

4 Prácticas

4.1 Práctica 1: Manejo de Salidas Digitales con Controllino Mega.

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», lo que permitirá familiarizarse con el uso de variables predefinidas para manipular las salidas digitales del dispositivo.

El objetivo es comprender cómo utilizar dichas variables para generar una secuencia de encendido de focos LED, tal como se haría en una aplicación de control secuencial básica en automatización.

Librería de Controllino [🔗](#).

4.1.1 Materiales requeridos

- Tablero de control con Controllino Mega integrado.
- Fuente de alimentación del tablero.
- Cable USB tipo B 2.0.
- PC con Arduino IDE instalado y configurado para Controllino.

4.1.2 Pasos para la ejecución de la práctica

• Hardware

1. Conectar la fuente de alimentación del tablero, y presionar el switch con la etiqueta **FUENTE AC/DC**.
2. Encender el Controllino activando el switch identificado con la etiqueta **CONTROLLINO**.
3. Conectar el cable USB tipo B 2.0 desde el Controllino Mega a la PC.
4. Abrir el entorno Arduino IDE y seleccionar la placa Controllino Mega desde el gestor de placas.
5. Verificar que el puerto COM asignado corresponda al del Controllino Mega.
6. No es necesario realizar cableado adicional. En esta práctica:
 - Se utilizará las salidas digitales **D0 D1 D2 D6 D7 D8 D12 D13 D14**, cada una de ellas ya está conectada a un foco LED visible en el tablero.

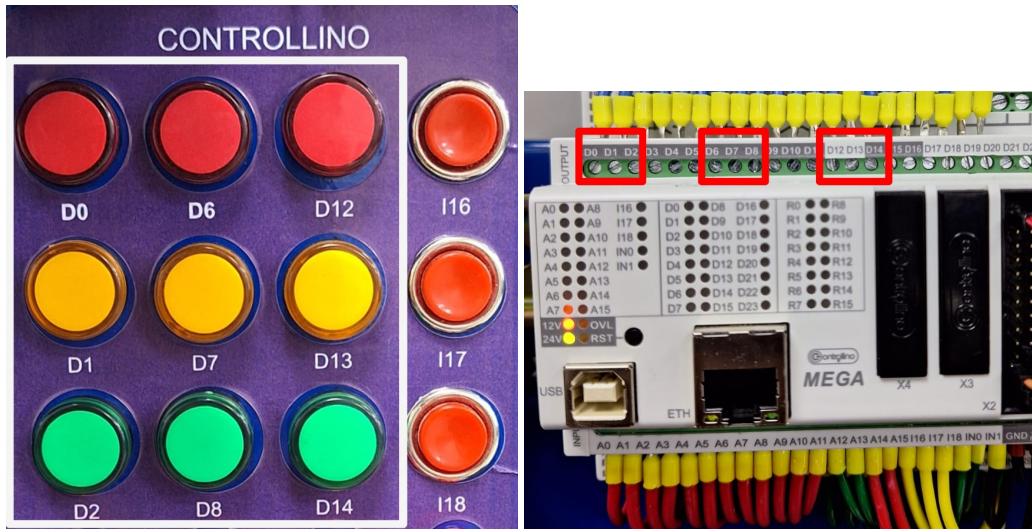


Figure 15: Correspondencia de las salidas digitales del Controllino Mega con los focos LED del tablero.

• Firmware

1. Descargar el programa ubicado en [Información Tableros/Prácticas/Practica1](#) y cargarlo al Controllino Mega desde Arduino IDE.
2. El programa se encarga de encender secuencialmente los LED conectados en los pines **D0 D1 D2 D6 D7 D8 D12 D13 D14**.
3. Explicación: La librería «Controllino.h» cuenta con variables predefinidas para el control de sus entradas y salidas. En el caso de esta práctica, solo se utiliza las salidas digitales ya mencionadas, las cuales serán llamadas mediante las variables predefinidas Controllino_D seguido del número de salida digital, por ejemplo, Controllino_D0 para usar la salida digital D0.

Ejemplo de uso de una salida digital

```

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

4. Una vez cargado el programa **Practica1.ino**, iniciará inmediatamente la secuencia de encendido de los LED.

4.1.3 Reto

Usar el Controllino Mega para controlar una matriz de 9 LEDs del tablero, dispuestos en una cuadrícula de 3x3. Los LEDs deben encenderse uno por uno siguiendo un patrón en espiral, en el siguiente orden:

LED 1	LED 2	LED 3
LED 4	LED 5	LED 6
LED 7	LED 8	LED 9

- Orden de encendido: $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 4 \rightarrow 5$ y se repite.
- Asigna un pin digital a cada LED usando variables predefinidas de la librería «Controllino.h».
- Cada LED debe encenderse durante 500 ms y luego apagarse, justo en ese instante se enciende el siguiente.
- Al finalizar, la secuencia se debe reiniciar.

Requisitos:

- Usar punteros.
- Usar retardos no bloqueantes.

4.1.4 Entregables de la práctica

Al finalizar la práctica, los estudiantes deberán presentar un informe en formato IEEE que contenga:

- Breve descripción de cómo se implementó la lógica de encendido, destacando el uso de:
 - Variables predefinidas de la librería «Controllino.h».
 - Punteros para recorrer la secuencia.
 - Técnicas de temporización no bloqueante (por ejemplo, uso de `millis()`).
- Incluir el código desarrollado del reto como anexo, con comentarios explicativos. Se valorará el uso correcto de punteros y retardos no bloqueantes.
- Evidencia fotográfica o enlace a un video corto donde se muestre los LEDs encendiéndose en la secuencia correcta.

4.2 Práctica 2: Control básico de salidas digitales y aplicación avanzada con FSM.

Esta práctica tiene como objetivo familiarizar al estudiante con el uso básico del PLC Controllino Mega, mediante el encendido y apagado de un foco LED por dos pulsantes en el tablero respectivamente.

4.2.1 Materiales requeridos

- Tablero de control con Controllino Mega integrado.
- Fuente de alimentación del tablero.
- Cable USB tipo B 2.0.
- PC con Arduino IDE instalado y configurado para Controllino.

4.2.2 Pasos para la ejecución de la práctica

• Hardware

1. Conectar la fuente de alimentación del tablero, y presionar el switch con la etiqueta **FUENTE AC/DC**.
2. Encender el Controllino activando el switch identificado con la etiqueta **CONTROLLINO**.
3. Conectar el cable USB tipo B 2.0 desde el Controllino Mega a la PC.
4. Abrir el entorno Arduino IDE y seleccionar como placa el Controllino Mega desde el menú de herramientas.
5. Verificar que el puerto COM asignado corresponda al del Controllino Mega.
6. No es necesario realizar cableado adicional. En esta práctica:
 - Se utilizará la salida digital **D0**, que ya está conectada a un LED visible en el tablero.
 - Se usará las entradas digitales **I16** e **I17**, que están conectadas a los botones del tablero.



Figure 16: Correspondencia de la salida **D0** con el foco LED del tablero.



Figure 17: Correspondencia de las entradas **I16** e **I17** con los botones del tablero.

- **Software**

1. Cargar el programa ubicado en **Información Tableros/Prácticas/Práctica2** al PLC desde el Arduino IDE.
2. El funcionamiento del programa consiste en encender el LED en **D0** si se presiona el botón **I16** y apagar el led si se presiona el botón **I17**.

4.2.3 Observaciones

- El retardo físico (rebote) del botón no afecta el funcionamiento en este caso, pero puede considerarse en prácticas futuras usando antirrebotes por software.

Control de led con dos botones.

