ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"SISTEMA DIDÁCTICO PARA AUTÓMATAS"

INFORME DE MATERIA INTEGRADORA

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO EN ELECTRICIDAD

ESPECIALIZACIÓN EN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

CARLOS LEONARDO VILLAFUERTE YAGUAL

EMIL RICARDO SOLÍS ACOSTA

GUAYAQUIL – ECUADOR

AÑO: 2016

AGRADECIMIENTO

Agradezco a Dios por todo lo que me ha dado, por darme la gracia de seguir compartiendo cada día de mi vida y la fortaleza para no desfallecer perseverando en todo lo que me propongo.

A mis padres que con su apoyo y cuidados he logrado tener una vida placentera, por sus consejos que se encuentran muy vigentes en mí.

A mis hermanos que con sus cuidados supieron verme en mi niñez.

DEDICATORIA

A Dios, a mis padres y al Ecuador entero, que con el fruto de este trabajo produzca el crecimiento de esta gran nación.

Emil Ricardo Solís Acosta.

DEDICATORIA

El presente proyecto lo dedico principalmente a mis padres por apoyarme sin reparos desde el inicio de mi carrera y durante los días pesados y sacrificados de mi carrera universitaria y al resto de personas que me han apoyado durante todo este proceso brindándome toda la fortaleza necesaria para nunca desistir de mis metas.

Carlos Leonardo Villafuerte Yagual

TRIBUNAL DE EVALUACIÓN

Ph.D. Wilton Agila	**Ing. Alberto Larco**
PROFESOR EVALUADOR	PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

.....

Carlos Leonardo Villafuerte Yagual

.....

Emil Ricardo Solís Acosta

RESUMEN

Este proyecto surge debido a la poca afluencia que hay de sistemas de automatización que solo se centren en la programación en lenguaje de contactos independiente del fabricante, ya que muchos de los tipos de software que abundan en el mercado necesitan del equipo de hardware en sí para lograr ejecutarse, entre otras problemáticas como las especificaciones del equipo etc.

Es por ello que se creó un software con fines didácticos que sirve de alternativa para la iniciación en la programación en lenguaje de contactos independiente de algún equipo, marca o fabricante que emulara un autómata tanto virtual como físicamente. Este software es un prototipo, pero ya puede ser introducido en prácticas en colegios técnicos o dirigido a principiantes que se quieren iniciar en la programación de autómatas. El sistema es creado en lenguaje Java junto con la plataforma de Hardware libre Arduino con el fin de poder controlar ciertos procesos dada una configuración electrónica ya establecida para la aplicación. Se utilizan técnicas de programación ya conocidas para hacer más simple y rápida la capacidad de respuesta.

En la etapa final del proyecto se implementan las salidas con una secuencia definida.

ÍNDICE GENERAL

CAPÍTULO 1

1. PROBLEMA, OBJETIVO Y ALCANCE DEL PROYECTO.

Existe demasiada información sobre autómatas en la literatura especializada; algo que tienen en común estos artículos es que todos se centran en su parte técnica, funcional y estructural, por lo que es muy común dejar de lado el ámbito de la programación de estos equipos, lo que lleva a buscar una solución a dicho problema con un objetivo específico y su respectivo alcance.

1.1 Descripción del problema

Desde que empezó el uso de autómatas en las operaciones industriales, se hizo necesario emplear software especializado en cada sistema donde se requería controlar tanto la secuencia de ciertos procesos como para el monitoreo de los mismos. Por otro lado, en la industria se tiene una variedad de sistemas conformados por equipos de distintos fabricantes, los cuales trabajan independientemente pero se necesita que trabajen de manera coordinada para mejorar el rendimiento de los procesos; este y muchos otros casos se dan hoy en día. Esto implica que la intervención de un software en la automatización de procesos es imprescindible para el desarrollo más eficaz de la adquisición automática de datos, sistemas de control, pruebas y mediciones.

Las computadoras suelen controlar un equipo de prueba, el cual es programado para simular un mando manual en alguna aplicación, lo que significa que un programa informático es el encargado de coordinar y controlar cada una de las operaciones.

Si bien cada fabricante tiene su software y este generalmente ofrece manuales de programación, estos no suelen ser útiles como herramienta pedagógica en la que el usuario aprenda a programar un autómata con un enfoque único en programación, sin que se esté ligado a un equipo especifico; así mismo muchos de estos programas necesitan del equipo en

cuestión para poder funcionar, reduciendo notablemente el área de aprendizaje al estudiante. Bajo este contexto, se propone el desarrollo de una aplicación de software que permite al estudiante un aprendizaje fácil en el diseño e implementación de secuencias de control para aplicación en PLC.

1.2 Objetivo

El objetivo de este proyecto es el desarrollo de un software didáctico que pueda ser de utilidad a los estudiantes de ingeniería que quieren iniciar en la programación de autómatas, de esta manera se enfrente a todas las dificultades o complejidades que se pueden dar al desarrollar programas para autómatas y afrontar cualquier caso real que se presente en un entorno industrial.

1.3 Alcance

El software a desarrollar en esta propuesta no está enfocado en la programación de un autómata específico, sino que es una herramienta pedagógica útil para la programación en general de estos dispositivos. Es así, que al inicio del segundo capítulo presenta una introducción resumida sobre autómatas, haciendo hincapié en los conocimientos básicos para que el estudiante pueda abordar el tema de la programación de autómatas mediante el lenguaje de contactos.

El proyecto pretende que el software tenga como base la creación de secuencias debidamente programadas, tales como las que se dan en procesos típicos industrializados; es así que se define el alcance del proyecto como un sistema computacional destinado a ser una herramienta útil para aprender a programar autómatas sin la necesidad de conocimientos específicos ligados al equipo de un fabricante. Cabe destacar que el software es de fácil ejecución en cualquier plataforma que tenga previamente instalado la máquina virtual de Java, por lo que el software estará

desarrollado bajo el lenguaje Java, aplicando la Programación Orientada a Objetos.

El presente informe no se enfoca en temas computacionales tales como, la introducción al lenguaje Java o desarrollo de diagramas UML, se tratará de explicar cómo se interpreta en un algoritmo cada elemento disponible, como los contactos, bobinas, temporizadores y contadores, etc.

En lo relacionado a las unidades de entradas y salidas de un autómata, dado que no suelen ser elementos explicados de una manera entendible en los distintos manuales que se encuentran en la red, el software propuesto tendrá una conexión con una plataforma de hardware libre que actuará como dispositivo de adquisición de datos, lo que permitirá al estudiante observar visualmente las señales de entradas y salidas. Es así que el estudiante podrá desarrollar una determinada secuencia y observarla de manera física, dichas salidas estarán limitadas por el hardware en cuestión.

Por tanto, el software no solo ayudara al estudiante a emular un autómata virtual sino que también podrá llevarlo a un nivel técnico haciendo controles, desde los más básicos como una simple secuencia de encendido de leds hasta controles avanzados típicos en la industria. Cabe indicar que el software no ayudará a resolver problemas concretos de ingeniería, sino solamente enseñará a programar basándose en modelos simplificados de automatismos complejos, pues la idea no es controlar un proceso detalladamente sino desarrollar un programa capaz de entender su lógica y manejarlo de manera eficaz.

Es necesario decir que el software a desarrollar prescindirá de una serie de herramientas que los autómatas modernos de hoy utilizan, así solo se utilizara contactos, bobinas, temporizadores y contadores, tal como se solía hacer con sistemas electromecánicos; dichos sistemas no eran impedimento para elaborar complejos sistemas de control. Esto ayudará al estudiante a comprobar que con los elementos más simples pero esenciales en un lenguaje de contactos, es posible configurar cualquier sistema lógico de

control pasando por alto herramientas y funciones específicas que nos proveen diversos fabricantes.

Finalmente, dada la integración del software con la plataforma física, el estudiante podrá llevar a cabo un control en tiempo real de algún proceso; esto sugiere envío y recepción de señales por parte del software.

1.4 Marco Teórico

Se puede definir a un autómata como un ordenador, más claramente como un dispositivo electrónico digital compuesto de chips y circuitos; consta de un microprocesador en el que se ejecutan los programas, memoria donde se guarda la información y buses por donde circulan datos a manera de información. Es muy similar a una computadora personal, mas este no cuenta con teclados, ratón, disqueteras, altavoces y otros periféricos, en su lugar cuenta con unidades de entrada y salida los cuales son los elementos más característicos de un autómata.

Estas unidades constan de bornes en los que se puede conectar toda clase de dispositivos, además las señales recibidas son adaptadas para poder ser manejadas por la circuitería interna. Estas unidades están preparadas para recibir información del entorno y para actuar sobre el mismo por medio de dispositivos conectados a las salidas.

Una definición más clara que se da al autómata programable, es que es un dispositivo electrónico usado como controlador lógico en la automatización de procesos en la industria, el cual es capaz de recibir, almacenar y procesar información del sistema a gobernar para conocer su estado. Este sistema le ayuda a tomar decisiones, activando dispositivos es capaz de actuar sobre el proceso.

Al autómata programable también se lo conoce como PLC (Controlador Lógico Programable), una de sus muchas propiedades es ejecutar una serie de instrucciones que conforman lo que se denomina un programa, el mismo que se puede ser modificado cuantas veces se requiera. Esta característica

permite una gran versatilidad y una gran capacidad de adaptación. Otra diferencia que tienen los autómatas con ordenadores comunes es que está preparado para soportar condiciones extremas de humedad, vibración, entre otras, dado que su principal función se da en entornos industriales.

Fue hasta que aparecieron los transistores que se empezó a utilizar circuitos electrónicos para cumplir las funciones de los controladores de procesos, es así que los automatismos dejaron el uso de relés que era lo que hasta la época se usaba, para usar resistencias, diodos y los muy populares transistores, hasta que se hizo notable la aparición de los circuitos integrados. Una de la ventajas de la revolución de la electrónica fue la disminución de volumen haciendo obsoletos los aparatosos relés, además el uso de componentes estáticos provocaban la disminución de un mantenimiento preventivo el cual era necesario, esto se debía a que las partes móviles no sufrían desgaste.

Muchas veces hemos escuchado sobre la velocidad de respuesta de ciertos dispositivos, pues esto se hizo notorio con la llegada de los elementos electrónicos, pero había un problema, ya que había que diseñar un nuevo sistema de control basado en circuitería electrónica siempre que se requería cambiar el proceso de producción.

Un gran cambio se produjo cuando el fabricante de automóviles General Motors realizo un concurso en el cual se convocaba a todo tipo de inventores e ingenieros para crear un controlador que fuese más fácil de programar y por sobre todo, que esté adaptado a entornos industriales, que fuese resistente, a la vez que tenga una larga vida útil. Es así como nació el primer autómata programable y que fue producido comercialmente, el Modicon 084 (Modular Digital Controller).

El control distribuido no se hizo posible hasta inicios de los años 70 cuando los autómatas tenían la posibilidad de establecer comunicaciones, luego de esto aparecieron los microprocesadores lo que derivó en la incorporación definitiva de los autómatas en los procesos industriales. Estos microcontroladores dieron la pauta para desarrollar controles más

sofisticados, además de agregar grandes ventajas en calidad de recursos como memoria y capacidades enormes de cálculo. En definitiva la adaptabilidad de los autómatas y el disponer de un aparato reutilizable para cualquier proceso permitían una mejora eficaz en las frecuentes modificaciones de los procesos operacionales que se daban y se dan actualmente en las industrias.

CAPÍTULO 2

2. SOFTWARE DIDÁCTICO PROPUESTO

La información siguiente se dedica a detallar el sistema ya planteado. Se decidió utilizar el lenguaje Java ya que aparte de ser el más utilizado en materia de software es también una de los más compatibles con diferentes tecnologías, tanto así que nos permite conectarnos con plataformas de hardware libre como lo es Arduino. La parte grafica será similar a otros software antes vistos, dado que todos suelen tener las mismas funciones es inevitable la similitud en lo que corresponde a la interfaz. El sistema cuenta con un contador y un temporizador para ejecutar la pruebas tanto virtual como de manera física para dar más realce a la emulación de un autómata. Hay varias técnicas de configuración de parámetros de elementos que se irán detallando a lo largo del documento. Las salidas físicas se observaran por medio de leds que ilustrarán la presencia y ausencia de señales.

2.1 Herramienta de Software Base

Para la realización del proyecto, inicialmente se tuvo cuidado en escoger la IDE adecuada para no tener problemas de integración con la parte física. Cuando se habla de IDE se habla del programa que facilitará digitar y compilar código para el sistema. Se eligió Netbeans un entorno de desarrollo libre y que fue hecho principalmente para el lenguaje de programación Java, esta IDE soporta todos los tipos de aplicación Java así mismo se requieren las últimas versiones no tanto para lograr la integración con Arduino sino para un fácil acoplamiento con el lenguaje en cuestión del cual si se necesitan las últimas actualizaciones. En este caso se eligió Netbeans 7.4, véase la figura 2.1.

