Escuela Universitaria
Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

**ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)**

ANEXOS

Dispositivo IoT para la medición de pH en agua.

IoT device for measuring pH in water.

424.23.24

Autor: Marco Ruiz López

Director: Dr. David Asiain Ansorena

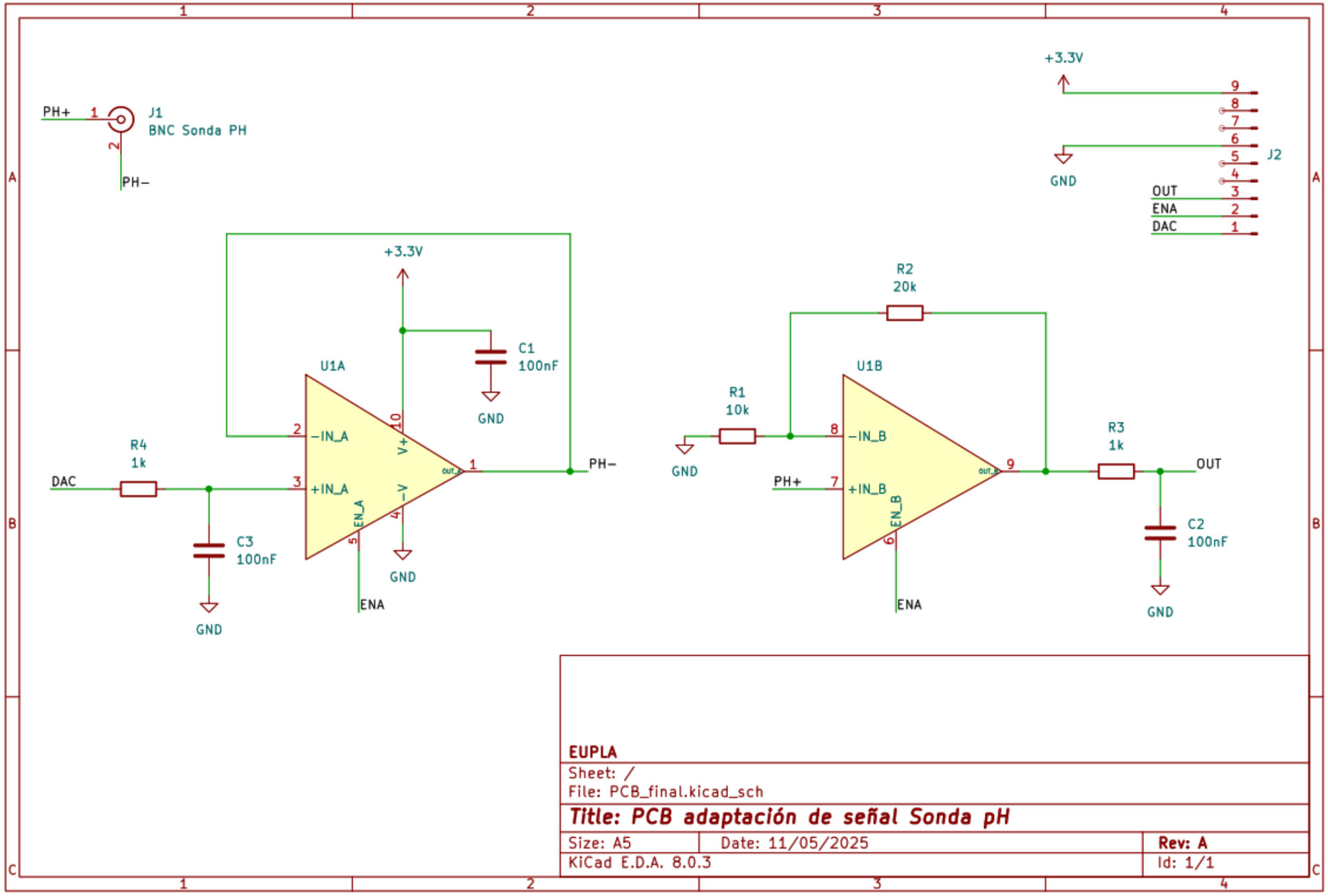
Fecha: 07 2025

Página intencionadamente en blanco.