```
1 // Practica 2: Encendido y apagado de un LED del tablero mediante  
2 // dos botones  
3 #include <Controllino.h> // Libreria de controllino  
4  
5 const int led = CONTROLLINO_D0;  
6 // Entrada I16 del controllino  
7 const int boton_encendido = CONTROLLINO_I16;  
8 // Entrada I17 del controllino  
9 const int boton_apagado = CONTROLLINO_I17;  
10  
11 void setup() {  
12     // variables de salida  
13     pinMode(led, OUTPUT);  
14  
15     // variables de entrada  
16     pinMode(boton_encendido, INPUT);  
17     pinMode(boton_apagado, INPUT);  
18 }  
19  
20 void loop() {  
21     // Encender el led con el botonI16  
22     if (digitalRead(boton_encendido) == HIGH && digitalRead(  
23         boton_apagado)==LOW) {  
24         digitalWrite(led, HIGH); // Encender LED  
25     }  
26  
27     // Apagar el led con el botonI17  
28     if (digitalRead(boton_apagado) == HIGH) {  
29         digitalWrite(led, LOW); // Apagar LED  
30     }  
31     delay(10); //retardo para evitar rebotes  
32 }
```

4.2.4 Reto

Parte A

Utilizar los tres botones del tablero de pruebas para controlar el patrón de encendido de los LED ubicados en forma de matriz 3x3. Tanto los botones como los LED ya se encuentran conectados directamente al Controllino Mega.

- Botón 1: Encendido en espiral normal.
- Botón 2: Encendido en espiral inverso.
- Botón 3: Reinicia y apaga todos los LEDs.

Parte B

Diseñar un sistema que controle dos semáforos (Semáforo A y Semáforo B) ubicados en una intersección perpendicular (Figura 18), usando el enfoque de máquina de estados finita (FSM). El sistema debe simular el comportamiento simple de los semáforos, de manera que nunca haya luz verde simultánea para ambos sentidos, y se respeten los tiempos estándar de duración de cada luz.

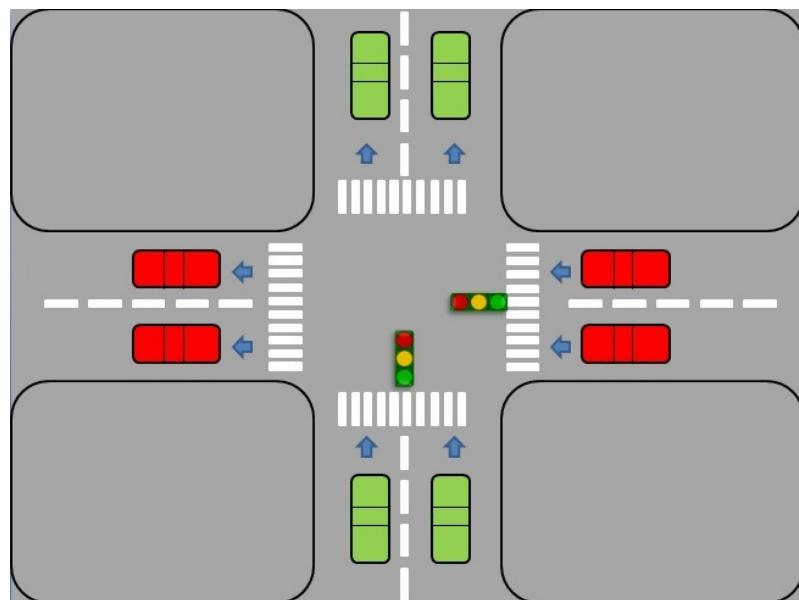


Figure 18: Circulación vehicular simple [3].

Requisitos:

- Usar retardos no bloqueantes.
- Usar estructuras.
- Usar datos tipo enum.

4.2.5 Entregables de la práctica

Al finalizar la práctica, los estudiantes deberán demostrar el funcionamiento correcto de la parte A y B, y presentar los siguientes entregables:

1. Informe en formato IEEE

El informe debe incluir las siguientes secciones:

- **Parte A: Control de LED por dos botones**
 - Explicar el uso de las variables predefinidas (por ejemplo: CONTROLLINO_D0, CONTROLLINO_I16, etc.), estructuras, punteros o técnicas implementadas.
 - Breve descripción del funcionamiento del sistema.
- **Parte B: Reto de control de semáforo con FSM**
 - Diagrama de estados del sistema.
 - Explicación del funcionamiento del sistema: estados, transiciones y condiciones.
 - Mención de aspectos clave implementados: uso de enum, estructuras y retardos no bloqueantes.
 - Añadir un enlace con un repositorio en GitHub.

2. Repositorio en GitHub

Cada equipo o estudiante deberá crear un repositorio en GitHub que contenga:

- **README principal** con una explicación detallada del desarrollo de la práctica:
 - Sección de la Parte A: describir objetivo y funcionamiento del programa.
 - Sección de la Parte B: incluir el diagrama de estados y explicar el comportamiento del sistema.
- **Estructura de carpetas organizada:**
 - Carpeta para la Parte A con el código fuente bien comentado.
 - Carpeta para la Parte B con los archivos correspondientes y el diagrama de estados.
- Incluir la guía de la práctica 2 (puede ser el archivo PDF entregado por el docente o una versión adaptada con anotaciones).

4.3 Práctica 3: Diseño de interfaz gráfica para el control de salidas en Controllino

Esta práctica tiene como objetivo controlar el brillo de un foco LED mediante modulación por ancho de pulso (PWM), utilizando una interfaz gráfica en una pantalla HMI (Human-Machine Interface). El valor del duty cycle será enviado desde la HMI al microcontrolador a través de comunicación serial, empleando la librería correspondiente. Esto permite ajustar dinámicamente la intensidad luminosa del LED desde la interfaz.

Para esta práctica se usará la HMI configurada con un widget **Spin Box**, que envía valores de 0 a 100 correspondientes al porcentaje del duty cycle, el cual ajusta el valor del PWM de una salida digital.

4.3.1 Materiales requeridos

- Tablero de control con Controllino Mega y HMI integrado.
- Fuente de alimentación del tablero.
- Cable USB tipo A a B 2.0.
- Cable USB tipo A a A.
- PC con Arduino IDE instalado y configurado para Controllino.
- Software Stone Designer GUI.

4.3.2 Pasos para la ejecución de la práctica

• Hardware

1. Conectar la fuente de alimentación del tablero, y presionar el switch con la etiqueta **FUENTE AC/DC**
