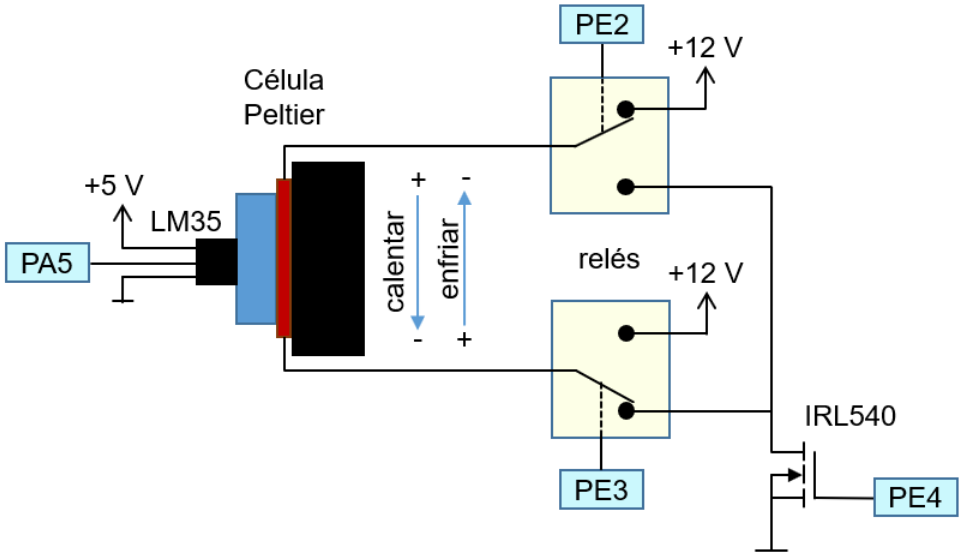
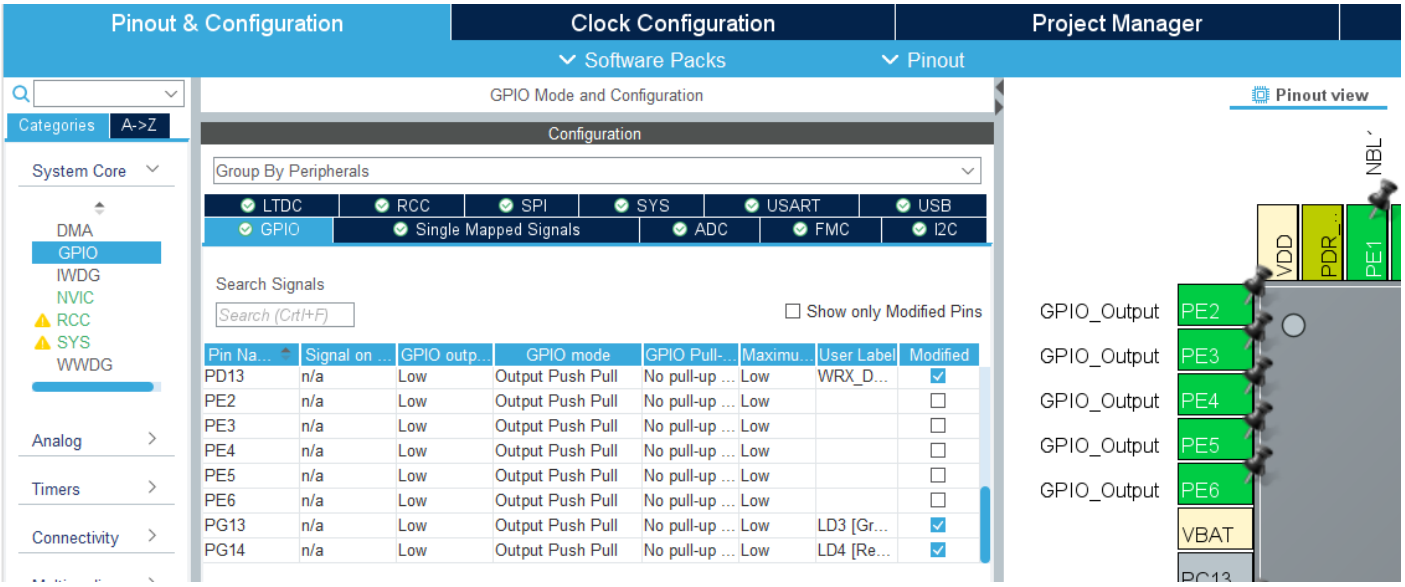
Sistemas de Control en Tiempo Real
Práctica 2

