

# Práctica 4: Manejo de Salidas Digitales con Controllino Mega

Tyrone Miguel Novillo Bravo

Facultad de Ingeniería

Departamento de Telecomunicaciones

Universidad de Cuenca

tyrone.novillo@ucuenca.edu.ec

Pablo Andrés Bermeo García

Facultad de Ingeniería

Departamento de Telecomunicaciones

Universidad de Cuenca

pablo.bermeo@ucuenca.edu.ec

## Resumen

This work presents the design and implementation of an educational automation control board integrating an industrial-grade PLCs (Controllino Mega). Techniques include the use of non-blocking delays based on `millis()` to enable concurrent task execution without halting the main loop, and pointer arithmetic on arrays to manage a 3x3 LED matrix sequence efficiently. The results demonstrate improved responsiveness, scalability, and maintainability in embedded control applications, highlighting the critical role of timing and memory management strategies in industrial automation.

## Index Terms

non-blocking delays, pointer arithmetic, embedded automation, PLC programming, HMI integration

## I. INTRODUCCIÓN

Esta práctica describe el diseño e implementación de un sistema de control didáctico basado en el PLC Controllino Mega, integrado en un tablero de automatización. El objetivo es programar en C/C++ sobre arquitecturas Arduino con técnicas avanzadas de temporización y manejo de memoria.

Principalmente se emplea la técnica de retardos no bloqueantes mediante la función `millis()`, que permite ejecutar tareas periódicas sin detener el flujo principal de `loop()`.

Por otro lado, también se manipularán secuencias de iluminación mediante punteros sobre arreglos. Al definir un arreglo que mapea los pines del Controllino Mega y un puntero, la aritmética de punteros posibilita recorrer dinámicamente la estructura de datos, simplificando la implementación de patrones de encendido en matriz 3x3 sin repetir sentencias de control.

Finalmente, el proyecto integra todos estos elementos en una práctica de laboratorio en la que se enciende una secuencia de nueve LEDs siguiendo un patrón arbitrario.

## II. MARCO TEÓRICO

### II-A. Tablero de Control

El presente informe tiene como finalidad documentar el funcionamiento y uso del tablero de control utilizado en prácticas de automatización. Este tablero está diseñado con fines didácticos y está equipado con dos PLC de grado industrial: el Controllino Mega y el ESP32 PLC 14. Ambos dispositivos permiten la implementación práctica de sistemas de control mediante sus entradas y salidas digitales y analógicas, las cuales están interconectadas a través de borneras y actuadores ubicados en la estructura del tablero [2].

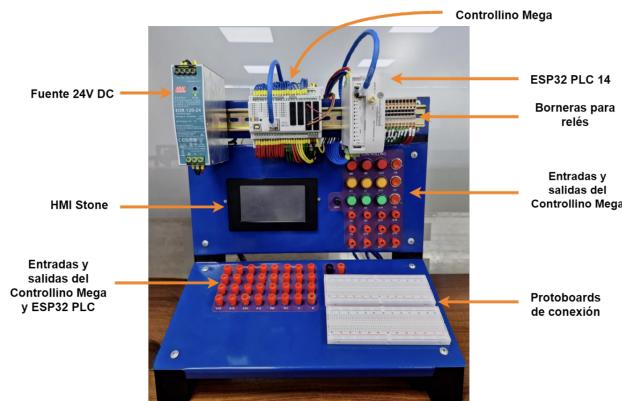


Figura 1. Componentes de la parte frontal del tablero. Fuente: Página 3 del manual.

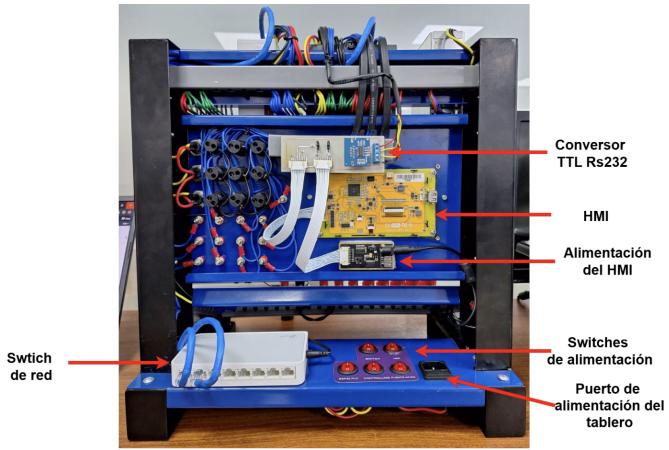


Figura 2. Componentes de la parte posterior del tablero. Fuente: Página 3 del manual.