2. Encender el Controllino activando el switch identificado con la etiqueta **CONTROLLINO**.
3. Conectar el cable USB tipo B 2.0 desde el Controllino Mega a la PC.
4. Encender el HMI activando el switch identificado con la etiqueta **HMI**.
5. Conectar el HMI al PC, esto se realiza conectando el cable al puerto USB tipo A del HMI, y el otro extremo del cable, conectarlo al puerto USB A del PC.

- **Diseño de la interfaz en el HMI**

1. Instalar el software STONE Designer GUI [🔗](#).
2. Abrir STONE Designer GUI e ir a la pestaña **Project** → **New Project** para crear un nuevo proyecto. Definir el nombre del proyecto, la carpeta en donde se almacenará, los baudios, las dimensiones de la pantalla y el brillo, como se muestra en la Figura 19.

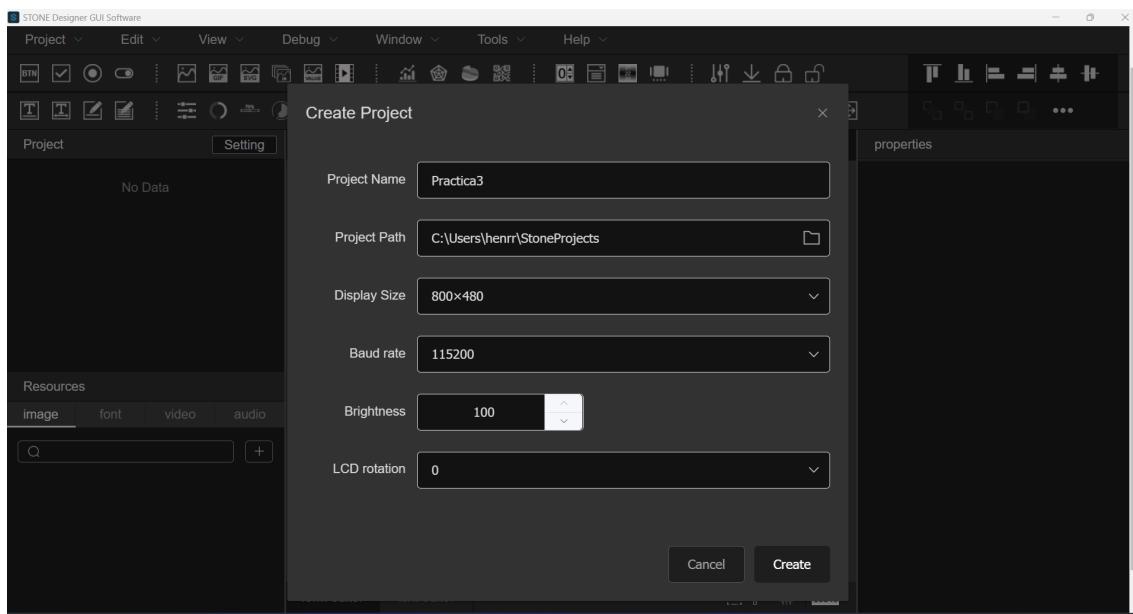


Figure 19: Configuración del proyecto en STONE Designer.

3. Al crear el proyecto se abre la ventana principal **home_page**.
4. Dar clic derecho en la ventana **home_page** → **Add widget** y agregar un widget de tipo **label** como se muestra en la Figura 20.

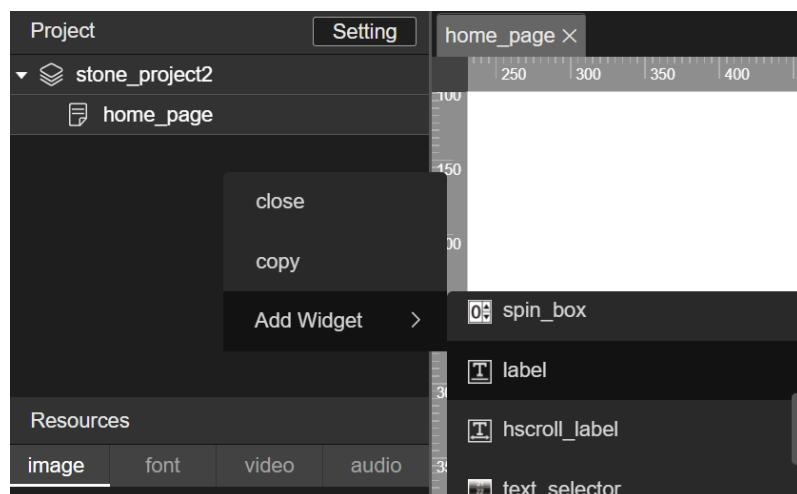


Figure 20: Añadir **label**.

5. Modificar las propiedades del widget **label** para aumentar el tamaño de la fuente. Para ello, dar clic en el widget y se abrirá la ventana de sus propiedades a la derecha.

Ajustar el tamaño del widget para que el texto ampliado se visualice correctamente, como se muestra en la Figura 21. El nombre o identificador de este widget es **label1**.

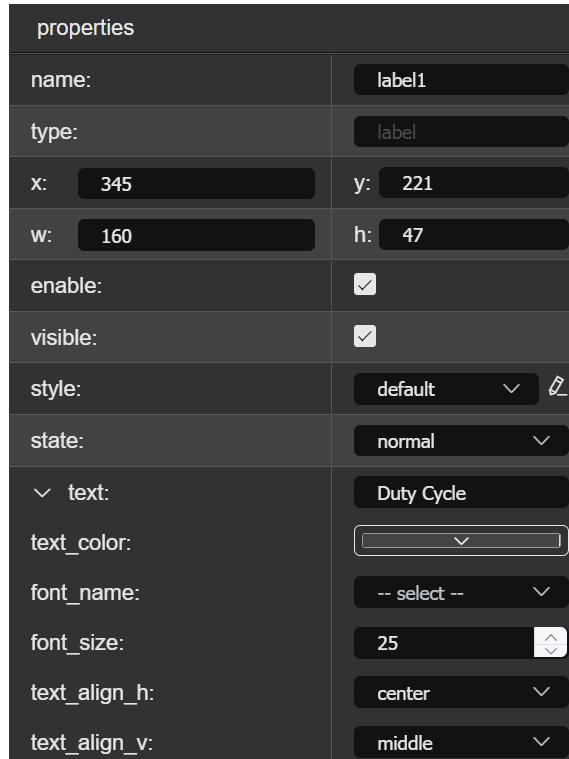


Figure 21: Configuración del **label1**.

6. Agregar un widget de tipo **spin_box** seleccionando **home_page** → **Add Widget** → **spin_box** como se muestra en la Figura 22.

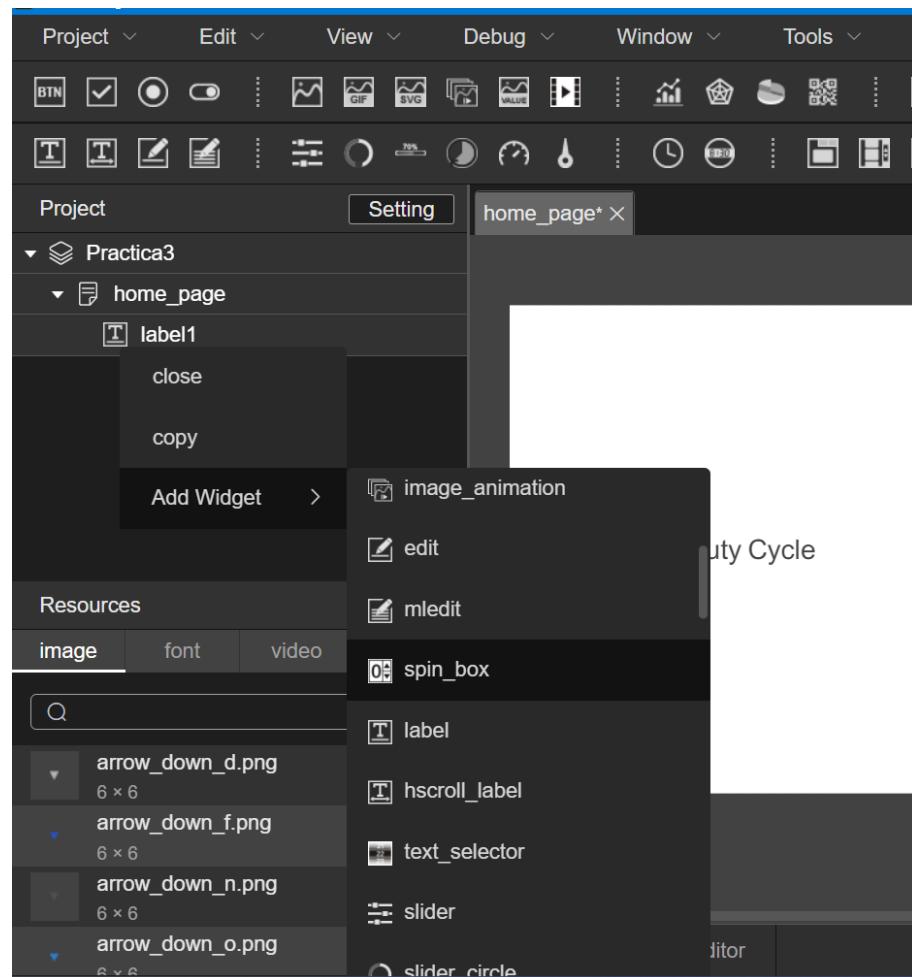


Figure 22: Añadir un widget **spin_box**.

7. Seleccionar el **spin_box** y modificar el tamaño de la fuente en la ventana de propiedades, así como también el límite de los valores permitidos y el tipo de datos (en este caso tipo **int**), como se muestra en las Figuras 23 y 24 respectivamente. Recordar el nombre de este widget, en este caso es **spin_box1**.

properties	
name:	spin_box1
type:	spin_box
x:	416
y:	193
w:	115
h:	88
enable:	<input checked="" type="checkbox"/>
visible:	<input checked="" type="checkbox"/>
style:	default <input type="button" value=""/>
state:	normal <input type="button" value=""/>
text:	0
text_color:	<input type="color"/>
selected_text_color:	<input type="color"/>
tips_text_color:	<input type="color"/>
font_name:	-- select -- <input type="button" value=""/>
font_size:	60 <input type="button" value=""/>
text_align_h:	center <input type="button" value=""/>

Figure 23: Configuración de tamaño de fuente y alineamiento.

properties	
input_type:	int <input type="button" value=""/>
min:	0 <input type="button" value=""/>
max:	100 <input type="button" value=""/>
step:	1 <input type="button" value=""/>
tips:	Please enter yo
action_text:	done <input type="button" value=""/>
keyboard:	kb_default <input type="button" value=""/>
readonly:	<input type="checkbox"/>
password_visible:	<input type="checkbox"/>
open_kb_when_blurd:	<input checked="" type="checkbox"/>
close_kb_when_blured:	<input checked="" type="checkbox"/>
animation:	<input type="button" value=""/>
key_tone:	<input type="checkbox"/>
changing_feedback:	<input checked="" type="checkbox"/>
changed_feedback:	<input checked="" type="checkbox"/>

Figure 24: Configuración de tipo de dato, valor mínimo y máximo a almacenar.

- Finalmente organizar los widgets como se presenta en la Figura 25 y guardar el proyecto.

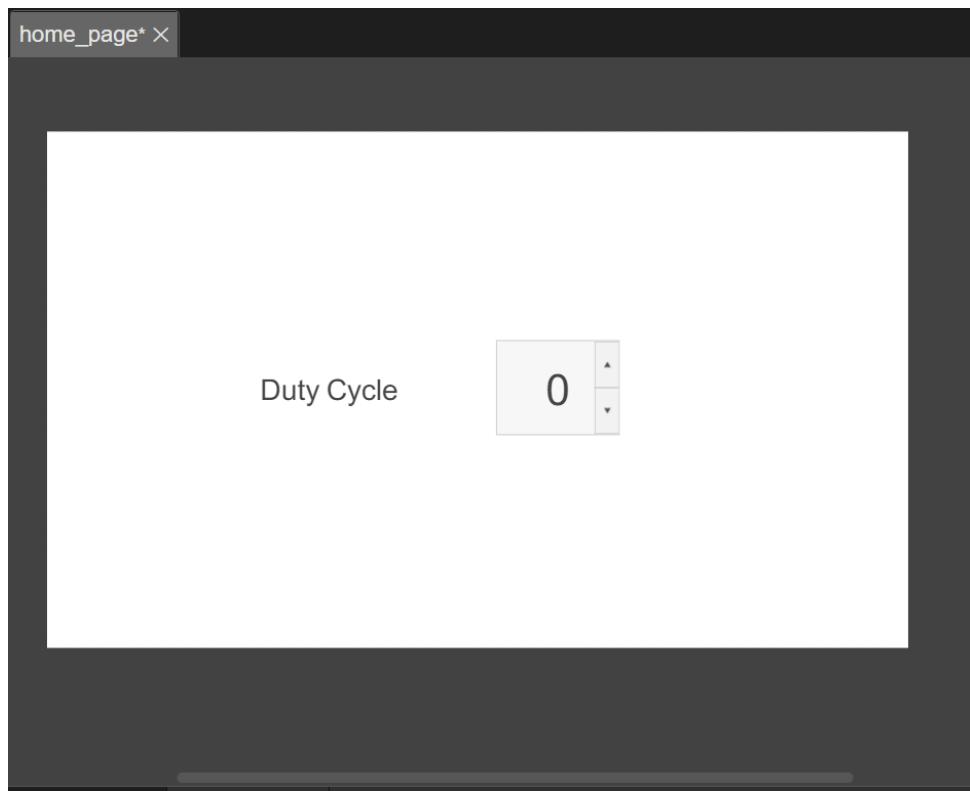


Figure 25: Widgets organizados.

- **Cargar interfaz en el HMI**

1. En la parte posterior del tablero, actuar el switch con la etiqueta **HMI** para encender el HMI.



Figure 26: Encender HMI.

2. Conectar el cable USB A en el puerto del HMI y de la computadora.



Figure 27: Cable USB conectado en el puerto USB A del HMI.



Figure 28: Cable USB conectado en el puerto USB A del PC.

3. Abrir el almacenamiento del HMI en la PC y eliminar la carpeta **default** como se muestra en la Figura 29 y reiniciar el HMI presionando el botón que se muestra en la Figura 30.

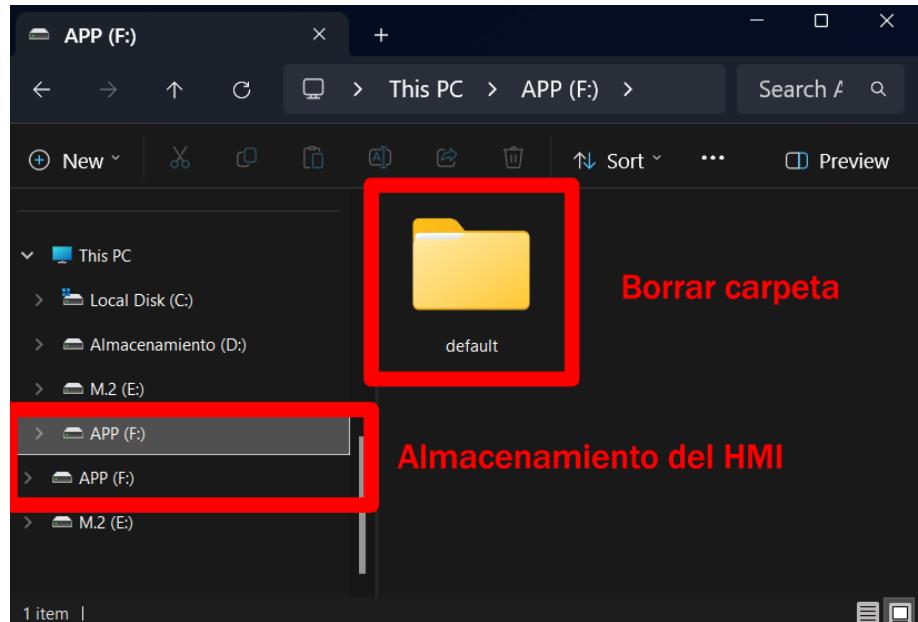


Figure 29: Eliminar carpeta **default** del HMI.



Figure 30: Botón de reinicio del HMI.

4. Abrir el proyecto **Practica3** en STONE Desginer GUI y dar clic en el ícono de **download** como se muestra en la Figura 31. Descargar el proyecto dentro del almacenamiento del HMI.

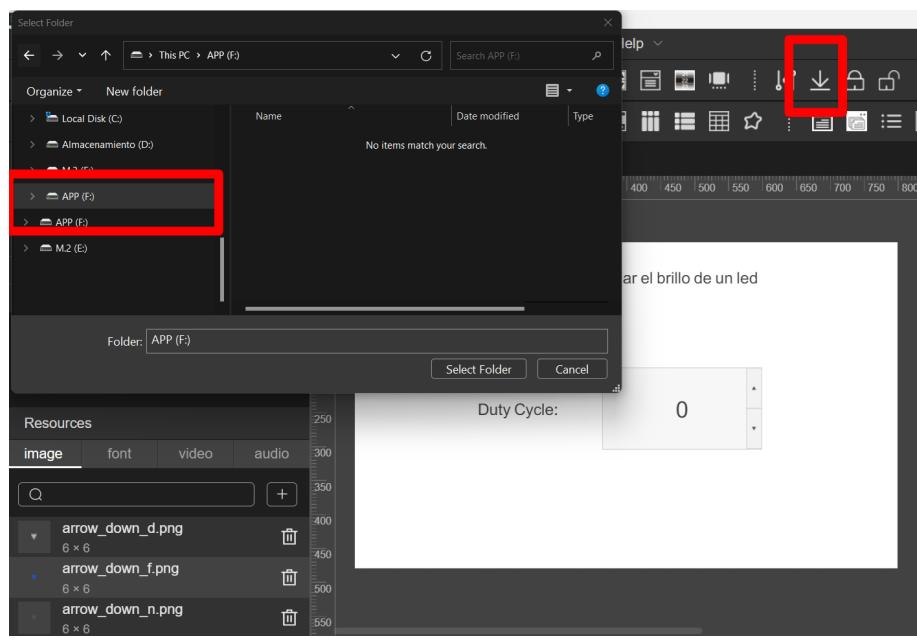


Figure 31: Descargar proyecto en el HMI.