En esta práctica realizaremos un control de temperatura en una placa STM32F429I-DISC1 utilizando una célula Peltier como actuador y un sensor LM35 para medir la temperatura generada.



Elabora una aplicación en la que se realizará el control de la temperatura, de forma que el usuario pueda indicar los parámetros del controlador y la consigna, desde la pantalla LCD de la placa, pudiendo incrementar o decrementar sus valores utilizando botones. En una aplicación Qt que se ejecutará en el computador de desarrollo se recibirán datos que permitan mostrar en una gráfica la consigna, la actuación del controlador y la temperatura medida.

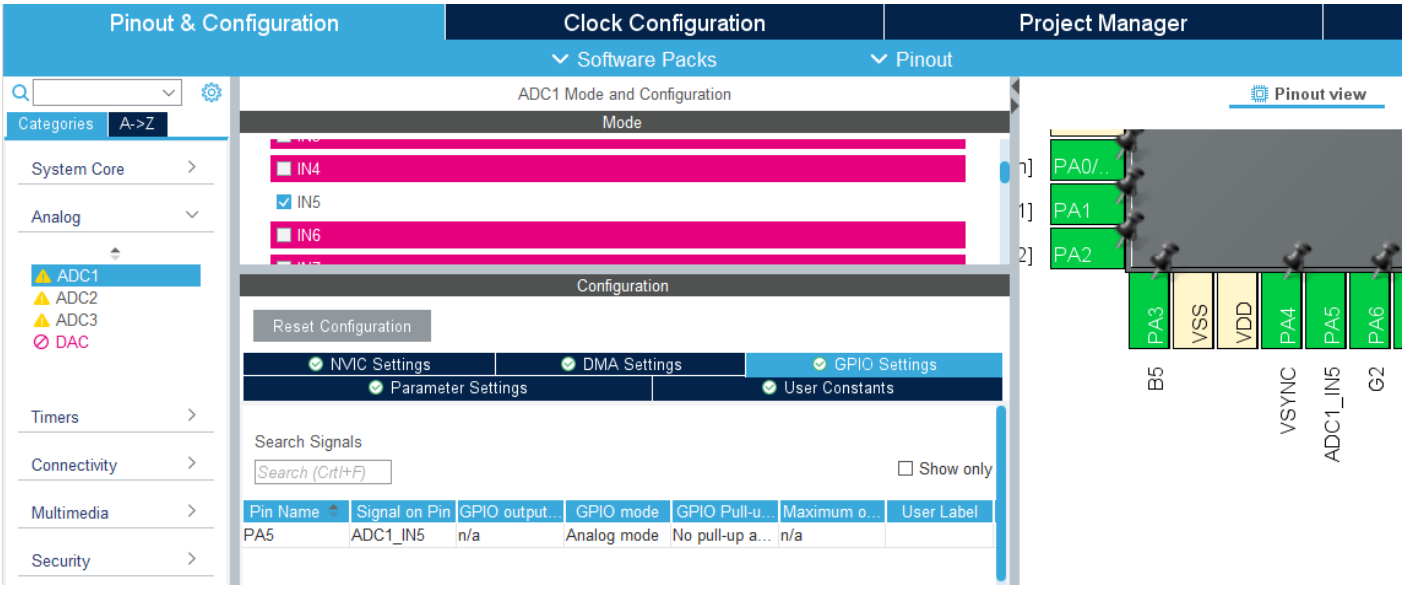
En el microcontrolador configuraremos las señales PE2, PE3 y PE4 como salidas digitales en modo push-pull para actuar sobre dos relés electromecánicos y sobre un transistor MOSFET IRL540. Mediante los dos relés se elige la polaridad aplicada a la célula Peltier para que baje o suba la temperatura de un bloque metálico donde se encuentra el sensor de temperatura. Al otro lado de la célula Peltier hay un disipador de calor con un ventilador. Mediante el transistor MOSFET se puede aplicar una actuación PWM con modulación de ancho de pulso.