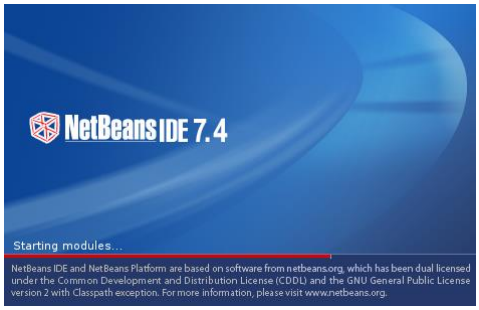


Fig. 2.1 NetBeans IDE 7.4**.**

Para realizar programas escritos en Lenguaje Java necesitamos algo fundamental como lo es el JDK (Kit de desarrollo Java), que no es más que un conjunto de herramientas de desarrollo para lenguaje Java con el que podremos compilar y ejecutar código. Una mejor descripción es que el JDK consta de una serie de programas y librerías que permiten desarrollar programas en dicho lenguaje.

Así mismo se necesita de una versión actualizada del JDK en este caso el JDK 6.0, ya que la integración con Arduino depende de las últimas actualizaciones en la que se incorporaron nuevas librerías destinadas a la conexión con nuevas tecnologías de hardware libre.

2.2 Herramienta de Hardware Base

Arduino es una plataforma de hardware libre, la cual se basa en una placa en la cual hay un microcontrolador en conjunto con un entorno de desarrollo; dicho microcontrolador es el Atmel AVR de 8 bits y también tiene puertos de entrada y salida todas denotadas en la placa así como entrada de alimentación de 5 volteos. El entorno de desarrollo se basa en el lenguaje de programación Processing dado que el microcontrolador es programable con un lenguaje de alto nivel, este mismo se encarga de realizar los procesos

lógicos y gestionar los recursos de los componentes externos conectados al dispositivo.

Para programar el dispositivo se necesita de una IDE propia de Arduino, llamada con el mismo nombre; en el desarrollo de este proyecto se utiliza la versión ARDUINO UNO 6.1.5, véase la figura 2.2.

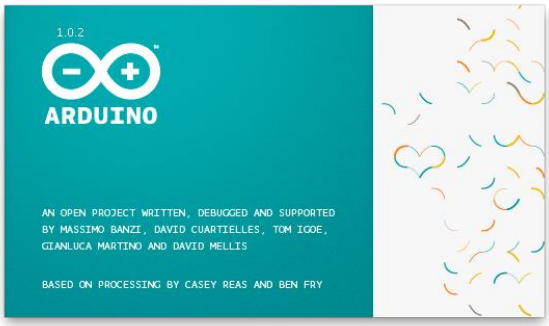


Fig. 2.2 ARDUINO UNO 6.1.5.

Arduino cuenta con varios modelos de placas a elegir, dependiendo del uso que se le quiera dar, cada una de estas con características particulares pero todas tienen sus generalidades; para este proyecto se utiliza la placa Arduino UNO, véase en la figura 2.3 las características de la tarjeta electrónica.

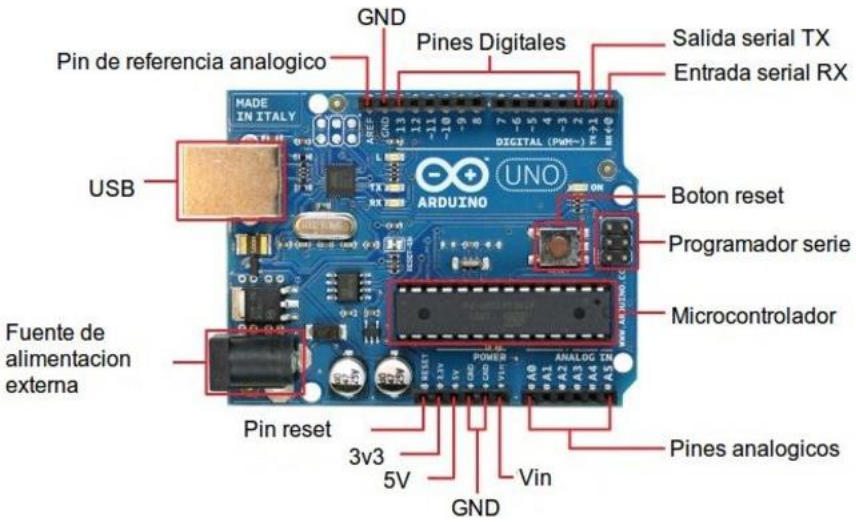


Fig. 2.3 Tarjeta electrónica de ARDUINO UNO.

2.3 Tecnologías Cableadas

Cuando se refiere a tecnologías cableadas se habla de neumática y electromecánica la cual surgió en la naciente industria del automóvil lo que dio un importante avance en el desarrollo y utilización de automatismos. Los sistemas neumáticos sugieren la ayuda de secuenciadores y temporizadores los cuales eran capaces de realizar y ejecutar operaciones que permitían la automatización de procesos, lo que hacían era ejecutar programas lo cual era una solución acorde con la tecnología aplicada ya que eran en gran parte cilindros neumáticos los que gobernaban y accionaban la maquinaria.

Básicamente esta manera de programar consistía en la interconexión de relés electromagnéticos los cuales permitían crear un secuencia lógica capaz de controlar un proceso por más complejo que sea, estos dispositivos electromagnéticos hacían posible la conmutación de grandes corrientes, lo que permitía activar una variedad de aparatos eléctricos; es así como en grandes plantas automotrices se instalaban grandes armarios en paralelo con las líneas de montaje, con el fin de elaborar el sistema que controlaba el proceso de fabricación, ya que dentro de estos armarios se utilizaba circuitería con relés.

En muchas ocasiones se debían realizar conexiones que incluían el uso de una cantidad exorbitante de relés, lo que claramente suponía un esfuerzo enorme de diseño e instalación.

Los fallos eran muy difíciles de detectar aun visualmente no se solía denotar las fallas lo que lo hacía una labor muy compleja; por tanto se hacia el uso de contactores y relés en sectores cumbres para facilitar la ubicación de las fallas.

Debido al desgaste de los elementos electromecánicos solo se sugería un número muy limitado de maniobras, ya que esto implicaba un continuo mantenimiento y sustitución de los elementos.

Siempre que se requería cambiar el proceso de producción, había también que cambiar todo el sistema de control; lo que no era eficiente en términos de productividad.

2.4 Descripción del Código Simple

La descripción gráfica de la interacción de los programas se puede apreciar en la figura 2.4.

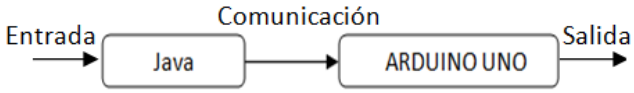


Fig. 2.4 Interacción de los programas

Para la descripción del código se encuentran varios métodos o procedimientos que ayudan a entender de una manera más clara la totalidad del código; así mismo el código cuenta con líneas de comentarios dentro del programa ayudar en su comprensión. El código tuvo que ser varias veces actualizado a medida que se agregaban nuevos elementos así mismo tuvo que ser modificado dado que muchas veces se descubrían errores a medida que el proyecto avanzaba.

La descripción no se enfoca en los métodos de programación, ni en líneas de código Java, sino en la interpretación del código en términos de automatización; es así que se describirá por partes y solo las partes fundamentales que dan vida al proceso.

Se muestra en la figura 2.5 las declaraciones de las variables que contiene el código.

```
public class nuevo extends javax.swing.JFrame {


    JLabel uno[][]=new JLabel[4][5];
    JTextField dos[][]=new JTextField[4][5];
    Salidas Ventana = new Salidas();
    int estado[][]=new int[4][5];
    int aux[]=new int[4];
    int estado2[]=new int[4];
    int posicion[]= new int[4];
    int posicionEsT[]=new int[4];
    int temp=0, tiempo=0, conteo=0;
    int salvando=0, salvando2=0;
    int contador=0;
    int tempA[]= new int[4];
    int tempContador[]= new int[4];
    String cadena="";
    Timer t;

     public void ContactosBobinas(int a, String name, String nameImage){...30 lines }
     public void ValoresABobinas(int i){...52 lines }
     public void Linea(int i, String name, String nameImage){...24 lines }
     public void ContactosANumeros(int i, String name, String nameImage){...14 lines }
     public void GuardaTemporal(){...5 lines }
     public void ContactosIguales(String name, String nameImage, int guar, int guar2){...23 lines }
     public void inicializacion(){...26 lines }
     public void FuncionInicio(){...36 lines }
     public void Focos(){...20 lines }
```

Fig. 2.5 Primera vista del código.

La figura 2.6 muestra porción del código que grafica gran parte de nuestro panel de pruebas, tales como las casillas de imagen y de texto. Hay partes del panel que se graficaron con la ayuda de la paleta de Java por tanto ese código es generado por el mismo programa por tanto no es mostrado; basta con aumentar el ciclo de repetición en el código para ampliar el número de casillas.

```
public nuevo() {
    initComponents();

    for(int k=0; k<=3; k++)
        for(int l=0; l<=4; l++){
          uno[k][l]= new JLabel();
          dos[k][l]=new JTextField();
        }
    int a=0, b=0;//CREACION DE TODAS LAS CASILLAS
        for(int i = 30; b!=5; i+=100){
            for(int j = 30; a!=4; j+=100){

                uno[a][b].setBounds(i,j ,100,100);
                dos[a][b].setBounds(i,j+70 ,100,30);
                uno[a][b].setBorder(BorderFactory.createLineBorder(new java.awt.Color(10, 0, 0)));
                dos[a][b].setBorder(BorderFactory.createLineBorder(new java.awt.Color(10, 0, 0)));
                uno[a][b].setTransferHandler(new TransferHandler("icon"));
                dos[a][b].setHorizontalAlignment(javax.swing.JTextField.CENTER);
                jPanel2.add(dos[a][b]);
                jPanel2.add(uno[a][b]);

                a++;}
           b++;
           a=0;
        }
```

Fig. 2.6 Bloque creador de las casillas de texto e imagen del panel.

Los gráficos vistos en el programa, tales como contactos y bobinas, se insertaron como imágenes que son traducidas a texto, para lograr ser comprendidos por el sistema; como se lo muestra en la figura 2.7.

```
MouseListener m1 = new MouseListener() {//EVENTO DE COPIA ARRASTRANDO
 public void mouseClicked(MouseEvent e){}
 public void mouseReleased(MouseEvent e){}
 public void mouseEntered(MouseEvent e){}
 public void mouseExited(MouseEvent e){}
 public void mousePressed(MouseEvent e){

      JComponent jc = (JComponent)e.getSource();
      TransferHandler th = jc.getTransferHandler();
      th.exportAsDrag(jc, e, TransferHandler.COPY);

}};
```

Fig. 2.7 Código que permite el arrastre de imágenes.

La figura 2.8 muestra el grupo de código conocido como “método”, el cual permite el arrastre de imágenes hacia las casillas. La figura 2.29 complementa este proceso al adherir las imágenes a las casillas.

```
jLabel1.addMouseListener(m1);
jLabel2.addMouseListener(m1);
jLabel7.addMouseListener(m1);
jLabel8.addMouseListener(m1);
jLabel9.addMouseListener(m1);
jLabel10.addMouseListener(m1);
jButton1.addActionListener(n2);
jLabel1.setTransferHandler(new TransferHandler("icon"));//recibe la capacidad de traslado y recibir pegado
jLabel2.setTransferHandler(new TransferHandler("icon"));
jLabel7.setTransferHandler(new TransferHandler("icon"));
jLabel8.setTransferHandler(new TransferHandler("icon"));
jLabel9.setTransferHandler(new TransferHandler("icon"));
jLabel10.setTransferHandler(new TransferHandler("icon"));
```

Fig. 2.8 Capacidad de traslado y pegado de imágenes.

```
public void inicializacion(){
                //==================  INICIALIZACION DE ELEMENTOS  ======================================

      for (int i=0; i<4; i++){
           posicion[i]=3;//"0" BOBINAS ENCENDIDAS, "1" BOBINAS APAGADAS, "2" NO HAY BOBINA, "3" NO HAY ESTADO
      }
      for (int i=0; i<4; i++){
         for (int j=0; j<4; j++){
            estado[i][j]=0;//INICIALIZANDO
         }
      }
      for (int i=0; i<4; i++){
          aux[i]=1;
      }
      for (int i=0; i<4; i++){
            estado2[i]=0;//INICIALIZANDO
      }
      for (int i=0; i<4; i++){
           posicionEsT[i]=0;//"0" BOBINAS ENCENDIDAS, "1" BOBINAS APAGADAS, "2" NO HAY BOBINA, "3" NO HAY ESTADO
      }
      for (int i=0; i<4; i++){
          tempContador[i]=1;
      }
```

Fig. 2.9 Inicialización de variables fundamentales del proceso.