5. Reiniciar el HMI presionando el botón que se encuentra en la parte posterior (Figura 30) y verificar que la interfaz ha cargado correctamente, caso contrario repetir el procedimiento.
6. En al Figura 32 se presenta la interfaz cargada en el HMI.

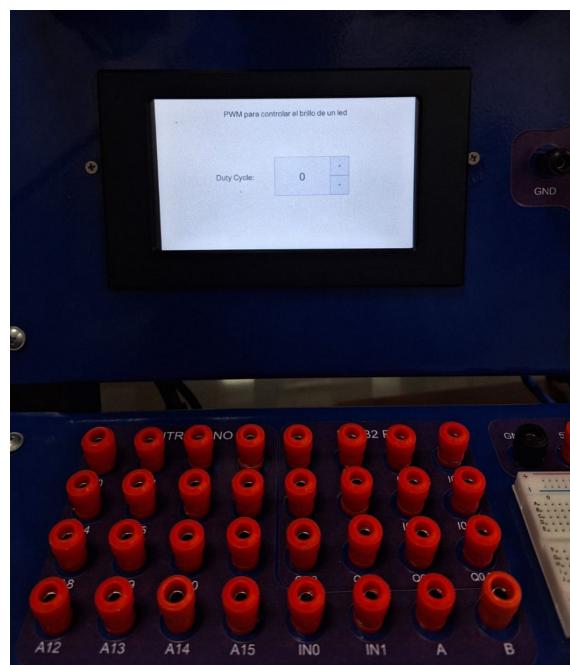


Figure 32: Interfaz en el HMI.

- **Firmware**

El firmware se encarga de obtener el valor del **spin_box** de la interfaz en el HMI, este corresponde al porcentaje de duty cycle para generar la señal PWM en el pin conectado el foco LED.

El código fuente **Practica3.ino** de esta práctica se encuentra en [Información Tableros/Prácticas/Practica3/Practica3.ino](#).

1. Abrir el programa **Practica3.ino**.
2. El programa utiliza la librería «Stone_HMI_Define.h» para obtener la trama de datos que retorna el HMI cuando se realiza alguna acción, sin embargo, la librería «Procesar_HMI.h» se encarga de procesar la trama y obtener el valor del porcentaje de duty cycle colocado en el widget **spin_box**.

El valor obtenido corresponde al duty cycle de PWM, que debe de estar en el rango de 0% a 100%. Este se mapea a valores de 0 a 255, mismos que son usados por la función `analog_write()` para PWM en el pin asignado al foco LED.

3. Para obtener la trama de respuesta del HMI, la librería Stone_HMI_Define.h requiere definir primero la estructura `hmi_msg`, que almacenará los datos recibidos desde el HMI. Esta variable será actualizada cada vez que se reciba un mensaje.
4. Se configura el pin digital **D0** del Controllino como salida, ya que se utilizará para generar la señal PWM que controla el brillo del LED.
5. A través de la función `get_value()` que recibe como parámetros el tipo de widget y su nombre, se obtiene el valor actual del widget **spin_box1**.
6. Si el valor recibido está dentro del rango válido (0 a 100), se convierte ese porcentaje a un valor entre 0 y 255 usando la función `map()`. Este valor es compatible con la función `analogWrite()`, que se encarga de generar la señal PWM.
7. Finalmente, se escribe el valor PWM al pin del foco LED y se imprime en el monitor serial el porcentaje de ciclo de trabajo y el valor PWM correspondiente.
8. Si el valor recibido no está dentro del rango permitido, se muestra un mensaje de advertencia en el monitor serial.

4.3.3 Observaciones

- La señal PWM en este caso opera con la frecuencia por defecto del pin **D0**.

4.3.4 Reto

Ampliar el programa actual para controlar dos LEDs de forma independiente, utilizando elementos de la interfaz gráfica (spin boxes) para regular el brillo de cada LED y botones físicos del tablero para encender y apagar cada uno de ellos.

Descripción de la actividad:

- Agregar un segundo widget tipo SpinBox a la interfaz gráfica. Este segundo SpinBox permitirá controlar el duty cycle (porcentaje de ciclo de trabajo PWM) del segundo LED conectado al tablero.
- Configurar dos botones físicos en el tablero:
 - El primer botón físico controlará el encendido/apagado del primer LED.
 - El segundo botón físico controlará el encendido/apagado del segundo LED.
- Cada SpinBox deberá estar asociado de manera independiente a su respectivo LED, de forma que, al mover el SpinBox, se ajuste la intensidad del brillo del LED correspondiente, sin afectar al otro LED.
- La acción de los botones físicos debe ser independiente del valor del SpinBox:
 - El botón únicamente habilitará o deshabilitará el encendido del LED, pero el brillo será determinado por el valor actual del SpinBox asociado.
- El sistema debe garantizar que, si un LED está apagado por el botón físico, no se active aunque se modifique el SpinBox correspondiente (hasta que el botón vuelva a presionarse).

4.3.5 Entregables de la práctica

Al finalizar la práctica, los estudiantes deberán demostrar el funcionamiento correcto de la interfaz, y presentar los siguientes entregables:

1. Informe en formato IEEE

El informe deberá contener los siguientes apartados:

- Descripción del procedimiento seguido para el diseño de la interfaz, incluyendo la incorporación de los widgets faltantes correspondientes a la Práctica 3.
- Breve explicación del funcionamiento general del sistema.
- Evidencia de las pruebas realizadas, incluyendo capturas del funcionamiento (por ejemplo, fotografías del LED en distintos niveles de brillo).
- Enlace a un repositorio que contenga tanto el código fuente para el Controllino como el proyecto correspondiente en Stone Designer GUI.

4.4 Práctica 4: Adquisición de datos de motor y gráfica.

