open source hardware

Maker Faire Bilbao

Automatización Industrial & Arduino™

Metodología de programación en C basada en máquinas de estado *(Grafcet)*

Aritz Elgezabal Núñez

Ander Fernández León

Jon Ander Gómez Rico

Miguel Pérez de Laborda

Fecha de publicación: 18 de Noviembre de 2016

Hoja índice de la memoria

1. Objeto

Gran parte de los equipos de producción industriales funcionan de forma automática o en su defecto pueden ser automatizados. Durante nuestra formación en el área de la automatización industrial, hemos encontrado difícil realizar alguna práctica en el hogar (cosa que nos permitiría mejorar nuestras competencias profesionales). La causa de esto no es más que el elevado coste de los autómatas industriales y de las licencias del software que se utiliza para programarlos. Al mismo tiempo observamos que estos PLCs que el mercado nos ofrece son tan caros como obsoletos.

Por ello, este proyecto ofrece una solución barata y libre, que nos permite adaptar un simple microcontrolador, como sería el incluido en Arduino™, al entorno industrial y sus sistemas de programación actuales basados en máquinas de estado.

2. Alcance

Este proyecto va dirigido en especial a universidades, centros formativos, profesionales, alumnos y *makers* que quieran tener la opción de automatizar procesos bajo normativas industriales o similares. Al mismo tiempo, la idea es dejar atrás los ya, a nuestro modo de ver, anticuados autómatas industriales, y avanzar a nuevos sistemas más novedosos en el área de la automatización. Sólo partiendo de herramientas libres y accesibles vemos posible esta meta.

3. Antecedentes

Teniendo en cuenta que se utilizará Arduino™ y su IDE para programar el autómata, se ha decidido crear una metodología práctica que culmine con la traducción del programa al lenguaje de Arduino™. La sencillez de diseño de los programas informáticos utilizando máquinas de estado conduce a escoger el *Grafcet* como método para comenzar con los programas que se realicen.

Por lo tanto, la metodología que se plantea aquí pretende convertir un esquema gráfico de máquinas de estado (llamado grafcet) a un código de programa en C/C++ que el IDE de Arduino™ sea capaz de compilar.

4. Normas y referencias

4.1 Disposiciones legales y normas aplicadas

Normativas y Reglamentos Industriales

En cuanto a normativas y reglamentos técnicos relativos al uso de la electricidad en el entorno industrial, el proyecto se encuentra limitado en ciertos aspectos por las normativas vigentes.

El proyecto cumple los requisitos de las directivas 2006/95/EC (Baja tensión), 2004/108/CE (Compatibilidad electromagnética) y RoHS 2011/65/EU de la Unión Europea.

Reglamento Electrotécnico para baja tensión (España):

"Artículo 6. Equipos y materiales."

1. *"Los materiales y equipos utilizados en las instalaciones deberán ser utilizados en la forma y para la finalidad que fueron fabricados. Los incluidos en el campo de aplicación de la reglamentación de trasposición de las Directivas de la Unión Europea deberán cumplir con lo establecido en las mismas. En lo no cubierto por tal reglamentación se aplicarán los criterios técnicos preceptuados por el presente Reglamento. En particular, se incluirán junto con los equipos y materiales las indicaciones necesarias para su correcta instalación y uso, debiendo marcarse con las siguientes indicaciones mínimas."*
    a. *"Identificación del fabricante, representante legal o responsable de la comercialización."*
    b. *"Marca y modelo."*
    c. *"Tensión y potencia (o intensidad) asignadas."*
    d. *"Cualquier otra indicación referente al uso específico del material o equipo, asignado por el fabricante."*
2. *"Los órganos competentes de las Comunidades Autónomas verificarán el cumplimiento de las exigencias técnicas de los materiales y equipos sujetos a este Reglamento. La verificación podrá efectuarse por muestreo."*

"Artículo 17. Receptores y puesta a tierra."

*"Sin perjuicio de las disposiciones referentes a los requisitos técnicos de diseño de los materiales eléctricos, según lo estipulado en el **artículo 6**, la instalación de los receptores, así como el sistema de protección por puesta a tierra, deberán respetar lo dispuesto en las correspondientes instrucciones técnicas complementarias."*

"Artículo 20. Mantenimiento de las instalaciones."

