Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0624: Laboratorio de Microcontroladores

II ciclo 2024

Laboratorio # 2
GPIOs, Timers y FSM

Christabel Alvarado Anchía - B80286
Erick Marín Rojas - B94544
Profesor: MSc. Marco Villalta Fallas

18 de setiembre de 2024

Índice

Índice de figuras

Índice de tablas

1. Resumen

En este laboratorio se tiene como objetivo principal familiarizarse con el funcionamiento del ATtiny 4313. Para lograr esto se desarrolló un juego de Simón dice en el que se cuenta con 4 leds de distintos colores y 4 botones correspondientes, el sistema ilumina los leds en cierto orden y el usuario debe presionar los botones correspondientes siguiendo el mismo patrón. Además, el juego emite sonidos al iluminar los leds y mientras no se encuentre en una partida presionar cualquier botón permite iniciar la partida. El diseño e implementación de este juego permitió a los estudiantes comprender de mejor manera el uso de GPIOs, interrupciones, timers y FSM. En este informe se resumen los aspectos más importantes del ATtiny 4313; luego, se explica y analiza tanto el hardware como el software diseñados para simular el juego Simon dice.

Repositorio de GitHub `https://github.com/ErickMaRi/Laboratorios-Grupales-IE0624`

2. Nota teórica

2.1. ATtiny4313

El microcontrolador de 8 bits, ATtiny4313, pertenece a la familia AVR (basados en arquitectura RISC). Según la hoja del fabricante [1] se considera un microcontrolador de alto rendimiento y bajo consumo, con 120 instrucciones. Además, cuenta con una memoria flash de 4k bytes, una SRAM de 256 bytes y una EEPROM de 256 bytes. Respecto a pines de entrada y salida cuenta con 18 GPIOs, 1 USART full duplex, USI, cuatro canales PWM, un timer de 8 bits y otro de 16 bits.

2.1.1. Diagrama de pines

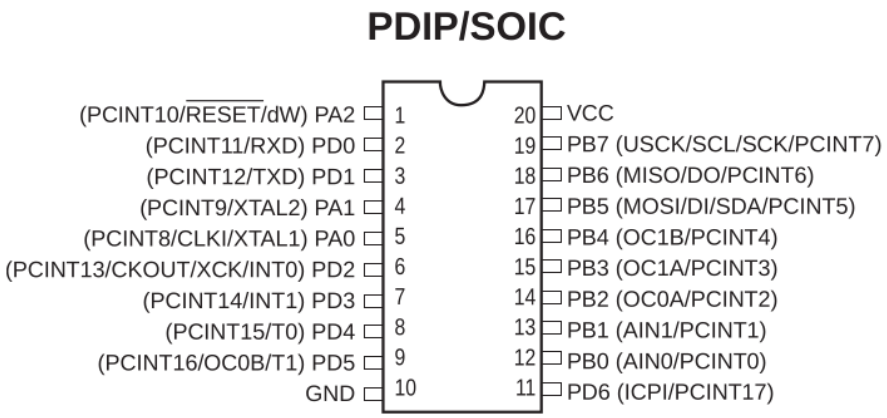

Figura 1: Diagrama de pines de ATtiny4313 [1]