Esta práctica consiste en controlar la alimentación del motor DC del Entrenador de Planta de Control (EPC) mediante una señal PWM regulada con un slider, al tiempo que se recogen los datos necesarios para calcular su velocidad en RPM y se muestran tanto el duty cycle aplicado como la velocidad del motor en una gráfica del HMI Stone.

El motor del EPC requiere una señal de control de 5 V, pero las salidas digitales del Controllino Mega funcionan a 24 V, por lo que se usa el puerto **X1**, que opera a 5 V.

En concreto, el pin **D0** de **X1** emite la señal PWM al motor DC, mientras que la salida **MDC** del encoder del EPC genera pulsos que entran por el pin **IN1** de ese mismo puerto.

Según la ficha técnica del EPC, 36 pulsos equivalen a 1 RPM; esta relación se programa en el Controllino Mega para convertir el conteo de pulsos en velocidad.

Para ello se configuran dos interrupciones:

- Una interrupción por timer de comparación que, cada segundo, recalcula las RPM del motor.
- Una interrupción externa utilizada para contar los pulsos procedentes de MDC, necesarios para calcular las RPM.

Finalmente, tanto el porcentaje de ciclo de trabajo como la velocidad resultante se representan gráficamente en el HMI.

4.4.1 Materiales requeridos

- Tablero de control con Controllino Mega y HMI integrado.
- Entrenador de planta de control EPC.
- Fuente de alimentación del tablero.
- Cable USB tipo A a B2.0.
- Cable USB tipo A a A.
- PC con Arduino IDE y Stone Designer GUI instalado y configurado.
- Software Stone Designer GUI.
- 6 jumpers macho a hembra largos.
- 6 jumpers macho a macho largos.
- Destornillador plano pequeño

4.4.2 Pasos para la ejecución de la práctica

- **Hardware**

1. Conectar el cable de alimentación del tablero.
2. Actuar el switch en la parte posterior del tablero con la etiqueta **FUENTE AC/DC** para habilitar la alimentación.
3. Actuar el switch en la parte posterior del tablero con la etiqueta **CONTROLLINO y HMI**.



Figure 33: Encender Controllino Mega y HMI.

4. Conectar el Controllino al PC.
5. Conectar el HMI al PC mediante el USB tipo A, desde el puerto USB A en la parte posterior del HMI al USB tipo A del PC.



Figure 34: Cable USB conectado en el puerto USB A del HMI.



Figure 35: Cable USB conectado en el puerto USB A del PC.

6. Realizar las conexiones que se presentan en la Figura 36, las cuales corresponden a:

- **GND** del Controlino Mega del puerto **X1** o **GND** tablero → **GND MOTOR DC** del EPC.
- Pin **D0** del puerto **X1** del Controllino Mega → **IN MOTOR DC** del EPC.
- Pin **IN1** del puerto **X1** del Controllino Mega → **MDC ENCODER** del EPC.

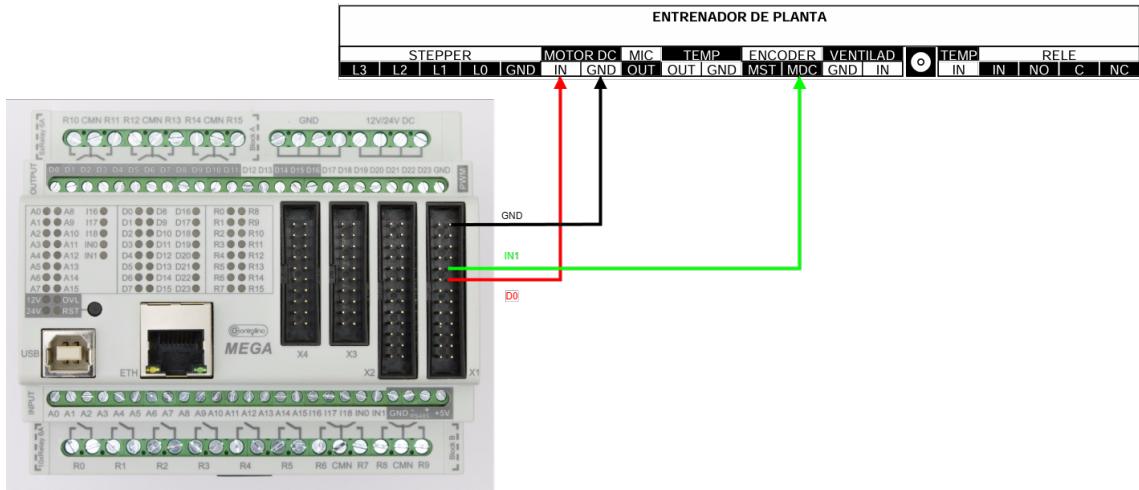


Figure 36: Conexiones desde el Controllino Mega al EPC. Se utiliza el puerto **X1** del Controllino Mega, ya que estos operan a 5V.

7. Alimentar el EPC.

- **Diseño de interfaz en el HMI**

1. Abrir Stone Designer GUI e ir a la pestaña **Project** → **New Project** para crear un nuevo proyecto. Definir el nombre del proyecto, la carpeta en donde se almacenará, los baudios, las dimensiones de la pantalla y el brillo, como se muestra en la Figura 37.

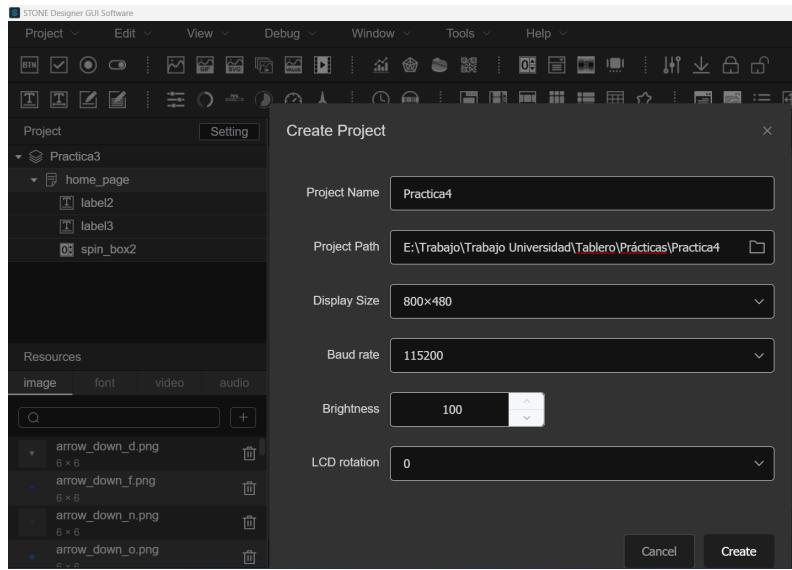


Figure 37: Configuración del proyecto en Stone Desginer GUI.

2. Dar clic derecho en la ventana **home_page** → **Add widget** y agregar un widget de tipo **slider** como se muestra en la Figura 38.

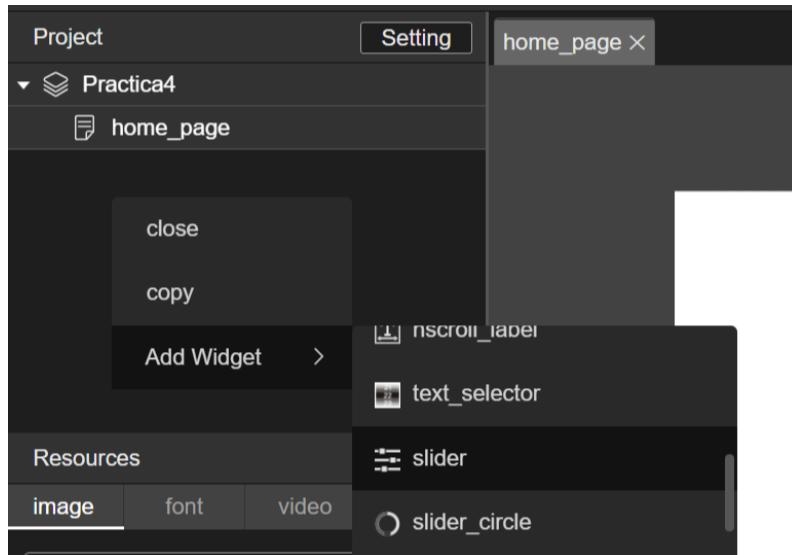


Figure 38: Agregar un widget de tipo **slider**.

3. Modificar las propiedades del widget **slider1** para que tenga los límites de valores entre 0 y 100. Además, habilitar la opción de vertical para

que el slider sea vertical. Modificar el ancho y alto del widget para que ahora funcione de forma vertical.

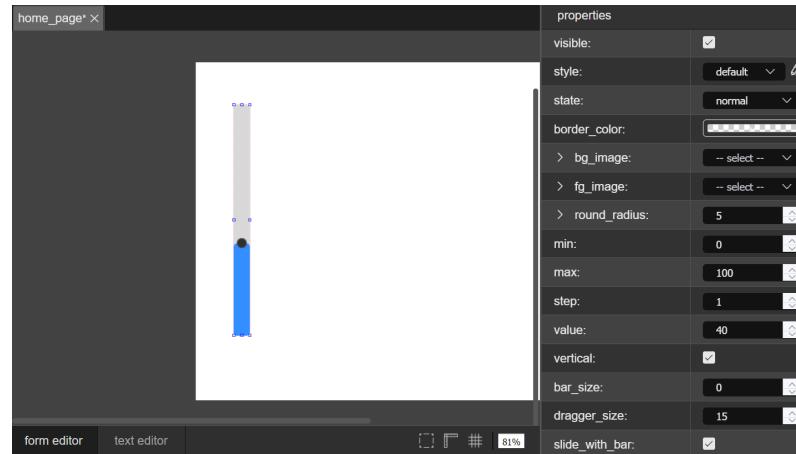


Figure 39: Modificar el **slider** para que sea vertical.

4. Agregar cinco widgets tipo **label** a la ventana **home_page** dando clic derecho en **home_page** → **Add Widget** → **label** como se muestra en la Figura 40.

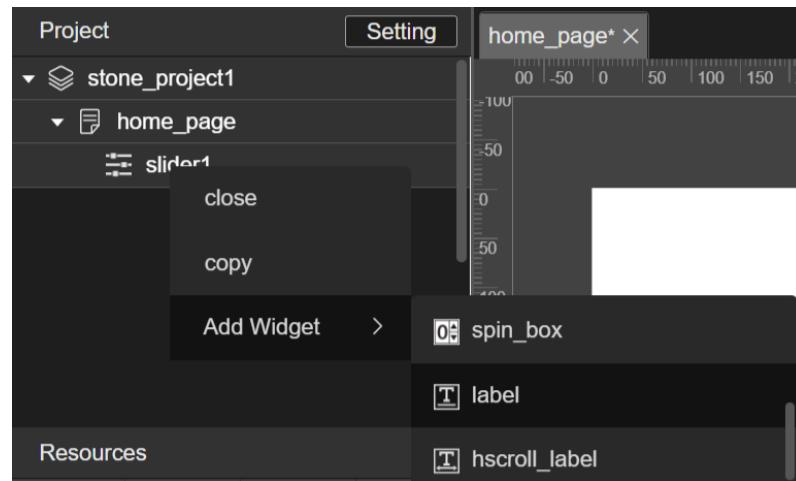


Figure 40: Agregar **label** a **home_page**.

5. Modificar los parámetros de cada uno de los **label** como se muestra en las siguientes figuras.

properties	
name:	label1
type:	label
x:	589
y:	167
w:	160
h:	46
enable:	<input checked="" type="checkbox"/>
visible:	<input checked="" type="checkbox"/>
style:	default ▾
state:	normal ▾
text:	Duty Cycle
text_color:	▼
font_name:	-- select -- ▾
font_size:	25 ▾
text_align_h:	center ▾
text_align_v:	middle ▾

Figure 41: Verificar el nombre del widget **label1**, modificar el texto a **Duty Cycle** y aumentar el tamaño del texto a **25** y del recuadro del widget para que se pueda visualizar.

properties	
