



22.57 - Microprocesadores y Control

Instituto Tecnológico de Buenos Aires

Documentación del equipo

Grupo 3

Profesores Daniel Jacoby

Nicolas Magliola

Alumnos Galo D'hers

Lourdes Alejandro

Lucas Neira

Tomás Torea

Fecha 20/02/2025

Estructura del programa: conexión entre drivers

A continuación, se detalla para cada driver en particular si posee o no un archivo .c y .h, además de los drivers que se incluyen en cada uno de los respectivos archivos.

Driver	Descripción	Driver.c	Driver.h
board	Definición de los puertos de la board a utilizar en cada driver	<i>No posee archivo .c</i>	gpio.h
common	Definición de macros de utilidad para el general del resto de drivers	<i>No posee archivo .c</i>	<stdint.h> <stdbool.h> <stdlib.h>
drv_adc10*	Configuración de puerto en modo analógico, junto a la posibilidad de lectura de su valor	drv_adc10.h board.h timer.h	hardware.h
drv_ds18b20	Implementación de comunicación One Wire para la posibilidad de lectura de valor de temperatura del sensor DS18B20	drv_ds18b20.h board.h gpio.h timer.h intrinsics.h	<i>No se incluyen drivers</i>
drv_eeprom	Implementación de escritura y lectura de bytes y strings en espacios de memoria de la EEPROM 24LC256 utilizando comunicación I2C	drv_eeprom.h drv_i2c.h utils.h	<i>No se incluyen drivers</i>
drv_i2c	Implementación de transmisión y recepción de bytes a través de comunicación I2C	drv_i2c.h hardware.h board.h gpio.h	<i>No se incluyen drivers</i>
drv_pwm	Configuración de puerto para PWM, junto a la posibilidad de modificar el Duty Cycle del mismo	drv_pwm.h hardware.h board.h gpio.h	<i>No se incluyen drivers</i>
drv_serial_comm	Implementación de transmisión y recepción de información a través del Serial utilizando protocolo de comunicación y comunicación UART	drv_serial_comm.h drv_uart.h utils.h protocols.h	<i>No se incluyen drivers</i>
drv_spi	Implementación de transmisión de bytes a través de comunicación SPI	drv_spi.h timer.h gpio.h board.h utils.h	<i>No se incluyen drivers</i>
drv_sw	Configuración de puerto como entrada de señal de switch / botón con posibilidad de ejecución de tarea externa en el momento de pulsación	drv_sw.h gpio.h board.h timer.h	common.h
drv_uart	Implementación de transmisión y recepción de bytes de información a través de comunicación UART	drv_uart.h hardware.h common.h	<i>No se incluyen drivers</i>

		board.h timer.h	
gpio	Implementación de configuración de puertos GPIO para distintos modos de uso.	gpio.h hardware.h	common.h
hardware	Definición de variables generales respecto al Hardware a utilizar (MSP430)	<i>No posee archivo .c</i>	common.h <msp430.h>
main	Aplicación principal: incluye el sistema de control de temperatura, comunicación con la PC, comunicación con EEPROM 24LC256, comunicación con integrado 74HC595, lectura de sensor de temperatura DS18B20, seguridad y recuperación ante pérdidas de conexión con EEPROM y aplicación de PC de escritorio.	system.h gpio.h board.h timer.h drv_sw.h drv_serial_comm.h drv_pwm.h drv_eeprom.h drv_spi.h drv_ds18b20.h	<i>No posee archivo .h</i>
protocols	Implementación de protocolo de comunicación para uso en comunicación serial: empaquetamiento de información y desempaquetamiento de información	protocols.h	<i>No se incluyen drivers</i>
system	Definición de registros referidos al hardware junto a definición de condiciones iniciales de pines de GPIO.	system.h board.h gpio.h hardware.h	common.h
timer	Configuración del watchdog timer junto a funciones útiles para trabajar con el mismo.	timer.h hardware.h	common.h
utils	Implementación de funciones de utilidad para la conversión de datos de tipos string a numérico y viceversa, además de conversión de valores tipo int a su codificación en binario	utils.h	common.h

* Este driver no se utiliza en la aplicación principal final, pero está presente en el código fuente.

