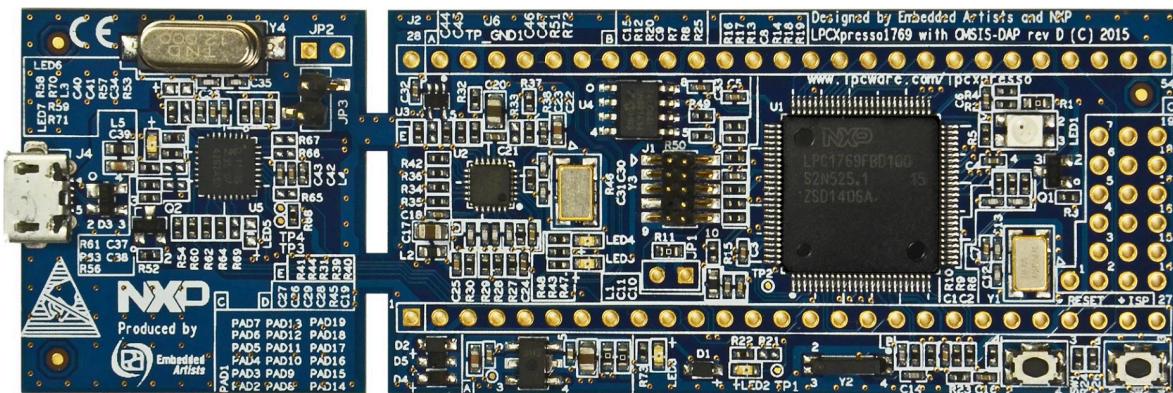


Universidad Nacional de Córdoba
Facultad de Ciencias Exactas, Físicas y Naturales

Electrónica Digital III

Trabajo Práctico Final

Sensado de Nivel de Agua



Integrantes:

Santiago Agustín Colque Ventura

Joaquin Andres Pary

Profesores:

