrada	S25
Salida	Valvula3		Entrada	S26
Salida	Banda7		Entrada	S27
Salida	Impresora		Entrada	S28
Salida	Valvula4		Entrada	S29
Salida	Bomba4		Entrada	S30
Salida	Banda8			
Salida	Quemador2			
Salida	Quemador3			

Tabla 2.1: Tabla Asignación de Señales de Salida

2.2.2.1. Suministro de materias primas

El subsistema de la etapa de suministro es el encargado de transportar, de manera continua, las materias primas desde una tolva hasta el subsistema de molienda, se compone de una banda transportadora, tolva, compuerta y una luz indicadora de petición de material ubicada en la tolva ver Figura 2.3.

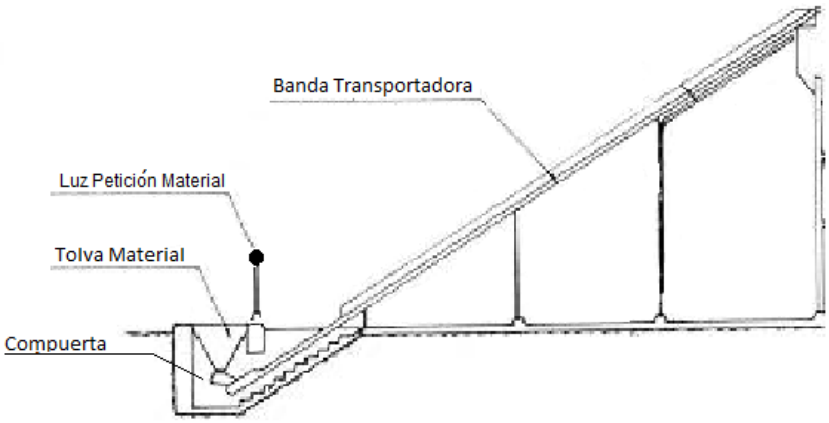

Figura 2.3: Diagrama etapa de suministro de materia prima

Este subsistema tiene un sensor global de encendido, seis sensores principales y dos sensores externos provenientes de etapas posteriores para representar la capacidad de almacenaje de barbotina en el subsistema posterior, permitiendo controlar un encendido o apagado en función de la capacidad disponible de barbotina, este subsistema es manejado desde una unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que componen el subsistema, mostrados en la Figura 2.4.

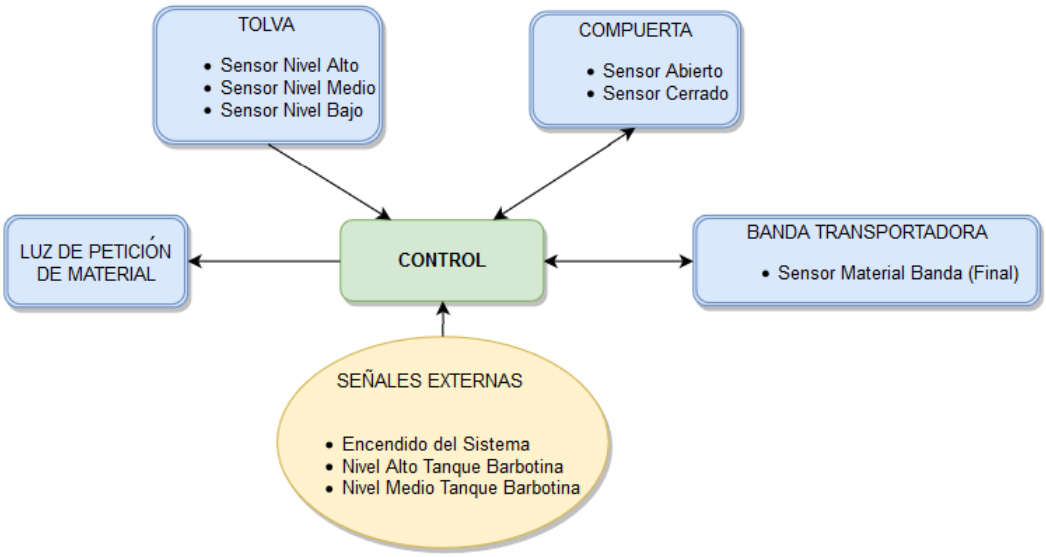

Figura 2.4: Diagrama de señales etapa de suministro de materia prima

Las señales de E/S que formarán parte de la lógica de comportamiento del subsistema en la máquina de estados se organizan así:

- Señales de entrada

  S0: Encender/Apagar

  S1: Sensor Material Inferior Tolva

  S2: Sensor Material Medio Tolva

  S3: Sensor Material Superior Tolva

  S4: Sensor Compuerta Cerrada

  S5: Sensor Compuerta Abierta

  S9: Sensor Nivel Alto Tanque Grande

  S10: Sensor Nivel Medio Tanque Grande

- Señales de salida

  Banda1: Banda Transportadora 1

  Compuerta_Izq: Mecanismo de Compuerta

  Compuerta_Der:

  Luz_Material: Luz Piloto Indicador de Petición de Material

El funcionamiento del subsistema diseñado es, una vez la orden global de encendido S0 se ha dado, se analiza el nivel de llenado de la tolva S1, S2 y S3. Si ésta no se

encuentra totalmente llena se enciende la luz de petición de material, una vez la tolva se ha llenado, la luz se apaga y la banda transportadora 1 da inicio mientras al mismo tiempo la compuerta empieza a abrirse, una vez que la compuerta ha alcanzado cierto nivel de apertura el material que se encuentra en la tolva empieza a fluir continuamente por la banda transportadora.

Cuando se da la orden de apagado S0 o también cuando el tanque de Barbotina ha alcanzado el nivel máximo S9, la unidad de control, da la orden de cerrar compuerta, mientras la banda sigue funcionando hasta que no haya material en el final de banda, esto es observado por la señal S6.

Las señales de salida que corresponden al mecanismo de Compuerta hacen referencia a las dos entradas de alimentación de un motor, el cual hace de actuador para abrir y cerrar la compuerta, donde la unidad de control identifica el estado de apertura de la compuerta usando las señales provenientes de S4 y S5.

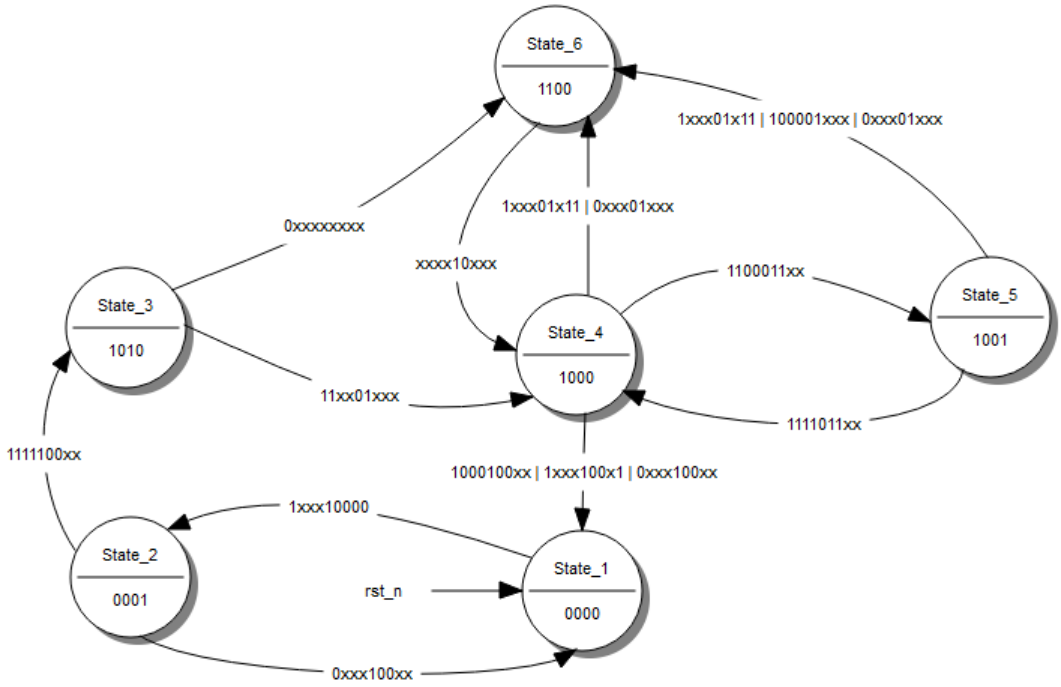

Figura 2.5: Modelo máquina de estados etapa suministro de materiales

En la Figura 2.5 se aprecia el diagrama de la máquina de estados que describe el comportamiento diseñado, las transiciones entre estados pueden ser de dos tipos, primero el vector completo de señales de sensor, esta forma indica que para disparar se necesitan de todas las señales que el subsistema tiene, la otra forma es un vector parcial de señales de sensor en el que se conserva el tamaño de dicho vector pero algunos de sus

valores pueden contener el caracter “X” mencionado anteriormente en la Sección 2.2. El significado de los estados es:

- Estados

  Estado1: Sistema apagado.

  Estado2: Tolva llenándose.

  Estado3: Banda encendida. Abriendo compuerta.

  Estado4: Sin cambios mientras hay material en tolva.

  Estado5: Tolva llenándose y pidiendo Material.

  Estado6: Cerrando compuerta.

2.2.2.2. Molienda de material

El subsistema de la etapa de molienda se encarga de recibir el material que se obtiene de la etapa de suministro, se adiciona agua para luego molerlo, entregando el material resultante conocido como Barbotina a un tanque grande de almacenamiento.

Este subsistema se compone principalmente de un molino, una válvula de paso de agua, un tanque pequeño o temporal para almacenar el material que va saliendo del molino, una bomba para impulsar la Barbotina una vez que el tanque temporal se ha llenado y un tanque grande para almacenar la Barbotina que va a ser usada en la próxima etapa de atomizado.

Para describir su funcionamiento se hace uso de un sensor global de encendido, cinco sensores principales y un sensor externo proveniente de la etapa anterior, específicamente la banda transportadora, este subsistema es manejado desde una unidad de control independiente, encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.6.

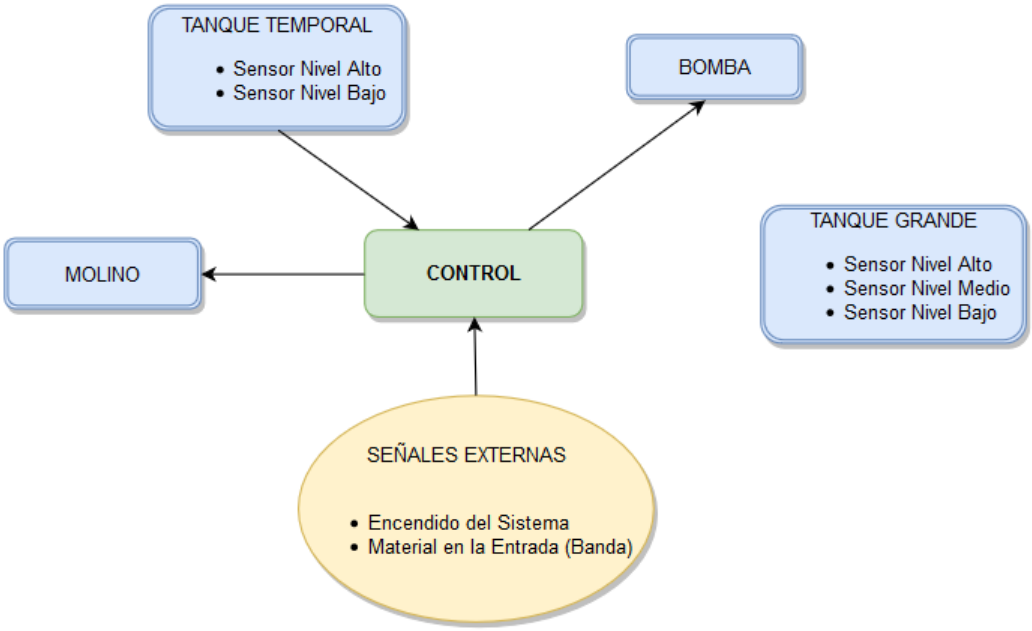

Figura 2.6: Diagrama de señales etapa de molienda

Las señales de E/S, se definieron así:

- Señales de entrada

  S0: Encender/Apagar.

  S6: Sensor material en Banda1.

  S7: Sensor nivel Tanque Temporal alto.

  S8: Sensor nivel Tanque Temporal bajo.

  S9: Sensor nivel Tanque Grande Alto.

  S10: Sensor nivel Tanque Grande Medio.

  S11: Sensor nivel Tanque Grande Bajo.

- Señales de salida

  Molino.

  Bomba1.

  Válvula1: Válvula de Suministro de Agua.

El funcionamiento del subsistema es: Una vez la orden global de encendido S0 se ha dado y que en la banda de la etapa de suministro exista material S6, la válvula empieza

a suministrar agua y al mismo tiempo el molino da inicio, El subsistema se mantiene en este estado hasta que el material que sale del molino (Barbotina) ha llenado el tanque temporal colocando en alto la señal S7, esto da inicio a la bomba, la cual se mantiene encendida hasta que el tanque temporal se ha quedado vacío dejando la señal S8 en bajo y así desplazar la Barbotina hacia el tanque grande de almacenamiento.

Una vez que el sistema detecta que la señal S6 se coloca en bajo, se procede a vaciar el tanque temporal y ya no se suministra agua en la entrada, eventualmente el molino se apaga. Las señales provenientes del tanque grande S9, S10 y S11 no se tienen en cuenta para el diseño de la máquina de estados de este subsistema.

Hay que aclarar que el subsistema se mantiene encendido siempre y cuando exista material entrante en la banda transportadora del subsistema previo y que para que exista material en la banda, el tanque grande de almacenamiento de barbotina debe tener capacidad disponible, lo que quiere decir que existe una dependencia con la presencia de material en la banda de suministro para determinar si el subsistema debe seguir encendido o debe realizar la secuencia de apagado, también la capacidad máxima de almacenado del tanque grande tiene un margen extra, ya que durante el lapso de apagado tanto de suministro como de molienda, existe algo de material en el proceso anterior.

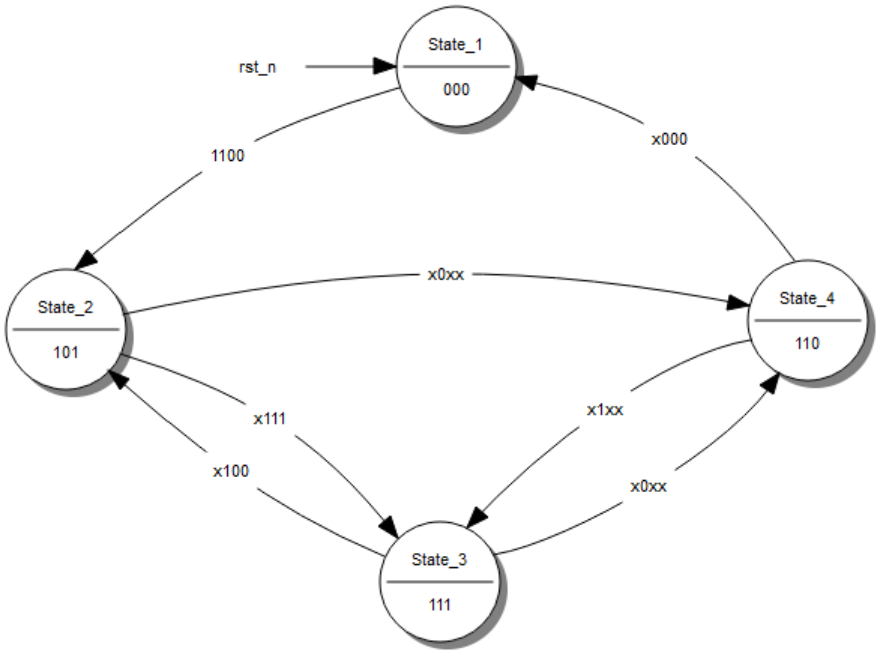

Figura 2.7: Modelo máquina de estados etapa molienda de material

En la Figura 2.7 se aprecia el diagrama de la máquina de estados diseñada y los estados se definen como:

- Estados

  Estado1: Molino apagado.

  Estado2: Molino encendido mientras se suministra agua. Bomba apagada.

  Estado3: Bomba encendida.

  Estado4: Suministro de agua detenido mientras bomba sigue funcionando.

2.2.2.3. Atomizado y secado

El subsistema de la etapa de atomizado y secado se encarga de tomar material del tanque de almacenamiento de Barbotina, rociarlo en el interior de una cámara en la que el aire es calentado para que las pequeñas gotas de Barbotina se conviertan en un fino polvo, el cual es transportado a un tanque de almacenaje para dicho polvo.

Este subsistema se compone principalmente de un columna de secado, un quemador de gas, ventilador, dos bandas transportadoras y un tanque para almacenar el polvo producido ver Figura 2.8.

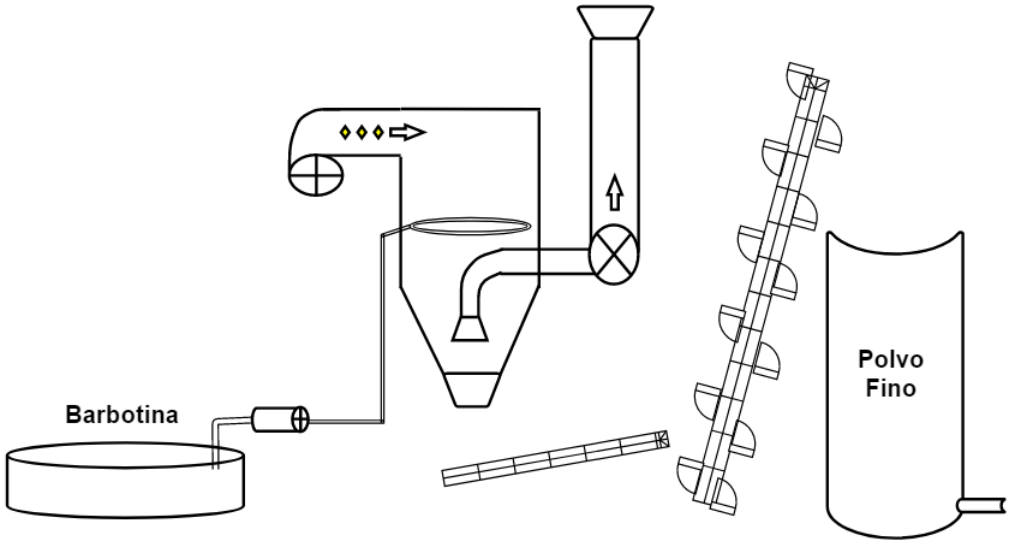

Figura 2.8: Diagrama etapa de atomizado y secado

Para su funcionamiento hace uso de un sensor global de encendido, cinco sensores principales y un sensor externo proveniente de la etapa anterior manejado desde una unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.9.

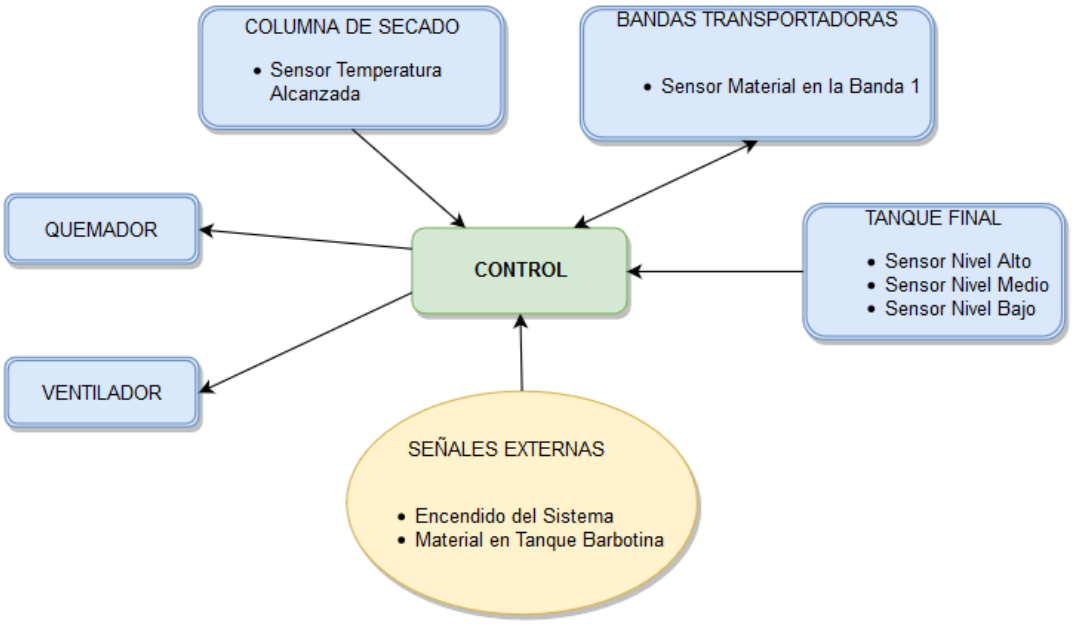

Figura 2.9: Diagrama de señales etapa de atomizado y secado

Para ello se organizaron tanto las señales de entradas como de salidas que formarán parte de la lógica de comportamiento del subsistema en la máquina de estados diseñada.

- Señales de entrada

  S0: Encender/Apagar.

  S11: Sensor nivel bajo tanque almacenaje Barbotina.

  S12: Presencia de material en la banda transportadora.

  S13: Sensor Setpoint o referencia de la temperatura.

  S14: Sensor Tanque Polvillo Nivel Bajo.

  S15: Sensor Tanque Polvillo Nivel Medio.

  S16: Sensor Tanque Polvillo Nivel Alto.

- Señales de salida

  Quemador1.

  Ventilador.

  Banda2.

  Bomba2.

  Banda3.

El funcionamiento diseñado para esta etapa es: Una vez la orden global de encendido S0 se ha dado siempre y cuando exista material con que trabajar S11, el quemador y el ventilador pueden dar inicio. Cuando la temperatura alcanza el setpoint establecido S13 se coloca en alto, dando lugar al encendido de la banda transportadora 2 y del encendido de la bomba 2 para impulsar la Barbotina de tanque de almacenaje al interior de la columna de secado, de esta manera producirá un fino polvo, el cual cae sobre la banda transportadora 2, ésta a su vez, en la parte final dispone de un sensor S12, que es usado para indicar la presencia de material, cuando este sensor se encuentra en estado alto la banda transportadora 3, da inicio para depositar el polvo en el tanque de almacenaje que va a ser usado en la próxima etapa de prensado.

El subsistema se mantiene en funcionamiento siempre que el tanque final no se llene, esto se detecta a partir de las señales S15 y S16, donde una vez que se llena, se mantiene apagado o en espera hasta que la señal de nivel medio S15 cambie a estado bajo.

Para el diseño de la máquina de estados la señal S14 no se tiene en cuenta ya que el nivel bajo solo le interesa a la etapa que sigue.

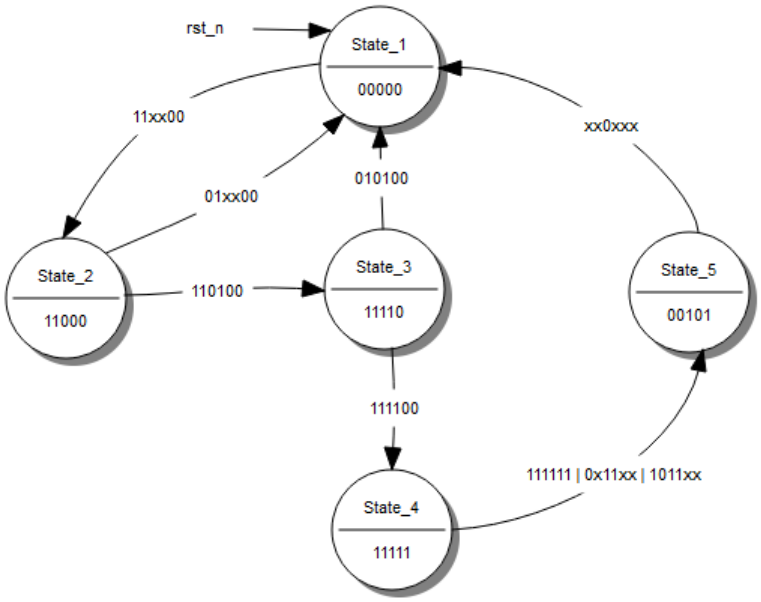

Figura 2.10: Modelo máquina de estados etapa atomizado y secado

En la Figura 2.10, se aprecia el diagrama de la máquina de estados diseñada, cuyos estados significan:

- Estados

  Estado1: Sistema apagado.

  Estado2: Ventilador y quemador encendidos.

  Estado3: Todo Encendido excepto bandas.

  Estado4: Todo encendido.

  Estado5: Solo bandas encendidas.

2.2.2.4. Prensado

En el subsistema de prensado las baldosas cerámicas empiezan a tomar forma, éste se compone de dos partes mecánicas: una la prensa misma, y la otra es un mecanismo que se encarga de retirar la cerámica recién hecha de la prensa y de dejar nuevo material para la próxima pieza cerámica Figura 2.11. Esta última dispone de una válvula que permite o no, el paso del polvo que es succionado por una aspiradora para ser depositado en el interior de la prensa.

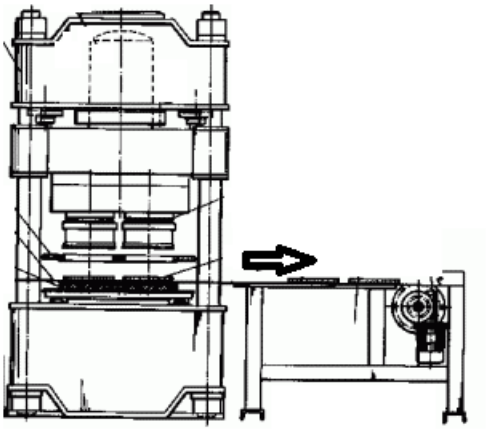

Figura 2.11: Diagrama etapa de prensado

Este subsistema usa un sensor global de encendido, cinco sensores principales y dos sensores externos uno proveniente de la etapa anterior relacionado con la disponibilidad de material en el tanque del fino polvo y el otro de un sensor de Setpoint o referencia de temperatura del subsistema de horneado, además está conformado por una unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.12.

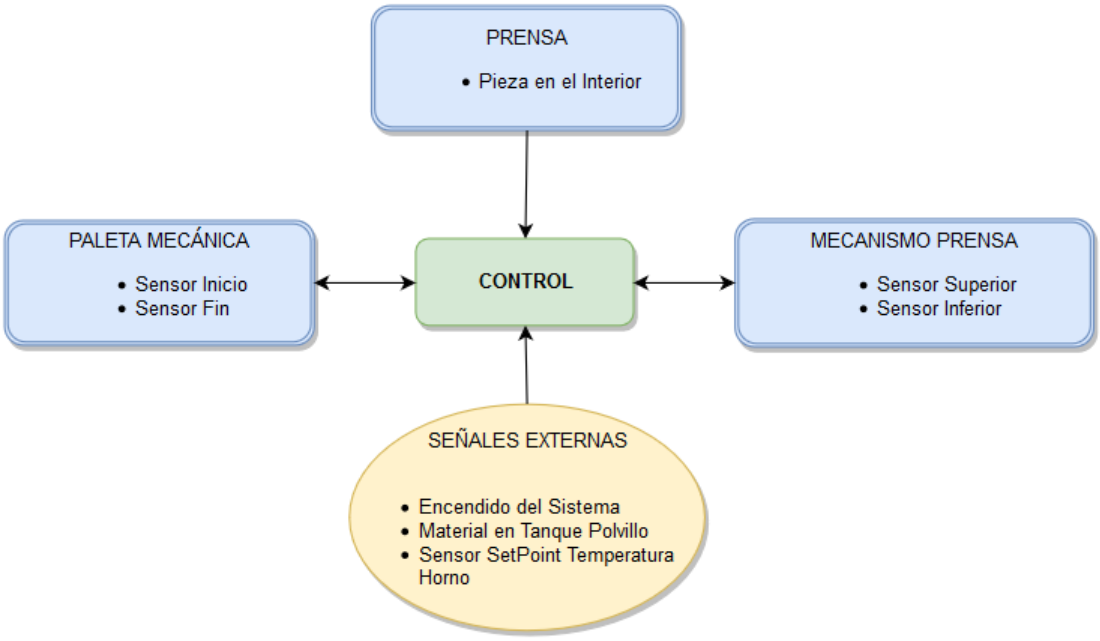

Figura 2.12: Diagrama de señales etapa de prensado

Las señales de E/S que formaran parte de la lógica de comportamiento del subsistema en la máquina de estados diseñada, se definieron así:.