name:	label2
type:	label
x:	338
y:	225
w:	50
h:	38
enable:	<input checked="" type="checkbox"/>
visible:	<input checked="" type="checkbox"/>
style:	default ▾
state:	normal ▾
text:	0
text_color:	▼
font_name:	-- select -- ▾
font_size:	25 ▾
text_align_h:	center ▾
text_align_v:	middle ▾

Figure 42: Verificar el nombre del widget **label2**, modificar el texto a **0** y aumentar el tamaño del texto a **25** y del recuadro del widget para que se pueda visualizar.

properties	
name:	label3
type:	label
x:	517
y:	128
w:	160
h:	43
enable:	<input checked="" type="checkbox"/>
visible:	<input checked="" type="checkbox"/>
style:	default ▾ 
state:	normal ▾
text:	RPM
text_color:	<input type="color"/>
font_name:	-- select -- ▾
font_size:	25  
text_align_h:	center ▾
text_align_v:	middle ▾

Figure 43: Verificar el nombre del widget **label3**, modificar el texto a **RPM** y aumentar el tamaño del texto a **25** y del recuadro del widget para que se pueda visualizar.

properties	
name:	label4
type:	label
x:	582
y:	233
w:	160
h:	51
enable:	<input checked="" type="checkbox"/>
visible:	<input checked="" type="checkbox"/>
style:	default ▾ 
state:	normal ▾
text:	0
text_color:	<input type="color"/>
font_name:	-- select -- ▾
font_size:	25  
text_align_h:	center ▾
text_align_v:	middle ▾

Figure 44: Verificar el nombre del widget **label4**, modificar el texto a **0** y aumentar el tamaño del texto a **25** y del recuadro del widget para que se pueda visualizar.

properties	
name:	label5
type:	label
X:	476
y:	53
w:	160
h:	54
enable:	<input checked="" type="checkbox"/>
visible:	<input checked="" type="checkbox"/>
style:	default ▾ 
state:	normal ▾
text:	Motor
text_color:	
font_name:	-- select -- ▾
font_size:	40 ▾
text_align_h:	center ▾
text_align_v:	middle ▾

Figure 45: Verificar el nombre del widget **label5**, modificar el texto a **Motor** y aumentar el tamaño del texto a **40** y del recuadro del widget para que se pueda visualizar.

6. Reordenar los elementos como se muestra en la Figura 46.

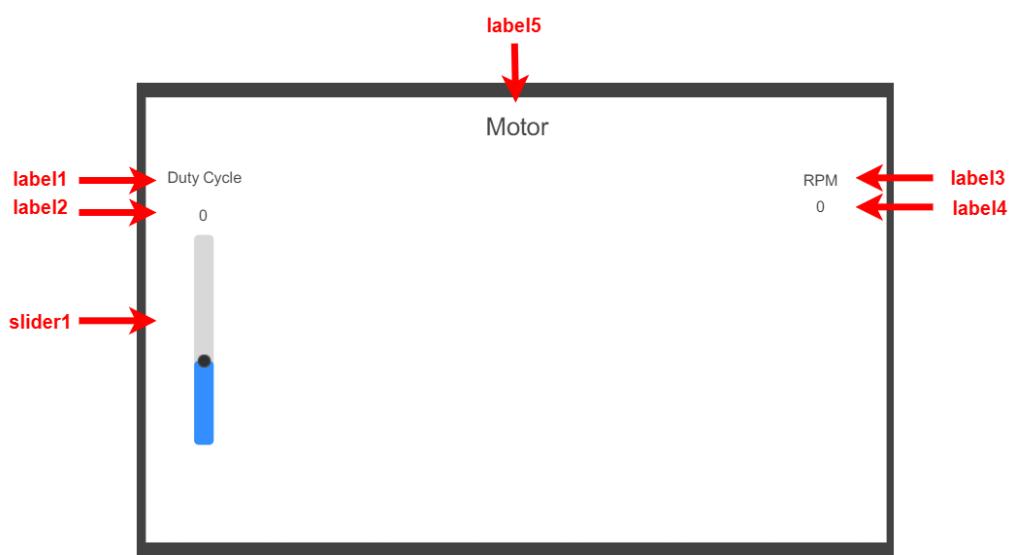


Figure 46: Ubicación de widgets.

7. Agregar un widget de tipo **chart_view** como se muestra en la Figura 47 y ubicarlo en el centro de la interfaz.

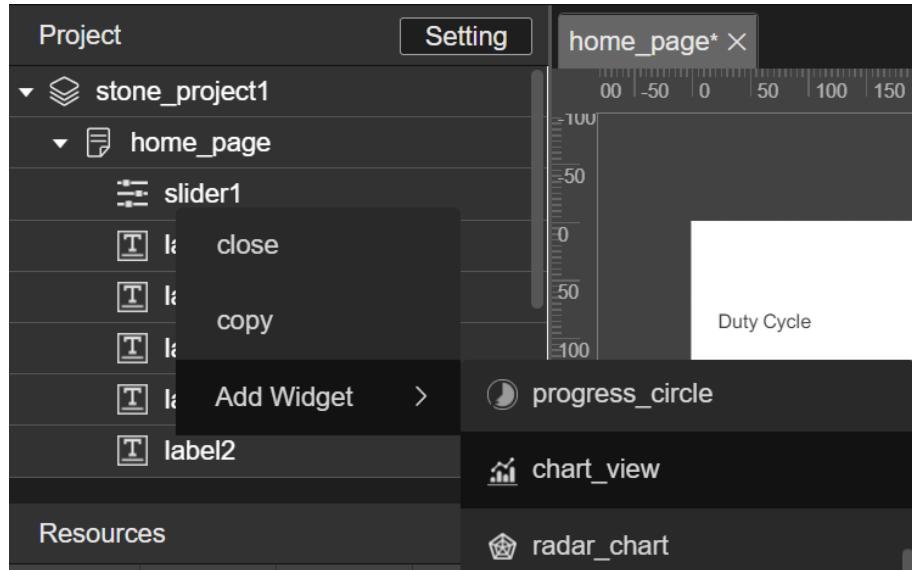


Figure 47: Agregar el widget **chart_series**.

8. Eliminar el elemento **bar_series1** del **chart_view1** (Figura 48).

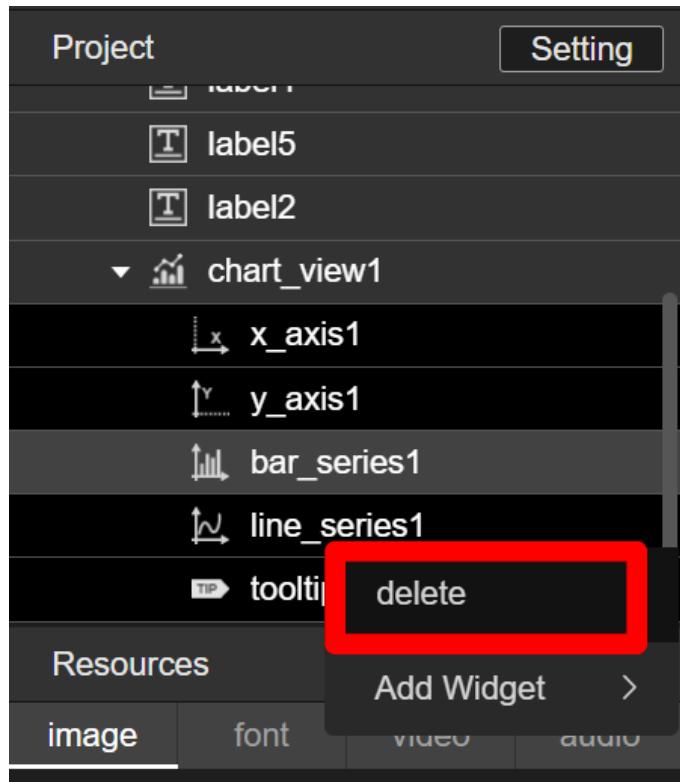


Figure 48: Borrar widget **bar_series1**.

9. Añadir un elemento tipo **lineseries** y **y_axis** al **chart_view** como se muestra en la Figura 49.

tra en las figuras 49 y 50 respectivamente, estos elemento automáticamente tomarán los nombres **lineseries2** y **y_axis2**.

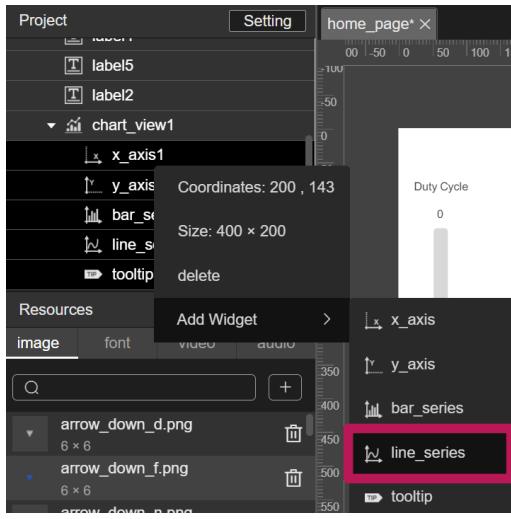


Figure 49: Agregar elemento **line_series2**.

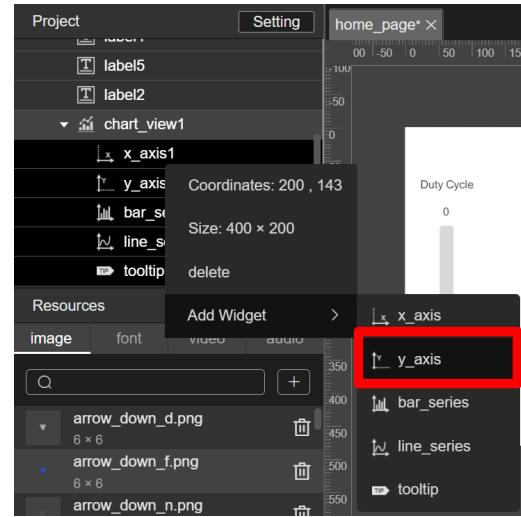


Figure 50: Agregar elemento **y_axis2**.

10. Configurar los elementos del **chart_view1** de la siguiente manera:

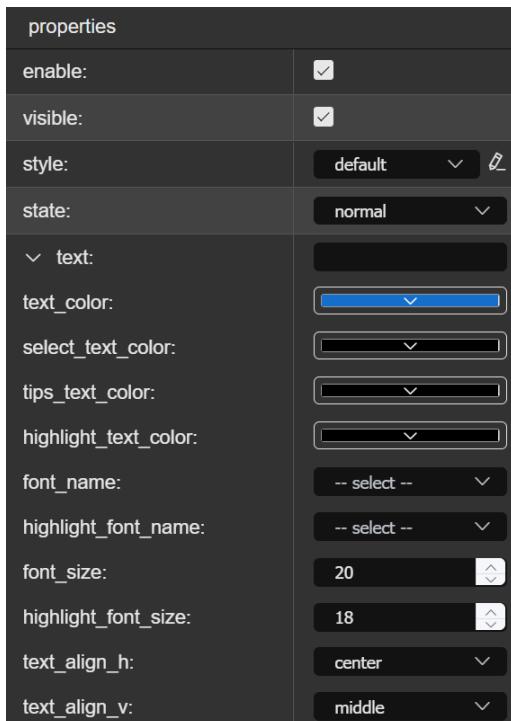


Figure 51: En el widget **y_axis1** cambiar el color del texto a **azul** y el tamaño de la fuente a **20**.

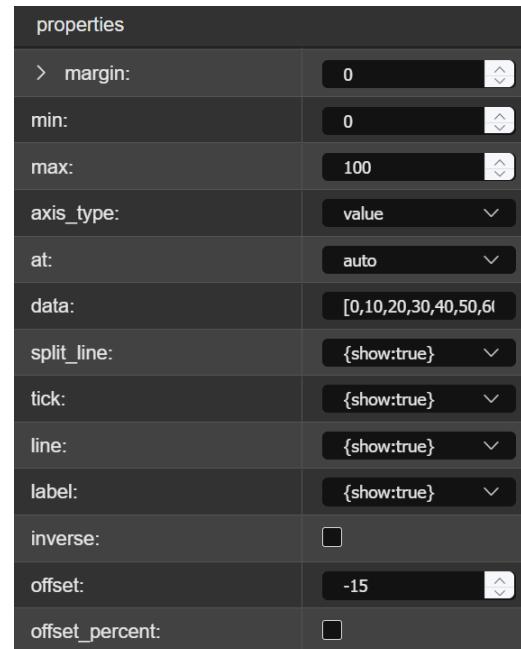


Figure 52: En el widget **y_axis1** colocar el valor mínimo a **0** y el máximo a **100**. Modificar el parámetro **data** e ingresar la secuencia [0,10,20,30,40,50,60,70,80,90,100].

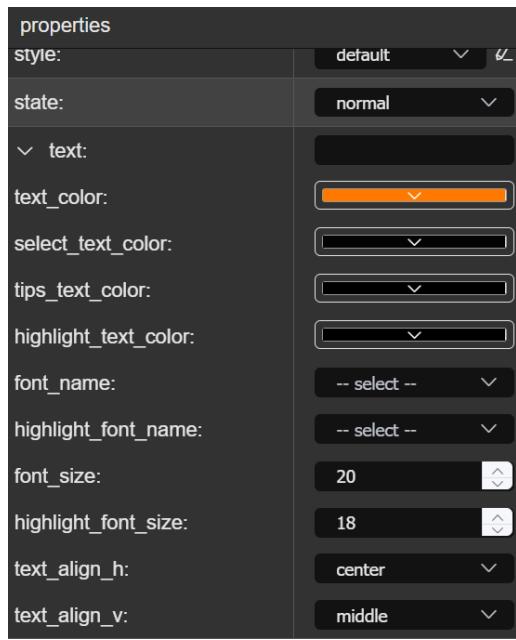


Figure 53: En el widget **y_axis2** cambiar el color del texto a **naranja** y el tamaño de la fuente a **20**.

properties	
> margin:	0
min:	0
max:	4000
axis_type:	value
at:	right
data:	[0,500,1500,1500,2000,2500,3000,3500,4000]
split_line:	{show:true}
tick:	{show:true}
line:	{show:true}
label:	{show:true}
inverse:	<input type="checkbox"/>
offset:	20
offset_percent:	<input type="checkbox"/>

Figure 54: En el widget **y_axis2** colocar el valor mínimo a **0** y el máximo a **4000**. Modificar el parámetro **data** e ingresar la secuencia [0,500,1500,2000,2500,3000,3500,4000]. Además, colocar el parámetro **at** en **right** para colocar la numeración a la derecha.

properties	
line_border_color:	<input type="color" value="#0070C0"/>
symbol_bg_color:	<input type="color" value="white"/>
symbol_border_color:	<input type="color" value="#0070C0"/>
line_border_width:	5
symbol_round_radius:	4
value:	
series_axis:	
value_axis:	y_axis1
line:	{smooth:true} ▾
area:	{show:false} ▾
symbol:	{show:false} ▾
capacity:	100
offset:	0
display_mode:	push
value_animation:	500

Figure 55: En el widget **line_series1** definir el color de la línea **line_border_color** a azul. El ancho de la línea en **5**. El parámetro **value_axis** en **y_axis1**, colocar **area** y **symbol** en **false**, **capacity** en **100** y **mode_display** en **push**.