2.1.2. Diagrama de bloques

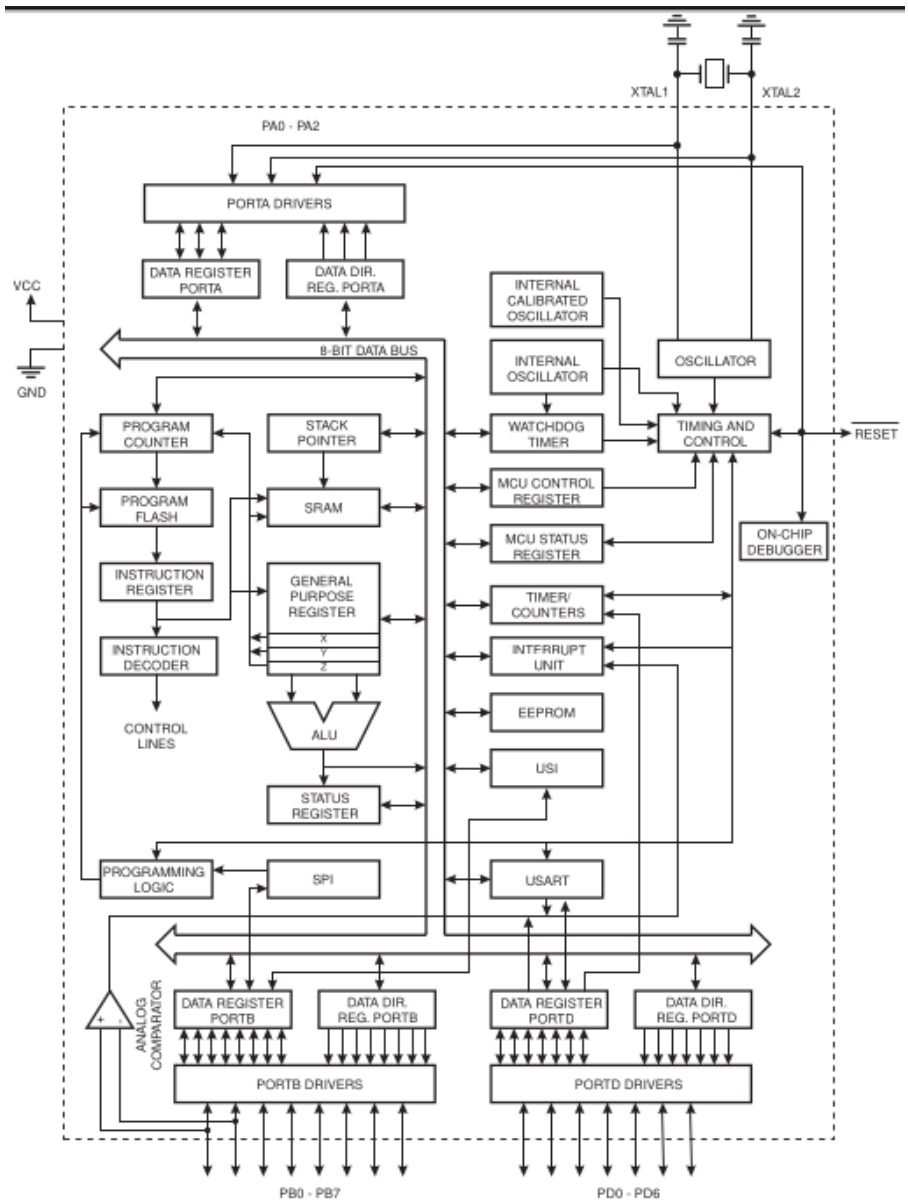

Figura 2: Diagrama de bloques de ATtiny4313 [1]

2.1.3. Registros relevantes

Para el desarrollo de este laboratorio resulta necesario conocer detalles sobre varios registros del ATtiny4313, entre estos destacan:

- **DDRB:** Se utiliza para definir los pines del puerto B como salidas o entradas. Es un registro de 8 bits donde 1 corresponde a salida y 0 a entrada. DDRA y DDRD se comportan de manera similar.

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3: Descripción de DDRB [1]

- **PORTB:** Permite encender y apagar los pines del puerto B, contiene resistencias de pull up internas. PORTA y PORTD se comportan de manera similar.

Bit	7	6	5	4	3	2	1	0	
0x18 (0x38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4: Descripción de PORTB [1]

- **GIMSK:** Este registro es utilizado para habilitar o deshabilitar distintas interrupciones.

Bit	7	6	5	4	3	2	1	0	
0x3B (0x5B)	INT1	INT0	PCIE0	PCIE2	PCIE1	–	–	–	GIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5: Descripción de GIMSK [1]

- **PCMSK1:** Permite habilitar o deshabilitar las interrupciones por cambio de estado en los pines de PORTD. Tiene un comportamiento similar a PCMSK2

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	–	–	–	–	–	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 6: Descripción de PCMSK1 [1]

- **MCUCR:** Permite habilitar los modos de interpretación de interrupciones.

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	PUD	SM1	SE	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 7: Descripción de MCUCR [1]

- **TIMSK:** Permite seleccionar el tipo de interrupción generada por el contador 0 (overflow).

Bit	7	6	5	4	3	2	1	0	
0x39 (0x59)	TOIE1	OCIE1A	OCIE1B	–	ICIE1	OCIE0B	TOIE0	OCIE0A	TIMSK
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 8: Descripción de TIMSK [1]

- **TCNT0:** Permite acceder, en modo escritura o lectura, al contador de 8 bits del timer.

Bit	7	6	5	4	3	2	1	0	
0x32 (0x52)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 9: Descripción de TCNT0 [1]

- **TCCR0B:** Permite configurar la fuente de reloj del registro B. TCCR0A tiene un comportamiento similar.