- Señales de entrada

  S0: Encender/Apagar.

  S14 Sensor Nivel Bajo Tanque Polvillo.

  S17: Sensor Prensa Arriba.

  S18: Sensor Prensa Abajo.

  S19: Sensor Paleta Inicio.

  S20: Sensor Paleta Fin.

  S21: Sensor Presencia Objeto (Cerámica).

  S30 Sensor Setpoint Alcanzado Temperatura Horno.

- Señales de salida

  Prensa_Arriba Mecanismo de Prensa.

  Prensa_Abajo.

  Paleta_Inicio Mecanismo de Retirar Pieza.

  Paleta_Fin.

  Válvula2.

  Aspiradora.

El funcionamiento de esta etapa es: Una vez la orden global de encendido S0 se ha dado, existe material con que trabajar S14 y el SetPoint de temperatura sea alcanzado en el Horno S30, la prensa podrá dar inicio, haciendo que la paleta mecánica encargada retire la pieza que se encuentre en el interior, se desplace a la posición final S20, mientras la válvula y la aspiradora depositan material S21 en la prensa. Cuando la paleta mecánica llega a la posición final, ésta regresa a donde estaba S19 para luego iniciar el descenso de la prensa y crear la pieza cerámica.

Este proceso se repite continuamente y solo se detiene cuando se da la señal de apagado o cuando el material con que trabajar se acabe S14 en estado bajo.

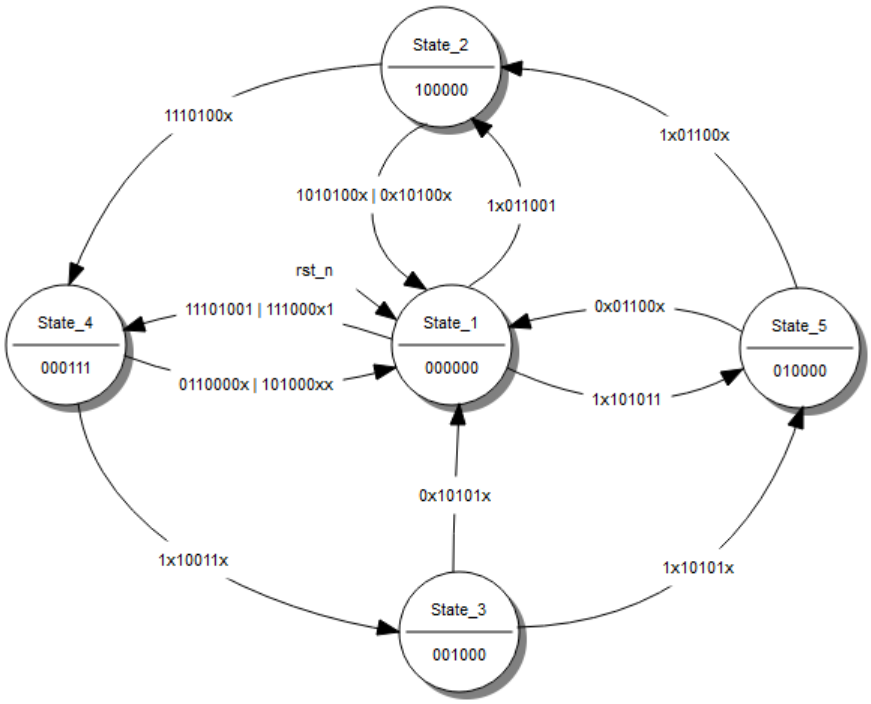

Figura 2.13: Modelo máquina de estados etapa prensado

En la Figura 2.13 se aprecia el diagrama de la máquina de estados, cuyos estados se describen como:

- Estados

  Estado1: Sistema apagado.

  Estado2: Prensa subiendo.

  Estado3: Mecanismo regresando a posición inicial.

  Estado4: Mecanismo moviéndose a posición final. Válvula abierta.

  Estado5: Prensa bajando.

2.2.2.5. Girado

En esta etapa las piezas son recibidas para ser colocadas al derecho, ya que la prensa entrega la pieza cerámica al revés, este subsistema se compone de dos bandas transportadoras y un mecanismo de girado Figura 2.14 .

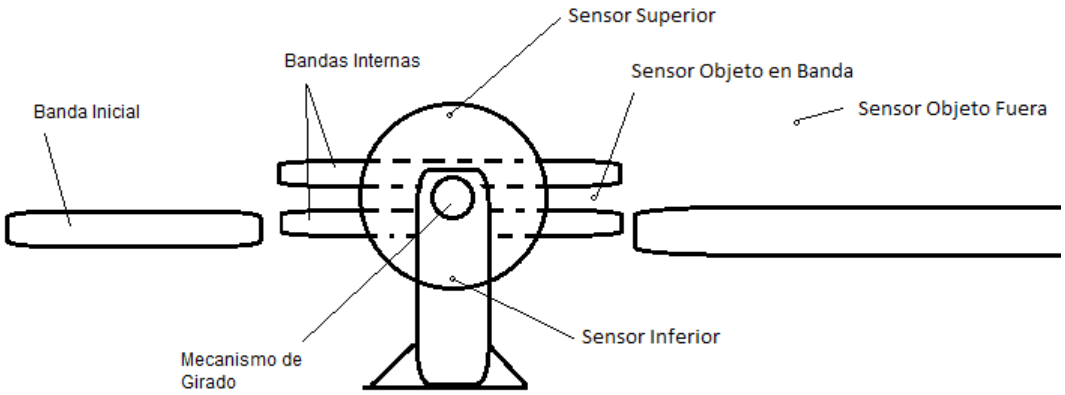

Figura 2.14: Diagrama etapa de girado

Este subsistema tiene un sensor global de encendido, tres sensores principales y dos sensores externos provenientes de etapas posteriores relacionados con la salida de una pieza cerámica del mecanismo y del Setpoint o referencia de temperatura alcanzada en la etapa de horneado; además, está conformado por una unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.15.

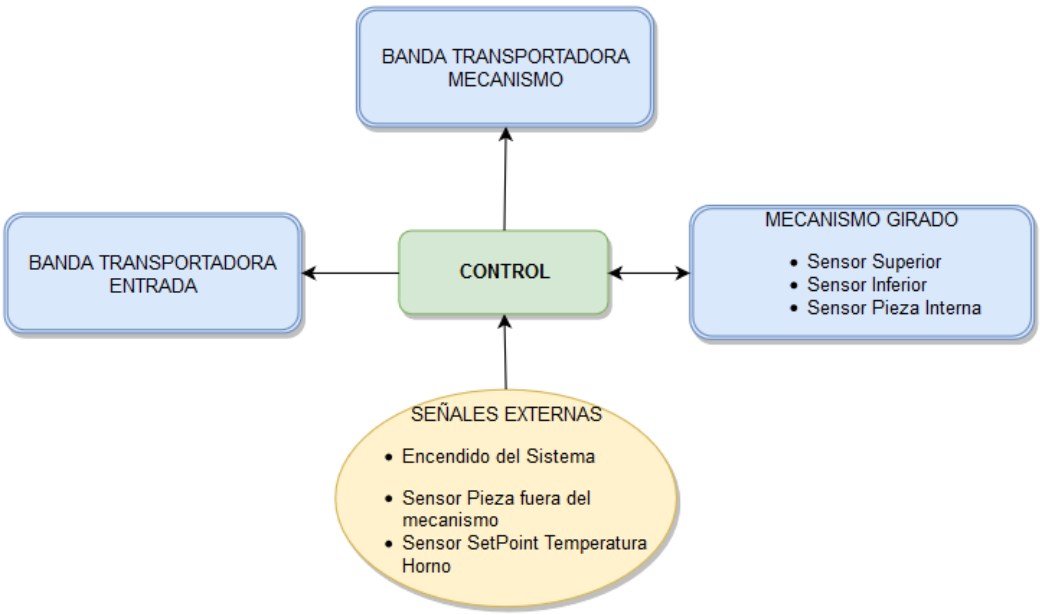

Figura 2.15: Diagrama de señales etapa de girado

Las señales de E/S forman parte de la lógica de comportamiento del subsistema en la máquina de estados diseñada.

- Señales de entrada

  S0: Encender/Apagar.

  S22: Sensor Superior Mecanismo.

  S23: Sensor Inferior Mecanismo.

  S24: Sensor Presencia dentro de Mecanismo.

  S25: Sensor Presencia fuera de Mecanismo.

  S30 Sensor SetPoint Alcanzado Temperatura Horno.

- Señales de salida

  Banda5 Banda Transportadora 5.

  Gira_Izq Mecanismo de Girado.

  Gira_Der

  Banda4_Iqz Banda Transportadora 4.

  Banda4_Der

El funcionamiento de este subsistema es: Una vez que recibe la orden de encendido S0 se mantiene en espera y no da inicio hasta que la temperatura del horno haya sido alcanzada S30, una vez que estas dos condiciones se han cumplido la banda transportadora 5, que es la banda transportadora que recibe las piezas entregadas por la prensa, entra a funcionar al igual que la banda transportadora 4.

Cuando el sensor que se encuentra en el interior del mecanismo S24, detecta una pieza cerámica, inmediatamente la banda transportadora 4 se detienen y el mecanismo gira 180°, colocando en estado bajo el sensor superior del mecanismo de girado S22, una vez que el sensor inferior S23 se coloca en estado alto, la banda transportadora 4 gira en sentido contrario hasta que el sensor S25, que indica que la pieza cerámica se encuentra fuera del mecanismo, se coloca en alto, indicando que es seguro girar sin dañar la pieza cerámica para regresar el mecanismo de girado a su posición original, esperando la llegada de una nueva pieza repitiendo este proceso sucesivamente.

En el momento que se da la señal de apagado S0 en bajo, todas las señales de salida son colocadas en estado bajo, de esta manera el comportamiento del subsistema diseñado tiende a detenerse inmediatamente en la posición en que se encuentre.

Las señales de salidas tanto para banda transportadora 4, como para el mecanismo de girado corresponden a las de polarizado de motores para cambiar el sentido de giro.

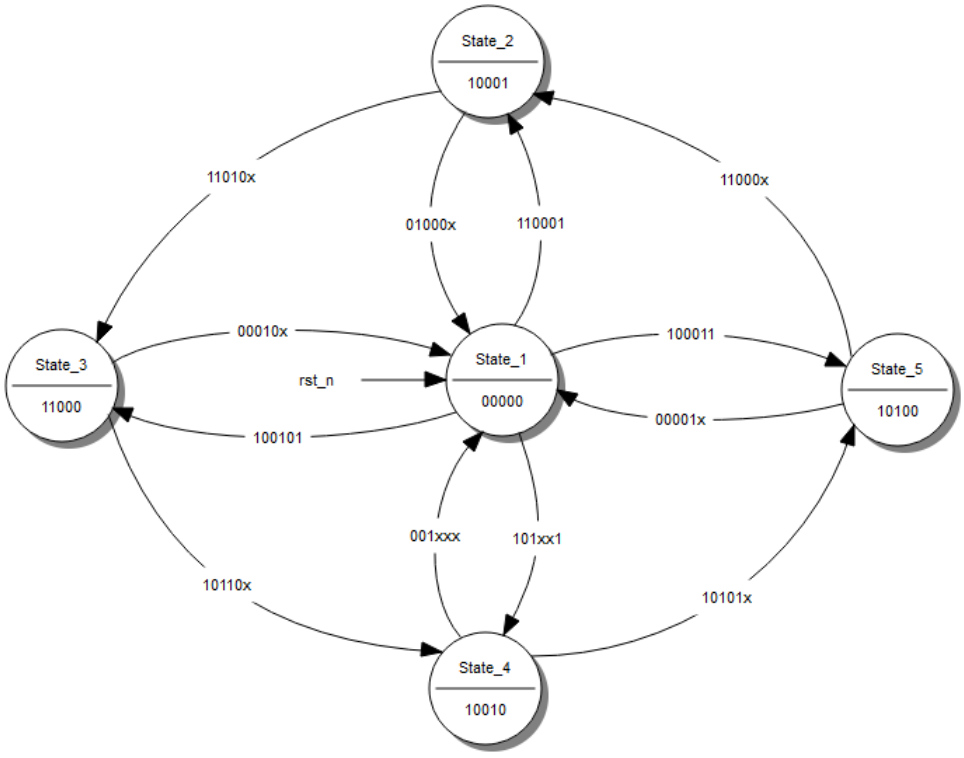

Figura 2.16: Modelo máquina de estados etapa girado

En la Figura 2.16 se aprecia el diagrama de la máquina de estados diseñada. Sus estados asociados son:

- Estados

  Estado1: Sistema apagado.

  Estado2: Bandas girando normalmente.

  Estado3: Bandas apagadas. Mecanismo girando a izquierda.

  Estado4: Bandas girando inversamente.

  Estado5: Bandas apagadas. Mecanismo girando a derecha.

2.2.2.6. Secado por luz UV

El subsistema de la etapa de secado por luz ultravioleta es el encargado de retirar los restos de humedad que quedan en la cerámica después de que sale de la prensa y es girada al derecho. Se compone de un grupo de lámparas UV ubicadas en una cámara, que es atravesada por una banda transportadora en la cual en su parte final rocía levemente con agua la pieza cerámica para bajar la temperatura.

Este subsistema comprende: Un sensor global de encendido, dos sensores principales y un sensor externo proveniente de una etapa posterior relacionada con el Setpoint o referencia de temperatura alcanzada en la etapa de horneado, también está conformado por una unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.17.

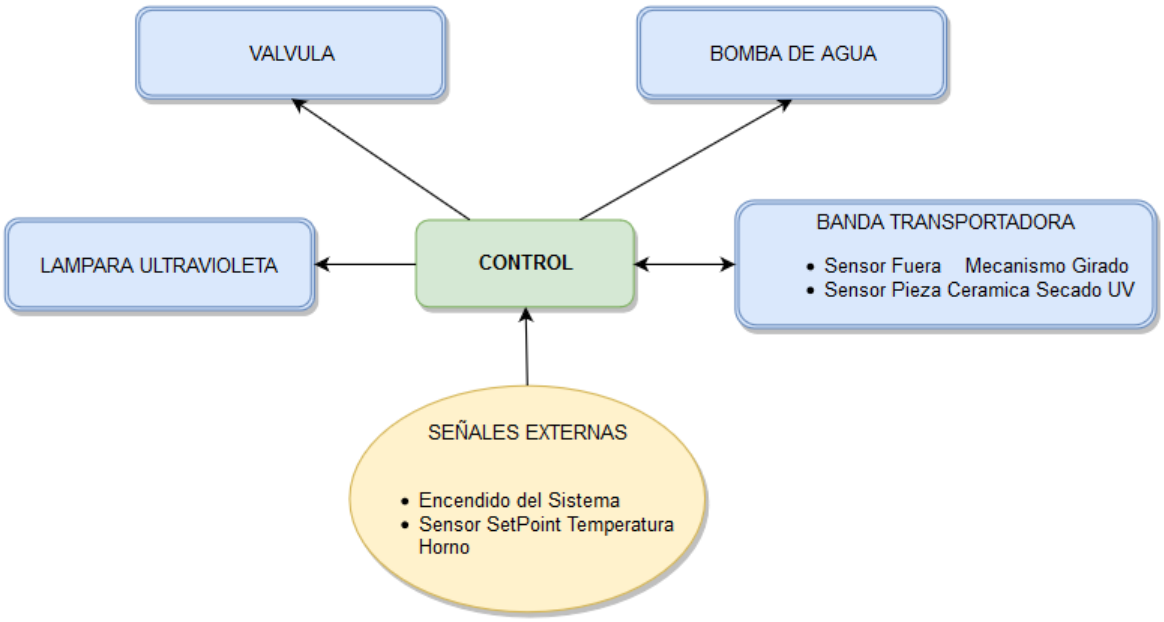

Figura 2.17: Diagrama de señales etapa de secado por luz UV

Las señales de E/S que forman parte de la lógica de comportamiento del subsistema en la máquina de estados diseñada son:

- Señales de entrada

  S0: Encender/Apagar.

  S25: Sensor presencia fuera de Mecanismo.

  S26: Sensor presencia objeto (Cerámica).

  S30 Sensor Setpoint Alcanzado Temperatura Horno.

- Señales de salida

  Lámpara.

  Banda6.

  Bomba3.

  Válvula3.

La dinámica de comportamiento del subsistema diseñado consiste en que una vez el sensor de presencia fuera del mecanismo S25 está en alto, las lámparas se encienden, en ese trayecto, la cerámica ya se encuentra en el interior de la cámara, y hasta que el sensor en el interior no se coloque en bajo, las lámparas seguirán encendidas, además

ese sensor en el interior cumple otra función, ya que si por alguna razón se da la señal de apagado y se enciende luego, detectará la presencia de una pieza cerámica, entonces encenderá las Lámparas hasta que salga la cerámica, colocando la señal del sensor S26 en bajo.

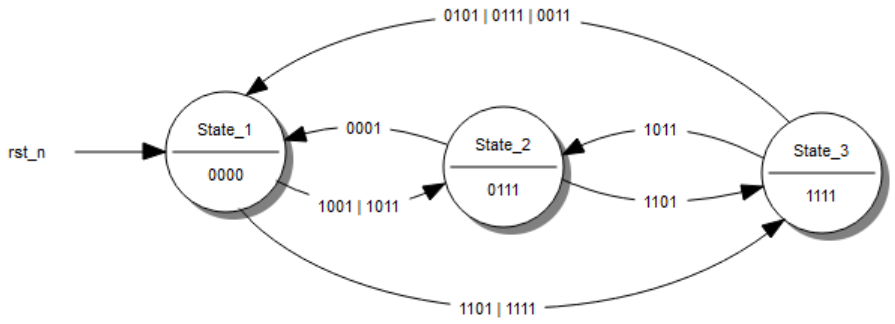

Figura 2.18: Modelo máquina de estados etapa secado por luz UV

En la Figura Figura 2.18 se aprecia el diagrama de la máquina de estados diseñada. Los estados asociados son:

- Estados

  Estado1: Sistema Apagado.

  Estado2: Bandas, Bomba y Válvula Activadas, Lámpara Apagada.

  Estado3: Lámparas Encendida.

2.2.2.7. Impresión

El subsistema de la etapa de impresión es el encargado de grabar o plasmar una imagen o color en las piezas cerámicas para darle un acabado deseado. Se compone principalmente de la impresora, de una válvula y una bomba para aplicar una resina base mientras se desplaza en la banda transportadora, para luego si ingresar a la impresora.

Esta etapa es gobernada por dos sensores principales uno a la entrada de la impresora y otro dentro de la impresora, también depende del sensor global de encendido y del sensor externo proveniente de una etapa posterior relacionada con el Setpoint o referencia de temperatura alcanzada en la etapa de horneado. De igual forma está conformado por una unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.19.

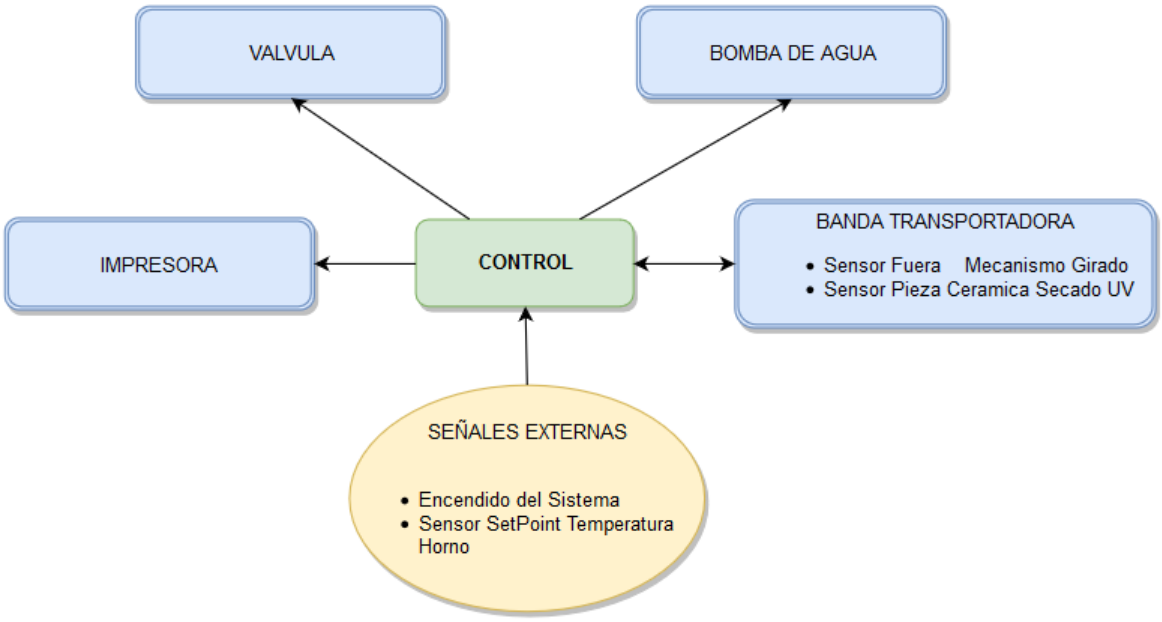

Figura 2.19: Diagrama de señales etapa de impresión

Las señales de E/S que formarán parte de la lógica de comportamiento del subsistema en la máquina de estados diseñada son:

- Señales de entrada

  S0: Encender/Apagar.

  S27: Sensor presencia fuera de la impresora.

  S28: Sensor presencia objeto (Cerámica).

  S30 Sensor Setpoint alcanzado temperatura horno.

- Señales de salida

  Banda7.

  Impresora.

  Válvula4.

  Bomba4.

La dinámica de comportamiento del subsistema es: Una vez el sensor de presencia fuera de la impresora S27 está en alto la impresora da inicio, en ese trayecto, la cerámica ya se encuentra en el interior de la cámara de impresión, y hasta que el sensor en el interior no se coloque en bajo, la impresora seguirá funcionando; además, ese sensor en el interior cumple otra función: si por alguna razón se da la señal de apagado y se

enciende luego, detectará la presencia de una pieza cerámica, entonces encenderá la impresora hasta que salga la cerámica colocando la señal del sensor S28 en bajo.

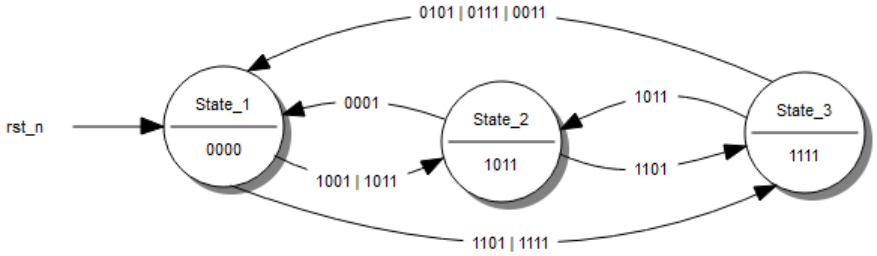

Figura 2.20: Modelo máquina de Estados Etapa Impresión

En la Figura 2.20 se aprecia el diagrama de la máquina de estados diseñado, cuyos estados son:

- Estados

  Estado1: Sistema Apagado.

  Estado2: Bandas, bomba y válvula Activas, Impresora Apagada.

  Estado3: Impresora Encendida.

2.2.2.8. Horneado

El subsistema de la etapa de horneado es el más simple; pero vital de todo el proceso de producción; ya que, en esta etapa la pieza cerámica adquiere las propiedades de resistencia y durabilidad que se esperan de la pieza como producto final. Está compuesta por el horno mismo, una banda resistente a las altas temperaturas y de quemadores para mantener una temperatura adecuada en el interior del horno Figura 2.21.

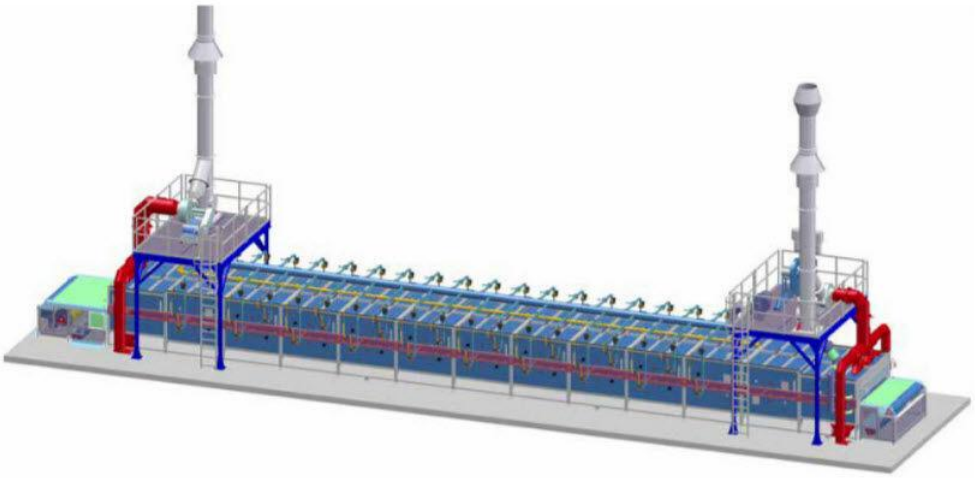

Figura 2.21: Diagrama horno de banda transportadora

Su funcionamiento se basa en el sensor global de encendido y dos sensores principales uno para detectar la presencia de piezas cerámicas en el interior del horno y otro para el Setpoint de temperatura en el interior del horno. El subsistema es manejado por unidad de control independiente encargada de activar o desactivar cada uno de los elementos físicos que lo componen ver Figura 2.22.

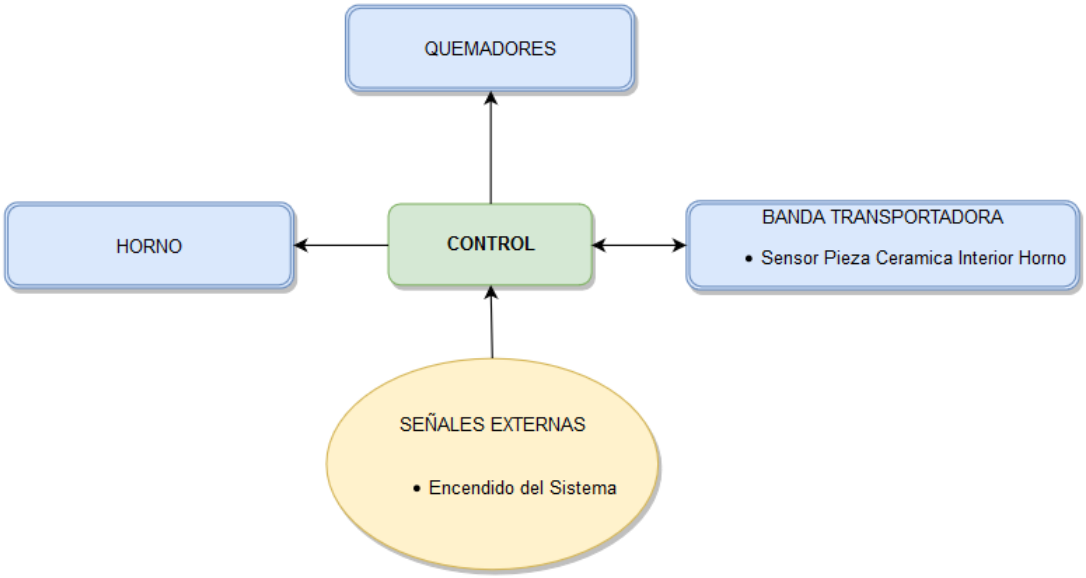

Figura 2.22: Diagrama de señales etapa de horneado

Las señales de E/S que forman la parte de la lógica de comportamiento del subsistema en la máquina de estados diseñada son:

- Señales de entrada

  S0: Encender/Apagar.