Entre sus principales elementos se encuentra una pantalla HMI marca STONE, que facilita la interacción hombre-máquina. Esta interfaz se comunica mediante protocolo RS232 y permite la carga de proyectos desde PC o USB para el monitoreo y control de procesos. Su modelo es el STWC050WT-01, con una resolución de 800x480 píxeles y alimentación propia de 12 V [2].

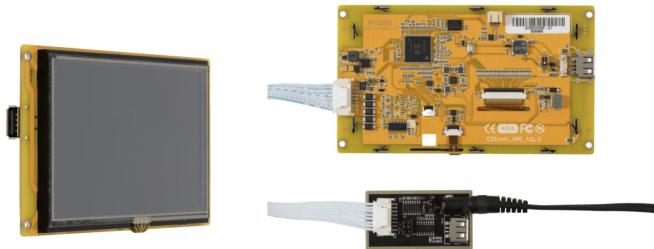


Figura 3. Vista frontal y posterior del HMI STONE. Fuente: Página 9 del manual.

En cuanto a los controladores, el Controllino Mega está basado en Arduino y presenta múltiples puertos para entradas y salidas, junto con indicadores LED, botones, relés y conectores RS485. Cada uno de estos componentes se encuentra mapeado y descrito en detalle, lo cual facilita su uso en diferentes prácticas de laboratorio.



Figura 4. Controllino Mega. Fuente: Página 4 del manual.

De manera similar, el ESP32 PLC 14 incorpora conectividad avanzada mediante Ethernet, WiFi, RS485 y LoRa, y también puede ser programado desde Arduino IDE. Este módulo cuenta con borneras para entradas y salidas digitales y relés que

permiten su uso versátil en entornos de control industrial.



Figura 5. ESP32 PLC 14 y sus borneras. Fuente: Página 7 del manual.

El procedimiento de encendido del tablero es secuencial y debe seguir un orden específico para garantizar el correcto funcionamiento del sistema. Primero se energiza la fuente principal, que proporciona 24 VDC a los controladores. Posteriormente, se encienden individualmente el Controllino Mega y el ESP32 PLC 14 mediante interruptores dedicados. Finalmente, tanto la interfaz HMI como el switch de red pueden ser activados por separado, ya que poseen fuentes independientes.

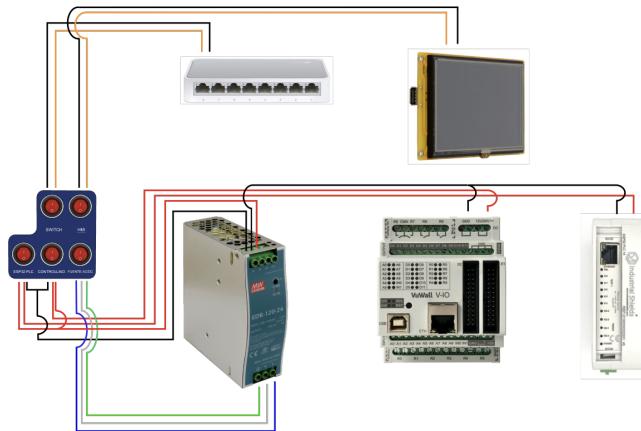


Figura 6. Conexión de energía del tablero. Fuente: Página 12 del manual.

Este conjunto de elementos conforma un entorno integral para el aprendizaje práctico de la automatización, brindando una base sólida para el desarrollo de prácticas que simulan aplicaciones reales.