Funciones principales de los drivers utilizados en main.c

- drv_adc10.h

```
@brief Devuelve el valor del ADC en formato tipo int (0 a 1023)  
unsigned int readADCvalue(void);
```

```
@brief Devuelve el valor del ADC en formato tipo int (0 a 1023)  
unsigned int readADCvalue(void);
```

- drv_eeprom.h

```
@brief Inicialización del driver  
void drvEEPROMInit(void);
```

```
@brief Escribe en el EEPROM la data proporcionada de tipo INT, FLOAT, CHAR o STR sin necesidad de aclarar la dirección en memoria ('I' - int, 'F' - float, 'C' - char, 'S' - string)
```

```
@param data_type Tipo de dato del dato a escribir
```

```
@param data Dirección de la variable que almacena el dato a escribir
```

```
@param cell_position Número de espacio en memoria asignado al tipo de variable que se desea ocupar
```

```
void writeDataIntoEEPROM(unsigned char data_type, void* data, unsigned char cell_position);
```

```
@brief Lee del EEPROM la data de tipo INT, FLOAT, CHAR o STR sin necesidad de aclarar la dirección en memoria ('I' - int, 'F' - float, 'C' - char, 'S' - string)
```

```
@param data_type Tipo de dato del dato a leer
```

```
@param receiver Receptor del dato a leer
```

```
@param cell_position Número de espacio en memoria asignado al tipo de variable que se desea ocupar
```

```
void readDataFromEEPROM(unsigned char data_type, void* receiver, unsigned char cell_position);
```

```
@brief Chequea el estado de la comunicación EEPROM
```

```
@return Devuelve COMM_EEPROM_NORMAL o COMM_EEPROM_ERROR
```

```
unsigned char checkCommStatusEEPROM(void);
```

- drv_pwm.h

```
@brief Inicialización del driver  
void drvPWMIInit(void);
```

```
@brief Modificación del DC del PWM
```

```
@param fraction Puntero a un float que representa el Duty Cycle (valor de 0 a 1)
```

```
void modifyDC(float* fraction);
```

- drv_serial_comm.h

```
@brief Inicialización del driver  
void drvSerialCommInit(void);
```

```
@brief Transmite información por Serial a través de comunicación Serial UART utilizando un protocolo
```

```
@param data_type Tipo de dato del dato a transmitir ('I' - int, 'F' - float, 'C' - char, 'S' - string)
```

```
@param data Dirección de la variable que almacena el dato a transmitir
```

```
@param code Código asociado al dato para su consecuente decodificación
```

```
void transmitDataSerialComm(unsigned char data_type, void* data, unsigned char code);
```

```
@brief Recibe información por Serial a través de comunicación UART utilizando un protocolo
```

@param data_type Tipo de dato del dato a recibir, necesario para su correcta decodificación ('I' - int, 'F' - float, 'C' - char, 'S' - string)

@param reciever Dirección de la variable que almacenará el dato ya decodificado

@param code Código asociado al dato esperado para que efectivamente se almacene la información en 'reciever'

void receiveDataSerialComm(unsigned char data_type, void* reciever, unsigned char code);

@brief Chequea el estado de la comunicación SERIAL

@return Devuelve COMM_SERIAL_NORMAL o COMM_SERIAL_ERROR

unsigned char checkCommStatusSerial(void);

- drv_spi.h

@brief Inicialización del driver

void drvSPIInit(void);

@brief Envía un único byte a través de la comunicación SPI

@param byte_data Byte a enviar a través de la comunicación SPI

void sendByteSPI(unsigned char byte_data);

- drv_sw.h

@brief Función de inicialización para el driver

void initDrvSw(void);

@brief Lee el estado actual del switch

@param pin Id del pin de acuerdo con PORTNUM2PIN()

uint8_t readSwitchState(int pin);

@brief Ejecuta una única vez la función pasada por puntero al activar el switch

@param pin Id del pin de acuerdo con PORTNUM2PIN()

@param ptr Puntero a función que tiene parámetro void y devuelve un void, es decir, una función ejecutable que no tiene parámetro y no devuelve nada.

void executeOnSwitchChange(int pin, void (*ptr)(void));

- timer.h

@brief Initialize timer and corresponding peripheral

void timerInit(void);

@brief Begin to run a new timer

@param ticks time until timer expires, in ticks

@return Timeout value

ticks_t timerStart(ticks_t ticks);

@brief Verify if a timer has run timeout

@param timeout timeout to check for expiration

@return 1 = timer expired!

char timerExpired(ticks_t timeout);

@brief Wait the specified time. Valid for delays smaller than 32 seconds

@param ticks time to wait in ticks

void timerDelay(ticks_t ticks);