  S29: Sensor Pieza Cerámica Interior Horno.

  S30 Sensor Setpoint Alcanzado Temperatura Horno.

- Señales de salida

  Banda8.

  Quemador2.

  Quemador3.

El horno consiste en una cámara de gran longitud que puede llegar a medir 30 metros de largo, durante el trayecto, de manera continua, se desplazan piezas cerámicas a la vez que se van cociendo hasta dar el acabado final.

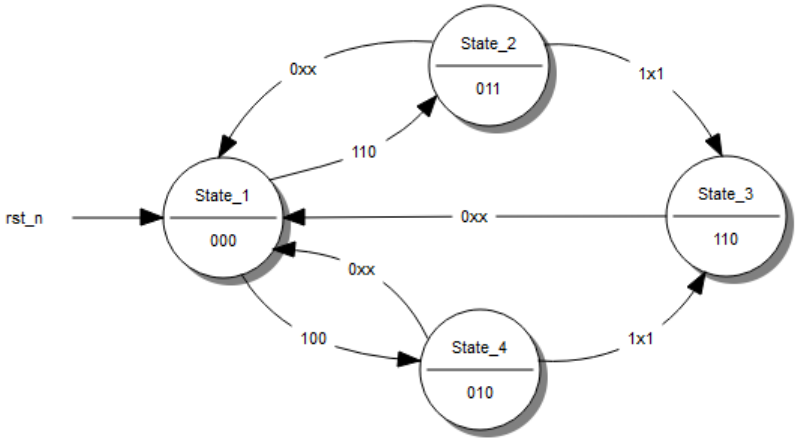

Figura 2.23: Modelo máquina de estados etapa horneado

En la figure Figura 2.23 se aprecia el diagrama de la máquina de estados diseñada. Los estados son:

- Estados

  Estado1: Sistema apagado.

  Estado2: Banda interna apagada. Horno en encendido rápido (2 Quemadores).

  Estado3: Banda interna encendida. Horno encendido normal (1 Quemador).

  Estado4: Banda interna apagada. Horno encendido normal (1 Quemador).

2.2.3. Verificación de integración entre modelo

Para verificar que la integración de los modelos de máquinas estados obtenidos corresponden al comportamiento ideal del proceso de producción, se analiza cada subsistema generando un cambio manual en las señales de entrada (sensores) a manera de simular un evento de planta como puede ser el llenado de un tanque o la presencia de un objeto, buscando encontrar un cambio de estado en el subsistema partiendo desde el estado inicial, si no existe cambio desde el estado inicial al estado siguiente, significa que en una de las transiciones de salida del estado actual, el vector de entrada puede estar mal como también puede que al estar en dicho estado el cambio de la señal no lo afecte, esto se realiza para cada uno estados que tiene el subsistema siguiente.

2.2.4. Verificación de Propiedades estructurales

En los modelos de máquinas de estados diseñados se puede apreciar que la principal característica de éstos es que sus señales son de naturaleza binaria, esto permite que a partir de un modelo de máquina de estado se pueda obtener un modelo equivalente de autómata o de PN. Las PN asociadas a la máquinas de estados diseñadas, cumplen con las siguientes características:

- Los estados en las máquinas de estados son equivalentes a los lugares en una PN, donde los vectores de salidas corresponden a las ordenes o tareas que se realizan en los lugares de una PN.
- Las transiciones en las máquinas de estados están asociadas a un vector de tamaño uniforme el cual es el mismo vector asociado para disparo de las transiciones de la PN, este vector es cambiado de manera tal que aquellas señales con carácter “X” son ignoradas y no se colocan en la función asociada a la transición de la PN.
- Los arcos de la PN se obtienen a partir de las conexiones realizadas por las transiciones en las máquinas de estados, conectando los lugares previos y posteriores a cada transición.

Con base a las 3 características expuestas se obtienen los modelos de PN asociados a los controladores de los subsistemas, donde los pesos de los arcos son siempre “1”, no existen auto bucles debido a la forma en que se diseñaron las máquinas de estados, también hay ausencia de transiciones sumidero porque durante el diseño de las máquinas de estado se tiene en cuenta que siempre existan ciclos internos en los que se pueda retornar a cualquier lugar incluso al lugar inicial de los subsistemas.

Las PN obtenidas con su tabla de funciones de disparo para las transiciones, se muestran en las figuras, Figura 2.24, Figura 2.25, Figura 2.26, Figura 2.27, Figura 2.28, Figura 2.29, Figura 2.30, Figura 2.31.

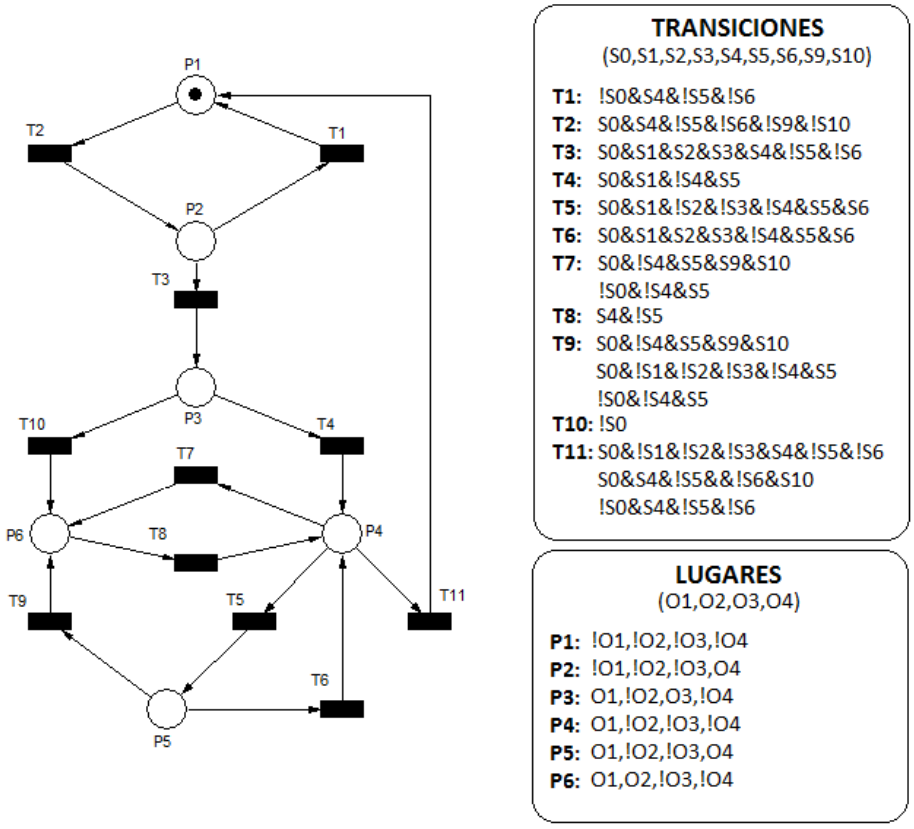

Figura 2.24: PN Suministro de materia prima

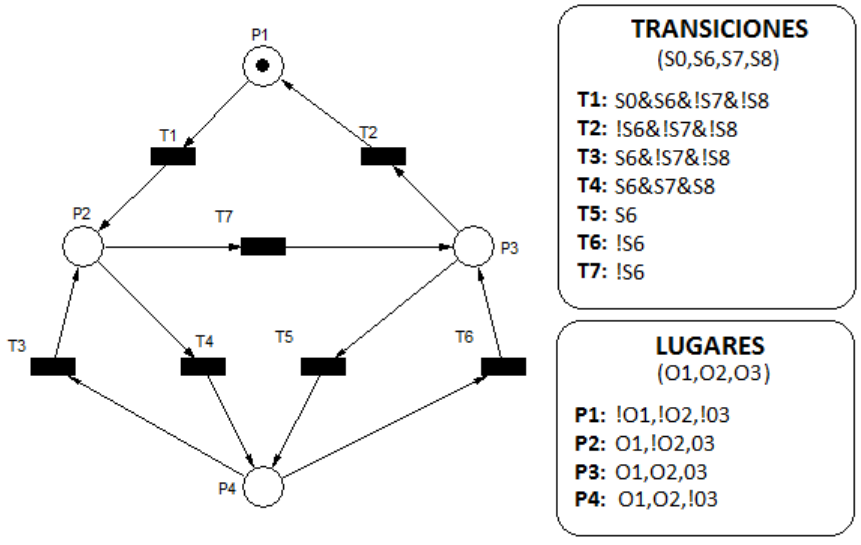

Figura 2.25: PN Molienda de material

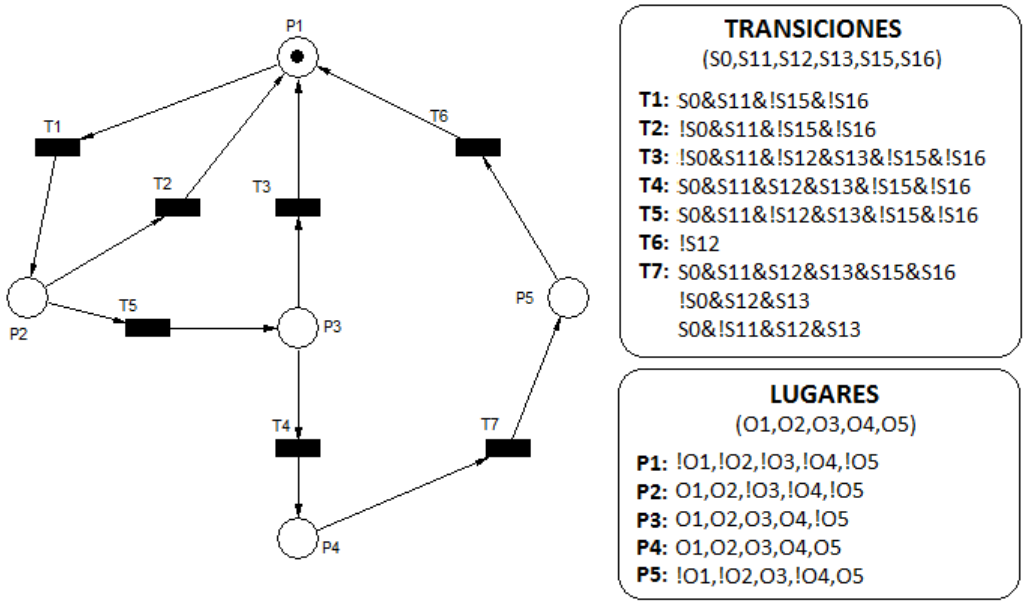

Figura 2.26: PN Atomizado y secado

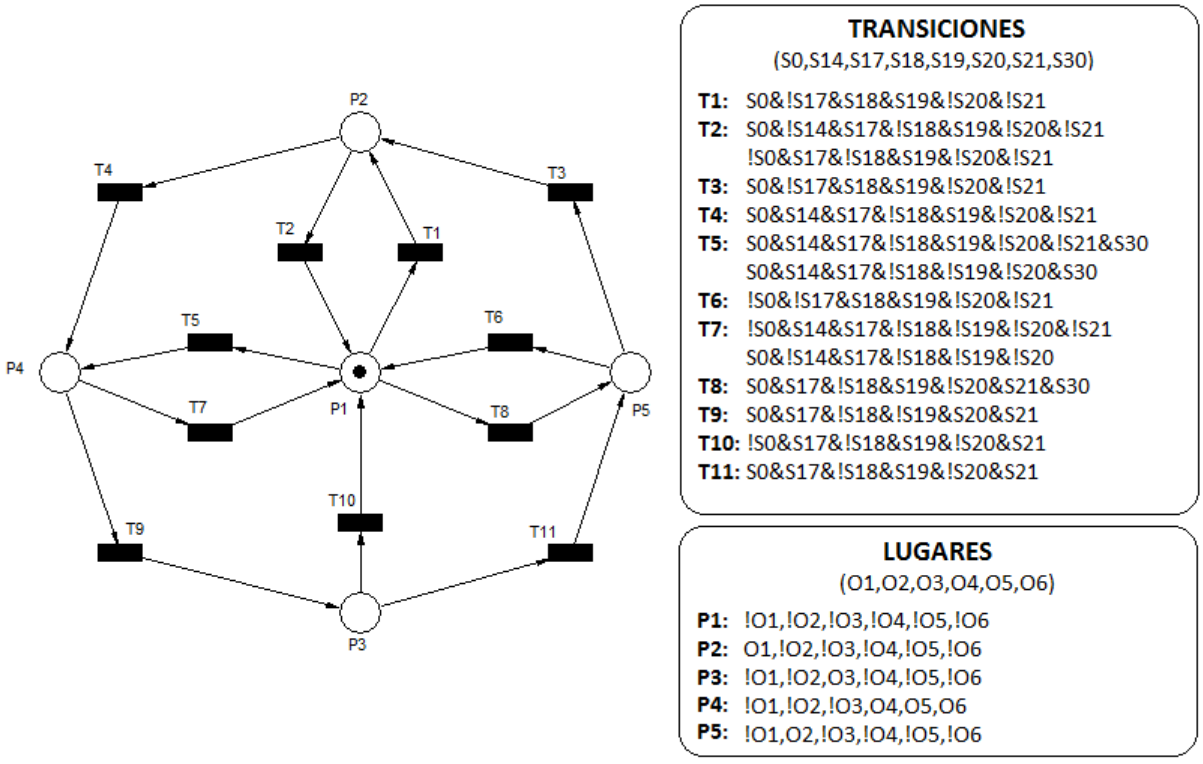

Figura 2.27: PN Prensado

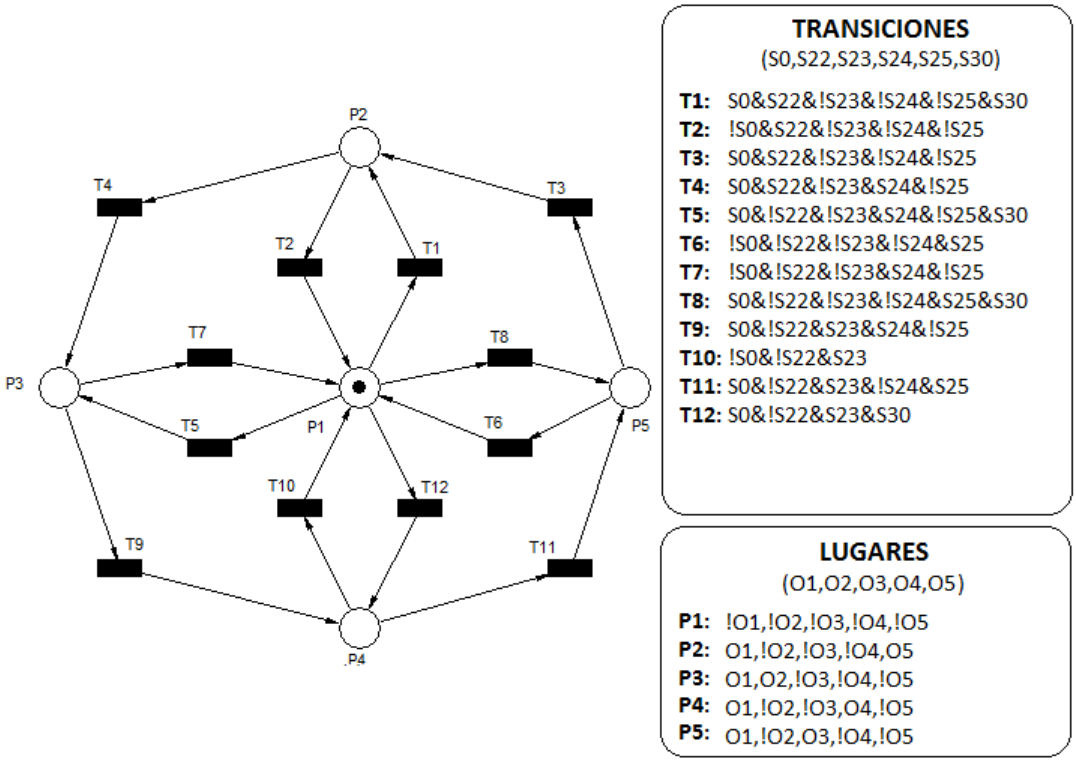

Figura 2.28: PN Girado

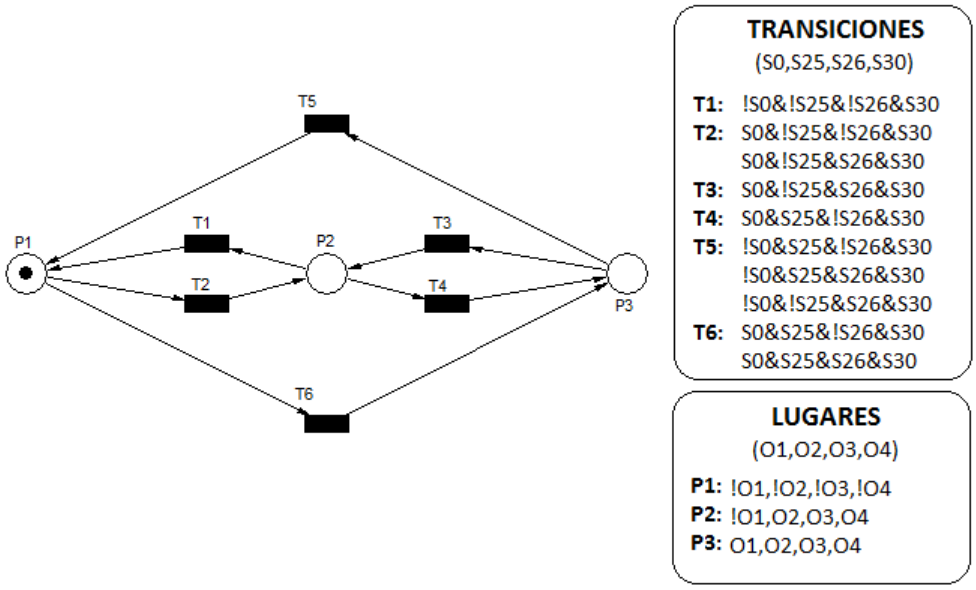

Figura 2.29: PN Secado por luz UV

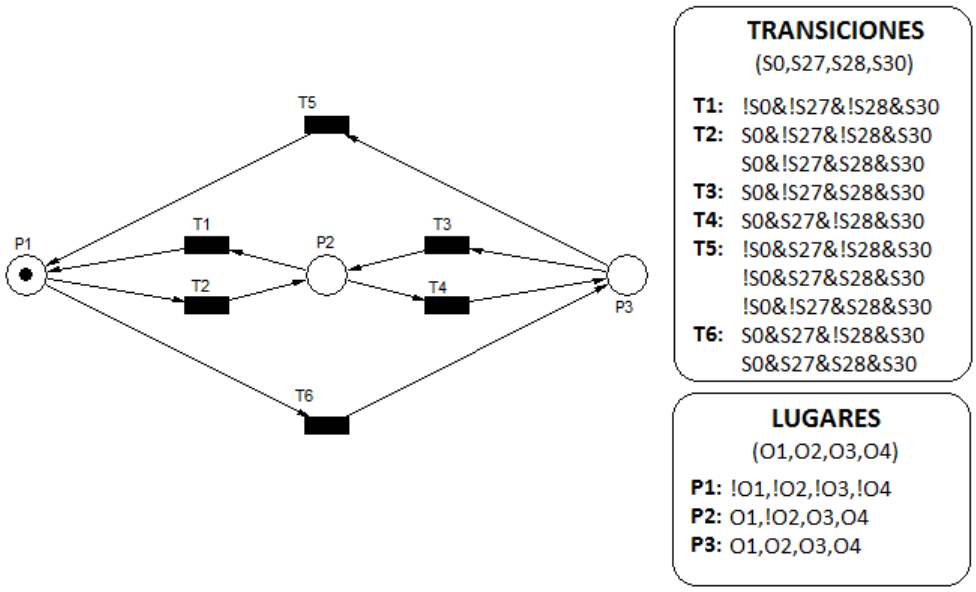

Figura 2.30: PN Impresión

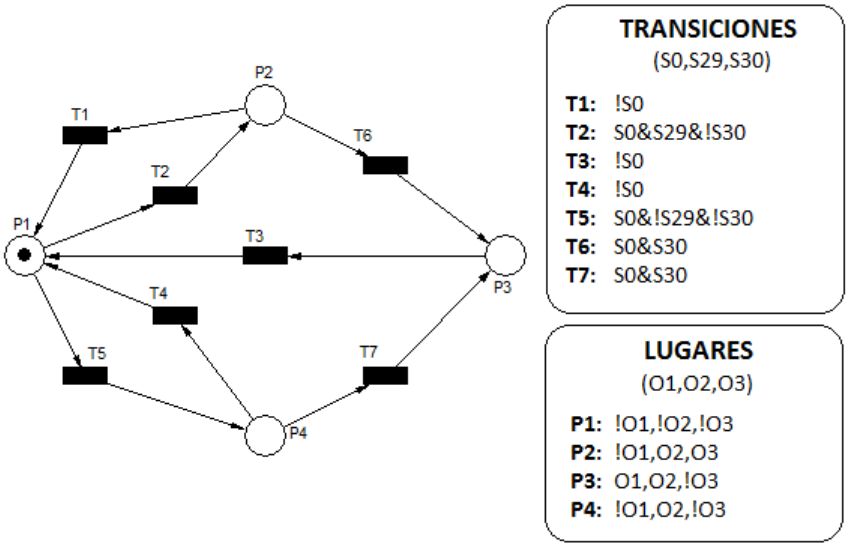

Figura 2.31: PN Horneado

En las PN obtenidas las cuales están basadas en los controladores, se aprecia que en su diseño no existen transiciones sumidero ya que para todos los lugares existentes siempre hay un ciclo interno que le permite llegar a cualquier lugar, para comprobar esto se hace una verificación de vivacidad y de ciclicidad, teniendo en cuenta las propiedades estáticas que se describen en [42] mediante el desarrollo del árbol de alcanzabilidad en cada una de las PN diseñadas:

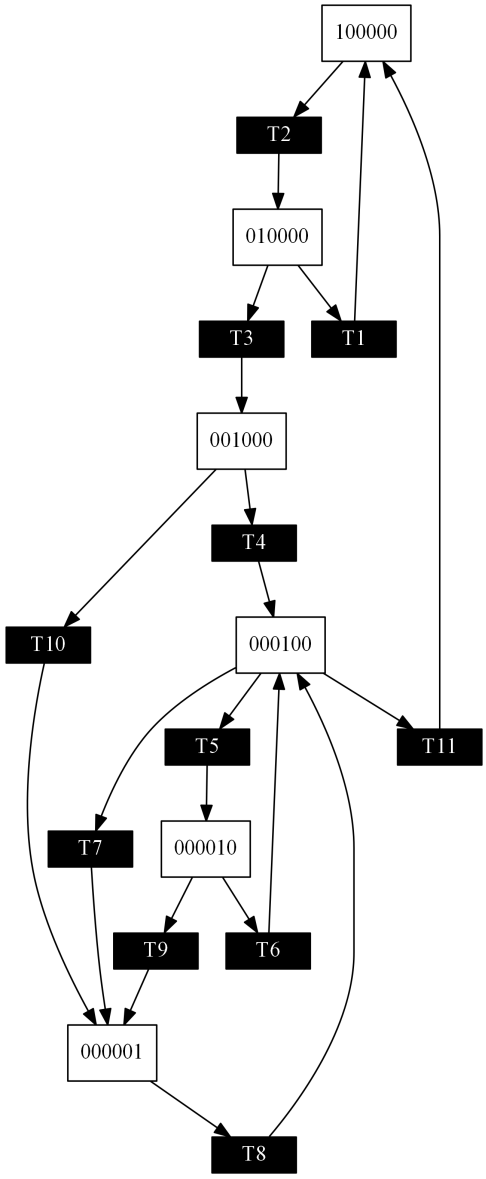

Figura 2.32: Árbol de alcanzabilidad Suministro de materia prima

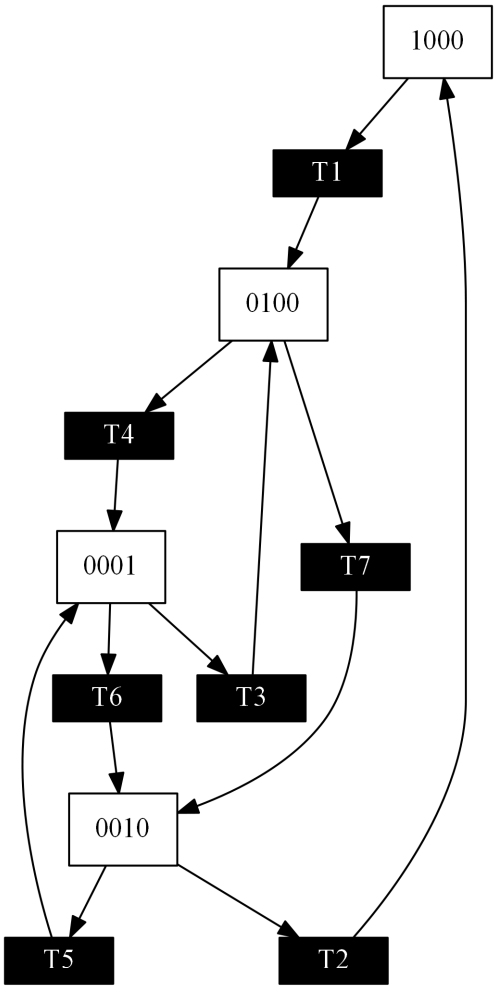

Figura 2.33: Árbol de alcanzabilidad Molienda de material

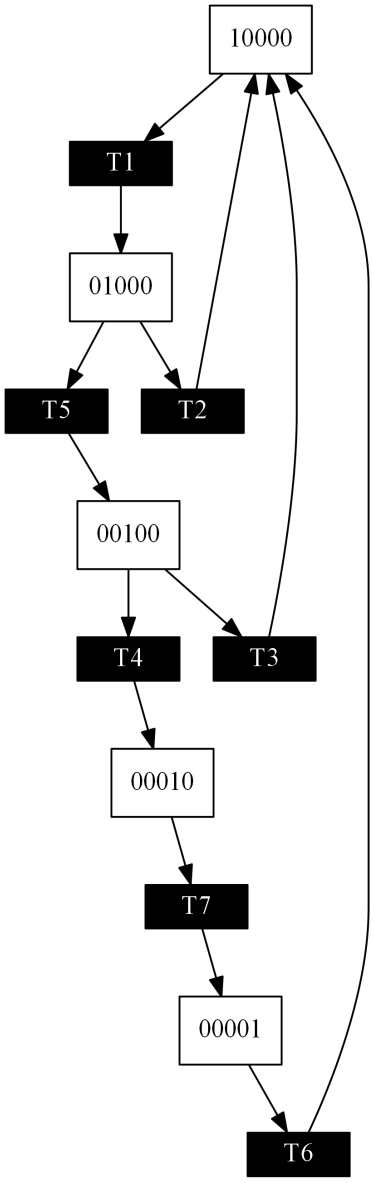

Figura 2.34: Árbol de alcanzabilidad Atomizado y secado

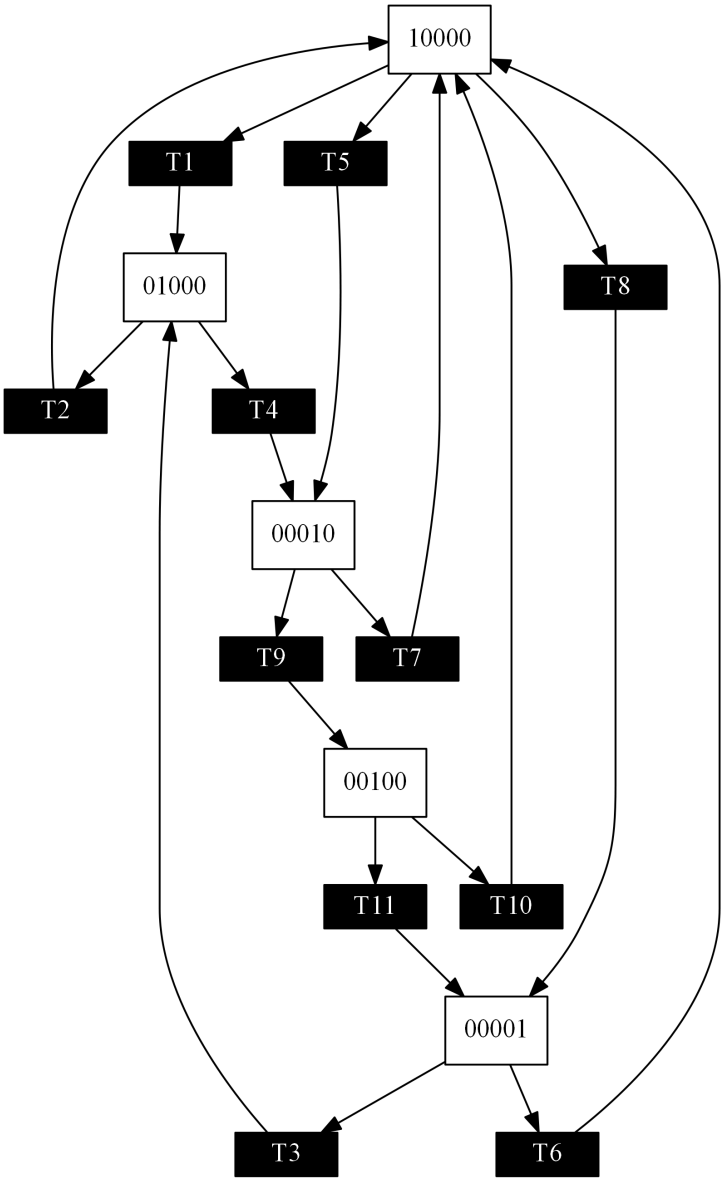

Figura 2.35: Árbol de alcanzabilidad Prensado

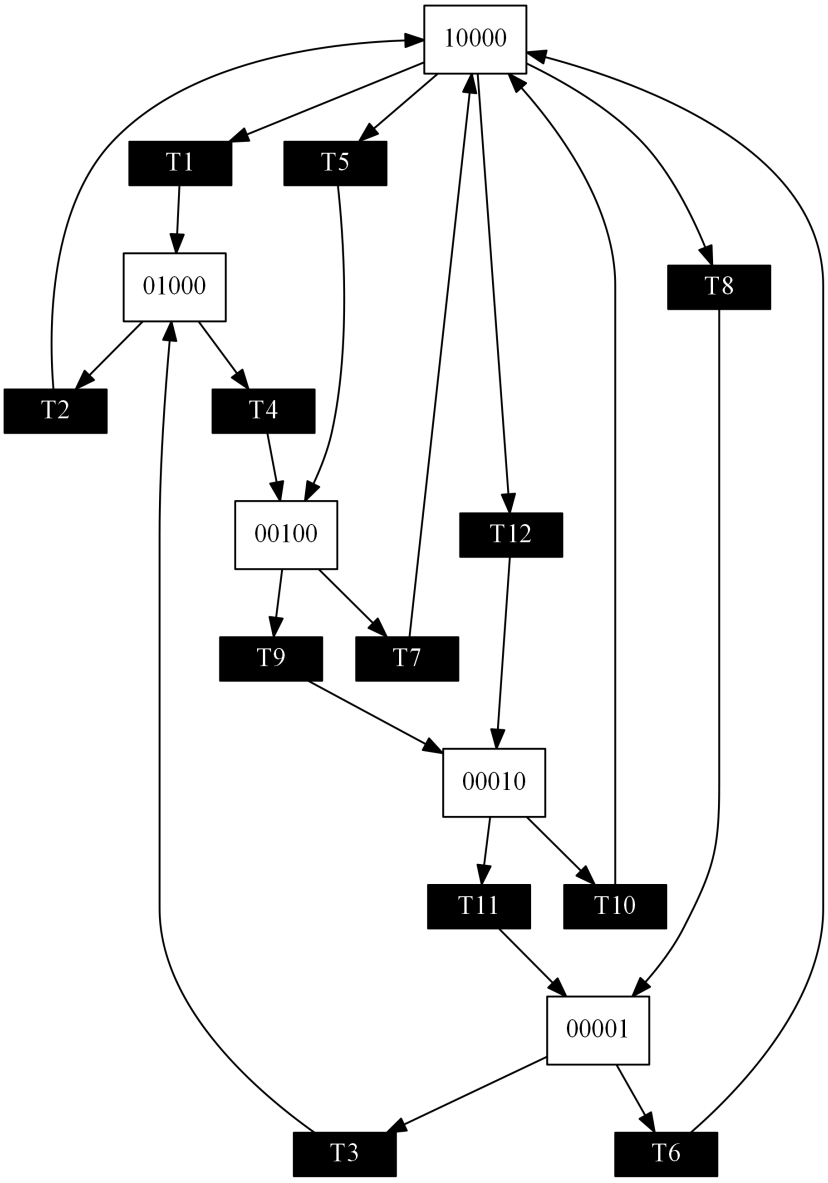

Figura 2.36: Árbol de alcanzabilidad Girado

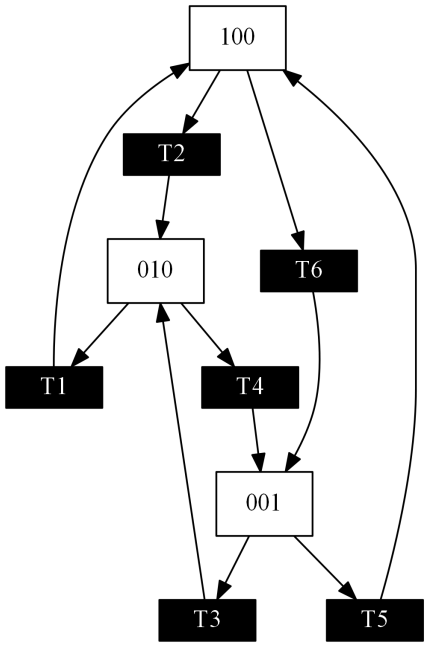

Figura 2.37: Árbol de alcanzabilidad Secado por luz UV

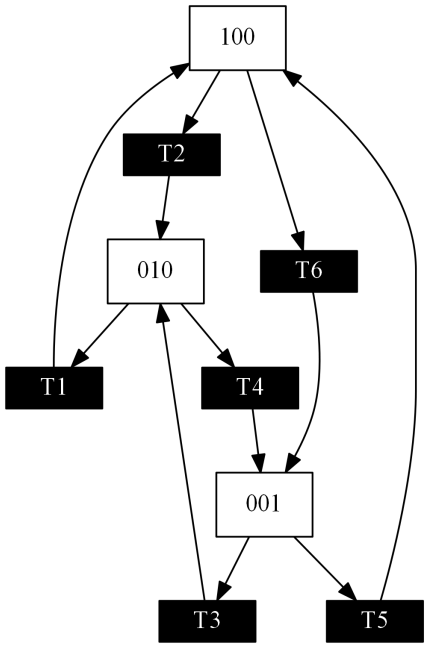

Figura 2.38: Árbol de alcanzabilidad Impresión

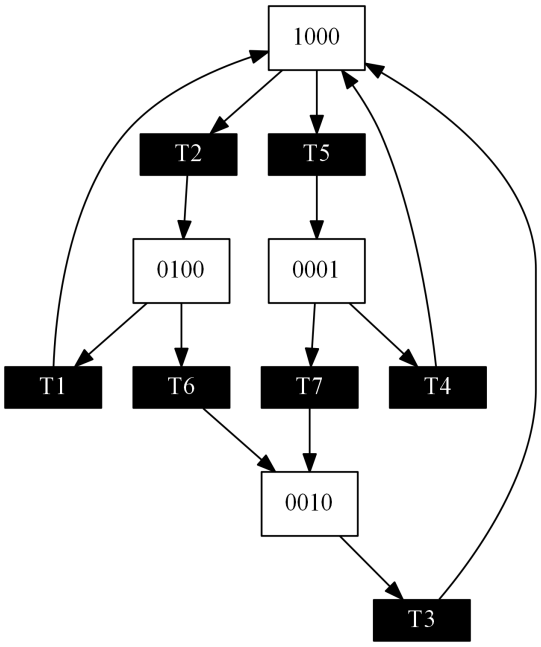

Figura 2.39: Árbol de alcanzabilidad Horneado

De los arboles de alcanzabilidad generados a partir de las PN de controlador se comprueba que las PN diseñadas son estructuralmente vivas, porque poseen un marcado inicial el cual corresponde al lugar P1 equivalente al primer lugar creado, también son controlables ya que desde el diseño siempre existen arcos que permiten el desplazamiento de una marca a la vez, también se demuestra que las PN son limitadas estructuralmente por que en ningún momento el número de marcas en los lugares es mayor a "1", debido a que en el diseño de los subsistemas se modelan las decisiones que los controladores deben dar en función de un vector de observaciones de entrada (sensores) por ello el peso de las transiciones son siempre "1".

Debido a que las transiciones poseen solo un único lugar de entrada y de salida, esto también implica que las PN son conservativas porque la suma de las marcas para cada lugar P siempre es "1", de igual manera son repetibles ya que existen secuencias sin transiciones sumidero, esto quiere decir que siempre regresan a un estado inicial.

3 IMPLEMENTACIÓN DE LOS MODELOS EN LA TARJETA DESARROLLO

A partir del modelado del sistema de eventos discretos diseñado con máquinas de estados, se generaron un conjunto de señales de E/S, las cuales tienen un comportamiento binario. Estas señales se representan como vectores de señales de salida para los estados y de vectores de señales de entrada para las transiciones en cada subsistema de manera independiente.

Para implementar los modelos en tarjetas de desarrollo se hace necesario establecer, diseñar y/o definir un procedimiento el cual sirva para traducir de manera adecuada y clara el comportamiento que la tarjeta debe realizar, acorde al funcionamiento de cada subsistema desde el punto de vista de la planta y del controlador.

3.1. Integración Planta - Controlador

El sistema de eventos discreto diseñado para obtener baldosas cerámicas según la Tabla 2.1, tiene 31 señales de sensores y 34 señales de actuadores, para un total de 65 señales, el gran número de señales requeridas obliga a buscar una tarjeta de desarrollo que posea abundantes pines de conexión donde cada conexión sea de tipo digital y no analógica.

Haciendo una búsqueda de tarjetas teniendo en cuenta características particulares como cantidad de pines disponibles, bajo costo, lenguaje de programación y velocidad del procesador, se encontraron dos tipos de tarjetas de desarrollo, una es la Arduino Mega que posee suficientes pines para cumplir con el requerimiento de señales, la otra opción es la Arduino Due que es una versión evolucionada y veloz con respecto a la versión Arduino Mega. Ésta, debido a sus características mejoradas, su costo es un poco más alto, las demás opciones fueron descartadas a causa del costo elevado o cantidad de pines insuficientes como la Beagleboard, Raspberrypi3, también las FPGA que si bien tiene suficientes pines su costo hace que sea descartada.

De las dos opciones que se obtuvieron, se decide primero determinar qué tan veloz va a ser la actualización de vectores entrada - salida en el sistema planta - controlador, con

base a esto se establece un tiempo de actualización de muestreo cada 200 ms para la planta, y a partir de este parámetro se decide por usar la tarjeta de desarrollo Arduino Mega ya que es la que menor costo tiene y cumple con los requerimientos establecidos para su selección.

La tarjeta de desarrollo Arduino Mega dispone de una capacidad de 72 señales digitales, de las cuales 70 son utilizables y 2 restantes están destinas para la comunicación serial con la computadora. La asignación de pines con las señales se realiza usando la Tabla 2.1, organizando los primeros pines con todas las señales de entradas y para las señales de salida los pines restantes y luego se genera un primer código de asignaciones de señales compatible con lenguaje C mostrado en la Figura 3.1 y Figura 3.2 el cual será utilizado para la simplificación de los códigos generados en el proceso de integración.

```
//Definicion variable global de inicio
#define S0 2        //Boton Encendido Global Proceso Controlado Por Puerto Serie
//mensaje de activar     I
//Mensaje de desactivar  D

//definicion de señales de entrada
#define S1  3       //Sensor Material Inferior Tolva
#define S2  4       //Sensor Material Medio    Tolva
#define S3  5       //Sensor Material Superior Tolva
#define S4  6       //Sensor Compuerta Cerrada
#define S5  7       //Sensor Compuerta Abierta
#define S6  8       //Sensor Material Banda
#define S7  9       //Sensor Nivel Alto  Tanque
#define S8  10      //Sensor Nivel Bajo  Tanque
#define S9  11      //Sensor Nivel Alto  Tanque Grande
#define S10 12      //Sensor Nivel Medio Tanque Grande
#define S11 13      //Sensor Nivel Bajo  Tanque Grande
#define S12 14      //Sensor Material Banda Atomizado
#define S13 15      //Sensor Set Point Alcanzado Temperatura Atomizado
#define S14 16      //Sensor Nivel Bajo  Tanque Polvillo
#define S15 17      //Sensor Nivel Medio Tanque Polvillo
#define S16 18      //Sensor Nivel Alto  Tanque Polvillo
#define S17 19      //Sensor Prensa Arriba
#define S18 20      //Sensor Prensa Abajo
#define S19 21      //Sensor Paleta Inicio
#define S20 22      //Sensor Paleta Fin
#define S21 23      //Sensor Objeto En la Prensa
#define S22 24      //Sensor Superior Mecanismo Girado
#define S23 25      //Sensor Inferior Mecanismo Girado
#define S24 26      //Sensor Interior Mecanismo Girado
#define S25 27      //Sensor Fuera    Mecanismo Girado
#define S26 28      //Sensor Pieza Ceramica Secado UV
#define S27 29      //Sensor Pieza Ceramica Antes  Impresora
#define S28 30      //Sensor Pieza Ceramica Dentro Impresora
#define S29 31      //Sensor Pieza Ceramica Interior Horno
#define S30 32      //Sensor Set Point Alcanzado Temperatura Horno
```

Figura 3.1: Asignación de entradas en C

```
//definicion de señales de salida

#define Banda1        33
#define Compuerta_Izq 34
#define Compuerta_Der 35
#define Luz_Material  36

#define Molino        37
#define Bomba1        38
#define Valvula1      39

#define Quemador1     40
#define Ventilador    41
#define Banda2        42
#define Bomba2        43
#define Banda3        44

#define Prensa_Arriba 45
#define Prensa_Abajo  46
#define paleta_Inicio 47
#define Paleta_Fin    48
#define Valvula2      49
#define Aspiradora    50

#define Banda5        51
#define Gira_Izq      52
#define Gira_Der      53
#define Banda4_Iqz    A0
#define Banda4_Der    A1


#define Lampara       A2
#define Banda6        A3
#define Bomba3        A4
#define Valvula3      A5

#define Banda7        A6
#define Impresora     A7
#define Valvula4      A8
#define Bomba4        A9

#define Banda8        A10
#define Quemador2     A11
#define Quemador3     A12
```

Figura 3.2: Asignación de salidas en C

La propuesta de modelado del sistema escogido, presentada en el capítulo 2 Sección 2.1 se basa en la división en subsistemas y del comportamiento de cada uno de ellos, donde se representa como máquinas de estados independientes que representan las dinámicas conjuntas entre el comportamiento de la planta y del controlador.

El procedimiento de integración diseñado surge del proceso iterativo de ensayo y error para integrar de manera directa los modelos de máquinas de estados en la tarjeta de desarrollo, quiere decir que cada una de las máquinas de estados representadas en lenguaje C fueron agrupadas incluyendo la planta y el controlador, este primer intento consistía en diseñar a partir de una estructura condicional SWITCH el comportamiento del controlador y a partir de este, adicionar en donde sea necesario el comportamiento

dinámico usando contadores de tiempo basados en la función «millis» que dispone la tarjeta de desarrollo Arduino Mega. Esto se usó para determinar cuanto tiempo se debe esperar la tarjeta para causar la evolución de las señales E/S.

Eventualmente esta forma de ingresar las máquinas de estados, presentaba una serie de inconvenientes al momento de establecer el comportamiento dinámico, ya que implicaba replicar muchas veces un mismo segmento de código dinámico en muchas partes del mismo, generando un incremento enorme en el código resultante y dificultando su entendimiento. Con base a este primer intento se concluyó que el comportamiento dinámico y el del controlador son parecidos pero no deben ir unidos ya que aumenta en gran medida el código resultante.

Se realizó nuevamente la integración, esta vez teniendo en cuenta que el comportamiento dinámico de planta es similar al comportamiento realizado por el controlador ya que se basa en la estructura condicional SWITCH, debía ir aparte con el fin de evitar el incremento de código repetido. La integración de la planta - controlador, se pudo lograr sin errores durante la verificación usando el IDE de desarrollo Arduino, aunque aparentemente el IDE de desarrollo indicaba no presentar errores se procedió a realizar la verificación de cada subsistema contrastando los vectores obtenidos con los vectores de E/S que se definieron en la máquina de estados, logrando comprobar que en cada etapa por separado el comportamiento de la máquina de estado correspondía con el código diseñado, así que se procede luego a analizar el comportamiento general a partir de una serie de muestras obtenidas durante un intervalo de tiempo alrededor de una hora.

Al realizar el análisis en el sistema general, se buscó verificar el correcto comportamiento de la interacción entre los subsistemas (comportamiento de elementos intermedios, tanques y bandas transportadoras), se encontró que no era adecuado el comportamiento dinámico usando temporizadores, ya que de esta manera no existía la sincronización de los subsistemas, generando un inicio de cada subsistema al mismo tiempo.

Luego de realizar el cambio de temporizadores a contadores de unidades volumétricas en los tanques y el uso de cadenas de bits en las bandas transportadoras representando el desplazamiento de piezas, se obtuvo la simplificación del código del controlador pero un incremento estructural en el código de la planta.

Durante la verificación en el IDE de desarrollo se comprobó que no existían errores en el código general, luego se realizó la verificación de los vectores de E/S, donde se logró comprobar efectivamente el correcto funcionamiento de la señales con respecto a los modelos de máquinas de estado y al proceso escogido. Se concluyó que el código obtenido correspondía positivamente al comportamiento esperado, pero el segmento para representar la planta no era muy claro a causa del incremento de líneas, sino también a pequeños segmentos repetidos correspondientes al comportamiento de cada elemento intermedio entre los subsistemas (Tanques y bandas transportadoras); por ello se realizó una modificación en el proceso de integración en la tarjeta de desarrollo.

En el intento de mejora se realizó la separación del comportamiento de planta en dos partes, una primera parte especifica cada elemento que compone un subsistema, donde se almacenan el estado actual de dicho elemento ya sea nivel de llenado, porcentaje de apertura de una compuerta, piezas en un determinado lugar en las bandas transportadoras entre otros, y una segunda parte basada en la evolución de la planta usando un único temporizador para cada subsistema, donde solo actualiza las variables contador de cada uno de los elementos de la planta, y de desplazar o rotar bits de bandas transportadoras para representar el movimiento en las mismas.

Para los segmentos de código únicos asignados a cada elemento se adicionan una serie de condiciones IF en las que se compara llenado para los tanques con un valor o intervalo de llenado, también se compara la presencia de un "1" en un punto específico de las cadenas de bits que representan las bandas transportadoras indicando la presencia de una pieza en un punto de la banda. Todo esto con el objetivo de disparar o causar la evolución de las señales correspondientes a los sensores incluidos en cada elemento de los subsistemas, dividiendo la integración en las siguiente partes vistas en :

Segmentos

Segmento planta: Esta parte de segmento se encarga de actualizar los estados actuales de los elementos de cada subsistema a un intervalo de tiempo dado, la evolución de la estados actuales dependen del estado "0" o "1" en que las señales de las salidas se encuentren.

Segmento características de elementos: Comprende las características propias de un elemento, ya sea nivel, temperatura, presencia de un objeto en un punto dado entre otros. También se encarga de generar la evolución de la señales de entrada (sensores) asociadas a cada elemento.

Segmento controlador: Se encarga de representar el comportamiento lógico y de control diseñado en las máquinas de estado, también de generar la evolución de la señales de salida en cada subsistema.

3.1.1. Procedimiento Controlador

Para el desarrollo de un segmento de código del controlador, se parte del modelo de máquina de estados, en el cual cada transición corresponde unicamente a un vector de señales de entrada, sin contemplar aquellos vectores de entrada generados durante el cambio de una transición a otra causado por la señal de salida generada del estado actual, un ejemplo de esto es cuando se tiene una compuerta y se tiene dos sensores en sus extremos y el vector de entrada ignorado es el obtenido cuando la compuerta se encuentra a mitad de camino, ya sea para abrirse o cerrarse.

El procedimiento realizado se puede explicar con base al ejemplo que se muestra en la Figura 3.3 el cual tiene dos señales de entrada, dos señales de salida y tres estados. A

cada estado se le ha asignado un vector de señales de salida único, cada estado tiene una transición de salida correspondiente al vector de señales de entrada que dispara el cambio de estado.

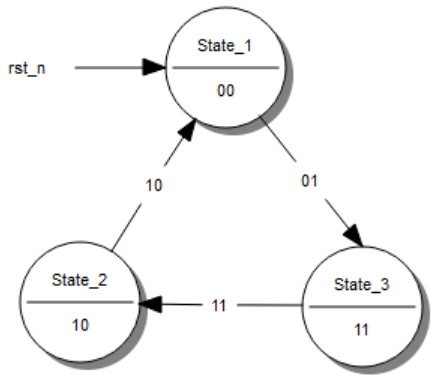

Figura 3.3: Máquina de Estados de Ejemplo

Para representar el anterior modelo en lenguaje de programación C, primero se procede a generar una estructura condicional SWITCH dependiente de una variable que indica el estado actual en que el sistema se encuentra, esta estructura tiene como único fin separar “n” segmentos de código como “n” estados existen.

Dentro de cada segmento de código se realiza una comprobación IF la cual verifica un vector de señales de entrada con el vector asociado a las transiciones de salida que tiene cada estado, cuando se verifica la función IF el vector de señales de salida es actualizado, luego la variable de la cual depende la estructura SWITCH es actualizada con el estado siguiente de la transición que se disparó, ver Figura 3.4

```
void Controlador()
{
  switch (Estado)
  {
    case 1:
      if (Entradas(0, 1))
      {
        Salidas(1, 1);
        Estado = 3;
      }
      break;
    case 2:
      if (Entradas(1, 0))
      {
        Salidas(0, 0);
        Estado = 1;
      }
      break;
    case 3:
      if (Entradas(1, 1))
      {
        Salidas(1, 0);
        Estado = 2;
      }
      break;
  }
}
```

Figura 3.4: Estructura del código de Controlador

Como se puede apreciar en el ejemplo de código, para el estado "1" existe una transición de salida, la cual tiene como vector de señales de entrada [0, 1] dentro del segmento de la condición WITCH deben haber tantos IF como transiciones de salida existen, donde la función para este caso en particular llamada Entradas(int, int), solo retorna verdadero o falso cuando compara el vector que se le da como parámetro y en caso de que esto se cumpla la función Salidas(int, int) solo actualiza las señales de salidas con el vector dado como parámetro para luego actualizar la variable Estado siguiente que apunta dicha transición.

3.1.2. Procedimiento Características de Planta

Para el desarrollo de los segmentos de código que definen las características de los elementos de la planta, se establecen archivos separados para cada uno de los tipos de elementos o partes que comprenden un proceso, por ejemplo, para la etapa de prensado diseñada anteriormente existen dos archivos uno para las características del mecanismo de paleta el cual retira la pieza cerámica en el interior de la prensa y otro para el mecanismo de pistón de la prensa que sube y baja.

Las características asociadas a un único elemento pueden variar según la dinámica que se quiere realizar, ya sea para simular el llenado de un tanque o también como el

desplazamiento de una pieza de un punto “A” a un punto “B”. Por ello se establecen las estructuras básicas de código para algunos tipos de elementos usados en el caso de estudio, contemplando las condiciones de sensores asociados a dichos elementos.

3.1.2.1. Elementos de Tipo Banda Transportadora

Este tipo de elementos se caracterizan por estar formados de una cadena de bits o variables que rotan de igual manera que lo realiza una banda transportadora real, cuando existe presencia de un objeto en la banda transportadora, los lugares bits o espacios de memoria que representan la posición donde se encuentra el objeto, se mantienen en estado alto o “1”, en la Figura 3.5 se puede apreciar la estructura de código base para este tipo de elementos, donde existe una función de condiciones en la que se analiza la existencia de un “1” en el último bit o espacio de memoria indicando la presencia de un objeto en la banda transportadora y generando un cambio en el sensor1 asociado al elemento banda transportadora. Se deben añadir tantas condiciones IF - ELSE por cada sensor asociado.

```
//Banda

#define Bits_Banda 30           // indica el Total de posiciones en la Banda

//Cadena_... Usada para representar el desplazamiento de objetos en la Banda
int Cadena_Banda[Bits_Banda] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

void Suministro_Banda()
{
  if (Cadena_Banda[Bits_Banda - 1])
  {
    digitalWrite(sensor1, HIGH);
  }
  else
  {
    digitalWrite(sensor1, LOW );
  }
  ////////////////////////////////////////////////////////////////////////////////////////////
}
```

Figura 3.5: Estructura del código para elementos tipo Banda transportadora

3.1.2.2. Elementos de Temperatura y Movimiento Mecánico

Este tipo de elementos está formado por una variable contador, la cual suele indicar la posición del elemento mecánico en términos de porcentaje en un momento dado o el valor de temperatura en grados centígrados, también se compone de los parámetros máximo y mínimo en el que la variable puede cambiar, incluyendo parámetros como la velocidad de incremento o decremento de la variable contador, y valores en los que

una señal de sensor cambia de estado (valores máximos, mínimo e intermedios). En la Figura 3.6 se puede apreciar la estructura base de este tipo de elementos.

```
//Compuerta

uint8_t Apertura_Compuerta = 0; //Indica el nivel de Apertura en Porcentaje

#define Compuerta_Abierta 100   //100% de Apertura
#define Compuerta_Medio    40   //  X% de Apertura
#define Compuerta_Cerrada   0   //  0% de Apertura

#define Apertura_Compuerta_Razon 5      //  Velocidad de Apertura [ X% / Intervalo de tiempo]

void Compuerta ()
{
  if (Apertura_Compuerta >= Compuerta_Abierta)
  {
    digitalWrite(sensor1, HIGH);
    digitalWrite(sensor2, LOW );
    Apertura_Compuerta = Compuerta_Abierta;
  }
  else if (Apertura_Compuerta > Compuerta_Cerrada)
  {
    digitalWrite(sensor1, LOW );
    digitalWrite(sensor2, LOW );
  }
  else
  {
    digitalWrite(sensor1, LOW );
    digitalWrite(sensor2, HIGH);
    Apertura_Compuerta = Compuerta_Cerrada;
  }
  ////////////////////////////////////////////////////////////////////////////////////////////////////
}
```

Figura 3.6: Estructura del código Para Elementos de Temperatura o Movimiento Mecánico

3.1.2.3. Elementos de Almacenamiento

Este tipo de elementos es muy similar a los de tipo mecánico; ya que, también se basan en una variable contador, sus parámetros hacen referencia a los niveles de llenado a partir de unidades volumétricas que el tanque o elemento de almacenamiento pueda tener, también posee parámetros de velocidad de llenado en términos de unidades volumétricas en un intervalo de tiempo, diferenciándose en las condiciones que generan cambio en los sensores, debido a que ya no dependen de intervalos entre parámetros, sino tienen un comportamiento on/off lo que quiere decir que se obtiene un “1” de un sensor cuando el nivel de llenado supera el valor establecido en el parámetro asociado a ese sensor. La estructura de código se puede apreciar en la Figura 3.7.

```
//Tanque

int16_t LLenado_Tanque = 0;     //Indica el nivel de LLenado en Unidades Volumetricas

#define LLenado_Tanque_Alto  300    //  300 Unidades Volumetricas
#define LLenado_Tanque_Medio 100    //  100 Unidades Volumetricas
#define LLenado_Tanque_Bajo    0    //  0   Unidades Volumetricas

#define LLenado_Tanque_Razon  4 //  Velocidad de LLenado [Unidades Volumetricas / Delta de Tiempo]
#define Vaciado_Tanque_Razon  2 //  Velocidad de Vaciado [Unidades Volumetricas / Delta de Tiempo]


void Tanque ()
{
  if (LLenado_Tanque >= LLenado_Tanque_Alto)
  {
    digitalWrite(sensor3, HIGH);
    //LLenado_Tanque = LLenado_Tanque_Alto;
  }
  else
  {
    digitalWrite(sensor3, LOW );
  }
  if (LLenado_Tanque >= LLenado_Tanque_Medio)
  {
    digitalWrite(sensor2, HIGH);
  }
  else
  {
    digitalWrite(sensor2, LOW );
  }
  if (LLenado_Tanque >  LLenado_Tanque_Bajo)
  {
    digitalWrite(sensor1, HIGH);
  }
  else
  {
    digitalWrite(sensor1, LOW );
    LLenado_Tanque = LLenado_Tanque_Bajo;
  }
  ////////////////////////////////////////////////////////////////////////////////////////////
}
```

Figura 3.7: Estructura del código para Elementos de Almacenamiento

3.1.3. Procedimiento de Comportamiento de Planta

El procedimiento a realizar para establecer el comportamiento de la planta se define en estructuras de tiempo - contador, en las cuales se establece un periodo de tiempo que hace las veces de periodo de muestreo en una señal analógica, el cual es el encargado de actualizar las variables de estado de cada elemento del sistema en periodo de tiempo. El propósito de establecer estos tiempos, es simular el comportamiento dinámico de los componentes de la planta de manera discreta desde la simulación interna en la tarjeta de desarrollo, como también el realizar la integración entre elementos de las etapas de proceso.