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 10: Descripción de TCCR0B [1]

2.2. Conceptos importantes para el diseño del circuito

2.2.1. FSM: Maquina de estados finita

Una maquina de estados consiste en un algoritmo que indica una serie de instrucciones a ser ejecutadas según el estado que se ejecute. Cada estado representa un comportamiento específico del sistema, y las transiciones entre distintos estados se activan por eventos o entradas específicas. En C las FSM pueden ser implementadas usando if-else o switch.
Las maquinas de estado finitas se pueden dividir en deterministas y no deterministas. Las deterministas se caracteriza porque desde cualquier estado, solo hay una transición para cualquier entrada permitida. [2] Mientras que las no deterministas varían en que dada una entrada desde un estado particular, esta puede conducir a más de un estado diferente. [2]

2.2.2. Resistencias limitadoras de corriente

En el diseño de circuitos es común conectar una resistencia en serie con los LEDs a utilizar para proteger al mismo de posibles excesos de corriente. Para calcular el valor adecuado de estos resistores se utiliza la siguiente ecuación:

$$R = \frac{V_{DD} - V_{LED}}{I_{max}} \tag{1}$$

Para efectos de este laboratorio ese sabe que el voltaje del ATtiny4313 es 5V , la corriente forward, según el simulador, es de 20mA y el voltaje forward es 2V. Con esto se obtiene que la resistencia debe ser de 150Ω; sin embargo, se decidió utilizar resistencias mayores para evitar errores, considerando que se trabajó con múltiples LED.

2.2.3. Filtro paso bajo

Los filtros paso bajo permiten limitar las señales que lo atraviesan, de manera que solamente pasan las frecuencias bajas. En el procesamiento de audio utilizar filtros de paso bajo trae como beneficio un sonido más suave o dulce. [3]. Además, ayuda a eliminar picos y ruido en la señal y a evitar sobrepaso de corriente.

2.3. Componentes

Para el diseño del circuito se deben utilizar los siguientes componentes, con un precio total de ₡3251. Para encontrar los precios se utilizaron sitios de venta de componentes electrónicos dentro de Costa Rica, como MicroJPM y Mouser.

Componente	Cantidad	Precio por unidad
ATtiny4313	1	₡833
Resistor 220Ω	4	₡26
Resistor 18$k\Omega$	1	₡26
Capacitor 47nF	1	₡52
LED rojo	1	₡259
LED azul	1	₡259
LED amarillo	1	₡259
LED verde	1	₡259
Botón	4	₡105
Buzzer	1	₡780
Amplificador	1	₡494

Tabla 1: Componentes utilizados

3. Desarrollo

Se detalla el procedimiento llevado a cabo para la implementar el juego de Simon en el microcontrolador ATtiny4313. Se explica tanto la configuración de hardware como la implementación del código en C.

3.1. Configuración de Hardware

El hardware se configuró de la siguiente manera:

- **LEDs:** Se conectaron cuatro LEDs (rojo, azul, amarillo y verde) a los pines PB0, PB1, PB2 y PB3 respectivamente, utilizando resistencias de 220Ω para limitar la corriente.
- **Botones:** Se utilizaron cuatro botones conectados a los pines PA1, PD1, PD2 y PD3. Estos pines se configuraron como entradas con resistencias pull-up internas activadas, para detectar la pulsación de los botones, que referirían el pin a la tierra.
- **Buzzer:** Un buzzer se conectó al pin PB4, permitiendo la generación de sonidos mediante modulación de frecuencia.
- **Alimentación:** El microcontrolador se alimenta con 5V por defecto en el ambiente de SimulIDE.

Las conexiones se pueden apreciar en la figura 11.