- drv_ds18b20.h

@brief Inicialización del driver

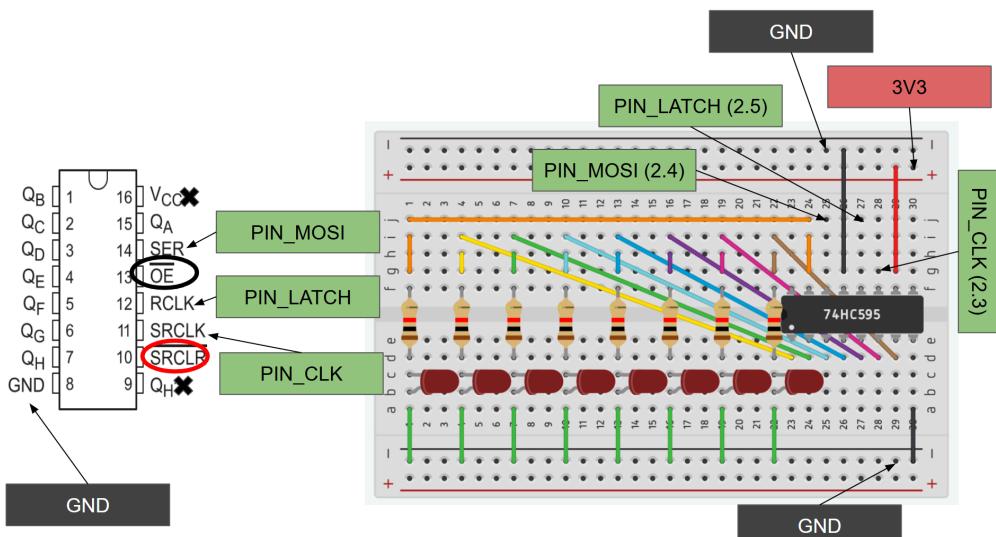
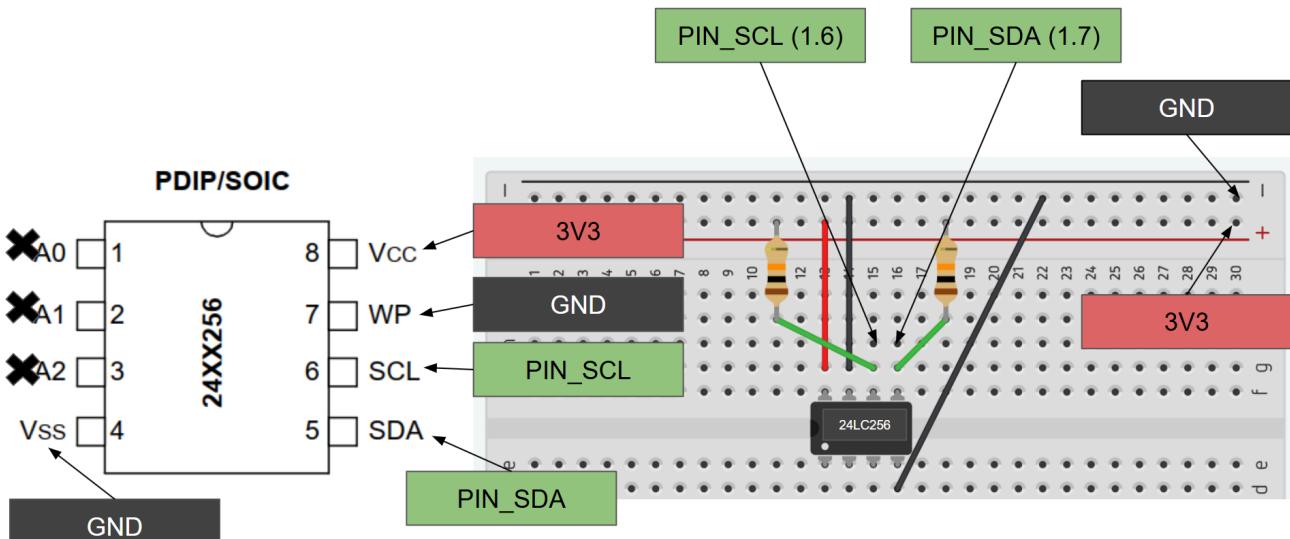
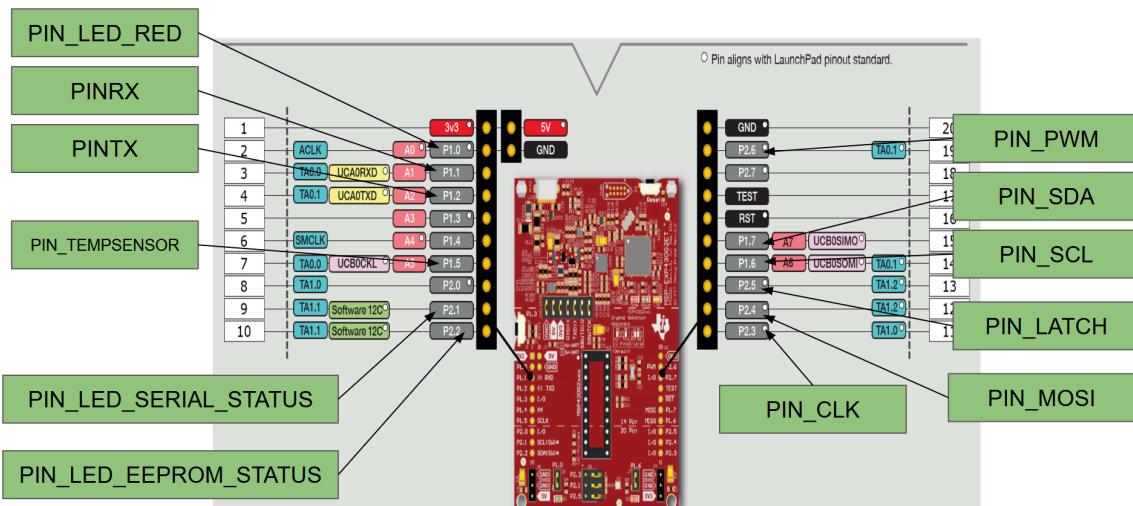
void timerInit(void);