Para manejar las señales digitales utilizamos, por ejemplo:

```
#include "stm32f4xx_hal.h"

HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);  // Pone PE2 a 0
HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);    // Pone PE3 a 1
```

El sensor de temperatura genera una señal analógica que se conecta a la señal PA5 del microcontrolador, y que se medirá con un convertidor AD de 12 bits con rango de tensión de 0 a 3.3V.

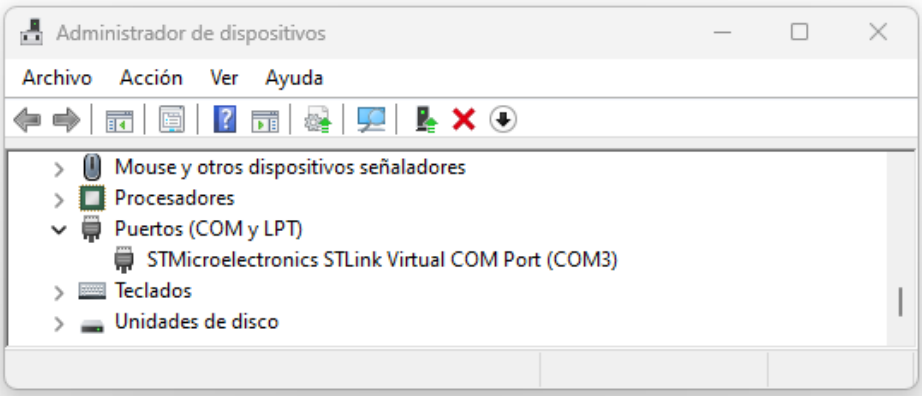


Para medir la señal del sensor hay que utilizar:

```
#include "stm32f4xx_hal.h"

HAL_ADC_Start(&hadc1); // Arranca la medida en el convertidor AD número 1 del microcontrolador
HAL_ADC_PollForConversion(&hadc1, 100);  // Espera fin de conversión con un timeout
int medida = HAL_ADC_GetValue(&hadc1);  // Recoge los 12 bits resultado de la conversión
HAL_ADC_Stop(&hadc1);  // Detiene la conversión
```

A través del mismo cable USB utilizado para alimentar y programar el microcontrolador se implanta una comunicación serie asíncrona mediante la cual se puede enviar y/o recibir información desde el computador de desarrollo.

Cuando se conecta la placa al computador, en Windows se crea un canal que se puede determinar mediante el Administrador de Dispositivos:



En este canal se utiliza una velocidad de 115200 Baud, no se utiliza paridad, con un bit de stop y transmitiendo 8 bits por byte.

Seguidamente se describe un programa codificado en C++ para el computador de desarrollo y utilizando Qt de forma que se reciben y envían datos a través de este canal.