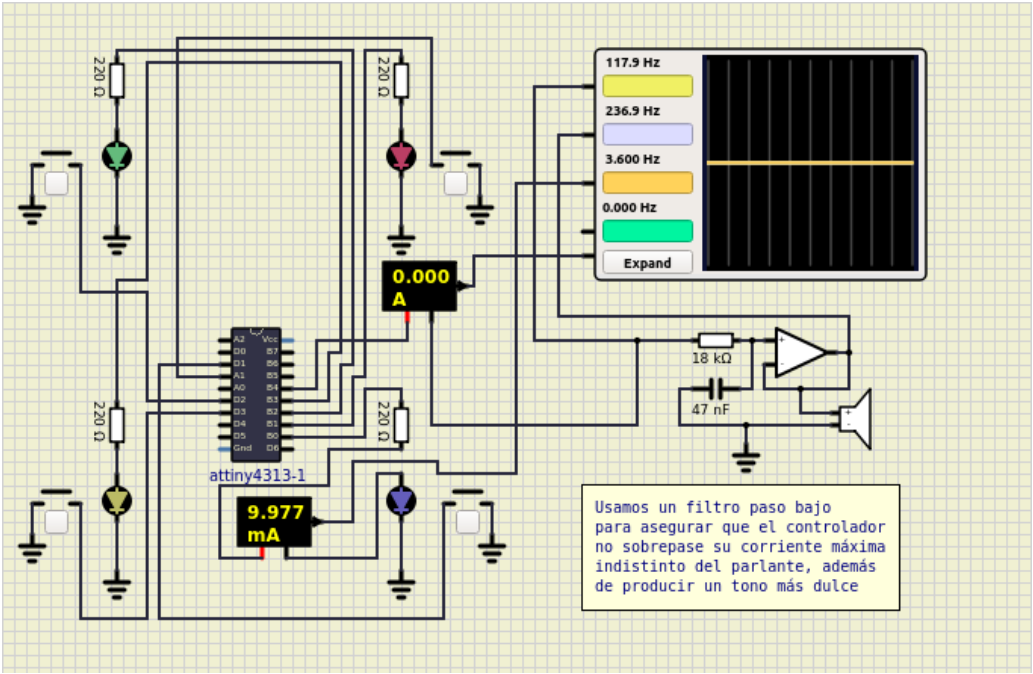

Figura 11: Podemos observar los LEDS de los distintos colores encendidos al inicio del juego, cada uno usando casi 10 mA de los 60 disponibles.

3.2. Implementación del Código

El código fue desarrollado en C, utilizando las librerías estándar del AVR. Se detallan las partes más importantes del código y su función dentro del programa. Se encienden los LEDs mientras se produce una nota, para las entradas y salidas.

3.2.1. Variables Globales y Definiciones

Se definieron constantes para los estados de la máquina de estados finitos (FSM) y para las frecuencias de las notas musicales utilizadas:

```
#define ESPERA 0
#define INICIO 1
#define MOSTRAR 2
#define REVISAR 3
#define FIN 4

#define NOTA1 392 // G4
#define NOTA2 440 // A4
#define NOTA3 493 // B4
#define NOTA4 523 // C5
```

Se eligieron las notas sin partir de un acorde, arpegio o arreglo de frecuencias en específico, simplemente se seleccionaron cuatro entre diez iniciales, que sonaban bien juntas, debido a la baja fidelidad del atraso en microsegundos.

Las variables globales principales son:

- **estado:** Indica el estado actual.
- **entrada_usuario:** Almacena el botón que el usuario presionó.
- **secuencia[13]:** Almacena la secuencia de LEDs que el usuario debe repetir.
- **turno:** Lleva el conteo del nivel actual en el juego para controlar la dificultad.
- **indice_secuencia:** Utilizado para comparar la entrada del usuario con la secuencia.
- **enable y overflow_cont:** Utilizados para el control de retardos con el temporizador.

3.2.2. Configuración de Puertos y Registros

En el método `main()`, se realiza la configuración de los puertos y registros necesarios:

- **Configuración de salidas:** Se configuran los pines PB0 a PB3 y PB4 como salidas para controlar los LEDs y el buzzer.
- **Configuración de entradas:** Los pines PA1, PD1, PD2 y PD3 se configuran como entradas con resistencias pull-up activadas.
- **Configuración de interrupciones:** Se habilitan las interrupciones por cambio de pin (*Pin Change Interrupts*) en los puertos A y D, permitiendo detectar cuando se presiona un botón.
- **Configuración del Timer0:** Se configura el Timer0 en modo normal con un prescaler de 64, y se habilita la interrupción por desbordamiento para contar los retardos.

Los registros utilizados para estas configuraciones son:

- **DDRB, DDRA, DDRD:** Configuran los pines como entradas o salidas.

- **PORTA, PORTD:** Activan las resistencias pull-up internas.
- **GIMSK, PCMSK1, PCMSK2:** Configuran las interrupciones por cambio de pin.
- **MCUCR:** Configura las interrupciones externas.
- **TIMSK, TCNT0, TCCR0A, TCCR0B:** Configuran el Timer0 y sus interrupciones.