INDICE DE CONTENIDO

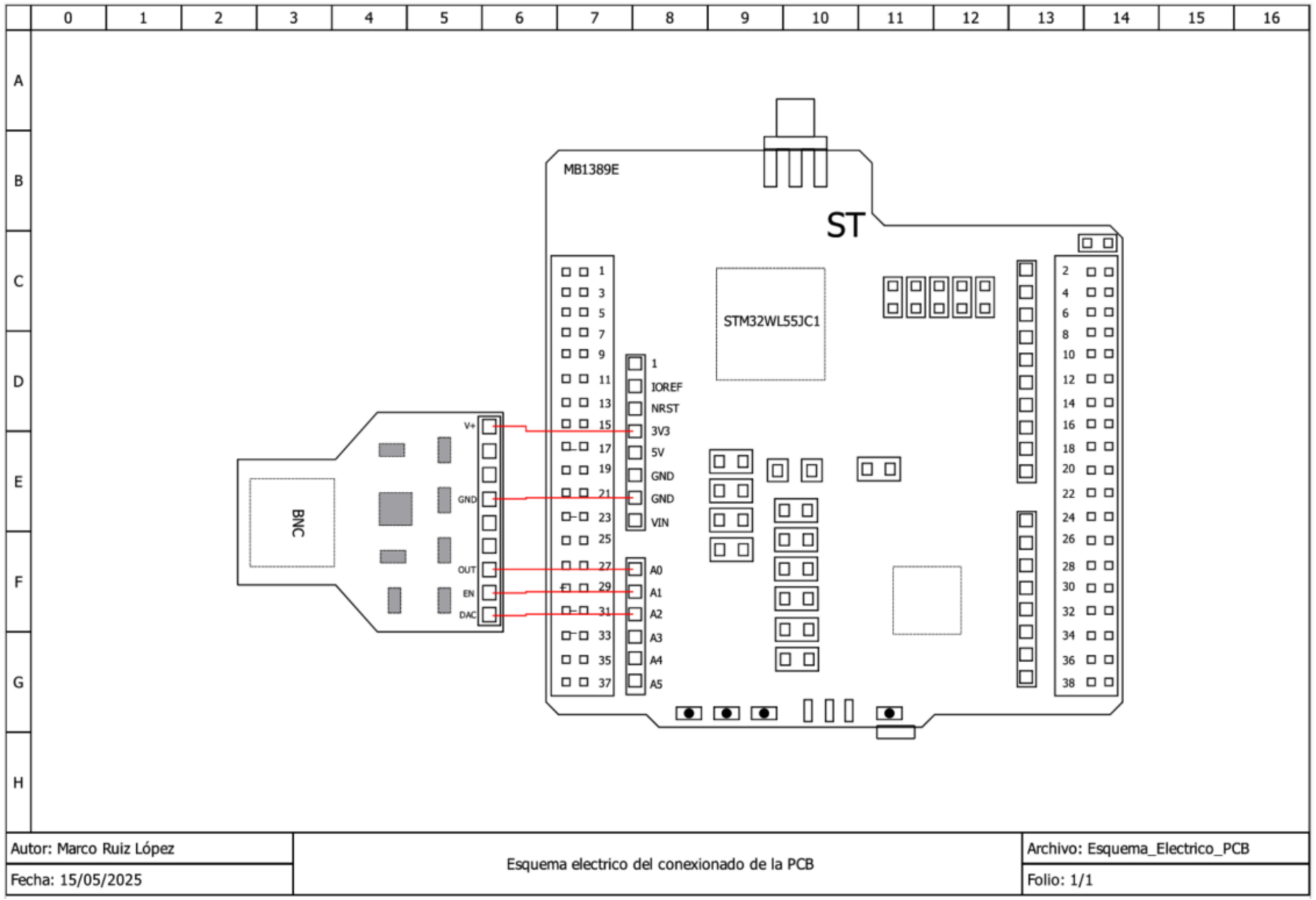
1. PLANOS DEL ESQUEMA ELECTRÓNICO.