Para la realización del conteo de tiempo se aprovecha que la tarjeta de desarrollo Arduino Mega dispone de una función «Millis» que retorna el valor del total de milisegundos transcurridos desde que la tarjeta de desarrollo fue encendida. Esta función es utilizada inicialmente para tomar un valor inicial de tiempo y seguidamente verificar de manera continua y repetitiva el tiempo transcurrido, así se logra simular el comportamiento de las señales analógicas de salida de un elemento o dispositivo dado

en intervalos de tiempo. Además al hacer uso de esta función, se logra ejecutar más de un proceso a la vez dentro de una misma tarjeta de desarrollo, con el fin de replicar el comportamiento que tiene un PLC real siendo emulado en la tarjeta Arduino Mega.

La estructura de código para la simulación del comportamiento de planta se puede apreciar en la Figura 3.8.

```
//Planta
boolean Activado_Planta = true;
#define Tiempo_Planta 200
uint32_t tiempo_Actual_Planta;
int64_t tiempo_Total_Planta;

void Planta ()
{
  if (Activado_Planta)
  {
    tiempo_Actual_Planta = millis();
    Activado_Planta = false;
  }
  tiempo_Total_Planta = millis() - tiempo_Actual_Planta;
  if (tiempo_Total_Planta >= Tiempo_Planta)
  {
    ////////////////////////////////////////////////////////////////////////////

    // Condiciones de cambio para variables asociadas a los elementos de una etapa

    ////////////////////////////////////////////////////////////////////////////
    tiempo_Actual_Planta = millis();
  }
}
```

Figura 3.8: Estructura del código Para Tiempo de Actualización de una Etapa

En la estructura mostrada en la Figura 3.8 se deben añadir “n” condiciones de cambio de variables como “m” elementos existan el la planta diseñada. La condiciones de cambio para variables asociadas a los elementos de una etapa, se caracterizan por ser disparadas o controladas por el estado de la señal asociada a cada salida, por ejemplo en la Figura 3.9, la señal asociada a la banda transportadora_1 se encuentra en estado alto o “1”, esto quiere decir que cada vez que la condición de tiempo se ejecute, se evalúa la condición de estado de la señal de salida correspondiente a la banda, donde esta señal al estar en estado alto realiza un desplazamiento de bits una sola vez, simulando el desplazamiento de una pieza en la banda en el intervalo de tiempo establecido generando de esta manera la evolución del estado actual del elemento banda transportadora (actualización de la cadena de bits), de igual manera ocurren con cada elemento usado en la planta (tipo temperatura, tanques, elementos mecánicos), realizando la actualización del estado actual mediante los parámetros de incremento y decremento establecidos para cada uno.

```
////////////////////////////////////////////////////////////////////////////////////////////////////
if (digitalRead(Banda5))
{
  if (Cadena_Gira_Banda[Bits_Gira_Banda - 1])
  {
    Cadena_Gira_Banda2[0] = 1;
  }
  for (int i = 1; i <= Bits_Gira_Banda; i++)
  {
    Cadena_Gira_Banda[Bits_Gira_Banda - i] = Cadena_Gira_Banda[Bits_Gira_Banda - 1 - i];
  }
  Cadena_Gira_Banda[0] = 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

Figura 3.9: Ejemplo Estructura de Condición de Cambio Banda Transportadora

3.1.4. Procedimiento Integración Planta Controlador

El procedimiento de integración de planta - controlador consiste en el uso de un ciclo infinito, en el cual primero se actualizan las variables asociadas a cada elemento de la planta, luego se actualiza el comportamiento de la planta en un intervalo de tiempo, por último se comprueban las señales de entrada mediante la estructura de controlador, todo esto haciendo uso de las funciones generadas en cada segmento de código en los subsistemas, de esta manera se puede apreciar el comportamiento planta - controlador que se diseñó observando claramente el comportamiento en lazo cerrado de la planta, ver Figura 3.10.

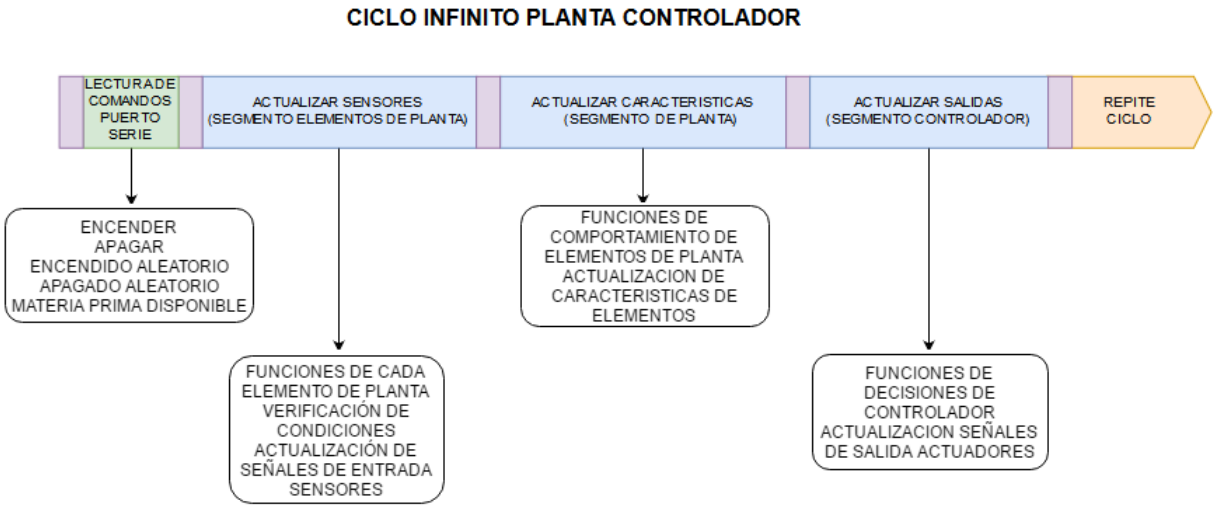

Figura 3.10: Ciclo infinito planta controlador

Nótese que existe una tarea al principio del ciclo, esta tarea de lectura de comandos por puerto serie, corresponde a la recepción de un comando a la vez para disparar eventos externos, como lo son el encendido, apagado, reinicio manual incluyendo otros eventos externos no observables como el de materia prima agotada o el evento del operario no realiza su función de suministrar materia prima.

Esto implica que para obtener el total de combinaciones de señales durante un muestreo, es necesario añadir funciones que manipulen estas señales o eventos manualmente, para asegurar que en algún momento estas señales puedan cambiar y obtener el total de combinaciones que posea el sistema de eventos discretos escogido.

Aquellos eventos de encendido y apagado aleatorio se basan en el uso de una variable que mediante la función «RAMDOM» que dispone la tarjeta de desarrollo Arduino Mega se obtiene un número entre 0 y 100, este número es comparado con un valor de referencia también entre 0 y 100 el cual representa un intervalo en el que la señal de encendido se debe mantener en estado alto o "1", y de no ser así la señal se mantiene en estado bajo o "0" la ejecución automática del encendido y apagado aleatorio se realiza cada cierto lapso de tiempo, con esto lo que se busca es obtener el comportamiento de la planta en cada uno de los subsistema en cada uno de sus estados.

Dentro de la tarjeta de desarrollo Arduino Mega únicamente se tendría como resultado el total de señales que ésta genera, siendo emuladas en sus puertos digitales permitiendo así censar mediante otra tarjeta Arduino Mega que tiene como función capturar o adquirir estas señales, a modo que para el caso de estudio se podría ver este sistema de eventos discretos como un sistema de caja negra y posteriormente aplicar y obtener el Autómata y la PN basándose en los algoritmos suministrados en [1, 30] .

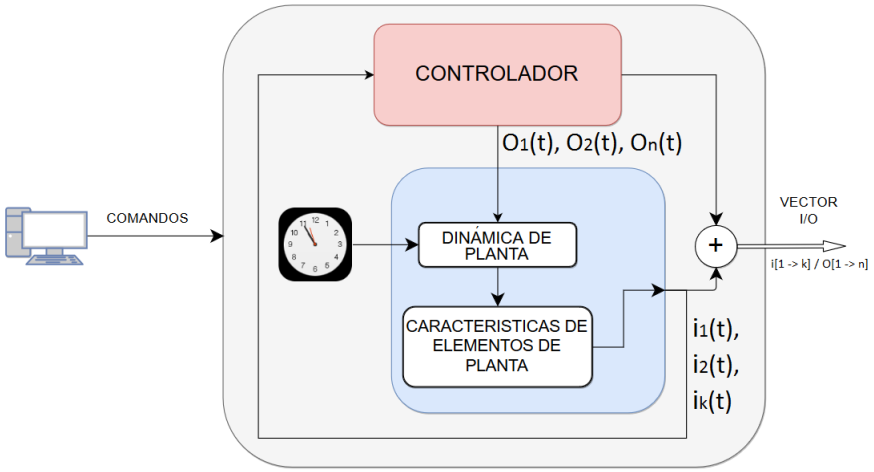

Figura 3.11: Diagrama Bloques de Sistema de Eventos Discretos Diseñado

Los códigos generados a partir del los procedimientos planteados en la integración de

planta controlador resultan ser demasiado numerosos generando un gran número de ficheros independientes. Los códigos obtenidos se han colocado como un anexo bajo el nombre «anexo A».

3.2. Implementación de adquisición de señales

La adquisición de señales se diseña de igual forma sobre una tarjeta de desarrollo Arduino Mega, ya que la estructura de pines es idéntica a la tarjeta del sistema de eventos discretos, donde también se ejecuta un ciclo infinito, en el cual primero se lee un comando de inicio o parado para la captura de datos mediante el puerto serie.

Esta parte de desarrollo del código funciona de manera continua y sin retardo con el fin de capturar cambios de señales que puedan ocurrir muy rápidamente, para lograr esto se realiza la lectura de ocho bits de cada uno de los puertos mostrados en la Figura 3.12 que el procesador Atmega 2560 posee, de esta manera se logra agilizar y aprovechar de mejor forma los ciclos de procesador evitando seguir la forma tradicional de lectura pin a pin y de mantener una frecuencia de lectura uniforme en la adquisición de señales.

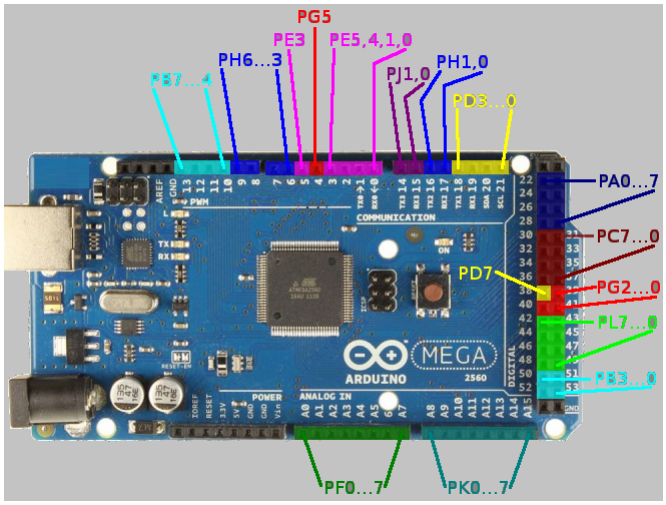

Figura 3.12: Puertos de Arduino Mega

3.2.1. Comunicación Con Dispositivo de Desarrollo

Como una tarjeta de adquisición de señales por si sola no es muy útil, se necesita de comunicación continua con una parte software que sea la encargada de adquirir los datos de la tarjeta y procesarlos para obtenerlos de manera ordenada y realizar un posterior procesamiento.

La tarjeta Arduino Mega posee un puerto serial, el cual funciona a través de un pequeño integrado que hace de puente de comunicación entre una computadora y el micro-controlador Atmega 2560 por medio de un puerto USB, para la comunicación entre el software de adquisición de señales se estableció que los datos serían enviados mediante caracteres ASCII para hacer visible los mensajes que la tarjeta envía usando cualquier monitor serial, adicionalmente se establece una velocidad de comunicación del puerto serie en 230.400 baudios para que el tiempo en que tarda en enviarse todos los datos de la lectura no tengan efecto aparente sobre el muestreo y lograr así una frecuencia de muestreo uniforme que depende de cuanto tiempo demora el envío de una trama de datos con duración de 2.5 mili-segundos cada una a intervalos de 10.2 mili-segundos ver Figura 3.13.

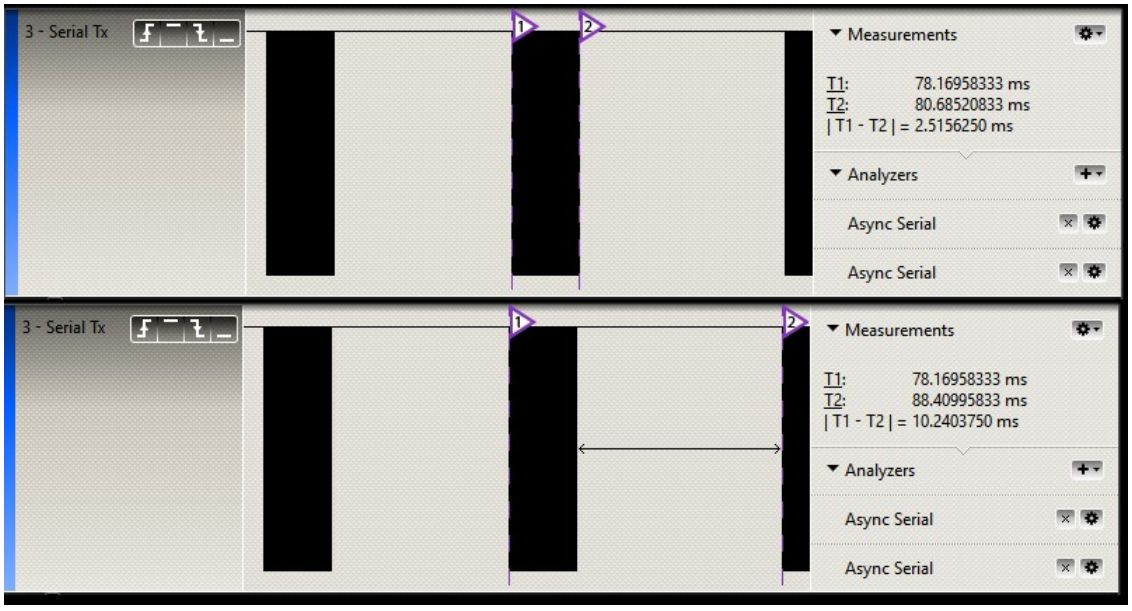

Figura 3.13: Tiempos de adquisición de señales

Como se busca es tener una frecuencia de muestreo uniforme, la lectura de los pines de la tarjeta se realiza mediante palabras de 8 bits. Se presenta un problema durante el envío de datos debido a que al enviar un dato obtenido de uno de los puertos de la tarjeta, ésta enviaría un número comprendido entre el rango de 0 y 255, esta variación

de 1 a 3 caracteres hace que la trama de caracteres enviada durante un muestreo sea no uniforme.

Para ello se establece que la tarjeta convierta el valor de 8 bits en dos caracteres hexadecimal, los cuales son enviados, primero un carácter indicando el puerto al que corresponde el dato, seguido de unos paréntesis que contienen el valor hexadecimal del puerto correspondiente, de igual forma son enviados los demás puertos que la tarjeta Arduino Mega tiene, finalizando con un carácter "W" ver Figura 3.14.

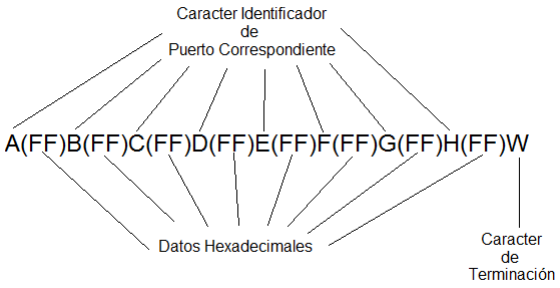

Figura 3.14: Trama de Datos Comunicación Serial

Nótese que ahora los datos enviados al estar en forma hexadecimal el tamaño de la trama de datos es siempre uniforme, corrigiendo el problema de tamaño variable para así obtener tiempos uniformes de muestreo.

Al igual que la tarjeta del sistema de eventos discretos, esta tarjeta debe tener instrucciones de control que son enviadas a través del puerto serial, éstas con el propósito de iniciar y detener el envió de datos adquiridos que la tarjeta hace de forma continua.

4 APLICACIÓN SOFTWARE

El desarrollo de la aplicación software, surge de la necesidad de tener una herramienta de trabajo para adquirir datos e identificar los modelos de comportamiento a partir de los métodos de identificación seleccionados y presentados en la sección Subsección 1.2.3.

El desarrollo de la aplicación software requiere escoger un entorno que permita crear la interfaz, agregar las funcionalidades necesarias y compilar la aplicación para que sea ejecutada sin necesidad de instalar paquetes software adicionales en la computadora.

El funcionamiento esperado de la aplicación consiste en que mediante la programación de los algoritmos elegidos y a partir de las señales del sistema emulado adquiridas por la misma aplicación se puedan identificar los modelos en código y también de manera gráfica.

La identificación se realiza mediante el análisis de un conjunto de muestras de señales adquiridas haciendo uso de la comunicación entre la aplicación a desarrollar y la tarjeta de adquisición establecida en Sección 3.2, lo que se obtiene en esta parte es un archivo que almacena todas las tramas de datos Figura 3.14; las cuales mediante el aislamiento o selección de señales se generan los vectores de unos y ceros del sistema o subsistema a identificar.

En este capítulo, no se explica el funcionamiento de la herramienta compiladora; sin embargo, sí se mencionarán e indicarán las áreas o espacios que se usan en la herramienta, también se explicarán las características que tiene la aplicación desarrollada y su funcionamiento interno.

4.1. Entorno de desarrollo

El entorno de desarrollo es una herramienta que permite a partir de un lenguaje de programación, generar una interfaz gráfica de una aplicación, la cual durante las etapas de desarrollo da forma a la aplicación, mejorando su aspecto o añadiendo características o funcionalidades, las aplicaciones desarrolladas suelen ser usadas en Windows o también en otros sistemas operativos.

Para el desarrollo de la aplicación del presente trabajo se ha escogido Windows como sistema operativo, debido a su amplio uso debido a que para la ejecución no se requiere conocer comandos adicionales, a diferencia de otros sistemas operativos.

En cuanto al lenguaje de programación existen varias opciones, las más conocidas son Java, C#, C++ y Visual Basic. De estos lenguajes de programación, Java se descarta porque necesita de un paquete externo instalado en la computadora, Visual Basic también es descartado, porque la estructura de programación suele ser diferente en la anidación de variables, comparado con el tradicional lenguaje C. C# y C++ que en su forma son parecidos entregan resultados idénticos, éstos se diferencian solamente en algunas funciones, por ello da igual escoger cualquiera de los dos lenguajes, así que se elige C# como lenguaje de programación para el desarrollo.

- Plataformas de Desarrollo Disponibles:

  NetBeans IDE: Es una plataforma libre de código abierto [43] basada esencialmente en lenguaje java.

  Visual Studio: Es una plataforma privativa bastante conocida y completa desarrollada por Microsoft [44], posee un amplio uso de lenguajes.

  Dev-C++: Es una aplicación dedicada al desarrollo mediante el lenguaje C /C++ [45], su uso suele ser un poco complicado; ya que el diseño se soporta directamente en código y no en interfaces gráficas.

  Eclipse SDK: Es una plataforma de software [46]de código abierto similar al NetBeans, dedicada también al desarrollo de aplicaciones web.

  Android Studio: Es una plataforma de desarrollo dedicada a aplicaciones móviles [47], soportada por el IDE de desarrollo Eclipse.

  Code::Blocks: Es una plataforma libre [48] similar a dev C++, limitada al lenguaje C / C++.

  Qt Creator: Es un IDE de desarrollo privativo [49] muy similar con Visual Studio, se puede decir que es su competencia.

Comparando la disponibilidad, como también los requerimientos que suelen tener las aplicaciones finales, se escoge Visual Studio community en su versión 2015 [44] ya que este hace uso del lenguaje C#; además, en esta versión las licencias permiten su uso para el desarrollo de aplicaciones como la que se necesita, también las aplicaciones desarrolladas con este IDE o entorno de desarrollo no suelen tener limitaciones en cuanto a necesitar paquetes o programas adicionales instalados.

4.1.1. Reconociendo el entorno de desarrollo

Visual studio Community [44]es un IDE , muy completo de desarrollo de aplicaciones multiplataforma y que se encuentra en constante actualización debido a la aparición de nuevas tecnologías y dispositivos, esta herramienta se basa en la compilación de código en diversos lenguajes entre los cuales se puede encontrar C#. A partir de dicho código, se puede revisar el correcto funcionamiento de las aplicaciones durante su desarrollo, así como la prueba detección de fallos en las aplicaciones a medida que se les agregan nuevas características o funciones, las áreas de trabajo para C# dentro de la herramienta de desarrollo se muestran en la Figura 4.1.

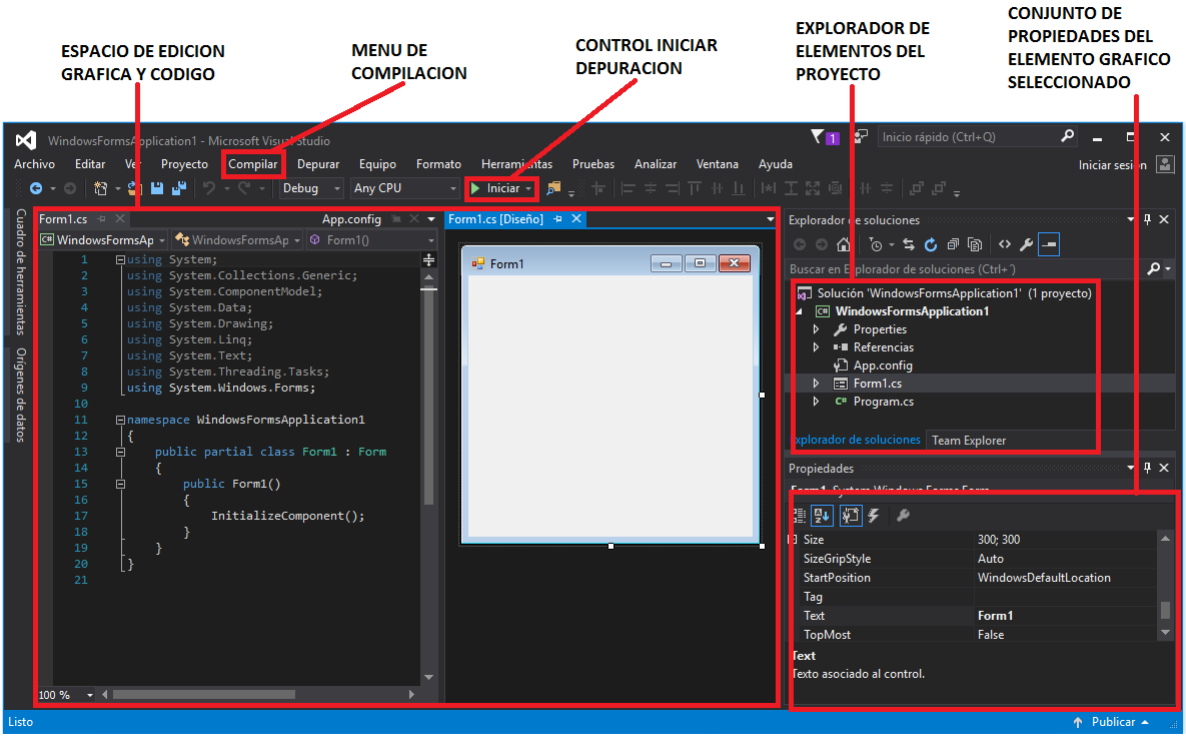

Figura 4.1: Áreas de Trabajo del IDE Visual Studio Community

Ya que se conoce la interfaz de la herramienta de desarrollo, se establecen qué funciones y características debe tener la aplicación que se desea realizar, para ello se tiene en cuenta lo siguiente, la interfaz de la aplicación para el usuario debe ser fácil de entender y clara, no debe ser muy cargada de elementos y tener todas las opciones y menús necesarias para su funcionamiento. A continuación se listan las funciones y características requeridas:

- Subconjunto de elementos de control destinados para la adquisición y aislamiento de señales.
- Subconjunto de elementos de control para la emulación del sistema de eventos discretos.
- Subconjunto de elementos de control para la identificación del autómata asociado a las señales.
- Subconjunto de elementos de control para la identificación de la PN asociada a las señales.
- Interfaz de configuración de señales activas.

4.1.2. Interfaz principal

En el mundo de marketing se busca siempre tener un impacto positivo en aceptación de algún producto en los usuarios ya sea publicitándolo, mostrando sus características o suministrando muestras del mismo, de igual manera esto se aplica en el desarrollo web y desarrollo software. En [50] mencionan un test llamado «Test de los 5 segundos», este test está basado en la observación de los usuarios donde se analiza su comportamiento y se analiza la velocidad de decisión e intuitividad cuando usan una página web, mostrando la capacidad comunicativa del diseño. Esto genera la pregunta ¿que tiene en común el marketing, el test de los 5 segundos con el desarrollo de la interfaz en un software?, pues en [50] hay una nota que dice *«la primera impresión de un usuario al ver el diseño de la interfaz, es una información muy importante sobre su usabilidad»* esto aplicado en el desarrollo de un software indica que la primera impresión que el usuario tiene al ver y usar un software, es la que define si es bien aceptada o no, esto depende principalmente de qué tan intuitiva es la interfaz y de la manera en que los elementos que contiene, se encuentren bien organizados. La aceptación también puede ser diferente dependiendo de quien sea el usuario final.

Pensando en ello se diseña una interfaz con las funciones mínimas necesarias para no recargar de elementos y que de esta forma sea fácil de entender. En la Figura 4.2 se muestra la distribución asignada de la aplicación a diseñar.

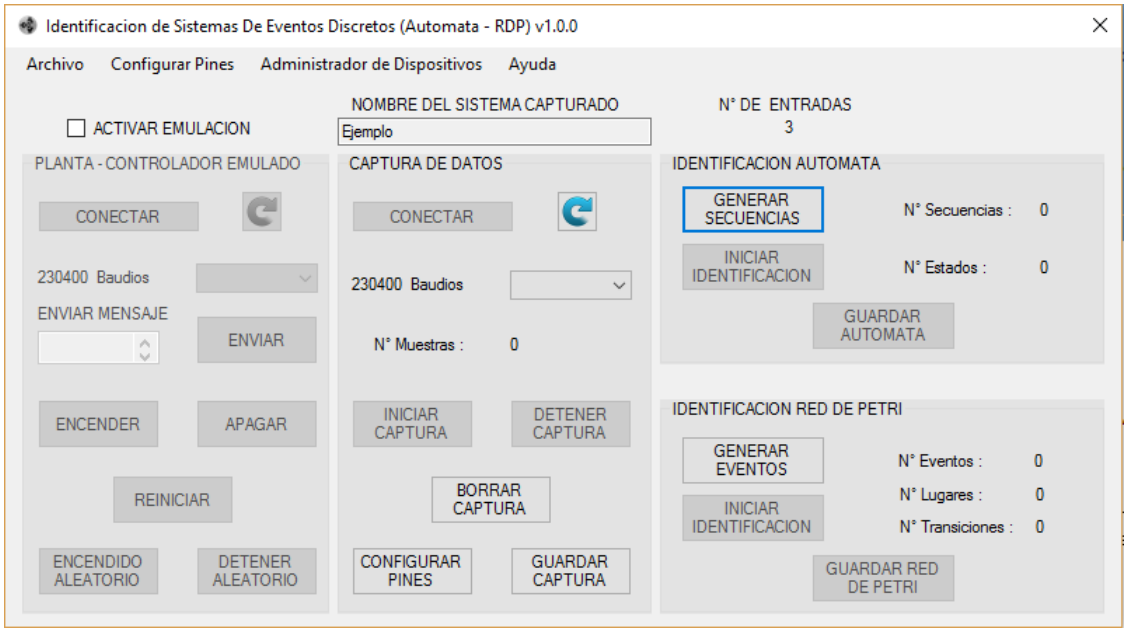

Figura 4.2: Interfaz Principal de la aplicación

Tal como se aprecia en la Figura 4.2, la interfaz queda divida en cuatro segmentos de control como son:

- El emulador de la planta - controlador.
- La adquisición de señales.
- La identificación de autómata.
- La identificación de la PN.

Para los tres últimos segmentos se hace necesario, diseñar una interfaz de configuración de señales, para definir los pines asignados y establecer el tipo de señal.

La interfaz principal también está compuesta por una barra de menús, en donde se cuenta con el acceso directo al administrador de dispositivos, el cual tiene como propósito conocer el nombre del puerto serial COM# asignado por la computadora a las tarjetas de adquisición y de emulación de la planta - controlador.

4.1.3. Interfaz de configuración de señales

La interfaz de configuración hace referencia a los nombres asignados a cada señal de los pines de la tarjeta de desarrollo, cada pin tiene un cuadro de verificación, el cual cumple la función de indicar si dicha señal se tiene o no en cuenta para la identificación,

al igual que otro cuadro de verificación, que está destinado para indicar si una señal es de tipo entrada o salida.

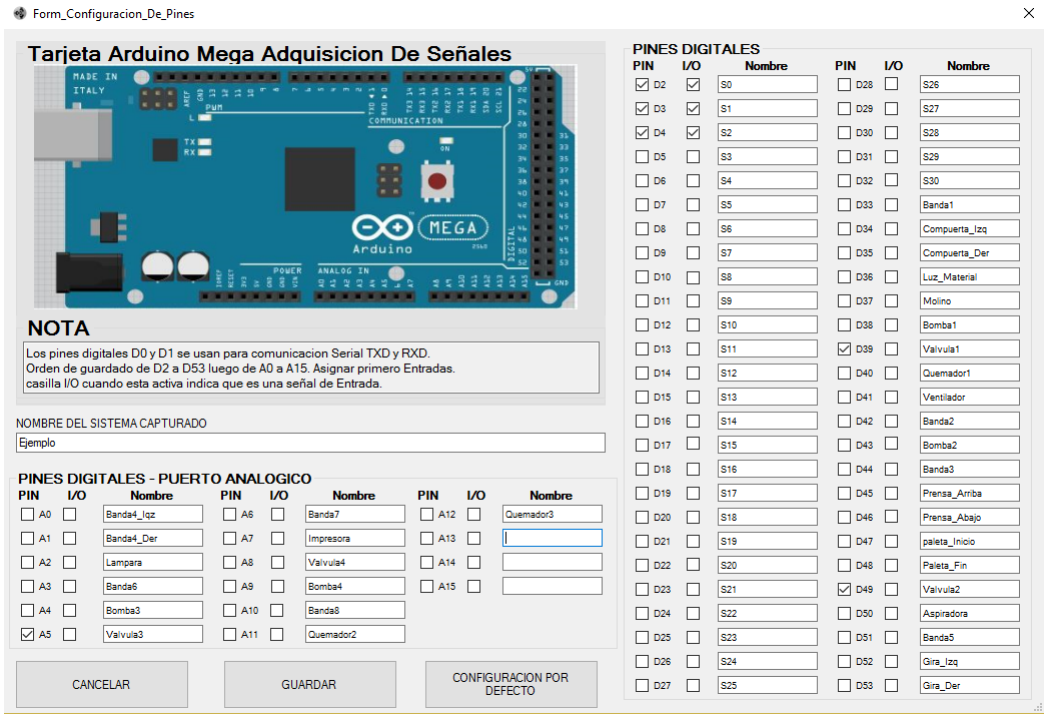

Figura 4.3: Interfaz Configuración de Señales

En esta interfaz Figura 4.3 se tiene en cuenta el orden en que se asignan los tipos de señales, el orden asignado de primero a último corresponde desde el PIN D2 hasta D53 seguido del PIN A0 hasta A15, colocando primero las señales correspondientes a los sensores de la Planta y de señales de mando externas (Encendido...), luego se colocan el resto de señales de órdenes de control, provenientes de las salidas del Controlador, cuando se han realizado modificaciones el archivo de configuraciones de pines es actualizado, permitiendo de esta manera conservar los cambios incluso después de haber cerrado y abierto la aplicación.

4.2. Espacios de la aplicación

4.2.1. Área de planta - controlador

Esta área se encarga de establecer comunicación entre la aplicación y el sistema planta - controlador, emulado en la tarjeta de desarrollo, principalmente se compone de botones

cuya función es enviar instrucciones de propósito general, como: encendido, apagado, reinicio de la simulación interna a las condiciones iniciales y de funciones de activado y desactivado para el encendido aleatorio.

También está compuesto de la opción de enviar parámetros diferentes, aspecto que resulta muy útil en el caso de que existan señales o eventos externos que no hacen parte del comportamiento normal de la planta, como puede ser: un motor dañado, el operario no responde o también cuando el material se ha acabado.

El uso de esta área no es obligatoria para el funcionamiento de la aplicación, por que la adquisición de señales también puede ser obtenida desde una planta real.

4.2.2. Área de aislamiento de señales

En esta área se desarrolla una de las funciones principales de la aplicación la cual mediante la comunicación bidireccional con la tarjeta de adquisición se obtienen los datos en forma de cadena de parámetros hexadecimales, los datos son temporalmente guardados en un fichero «Cactura.tmp» con el objeto de realizar un proceso de aislamiento de señales que consiste en la separación de cada uno de los parámetros contenidos en las muestras mediante la extracción de los bits basándose en la configuración de señales asignadas en la interfaz de pines.

Estos parámetros son organizados formando vectores de unos y cero, representando de esta manera cada una de las muestras obtenidas, para finalmente ser guardados todos los vectores en un fichero con formato «*.CSV» compatible con Excel y con editores de texto, en la Figura 4.4 se muestra el proceso realizado.

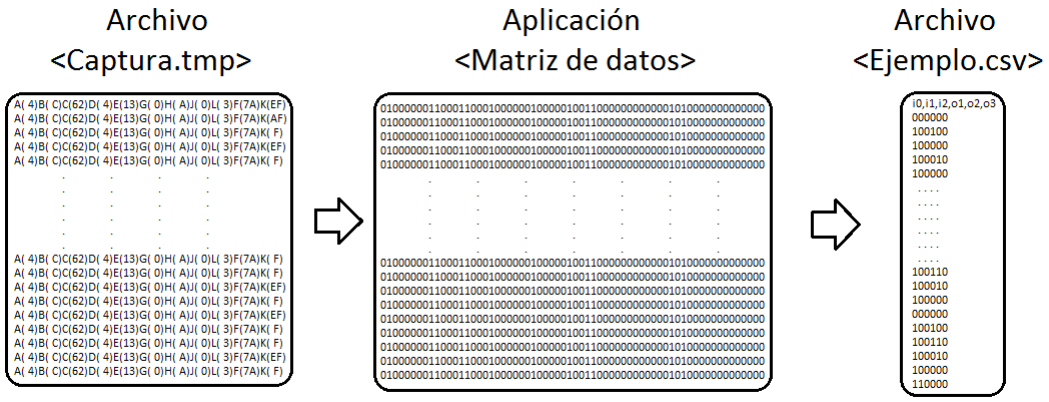

Figura 4.4: Proceso de aislamiento de señales

4.2.3. Área de identificación de autómata

El área de identificación de un autómata parte de la existencia de un archivo con formato «*.CSV», este archivo contiene todos los vectores muestreados, formados por aquellas señales seleccionadas en la interfaz de configuración de señales Figura 4.3.

Partiendo de la Figura 1.1 de la Subsubsección 1.2.3.1 que describe el algoritmo para la identificación del autómata, se realizan 3 pasos:

Primero se crean los conjuntos de estados y transiciones del autómata.

Algoritmo 4.1 Algoritmo Generador de Secuencias