El código de la figura 2.9 muestra la inicialización de varios arreglos, en el que cada uno juega un papel vital en el proceso. Tenemos un arreglo para representar las salidas, otro que representa los casilleros de imagen, uno que representa los casilleros de texto; tenemos arreglos auxiliares que ayudan en la secuencia de revisión cuando se enciende o apaga una bobina en determinado momento.

```
public void ContactosBobinas(int a, String name, String nameImage){
              for (int k=0; a<4; k++){//COMPARACION DE BOBINA TRAS BOBINA
                   for (int l=0; l<4; l++){ //RECORRIDO EN LA FILA O LINEA
                       if ((dos[k][l].getText().equals(dos[a][4].getText()))&&    (  (((posicion[a]==0))&&(tempA[a]!=posicion[a]))

                             name=String.valueOf(uno[k][l].getIcon());
                             nameImage=(String)name.substring(name.lastIndexOf("/")+1);

                             if (nameImage.equals("contactoOpen.png") ){
                                uno[k][l].setIcon(new ImageIcon(getClass().getResource("/contactoClosed.jpg")));   }
                             else
                                if (nameImage.equals("contactoClosed.jpg")){
                                    uno[k][l].setIcon(new ImageIcon(getClass().getResource("/contactoOpen.png")));}
                                else
                                    if (nameImage.equals("ParaleloAbierto.png") ){
                                        //JOptionPane.showMessageDialog(null, posicion[a]+ " "+ tempContador[a]);
                                        uno[k][l].setIcon(new ImageIcon(getClass().getResource("/ParaleloCerrado.png"))); }
                                    else{
                                        uno[k][l].setIcon(new ImageIcon(getClass().getResource("/ParaleloAbierto.png")));}

                       }
                   }
                   if (k==3){
                       k=-1;
                       a=a+1;
                   }
               }//(int k=0; a<4; k++)   FIN DEL FOR QUE RECORRE LAS BOBINAS


}
```

Fig. 2.10 Código perteneciente a los contactos de las bobinas.

La figura 2.10 pertenece al código en el que una bobina reconoce a sus contactos en el panel, para así tomar la decisión de abrir o cerrar sus contactos, dependiendo del estado de la bobina.

La figura 2.11 muestra el código que traduce las imágenes colocadas en las casillas a números, que podrán ser manejados por el programa para efectos de comparación y validación. Los contactos abiertos son representados por un número; así mismo los contactos cerrados tienen su representación numérica. Es obvio que estos contactos al cambiar de estado, su representación numérica también cambia.

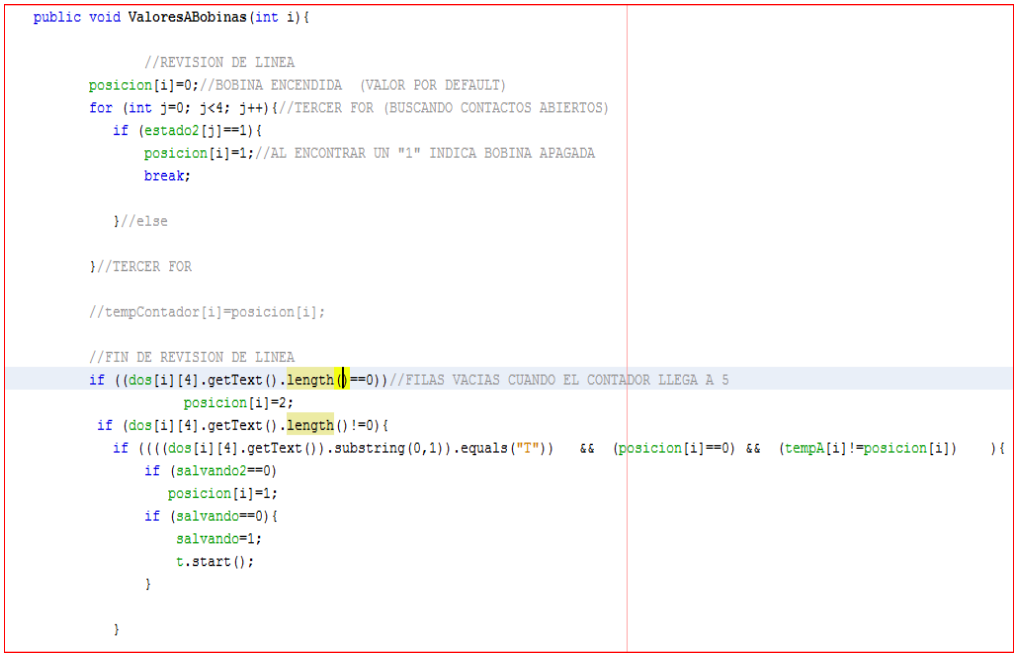
```
public void ContactosANumeros(int i, String name, String nameImage){
                for (int j=0; j<4; j++){//FOR SECUNDARIO(VALORES DE LA FILA CORRESPONDIENTE)

               if ((dos[i][j].getText().length())!=0){
                   estado[i][j]=0;
                   name=String.valueOf(uno[i][j].getIcon());
                   nameImage=(String)name.substring(name.lastIndexOf("/")+1);
                   if ((nameImage.equals("contactoOpen.png") )||(nameImage.equals("ParaleloAbierto.png"))){
                       estado[i][j]=1;//PARA CONTACTOS ABIERTOS EL ESTADO ES "1"
                   }
               }
             }//FOR SECUNDARIO

}
```

Fig. 2.11 Código para la representación numérica de los contactos.

```
public void Linea(int i, String name, String nameImage){
      for (int j=0; j<4; j++){//TERCER FOR
          estado2[j]=0;//INICIALIZANDO CADA VEX QUE ENTRA
          if (estado[i][j]==1){
              estado2[j]=1;
             // posicion[i]=1;//AL ENCONTRAR UN "1" INDICA BOBINA APAGADA
              if (i<2)//PROVISIONAL DEBIDO AL TAMAÑO DEL CASILLERO
              for (int m=1; m<3; m++){//
                  name=String.valueOf(uno[i+m][j].getIcon());
                  nameImage=(String)name.substring(name.lastIndexOf("/")+1);
                  if (nameImage.equals("ParaleloCerrado.png")){
                      estado2[j]=0;
                      break;
                  }

              }

          }//else

       }//TERCER FOR
```

Fig. 2.12 Código de transferencia de valores de los contactos.

La figura 2.12 muestra el código que transfiere a un arreglo los valores que determinaran si una bobina está abierta o cerrada; esta parte de código también limita el número de contactos en paralelo, que se pueden agregar a cualquier línea de código en lenguaje de bloques.



```
public void ValoresABobinas(int i){

              //REVISION DE LINEA
        posicion[i]=0;//BOBINA ENCENDIDA  (VALOR POR DEFAULT)
        for (int j=0; j<4; j++){//TERCER FOR (BUSCANDO CONTACTOS ABIERTOS)
           if (estado2[j]==1){
               posicion[i]=1;//AL ENCONTRAR UN "1" INDICA BOBINA APAGADA
               break;

           }//else

        }//TERCER FOR

        //tempContador[i]=posicion[i];

        //FIN DE REVISION DE LINEA
        if ((dos[i][4].getText().length()==0))//FILAS VACIAS CUANDO EL CONTADOR LLEGA A 5
                    posicion[i]=2;
         if (dos[i][4].getText().length()!=0){
           if ((((dos[i][4].getText()).substring(0,1)).equals("T"))   &&  (posicion[i]==0) &&  (tempA[i]!=posicion[i])    ){
               if (salvando2==0)
                  posicion[i]=1;
               if (salvando==0){
                   salvando=1;
                   t.start();
               }

           }
```

Fig. 2.13 Código que reconoce el estado de las bobinas.

La figura 2.13 nos muestra el código que finalmente decide que bobina está abierta o cerrada según la lectura de los arreglos que son representación del resto de elementos en las casillas.

La figura 2.14 cumple la misma función que le anterior con una variante, que esta sección de código también toma en cuenta el temporizador y contador en las casillas, alterando fuertemente los estados de las bobinas, además de recopilar información en cada iteración.

```
          //RECODIFICAR TODO ESTO (PRUEBAS DE ESCRITORIO..AUN TIENE SOLUCION//EL PROBLEMA SON SUS CONTACTOS)
         if (( ((dos[i][4].getText()).substring(0,1)).equals("C"))  ){
           tempA[i]=tempContador[i];
           tempContador[i]=posicion[i];
           if (((posicion[i]==0) && (tempContador[i]!=tempA[i])))
               contador= contador +1;
           if (contador==conteo){
                   posicion[i]=0;
                   contador=0;
           }
           else
              if(( contador>0)  &&   (contador<conteo)) {
                  posicion[i]=1;
                  tempA[i]=posicion[i];}
                 //JOptionPane.showMessageDialog(null, contador);
           //tempA[i]=posicion[i];
         }

        }//if (dos[i][4].getText().length()!=0)

}
```

Fig. 2.14 Código que reconoce temporizador y contador.

Todas estas secciones de códigos fueron puestas de manera estratégica con el fin de ejecutar de manera correcta y sin errores, las secuencias creadas por el estudiante; en cada ejecución se dan varias iteraciones, por tanto estas porciones de código son utilizadas varias veces en el transcurso del programa. En la siguiente imagen se muestra un procedimiento que recibe datos para ser utilizados por el programa, tales datos corresponden a la información que recibe el temporizador y contador; los datos ingresados deben ser números enteros, que serán manejados en el transcurso del proceso.

```
ActionListener retardo=new ActionListener(){

     public void actionPerformed(ActionEvent e){
        String LetraRetardo1="", LetraRetardo2="", Stiempo="";
        int indice=0;
              for(int l=0; l<=3; l++)
                if (e.getSource()==dos[l][4]){
                    indice=l;//SE GUARDA LA POSICION DE LA CASILLA EN LA QUE OCURRE EL EVENTO
                }
              LetraRetardo1=dos[indice][4].getText();
              LetraRetardo2=LetraRetardo1.substring(0,1);
              if (LetraRetardo2.equals("T"))  {//INICIO DEL IF
                 Stiempo=JOptionPane.showInputDialog("Ingrese el tiempo de retardo: ");
                 tiempo=Integer.parseInt(Stiempo);
              }
               if (LetraRetardo2.equals("C"))  {//INICIO DEL IF
                 Stiempo=JOptionPane.showInputDialog("Ingrese el conteo: ");
                 conteo=Integer.parseInt(Stiempo);
              }
```

Fig. 2.15 Código que recibe datos.

```
ActionListener temporizador = new ActionListener(){
   int minuto=0, segundo=0, segundos=0, segundosActuales=0, Va=0;
   public void actionPerformed(ActionEvent e){

               SimpleDateFormat fec =new SimpleDateFormat("HH:mm:ss");

                  cadena= String.valueOf(fec.format(new java.util.Date()));
                  minuto=Integer.parseInt(cadena.substring(3,5));
                  segundo=Integer.parseInt(cadena.substring(6, 8));
               if (Va==0){
                  segundos=((minuto*60)+segundo);
                  //tiempo=Integer.parseInt(Stiempo);
                  Va=1;
               }
               else
                 segundosActuales=((minuto*60)+segundo);
               if(segundosActuales==(segundos+ tiempo)){
                  //JOptionPane.showMessageDialog(null, tempA[0] + " " + posicion[0]);
                  //t.stop();
                  Va=0;
                  salvando=1;
                  salvando2=1;
                  FuncionInicio();
                  //Focos();
                  FuncionInicio();
                  //Focos();
                  FuncionInicio();
                  Focos();
                  t.stop();
                  salvando2=0;
                  salvando=0;
```

Fig. 2.16 Código para la temporización.

La figura 2.15 muestra el código que procesa los datos ingresados para la temporización y en la figura 2.16 se obtiene la hora del sistema para tener una referencia de tiempo.

```
public void Focos(){
        Ventana.setVisible(true);
        Ventana.jButton1.setBackground(Color.green);
        Ventana.jButton2.setBackground(Color.green);
        Ventana.jButton3.setBackground(Color.green);
        Ventana.jButton4.setBackground(Color.green);
        if ((posicion[0]==1)||(posicion[0]==2))
            Ventana.jButton1.setBackground(Color.red);
        if ((posicion[1]==1)||(posicion[1]==2))
            Ventana.jButton2.setBackground(Color.red);
        if ((posicion[2]==1)||(posicion[2]==2))
            Ventana.jButton3.setBackground(Color.red);
        if ((posicion[3]==1)||(posicion[3]==2))
            Ventana.jButton4.setBackground(Color.red);
            Ventana.jButton5.setBackground(Color.red);
            Ventana.jButton6.setBackground(Color.red);
```

Fig. 2.17 Código que permite la visualización de las salidas.

La figura 2.17 corresponde al código que permite visualizar las salidas en el panel donde se muestra el estado de las bobinas; el código ejecuta los cambios de color de los recuadros del panel de rojo a verde, según la información que le llega de las bobinas.

2.5 Configuración y Funcionamiento

Nótese a primera vista la interfaz del prototipo, véase la figura 2.4, en desarrollo que como vemos solo se centra en la funcionalidad básica de

lenguaje de contactos, mas no en el diseño de la interfaz, ya que por el momento es solo un panel de pruebas para lograr la lógica del algoritmo.