"Los titulares de las instalaciones deberán mantener en buen estado de funcionamiento sus instalaciones, utilizándose de acuerdo con sus características y absteniéndose de intervenir en las mismas para modificarlas. Si son necesarias modificaciones, éstas deberán ser efectuadas por un instalador autorizado."

4.2 Bibliografía

Es posible encontrar el nuevo reglamento de baja tensión nacional en: http://www.upv.es/electrica/newrbt_1.htm

4.3 Programas de cálculo

Desde ***el equipo de desarrollo VinGaS*** se apoya el movimiento de hardware libre y, del mismo modo, el proyecto se ha apoyado, sobre todo, en programas de código abierto para llevarse a cabo (con contadas excepciones).

Programación:

- Software Libre:

  → Arduino™ IDE *(Disponible en: http://www.arduino.cc/).*

Diseño gráfico:

- Software Libre:

  → GIMP *(Disponible en:http://www.gimp.org.es/).*

Organización de proyectos:

- Software Libre:

  → Planner *(Disponible en:https://packages.debian.org/es/jessie/planner).*

Documentación:

- Software Libre:

  → LibreOffice *(Disponible en: https://es.libreoffice.org/).*

5. Definiciones y abreviaturas

Arduino™

Arduino™ es una plataforma de desarrollo de prototipos con licencia de hardware libre. La plataforma incluye la placa de desarrollo, el IDE (Entorno de programación) y toda la comunidad de Makers (hacedores) que comparte proyectos, hardware, programas y librerías, lo que lo convierte en una plataforma versátil y de gran éxito en todo el mundo.

Autómata programable / PLC (Programmable Logic Computer)

Un controlador lógico programable es una computadora utilizada en automatización industrial, para automatizar procesos. Como por ejemplo, el control de la maquinaria de la fábrica en líneas de montaje o atracciones mecánicas.

Programa

Se le denomina programa a la lista de instrucciones escritas (en lenguaje específico) que se usa para controlar las tareas de una máquina, normalmente algún tipo de ordenador.

6. Análisis y soluciones

Programación

En cuanto a la programación, se ha desarrollado un método sistemático para convertir una representación gráfica como el GRAFCET en lenguaje Arduino™. Es posible convertir el GRAFCET al lenguaje de Arduino™ con otros métodos, pero, se ha encontrado este especialmente eficaz. Para ello:

➔ Primero, se dibuja el Grafcet.

Es la base en la que se apoya el programa. El grafcet debe seguir las normativas de Grafcet y ser fiel al proceso que se desea automatizar.

➔ Para pasar de grafcet a C es posible utilizar un array booleano para marcar las Etapas activas. De este modo, se guardan en 32 bytes un total de 256 etapas. (Cada array ocupa lo que se necesite, pero ese valor se usa como referencia). Un array es una cadena de variables (un grupo de variables ordenadas referenciadas por un nombre), en este caso de variables booleanas o bits (0,1) que se definen con el nombre del array y la cantidad de variables que contiene entre corchetes. Para definir el array de etapas, se inicializa un array de este modo:

```
//Declaración de variables
boolean etapa[256]={1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,[...],0,0,0,0};
```

➔ Es posible utilizar una instrucción ***for*** para que rellene el array con ceros y no tener que insertar manualmente 255 ceros:

```
//Declaración de variables
char cMatrix;
boolean etapa[256];

void setup() {
//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
etapa[0]=1;
}
```

➔ Insertar etapas (Grafcet)

Para insertar etapas en el programa, se utiliza la instrucción if haciendo referencia al array booleano anteriormente programado:

```
//Declaración de variables
char cMatrix;
boolean etapa[256];

void setup() {
//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
void loop() {
etapa[0]=1; //etapa de arranque en este ejemplo
}
//Etapa 0
if (etapa[0]) {
        //Instrucciones de la etapa
}
}
//Fin Etapa 0

//Etapa 1
if (etapa[1]) {
        //Instrucciones de la etapa
}
}
//Fin Etapa 1
```

➔ Insertar transiciones (Grafcet)