Esquema electrónico de la placa (PCB) encargada de la adaptación de señal del sensor de pH es el siguiente:



2. ESQUEMA DEL CONEXIONADO DE LA PCB.

A continuación, se observa el plano correspondiente a la conexión eléctrica entre la PCB diseñada y la placa "NUCLEO-WL55JC1":



3. CÓDIGO FUENTE.

En este apartado de los anexos se verán las partes del código desarrollado de mayor importancia en el prototipo. No se podrá ver todo el código al completo debido a su gran extensión.

3.1. APLICACIÓN LORAWAN

A continuación, se mostrarán las funciones más importantes para la aplicación LoRaWAN del software.

```
void LoRaWAN_Init(void)
{
  /* USER CODE BEGIN LoRaWAN_Init_LV */
  uint32_t feature_version = 0UL;
  /* USER CODE END LoRaWAN_Init_LV */

  /* USER CODE BEGIN LoRaWAN_Init_1 */

  /* Get LoRaWAN APP version*/
  APP_LOG(TS_OFF, VLEVEL_M, "APPLICATION_VERSION: V%X.%X.%X\r\n",
          (uint8_t)(APP_VERSION_MAIN),
          (uint8_t)(APP_VERSION_SUB1),
          (uint8_t)(APP_VERSION_SUB2));

  /* Get MW LoRaWAN info */
  APP_LOG(TS_OFF, VLEVEL_M, "MW_LORAWAN_VERSION:  V%X.%X.%X\r\n",
          (uint8_t)(LORAWAN_VERSION_MAIN),
          (uint8_t)(LORAWAN_VERSION_SUB1),
          (uint8_t)(LORAWAN_VERSION_SUB2));

  /* Get MW SubGhz_Phy info */
  APP_LOG(TS_OFF, VLEVEL_M, "MW_RADIO_VERSION:    V%X.%X.%X\r\n",
          (uint8_t)(SUBGHZ_PHY_VERSION_MAIN),
          (uint8_t)(SUBGHZ_PHY_VERSION_SUB1),
          (uint8_t)(SUBGHZ_PHY_VERSION_SUB2));

  /* Get LoRaWAN Link Layer info */
  LmHandlerGetVersion(LORAMAC_HANDLER_L2_VERSION, &feature_version);
  APP_LOG(TS_OFF, VLEVEL_M, "L2_SPEC_VERSION:     V%X.%X.%X\r\n",
          (uint8_t)(feature_version >> 24),
          (uint8_t)(feature_version >> 16),
          (uint8_t)(feature_version >> 8));

  /* Get LoRaWAN Regional Parameters info */
  LmHandlerGetVersion(LORAMAC_HANDLER_REGION_VERSION, &feature_version);
  APP_LOG(TS_OFF, VLEVEL_M, "RP_SPEC_VERSION:     V%X-%X.%X.%X\r\n",
          (uint8_t)(feature_version >> 24),
          (uint8_t)(feature_version >> 16),
          (uint8_t)(feature_version >> 8),
          (uint8_t)(feature_version));
```

```
  UTIL_TIMER_Create(&TxLedTimer, LED_PERIOD_TIME, UTIL_TIMER_ONESHOT, OnTxTimerLedEvent,
NULL);
  UTIL_TIMER_Create(&RxLedTimer, LED_PERIOD_TIME, UTIL_TIMER_ONESHOT, OnRxTimerLedEvent,
NULL);
  UTIL_TIMER_Create(&JoinLedTimer, LED_PERIOD_TIME, UTIL_TIMER_PERIODIC,
OnJoinTimerLedEvent, NULL);

  if (FLASH_IF_Init(NULL) != FLASH_IF_OK)
  {
    Error_Handler();
  }

  /* USER CODE END LoRaWAN_Init_1 */

  UTIL_TIMER_Create(&StopJoinTimer, JOIN_TIME, UTIL_TIMER_ONESHOT, OnStopJoinTimerEvent,
NULL);

  UTIL_SEQ_RegTask((1 << CFG_SEQ_Task_LmHandlerProcess), UTIL_SEQ_RFU, LmHandlerProcess);

  UTIL_SEQ_RegTask((1 << CFG_SEQ_Task_LoRaSendOnTxTimerOrButtonEvent), UTIL_SEQ_RFU,
SendTxData);