```
private static string curFile_Z_tmp = "./Automata.tmp";
private string curFile_Automata =     "./Automata.aut";

string[] secuencias = null;
string[] secuencia = null;

List<string> X          = new List<string>();          // Conjunto de Estados Guarda los Nombres
List<string> OMEGA      = new List<string>();          // Conjunto de Vectores asociados a los estados X
List<string> LANDA      = new List<string>();          // Conjunto de Simbolos de Salida (Mismo Vector)
List<List<string>> FND  = new List<List<string>>();    // Conjunto de Conjunto de Transiciones asociadas a un estado

Borrar_Archivo(curFile_Z_tmp);
X.Clear();
OMEGA.Clear();
LANDA.Clear();
FND.Clear();

string[] Vectores;
List<string> Lista_Secuencia = new List<string>();

if (File.Exists(curFile_File))
{
    Vectores = System.IO.File.ReadAllLines(curFile_File);
    string X0 = Vectores[1];
    string Actual = X0;
    Lista_Secuencia.Add(Actual);

    Boolean cambio = false;
    for (UInt32 i = 2; i < Vectores.Length; i++)
    {
        if (Vectores[i] != Actual)
        {
            Actual = Vectores[i];
            Lista_Secuencia.Add(Actual);
            if (Actual == X0)
            {
                csvcontent.Clear();
                csvcontent.AppendLine(string.Join(",", Lista_Secuencia.ToArray()));
                File.AppendAllText(curFile_Z_tmp, csvcontent.ToString());
                Lista_Secuencia.Clear();
                Lista_Secuencia.Add(Actual);
                No_Secuencias_Automata.Text = Convert.ToString( Convert.ToUInt32(No_Secuencias_Automata.Text) + 1 );
                cambio = true;
            }
        }
    }
}
```

Segundo se genera un fichero temporal llamado «automata.tmp» el cual va a contener las secuencias observadas en el conjunto de muestras, estas secuencias son obtenidas

mediante el Algoritmo 4.1, este algoritmo toma el primer vector de muestras como vector inicial de sistema a identificar, suele ser el vector obtenido cuando el sistema esta apagado (estado inicial del sistema a identificar), luego busca uno a uno, entre todo el conjunto de muestras un vector de muestra que sea igual al primer elemento observado, cuando se encuentra un vector igual significa que una secuencia es encontrada, entonces ésta es guardada y se continua buscando nuevas secuencias conservando el mismo vector inicial.

El archivo «automata.tmp» es el encargado de guardar las secuencias encontradas dentro del conjunto total de muestras. Para el autómata las secuencias se caracterizan porque su primer y último elemento son el mismo vector, por ello si no existe al menos una secuencia, no se podría realizar una identificación porque el archivo generado en ese caso estaría en blanco.

Algoritmo 4.2 Algoritmo Identificación Progresiva Autómata

```
button_Guardar_Automata.Enabled = false;
secuencias = System.IO.File.ReadAllLines(curFile_Z_tmp);
Boolean Activado = false;   // unicamente primer secuencia sigma que cumple la condicion primer y ultimo iguales
Int32 Contador = 0;         // contador de Total de estados detectados
string Xj = null;           // Guarda el nombre de un estado
string Prevx = null;        // Guarda el estado previo

for (UInt32 n = 0; n < secuencias.Length; n++) // NUmero de Secuencias detectadas
{
    secuencia = secuencias[n].Split(',');
    if (!Activado) // (secuencia[0] == secuencia[secuencia.Length - 1]) Verfica que existe ciclo
    {
        Xj = "X" + Convert.ToString(Contador);  // Crea nombre de estado X0
        X.Add(Xj);                              // Añade un nuevo estado X0
        OMEGA.Add(secuencia[0]);                // OMEGA[t] --> vector
        LANDA.Add(secuencia[0]);                // LANDA[t] --> vector
        FND.Add(new List<string>());            // FND[0] = null; crea un espacio de memoria vacio
        Prevx = Xj;                             // Actualiza el estado previo
        Contador++;
        Activado = true;
    }
    if (Activado)                               // Dentro del ciclo
    {
        for (UInt32 t = 1; t < secuencia.Length; t++)
        {
            if (OMEGA.Contains(secuencia[t]))                       // ya fue visto el vector?
            {
                Xj = X.ElementAt(LANDA.IndexOf(secuencia[t]));      // Obtener el estado asociado Xj
            }
            else                                                    // Es nuevo
            {
                Xj = "X" + Convert.ToString(Contador);              // Crea nombre de estado Xi
                X.Add(Xj);                                          // Añade un nuevo estado Xi
                OMEGA.Add(secuencia[t]);                            // OMEGA[t] --> vector
                LANDA.Add(secuencia[t]);                            // LANDA[t] --> vector
                FND.Add(new List<string>());                        // FND[t] = null;
                Contador++;
            }
            if (!FND.ElementAt(X.IndexOf(Prevx)).Contains(Xj)) // El estado actual no tiene un estado post con
            {                                                  // el nombre Xi?
                FND.ElementAt(X.IndexOf(Prevx)).Add(Xj);       // Crea transicion post estado con Xi
            }
            Prevx = Xj;                                        // Actualiza estado actual con el nuevo estado Xi
        }
    }
}
```

El tercer paso consiste en identificar las secuencias obtenidas del Algoritmo 4.1, en caso de encontrarse al menos una secuencia, el Algoritmo 4.2 de identificación descrito en la tesis doctoral de Stephane Klein [1] es ejecutado, éste consiste en la verificación de secuencias válidas mediante la comparación de igualdad de primer y último elemento de la secuencia, si la secuencia cumple con la condición de que el primer y último elemento sea el mismo, entonces es creado un primer estado «X_i» asociado al vector visto, actualizando el estado actual y es creado un subconjunto de transiciones que almacena los estados posteriores al estado «X_i», luego se analiza el siguiente elemento de la secuencia verificando si el vector ya ha sido asociado con algún estado, en caso de que el vector esté asociado, significa que ya existe un estado «X_v» lo que lleva a verificar si existe una transición asociada al estado «X_v» en el subconjunto de transiciones del estado actual «T_i», si ya existe la transición, no se hace nada y se continua con el siguiente elemento y se actualiza el estado actual, si no existe la transición entonces se crea la transición, se actualiza el estado actual y se continua con el siguiente elemento observado.

Cuando una secuencia no cumple la condición de que el primer y último elemento sea igual, entonces esta secuencia es eliminada y se continua con la siguiente, el proceso se repite continuamente hasta terminar todas las secuencias.

Una vez que se ha terminado de analizar todas las secuencias se realiza un paso adicional que consiste en el guardado del autómata de dos formas, la primera es un script con nombre del sistema analizado bajo el formato «*.AUT», en este archivo se guarda primero el nombre del estado, luego el marcado asociado a dicho estado, seguido de todos los nombres de los estados posteriores a dicho estado (transiciones).

La segunda forma de guardado es mediante la generación automática del grafo en formato imagen «*.PNG», este grafo se da mediante la transcripción del autómata generado en una estructura de tipo DOT, la cual es pasada mediante una herramienta externa llamada Graphviz [51] que es la encargada de generar automáticamente el grafo permitiendo analizar y entender el autómata de manera mas rápida.

4.2.4. Área de identificación de PN

Al igual que se realiza en la identificación del autómata, esta área parte de la existencia del archivo de muestras «*.CSV», a partir de este archivo contenedor se genera un nuevo fichero llamado «RDP.tmp» el cual es el encargado de almacenar los eventos totales del sistema sin contemplar si existe o no ciclos, de esto se encarga la primera parte del Algoritmo 4.3 el cual realiza la diferencia de vector actual con el vector anterior para generar cada uno de los vectores de eventos del conjunto total de muestras.

Algoritmo 4.3 Algoritmo Generador de Eventos

```
private static string curFile_E_tmp = "./RDP.tmp";
private string curFile_RDP = "./RDP.ipn";

List<List<string>> E = new List<List<string>>(); // Conjunto Vectores de Eventos Ocurridos en el Muestreo
List<string> Ev = new List<string>();            // Conjunto de Vectores de Eventos existentes Unicos sin Repeticion
List<Int32> ET = new List<Int32>();              // Conjunto de Vectores de Eventos existentes Repetibles

List<Int32> T = new List<Int32>();                 // Conjunto de Transiciones
string[] W_Entrada = null;
List<string> TLanda = new List<string>();          // Conjunto de simbolos de entrada de evento
List<string> TLandaSalida = new List<string>();    // Conjunto de simbolos de salida de evento
List<Int32> Tpre  = new List<Int32>();             // Conjunto de PRE transicion
List<Int32> Tpost = new List<Int32>();             // Conjunto de POST transicion

Int32 M0 = 0;                                      // ID del lugar inicial
List<string> P = new List<string>();               // Conjunto de Lugares
List<List<Int32>> Ppre  = new List<List<Int32>>(); // Conjunto de arcos de entrada de Lugares
List<List<Int32>> Ppost = new List<List<Int32>>(); // Conjunto de arcos de salida de Lugares

Borrar_Archivo(curFile_E_tmp);
Borrar_Archivo(curFile_RDP);
E.Clear();

UInt32 Contador = 0;
string[] Vectores;

if (File.Exists(curFile_File))
{
    Vectores = System.IO.File.ReadAllLines(curFile_File);
    string Anterior;
    string Actual;
    string evento_segmento;

    for (UInt32 i = 2; i < Vectores.Length; i++)
    {
        Anterior = Vectores[i - 1];
        Actual = Vectores[i];
        E.Add(new List<string>());
        for (UInt16 j = 0; j < Vectores[i].Length; j++)
        {
            evento_segmento = Convert.ToString(Convert.ToUInt16(Actual[j]) - Convert.ToUInt16(Anterior[j]));
            E[Convert.ToInt16(i) - 2].Add(evento_segmento);
        }
        csvcontent.Clear();
        csvcontent.AppendLine(string.Join(",", E[Convert.ToInt16(i) - 2].ToArray()));
        File.AppendAllText(curFile_E_tmp, csvcontent.ToString());
        Contador++;
    }
}
```

Para la identificación de la PN, el Algoritmo 4.4Algoritmo 4.5Algoritmo 4.6 suministrado en la tesis doctoral de Estrada Vargas [30] es traducido de pseudocódigo a lenguaje C#, el código rediseñado se divide en tres partes, debido a que el pseudocódigo no estaba organizado de forma secuencial.

En la primera parte del Algoritmo 4.4 se realiza la inicialización de las variables a usar (conjuntos de lugares, transiciones y arcos) y de la creación de un ciclo FOR el cual consiste en contar o ubicar cual es la muestra actual en cada iteración.

Algoritmo 4.4 Algoritmo Identificación para PN Parte 1

```
Ev.Clear();
ET.Clear();
T.Clear();
W_Entrada = null;
TLanda.Clear();
TLandaSalida.Clear();
Tpre.Clear();
Tpost.Clear();
M0 = 0;
P.Clear();
Ppre.Clear();
Ppost.Clear();

string[] W;  //conjunto de muestras
string[] Tj; //conjunto de eventos

W = System.IO.File.ReadAllLines(curFile_File);
Tj = System.IO.File.ReadAllLines(curFile_E_tmp);
Boolean Nuevo = false;
Int32 TCurrent = 0; //indice de Transicion actual
Int32 PCurrent = 0; //indice Lugar actual

string Entrada = null;
W_Entrada = W[0].Split(',');            // Guarda los nombres de las señales por separado
List<string> Landa_Entradas = new List<string>();
List<string> Landa_Salidas = new List<string>();
Landa_Entradas.Clear();

P.Add(W[1]);                    // Crea el lugar P0 y asigna U(P0) = W[1]
Ppre.Add(new List<Int32>());    // crea un espacio de memoria arcos pre del lugar creado
Ppost.Add(new List<Int32>());   // crea un espacio de memoria arcos post del lugar creado

for (Int32 j = 0; j < Tj.Length; j++)
{
```

A partir del momento en que el Algoritmo 4.5 es inicializado, éste comienza a analizar todos y cada una de los eventos generados anteriormente, el algoritmo consiste primero en la verificación de la existencia del evento observado, en el conjunto de eventos únicos, el cual almacena los vectores de eventos sin repetición, si bien se ha cumplido esta condición existen dos posibles casos, primero se verifica si existe una transición post en el lugar actual que esté relacionada con el evento observado, si esto se cumple obtiene el indice del lugar post de dicha transición encontrada y actualiza las variables de lugar actual y transición actual, en caso de que no exista una transición post que esté asociada a dicho evento observado, se procede a buscar en todo el conjunto de transiciones, una transición que esté asociada al evento observado, y se verifica si el lugar previo a dicha transición tiene el mismo marcado que el vector de muestra actual, si eso es correcto significa se se debe actualizar el arco post de la transición actual, para luego unir los lugares actual con el lugar encontrado, creando de esta manera una secuencia interna de la PN, si se da el caso que no exista un lugar previo que tenga el mismo marcado con la muestra actual, el Algoritmo 4.6 opta por tomar dicho evento como si fuera un evento nuevo, generando una transición y un lugar nuevo para este evento y el marcado actual, la transición creada es relacionada con el vector de evento observado y el lugar relacionado con el marcado actual o muestra de observación actual.

Algoritmo 4.5 Algoritmo Identificación para PN Parte 2

```
if (Ev.Contains( Tj[j] ))   // Verifica si el vector Tj[j] ha sido visto
{   // verfica si existe una transicion post asociada al evento Tj
    if( Ppost[PCurrent].Exists(i => ( ET[i] == Ev.IndexOf(Tj[j])) ) )
    {
        TCurrent = Ppost[PCurrent].Find(i => ( ET[i] == Ev.IndexOf(Tj[j]) ) ); // Actualiza Transicion actual
        PCurrent = Tpost[TCurrent];  // Actualiza Lugar actual
    }   // verifica si existe una transicion asociada al evento Tj
    else if (T.Exists(i => ( ( ET[i] == Ev.IndexOf(Tj[j]) ) && ( P[Tpre[i]] == P[PCurrent] ) ) ) )
    {   // Actualiza Transicion actual
        TCurrent = T.Find(i => ((ET[i] == Ev.IndexOf(Tj[j])) && (P[Tpre[i]] == P[PCurrent])));
        //MessageBox.Show("T" + Convert.ToString(TCurrent) + " Etapa Union " + Convert.ToString(PCurrent));
        for (Int32 v = 0; v < Ppre[PCurrent].Count(); v++)
        {   // Actualiza arcos post del lugar actual con los arcos del lugar a unir
            Tpost[Ppre[PCurrent][v]] = Tpre[TCurrent];
        }
        //MessageBox.Show(string.Join(",", Ppre[Tpre[TCurrent]]) + " union " + string.Join(",", Ppre[PCurrent]));
        //añade al conjunto Ppre del lugar destino el contenido pre del lugar actual
        Ppre[Tpre[TCurrent]] = Ppre[Tpre[TCurrent]].Union(Ppre[PCurrent]).ToList();
        Ppre[PCurrent].Clear();
        //MessageBox.Show(string.Join(",", Ppre[Tpre[TCurrent]]) + " union " + string.Join(",", Ppre[PCurrent]));
        for (Int32 v = 0; v < Ppost[PCurrent].Count(); v++)
        {   // Actualiza arcos pre del lugar actual con los arcos del lugar a unir
            Tpre[Ppost[PCurrent][v]] = Tpre[TCurrent];
        }
        //añade al conjunto Ppost del lugar destino el contenido post del lugar actual
        Ppost[Tpre[TCurrent]] = Ppost[Tpre[TCurrent]].Union(Ppost[PCurrent]).ToList();
        Ppost[PCurrent].Clear();

        if(PCurrent == M0)
        {
            M0 = Tpre[TCurrent]; // Actualiza el marcado inicial de la red de petri si el lugar actual = M0 actual
        }
        // Borrado del lugar que se une
        P.RemoveAt(PCurrent);
        Ppre.RemoveAt(PCurrent);
        Ppost.RemoveAt(PCurrent);
        for(Int32 v = 0; v < T.Count(); v++)
        {
            if(Tpre[v] > PCurrent)
            {
                Tpre[v]--;  //corrige el indice del conjunto Tpre
            }
            if(Tpost[v] > PCurrent)
            {
                Tpost[v]--; //corrige el indice del conjunto Tpost
            }
        }
        PCurrent = Tpost[TCurrent]; // Actualiza Lugar actual
    }
    else    // Tomar el evento como si fuera nuevo
    {
        Nuevo = true;
    }
}
```

La última parte del algoritmo es la encargada de la creación de lugares y transiciones, en este algoritmo se realiza las asociaciones de los eventos con las transiciones y de asociar el marcado actual con el lugar creado, para luego unir estos dos elementos a la estructura de PN que se tiene en el momento mediante el uso de arcos, donde son representados con los indices de los conjuntos de lugares y transiciones.

Algoritmo 4.6 Algoritmo Identificación para PN Parte 3

```
    if( (!Ev.Contains(Tj[j])) || Nuevo)     // vector evento es nuevo
    {
        if (!Ev.Contains(Tj[j]))
        {
            Ev.Add(Tj[j]);  // añade Vector evento unico al conjunto Ev
            // Creacion de simbolos de entrada asignados
            for (Int32 i = 0; i < No_Entradas; i++)
            {
                if (E[j][i] == "1")
                {
                    Entrada = "I" + Convert.ToString(i) + "_1";
                    Landa_Entradas.Add(Entrada);
                }
                else if (E[j][i] == "-1")
                {
                    Entrada = "I" + Convert.ToString(i) + "_0";
                    Landa_Entradas.Add(Entrada);
                }
            }
            if (Landa_Entradas.Count == 0)
            {
                Entrada = "\u03B5"; // caracter epsilon
                Landa_Entradas.Add(Entrada);
            }
            TLanda.Add(string.Join(",", Landa_Entradas));
            Landa_Entradas.Clear();