## *II-B. Punteros*

En C/C++, un *puntero* es una variable que almacena la dirección de memoria de otra variable, en lugar de su valor. Esto permite, por ejemplo, recorrer arreglos de manera eficiente o pasar referencias a funciones sin duplicar datos. Para declarar un puntero a un entero, se escribe:

Punteros

```
1 % variable normal  
2 int x = 42;  
3 % p contiene la direcci n de x  
4 int* p = x;
```

int\* led = LEDS; La operación & obtiene la dirección de la variable, y la operación de desreferencia \* accede al valor apuntado. Con aritmética de punteros, un puntero *p* puede avanzar en memoria en pasos del tamaño del tipo apuntado: al hacer *p* + 1 se salta al siguiente entero en memoria.

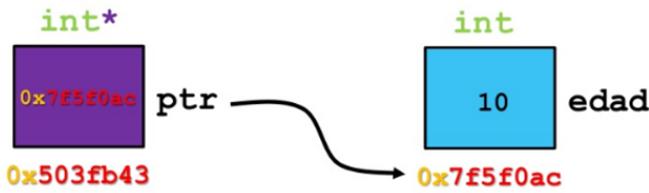


Figura 7. Diapositiva de la presentación: conceptos básicos de punteros.

### II-C. Retardos no bloqueantes

En aplicaciones embebidas es habitual necesitar ejecutar múltiples tareas de forma concurrente (lectura de sensores, comunicaciones, etc.). El uso de `delay()` detiene completamente el programa durante el tiempo especificado, impidiendo cualquier otra operación.

Para evitar este bloqueo, se emplea la función `millis()`, que devuelve el tiempo transcurrido en milisegundos desde el inicio del programa. El patrón consiste en:

1. Guardar en `t\_previo` el instante de la última acción.
2. En cada iteración de `loop()`, leer el tiempo actual en `t\_actual`.
3. Comparar `t\_actual - t\_previo` con el intervalo deseado.
4. Si ha transcurrido el tiempo, realizar la acción y actualizar `t\_previo`.

De este modo, el bucle principal nunca se detiene y puede atender otras tareas mientras espera el siguiente evento temporizado.

## III. DESARROLLO

En esta práctica se utilizará el módulo Controllino Mega para encender una secuencia de focos LED conectados en el tablero de prácticas. El código será desarrollado utilizando la librería «Controllino.h», que permite el uso de variables predefinidas para manipular las salidas digitales del dispositivo.

El siguiente código muestra la configuración inicial para usar el módulo que permite configurar una salida digital y alternar su estado entre alto y bajo, encendiendo y apagando un LED con un retardo de 500 milisegundos.

### Ejemplo de uso de una salida digital

```

1 //Libreria de controllino
2 #include <Controllino.h>
3 void setup ( ) {
4
5 // Salida digital D0
6 pinMode (CONTROLLINO_D0 , OUTPUT) ;
7 }
8
9 void loop ( ) {
10 //Salida D0 en alto
11 digitalWrite(CONTROLLINO_D0, HIGH);
12 delay(500);
13 //Salida D0 en bajo
14 digitalWrite(CONTROLLINO_D0, LOW);
15 }
```

Ahora bien, se explicará el código para controlar una matriz de 9 LED dispuestos en una cuadrícula de  $3 \times 3$ . Dichos LEDs están interconectados a los pines de la tarjeta Controlino Mega. El objetivo es encender los LED con la secuencia: LED 1 → LED 2 → LED 3 → LED 6 → LED 9 → LED 8 → LED 7 → LED 4 → LED 5, repitiéndose indefinidamente. La disposición de los LEDs se muestran en la Figura 8:



Figura 8. Disposición de los LEDs interconectados con Controllino en el tablero de control

El código del anexo 63 corresponde a la implementación de la secuencia mencionada utilizando punteros y retardos no bloqueantes.

Para lograr la secuencia de los LEDs, se ha usado retardos no bloqueantes utilizando la función `millis()` en lugar de emplear la función `delay()`. Esto permite que el microcontrolador continúe ejecutando otras tareas sin interrupciones, lo cual no es posible con `delay()`, ya que esta función detiene la ejecución durante el tiempo especificado.

