Trabajo Práctico Final

Simulador Instrumento Musical

Simón Saillen; Rodrigo S. Vargas.

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

Electronica Digital III.

Julio Sanchez; Fernando Gallardo.

21/11/2024

INTRODUCCIÓN

En este trabajo práctico, se desarrolló un sistema que simula el funcionamiento de un instrumento musical utilizando la placa de desarrollo LPC1769. La implementación consiste en un teclado musical que permite generar distintas notas mediante el uso de un buzzer pasivo. Este dispositivo convierte señales eléctricas en sonidos, reproduciendo frecuencias específicas asociadas a las notas musicales.
El proyecto tiene como objetivo principal aplicar conceptos fundamentales de microcontroladores, manejo de periféricos y generación de señales PWM (Pulse Width Modulation) para crear una experiencia interactiva que combina hardware y software.
A través de este desarrollo, se integran conocimientos sobre programación en lenguaje C, configuración de registros específicos del LPC1769, manejo de librerías, drivers y diseño de sistemas embebidos.

DESARROLLO

Para la implementación del instrumento musical se utilizó el lenguaje de programación C, manejo de registros, librerías CMSIS (Cortex Microcontroller Software Interface Standard) y drivers (funciones pre-definidas para el manejo de periféricos y otros).
Los módulos utilizados son EINT3 (por GPIO), Timer0, Timer1, ADC, DMA y DAC, su funcionamiento, código y diagrama de flujo los explicaremos abajo:

Timer0 y DAC:

En un principio íbamos a utilizar DMA - DAC, incluso lo llegamos a implementar y hacer funcionar, pero nos encontramos con un problema al utilizar interrupciones, al parecer los módulos DMA-DAC al tener que mandar tan seguido datos mediante el bus, si son interrumpidos se rompe esa conexión, intentamos varias veces de varias formas hacerlo funcionar pero siempre se terminaba rompiendo la conexión.
Fue entonces cuando tuvimos que reformular la lógica e implementar el DAC de otra forma, ya que es el módulo más importante para nuestro trabajo, pues genera la señal del buzzer pasivo.

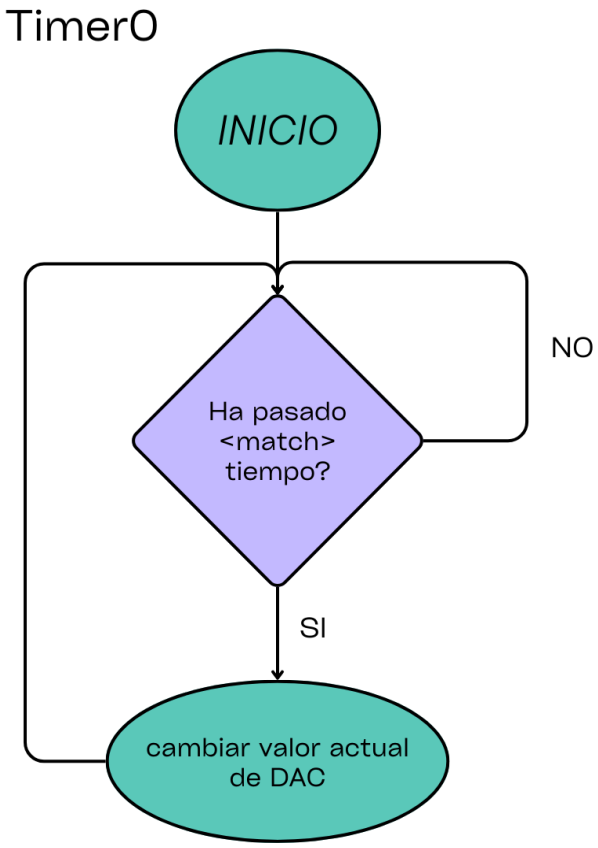
Logramos implementar el módulo DAC con la ayuda del Timer0, este último genera una interrupción cada tiempo de match y cambia el valor actual de DAC por el valor siguiente en su arreglo, por necesidad de una señal cuadrada para el buzzer, el arreglo de waveform del DAC es de 2 valores: 1023 y 0 (correspondientes a 3.3V y 0V). El valor de match inicia en la nota DO (262 Hz), y varía conforme a la tecla presionada en el teclado matricial (lo explicaremos en EINT3).