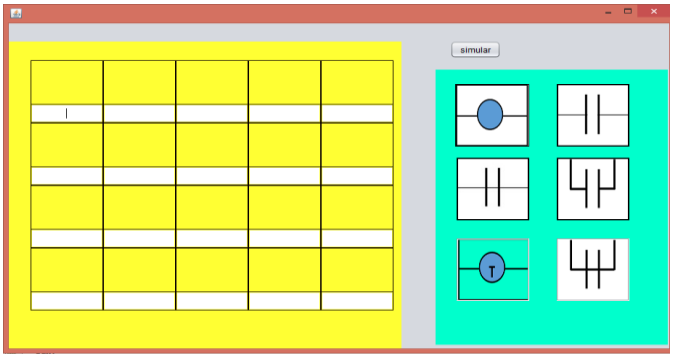


Fig. 2.18 Panel de pruebas del prototipo.

En la figura 2.18, verá enumerados los elementos disponibles del panel de pruebas del prototipo, anunciados de esta manera:

Lado derecho del panel.

1 Contactos Abiertos
2 Contactos cerrados
3 Contactos abiertos en paralelo
4 Contactos cerrados en paralelo
5 Temporizadores
6 Bobinas

Lado izquierdo del panel:

7 Botón de simular (inicia la simulación)
8 Botón de stop (detiene la simulación)
9 Casillero de texto

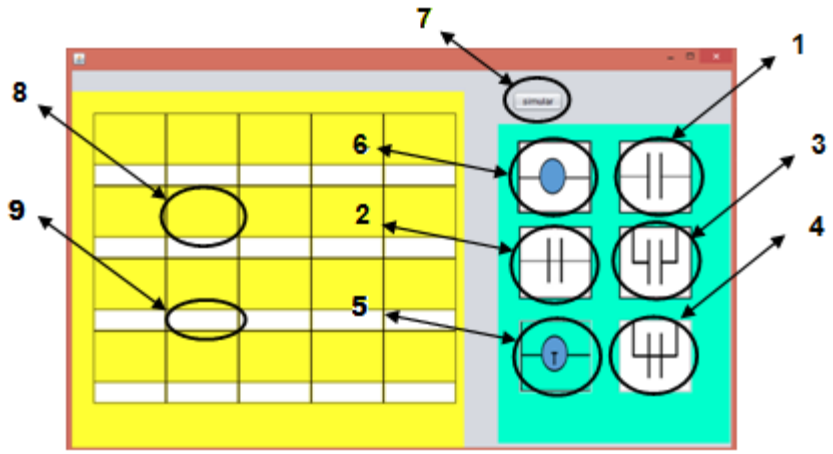


Fig. 2.19 Descripción de los componentes del panel.

Al dar click en el botón de simulación, índice 7 de la figura 2.19, aparece un panel que muestra varias bobinas, como lo indica la figura 2.20, en la que podrá visualizar las bobinas energizadas en color verde y desenergizadas en color rojo.

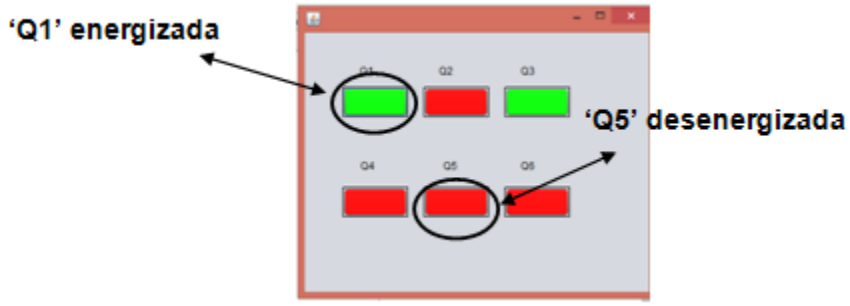


Fig. 2.20 Salidas $\boldsymbol{Q_1}$, $\boldsymbol{Q_3}$ están activadas y $\boldsymbol{Q_2}$, $\boldsymbol{Q_4}$, $\boldsymbol{Q_5}$ y $\boldsymbol{Q_6}$ desactivadas.

Como vemos solo se muestran las bobinas de salida, denominadas con su inicial "Q", mas no aparecen las bobinas auxiliares denominadas con su inicial "M". Las bobinas energizadas aparecen de color verde mientras que las desenergizadas aparecen con el color rojo. En este prototipo solo nos limitamos a 6 salidas ya que solo se necesita probar la eficacia del algoritmo

que gobernará el sistema. El número de salidas será ampliado a 50 salidas en el sistema finalizado.

Para realizar la primera prueba se procede a construir en el panel de casillas la secuencia de salida por medio de los elementos en la parte derecha, entonces procedemos a deslizar los elementos de accionamiento hasta los casilleros, dando click en el recuadro del elemento y manteniendo apretado el botón izquierdo del mouse se mueve el cursor hasta llegar a la casilla deseada, como se indica en la figura 2.21.

Fig. 2. 21 Arrastre de un contacto abierto hacia las casillas.

En la figura 2.21, se muestra el arrastre de otros componentes hacia las casillas.

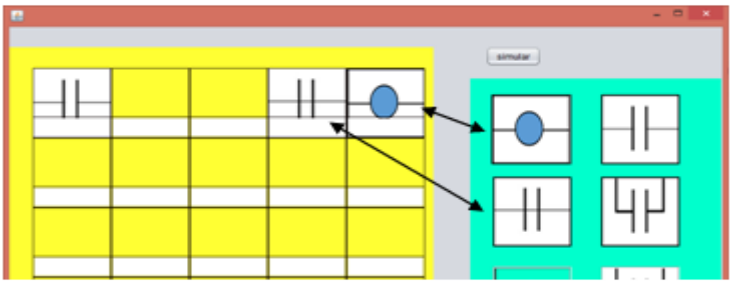


Fig. 2. 22 Arrastre de un contacto cerrado y bobina hacia las casillas.

Cabe recalcar que el algoritmo necesita que las bobinas tanto auxiliares como principales, el contador y temporizador deben ser posicionados en la columna final, como se muestra en la figura 2.23, de otra manera el programa no

funcionará; como vemos tiene las misma metodología que la mayoría de software de automatización, ya que estos se tomaron como referencia a la hora de hacer el proyecto.

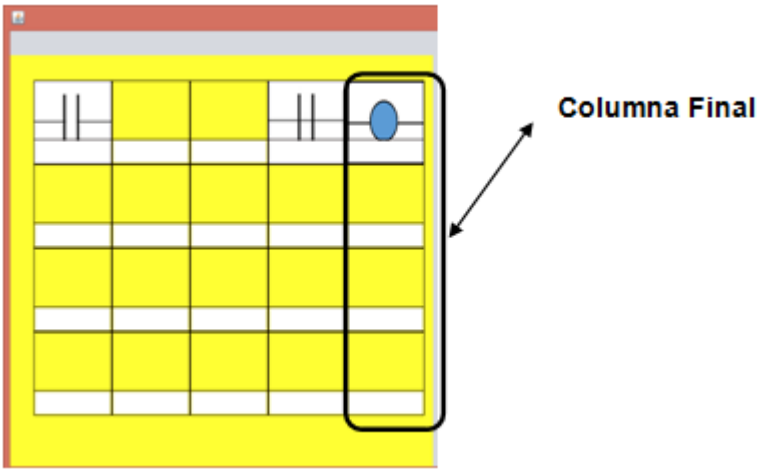


Fig. 2. 23 Posición de las bobinas.

Para introducir líneas en los casilleros simplemente basta con hacer un doble click en la casilla que se desea sobreponer una línea, claro está, esto se hace antes de oprimir el botón de simulación; una vez los gráficos han sido colocados en las casillas se puede proceder a darle nombre a cada elemento puesto, no necesariamente se puede llevar este orden, los gráficos bien pueden ser colocados luego de colocar nombres en las casillas de texto. Es así como verá en la figura 2.24:

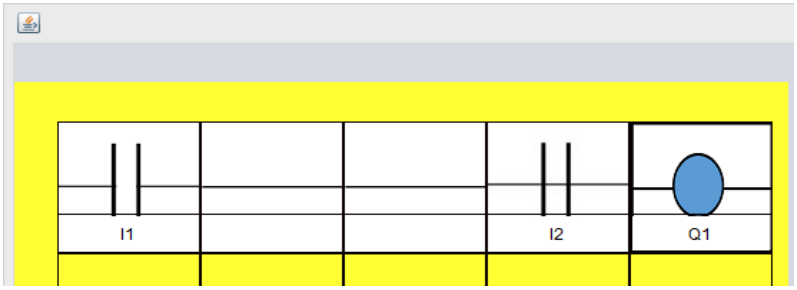


Fig. 2. 24 Línea de código en lenguaje de bloques.

Para las notaciones de los elementos optamos por las típicas referencias que se le dan a estos dispositivos en distintos software; es así que todos tienen una referencia que conlleva una letra y un número. La letra identificara el tipo de elemento, seguido de un número que en conjunto le dan nombre para identificar a dicho elemento, como también lo verá en la figura 2.24; por tanto:

‘I1’ corresponde a un primer contacto o switch

‘I2’ corresponde a un segundo contacto o switch

‘M1’ corresponde a una primera bobina auxiliar con sus respectivos contactos

‘Q1’ corresponde a una primera bobina principal con sus respectivos contactos

‘T1’ corresponde a un temporizador ‘On Delay’ con sus contactos

‘C1’ corresponde a un contador con sus respectivos contactos

Volviendo a la línea de código en lenguaje de bloques anteriormente expuesta, podemos ver que al dar click en el botón simular aparece el panel de visualización de salidas, con el resultado correspondiente a la línea formada.

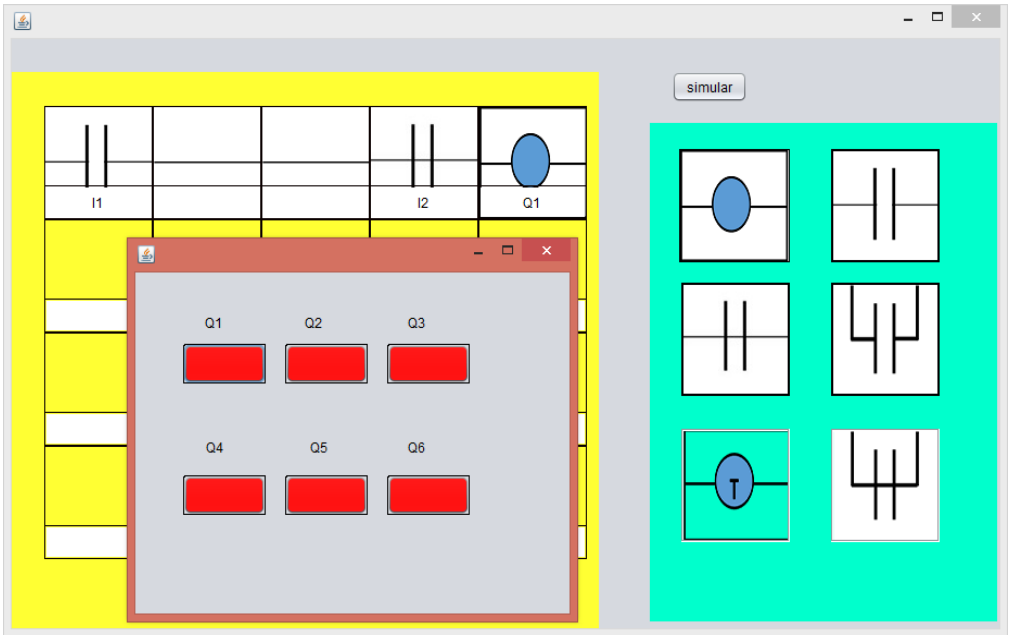


Fig. 2.25 Línea de código con panel de salidas.

En la imagen se aprecia como ‘Q1’ se encuentra en color rojo, lo que indica que dicha bobina esta desenergizada, el resto de bobinas no participan del código graficado, por tanto también permanecen apagadas.

A dar click en el botón de simular se entra en estado de simulación, lo que indica que podemos hacer cambios de estado en los elementos ya expuestos, tales como cerrar o abrir contactos, lo que interferirá en las salidas logrando energizar o desenergizar bobinas. Esto se logra dando doble click en algún contacto así este abrirá estando cerrado y cerraran los que en principio estaban abiertos.

En la figura 2.26 muestra a ’I1’ en estado inicial como abierto y en la figura 2.27 después del doble click muestra al mismo contacto como cerrado, dando así que ‘Q1’ cambia a color verde, como lo indica la figura 2.28.

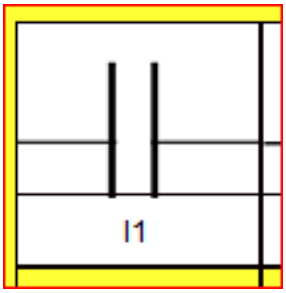


Fig. 2.26 Antes del doble click.

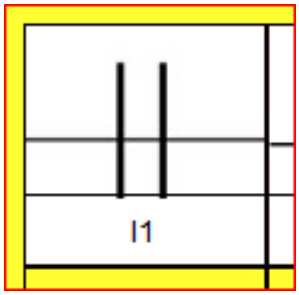


Fig. 2.27 Después del doble click.