Para controlar la función `millis()`, se declaran tres variables de tipo `unsigned long`: `t_previo`, `intervalo` y `t_actual`. La variable `intervalo` representa el tiempo de espera deseado, en este caso, 500 milisegundos (medio segundo). La variable `t_previo` almacena el instante en que se realizó la última acción, mientras que `t_actual` se actualiza continuamente con el valor devuelto por `millis()`.

El uso del tipo `unsigned long` es necesario porque la función `millis()` devuelve un valor numérico que representa el tiempo en milisegundos desde que se inició el programa, y este valor puede crecer rápidamente a medida que pasa el tiempo. Al ser una variable sin signo (`unsigned`), permite representar un rango mayor de valores positivos, llegando hasta aproximadamente 4,294,967,295 milisegundos (alrededor de 49 días).

Dentro del `loop()`, se evalúa si ha transcurrido el tiempo establecido en `intervalo` comparando la diferencia entre `t_actual` y `t_previo`. Si dicha condición se cumple, significa que ha pasado un segundo, por lo tanto, se actualiza `t_previo` con el valor actual para comenzar a contar un nuevo intervalo. Esto permite ejecutar acciones temporizadas sin bloquear el flujo del programa.

#### Retardos no bloqueantes

```

1 // Variable para guardar el tiempo de la ultima actualizacion
2 unsigned long t_previo = 0;
3 // Intervalo de espera en milisegundos
4 unsigned long intervalo = 500;
5 // Variable para almacenar el tiempo actual
6 unsigned long t_actual = 0;
7
8 void loop() {
9 // Tiempo actual desde que inicio el programa
10 t_actual = millis();
11
12 // Verifica si ha pasado el intervalo especificado desde la ultima vez
13 if(t_actual - t_previo >= intervalo){
14 // Actualiza el tiempo previo
15 t_previo = t_actual;
16 }
17 }
```

Para definir el LED a encender se usaron punteros basados en un arreglo. Para ellos, se define el arreglo llamado `LEDS` que contiene las referencias a los pines digitales asignados a cada LED, utilizando las constantes predefinidas de la librería. Este arreglo está definido de tal manera que sigue el orden requerido para la secuencia. A este arreglo se le asigna un puntero `led`, que apunta a la primera posición del arreglo `LEDS`.

Dentro de la función `loop()`, el puntero es referenciado con `*(led + i)`, lo que permite acceder directamente al valor del pin correspondiente.

Inicialmente, se activa el primer LED con `digitalWrite(*(led), HIGH)`. Posteriormente, en cada iteración del ciclo, se apaga el LED actual con `digitalWrite(*(led + i), LOW)`, se incrementa el contador `i`, y se enciende el siguiente LED mediante `digitalWrite(*(led + i), HIGH)`. Finalmente, si el valor de `i` alcanza el final del arreglo (valor 9), se reinicia a cero para repetir la secuencia de manera indefinida.

### Punteros

```

1 // Variable para recorrer el arreglo de LEDs
2 int i = 0;
3
4 // Arreglo con los pines digitales asignados a cada LED
5 int LEDS[9] = {
6     CONTROLLINO_D0,
7     CONTROLLINO_D6,
8     CONTROLLINO_D12,
9     CONTROLLINO_D13,
10    CONTROLLINO_D14,
11    CONTROLLINO_D8,
12    CONTROLLINO_D2,
13    CONTROLLINO_D1,
14    CONTROLLINO_D7
15 };
16
17 // Puntero que apunta al primer elemento del arreglo LEDS
18 int* led = LEDS;
19
20 void loop() {
21
22     // Enciende el primer LED
23     digitalWrite(*(led), HIGH);
24
25     // Apaga el LED actual apuntado por el puntero mas el indice i
26     digitalWrite(*(led + i), LOW);
27
28     // Aumentar el corredor de memoria (siguiente LED)
29     i = i + 1;
30
31     // Enciende el siguiente LED
32     digitalWrite(*(led + i), HIGH);
33
34     // Reinicia el indice
35     if(i == 9){
36         i = 0;
37     }
38 }
```

## IV. CONCLUSIONES

La implementación de retardos no bloqueantes mediante la función `millis()` permite mantener el bucle principal libre. Al comparar continuamente el tiempo actual con el instante de la última acción (`t\_actual - t\_previo`), el sistema programa eventos periódicos sin detener la ejecución, lo que permite atender simultáneamente tareas de lectura de sensores y comunicaciones.