Snippets de Código (solo mostramos los Handlers):

```
void TIMER0_IRQHandler(void) {
    i = (i+1) % 2;
    DAC_UpdateValue(LPC_DAC, dac_waveform[i] / divisor);
    LPC_TIM0->IR |= (1<<0);
}

uint8_t divisor = 1; // Este valor cambia en el ADC_IRQHandler
```

(Debido a que el DAC no genera una interrupción no tenemos handler que mostrar, de todos modos el DAC solo realiza lo que el Timer0 le dice).

Diagramas de flujo:



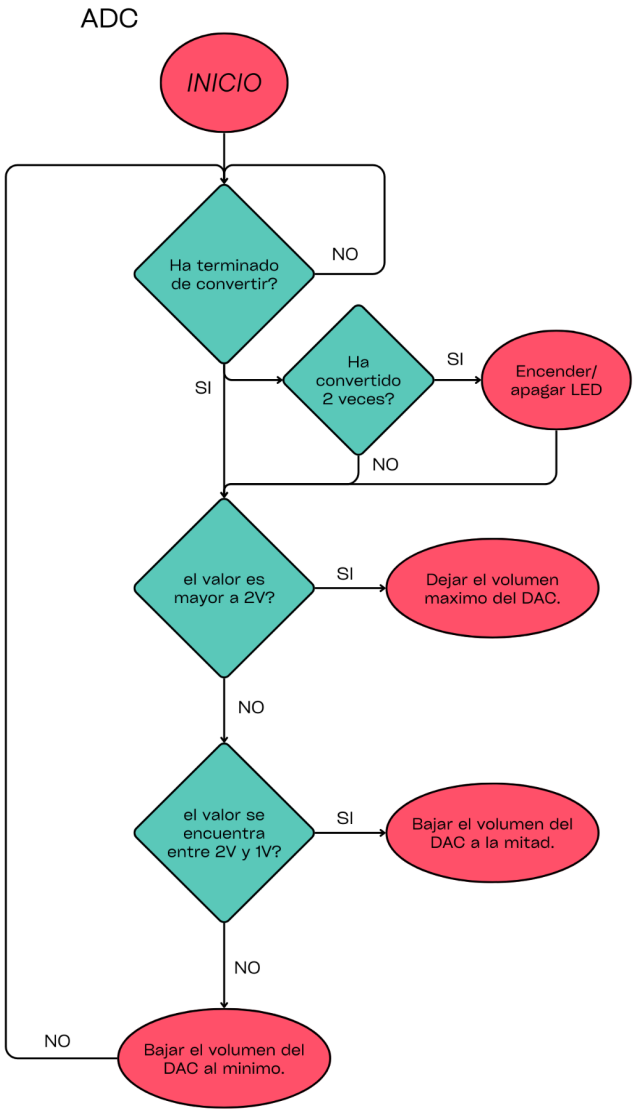
Timer1 y ADC:

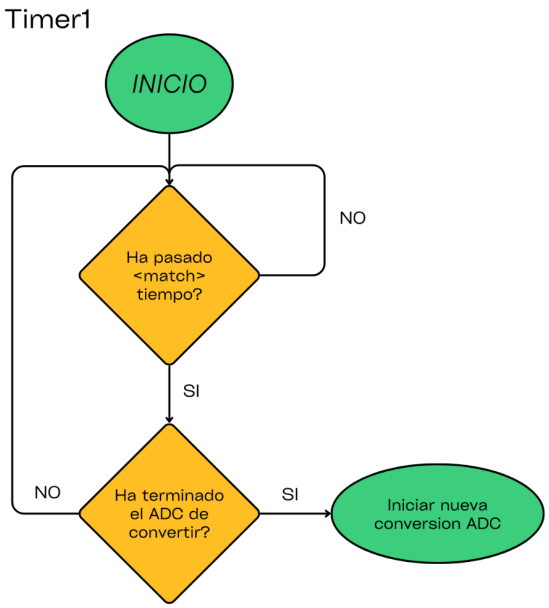
Al ADC lo implementamos mediante el uso de Timer1, este último cada 0.5 segundos aproximadamente genera una interrupción, donde dentro de esta inicia la conversión, luego al terminar el ADC interrumpe, prende/apaga un led conforme a la cantidad de conversiones y en base al valor leído del potenciómetro decide que realizar con el volumen del buzzer (la amplitud del DAC).