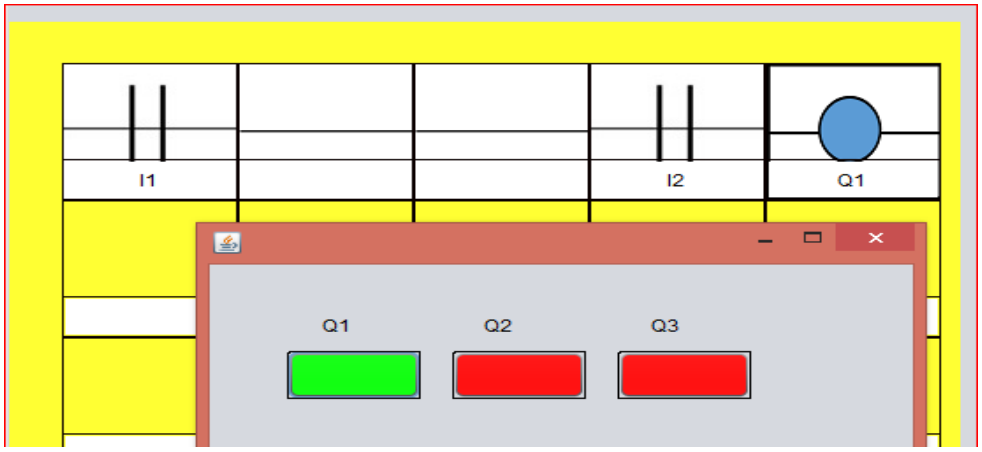


Fig. 2.28 Proceso de cambio de estado de un contacto normalmente abierto.

En la figura 2.29 verá el clásico ejemplo del enclavamiento de una bobina. Como primer paso arrastramos los elementos a las casillas antes de simular, así mismo tendremos una bobina el cual tendrá su contacto que se encontrara en paralelo a ‘I1’ el que permitirá el enclavamiento. Al dar el mismo nombre al

contacto que la bobina el programa reconocerá que dicho contacto pertenece a esa bobina, si esta energizada el contacto normalmente abierto cerrara automáticamente, dicho proceso ocurrirá con toda bobina energizada que tenga uno o más contactos. En caso de que la bobina tenga contactos normalmente cerrados, estos abrirán automáticamente al energizarse; el panel de salida no verá estos cambios, solo mostrara el estado de las bobinas principales.

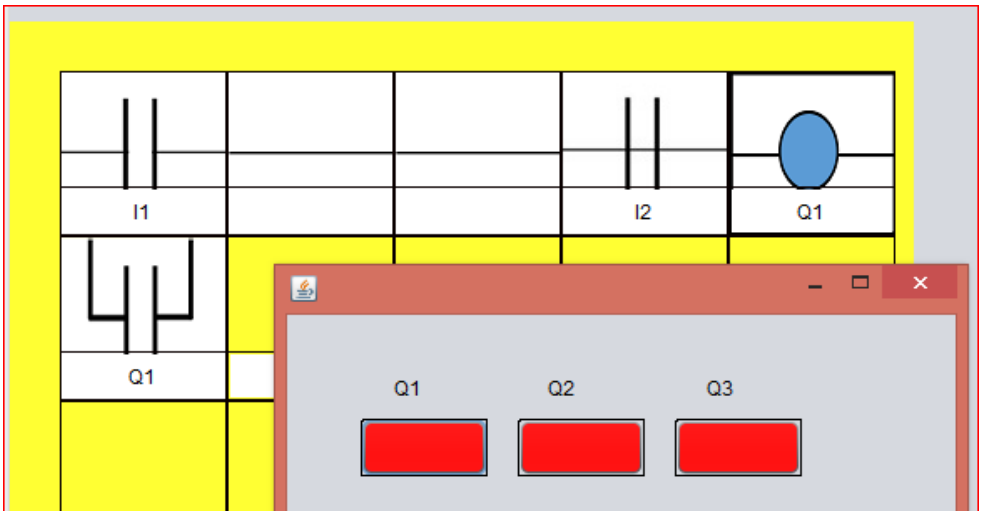


Fig. 2.29 Enclavamiento de una bobina ‘Q1’.

Luego de pasar al estado de simulación dando click al botón de simular podemos hacer el enclavamiento dando doble click al contacto ‘I1’ cerrándolo y dando paso a energizar la bobina cerrando su contacto, con lo cual ya no importa el estado de ‘I1’ dado que ya no afecta el estado de la bobina y la única manera de desenergizar ‘Q1’ es abriendo el contacto ‘I2’ que actúa como pulsador de paro. La imagen siguiente muestra el resultado de cerrar ‘I1’, el resultado se visualiza en la figura 2.30.

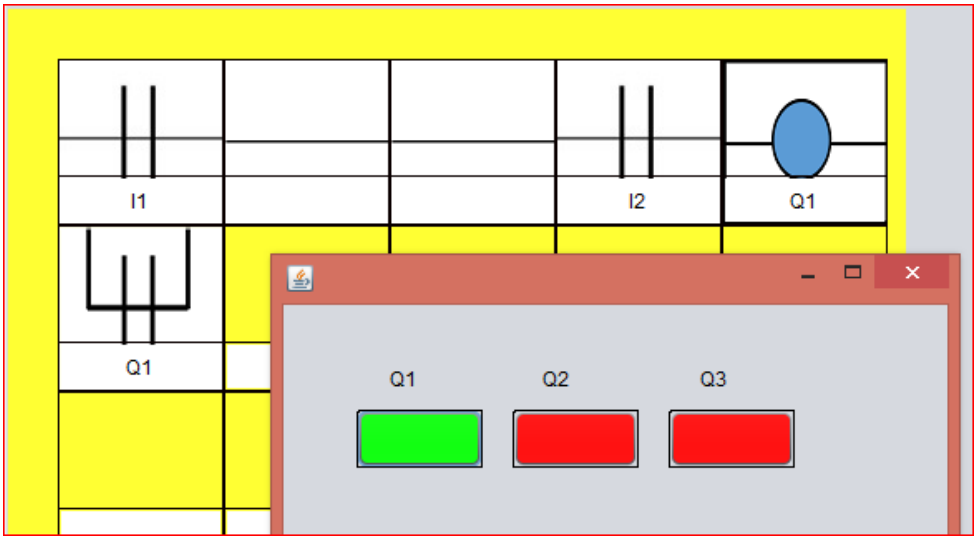


Fig. 2.30 Cerrando 'I1' para lograr el enclavamiento de la bobina 'Q1'.

En la figura 2.31 podremos visualizar el resultado de abrir 'I1'.

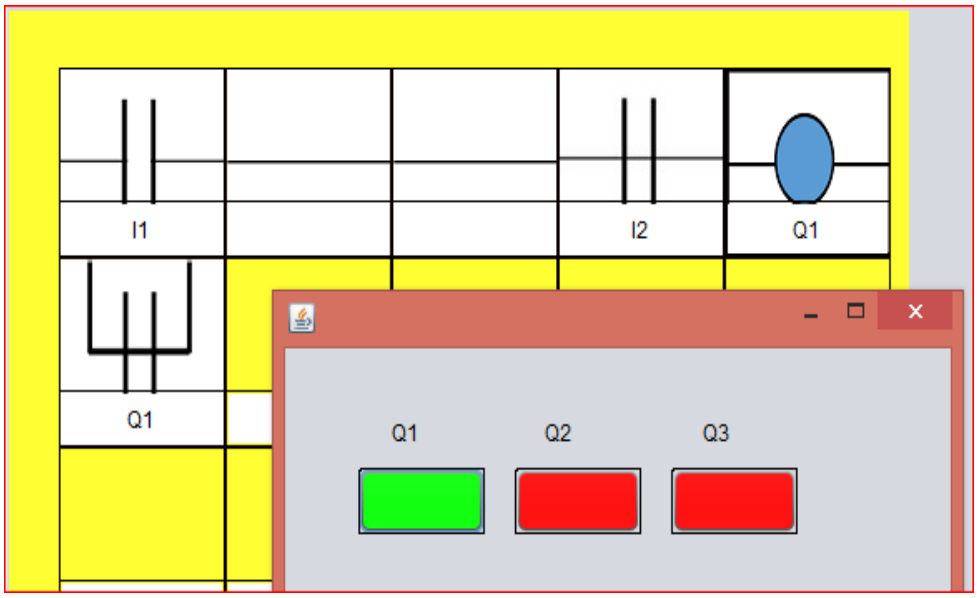


Fig. 2.31 Abriendo contacto 'I1' para corroborar el enclavamiento.

En la figura 2.32 muestra a 'Q1' desenergizada al abrir 'I2'.

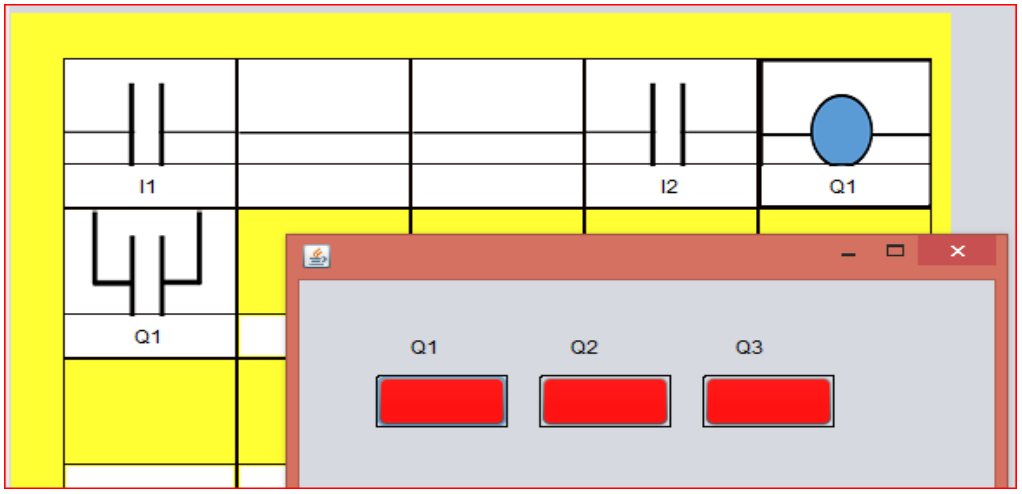


Fig. 2.32 Abriendo contacto ‘I2’ para desenergizar la bobina ‘Q1’.

Hay que decir que también se puede se poner el mismo nombre a varios contactos, estos actuaran como uno solo, estando en diferentes posiciones, ya sean estos contactos normalmente abiertos o cerrados.

También se tiene un temporizador On Delay que energizará una bobina luego de transcurrido un tiempo definido por el usuario antes de entrar al estado de simulación. Esto se logra al dar el nombre al elemento en su correspondiente casilla de texto y luego oprimir el botón del teclado ‘enter’, esto conseguirá la salida de un mensaje que pedirá al usuario teclear un entero que será el tiempo en segundos que tardara la bobina en energizarse, como se lo indica en la figura 2.33.

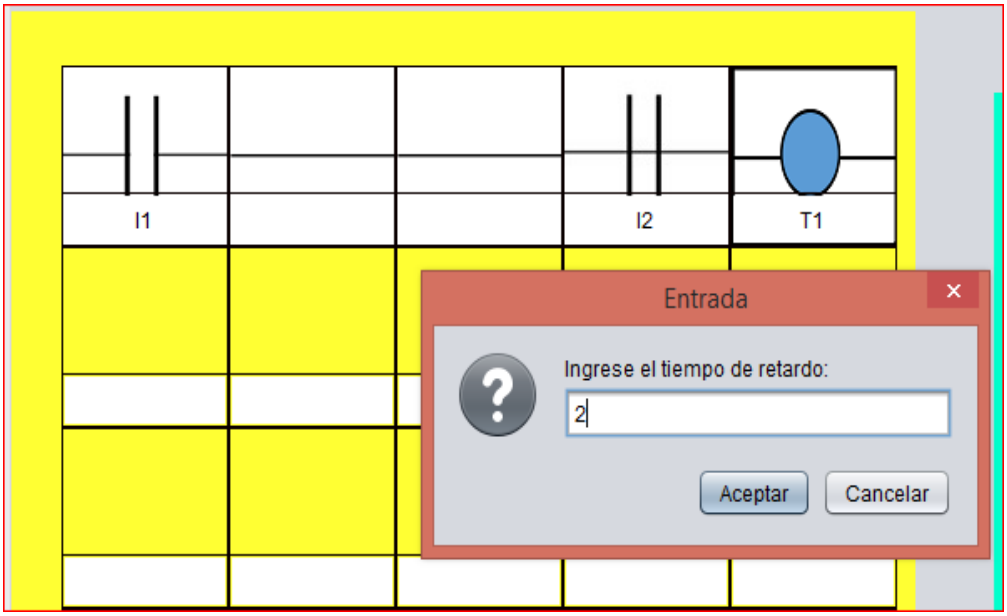


Fig. 2.33 Ingreso de tiempo de retardo en segundos.

Ahora se mostrará el diagrama completo del código en lenguaje de bloques antes de entrar al estado de simulación. Veremos que 'T1' no es una salida pero este ayudara con un contacto 'T1' que será que le dará el paso a energizar la bobina que tendremos como salida, como lo indica la figura 2.33.