properties	
line_border_color:	<input type="color" value="#FF8C00"/>
symbol_bg_color:	<input type="color" value="white"/>
symbol_border_color:	<input type="color" value="#0070C0"/>
line_border_width:	5
symbol_round_radius:	4
value:	0
series_axis:	
value_axis:	y_axis2
line:	{smooth:true} ▾
area:	{show:false} ▾
symbol:	{show:false} ▾
capacity:	100
offset:	0
display_mode:	push
value_animation:	500

Figure 56: En el widget **line_series2** definir el color de la línea **line_border_color** a naranja. El ancho de la línea en **5**. El parámetro **value_axis** en **y_axis2**, colocar **area** y **symbol** en **false**, **capacity** en **100** y **mode_display** en **push**.

properties	
> margin:	0
min:	0
max:	100
axis_type:	value
at:	auto
data:	[0,10,20,30,40,50,60,70,80,90,10]
split_line:	{show:true}
tick:	{show:true}
line:	{show:true}
label:	{show:true}
inverse:	<input type="checkbox"/>
offset:	0
offset_percent:	<input type="checkbox"/>
> animation:	+
key_tone:	<input type="checkbox"/>

Figure 57: En el widget **x_axis1** cambiar el tamaño de la fuente a **20**, el valor mínimo a **0**, el valor máximo a **100** y el parámetro **data** con la siguiente secuencia [0,10,20,30,40,50,60,70,80,90,10].

11. Finalmente modificar el tamaño del widget **chart_view1** al que se desee y ubicar correctamente en la interfaz como se muestra en la Figura 58.

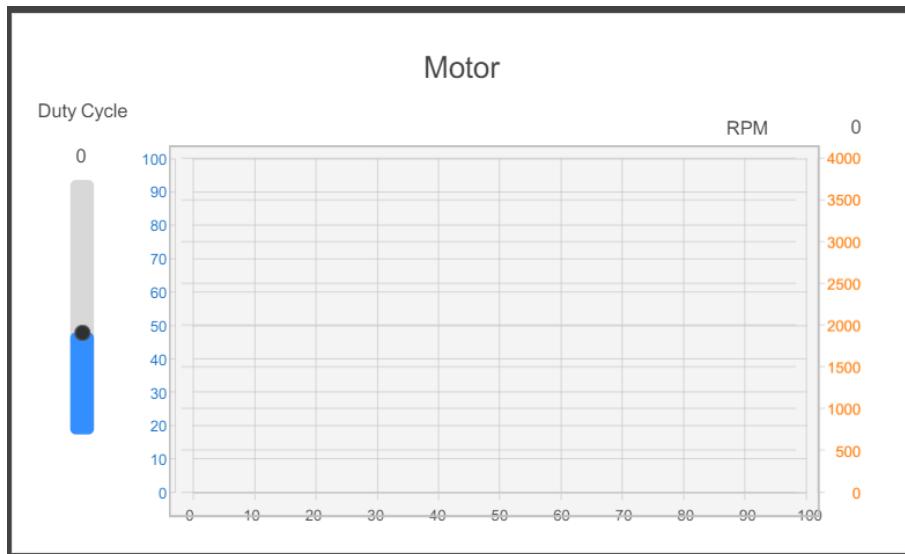


Figure 58: Interfaz finalizada.

- Cargar interfaz en el HMI

1. Abrir el almacenamiento del HMI en la PC y eliminar la carpeta **default** como se muestra en la Figura 59, presionar el botón de **Reset** como se muestra en la Figura 60.

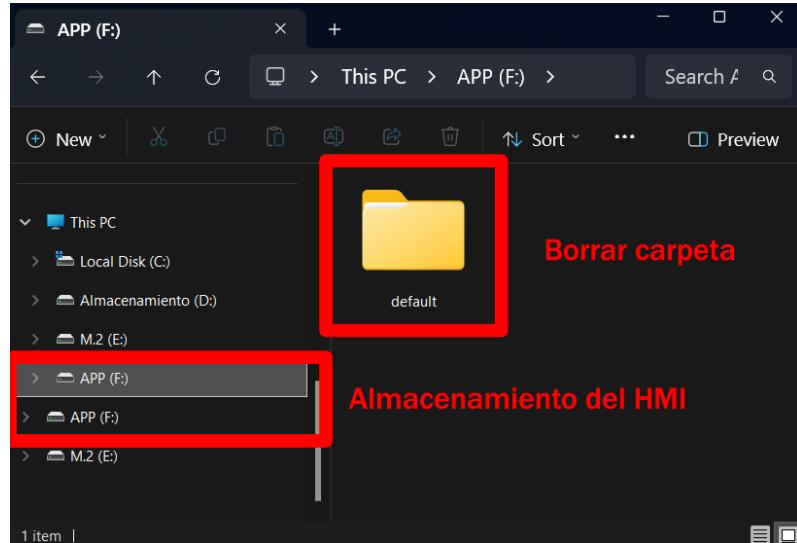


Figure 59: Eliminar carpeta **default** del HMI.

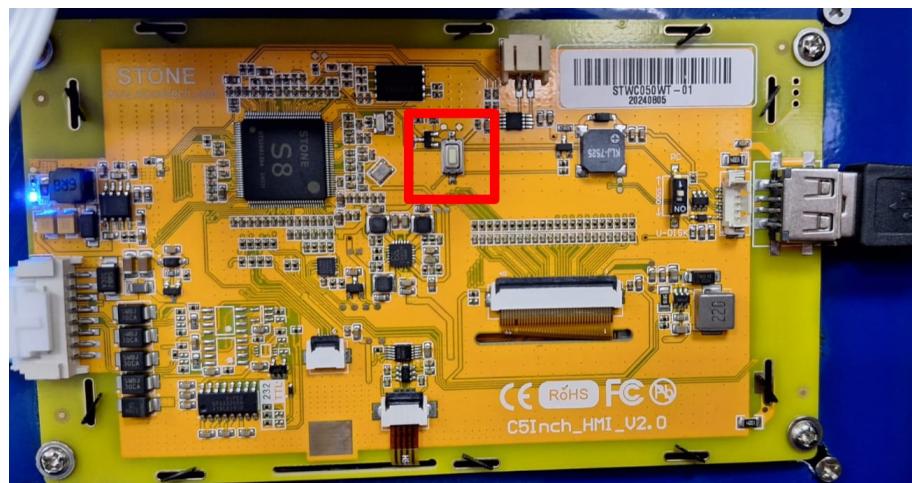


Figure 60: Botón de reinicio del HMI.

2. Abrir el proyecto **Practica4** en STONE Desginer GUI y dar clic en el ícono de **download** como se muestra en la Figura 61. Descargar el proyecto dentro del almacenamiento del HMI.

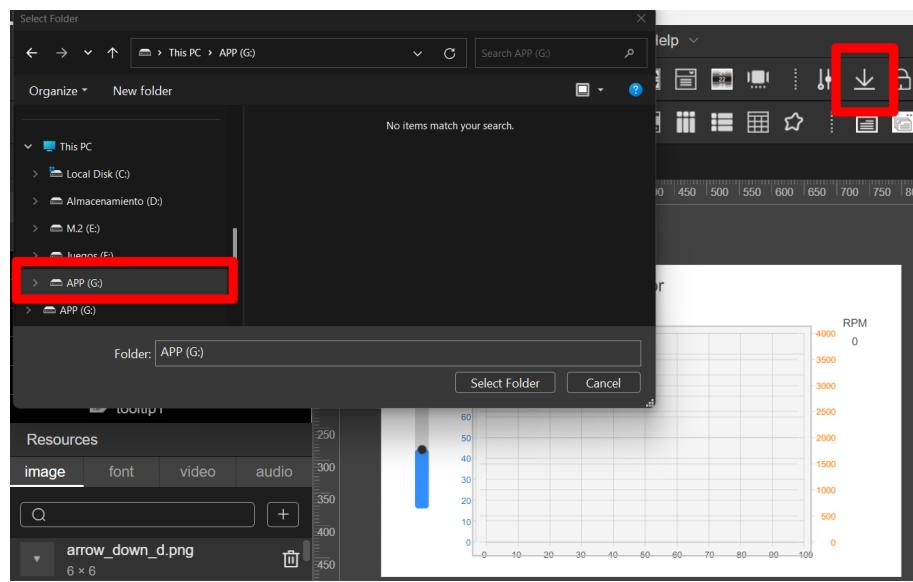


Figure 61: Descargar proyecto en el HMI.

3. Reiniciar el HMI presionando el botón que se encuentra en la parte posterior (Figura 60) y verificar que la interfaz ha cargado correctamente, caso contrario repetir el procedimiento.
4. La Figura 62 presenta la interfaz cargada en el HMI.



Figure 62: Interfaz en el HMI.

- **Firmware**

El firmware se encarga de leer un valor de duty-cycle desde un slider en la HMI, generar una señal PWM al motor en función de ese valor, contar los pulsos del encoder del motor y calcular las RPM, y enviar periódicamente datos a la HMI para actualizar gráficas y etiquetas. Para garantizar la fidelidad del conteo de pulsos, las RPM se calculan dentro de una ISR de Timer1, de modo que no se pierdan flancos, incluso si la comunicación con la HMI bloquea el loop principal.

El código fuente de esta práctica se encuentra en **Información Tableros/Prácticas/Practica4/Practica4.ino** ↗

1. **Configurar la interrupción por Timer1.** Para configurar la interrupción por timer en 1 segundo para calcular las RPM, seguimos estos pasos:

Definir el intervalo deseado. Queremos que la interrupción ocurra cada $T = 1$ s.

Calcular el prescaler y la duración de cada “tick”. El ATmega2560 funciona a $f_{CPU} = 16$ MHz, es decir, cada ciclo de reloj dura

$$T_{CPU} = \frac{1}{f_{CPU}} = \frac{1}{16\,000\,000} \text{ s} = 62.5 \text{ ns.} \quad (1)$$

Con un prescaler de 256, el timer incrementa su contador cada

$$T_{\text{tick}} = T_{CPU} \times 256 = 62.5 \text{ ns} \times 256 = 16 \mu\text{s.} \quad (2)$$

Calcular cuántos ticks caben en 1 segundo. Para que transcurra 1 s con intervalos de $16 \mu\text{s}$, se necesitan

$$\text{ticks} = \frac{1 \text{ s}}{16 \times 10^{-6} \text{ s}} = 62\,500. \quad (3)$$

Como el Timer1 es de 16 bits, puede contar de 0 a 65535, y por tanto 62500 cabe sin problema.

Configurar el modo CTC y el registro OCR1A. En modo CTC (Clear Timer on Compare match), el timer compara su contador TCNT1 con el registro OCR1A. Cuando TCNT1 = OCR1A, se dispara la interrupción TIMER1_COMPA_vect y el timer se reinicia a cero automáticamente.

Por tanto, fijamos

$$\text{OCR1A} = 62\,500 \quad (\text{ticks para 1 s}). \quad (4)$$

De este modo, cada 62500 ticks (es decir, cada 1s) el Timer1 genera la interrupción y llama a la rutina ISR(TIMER1_COMPA_vect), donde se pueden calcular las RPM.

2. **Configurar la interrupción externa para conteo de pulsos.** Utilizamos la función