  UTIL_SEQ_RegTask((1 << CFG_SEQ_Task_LoRaStoreContextEvent), UTIL_SEQ_RFU, StoreContext);
  UTIL_SEQ_RegTask((1 << CFG_SEQ_Task_LoRaStopJoinEvent), UTIL_SEQ_RFU, StopJoin);

  /* Init Info table used by LmHandler*/
  LoraInfo_Init();

  /* Init the Lora Stack*/
  LmHandlerInit(&LmHandlerCallbacks, APP_VERSION);

  LmHandlerConfigure(&LmHandlerParams);

  /* USER CODE BEGIN LoRaWAN_Init_2 */
  UTIL_TIMER_Start(&JoinLedTimer);

  /* USER CODE END LoRaWAN_Init_2 */

  LmHandlerJoin(ActivationType, ForceRejoin);

  if (EventType == TX_ON_TIMER)
  {
    /* send every time timer elapses */
    UTIL_TIMER_Create(&TxTimer, TxPeriodicity, UTIL_TIMER_ONESHOT, OnTxTimerEvent, NULL);
    UTIL_TIMER_Start(&TxTimer);
  }
  else
  {
    /* USER CODE BEGIN LoRaWAN_Init_3 */

    /* USER CODE END LoRaWAN_Init_3 */
  }

  /* USER CODE BEGIN LoRaWAN_Init_Last */

  /* USER CODE END LoRaWAN_Init_Last */
}
```

```
static void SendTxData(void)
{
  /* USER CODE BEGIN SendTxData_1 */
          uint16_t batteryLevel = SYS_GetBatteryLevel();
          uint16_t temperature = SYS_GetTemperatureLevel();
          float phLevel = GetPHLevel(750,1000);     // añadir los parametros internos
          //float phLevel = 2.45;
          uint16_t phLevelScaled = (uint16_t)(phLevel*100);


          APP_LOG(TS_ON, VLEVEL_M, "VDDA: %d\r\n", batteryLevel);
          APP_LOG(TS_ON, VLEVEL_M, "temperatura: %d\r\n", temperature);
          APP_LOG(TS_ON, VLEVEL_M, "pH: %d\r\n", phLevelScaled);

          LmHandlerErrorStatus_t status = LORAMAC_HANDLER_ERROR;
          UTIL_TIMER_Time_t nextTxIn = 0;

          AppData.Port = LORAWAN_USER_APP_PORT;

          AppData.Buffer[0] = (uint8_t)((batteryLevel >> 8 ) & 0xff);
          AppData.Buffer[1] = (uint8_t)(batteryLevel & 0xff);
          AppData.Buffer[2] = (uint8_t)(temperature >> 8) & 0xFF;
          AppData.Buffer[3] = (uint8_t)(temperature & 0xFF);
          AppData.Buffer[4] = (uint8_t)((phLevelScaled >> 8) & 0xFF);
          AppData.Buffer[5] = (uint8_t)(phLevelScaled & 0xFF);
          AppData.BufferSize = 6;

          status = LmHandlerSend(&AppData, LmHandlerParams.IsTxConfirmed, true);

          if (LORAMAC_HANDLER_SUCCESS == status) {
                    APP_LOG(TS_ON, VLEVEL_L, "SEND REQUEST\r\n");
          }else if (LORAMAC_HANDLER_DUTYCYCLE_RESTRICTED == status) {
                    nextTxIn = LmHandlerGetDutyCycleWaitTime();
                    if (nextTxIn > 0) {
                              APP_LOG(TS_ON, VLEVEL_L, "Next Tx in  : ~%d second(s)\r\n",
(nextTxIn / 1000));
                    }
          }

          if (EventType == TX_ON_TIMER) {
                    UTIL_TIMER_Stop(&TxTimer);
                    UTIL_TIMER_SetPeriod(&TxTimer, MAX(nextTxIn, TxPeriodicity));
                    UTIL_TIMER_Start(&TxTimer);
          }

  /* USER CODE END SendTxData_1 */
}
```