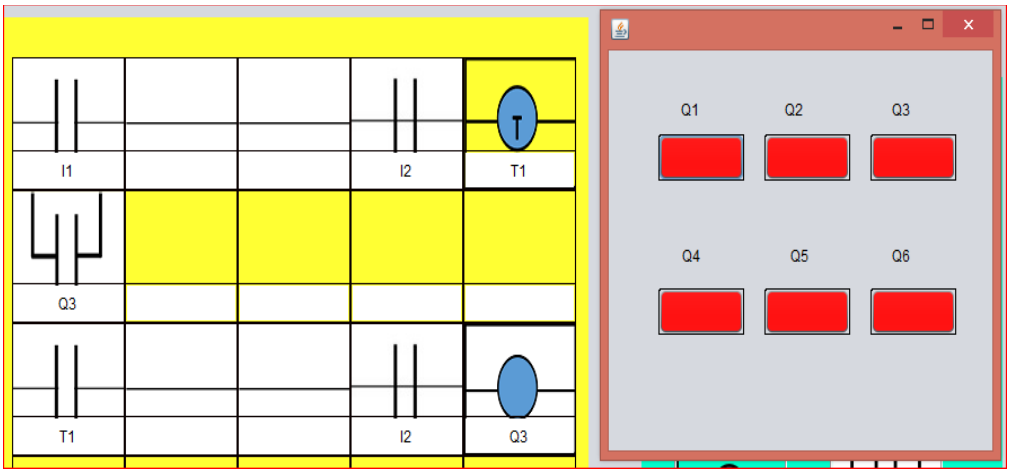


Fig. 2.34 Enclavamiento con temporizador.

En la figura 2.35 notamos que a pesar de que ‘I1’ está cerrado el contacto de ‘T1’ no se cierra y por tanto no energiza ‘Q3’, dado que aún no transcurren los dos segundos que fueron definidos por el usuario.

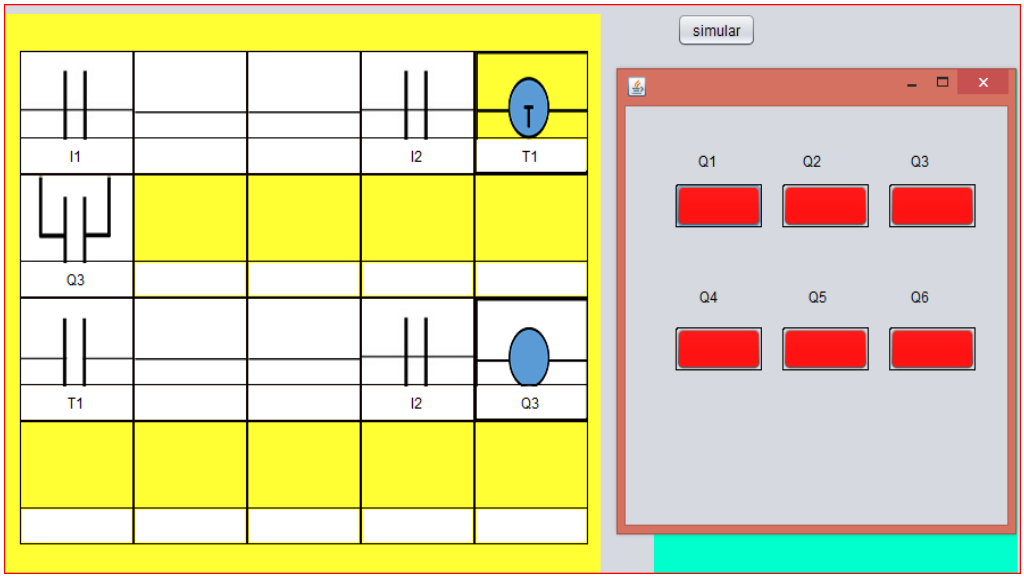


Fig. 2.35 ‘T1’ a la espera de energizar ‘Q3’.

Después de los 2 segundos ‘Q3’ se energiza como lo indica la figura 2.36.

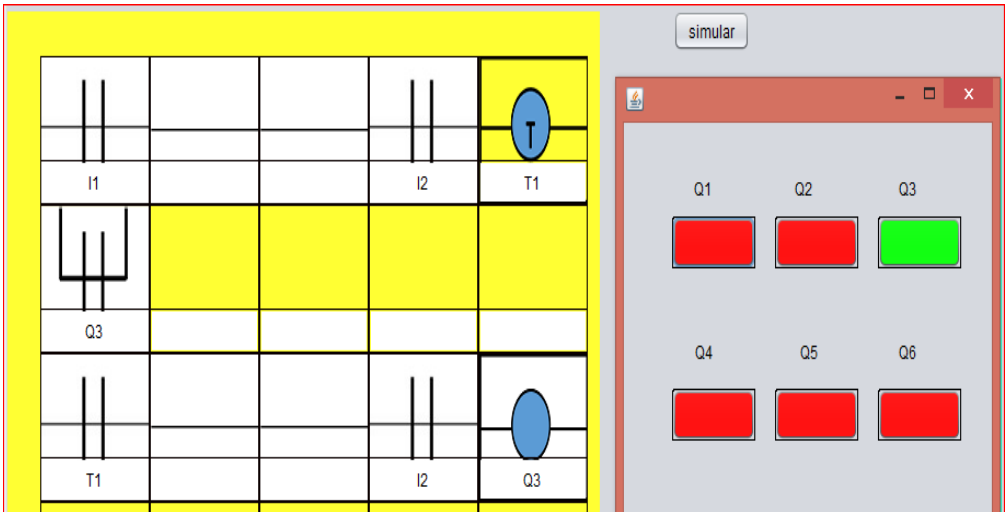


Fig. 2.36 Enclavamiento de ‘Q3’ por medio de ‘T1’.

El sistema también cuenta con un contador que opera de la misma manera que el temporizador, así como antes se describió su nombre empieza con la inicial ‘C’. En la siguiente imagen vemos su uso agregando un número el cual representa las veces que el contador recibe altos de señal para cerrar su contacto, como lo muestra la figura 2.37. El sistema detecta si se trata de un contador o temporizador por medio de la letra inicial.

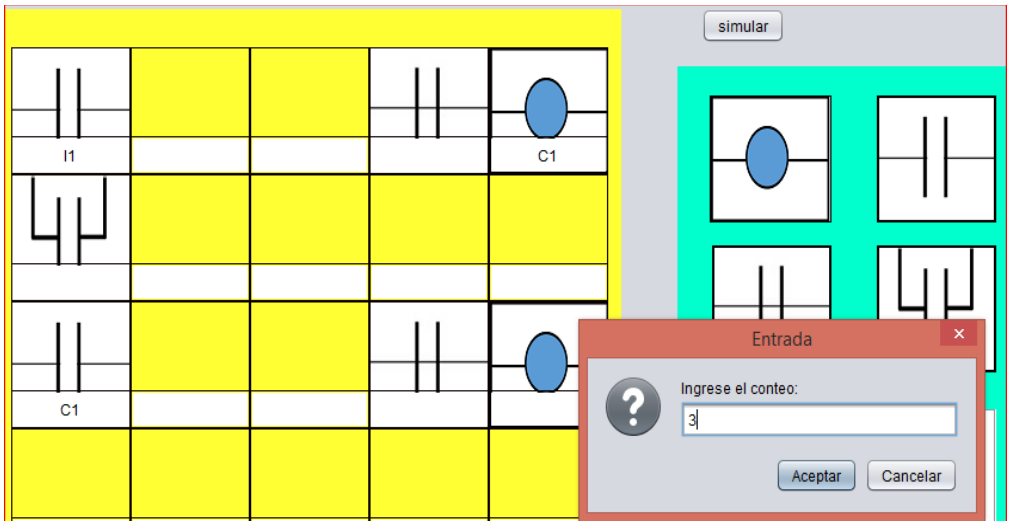


Fig. 2.37 Ingresando el número de altos.

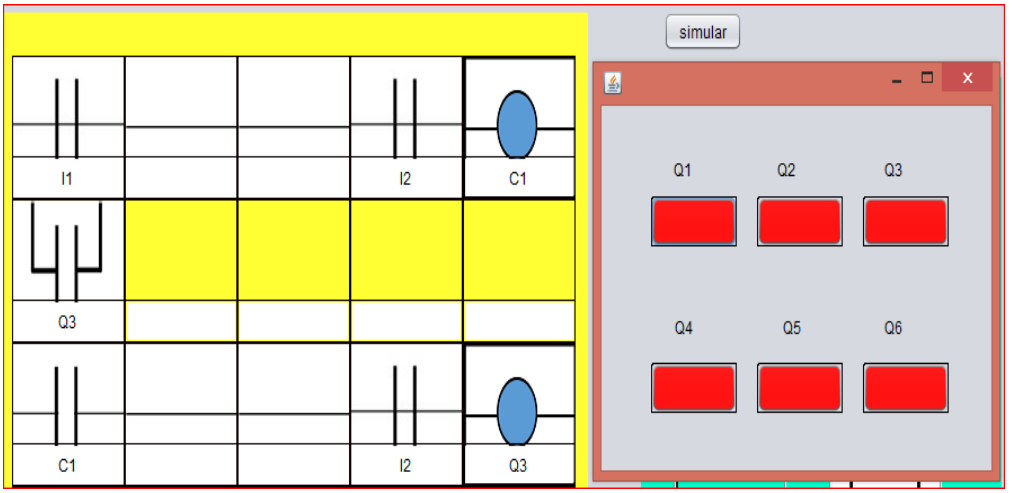


Fig. 2.38 Primer alto de señal al contador.

Vemos que al cerrar ‘I1’ no ocurre el enclavamiento puesto que hemos definido 3 altos de señal a ‘C1’ para que cierre su contacto y así enclava ‘Q3’como en la figura 2.38. Y en la figura 2.39 muestra el enclavamiento luego de cerrar 3 veces el contacto ‘I1’

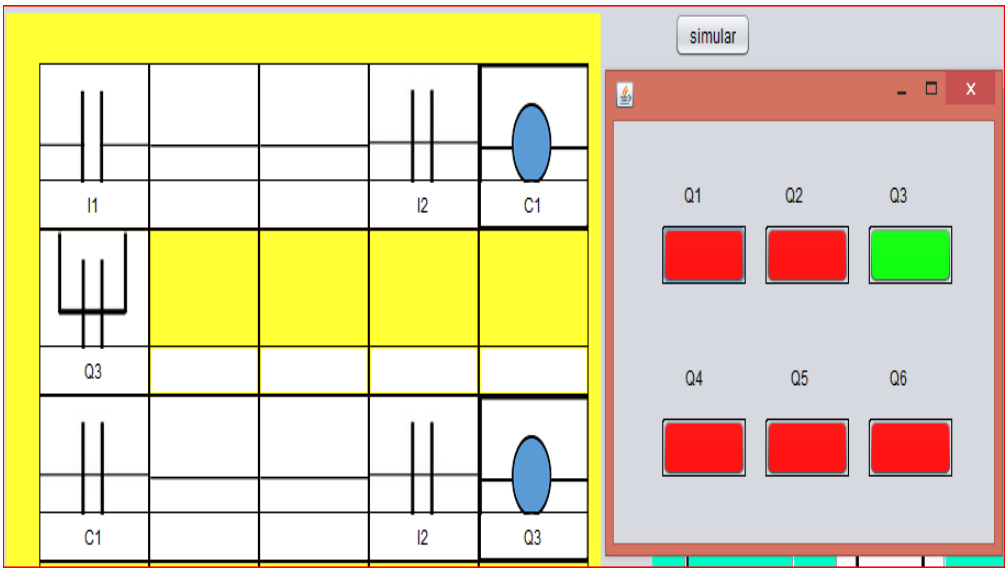


Fig. 2.39 Enclavamiento por medio de un contador.

CAPÍTULO 3

3. RESULTADOS

En el capítulo 2 se definió la interpretación de lo que sería la interfaz gráfica, dicho interfaz solo era un prototipo con la finalidad de probar el algoritmo que gobernará las secuencias de los procesos a los que sea aplicado. Ahora se ha reestructurado la interfaz para que sea más llamativa al usuario.

3.1 Interfaz Gráfica definitiva

La imagen siguiente muestra la interfaz final, véase en la figura 3.1, así mismo se aumentó el número de casillas en las que se puede agregar código en lenguaje de contactos. Específicamente ahora tenemos un recuadro de 14 x 8 casillas; es de aclarar que se puede incluso agregar muchas más casillas; pero esto hace que el programa utilice muchos recursos, lo que le quita eficiencia y velocidad de ejecución.