En un proyecto de tipo Qt Widgets hay que añadir al archivo con extensión `.pro` la posibilidad de utilizar el canal serie, añadiendo `serialport` a la primera línea de texto:

```
QT       += core gui serialport
```

En el archivo `mainwindow.h` declaramos lo siguiente:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class VentanaPrincipal : public QMainWindow
{
    Q_OBJECT

public:
    VentanaPrincipal(QWidget *parent = nullptr);
    ~VentanaPrincipal();

private slots:
    void handlerRecepcion();  // Método que se ejecuta cuando finaliza una recepción en el canal
    void handlerEnvio(qint64 numBytes);  // Método que se ejecuta cuando finaliza un envío en el canal
    void botonEnvioClick();  // Se ejecuta cuando se pulsa un botón

private:
    Ui::MainWindow *ui;
    QSerialPort canal;  // Objeto para manejar el canal serie
    QByteArray bytesRecibidos;  // Colección donde se almacenan los bytes recibidos por el canal
    uint32_t n;  // Dato a enviar por el canal
};
#endif // MAINWINDOW_H
```

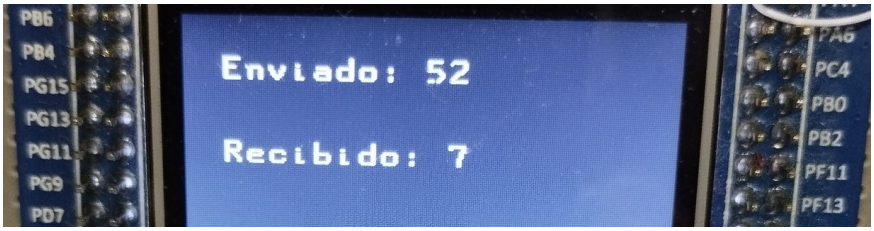
Los métodos de la clase `VentanaPrincipal` se definen en el archivo `mainwindow.cpp`:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
```