Snippets de Código:

```
void TIMER1_IRQHandler(void) {
    if (!ADC_ChannelGetStatus(LPC_ADC, 0, 1)) {
        ADC_StartCmd(LPC_ADC, ADC_START_NOW);
    }
    LPC_TIM1->IR |= (1<<0);
}

void ADC_IRQHandler(void) {
    uint16_t value = ADC_ChannelGetData(LPC_ADC, 0);
    static uint8_t flag = 0;

    if(flag % 2 == 0) {
        LPC_GPIO2->FIOSET |= (1<<13);
    } else {
        LPC_GPIO2->FIOCLR |= (1<<13);
    }

    flag = (flag + 1) % 5;

    if(value < 4096 && value >= 2731) {
        divisor = 1;
    } else if(value < 2731 && value >= 1365) {
        divisor = 2;
    } else if(value < 1365 && value >= 0) {
        divisor = 3;
    }
    LPC_ADC->ADGDR &= ~(1 << 31); // Limpia la flag DONE
}
```

Diagramas de Flujo:





EINT3 (GPIOINT):

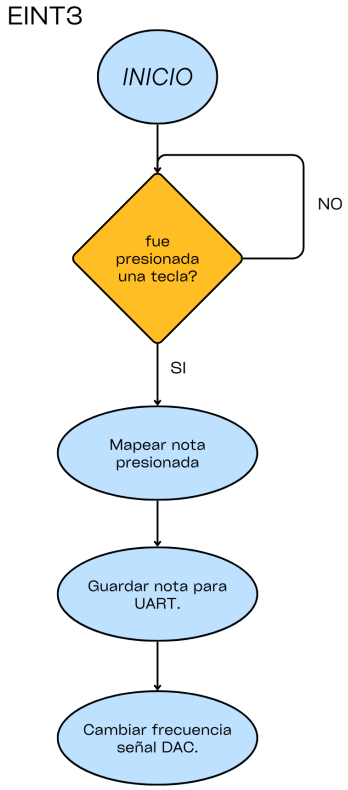
Para el cambio de notas musicales utilizamos un teclado matricial y mediante interrupciones de GPIO podemos saber que tecla fue presionada y que valor musical le asignamos a cada tecla y actualizar el valor en el DAC para que refleje la nueva nota (frecuencia) y la guardamos en un buffer para el módulo UART.

Snippets de Codigo:

```
#define SALIDA_TECLADO (0XF << 4)
#define ENTRADA_TECLADO 0XF

void EINT3_IRQHandler(void) {
    NVIC_DisableIRQ(TIMER0_IRQn);
    NVIC_DisableIRQ(EINT3_IRQn);

    uint8_t indice = obtener_teclaMatricial();
    delay(200); // Antirrebote

    memset(uartBuffer, '\0', sizeof(uartBuffer));
    mapearNota(indice);
    DMA_Config();

    TIM_UpdateMatchValue(LPC_TIM0, 0, matches[indice]);
    TIM_ResetCounter(LPC_TIM0);

    LPC_GPIOINT->IO2IntClr |= 0xf;
    NVIC_EnableIRQ(TIMER0_IRQn);
    NVIC_EnableIRQ(EINT3_IRQn);
}

uint8_t obtener_teclaMatricial(void) {
    uint8_t fila = 0, columna = 0;
    // Voy pin a pin verificando cual es la fila apretada
    while(fila != 4) {
        if(GPIO_ReadValue(2) & (1 << fila))
            break;

        fila++;
    }

    if(fila >= 4)
        return 4; //Si no encontro devuelve 4
```

```
    // Para obtener la columna barro un 0 por las columnas
    while(columna != 4) {
        LPC_GPIO2->FIOPIN0 = ~(1 << (4+columna));

        if(!((FIO_ByteReadValue(2,0)) & ENTRADA_TECLADO))
            break;

        columna++;
    }

    if(columna >= 4)
        return 4;

    FIO_ByteSetValue(2, 0, SALIDA_TECLADO);

    return (fila*4 + columna);
}
```

Diagrama de Flujo:



DMA - UART:

Utilizando DMA, transmite a la PC mediante el módulo UART, se envía un string correspondiente a la nota presionada, terminando cada mensaje con un carácter de nueva línea (\n).
Por otra parte, en la PC, un programa en Python abre el puerto serial correspondiente e intenta leer las líneas recibidas. Cada línea leída se compara con una biblioteca de valores predefinidos. Si el string recibido coincide con alguno de los valores de la biblioteca, se actualiza una variable global que lleva el conteo de cuántas veces se ha presionado esa tecla específica. Esto permite registrar y procesar las interacciones en tiempo real.