3.2.3. Funciones Principales

- `parpadear(int n)`: Hace que todos los LEDs parpadeen `n` veces, utilizado para indicar el inicio o fin del juego.
- `encenderLed(int n)`: Enciende el LED correspondiente al número `n` y reproduce una nota asociada. Utiliza la función `tocarNota()` para generar el sonido.
- `delay(int ms)`: Implementa un retardo de `ms` milisegundos utilizando el Timer0 y las interrupciones por desbordamiento.
- `delay_us(unsigned int us)`: Implementa un retardo en la escala de los microsegundos declarándo tres enteros volátiles.
- `tocarNota(int frecuencia)`: Genera un tono en el buzzer a la frecuencia especificada, controlando el ciclo de encendido y apagado del pin PB4. Utiliza `delay_us(unsigned int us)` para controlar la frecuencia.
- `tonadaInicio()` **y** `tonadaError()`: Reproducen melodías específicas al inicio del juego y cuando ocurre un error, respectivamente, usan `tocarNota(int frecuencia)` para producir las tonadas.
- `imprimirSecuencia()`: Muestra la secuencia de LEDs que el usuario debe memorizar, ajustando la duración según el nivel actual.
- `iniciarsSecuencia()`: Genera una secuencia aleatoria de números entre 0 y 3 utilizando una semilla fija.
- `FSM()`: Implementa la máquina de estados finitos que controla la lógica del juego, gestionando los estados ESPERA, INICIO, MOSTRAR, REVISAR y FIN.

3.2.4. Interrupciones

Se implementaron las siguientes rutinas de servicio de interrupción (ISR):

- `ISR(TIMER0_OVF_vect)`: Maneja el desbordamiento del Timer0, incrementando el contador de overflows para controlar los retardos en milisegundos.
- `ISR(PCINT1_vect)` **y** `ISR(PCINT2_vect)`: Detectan cambios en los pines de entrada (botones presionados) y actualizan la variable `entrada_usuario` con el botón presionado, además de reproducir la nota correspondiente.

3.2.5. Máquina de Estados Finita (FSM)

La FSM se implementa en la función `FSM()`, y controla el flujo del juego mediante los siguientes estados:

1. **ESPERA:** El juego espera a que el usuario presione cualquier botón para iniciar.
2. **INICIO:** Se reproduce la tonada de inicio y los LEDs parpadean dos veces. Se inicializa la secuencia aleatoria y se reinicia el contador de turnos.
3. **MOSTRAR:** Se muestra la secuencia de LEDs al usuario. La velocidad de presentación disminuye ligeramente con cada turno para aumentar la dificultad.
4. **REVISAR:** El juego espera la entrada del usuario y compara cada botón presionado con la secuencia. Si el usuario acierta, continúa; si falla, cambia al estado FIN.
5. **FIN:** Se reproduce la tonada de error y los LEDs parpadean tres veces. El juego regresa al estado ESPERA.

3.2.6. Procedimiento de Juego

El flujo del juego es el siguiente:

1. **Inicio en ESPERA:** El programa inicia y espera que el usuario presione un botón.
2. **Transición a INICIO:** Al detectar una entrada, el juego comienza. Se reproducen señales visuales y sonoras para indicar el inicio.
3. **Generamos y Mostramos Respuesta:** Se genera una secuencia aleatoria de LEDs y se muestra al usuario, incrementando en longitud en cada turno.
4. **Revisamos la Entrada del Usuario:** El usuario intenta repetir la secuencia presionando los botones correspondientes. El programa verifica la exactitud de cada entrada en tiempo real.
5. **Progresión o Finalización:** Si el usuario completa correctamente la secuencia, el juego avanza al siguiente turno. Si el usuario falla, se indica el error y el juego termina, regresamos a ESPERA.