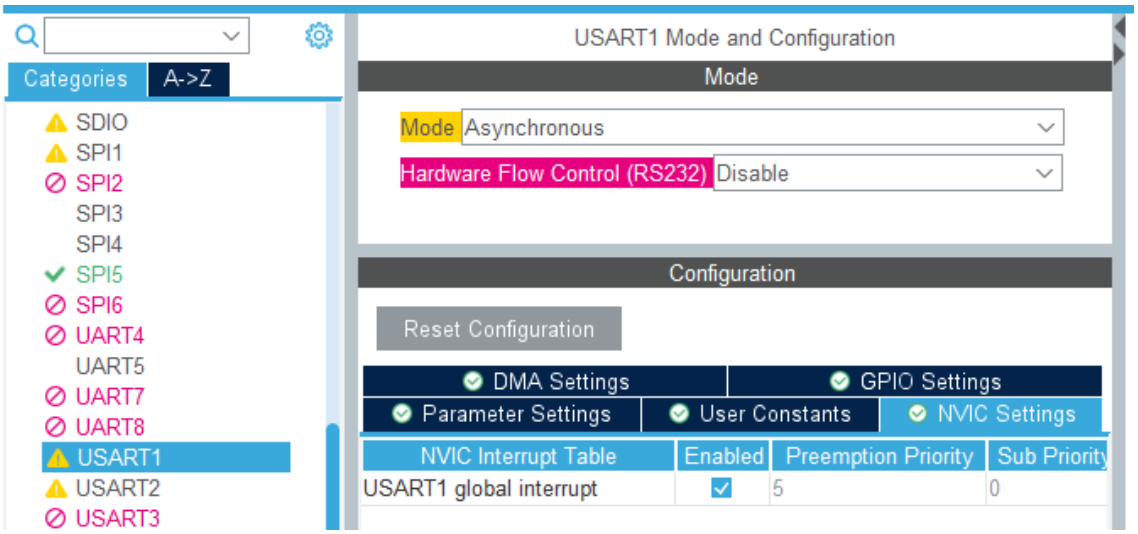
```
VentanaPrincipal::VentanaPrincipal(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow), canal() {

    ui->setupUi(this);
    n = 0;  // Inicializa el dato a enviar en el canal

    canal.setPortName("COM3");
    // Se utiliza el canal serie identificado en el sistema operativo como "COM3"

    canal.setBaudRate(QSerialPort::Baud115200);
    canal.setParity(QSerialPort::NoParity);
    canal.setStopBits(QSerialPort::OneStop);
    canal.setDataBits(QSerialPort::Data8);
    // Establece (mediante enumerados definidos en la clase QSerialPort) los parámetros de
    // comunicación: 155200 baudios, sin paridad, con un bit de parada y con 8 bits por byte

    if (canal.open(QIODevice::ReadWrite)) {  // Abre el canal y si no hay problemas ...

        canal.clear();  // Borra los buffers de recepción y transmisión del canal

        connect(&canal, &QSerialPort::readyRead, this, &VentanaPrincipal::handlerRecepcion);
        // Asocia el método handlerRecepcion de esta clase VentanaPrincipal al evento que
        // se produce cuando se ha recibido algo por el canal

        connect(&canal, &QSerialPort::bytesWritten, this, &VentanaPrincipal::handlerEnvio);
        // Asocia el método handlerEnvio de esta clase VentanaPrincipal al evento que
        // se produce cuando finaliza un envío

        connect(ui->botonEnvio, SIGNAL(released()), this, SLOT(botonEnvioClick()));
        // Utiliza el método connect() recibido mediante herencia desde la clase base QMainWindow para
        // conectar el evento que se produce cuando se pulsa sobre el botón con el método
        // botonEnvioClick()

    } else ui->etiquetaRecepcion->setText("Error abriendo canal");
    // Si no se puede abrir el canal, muestra un mensaje de error en la ventana
}


void VentanaPrincipal::handlerRecepcion() {
    // Se ejecuta cuando se recibe algo en el canal

    bytesRecibidos.append(canal.readAll());
    // Añade todos los bytes recibidos en el canal a la colección bytesRecibidos

    if (bytesRecibidos.contains('\n')) {  // Si se ha recibido un salto de línea '\n' ...

        QString cadena(bytesRecibidos);
        // Convierte los bytes recibidos a una cadena de caracteres

        cadena = cadena.trimmed();
        // Elimina posibles separadores al comienzo y al final de la cadena. Por ejemplo,
        // espacios en blanco y caracteres retorno de carro '\r' y alimentación de línea '\n'

        ui->etiquetaRecepcion->setText("Recibido: " + cadena);
        // Muestra el texto recibido en un QLabel de la ventana

        bytesRecibidos.clear();  // Borra la colección de bytes para preparar otra recepción
    }
}


void VentanaPrincipal::handlerEnvio(qint64 numBytes) {
    // Se ejecuta cuando finaliza un envío. En el parámetro numBytes se indica el número
    // de bytes enviados

    ui->etiquetaEnvio->setText("Enviado: " + QString::number(n));
    // Visualiza en un QLabel el dato enviado
}
```

```
void VentanaPrincipal::botonEnvioClick() {

    n++;  // Se incrementa cada vez que se pulsa el botón


    QByteArray bytes;  // Objeto que permite manejar una colección de bytes

    bytes.append((char*)&n, sizeof(uint32_t));
    // Añade los bytes de la variable n. El primer parámetro es un puntero de tipo char* que
    // apunta a una dirección de la cual se recogen tantos bytes como se indica en el segundo
    // parámetro

    canal.write(bytes);
    // Envía los bytes a través del canal
}


VentanaPrincipal::~VentanaPrincipal() {
    delete ui;
}
```

En la placa STM32F429I-DISC1 preparamos una aplicación que lanza una tarea en la que se enviará contínuamente a cada segundo por el canal serie asíncrono un entero de 32 bits que se irá incrementando a cada envío. También se recibirá mediante interrupción un entero de 32 bits que se mostrará en pantalla:



Para que la recepción en el canal se pueda tratar mediante interrupción en el microcontrolador, hay que habilitar **`USART1 global interrupt`** en **`Connectivity - USART1`**