Julio Sanchez

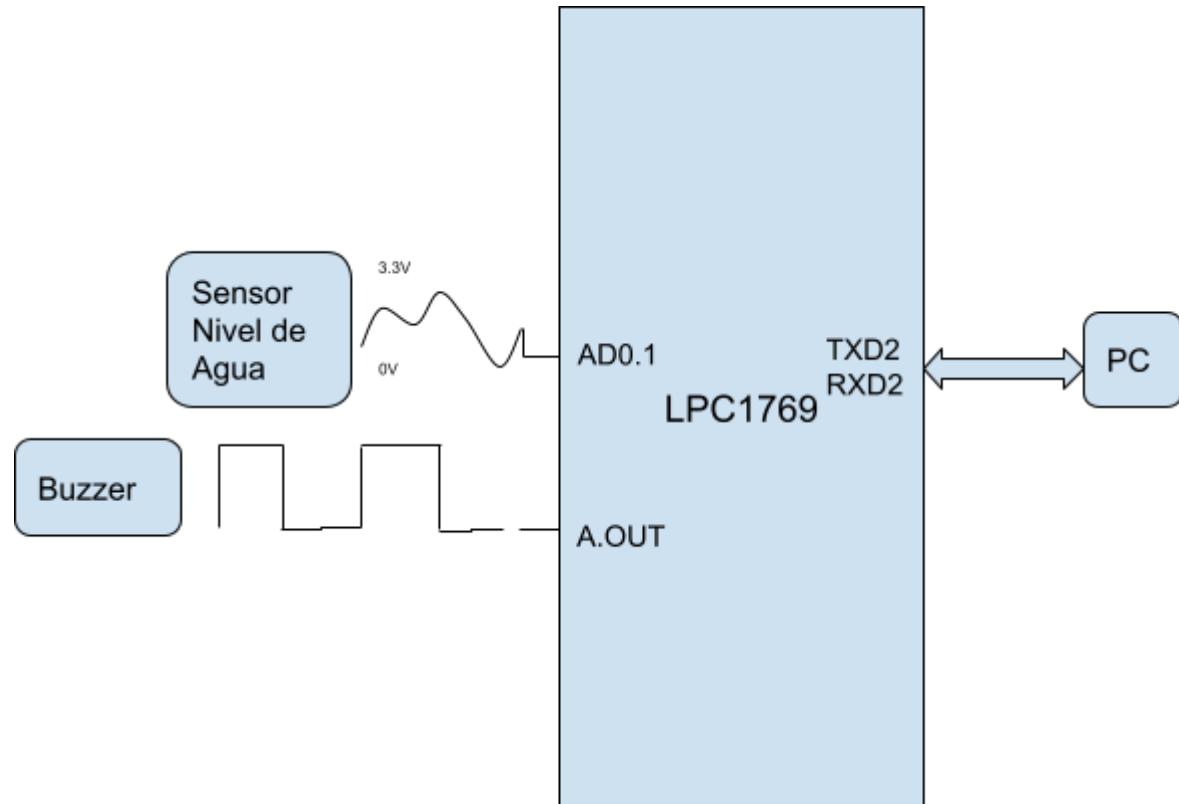
Fernando Gallardo

Introducción

Este proyecto tiene como objetivo principal desarrollar un sistema práctico de control de nivel de líquido aplicable en diversos escenarios. La base de este sistema es un sensor de nivel de agua que genera una señal analógica. Para medir este nivel, utilizamos el conversor analógico a digital (ADC) configurado en conjunto con el match del temporizador TIMER0.

Cuando la medición cae por debajo del nivel establecido, entra en acción el conversor digital a analógico (DAC) con Direct Memory Access (DMA). Esto desencadena la generación de una señal de modulación por ancho de pulso (PWM), la cual se conecta a un buzzer para proporcionar una alerta audible.

Además, hemos incorporado una funcionalidad de conectividad mediante UART2. Esto permite la transmisión de datos y da un aviso cuando el nivel de agua es más bajo que el umbral establecido. Esta característica no solo facilita la supervisión remota del sistema, sino que también brinda la posibilidad de ajustar el nivel mínimo de agua (el umbral) directamente desde la PC.



Dispositivos utilizados

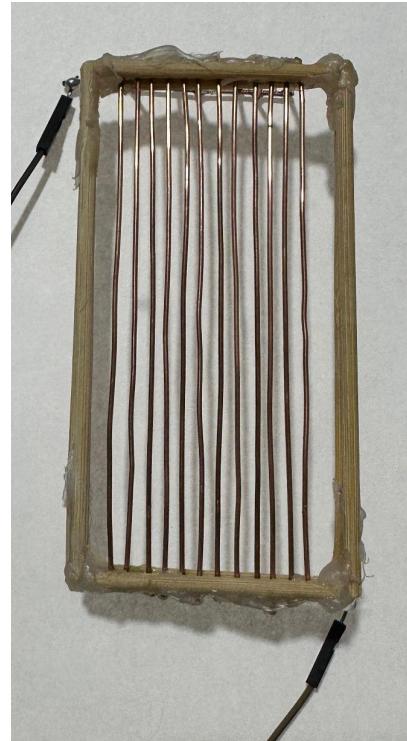
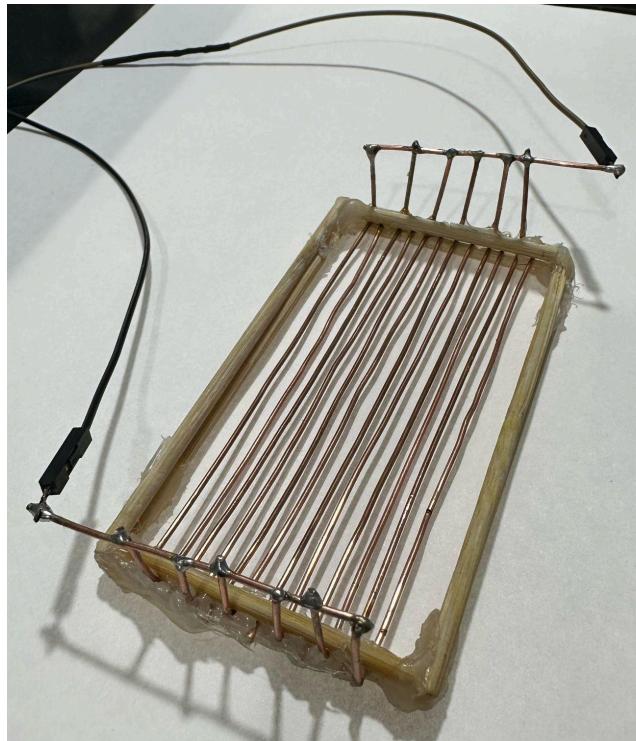
Adaptador USB a TTL PL2303hx



Sensor de nivel de agua HW-038



Sensor de nivel de agua artesanal



Implementación

Los periféricos de la placa utilizados en el proyecto son los siguientes:

- ADC (Canal 1)
- TIMER0
- DAC
- DMA (Canal 7)
- UART2

ADC

Se utilizó el ADC con un voltaje de referencia de 0V a 3.3V (por defecto) y configurado para iniciar la conversión de la señal recibida por AD0.1 cuando ocurra un flanco de subida en el pin MAT0.1, además se activaron las interrupciones de manera que se generen al terminar cada conversión. A continuación un extracto del código utilizado para el setup del ADC.