            // Creacion de simbolos de salida asignados

            for (Int32 i = No_Entradas; i < E[j].Count(); i++)
            {
                if (E[j][i] == "1")
                {
                    Entrada = "O" + Convert.ToString(i- No_Entradas) + "_1";
                    Landa_Salidas.Add(Entrada);
                }
                else if (E[j][i] == "-1")
                {
                    Entrada = "O" + Convert.ToString(i- No_Entradas) + "_0";
                    Landa_Salidas.Add(Entrada);
                }
            }
            if (Landa_Salidas.Count == 0)
            {
                Entrada = "\u03B5"; // caracter epsilon
                Landa_Salidas.Add(Entrada);
            }
            TLandaSalida.Add(string.Join(",", Landa_Salidas));
            Landa_Salidas.Clear();
        }
        ET.Add( Ev.IndexOf(Tj[j]) );   // añade indice del vector evento unico al conjunto ET
        T.Add( ET.Count() - 1 );       // Guarda el indice del evento creado. es el indice del ultimo elemento añadido en ET
        Tpre.Add(PCurrent);            // crea un espacio de memoria arcos pre de la Transicion nueva
        P.Add(W[j + 2]);               // Crea la Marca U(Pj) = W(j)   // el j + 2 elemento W post que genera el evento
        Ppre.Add(new List<Int32>());   // crea un espacio de memoria arcos pre del lugar creado
        Ppost.Add(new List<Int32>());  // crea un espacio de memoria arcos post del lugar creado
        Ppost[PCurrent].Add(T.Count() - 1); // Añade un arco para las transiciones de salida creada (ID transicion creada)
        Tpost.Add(P.Count() - 1);          // Añade un arco para los Lugares de salida de T (es el ID del lugar post)
        Ppre[P.Count() - 1].Add(T.Count() - 1); // Añade arco previo al lugar creado

        PCurrent = P.Count() - 1;              //actualiza lugar actual como el nuevo lugar
        TCurrent = T.Count() - 1;              //actualiza transicion actual como la nueva transicion
        Nuevo = false;
    }
}
```

Cuando se ha generado un lugar y una transición inmediatamente se actualizan las variables de lugar actual Pcurrent y transición actual Tcurrent repitiendo el proceso hasta llegar al último elemento en el conjunto de eventos obtenidos en el muestreo. Una vez que la PN identificada a sido guardada dentro de un script «*.RDP», se crea una segundo guardado que es mediante la generación automática del grafo en formato imagen «*.PNG» este grafo es generado mediante la transcripción de la PN generada

en una estructura de tipo «DOT» la cual es pasada mediante una herramienta externa Graphviz [51] que es la encargada de generar automáticamente el grafo permitiendo analizar y entender la PN de manera mas rápida.

4.3. Generación automática de gráficos

Esta última función de la aplicación desarrollada solo es ejecutada mediante las opciones de guardado en las áreas de identificación, consiste en la generación de un script «*.GV» adaptado mediante una estructura, la cual el programa graphviz es capaz de entender y a partir de éste, generar o crear un archivo imagen que contiene el grafo asociado a cada modelo identificado, para ello se debe descargar e instalar Graphviz [51] que es un software de código abierto de visualización de gráficos usado en diversas áreas de estudio.

Una vez que se ha instalado, por lo general esta aplicación suele crear una integración en el equipo, para poder ser ejecutado mediante de comandos CMD, pero no siempre ocurre por si sola, así que es necesario integrar la aplicación manualmente al símbolo del sistema CMD. (Los pasos a realizar para la integración en las recientes versiones de windows se describen en el «anexo B»)

La generación de los grafos se basa en una estructura base presentada a continuación:

```
digraph G {
subgraph Conjunto_Elemntos {
node [shape=circle];
Elemento0 [peripheries=2,color=lightblue,style=filled,label= "Nombre_Etiqueta"];
Elemento1 [label= "EtNombre_Etiquetaiqueta"];
 .
 .
 .
ElementoN [label= "EtNombre_Etiquetaiqueta"];
}
Elemento0 -> Elemento1,Elemento2
Elemento1 -> Elemento2
 .
 .
 .
ElementoM -> ElementoM

}
```

Figura 4.5: Estructura del script para Gráficos

Como se aprecia en la Figura 4.5, existe un subconjunto llamado Conjunto_elementos, este tipo de subconjuntos es la forma en que cada conjunto ya sean estados, lugares o transiciones deben ser descritos, con el fin de mantener las características propias de cada conjunto, como el tipo de elemento a dibujar, puede ser un rectángulo un círculo o el elemento que se quiera gráficar. Cada elemento dentro de estos subconjuntos hace

referencia a los objetos existentes, y éstos poseen características particulares como son: la etiqueta del marcado actual o la posición actual.

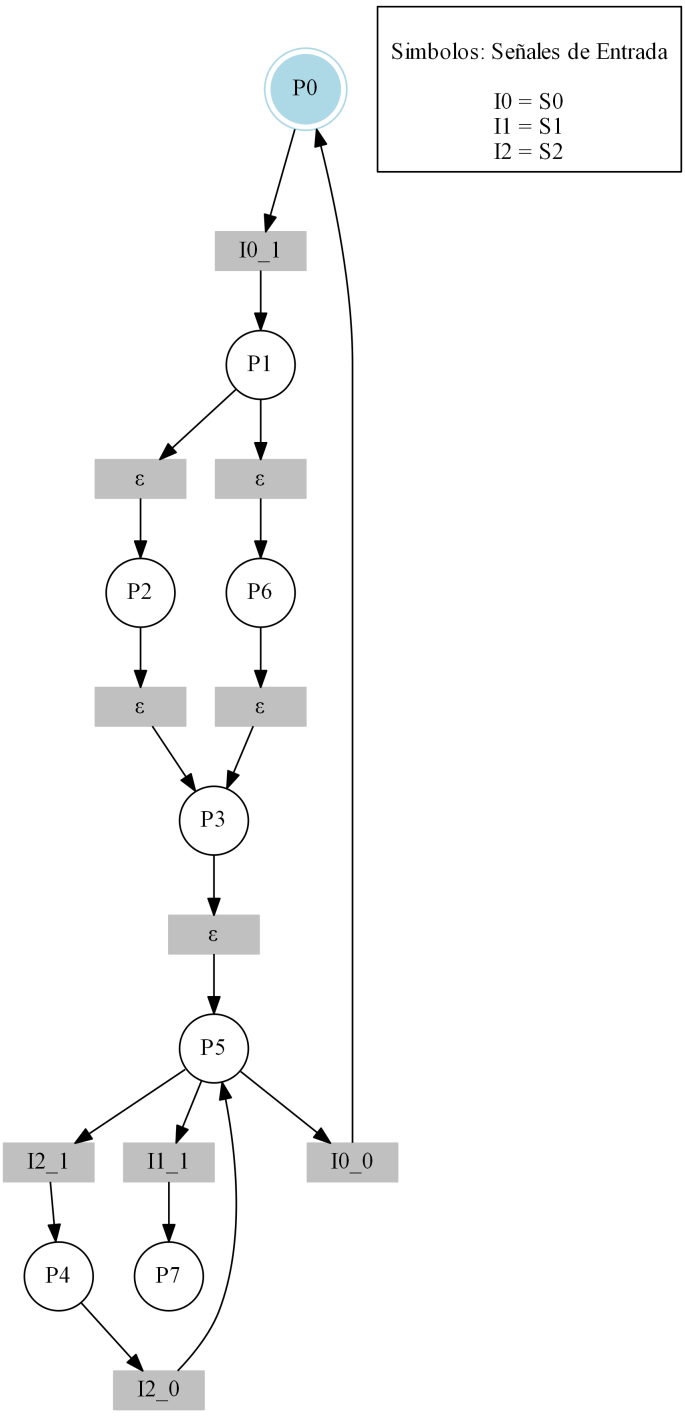

Figura 4.6: Ejemplo PN

Para que en la graficación se obtengan modelos con interconexiones, se debe mencionar el elemento actual, seguido de una flecha con los elementos a los cuales se debe conectar, en el caso de modelos de autómatas sería el elemento estado actual seguido de una flecha "–>" y ésta seguida de todos los elementos incluidos en el conjunto de estados posteriores separados por comas, y para la PN se realiza lo mismo con las transiciones y los lugares.

Las gráficas se obtienen mediante la ejecución del comando CMD: Comando CMD Generador de Gráficos: *« dot -Tpng -Gdpi=300 Ejemplo.gv > Ejemplo.png ».*

El correcto funcionamiento del algoritmo generador de gráficos fue probado con el ejemplo suministrado en [30], Capítulo 3 del algoritmo de identificación de PN, creando un fichero que contiene los vectores de muestras usados en el ejemplo, se obtuvo la figura Figura 4.6 de PN, la cual coincide con la mostrada en dicha referencia. De esta forma se pudo verificar que el algoritmo realizado ha quedado correcto, de igual forma que la programación de la aplicación.

Con base a lo anterior se concluye que la aplicación desarrollada es una buena propuesta para realizar el proceso de identificación de PN y autómatas en el proceso emulado.

5 IDENTIFICACIÓN DEL COMPORTAMIENTO OBSERVABLE Y NO OBSERVABLE EN EL CASO DE ESTUDIO

La identificación del comportamiento observable y no observable se realiza sobre el caso de estudio del DES que emula el proceso de producción de baldosas cerámicas. El sistema general es tratado como un sistema de caja negra del cual solo se conoce sus señales E/S. La identificación del sistema es realizada haciendo uso de la aplicación desarrollada en el Capítulo 4 basándose en los algoritmos suministrados en [1, 30]. La aplicación genera a partir de un amplio grupo de señales E/S u observaciones adquiridas, los modelos de los autómatas e IPNs asociados a cada uno de los subsistemas analizados.

Los modelos obtenidos representan el comportamiento observable. Para identificar comportamientos no observables, en este capítulo se define una prueba de detectabilidad que es aplicada a los subsistemas identificados.

5.1. Identificación del comportamiento observable del sistema.

La identificación del comportamiento observable del sistema hace referencia a la obtención de los modelos de funcionamiento del sistema estudiado a partir de las señales de E/S representados como autómatas e IPNs.

5.1.1. Aplicación de los algoritmos

Con la implementación de la aplicación diseñada en el Capítulo 4 se identifican los modelos de autómatas y PN asociados a los ocho subsistemas que componen el caso

de estudio. Debido al gran tamaño que los subsistemas presentan, solo se coloca el subsistema de menor tamaño obtenido, mientras que el resto de los subsistemas se han colocado en el «Anexo C».

5.1.1.1. Autómata del subsistema Molino, Identificado con el método de Klein

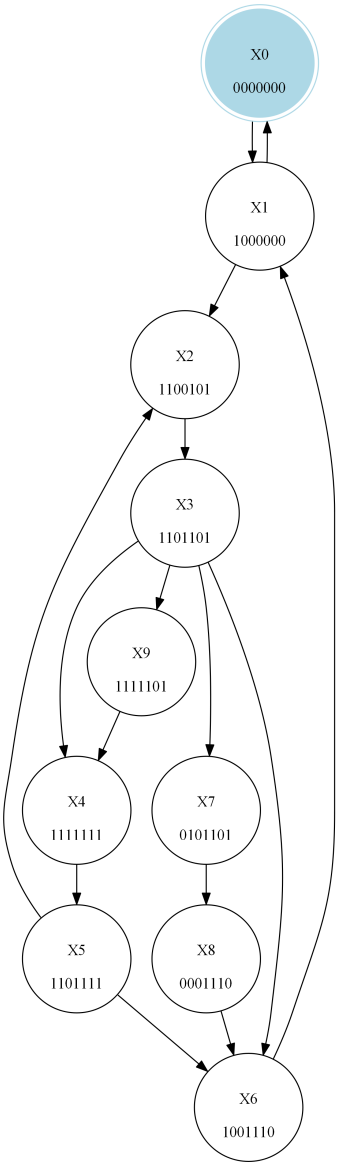

Figura 5.1: Autómata identificado subsistema molino

En la Figura 5.1 se aprecia el modelo del autómata identificado del subsistema molino, el cual describe una estructura simple de estados, la cual representa solamente los ciclos

encontrados en el sistema, tomando como estado inicial el estado marcado en color azul y el marcado de los estados equivale solo al vector sin procesar de las observaciones únicas encontradas. La identificación se realizó en tres partes, la primera parte de generación de secuencias válidas encontrando 28 secuencias tardando 35,5126 mili-segundos, la segunda parte que consiste en la identificación del subsistema se tardó 5,0024 mili-segundos, obteniendo 10 estados.