Para insertar transiciones, dentro de las etapas creadas, se introduce otra instrucción *if* en la que se indica la condición para el cambio. Dentro se indica la etapa nueva y se borra la etapa actual:

```
//Declaración de variables
char cMatrix;
boolean etapa[256];

void setup() {
//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
}
void loop() {
etapa[0]=1; //etapa de arranque en este ejemplo

//Etapa 0
if (etapa[0]) {
        //Instrucciones de la etapa
        if (Transición "0 a 1")
        {
                //Contra-Instrucciones de la etapa
                etapa[0]=0;
                etapa[1]=1;
        }
}
//Fin Etapa 0

//Etapa 1
else if (etapa[1]) {
        //Instrucciones de la etapa
        if (Transición "1 a 0") {
```

```
                    //Contra-Instrucciones de la etapa
                    etapa[1]=0;
                    etapa[0]=1;
            }
}
//Fin Etapa 1

}
```

➔ Salidas

Según la norma que rigen los autómatas industriales, el ciclo de scan consta de 3 fases; Lectura de sensores, comprobación de estado, activación de salidas. Por lo tanto, la activación de las salidas pertenece a una sección diferente que se añade al final del programa principal.

```
//Declaración de variables
char cMatrix;
boolean etapa[256];
void setup() {
//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
}
void loop() {
etapa[0]=1; //etapa de arranque en este ejemplo

//Etapa 0
if (etapa[0]) {
        //Instrucciones de la etapa
        if (Transición "0 a 1") {
                //Contra-Instrucciones de la etapa
                etapa[0]=0;
                etapa[1]=1;
        }
}
//Fin Etapa 0

//Etapa 1
else if (etapa[1]) {
        //Instrucciones de la etapa
        if (Transición "1 a 0") {
                //Contra-Instrucciones de la etapa
                etapa[1]=0;
```

```
                etapa[0]=1;
        }
}
//Fin Etapa 1

//Activación de salidas
if (etapa 0 || etapa 1) {
 digitalWrite(7,HIGH);
digitalWrite(2,LOW);
 digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
 digitalWrite(6,LOW);
digitalWrite(8,LOW);
[...]
}
//Fin de activación de salidas
```

➔ Temporizadores

Los temporizadores en Arduino se realizan con interrupciones puesto que son más seguros y mantienen activo el ciclo de scan del microcontrolador. Para ello se utiliza la librería ***TimerOne.h***. Esta librería ofrece una serie de instrucciones de las que el programa se puede beneficiar.

Se utilizan las instrucciones ***Timer1.attachinterrupt(función_a_ejecutar); Timer1.Initialize(us);.*** La primera instrucción, permite configurar la función que ejecutará cuando la interrupción se active. La segunda instrucción, configura el tiempo que pasará desde que se llama a la interrupción hasta que se activa en microsegundos y activa el cronómetro. Teniendo esto en cuenta, se puede crear una interrupción cíclica. Esta interrupción se activa cada *X* segundos y entra en una función que nosotros queramos. Para temporizar, se utiliza esta función para contar ciclos (que equivaldrían a segundos, milisegundos o lo que el programador decida) y guardar este número en un entero sin signo (unsigned Int) dado que el tiempo sólo avanza hacia adelante. La variable deberá ser tipo *volatile* para que el valor pueda ser compartido por el programa principal y la interrupción.

```
#include <TimerOne.h>

//Declaración de variables
char cMatrix;
volatile unsigned int uiTemporizador=0;
boolean etapa[256];

void setup() {
//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
Timer1.attachinterrupt(temporizador);
Timer1.initialize(100000); // Temporización de 100 ms (100000 us)
}

void loop() {
etapa[0]=1; //etapa de arranque en este ejemplo
```

```
//Etapa 0
if (etapa[0]) {
        //Instrucciones de la etapa
        if (Transición "0 a 1") {
                //Contra-Instrucciones de la etapa
                etapa[0]=0;
                etapa[1]=1;
                uiTemporizador=0;
        }
}
//Fin Etapa 0

//Etapa 1
else if (etapa[1]) {
        if (uiTemporizador=1) {
                uiTemporizador=0
                etapa[1]=0;
                etapa[0]=1;
        }
}
//Fin Etapa 1

}
void temporizador() {
        uiTemporizador++;
}
```