Frecuencia de muestreo de 100 KHz:

```
ADC_Init(LPC_ADC, 100000);
```

Modo burst deshabilitado:

```
ADC_BurstCmd(LPC_ADC, DISABLE);
```

Habilitación del canal 1 del ADC:

```
ADC_ChannelCmd(LPC_ADC, 1, ENABLE);
```

La conversión comienza cuando ocurre un evento con MAT0.1:

```
ADC_StartCmd(LPC_ADC, 4);
```

Selección de flanco de subida como evento de MAT0.1:

```
ADC_EdgeStartConfig(LPC_ADC, 0);
```

Habilitación de la interrupción del ADC cuando termine de convertir:

```
ADC_IntConfig(LPC_ADC, 1, ENABLE);
```

```
NVIC_EnableIRQ(ADC IRQn);
```

TIMER0

Configurado para producir un pulso por MAT0.1 cada 30 segundos según los siguientes cálculos:

$$Cclk = 100[MHz]$$

$$Pclk = 100[MHz]$$

$$PR = 999$$

$$f_{res} = \frac{Pclk}{PR+1} = 100[KHz]$$

$$MR1 = \frac{30[Seg]}{1/f_{res}} = 3000000$$

Se realizó la configuración correspondiente para que TIMER COUNTER se reinicie cada vez que se produce el match con MR1, así como para que se produzca un pulso en el pin MR1 por cada match.

Prescaler y valor de match:

LPC_TIM0->PR = 999;

LPC_TIM0->MR1 = 30000;

Reseteo del TC cuando ocurre match entre MR1 y TC:

LPC_TIM0->MCR |= (1<<4);

Configuración de pulso en MAT0.1:

LPC_TIM0->EMR |= (3<<6);

Inicio de TC:

LPC_TIM0->TCR |= 0x03;

LPC_TIM0->TCR &= 0x02;

DAC

Teniendo en cuenta una señal PWM de 1000 Hz (periodo de 1ms) definida con 100 muestras, se debe actualizar el DAC cada 10us, y entonces el cálculo para el valor que hay que utilizar en la función que configura el TimeOut del DMA para el DAC es:

$$P_{clk} = 100[Mhz]$$

$$T_{clk} = 1 \times 10^{-8}[seg]$$

$$f_{PWM} = 1000[Hz]$$

$$T_{PWM} = 1/f_{PWM} = 1 \times 10^{-3}[seg]$$

$$Samples = 100$$

$$T_{Sample} = \frac{T_{PWM}}{100} = 1[\mu s]$$

$$x = 10 \times 10^{-6} / 1 \times 10^{-8} = 1000$$

Entonces la configuración consiste en:

DAC_CONVERTER_CFG_Type DAC_ConverterConfigStruct;

Activa el contador del DAC:

DAC_ConverterConfigStruct.CNT_ENA = SET;

Activa la transferencia de datos del DMA:

DAC_ConverterConfigStruct.DMA_ENA = SET;

Enciende el DAC:

DAC_Init(LPC_DAC);

Asigna el valor para pedir una muestra cada 10uS:

DAC_SetDMATimeOut(LPC_DAC,1000);

DAC_ConfigDACConverterControl(LPC_DAC, &DAC_ConverterConfigStruct);

DMA

El DMA se configuró de la siguiente manera:

GPDMA_LLI_Type LLI;

Dirección de memoria de la primera muestra de la señal PWM y del DAC:

LLI.SrcAddr = (uint32_t) &(dac_pwm);

LLI.DstAddr = (uint32_t) & (LPC_DAC->DSCR);

Cuanto se termine la transferencia de todos los datos vuelve a comenzar:

LLI.NextLLI = (uint32_t) &LLI;

Tamaño de transferencia es de 10, ancho de la fuente y destino son 32 bits y la dirección de memoria de la fuente aumenta luego de cada transferencia:

LLI.Control = SAMPLES | (2<<18) | (2<<21) | (1<<26);

Enciende el DMA:

GPDMA_Init();

GPDMA_Channel_CFG_Type GPDMAFcg1;

Canal 7 del DMA:

GPDMAFcg1.ChannelNum = 7;

GPDMAFcg1.SrcMemAddr = (uint32_t) &(dac_pwm);

GPDMAFcg1.DstMemAddr = 0;

GPDMAFcg1.TransferSize = SAMPLES;

GPDMAFcg1.TransferWidth = 0;

Tipo de transferencia de memoria a periférico:

GPDMAFcg1.TransferType = GPDMA_TRANSFERTYPE_M2P;

GPDMAFcg1.SrcConn = 0;

GPDMAFcg1.DstConn = GPDMA_CONN_DAC;

Dirección de memoria de la LINKEDLIST:

GPDMAFcg1.DMALLI = (uint32_t)&LLI;

GPDMA_Setup(&GPDMAFcg1);

GPDMA_ChannelCmd(7, ENABLE);

UART2

Estructuras de UART2:

```
UART_CFG_Type    UARTConfigStruct;  
UART_FIFO_CFG_Type UARTFIFOConfigStruct;
```

Configuración UART2 por defecto, 9600 baudios:

```
UART_ConfigStructInit(&UARTConfigStruct);
```

Inicializa el periférico:

```
UART_Init(LPC_UART2, &UARTConfigStruct);
```

Inicializa la FIFO por defecto:

```
UART_FIFOConfigStructInit(&UARTFIFOConfigStruct);  
UART_FIFOConfig(LPC_UART2, &UARTFIFOConfigStruct);
```

Habilitamos las transferencias:

```
UART_TxCmd(LPC_UART2, ENABLE);
```

Configuramos las interrupciones para la recepción, tanto para cuando los datos están disponibles como para cuando ocurre algún error:

```
UART_IntConfig(LPC_UART2, UART_INTCFG_RBR, ENABLE);  
UART_IntConfig(LPC_UART2, UART_INTCFG_RLS, ENABLE);  
NVIC_SetPriority(UART2 IRQn, 1);  
NVIC_EnableIRQ(UART2 IRQn);
```

INTERRUPCIÓN DEL ADC

Dentro del handler del ADC comparamos la lectura del nivel de agua con el umbral previamente establecido.

Para esto acondicionamos la muestra, que resulta en un valor representado por 10 bits.

El ADC nos da una estimación de cuánta parte del sensor está por encima del nivel de agua, por lo que operando para calcular el nivel del agua:

```
water_level = 1024-water_level;
```

Declaramos una bandera para saber si la primera vez que entramos al handler:

```
static int i = 0;
```

Seguido a esto, comparamos el nivel del agua con el umbral. Si el nivel es menor al esperado, se envía una alerta por UART2 y, si es la primera vez que se ingresa al handler, se da comienzo a la alerta mediante el buzzer configurando el DAC y DMA. En los llamados subsiguientes sólo se habilita o deshabilita el canal de DMA.

```
if(water_level < umbral){
    uint8_t string[] = {"Nivel de agua bajo\n\r"};
    UART_Send(LPC_UART2, string, sizeof(string), BLOCKING);
    if(i == 0){
        dac_config();
        dma_config();
    }
    GPDMA_ChannelCmd(0, ENABLE);
}
else {
    GPDMA_ChannelCmd(7,DISABLE);
}
}
return;
```

INTERRUPCIÓN DE UART2

En el handler de la interrupción por UART, primero verificamos si se produjo algún error:

```
uint32_t intsrc, tmp, tmp1;
intsrc = UART_GetIntId(LPC_UART2);
tmp = intsrc & UART_IIR_INTID_MASK;
if (tmp == UART_IIR_INTID_RLS){
    tmp1 = UART_GetLineStatus(LPC_UART2);
    tmp1 &= (UART_LSR_OE | UART_LSR_PE | UART_LSR_FE \
              | UART_LSR_BI | UART_LSR_RXFE);
    if (tmp1) {
        while(1){};
    }
}
```

Luego, procesamos los datos recibidos desde la pc, y respondemos con un mensaje de confirmación del nuevo umbral cuando los valores recibidos son válidos.

Finalmente asignamos a la variable umbral un valor diferente dependiendo del dato recibido.

```
if ((tmp == UART_IIR_INTID_RDA) || (tmp == UART_IIR_INTID_CTL)){
    UART_Receive(LPC_UART2, info, sizeof(info), NONE_BLOCKING);
    if(info[0] > 47 && info[0] < 53){
        UART_Send(LPC_UART2, "El umbral de deteccion es: ", sizeof("El
                           umbral de deteccion es: "), BLOCKING);
        UART_Send(LPC_UART2, info, sizeof(info), BLOCKING);
        UART_Send(LPC_UART2, "/4\n\r", sizeof("/4\n\r"), BLOCKING);

        if(info[0]-48 == 0){
            umbral = 350;
        }
        else if(info[0]-48 == 1){
            umbral = 700;
        }
        else if(info[0]-48 == 2){
            umbral = 775;
        }
        else if(info[0]-48 == 3){
            umbral = 805;
        }
        else if(info[0]-48 == 4){
            umbral = 830;
        }
    }
}
return;
```

Problemas Encuentrados

La dificultad más importante fue que el sensor para el nivel de agua que se pretendía utilizar no respondía a niveles intermedios de agua. Es decir, que al estar completamente seco la lectura recibida es de 0V y al estar húmedo, sin importar la cantidad de agua la lectura era siempre de 3.3V.