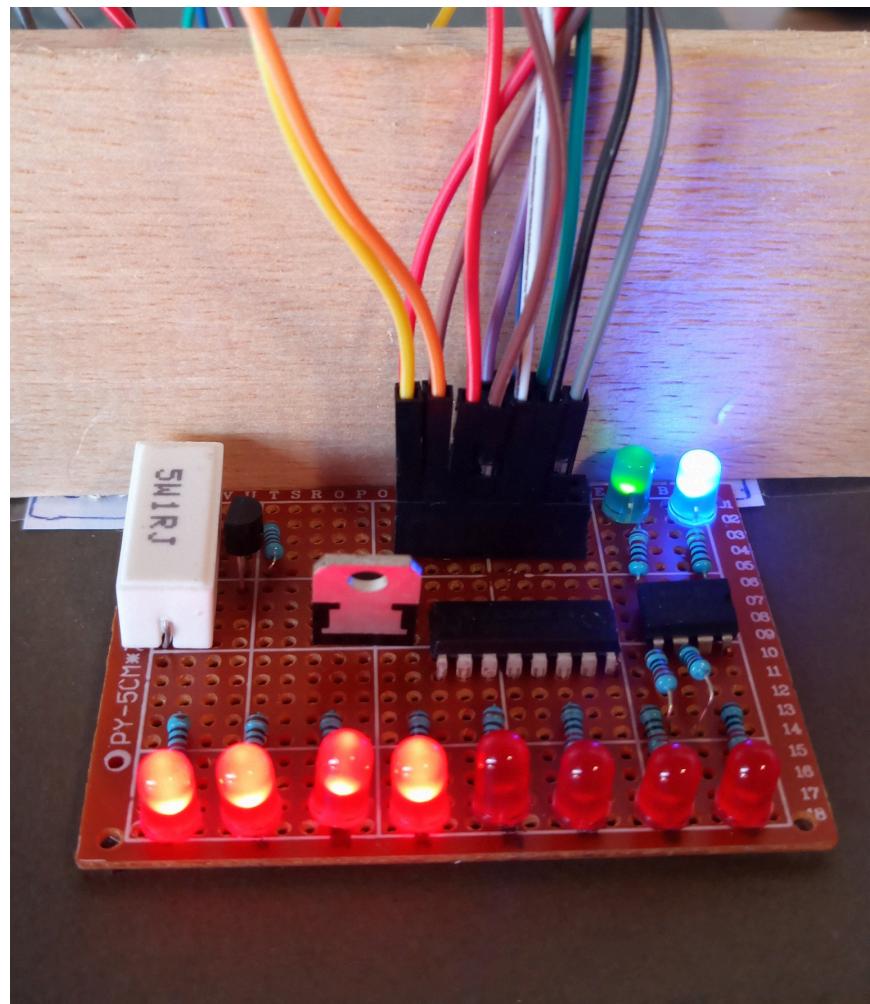
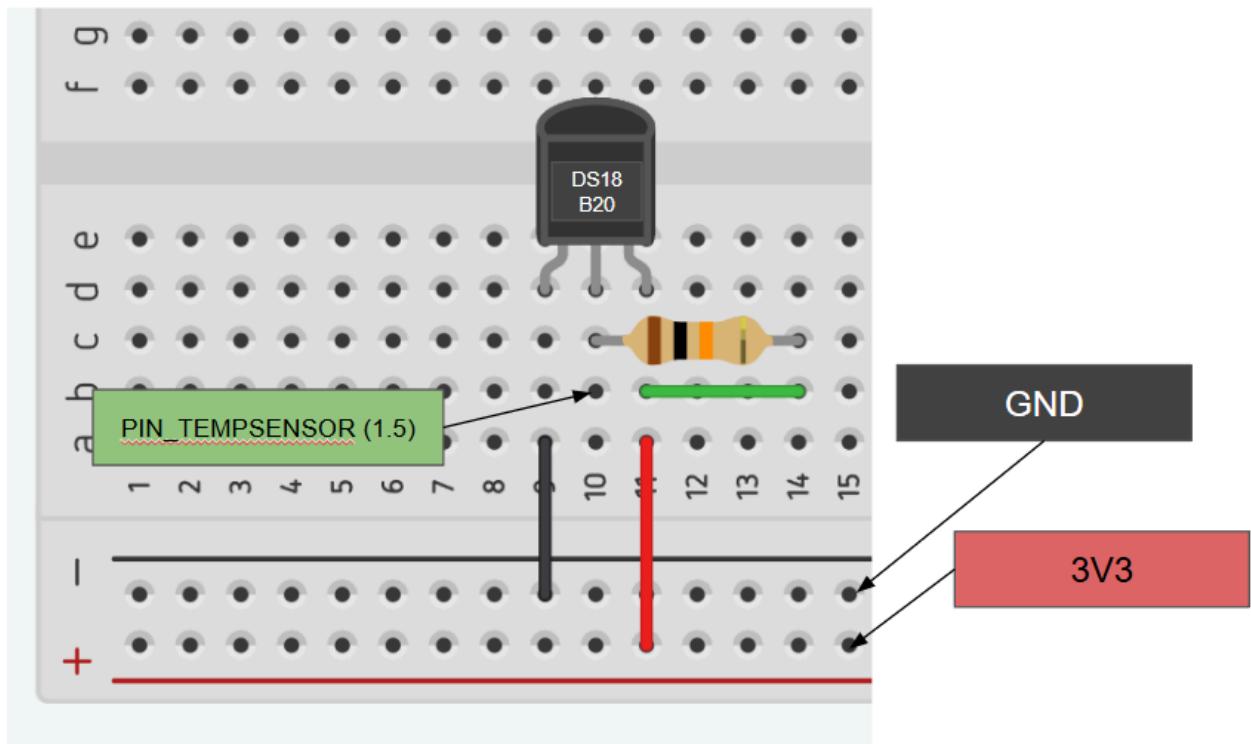
@brief Lectura del valor del último valor de temperatura registrado en grados Cº