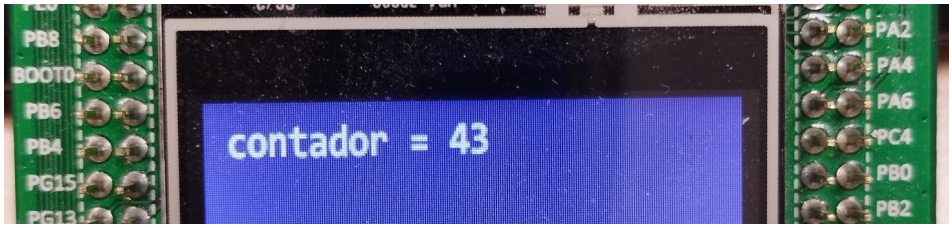
La transmisión y recepción se programa de la siguiente forma:

```
/* USER CODE BEGIN 4 */

uint8_t bufferRecepcion[10];
// Buffer de 10 bytes donde la rutina que atiende a la interrupción del canal
// guarda los bytes recibidos

uint32_t datoRecibido;  // Variable entera de 32 bits donde se va a guardar el dato recibido

int hayRecepcion = 0;  // Buleano que indica si se ha recibido algún dato


void HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart) {
    // La rutina que atiende a las interrupciones del canal llama a esta función cuando
    // se hayan recibido el número de bytes estipulados

    memcpy(&datoRecibido, bufferRecepcion, sizeof(uint32_t));
    // Copia los 4 bytes recibidos en la variable datoRecibido

    hayRecepcion = 1;  // Indica que se recibió un dato

    HAL_UART_Receive_IT(&huart1, bufferRecepcion, sizeof(uint32_t));
    // Con esta llamada se indica que cuando se reciben otro 4 bytes, que se guardarán
    // en bufferRecepcion, se vuelva a llamar a esta función HAL_UART_RxCpltCallback
}

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
  * @brief  Function implementing the defaultTask thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument) {
  /* init code for USB_HOST */
  MX_USB_HOST_Init();
  /* USER CODE BEGIN 5 */

    PantallaLCD pantalla;
    inicializaPantalla2Buffers(1, 0xFF000000, NULL, &pantalla);
    // Inicializa la pantalla para utilizar doble buffer y un fondo negro

    JuegoCaracteres jc;
    inicializaJuegoCaracteres(8, 11, 3, 0xFFFFFFFF, juego8x11, &jc);
    // Inicializa un juego de caracteres para escribir texto en blanco

    char cadena[30];  // Cadena de caracteres para preparar textos

    int contador = 0; // Contador cuyo valor se va a transmitir en el canal

    HAL_UART_Receive_IT (&huart1, bufferRecepcion, 4);
    // Prepara la recepción para tratarla mediante interrupción de forma que cuando se reciban
    // 4 bytes y se hayan guardado en bufferRecepcion, a continuación haya una llamada a la
    // función HAL_UART_RxCpltCallback desde la rutina que atiende a las interrupciones del canal

    while(1) {  // Repite indefinidamente ...

        osDelay(1000);  // Espera bloqueado durante 1000 milisegundos

        contador++;  // Incrementa el contador

        sprintf(cadena, "%d\r\n", contador);
        // Prepara en una cadena de caracteres el valor del contador, terminándolo con un carácter
        // '\r' de retorno de carro y un carácter '\n' de alimentación de línea

        if (HAL_UART_Transmit(&huart1, (uint8_t*)cadena, strlen(cadena), 5000) != HAL_OK)
```