➔ Paro de emergencia

El paro de emergencia tiene preferencia sobre el resto del programa por lo tanto se trata de una **interrupción accionada por hardware**. Para ello, se utilizan los pines correspondientes debido a que no todos los microprocesadores, o en nuestro caso plataformas de desarrollo, tienen las mismas entradas disponibles para interrupciones de este tipo. Se puede configurar la interrupción por nivel (HIGH o LOW) o por flancos (RISING o FALLING). Dado que la seta de emergencia tiene los contactos configurados como Normalmente Cerrados, interesa que en el momento que el pulsador deje de conducir (se abra el circuito) la interrupción se ejecute. Por eso lo se programa en su *flanco descendente o FALLING*. La configuración de la interrupción es la siguiente:

- *attachInterrupt(interrupt, ISR, mode)*

En la sección interrupt (dentro de los paréntesis) se indica la entrada que queremos destinar a la interrupción por hardware. En la sección mode (dentro del paréntesis) indicamos la configuración de disparo (HIGH, LOW, RISING o FALLING). ISR se refiere a la función que se ejecutará (esta función no podrá devolver información al programa principal). El programa quedaría así:

```
#include <TimerOne.h>

//Declaración de variables
char CMatrix;
volatile unsigned int uiTemporizador=0;
boolean etapa[256];

void setup() {
attachInterrupt(2,emergencia,FALLING);

//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
Timer1.attachinterrupt(temporizador);
Timer1.initialize(250000); // Temporización de 250 ms
```

```
}
void loop() {
etapa[0]=1; //etapa de arranque en este ejemplo

//Etapa 0
if (etapa[0]) {
        //Instrucciones de la etapa
        if ((Transición “0 a 1”) && (digitalRead(2)) ) {
                //Contra-Instrucciones de la etapa
                etapa[0]=0;
                etapa[1]=1;
                uiTemporizador=0;
        }
}
//Fin Etapa 0

//Etapa 1
else if (etapa[1]) {
        if (uiTemporizador=1) {
                uiTemporizador=0;
                etapa[1]=0;
                etapa[0]=1;
        }
}
//Fin Etapa 1

}
void temporizador() {
        uiTemporizador++;
}
void emergencia() {
        etapa[0]=1;
        etapa[1]=0;
}
```

➔ WatchDog

Se incluye un perro guardián en el programa para evitar que se quede trabado en alguna parte del programa. Para esto se utiliza la librería ***avr/wdt.h*** que ya viene en el compilador de Arduino IDE. Durante el Setup se desactiva el watchdog para que Arduino pueda configurar sin ejecutarse a contrarreloj con la instrucción ***wdt_disable();***. Al final del setup se vuelve a activar con la instrucción ***wdt_enable(WDTO_TIME)***; para que se active justo antes de iniciar el ciclo de scan. *WDTO_TIME* será el tiempo en el que el WatchDog saltará e irá de acuerdo con el microcontrolador. Para resetear el timer del Watchdog se utiliza la instrucción ***wdt_reset();***. El programa quedaría así:

```
#include <TimerOne.h>
#include <avr/wdt.h>

//Declaración de variables
char cMatrix;
volatile unsigned Int uiTemporizador=0;
boolean etapa[256];

void setup() {
wdt_disable();

attachInterrupt(2,emergencia,FALLING);

//Puesta a cero del array y arranque de las etapas iniciales de los grafcets
for (cMatrix=0;cMatrix<256;cMatrix++) {
        etapa[cMatrix]=0;
}
Timer1.attachinterrupt(temporizador);
Timer1.initialize(100000); // Temporización de 100 ms (100000 us)

wdt_enable(WDTO_500MS); //Watchdog a 500ms
}
void loop() {
etapa[0]=1; //etapa de arranque en este ejemplo
```

```
//Etapa 0
if (etapa[0])
{
        //Instrucciones de la etapa
        if ((Transición "0 a 1") && (digitalRead(2)) )
        {
                //Contra-Instrucciones de la etapa
                etapa[0]=0;
                etapa[1]=1;
                uiTemporizador=0;
        }
}
//Fin Etapa 0

//Etapa 1
else if (etapa[1]) {
        if (uiTemporizador=1) {
                uiTemporizador=0
                etapa[1]=0;
                etapa[0]=1;
        }
}
//Fin Etapa 1

}
void temporizador() {
        uiTemporizador++;
}
void emergencia() {
        etapa[0]=1;
        etapa[1]=0;
}
```