@param reciever Variable receptor del valor de temperatura tipo float

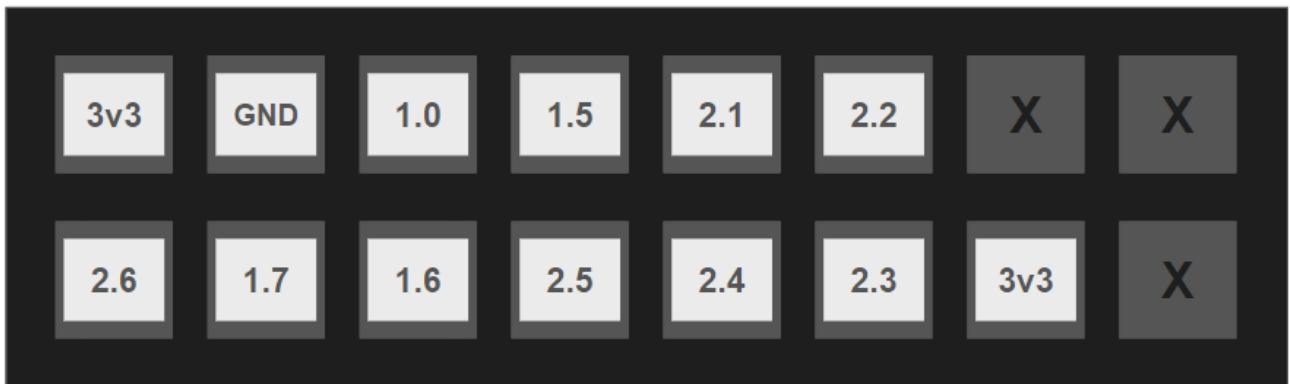
void readTemperatureDS18B20(float* reciever);