```
1 attachInterrupt(digitalPinToInterrupt(entrada),  
                  contarPulso, FALLING);
```

que asocia la línea de interrupción correspondiente al pin entrada (CONTROLLINO_IN1) a la rutina contarPulso(), disparándose en cada flanco de bajada del encoder.

La función de servicio de interrupción se define así:

```
1 void contarPulso() {  
2     conteo_pulsos++; // Incrementa en 1 el contador de  
                     // pulsos  
3 }
```

Cada vez que ocurre un flanco de bajada, el microcontrolador interrumpe la ejecución del loop(), ejecuta contarPulso() y luego vuelve al punto donde estaba, garantizando que ningún pulso se pierda aún si el programa principal está ocupado.

3. **Interacción con HMI.** Para recibir los datos desde el HMI es necesario colocar en el loop() la función Uart_HMI_Data_Receive(), esta función lee por el puerto Serial2 la trama que envía la HMI y la almacena en la estructura global hmi_msg.

Para obtener los datos como enteros o flotantes del slider, se utiliza la librería Procesar_HMI.h que implementa la función get_value(), para obtener el valor del slider cada 10ms.

En cambio para enviar datos al HMI, se utiliza la librería Stone_HMI_Define.h, de la cual, para esta práctica se utiliza las funciones:

Stone_HMI_Set_Text() para modificar el contenido de los widget tipo **label**.

STONE_push_series() para enviar valores a graficar en los **line_series** del widget **chart_view**.

4. **Calculo de RPM.** Dentro de la rutina de interrupción por timer, el contador de pulsos conteo_pulsos se usa para calcular las RPM. La fórmula utilizada es:

$$RPM = \left(\frac{conteo_pulsos * 60s}{36} \right) \quad (5)$$

Que considera los 60 segundos de un minuto y los 36 pulsos que representan una revolución del motor DC, según la infografía del EPC.

Después de calcular las RPM, se resetea la variable conteo_pulsos para comenzar a contar nuevamente en el siguiente ciclo de 1 segundo.

Finalmente, las RPM calculadas se convierten a texto y se envían al HMI para que se muestren en la interfaz, junto con otros datos como el valor del slider que controla el PWM del motor.

4.4.3 Reto

Diseñar e implementar un controlador PID que regule automáticamente la velocidad del motor DC del EPC a un valor de referencia definido por el usuario mediante la interfaz HMI.

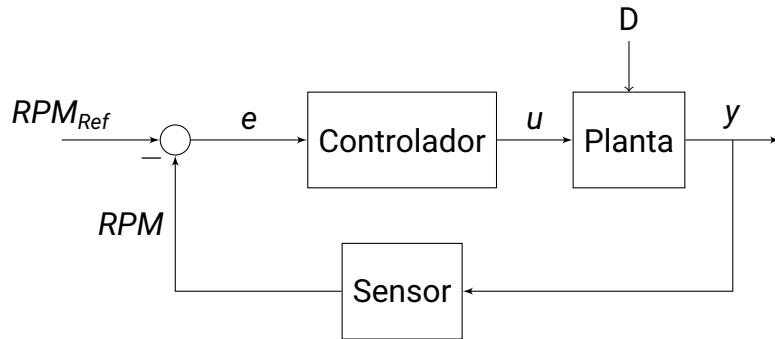


Figure 63: Esquema de lazo de control con controlador y planta

Cada grupo de trabajo deberá implementar un sistema de control PID que cumpla con las siguientes condiciones:

- La velocidad del motor debe alcanzar el valor de referencia en el menor tiempo posible, sin sobrepasarlo excesivamente.
- El sistema debe mostrar estabilidad (sin oscilaciones) y un error en estado estacionario inferior al 5%.
- El sistema debe mostrar en tiempo real: valor de referencia (setpoint) en el slider, velocidad actual (RPM), gráfica de la señal de control, comparativa entre referencia y velocidad actual.

Restricciones:

- El algoritmo PID debe ser implementado en código mediante la ecuación de recurrencias (no se permite el uso de librerías PID externas).

Pautas optionales:

- Implementar un botón de emergencia en el HMI que apague el motor.
- Agregar ajuste manual de las constantes K_p , K_i , K_d desde la interfaz gráfica.

4.4.4 Entregables de la práctica

Al finalizar la práctica, cada grupo debe demostrar el funcionamiento correcto de la interfaz y el controlador PID, además de presentar los siguientes entregables:

1. Informe en formato IEEE

El informe deberá contener los siguientes apartados:

- Descripción de los cambios realizados en la interfaz para adaptarla al reto.
- Explicación del procedimiento utilizado para obtener la función de transferencia del motor, incluyendo el uso de herramientas como LabVIEW, PLX-DAQ o MATLAB. Se deberá justificar también la elección del tiempo de muestreo programado en el Controllino.
- Descripción del proceso de ajuste de los parámetros del controlador (P, I y D), indicando si se utilizó alguna herramienta de apoyo como el PID Tuner para validar la respuesta del sistema.
- Presentar la respuesta del sistema ante una entrada escalón y frente a perturbaciones externas.
- Detalle de las interrupciones utilizadas en la implementación, junto con el cálculo y la justificación de su configuración.

A continuación, se detalla la tabla de evaluación que se utilizará para calificar el desarrollo y presentación de la práctica:

Criterio de Evaluación	Puntos
Sistema alcanza la referencia	30
Estabilidad del sistema	20
Uso correcto del HMI	15
Explicación del ajuste PID	15
Código organizado y documentado	10
Creatividad en la visualización	10
Total	100

Table 1: Rúbrica de evaluación