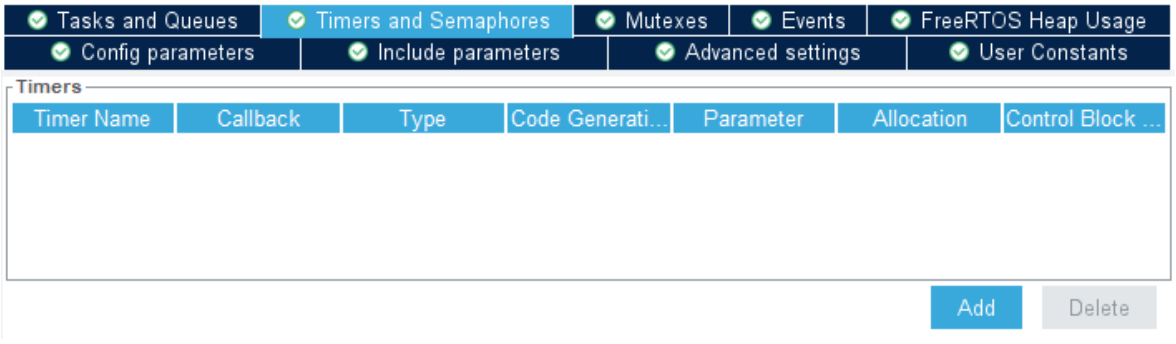
```
            Error_Handler();
        // Transmite la cadena de caracteres por el canal serie asíncrono implantado en el cable USB.
        // Se fija un timeout de 5000 ms, de forma que si la transmisión no finaliza antes de ese tiempo,
        // se produce un error que provoca la ejecución de la función Error_Handler() donde el programador
        // puede añadir instrucciones que tratan posibles errores de este tipo.

        sprintf(cadena, "Enviado: %d", contador);
        dibujaCadenaCaracteres(20, 20, cadena, &jc, 0, 100, 1, &pantalla);
        // Visualiza en la pantalla LCD un mensaje indicando el valor del contador

        if (hayRecepcion) {
            sprintf(cadena, "Recibido: %d", (int)datoRecibido);
            dibujaCadenaCaracteres(20, 60, cadena, &jc, 0, 100, 1, &pantalla);
        }
        // Muestra un mensaje en la pantalla LCD con el dato recibido, si se ha recibido alguno

        intercambiaBuffersLCD(&pantalla);
        // Intercambia los buffers para visualizar los mensajes escritos anteriormente
    }

  /* USER CODE END 5 */
}
```

En el microcontrolador se pueden utilizar también temporizadores que podemos crear en el sistema operativo FreeRTOS para ejecutar acciones repetitivas con un cierto período.

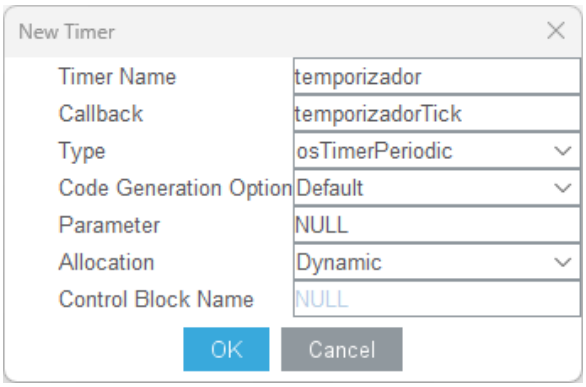
Seguidamente se describe una pequeña aplicación que muestra en pantalla un contador que se incrementa a cada segundo:



En el panel de configuración del microcontrolador yendo a **`Middleware - FreeRTOS - Timers and Semaphores`** y pulsando el botón **`Add`** podemos solicitar la creación de un temporizador



En **`Timer name`** se le asigna un identificador y en **`Callback`** se indica la función que se ejecutará a cada temporización.



Al regenerar el código de la aplicación, en **main.c** aparece la declaración de una estructura **temporizadorHandle** de tipo **osTimerId_t**

```
/* Definitions for temporizador */
osTimerId_t temporizadorHandle;
const osTimerAttr_t temporizador_attributes = {
  .name = "temporizador"
};
```