El uso de variables de tipo `unsigned long` para almacenar los valores de tiempo garantiza un rango de medición de hasta 4 294 967 295 ms (aprox. 49 días) sin desbordamientos prematuros, característica imprescindible en aplicaciones embebidas que deben operar de manera continua y confiable.

La gestión de la secuencia de LEDs mediante un arreglo `int LEDS[9]` y un puntero `int* led` permite recorrer dinámicamente los pines definidos con aritmética de punteros. Con expresiones como `*(led + i)` se accede a cada LED de forma compacta y escalable, facilitando modificaciones en la longitud u orden de la secuencia sin alterar la lógica central del programa.

Al evitar la función `delay()`, se elimina cualquier bloqueo de la CPU, lo que resulta crítico en entornos de control industrial donde la latencia y la capacidad de respuesta pueden afectar la seguridad y la precisión de los procesos. Este enfoque mejora la eficiencia del código y reduce la complejidad, favoreciendo su mantenimiento y futura ampliación.

## REFERENCIAS

- [1] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.
- [2] Departamento de Ingeniería Eléctrica, Electrónica y Telecomunicaciones, *Guía de uso de Tableros v2*, 1st ed., Ing. Henry Maldonado, Ed. Cuenca, Ecuador: Universidad de Cuenca, 2025.
- [3] H. Maldonado, *Punteros y retardos no bloqueantes*, presentación, Departamento de Ingeniería Eléctrica, Electrónica y Telecomunicaciones, Universidad de Cuenca, Cuenca, Ecuador, 2025.

## V. ANEXOS

```
1 // Practica 2: Encender todos los LEDs del tablero secuencialmente
2
3 //Libreria de la tarjeta
4 #include <Controllino.h>
5
6 // Variables para control del tiempo y el indice de recorrido
7 int i = 0;
8
9 // Tiempo en el que se realizo el ultimo cambio
10 unsigned long t_previo = 0;
11 // Intervalo entre cambios en milisegundos (medio segundo)
12 unsigned long intervalo = 500;
13 // Tiempo actual proporcionado por millis()
14 unsigned long t_actual = 0;
15
16 // Arreglo con los pines digitales de los LEDs en el orden deseado de encendido
17 int LEDS[9] = {
18     CONTROLLINO_D0,
19     CONTROLLINO_D6,
20     CONTROLLINO_D12,
21     CONTROLLINO_D13,
22     CONTROLLINO_D14,
23     CONTROLLINO_D8,
24     CONTROLLINO_D2,
25     CONTROLLINO_D1,
26     CONTROLLINO_D7};
27
28 // Puntero que apunta al primer elemento del arreglo de LEDs
29 int* led = LEDS;
30
31 void setup() {
32 }
33
34 void loop() {
35     // Actualizacion del tiempo actual
36     t_actual = millis();
37
38     // Enciende el primer LED si es la primera iteracion
39     if(i == 0){
40         digitalWrite(*(led), HIGH);
41     }
42
43     // Verifica si ha transcurrido el intervalo definido
44     if(t_actual - t_previo >= intervalo){
45
46         // Apaga el LED actual
47         digitalWrite(*(led + i), LOW);
48         // Actualiza el tiempo previo
49         t_previo = t_actual;
50
51         // Incrementa el indice para apuntar al siguiente LED
52         i = i + 1;
53
54         // Enciende el nuevo LED
55         digitalWrite(*(led + i), HIGH);
56
57         // Reinicio del indice para repetir la secuencia
58         if(i == 9){
59             i = 0;
60         }
61     }
62 }
```

Listing 1. Código correspondiente a la práctica 4

El siguiente [enlace](#) corresponde al repositorio que presenta código, archivos, documentación y evidencia del funcionamiento.



Figura 9. Repositorio de prácticas