3.2. CÓDIGO PARA LA LECTURA DE PH DE LA PCB.

```
/* Exported functions ------------------------------------------------------*/
/* USER CODE BEGIN EF */
static uint32_t ADC_ReadChannels(uint32_t channel)
{
  /* USER CODE BEGIN ADC_ReadChannels_1 */

  /* USER CODE END ADC_ReadChannels_1 */
  uint32_t ADCxConvertedValues = 0;
  ADC_ChannelConfTypeDef sConfig = {0};

  MX_ADC_Init();

  /* Start Calibration */
  if (HAL_ADCEx_Calibration_Start(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

  /* Configure Regular Channel */
  sConfig.Channel = channel;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLINGTIME_COMMON_1;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  if (HAL_ADC_Start(&hadc) != HAL_OK)
  {
    /* Start Error */
    Error_Handler();
  }
  /** Wait for end of conversion */
  HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);

  /** Wait for end of conversion */
  HAL_ADC_Stop(&hadc);   /* it calls also ADC_Disable() */

  ADCxConvertedValues = HAL_ADC_GetValue(&hadc);

  HAL_ADC_DeInit(&hadc);

  return ADCxConvertedValues;
  /* USER CODE BEGIN ADC_ReadChannels_2 */

  /* USER CODE END ADC_ReadChannels_2 */
```

```
}

static void DAC_mVoltaje(float mvolt){
        uint16_t dato_dac = (uint16_t)((mvolt*4095)/3300.0);
        HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
        HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dato_dac);
}

float GetPHLevel(int biasVoltage, uint32_t time){
        float ph;                       // lectura final con decimales de pH
        uint16_t adcRaw;                        // lectura cruda del adc
        uint32_t sumaRaw = 0;           // suma de las lecturas del ADC, usa
uint32_t para evitar desbordamientos
        uint16_t totalMuestras = 0;     // contador de muestras del ADC
        uint32_t avgRaw = 0;            // valor promedio del ADC
        uint32_t previousMillis = 0;
        uint32_t currentMillis;
        uint32_t duration = time;       // Duración en milisegundos

        // Activamos el ENABLE del amplificador
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);

        // Establecemos el voltaje de bias para la calibración
        DAC_mVoltaje(biasVoltage);

        // Iniciamos la recogida de datos
        previousMillis = HAL_GetTick();
        while(1){
                // Leemos el valor del ADC
                adcRaw = ADC_ReadChannels(ADC_CHANNEL_5);

                // Sumamos las lecturas del ADC
                sumaRaw += adcRaw;
                totalMuestras++;

                // Actualizamos el tiempo actual
                currentMillis = HAL_GetTick();

                // Si ha pasado el tiempo especificado calculamos la media
                if ((currentMillis - previousMillis) >= duration) {
                        avgRaw = sumaRaw / totalMuestras;    // Calcular el
promedio de las lecturas
                        break;  // Salimos del bucle
                }
        }

        // Desactivamos el ENABLE del amplificador
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);

        // Paramos el DAC
        HAL_DAC_Stop(&hdac, DAC_CHANNEL_1);
```

```
        // Calculamos el pH usando el valor promedio del ADC
        ph = 20.45273 - 0.00469 * (float)avgRaw;

        return ph;
}
```

Relación de documentos

(_) Memoria 125 páginas
(X) Anexos 12 páginas

La Almunia, a 01 de Julio de 2025

Firmado: Marco Ruiz López