```
X0,0000000,X1
X1,1000000,X2,X0
X2,1100101,X3
X3,1101101,X4,X6,X7,X9
X4,1111111,X5
X5,1101111,X2,X6
X6,1001110,X1
X7,0101101,X8
X8,0001110,X6
X9,1111101,X4
```

Tabla 5.1: Código Autómata subsistema molino

La última parte del proceso es mediante el sistema identificado en memoria, el cual es guardado en un fichero bajo el nombre «_2_Molino.aut» observable en la Tabla 5.1. En este se guardan los nombres de cada estado seguidos del marcado asociado, que es la observación sin repetición, seguidas del conjunto de nombres de los estados con los que se conecta mediante transiciones. El guardado de la identificación se tardó 27,0183 mili-segundos, también en esta parte se genera la gráfica automática del autómata identificado, esto suele tardar algunos segundos debido a que depende del procesamiento externo de la aplicación Graphviz [51].

5.1.1.2. IPN del subsistema Molino, Identificada con el método de Estrada - Vargas

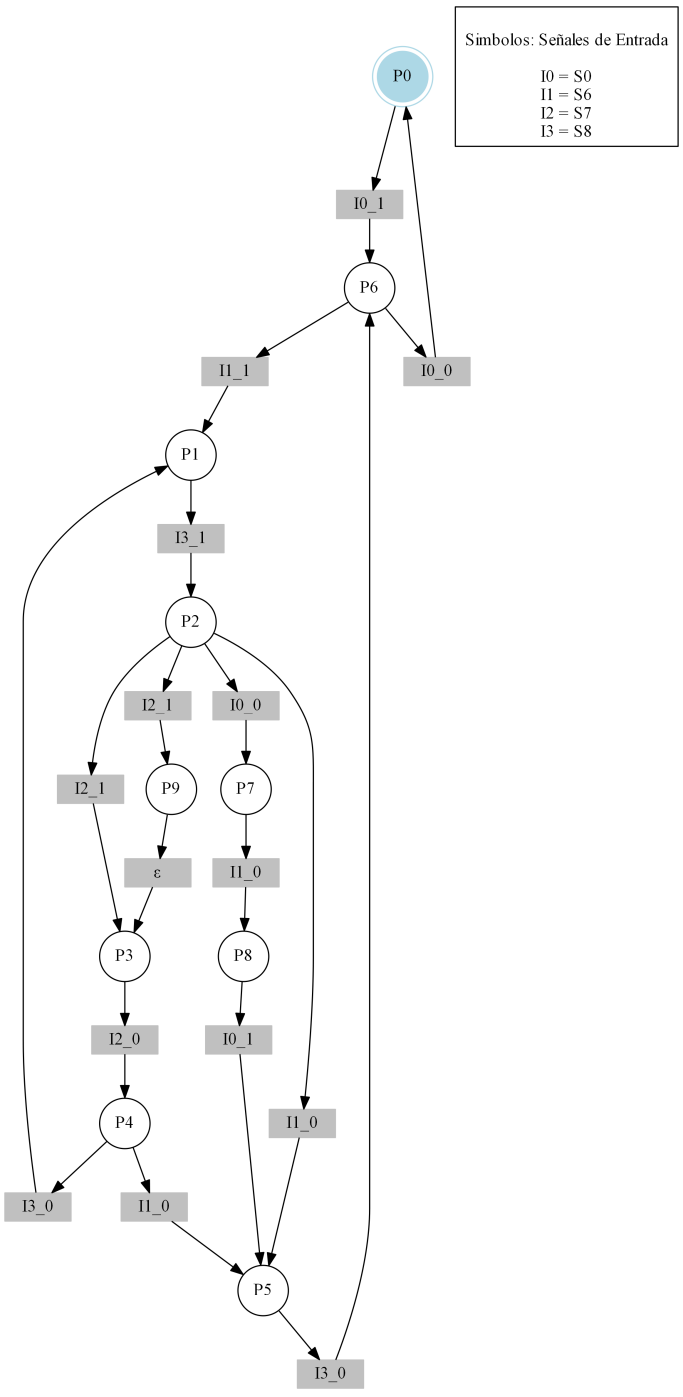

Figura 5.2: PN identificada subsistema molino

En la Figura 5.2 se aprecia el modelo de la PN identificada del subsistema molino, el cual describe una estructura más elaborada de lugares y transiciones donde se representa la dinámica de comportamiento, logrando observar el sistema de manera más completa; ya que, en este se procesan todas las observaciones. La identificación se realizó en tres partes, la primera parte de generación de eventos con un total de 145 observaciones se tardó 96,061 mili-segundos, la segunda parte que consiste en la identificación del subsistema se tardó 26,8157 mili-segundos, reduciendo el número de eventos a 12, obteniendo 10 lugares y 15 transiciones.

S0,S6,S7,S8,Molino,Bomba1,Valvula1,
I0,I1,I2,I3,O0,O1,O2,O3
M0,P0
E0,1,0,0,0,0,0,0,I0_1|||ε
E1,0,1,0,0,1,0,1,I1_1|||O0_1,O2_1
E2,0,0,0,1,0,0,0,I3_1|||ε
E3,0,0,1,0,0,1,0,I2_1|||O1_1
E4,0,0,-1,0,0,0,0,I2_0|||ε
E5,0,0,0,-1,0,-1,0,I3_0|||O1_0
E6,0,-1,0,0,0,1,-1,I1_0|||O1_1,O2_0
E7,0,0,0,-1,-1,-1,0,I3_0|||O0_0,O1_0
E8,-1,0,0,0,0,0,0,I0_0|||ε
E9,0,-1,0,0,0,0,-1,I1_0|||O2_0
E10,0,0,1,0,0,0,0,I2_1|||ε
E11,0,0,0,0,0,1,0,ε|||O1_1
P0,0000000,T8|||T0
P1,1100101,T1,T5|||T2
P2,1101101,T2|||T3,T6,T10,T13
P3,1111111,T3,T14|||T4
P4,1101111,T4|||T5,T9
P5,1001110,T6,T9,T12|||T7
P6,1000000,T7,T0|||T8,T1
P7,0101101,T10|||T11
P8,0001110,T11|||T12
P9,1111101,T13|||T14
T0,E0,P0,P6,I0_1
T1,E1,P6,P1,I1_1
T2,E2,P1,P2,I3_1
T3,E3,P2,P3,I2_1
T4,E4,P3,P4,I2_0

Tabla 5.2: Código PN subsistema molino Parte 1

T5,E5,P4,P1,I3_0
T6,E6,P2,P5,I1_0
T7,E7,P5,P6,I3_0
T8,E8,P6,P0,I0_0
T9,E9,P4,P5,I1_0
T10,E8,P2,P7,I0_0
T11,E6,P7,P8,I1_0
T12,E0,P8,P5,I0_1
T13,E10,P2,P9,I2_1
T14,E11,P9,P3,ε

Tabla 5.3: Código PN subsistema molino Parte 2

La última parte del proceso es mediante el sistema identificado en memoria el cual es guardado en un fichero bajo el nombre «_2_Molino.ipn» observable en la Tabla 5.2, Tabla 5.3en este se guardan los nombres de cada señal, la señal abreviada, el conjunto de eventos únicos observados, el conjunto de lugares y las transiciones tardando 30,0189 mili-segundos, también en esta parte es donde se genera la gráfica automática de la PN identificada, esto suele tardar algunos segundos debido a que depende del procesamiento externo de la aplicación Graphviz [51], en el modelo graficado el lugar inicial está determinado por color azul.

5.1.2. Comparación de los resultados con las dos metodologías

A partir del tiempo en mili-segundos obtenido en una computadora de características descritas en la Tabla 5.4, se puede deducir que los tiempos en que el sistema tardará en completar la identificación son considerablemente pequeños, por lo general para el caso de la identificación de la IPN tarda más en comparación al del autómata, esto puede deberse a que el número de operaciones en el algoritmo de identificación es mayor, el aumento del tiempo también se debe a que durante la preparación de las observaciones de E/S, se eliminan algunos vectores de observaciones para el caso de los autómatas, por ello la IPN consume aparentemente tiempo adicional.

Memora Ram: 12GB
Memoria de Video: 8GB
Procesador: Intel i5 2450M x64
Numero Nucleos: 4
Velocidad: 2500 - 3200Mhz
Sistema Operativo: Windows 10

Tabla 5.4: Características de computadora utilizada

Con base a los tiempos obtenidos durante el proceso de identificación en todos los subsistemas, se genera la Tabla 5.5 y la Tabla 5.6 en las cuales se organizan los tiempos de preparación de observaciones E/S y de identificación para ambos casos.

TIEMPOS AUTÓMATAS				
Generación [ms]	Identificación [ms]	Total [ms]	Generación [%]	Identificación [%]
19,0129	6,5045	25,5174	74,51	25,49
35,5126	5,0024	40,515	87,65	12,35
32,0206	7,0053	39,0259	82,05	17,95
15,6273	31,2328	46,8601	33,35	66,65
43,0327	27,0186	70,0513	61,43	38,57
63,5437	12,5118	76,0555	83,55	16,45
62,0403	15,5132	77,5535	80,00	20,00
30,0213	7,5041	37,5254	80,00	20,00
PROMEDIO			72,82	27,18

Tabla 5.5: Tabla Tiempos Autómatas

TIEMPOS REDES DE PETRI INTERPRETADA				
Generación [ms]	Identificación [ms]	Total [ms]	Generación [%]	Identificación [%]
168,7143	23,0152	191,7295	88,00	12,00
96,061	26,8157	122,8767	78,18	21,82
46,8572	15,6068	62,464	75,01	24,99
7240,1619	65,0479	7305,2098	99,11	0,89
7654,3947	62,5075	7716,9022	99,19	0,81
3999,2052	51,5348	4050,74	98,73	1,27
3995,4427	38,5275	4033,9702	99,04	0,96
78,1109	31,2525	109,3634	71,42	28,58
PROMEDIO			88,59	11,41

Tabla 5.6: Tabla Tiempos IPNs

para determinar que etapa del proceso de la identificación de autómata e IPN presenta mayor consumo de recursos computacionales, se calcula el tiempo porcentual que tarda la aplicación desarrollada en el Capítulo 4 en preparar y procesar las observaciones E/S al igual que el tiempo porcentual que se tarda en identificar cada modelo del DES. Tomando el tiempo total de cada subsistema como el 100 %, se determina el porcentaje equivalente para la identificación y la preparación de E/S.

La relación entre identificación y el tiempo de preparación según la Tabla 5.5 y la Tabla 5.6 presenta un comportamiento casi constante entre 70 % y 80 % independientemente del tamaño del sistema evaluado, como el DES del caso de estudio consiste en varios subsistemas, se calcula el valor promedio porcentual de los sub - sistemas y

luego son analizados logrando determinar que la mayor parte del tiempo para ambos métodos es dedicada a la preparación de las observaciones E/S traduciéndose esto como mayor consumo de recursos computacionales debido a que la aplicación hace uso de almacenamiento de archivos temporales durante la preparación de señales y este tiempo suele variar en función de las características de la computadora en que se esté trabajando.

De los tiempos anteriores, la identificación para autómatas con respecto a la identificación de IPNs es efectivamente mas rápida a causa de que el algoritmo en si no presenta demasiada complejidad; sin embargo, el hecho de ser más rápida no significa que el modelo obtenido sea mejor o brinde suficiente información, además el tiempo de guardado de los dos modelos identificados no depende del algoritmo sino más bien de la velocidad de guardado del dispositivo de almacenamiento, por ello no se tuvo en cuenta.

5.2. Identificación del comportamiento no-observable

5.2.1. Definición del comportamiento observable y no observable

Para un DES el lazo cerrado planta – controlador, el comportamiento de este sistema puede ser visto de dos maneras:

- Comportamiento observable, relacionado con los cambios en las señales de salida a partir de los cambios en la señales de entrada.
- Comportamiento No-observable que se genera cuando existe evolución en los estados internos del sistema sin cambio observable en las señales de entrada – salida.

Los Autómatas y las IPNs modeladas a partir de la señales de E/S, con base a los algoritmos presentados en [1, 30] proporcionan la información necesaria para identificar el comportamiento observable, definiendo las funciones de transición de estados con señales de E/S en el algoritmo de Autómatas y definiendo transiciones y lugares etiquetados con señales en el algoritmo con base a IPNs; pero ninguno de los dos modela comportamientos no-observables.

5.2.2. Definición de prueba de detectabilidad y aplicación en el DES

Tomando como base las propuestas de detectabilidad de la Subsección 1.2.4 se plantea una prueba de detectabilidad de eventos no-observables, la cual se aplica para los modelos de IPNs generadas a partir del algoritmo de [30]. Para ello se crean las siguientes definiciones:

Transición_observable: Una transición T_i es observable si el símbolo asociado a dicha transición es diferente de ε.

Transición_no_observable: Una transición T_i es no observable si el símbolo asociado a dicha transición es igual a ε.

Lugar_observable: Un lugar P_i es observable si en el conjunto de transiciones previas al lugar P_i, para cada transición se determina que son observables.

Lugar_no_observable: Un lugar P_i es no-observable si en el conjunto de transiciones previas al lugar P_i, se determina que son no observables.

Lugar_parcialmente_observable: Un lugar P_i es parcialmente observable si en el conjunto de transiciones previas al lugar P_i, al menos una transición es no observable.

Con base a las cinco definiciones anteriores se establece un algoritmo en seudo - código (Algoritmo 5.1) se define y se realiza una prueba de detectabilidad para los modelos de IPN obtenidos en [30], tomando los conjuntos de eventos únicos observados, lugares, transiciones y conjuntos de lugares y transiciones previas y posteriores.

Algoritmo 5.1 Algoritmo de detectabilidad para IPN

- busca en el conjunto de eventos únicos E aquellos eventos E_i que tiene símbolo de entrada igual a ε.
  - Busca en el conjunto de transiciones T aquellas transiciones T_i que están asociadas a los eventos encontrados E_i.

    Se asigna a las transiciones T_i como transiciones no observables.
    - Buscar los lugares P_i posteriores a las transiciones T_i encontradas.
      - Si existe transiciones T_i observables en el conjunto previo a los lugares P_i.

        El lugar P_i es parcialmente observable.
      - No

        El lugar es completamente no observable.

La detectabilidad de un DES se ve afectada principalmente por la velocidad que el sistema tiene, cuando el DES es demasiado rápido y el dispositivo de adquisición de señales no lo es, puede ocurrir que algunos datos sean ignorados, cuando esto ocurre se generan eventos de E/S en los cuales se evidencia como una respuesta instantánea del controlador del sistema. Si bien en algunos casos esto suele ocurrir, no siempre indican comportamiento errático del DES.

Para entender la ocurrencia de transiciones y lugares no observables se realiza un pequeño ejemplo con el siguiente conjunto de un muestreo de E/S.

$$\sigma = \begin{Bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{Bmatrix} \text{ que genera los eventos } Ev = \begin{Bmatrix} 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

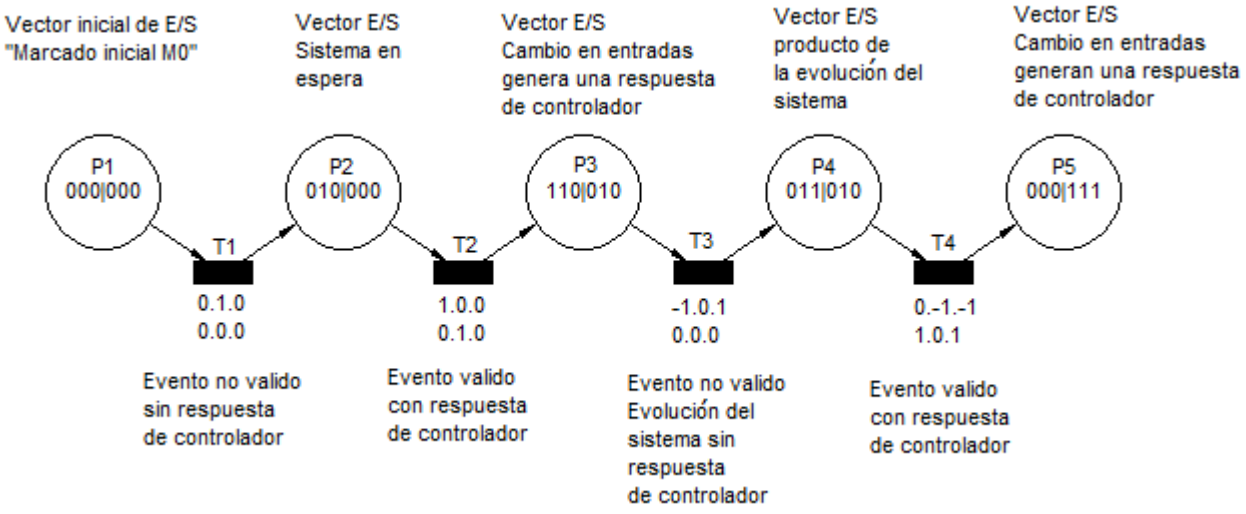

Figura 5.3: Ejemplo IPN detectabilidad.

Al revisar la Figura 5.3 de IPN con base a los conjuntos σ y Ev, se aprecia que existen tres tipos de eventos, el primer evento es del tipo "evolución del sistema sin respuesta de controlador" este evento es el que ocurre después de que el controlador ha enviado una orden de control nueva y durante ese proceso se genera una transición observable, el segundo tipo de evento es el de la "ocurrencia de una orden de control sin evidencia de cambios en las señales de entrada", este tipo de evento da origen a un evento no observable pero que es válido para el sistema, en algunos casos también puede tratarse de un evento de fallo en el sistema, como también retardos considerables en la toma de decisiones del controlador del sistema.

El último tipo de evento posible es "respuesta instantánea de controlador ante un evento de entrada del sistema" este evento suele ocurrir cuando se trata de sistemas más rápidos que el dispositivo de adquisición, o también cuando se tiene señales de entrada (sensores) sin retardo, clasificando este tipo de eventos como eventos observables.

Con base a la prueba de detectabilidad planteada, los lugares son dependientes de las transiciones previas, esto permite determinar de manera rápida la detectabilidad del sistema; ya que, para saber el estado actual solo se necesita que se cumpla, que evento

y el marcado de un lugar leído esté relacionado, logrando de esta manera identificar los segmentos de comportamiento dinámico (evolución del DES) y los segmentos de órdenes de control facilitando futuros análisis del sistema para detección de fallos en los DES.

5.2.3. Aplicación de algoritmo de detectabilidad en caso de estudio

En esta subsección se muestran los resultados de la prueba de detectabilidad para identificación del comportamiento no observable sobre las redes de petri obtenidas en la identificación del comportamiento observable del DES.

El resultado de la identificación no observable para cada uno de los subsistemas dio origen a la obtención de 3 conjuntos de elementos, el primer conjunto es el de eventos observados que presentan un evento de disparo en las señales de salida y no en las señales de entrada, en este se clasificaron los eventos encontrados como eventos de comportamiento no observable, el segundo conjunto se encuentra formado por aquellas transiciones que durante el proceso de identificación observable, se les asociaron con los eventos de comportamiento no observables del primer conjuntos identificando estas transiciones se les llama transiciones no observables. El ultimo conjunto es el conjunto formado por los lugares posteriores a las transiciones no observables encontradas, este conjuntos clasifica a los lugares encontrados como lugares no observables y lugares parcialmente observables.

El comportamiento no observable identificado puede tener diferentes causas para todos los 8 subsistemas del caso de estudio por ello a continuación se presentan la identificación no observable obtenida de cada uno de los subsistemas estudiados.

5.2.3.1. Suministro de materias primas

La identificación no observable para el subsistema de suministro de materia prima presente en la Tabla 5.7, indica que existe un cambio o evolución de las señales de salida cuando se ha recibido una orden del controlador asociada a la luz de petición de material, y la orden de apertura o cerrado de compuerta. Este comportamiento encontrado es causado por los retardos generados ya sea por causas externas (humana) y por la dinámica que presenta la planta ante una orden de control con la compuerta.

S0,S1,S2,S3,S4,S5,S6,S9,S10,Banda1,Compuerta_Izq,Compuerta_Der,Luz_Material
E13,0,0,0,0,0,0,0,0,0,0,0,0,-1,ε|||O3_0
E15,0,0,0,0,0,0,0,0,0,0,0,1,0,0,ε|||O1_1
E31,0,0,0,0,0,0,0,0,0,0,0,0,0,1,ε|||O3_1
TRANSICIONES NO OBSERVABLES
T17,E13,P15,P11,ε
T21,E15,P17,P18,ε
T52,E31,P40,P13,ε
LUGARES PARCIALMENTE OBSERVABLES
P11,1111011001000,T10,T17|||T11
P13,1100011011001,T13,T52|||T14
LUGARES NO OBSERVABLES
P18,1110011111100,T21|||T22

Tabla 5.7: Identificación no observable Suministro de materias primas

5.2.3.2. Molienda de material

El comportamiento no observable identificado para el subsistema de molienda de material visto en la Tabla 5.8, indica que el evento no observable solo ocurre cuando la Bomba1 es encendida, esto quiere decir que el efecto de encendido de la Bomba1 para las señales de entrada no es instantáneo entendiéndose como un retardo entre encendido de la Bomba1 y algún cambio en las señales de entrada del subsistema.

S0,S6,S7,S8,Molino,Bomba1,Valvula1
E11,0,0,0,0,0,1,0,ε|||O1_1
TRANSICIONES NO OBSERVABLES
T14,E11,P9,P3,ε
LUGARES PARCIALMENTE OBSERVABLES
P3,1111111,T3,T14|||T4

Tabla 5.8: Identificación no observable Molienda de material

5.2.3.3. Atomizado y secado

En el subsistema de Atomizado y secado, la identificación del comportamiento no observable presente en la Tabla 5.9, arroja que existen dos instantes en que se presenta evolución o cambio en los estados del subsistema, los cuales solo ocurren cuando la Banda2,Bomba2,Banda3 son encendidos.

S0,S11,S12,S13,S15,S16,Quemador1,Ventilador,Banda2,Bomba2,Banda3
E16,0,0,0,0,0,0,0,0,0,1,1,0,ε|||O2_1,O3_1
E17,0,0,0,0,0,0,0,0,0,0,0,1,ε|||O4_1
TRANSICIONES NO OBSERVABLES
T20,E16,P14,P15,ε
T22,E17,P16,P17,ε
LUGARES NO OBSERVABLES
P15,11010011110,T20|||T21
LUGARES PARCIALMENTE OBSERVABLES
P17,11110011111,T22,T3,T24|||T23,T4,T12,T25

Tabla 5.9: Identificación no observable Atomizado y secado

5.2.3.4. Prensado

La identificación de comportamiento no observable del subsistema de prensado presente en la Tabla 5.10, arroja que todos el actuadores del subsistema son desencadenadores de eventos no observables, donde todas las señales de salida presentan ambos cambios de estado “1” y “-1”, esto podría darse porque está presente un comportamiento ON - OFF o porque durante el funcionamiento del subsistema existe una dinámica cíclica.

S0,S14,S17,S18,S19,S20,S21,S30,Prensa_Arriba,Prensa_Abajo,paleta_Inicio,
Paleta_Fin,Valvula2,Aspiradora

E14,0,0,0,0,0,0,0,0,0,-1,0,0,1,1,1,ε|||O0_0,O3_1,O4_1,O5_1
E16,0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,ε|||O1_1,O2_0
E18,0,0,0,0,0,0,0,0,0,0,0,1,-1,-1,-1,ε|||O2_1,O3_0,O4_0,O5_0
E20,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,ε|||O0_1,O1_0
E22,0,0,0,0,0,0,0,0,0,0,0,0,-1,-1,-1,ε|||O3_0,O4_0,O5_0
E28,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,ε|||O0_1
E30,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,ε|||O1_1
E34,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,ε|||O5_0
E36,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,ε|||O2_0
E39,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,ε|||O0_0
E41,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,ε|||O5_1
E46,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,ε|||O1_0
TRANSICIONES NO OBSERVABLES
T14,E14,P13,P3,ε
T16,E16,P14,P8,ε
T20,E20,P16,P11,ε
T22,E22,P17,P6,ε
T33,E28,P26,P11,ε
T39,E30,P31,P8,ε
T45,E34,P33,P6,ε
T47,E36,P34,P8,ε
T60,E20,P44,P45,ε
T63,E39,P47,P2,ε
T65,E41,P48,P3,ε
T81,E46,P56,P11,ε
LUGARES PARCIALMENTE OBSERVABLES
P2,10101001000000,T1,T63,T72|||T2,T87
P3,11101001000111,T2,T12,T14,T65,T86|||T3,T79
P6,11100111001000,T5,T18,T22,T45|||T6,T66
P8,11101011010000,T7,T16,T39,T42,T47|||T8
P11,11011001100000,T10,T20,T33,T43,T81|||T11
P45,10011001100000,T60,T71|||T61

Tabla 5.10: Identificación no observable Prensado Parte 1

5.2.3.5. Girado

La identificación del subsistema de girado vista en la Tabla 5.11, indica que los eventos no observables son generados por todas señales de salida asociadas a los actuadores, donde solo una de las señales de salida presenta un solo cambios de estado “1” o “-1”, esto podría indicar que el cambio de estado para esta señal no presentan una dinámica cíclica aparente.

S0,S22,S23,S24,S25,S30,Banda5,Gira_Izq,Gira_Der,Banda4_Iqz,Banda4_Der
E13,0,0,0,0,0,0,0,0,1,-1,0,ε|||O2_1,O3_0
E15,0,0,0,0,0,0,0,-1,0,1,0,ε|||O1_0,O3_1
E16,0,0,0,0,0,0,0,0,-1,0,1,ε|||O2_0,O4_1
E18,0,0,0,0,0,0,0,1,0,0,-1,ε|||O1_1,O4_0
E20,0,0,0,0,0,0,0,0,0,0,-1,ε|||O4_0
E23,0,0,0,0,0,0,0,0,0,0,1,ε|||O4_1
E25,0,0,0,0,0,0,0,0,0,1,0,ε|||O3_1
E32,0,0,0,0,0,0,1,0,0,0,1,ε|||O0_1,O4_1
E37,0,0,0,0,0,0,0,0,0,-1,0,ε|||O3_0
TRANSICIONES NO OBSERVABLES
T13,E13,P11,P7,ε
T15,E15,P12,P5,ε
T17,E16,P13,P2,ε
T19,E18,P14,P3,ε
T21,E20,P15,P3,ε
T24,E23,P16,P2,ε
T26,E25,P17,P5,ε
T38,E32,P25,P2,ε
T44,E37,P27,P7,ε
LUGARES PARCIALMENTE OBSERVABLES
P2,11000110001,T1,T11,T17,T24,T34,T38,T48|||T2,T18,T20,T35
P3,11010111000,T2,T19,T21|||T3
P5,10110110010,T4,T15,T26,T52|||T5
P7,10101110100,T6,T13,T9,T44|||T7,T10

Tabla 5.11: Identificación no observable Girado

5.2.3.6. Secado por luz UV

La identificación no observable del subsistema de secado por luz UV se muestra en la Tabla 5.12, en este se aprecia que los eventos no observables son generados por cambios en todas las señales de salida asociadas a los actuadores, donde solo una de las señales de salida presenta ambos cambios de estado “1” y “-1”, esto podría darse porque presenta un comportamiento ON - OFF o porque durante el funcionamiento del subsistema existe una dinámica cíclica.

S0,S25,S26,S30,Lampara,Banda6,Bomba3,Valvula3
E7,0,0,0,0,1,0,0,0,ε|||O0_1
E9,0,0,0,0,-1,0,0,0,ε|||O0_0
E15,0,0,0,0,0,1,1,1,ε|||O1_1,O2_1,O3_1
TRANSICIONES NO OBSERVABLES
T7,E7,P6,P3,ε
T9,E9,P7,P5,ε
T15,E15,P10,P2,ε
LUGARES PARCIALMENTE OBSERVABLES
P2,10010111,T1,T5,T15|||T2,T6,T13
P3,11011111,T2,T7,T21|||T3,T17,T18
P5,10110111,T4,T9,T17|||T5,T10,T16

Tabla 5.12: Identificación no observable Secado por luz UV

5.2.3.7. Impresión

En el subsistema de impresión, la identificación del comportamiento no observable se muestra en la Tabla 5.13, donde se aprecia que todas las señales de salida asociadas a los actuadores desencadenan eventos no observables, sin embargo solo una de las señales de salida presenta ambos cambios de estado “1” y “-1”, esto podría darse porque presenta un comportamiento ON - OFF o porque durante el funcionamiento del subsistema existe una dinámica cíclica.

S0,S27,S28,S30,Banda7,Impresora,Valvula4,Bomba4
E7,0,0,0,0,0,-1,0,0,ε|||O1_0
E9,0,0,0,0,0,1,0,0,ε|||O1_1
E13,0,0,0,0,1,0,1,1,ε|||O0_1,O2_1,O3_1
TRANSICIONES NO OBSERVABLES
T7,E7,P6,P5,ε
T9,E9,P7,P3,ε
T13,E13,P9,P2,ε
LUGARES PARCIALMENTE OBSERVABLES
P2,10011011,T1,T5,T13|||T2,T8,T10
P3,11011111,T2,T9|||T3,T18
P5,10111011,T4,T7,T18|||T5

Tabla 5.13: Identificación no observable Impresión

5.2.3.8. Horneado

la identificación del comportamiento no observable del subsistema de horneado presente en la Tabla 5.14, indica que los eventos no observables son desencadenados por todas las señales asociadas a los actuadores del subsistema, donde solo uno de las señales de salida presenta ambos cambios de estado "1" y "-1", esto podría darse porque presenta un comportamiento ON - OFF o porque durante el funcionamiento del subsistema existe una dinámica cíclica.

S0,S29,S30,Banda8,Quemador2,Quemador3
E8,0,0,0,1,0,-1,ε|||O0_1,O2_0
E10,0,0,0,0,1,1,ε|||O1_1,O2_1
E12,0,0,0,0,1,0,ε|||O1_1
E13,0,0,0,1,0,0,ε|||O0_1
TRANSICIONES NO OBSERVABLES
T8,E8,P6,P2,ε
T10,E10,P7,P5,ε
T15,E12,P10,P1,ε
T17,E13,P11,P8,ε
LUGARES PARCIALMENTE OBSERVABLES
P1,100010,T0,T15|||T1,T16
P2,111110,T2,T6,T8|||T3,T11
P5,110011,T5,T10|||T6,T7
P8,101110,T11,T1,T17|||T12,T2

Tabla 5.14: Identificación no observable Horneado

CONCLUSIONES

En esta tesis se ha tratado el tema de identificación del comportamiento observable y no observable para Sistemas de Eventos Discretos (DES) a partir de un grupo de señales E/S de comportamiento binario. La identificación se centra primero de la obtención de los modelos de autómatas e IPN de un DES emulado, donde se asume no conocer el modelo ni tampoco el comportamiento que el sistema tiene, unicamente se conocen sus señales E/S.

Luego se propone un procedimiento para identificar el comportamiento no-observable del sistema emulado. Con base en los resultados obtenidos se concluye que es posible identificar el comportamiento que no es medible de un sistema caja negra, a partir del análisis del comportamiento de las señales de E/S observables, con base en los Algoritmos elegidos en el desarrollo del trabajo. En el proceso de obtención del resultado principal de este trabajo, se generaron otras contribuciones; entre las cuales se tiene:

- Diseño de un DES con señales E/S de naturaleza binaria.
- Simulación y emulación de un DES sobre una tarjeta de desarrollo desde componentes de planta.
- Desarrollo de una herramienta software semi - automatizada para adquisición e identificación de DES.

El diseño de un DES consiste en la obtención de modelos de comportamiento lógico del controlador a partir de señales E/S de naturaleza binaria donde se escoge un proceso industrial real para discretizar sus posibles señales, para ello se realizó una identificación y clasificación de todos los elementos o dispositivos requeridos para la realización de cada etapa del proceso escogido.

El comportamiento lógico se diseñó con base a la distribución de la señales E/S requeridas en cada subsistema. A partir de estas señales se recurrió a modelar el comportamiento lógico de los subsistemas mediante diagramas de máquinas de estados, tomando las señales de entrada como las respuestas de sensores y eventos externos (señal de encendido y apagado) y las señales de salida como la respuesta que entrega el controlador hacia la planta del DES.

El modelado del comportamiento lógico por máquinas de estado dio la ventaja de usar los modelos diseñados, para ser usados en el proceso de simulación y emulación del DES, esto se debe a que las máquinas de estados están orientadas a señales binarias.

El proceso de simulación del comportamiento normal del DES se realiza en tarjetas de desarrollo que permite experimentar el comportamiento dinámico del sistema mediante la emulación física las señales E/S. Dicho proceso se realizó de manera empírica a causa de la casi nula información sobre cómo representar las dinámicas de los elementos que componen una planta.

A partir del la emulación del DES en la tarjeta de desarrollo se termina por concluir que para realizar un proceso de simulación del comportamiento lógico y dinámico de un DES de tipo planta - controlador es más fácil separar en estructuras comunes la planta y el controlador (comportamiento lógico, características de planta, condiciones para evolución del sistema) donde la evolución del sistema es dada por condiciones de eventos (estado de sensores y actuadores) y no por temporizadores.

La aplicación software desarrollada permite adquirir, procesar, automatizar e identificar los modelos de autómata e IPN característicos de los subsistema del DES obteniendo solamente el comportamiento observable.

Al contemplar las limitaciones que los algoritmos generadores de modelos tienen, se hace necesario conocer el comportamiento no observable del DES. Para satisfacer esto se planteó una prueba de detectabilidad definiendo un algoritmo en seudo-código basado en IPN, el cual permite detectar y clasificar los elementos que hacen parte del comportamiento no observable. Mediante la clasificación de dichos elementos se realiza la identificación no observable de un DES. La identificación no observable basada en el algoritmo diseñado, resulta ser útil para la detección de fallo y detección de comportamientos no deseados en un DES.

Bibliografía

[1] S. Klein, L. Litz, and F. Felgner, "Identification of discrete event systems for fault detection purposes," 2005. [Online]. Available: https://pdfs.semanticscholar.org/d890/71e50daf39c00cb664ea835daf875acb58d2.pdf

[2] S. Shu and F. Lin, "Generalized detectability for discrete event systems," *Systems Control Letters*, vol. 60, no. 5, pp. 310 – 317, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167691111000223

[3] A. Guasch and U. P. de Catalunya, *Modelado y simulación: aplicación a procesos logísticos de fabricación y servicios*, ser. Politext Series. UPC, S.L., Edicions, 2002.

[4] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, S. S. B. Media, Ed. Springer US, 2008.

[5] A. Giua, "Supervisory control of petri nets with language specifications," in *Control of Discrete-Event Systems*, ser. Lecture Notes in Control and Information Sciences, C. Seatzu, M. Silva, and J. H. van Schuppen, Eds., vol. 433. Springer London, 2013, pp. 235–255. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-4276-8{_}12

[6] M. Muñoz, A. Correcher, E. García, and F. Morant, "Identification of stochastic timed discrete event systems with st-ipn," *Mathematical Problems in Engineering*, vol. 0, no. 00, p. 0, 2014.

[7] J. Saives, G. Faraut, and J. J. Lesage, "Identification of discrete event systems unobservable behaviour by petri nets using language projections," in *Control Conference (ECC), 2015 European*, July 2015, pp. 464–471.

[8] C. Girault and R. Valk, *Petri nets for systems engineering - a guide to modeling, verification, and applications*. Springer, 2003.

[9] A. citations More» T. Murata, "Petri nets - properties, analysis and applications„" *IEEE*, pp. 541 – 580, Apr 1989.

[10] M. Dotoli, M. P. Fanti, A. M. Mangini, and W. Ukovich, "On-line identification of petri nets with unobservable transitions," in *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, may 2008, pp. 449–454.

[11] K. H. Rueda, "Fault diagnosis using petri nets. a case study." Tenth LACCEI Latin American and Caribbean Conference - International Competition of Student Posters and Paper, 2012.

[12] L. P. J. VEELENTURF, "An automata-theoretical aproach to developing learning neural networks," *Cybernetics and Systems*, vol. 12, no. 1-2, pp. 179–202, 1981. [Online]. Available: http://dx.doi.org/10.1080/01969728108927670

[13] A. P. Estrada-Vargas, E. López-Mellado, and J. J. Lesage, "Identification of partially observable discrete event manufacturing systems," in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2013, pp. 1–7.

[14] R. Brena, *Automatas y Lenguajes Un Enfoque de Diseño,*, T. de Monterrey., Ed. Editorial, 2003.

[15] S. Klein, L. Litz, and J.-J. Lesage, "Fault detection of discrete event systems using an identification approach," in *16th IFAC world Congress*, Praha, Czech Republic, 2005. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00347763

[16] M. Roth, J.-J. Lesage, and L. Litz, "An fdi method for manufacturing systems based on an identified model," in *13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM2009)*, Moscow, Russia, 2009, p. Paper 58. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00424385

[17] M. Roth, J. J. Lesage, and L. Litz, "Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems," in *American Control Conference (ACC), 2010*, 2010, pp. 2601–2606.

[18] D. E. Jarvis, "An identification technique for timed event systems," *10th International Workshop on Discrete Event Systems*, vol. 10, pp. 181–186, 2010. [Online]. Available: http://www.ifac-papersonline.net/Detailed/42925.html

[19] H. Hu, M. Zhou, and Z. Li, "Supervisor optimization for deadlock resolution in automated manufacturing systems with petri nets," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 794–804, oct 2011.

[20] A. P. Estrada-Vargas, E. López-Mellado, and J.-J. Lesage, "A comparative analysis of recent identification approaches for discrete-event systems," *Mathematical Problems in Engineering*, vol. 2010, pp. 1–22, 2010. [Online]. Available: http://www.hindawi.com/journals/mpe/2010/453254.html

[21] M. E. Meda-Campana and E. Lopez-Mellado, "Required event sequences for identification of discrete event systems," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 4, 2003, pp. 3778–3783 vol.4.

[22] M. Meda-Campana and E. Lopez-Mellado, "Identification of concurrent discrete event system using petri nets." in *Proceedings of the 17th IMACS World Congress on Computational and Applied Mathematics*, July 2005, pp. 11–15.

[23] A. Giua and C. Seatzu, "Identification of free-labeled petri nets via integer programming," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, 2005, pp. 7639–7644.

[24] M. Dotoli, M. P. Fanti, and A. M. Mangini, "Real time identification of discrete event systems using petri nets," *Automatica*, vol. 44, no. 5, pp. 1209 – 1219, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109807004542

[25] M. Dotoli, M. P. Fanti, A. M. Mangini, and W. Ukovich, "On-line fault detection in discrete event systems by petri nets and integer linear programming," *Automatica*, vol. 45, no. 11, pp. 2665–2672, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109809003720

[26] ——, "Identification of the unobservable behaviour of industrial automation systems by petri nets," *Control Engineering Practice*, vol. 19, no. 9, pp. 958–966, 2011, special Section: DCDS09. The 2nd IFAC Workshop on Dependable Control of Discrete Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066110002017

[27] M. P. Cabasino, A. Giua, and C. Seatzu, "Identification of deterministic petri nets," in *Discrete Event Systems, 2006 8th International Workshop on*, 2006, pp. 325–331.

[28] ——, "Identification of petri nets from knowledge of their language," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 447–474, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10626-007-0025-0

[29] M. Cabasino, A. Giua, C. Mahulea, and C. Seatzu, "On decentralized observability of discrete event systems," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, Dec 2011, pp. 378–384.

[30] A. P. Estrada-Vargas, "Black-box identification of automated discrete event systems." Ph.D. dissertation, NORMALE SUPERIEURE DE CACHAN, 2013.

[31] C. Keroglou and C. N. Hadjicostis, "Detectability in stochastic discrete event systems," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 27–32, 2014, 12th IFAC International Workshop on Discrete Event Systems (2014). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015373766

[32] ——, "Detectability in stochastic discrete event systems," *Systems Control Letters*, vol. 84, pp. 21–26, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167691115001516

[33] S. Shu, F. Lin, and H. Ying, "Detectability of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 12, pp. 2356–2359, Dec 2007.

[34] S. Shu, F. Lin, H. Ying, and X. Chen, "State estimation and detectability of probabilistic discrete event systems," *Automatica*, vol. 44, no. 12, pp. 3054

– 3060, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109808003257

[35] A. S.-S. Luis Aguirre-Salas, "Sequence-detectability analysis of interpreted petri nets under partial state observations," *IEEE*, 2009.

[36] M. P. Cabasino, A. Giua, C. N. Hadjicostis, and C. Seatzu, "Fault model identification and synthesis in petri nets," *Discrete Event Dynamic Systems*, vol. 25, no. 3, pp. 419–440, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10626-014-0190-x

[37] A. P. Estrada-Vargas, E. López-Mellado, and J.-J. Lesage, "Input/output identification of controlled discrete manufacturing systems," *International Journal of Systems Science*, vol. 45, no. 3, pp. 456–471, 2014. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00207721.2012.724098

[38] Unidad de Planeación Minero Energética, "Ahorro De Energía En La Industria Cerámica," pp. 1–28, 2006.

[39] I. D. P. CERÁMICA, "Ceramic tiles - manufacturing process and technical classifications." [Online]. Available: www3.ipc.org.es/dms/pdfs/Baldosas_ceramicas_clasificacion_comercial/Manufacturing_process/4-4-1-B%20DOC6%20ing-DEF_vPDF.pdf

[40] L. E. Holloway and B. H. Krogh, "Synthesis of feedback control logic for a class of controlled petri nets," *IEEE Transactions on Automatic Control*, vol. 35, no. 5, pp. 514–523, May 1990.

[41] R. S. Stefan Duffner, "Qfsm." [Online]. Available: http://qfsm.sourceforge.net/

[42] L. D. Murillo, "Redes de petri: Modelado e implementación de algoritmos para autómatas programables," *Tecnología en Marcha*, vol. 21, no. 4, pp. 102–125, 2008. [Online]. Available: https://dialnet.unirioja.es/descarga/articulo/4835618.pdf

[43] S. Microsystems, "Netbeans ide," 2000–2010. [Online]. Available: https://netbeans.org/

[44] Microsoft, "Microsoft visual studio community 2015," 2015. [Online]. Available: https://www.microsoft.com/es-co/download/details.aspx?id=48146

[45] H. L. Colin Laplace, Mike Berg, "Bloodshed dev-c++." [Online]. Available: http://www.bloodshed.net/devcpp.html

[46] E. Foundation, "Eclipse sdk." [Online]. Available: http://download.eclipse.org/eclipse/downloads/

[47] G. Inc, "Android studio." [Online]. Available: https://developer.android.com/studio/index.html?hl=es-419

[48] "Code::blocks[aplicación pc]." [Online]. Available: http://www.codeblocks.org/downloads

[49] Q. Company, “Qt creator.” [Online]. Available: https://www.qt.io/ide/

[50] J. P. Fernández, *UF1843: Aplicación de técnicas de usabilidad y accesibilidad en el entorno cliente*, 1st ed. ic editorial, 2014.

[51] A. B. J. E. E. G. Y. H. Yahoo, “Graphviz.” [Online]. Available: http://www.graphviz.org/Download.php