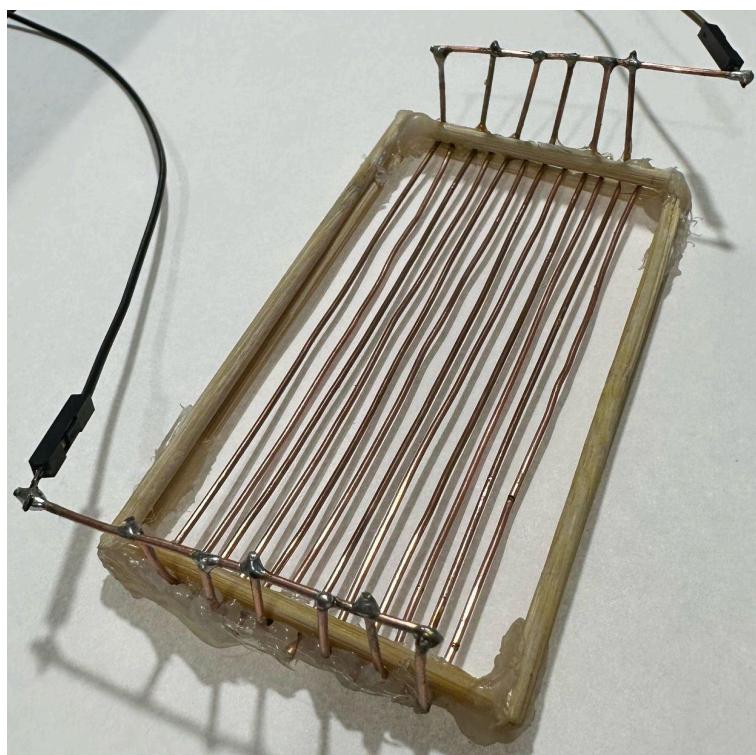
Este inconveniente provocaba que el proyecto no tuviera puntos grises, solo existían dos estados, blanco o negro.

Para solventar esta falencia decidimos fabricar de manera casera o artesanal nuestro propio sensor sensible a cambios en el nivel.

El sensor en cuestión deriva de la misma premisa que el comprado inicialmente. Es decir pistas de cobre, o en este caso alambres, conectados de forma intercalada a cada uno de los bornes de forma que posee una resistencia muy grande al no existir conexión física entre ambos bornes. Al estar sumergido la resistencia disminuye ya que el agua cierra el circuito y permite que, dependiendo de cuan sumergido se encuentre el sensor, varíe la resistencia lo que indirectamente hace que también varíe el voltaje medido por el ADC.

El procedimiento de fabricación completo se encuentra en la sección de anexos, con algunas diferencias pero siguiendo el mismo principio.

Nuestro sensor es capaz de detectar diferentes niveles de agua, lo que nos permitió establecer diferentes umbrales que el usuario puede elegir para su medidor, otorgando una mayor versatilidad y utilidad.



Conclusiones

Podemos enumerar las siguientes conclusiones:

- Es importante conocer muy a detalle los sensores y componentes a utilizar, tanto sus características como sus limitaciones.
- Realizar un sensor casero nos permitió ampliar el enfoque de nuestro proyecto. Inicialmente la comunicación UART era unidireccional, y sólo comunicaba el estado binario del sensor comercial. Fabricar nuestro propio sensor posibilitó la definición de diferentes umbrales dando mayor libertad al usuario.
- Aunque desde un punto de vista general podemos decir que manufacturar un sensor desde cero resultó provechoso, es cierto que también significó una etapa adicional en la línea de tiempo del proyecto, por lo que destacamos la importancia de la planificación y las investigaciones previas, que pueden prevenir inconvenientes como el que tuvimos con el sensor.

Anexos

[LPC1769/68/67/66/65/64/63 Product data sheet \(nxp.com\)](https://www.nxp.com/docs/en/data-sheet/LPC1769-68-67-66-65-64-63.pdf)

<https://www.instructables.com/Building-an-Arduino-Water-Level-Detection-Sensor/>

<https://www.biomaker.org/block-catalogue/2021/12/17/water-level-sensor-tzt-water-level-sensor>