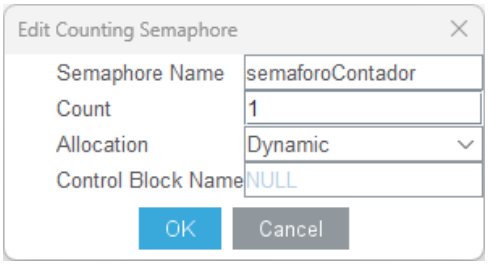
En la función **main** aparece una instrucción para la creación del temporizador

```
  /* Create the timer(s) */
  /* creation of temporizador */
  temporizadorHandle = osTimerNew(temporizadorTick, osTimerPeriodic, NULL, &temporizador_attributes);
```

Asimismo se genera automáticamente la función que se ejecutará a cada temporización

```
/* temporizadorTick function */
void temporizadorTick(void *argument)
{
  /* USER CODE BEGIN temporizadorTick */

  /* USER CODE END temporizadorTick */
}
```

Para que la tarea espere a que cada temporización modifique el valor del contador, se puede utilizar un semáforo que se puede crear en **Middleware - FreeRTOS - Timers and Semaphores**, por ejemplo creando el semáforo **semaforoContador** inicializado a **1**:



En **main.c** aparece la declaración de la variable **semaforoContadorHandle** de tipo **osSemaphoreId_t**:

```
/* Definitions for semaforoContador */
osSemaphoreId_t semaforoContadorHandle;
const osSemaphoreAttr_t semaforoContador_attributes = {
  .name = "semaforoContador"
};
```

y en la funcion **main()** hay una instrucción que crea el semáforo y le asigna su valor inicial:

```
/* Create the semaphores(s) */
/* creation of semaforoContador */
semaforoContadorHandle = osSemaphoreNew(1, 1, &semaforoContador_attributes);
```

En **main.c** hay que incluir los archivos

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "pantalla.h"
#include "Consolas9x18Amarillo.h"
#include <stdio.h>  // Para la declaración de sprintf
/* USER CODE END Includes */
```

En la tarea se ejecuta el siguiente código, utilizando la variable global **`contador`** que se incrementa a cada temporización en la función **`temporizadorTick`**:

```
/* USER CODE BEGIN 4 */

int contador = 0;   // Contador que se incrementa a cada temporización

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
  * @brief  Function implementing the defaultTask thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument)
{
  /* init code for USB_HOST */
  MX_USB_HOST_Init();
  /* USER CODE BEGIN 5 */

    char cadena[30];  // Cadena de caracteres para generar mensajes a visualizar en la pantalla

    PantallaLCD pantallaLCD;
    inicializaPantalla2Buffers(1, 0xFF000030, NULL, &pantallaLCD); // Inicializa la pantalla

    JuegoCaracteresAlpha juegoConsolas9x18;
    inicializaJuegoCaracteresAlpha(9, 18, 0, Consolas9x18Amarillo, &juegoConsolas9x18);
    // Inicializa un juego de caracteres

    osTimerStart(temporizadorHandle, 1000);  // Arranca el temporizador con un período de 1000 milisegundos

    while(1) {  // Repite indefinidamente ...

        osSemaphoreAcquire(semaforoContadorHandle, osWaitForever);
        // Ejecuta un wait sobre el semáforo para esperar a que se haya generado otro valor para el contador

        sprintf(cadena, "contador = %d", contador);
        dibujaCadenaCaracteresAlpha(10, 10, cadena, &juegoConsolas9x18, 0, 100, 1, &pantallaLCD);
        // Prepara el valor del contador en una cadena de caracteres y la visualiza en la posición (10, 10)

        intercambiaBuffersLCD(&pantallaLCD);
        // Intercambia los frame buffers para visualizar lo que se ha dibujado en el buffer oculto
    }

  /* USER CODE END 5 */
}

/* temporizadorTick function */
void temporizadorTick(void *argument)
{
  /* USER CODE BEGIN temporizadorTick */

        contador ++;  // Incrementa el contador a cada temporización

        osSemaphoreRelease(semaforoContadorHandle);
        // Ejecuta un signal sobre el semáforo para despertar a la tarea que visualiza el valor del
       // contador en pantalla

  /* USER CODE END temporizadorTick */
}
```