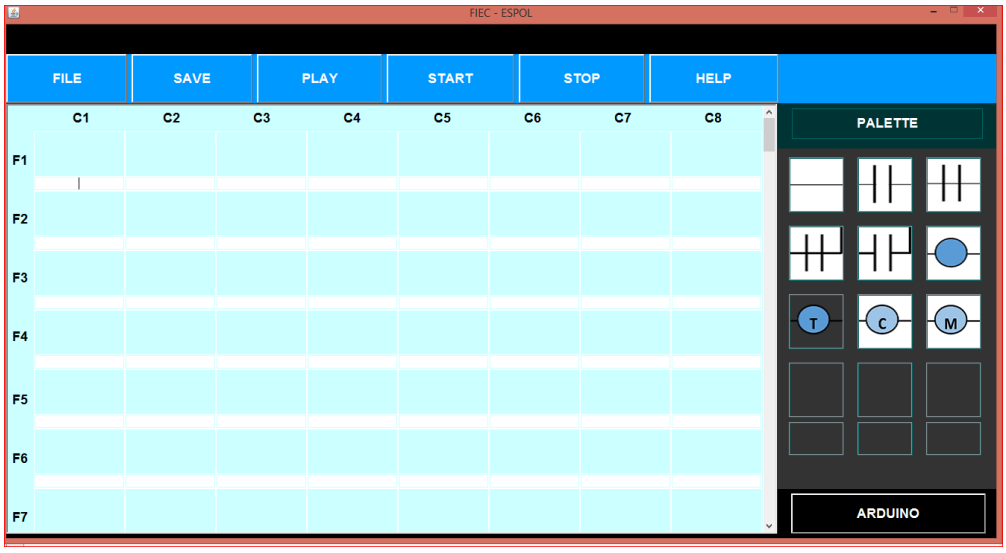


Fig. 3.1 Interfaz gráfica final.

3.2 Panel de Salidas

En la figura 3.2 vemos como luce el panel de salidas representados por recuadros con su respectiva referencia. Como se aprecia solo hay 14 salidas visibles. Su funcionamiento es igual al panel de visualización del capítulo, “verde” cuando hay una salida activa y “rojo” cuando la salida esta desactivada.

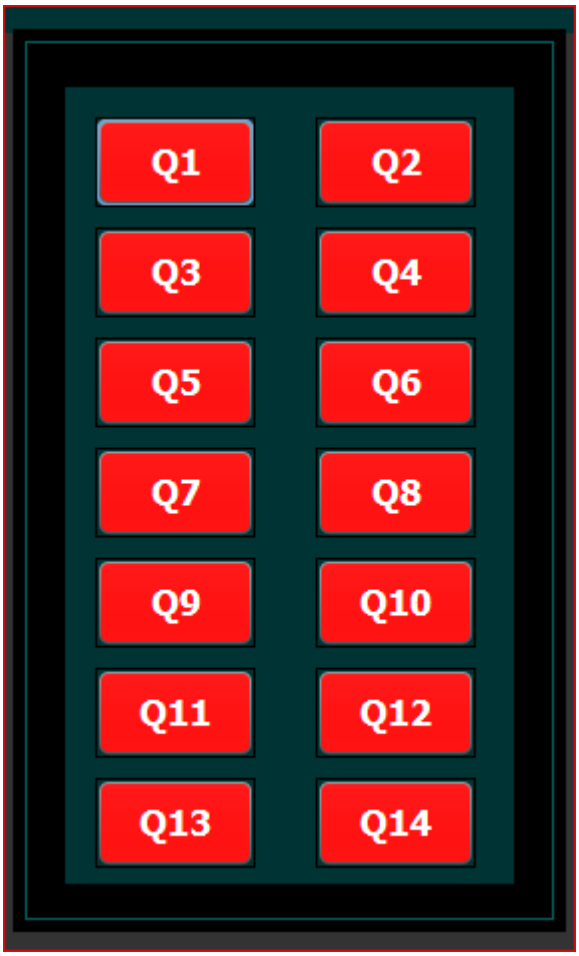


Fig. 3.2 Panel de Visualización.

Esta imagen aparece por encima del panel de paletas, desde donde se arrastran los elementos hacia las casillas; con esto se evita intentos de edición mientras se encuentra en modo de simulación.

3.3 Simulación

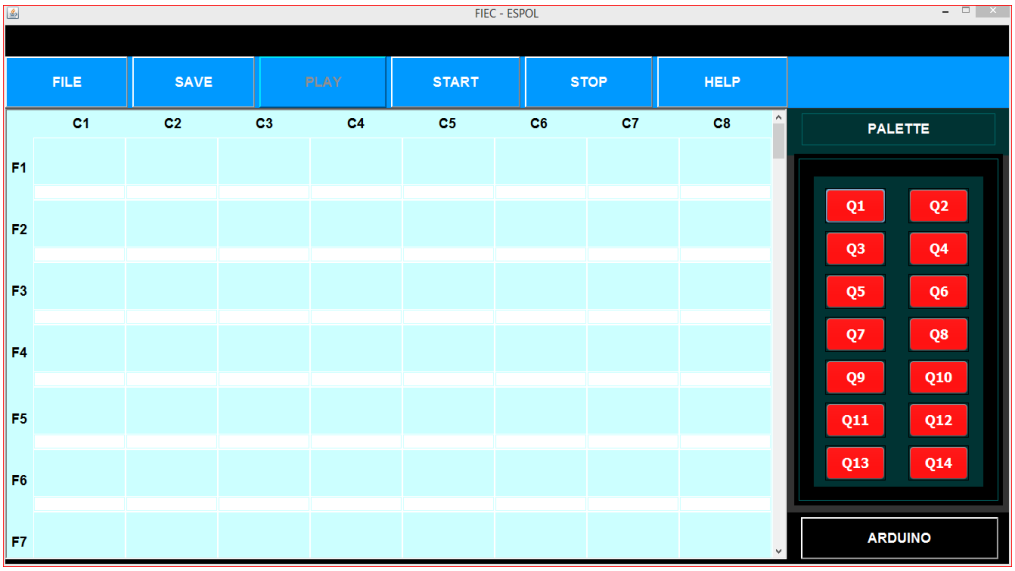


Fig. 3.3 Modo simulación.

Se puede notar en la figura 3.3 el botón de conexión a Arduino, que se encuentra en la parte inferior derecha, al dar click dará inicio la conexión al dispositivo, hay que aclarar que no hay una interfaz construida para configurar el puerto al que se conectará ni que salidas serán visibles en Arduino, todo esto se hará de forma manual. En este proyecto ya se tiene configurado de forma manual el puerto que leerá Java, que es el mismo que leerá Arduino; así mismo están configurados los pines de la tarjeta Arduino que se utilizaran. Todo esto se hará previamente en la programación en Arduino; la gráfica siguiente muestra la sección de código que setea los pines y puerto que se utilizará.

```
ArduinoSwitch
int input;

void setup() {
  pinMode(13, OUTPUT); // Declaramos que utilizaremos el pin 13 como salida
  pinMode(12, OUTPUT); // Declaramos que utilizaremos el pin 13 como salida
  Serial.begin(9600); //Se inicia la comunicación serie
}
```

Fig. 3.4 Configuración de pines.

Como vio en la figura 3.4, el código en Arduino ya está definido por tanto los pines de salida ya no pueden ser modificados a través de Java, pero si se puede variar el orden de salida de las “Q” asociados a cada pin; en otras palabras si se lo requiere se puede manipular el código en Java para cambiar las “Q” asociadas con cada pin. Para este proyecto se cuenta con lo siguiente:

PIN 13: Q1, PIN 12: Q2, PIN 11: Q3, PIN 10: Q4, PIN 9: Q5, PIN 8: Q6

Las salidas visibles van limitadas por el número de pines disponibles, en este caso solo se hace uso de 6 pines para observas las salidas, claro está que se pueden usar todos los pines disponibles de la tarjeta y luego hacer los cambios respectivos en el programa hecho en Java.

En la figura 3.5 muestra un programa funcional más completo utilizando todos los recursos que tiene, tanto como contactores, temporizadores, contadores y salidas.

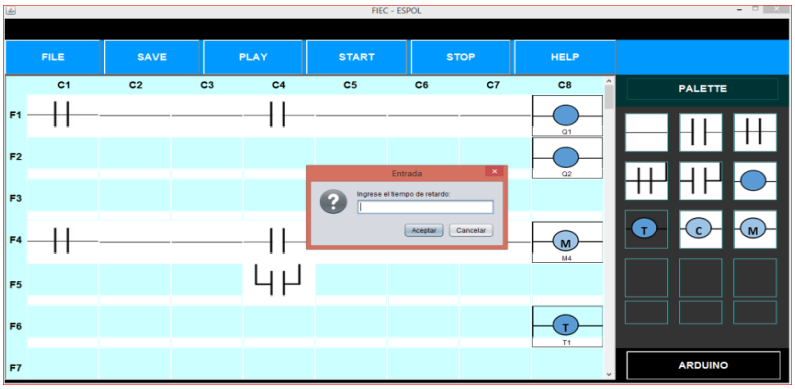


Fig. 3.5 Programa en funcionamiento.

En el ejemplo se agregan tanto un switch de marcha representado por un contacto “I1”, como un switch de paro representado por un contacto cerrado “I2”.

La figura 3.6 muestra el código completo que dará lugar a las secuencias de salidas en forma de luces. Y la figura 3.7 muestra el programa en modo simulación.

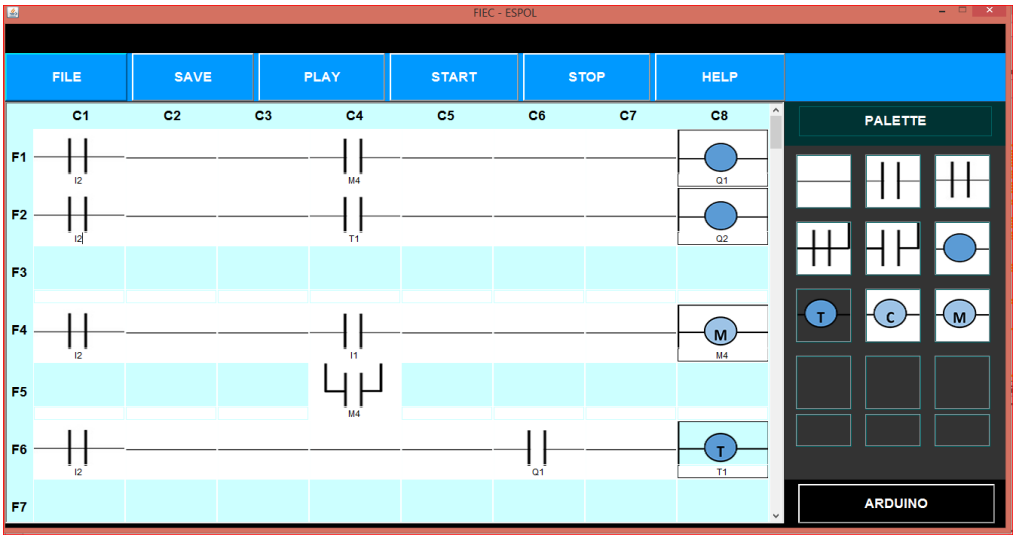


Fig. 3.6 Líneas de código en lenguaje de contactos.

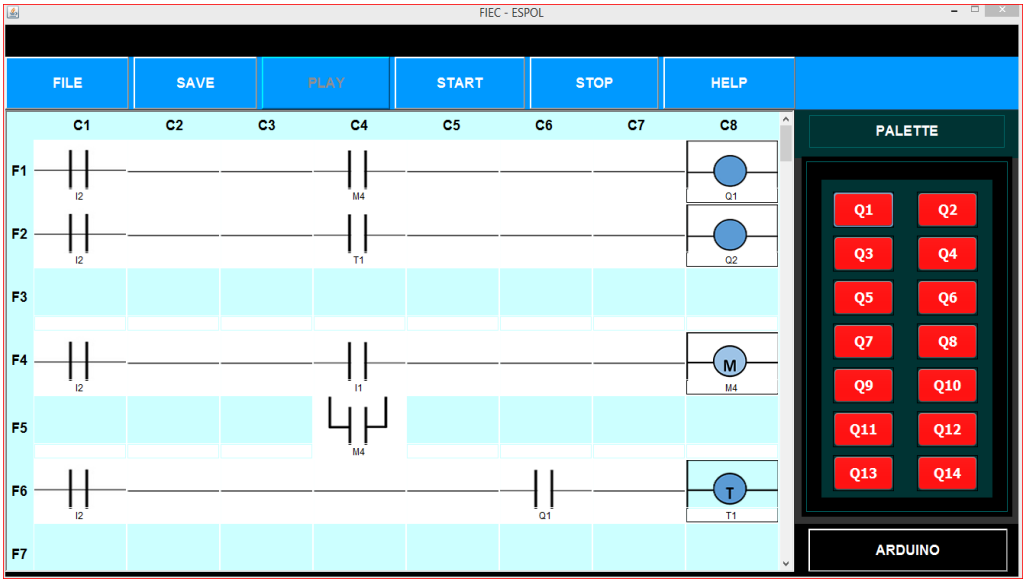


Fig. 3.7 En modo “simulación”.

A continuación la figura 3.8 muestra el pulsador de marcha activado al dar doble click sobre él.