TABLAS:

Para la representación de notas musicales debimos realizar una señal cuadrada, cada valor de frecuencia relacionado a nota musical fue calculado usando la siguiente fórmula matemática:

$$f(n) = 440 \cdot 2^{(n-49)/12}$$

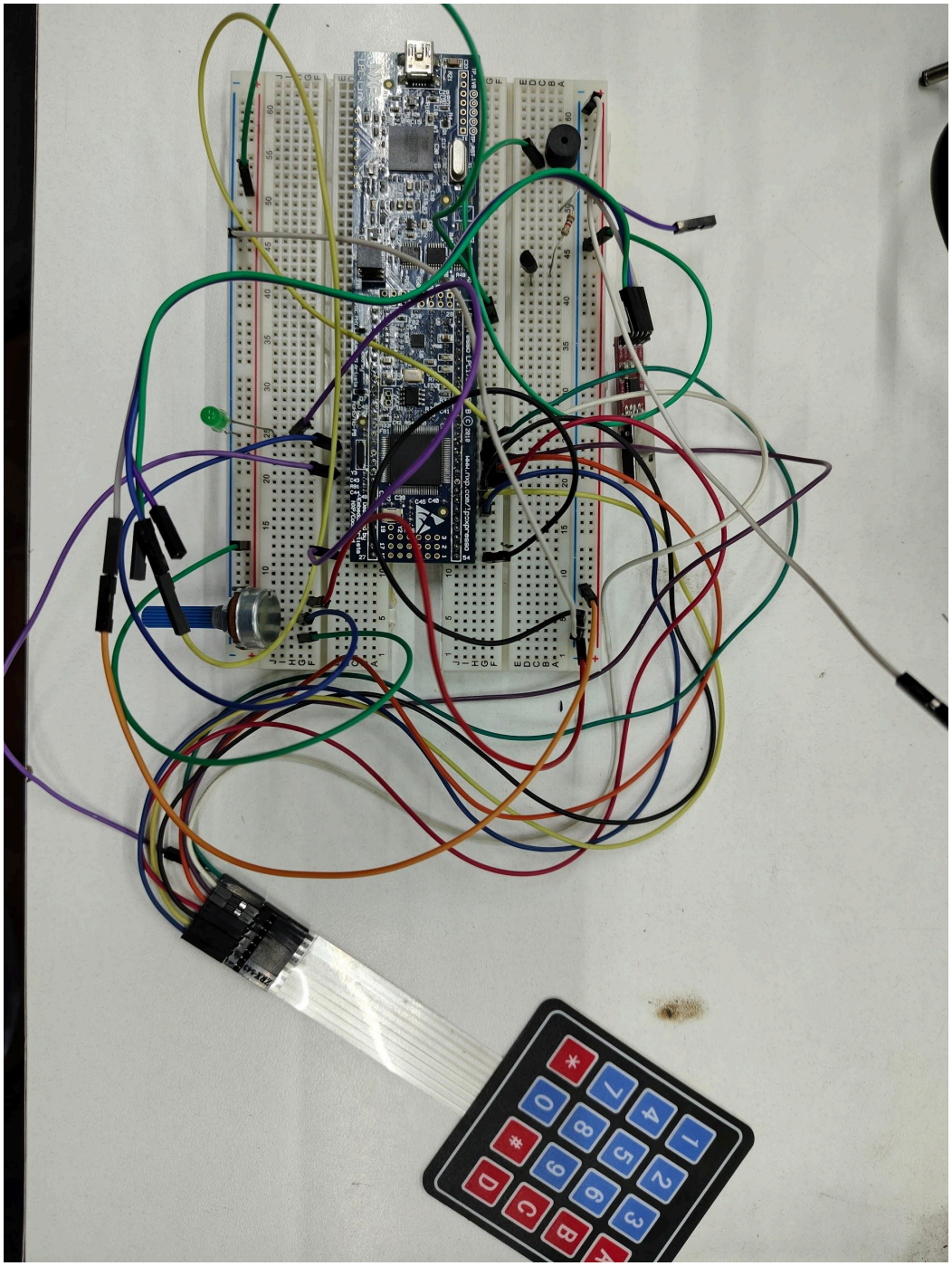
Donde:

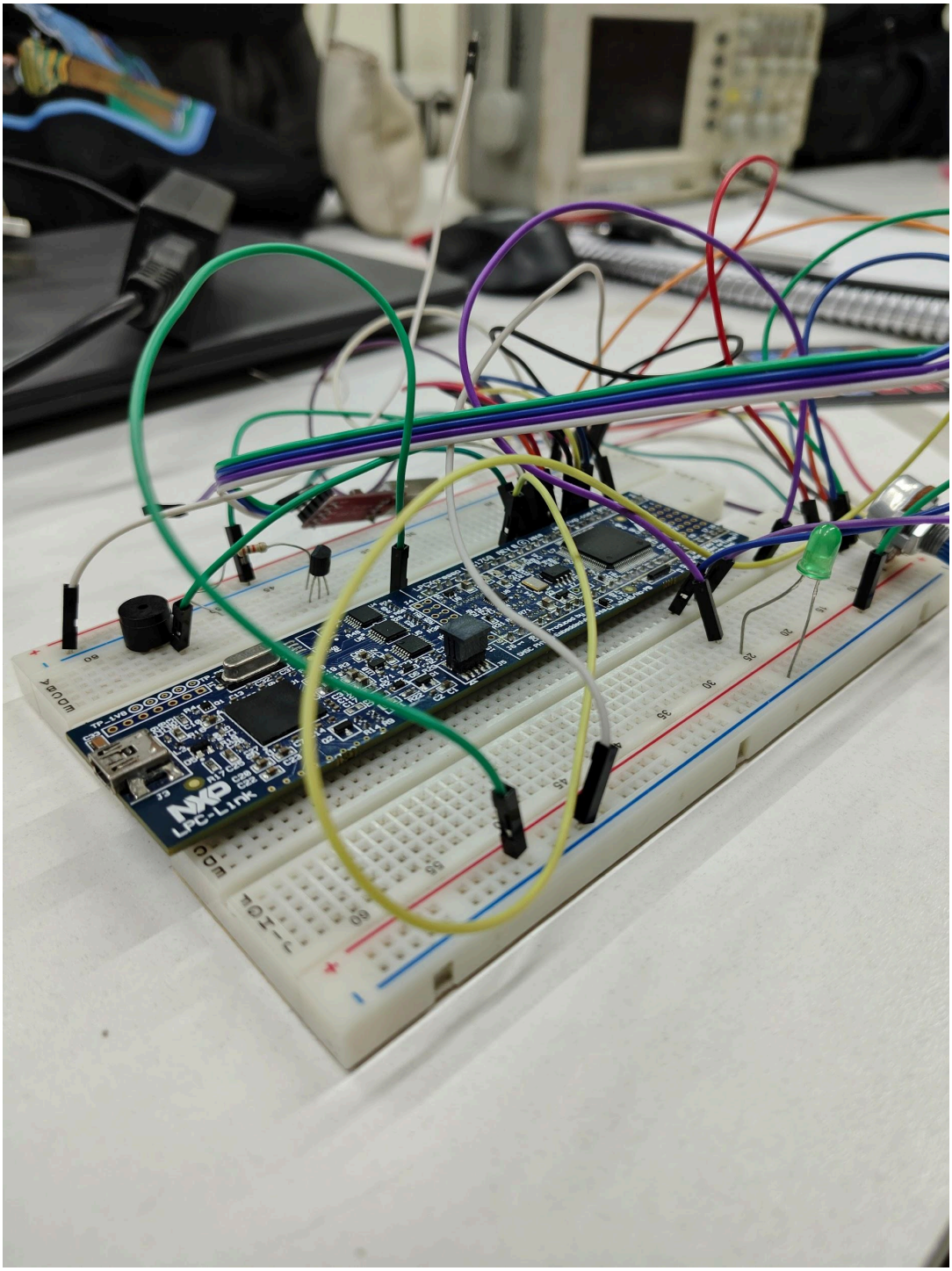
- n es el número de la nota en el teclado estándar.

Quedando la tabla:

Octava	DO	RE	MI	FA	SOL	LA	SI
1	262	294	330	349	392	440	494
2	523	587	659	699	784	880	988
3	1047	1175	-	-	-	-	-

RESULTADOS




NXP
LPC-Link

CONCLUSIONES

La realización de este proyecto fue un desafío mayor de lo que anticipamos, pero también una experiencia de gran aprendizaje. Aunque enfrentamos problemas iniciales, como una incompatibilidad entre módulos, que nos obligaron a replantear la lógica del sistema y cambiar de DMA-DAC a Timer-DAC, cada obstáculo fue una oportunidad para aplicar los conocimientos adquiridos en clase y explorar nuevas soluciones. Gracias a la perseverancia y trabajo en equipo, logramos superar las dificultades y concretar con éxito la idea central del proyecto.