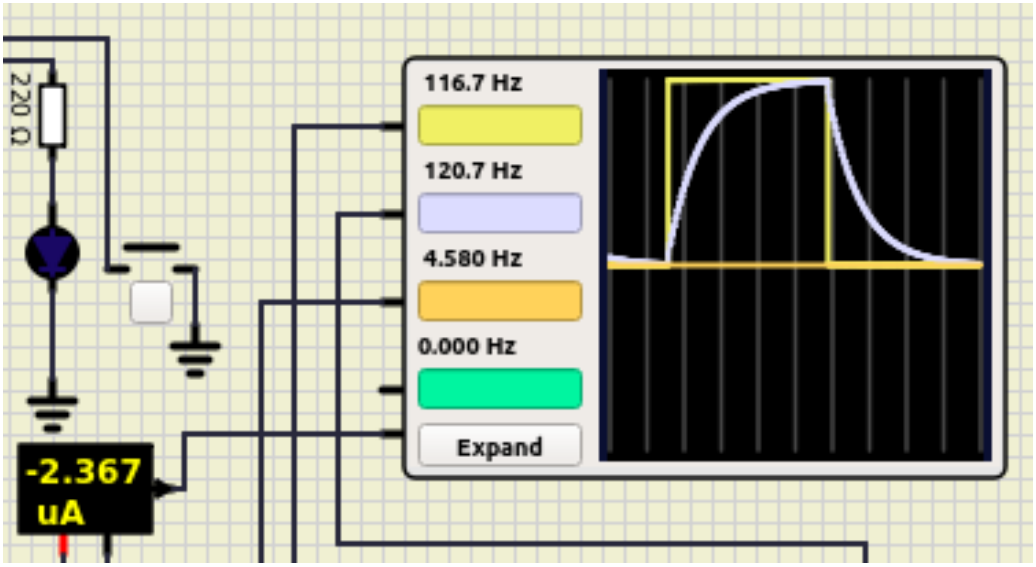

Figura 12: Se filtran los componentes de alta frecuencia de la señal cuadrada para producir un tono más dulce, se observa una corriente baja en el pin del controlador debido a la alta impedancia del circuito paso bajo usado. Aunque la corriente de entrada al filtro oscila bastante, lo hace por debajo de 1 mA.

3.3. Pruebas y Resultados

Se realizaron pruebas para verificar el correcto funcionamiento del juego. Usando el sistema diseñado en un ambiente simulado se comprobó que:

- El sistema espera a que el usuario presione un botón para iniciar el juego.
- Los LEDs y el buzzer responden correctamente a las señales del programa, siguiendo el tiempo que el enunciado solicita.
- Se muestra la secuencia generada al usario exitosamente.
- Se reciben las entradas del usuario a través de los botones.
- El juego incrementa la dificultad de forma progresiva, con 200 milisegundos menos por vez.
- La máquina de estados realiza las transiciones necesarias para poder jugar Simon.
- Para asegurar que el microcontrolador no sobrepase los límites de corriente indistinto del parlante a usar, se le separa con un op-amp del actuador, esto también se usa para producir una señal con un tono más dulce, como podemos observar en la figura 12.
- Se usa un buzzer para realimentad un sonido distinto por cada LED-botón, realimentando la idea de que están relacionados y haciendo el juguete más entretenido.

3.4. Consideraciones Adicionales

- **Generación de Números Aleatorios:** Se utilizó una semilla fija para `rand()`, por lo que la secuencia utilizada siempre es la misma, esto se hizo así para efectos de desarrollo pero para nivel de producción la semilla se debe leer de algún temporizador.

- **Temporización Imprecisa:** El `_delay_us()` aproxima un delay en la escala de los microsegundos instanciando enteros volátiles tres veces, por lo que no se debe fiar de la temporización para aplicaciones sensibles.
- **Limitación de la Corriente en los LEDs:** Para efectos de SimulIDE la corriente en los LEDs no cambia respecto al color por lo que no se dimensiona una resistencia distinta por color.

4. Conclusiones y recomendaciones

- Se logró simular de manera exitosa el juego simón dice según las especificaciones indicadas en el enunciado del laboratorio.
- Se implementó una máquina de estados finita de manera correcta, logrando representar el juego simón dice de manera clara.
- El uso de amperímetros y el osciloscopio permite observar el comportamiento correcto de las diversas señales del sistema.
- Es de suma importancia comprender el funcionamiento de los registros e interrupciones del microcontrolador para poder generar el código necesario de manera exitosa. Por esto se recomienda leer detalladamente la hoja del fabricante previo a iniciar el desarrollo del proyecto.
- Separando el controlador del resto del circuito logramos alcanzar corrientes en los pines bajas para la modulación de ancho de pulso que se usó en el buzzer.

Referencias

[1] A. Corporation, *ATtiny2313A/4313 Data Sheet*, 2011. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/doc8246.pdf

[2] M. Shead, "State machines: the ultimate guide to automata in computer science," 2018, accessed: 2024-09-18. [Online]. Available: https://www.freecodecamp.org/news/state-machines-basics-of-computer-science-d42855debc66/

[3] M. Alessi, "¿qué es un filtro paso bajo y cómo se utiliza en las mezclas?" 2024, accessed: 2024-09-18. [Online]. Available: https://emastered.com/es/blog/low-pass-filter

5. Anexos