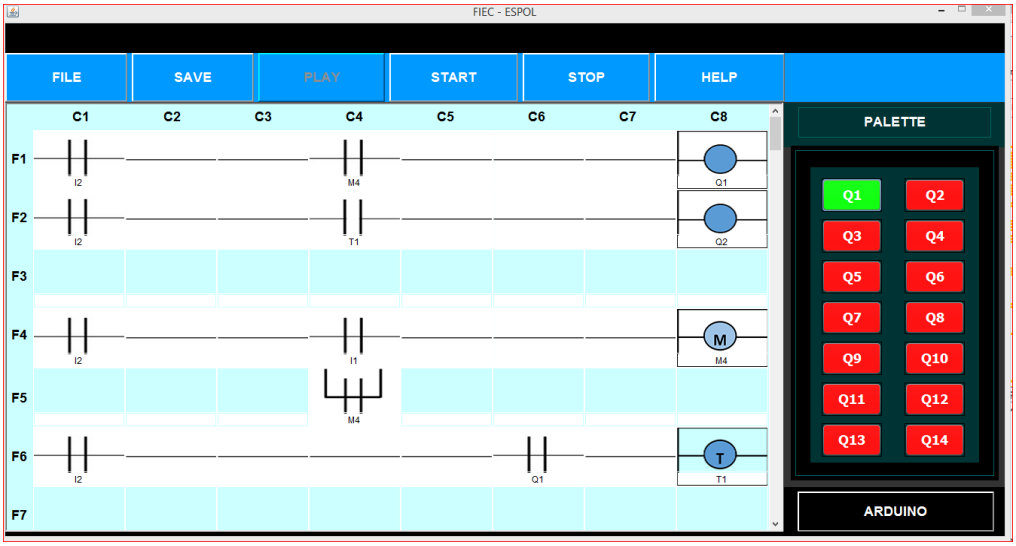


Fig. 3.8 Activación de “Q1”.

Luego de actuar el temporizador se obtiene la activación de ‘Q1’ y ‘Q2’, véase la figura 3.9.

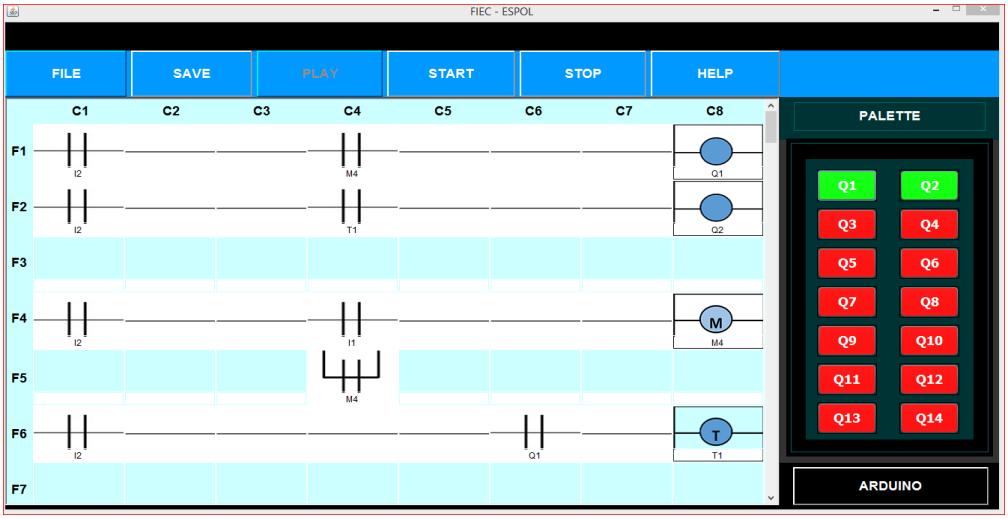


Fig. 3.9 Activación de “Q1” y “Q2”.

3.4 Conexión con Arduino

Al dar click en el botón de Arduino, dará inicio la conexión. En la imagen se muestra como este botón se pinta de rojo mostrando su activación, como se lo identifica en la figura 3.10.

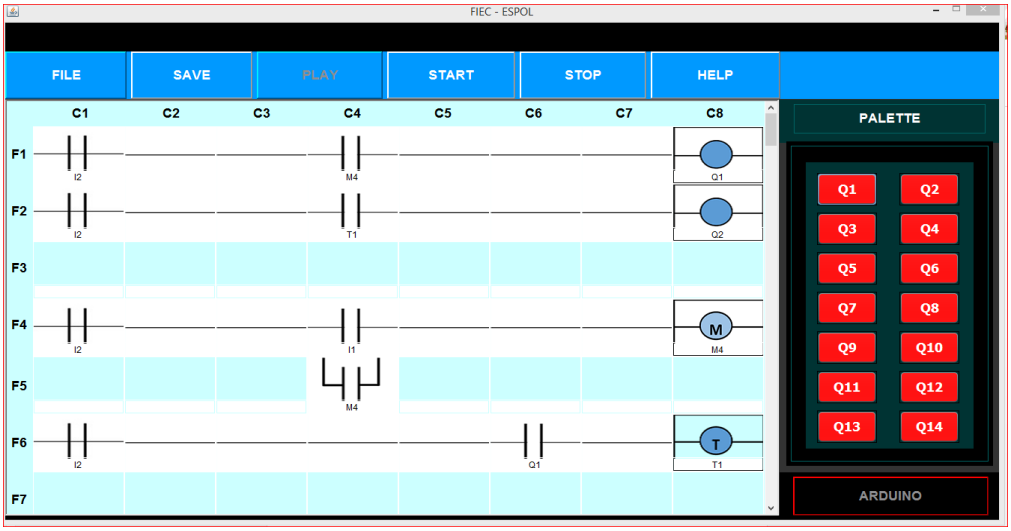


Fig. 3.10 Conexión Iniciada.

Se vuelve a dar click en el contacto de marcha, ahora con la conexión activada, se muestra la salida correspondiente en la tarjeta con la cual podrá ver las salidas activándose a través de la tarjeta, como se muestra en la figura 3.11 y figura 3.12.

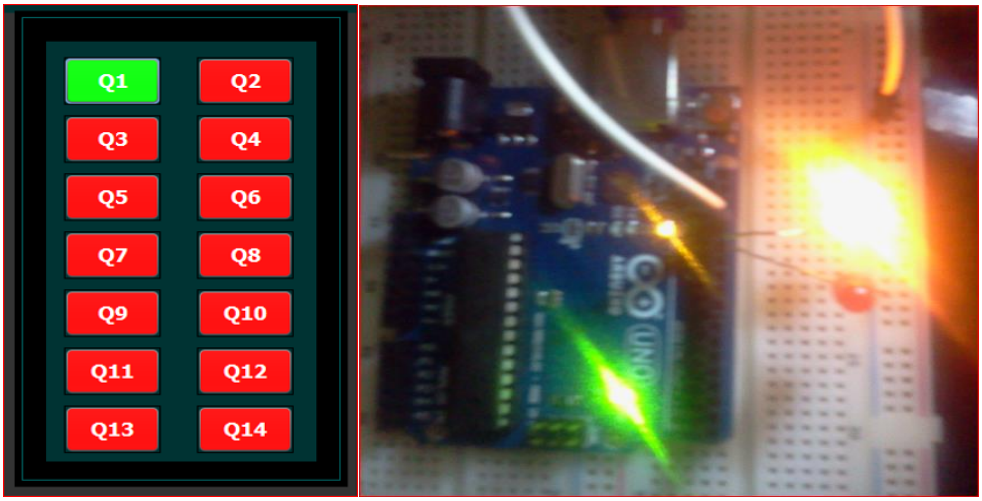


Fig. 3.11 Salidas en Arduino

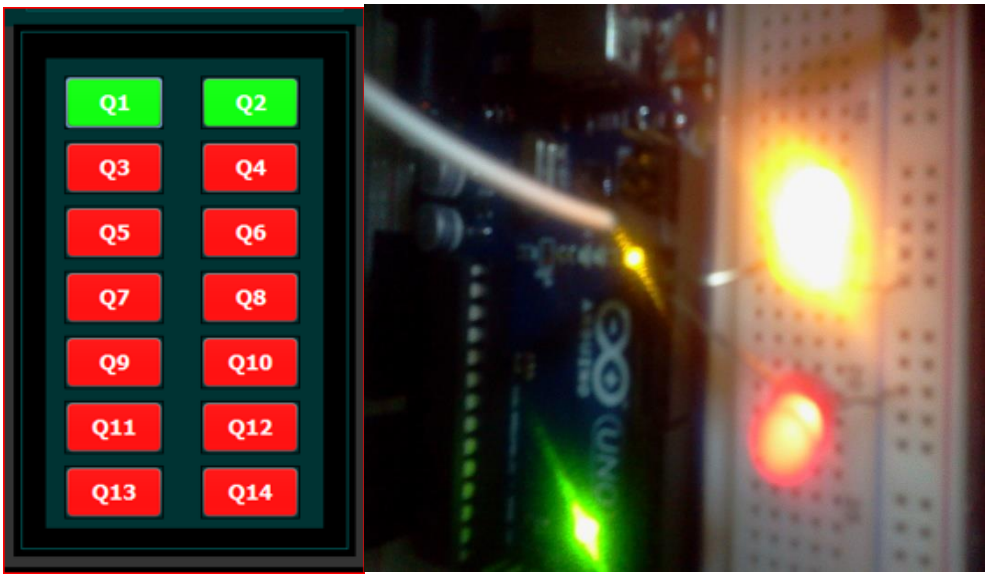


Fig. 3.12 Salidas “Q1” y “Q2” en Arduino

Luego de dar click en el contacto de paro, los leds se apagan; incluso se puede apreciar al momento de ejecutar, que la transmisión de datos a Arduino es muy rápida y no da problemas.

Los leds solo representan presencia de voltaje, el cual claramente se puede manejar, para posteriormente con alguna configuración electrónica dar un uso aplicativo representando algún proceso industrial u otro proceso que requiera de determinadas secuencias.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. Se ha logrado realizar la emulación de un autómata con ciertas limitantes de salida, tan solo utilizando herramientas muy generales tales como: lenguaje Java y la Tarjeta Arduino.

2. Se creó un Software capaz de simular las acciones de un autómata tanto que las salidas se pueden ver virtual como físicamente. Si bien este software no fue creado con el fin de competir con programas profesionales para autómatas, sí logra convertirse en una alternativa viable para las personas que recién se están adentrando en el tema de la programación en lenguaje de contactos.

3. En lo que corresponde al lenguaje, se concluye que es posible crear ciertas secuencias lógicas de control utilizándolo a tal punto de crear un software comparable con los paquetes de software especializados en automatización respecto a funcionamiento y desarrollo, mas no es tan sofisticado como los comerciales en lo que corresponde a diseño y funcionalidades varias.

4. Se tomó como referencia varios softwares profesionales dada su semejanza, con cada uno se utilizó la misma metodología para la visualización y manejo en la interfaz.

5. La tarjeta Arduino puede ser utilizada como un autómata, solo se necesita de la configuración electrónica correcta y adecuada. Esta tarjeta cumple la mayoría de los requerimientos de un autómata ya que tiene pines de entrada y salida, también tiene entradas analógicas. En este proyecto solo se utiliza las entradas digitales ya que son suficientes para corroborar las capacidades como autómata de esta tarjeta.

6. Para la conexión de Java con Arduino hacen falta librerías externas, esta integración se basa en la comunicación de Java con el puerto serial. Dicha interacción es detectada por Arduino, con lo que lee tal información capturada para procesarla y dar sus resultados en forma de señales de voltaje.

Recomendaciones

1. Al realizar el software se tuvo en cuenta la proporción de la interfaz en la pantalla dado que sufre distorsión y todos los componentes se reordenan por tanto el tamaño del programa ya está definido sin oportunidad a ser escalado, por lo que se recomienda considerar esto para evitar problemas con respecto a diseño.

2. Si bien el software está ampliamente validado para muchas acciones, aún restan ciertas validaciones que no fueron agregadas al programa porque resultan ser errores muy obvios a la hora de realizar el código en lenguaje de contactos. Es imprescindible revisar los códigos realizados y que estén perfectamente referenciados ya que se podría producir errores a la hora de simular o que el programa se comporte de manera errónea.

3. Al momento de realizar la conexión el sistema se detiene, por tanto no hay que alarmarse sino esperar a que se logre la conexión. Para lograrla se debe realizar manualmente, es decir, se necesita editar código tanto en Java como en la IDE de Arduino. El programa ya tiene conexiones definidas.

4. El programa está abierto a modificaciones, tales como la adición de una interfaz para la configuración de puertos y pines sin tener que tocar código. Es recomendable que este software pueda ser sujeto a optimizaciones de rendimiento de código.

BIBLIOGRAFÍA

[1] P. Digitel, “Programación basada en objetos,” en Como programar en Java, Ed. New Jersey: Prentice Hall, 2003, pp. 341-344.

[2] Anthony García González. (2015, Enero 21). PanamaHitek [Online]. Disponible en: http://panamahitek.com/category/arduinojava/

[3] José Carlos Villajulca. (2009, Octubre 24). Conociendo el Lenguaje Ladder [Online]. Disponible en: http://www.instrumentacionycontrol.net

[4] José Antonio Rodríguez. (2012, Enero 28). Comunicación serial Java y Arduino [Online]. Disponible en: http://josedeveloper.com/2012/01/28/comunicacion-serial-java-y-arduino/

ANEXOS

ÍNDICE DE FIGURAS