Conexiones: puertos del MSP430G2553 y PCB





donde las conexiones desde esta vista están establecidas como:



Manual de usuario

- **Luces LED individuales de la placa board**

La placa posee dos luces LED individuales. Ambas se utilizan para informar el estado de la conexión MSP430 – Serial y el estado de la conexión MSP430 – EEPROM. En ambos casos: si la luz se encuentra encendida de forma constante, entonces la comunicación entre elementos funciona normalmente; si la luz se encuentra en estado de encendido y apagado, entonces existe un problema en la comunicación.

- **Array de LEDs asociados a chip integrado 74HC595**

El array de LEDs sirve como referencia visual para interpretar el estado actual de la temperatura del resistor disipador de calor. Existen 8 LEDs en el array, en el cual se representará el rango de temperaturas entre 27.5°C y 50°C. Se toman pasos de 3.75°C. Contando de izquierda a derecha, el primer LED siempre se encuentra encendido, incluso si la temperatura es menor a 27.5°C; por otro lado, todos los LEDs estarán encendidos si la temperatura registrada es mayor a 50°C.

- **Botón integrado a la board del MSP430**

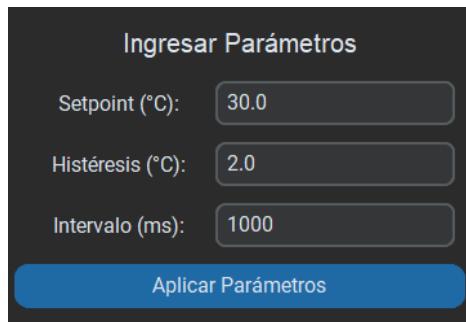
Este botón servirá para casos en donde se deba resetear las condiciones iniciales en caso de un error en la comunicación con el chip EEPROM: luego de recuperar la conexión normal entre el MSP y el EEPROM, con presionar el botón se recuperarán los valores almacenados en el EEPROM para continuar con el control de temperatura.

- **Configuración de set-point, histéresis e intervalo de transmisión de temperatura**

A través de una aplicación de escritorio puede configurarse el set-point, la histéresis y el intervalo de transmisión. Para ello el usuario dispondrá de tres cajas de entrada donde podrá ingresar cada nuevo dato. A continuación deberá seleccionar el botón “Aplicar Parámetros” y se modificarán los valores de set-point, histéresis y el intervalo de transmisión. Además, el programa verifica que los datos ingresados pertenezcan a un intervalo por seguridad, donde:

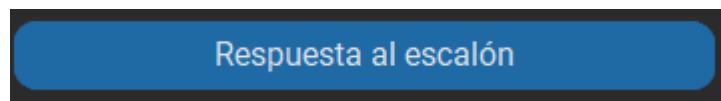
- el set-point pertenece de 20°C a 50°C

- la histéresis pertenece de 0.1°C a 10°C
- el intervalo de transmisión de datos pertenece de 250 ms a 32767 ms



- **Seteo de la respuesta al escalón (Duty-Cycle = 100%)**

A partir de la aplicación de escritorio es posible encender el calefactor. Para esto, el usuario debe presionar el botón de “Generar escalón”, lo que encenderá el calefactor indefinidamente. Cuando el usuario desee que el sistema vuelva a regirse por el sistema de control, sólo debe presionar el mismo botón y así el calefactor se encenderá y apagará en función de la temperatura sensada.



- **Aplicación**

La aplicación de escritorio, desarrollada en Python, está diseñada para facilitar la modificación de los valores de la EEPROM a través del puerto serie. Entre sus principales funciones, permite visualizar en tiempo real la gráfica de temperatura vs. tiempo, verificar la conexión del puerto serie y monitorear el estado del calefactor.

Además, cuenta con una opción para generar un escalón, manteniendo el calefactor encendido independientemente del sistema de control. Para mayor versatilidad, incluye un "modo simulación" que genera temperaturas aleatorias dentro del rango de histéresis, permitiendo probar y visualizar el funcionamiento de la aplicación sin necesidad de una conexión física.

