Juego de Reacción en ESP32 con Estímulos Visuales y Sonoros usando Micro Python

Angie Marisela García Ríos
Estefany Cuervo Suárez
Juan Felipe Arroyave Zapata
Electrónica digital II

Índice

1. Identificación del proyecto

- **Nombre del proyecto:** Juego de Reacción en ESP32 con Estímulos Visuales y Sonoros usando Micro Python
- **Integrantes del equipo: Angie Marisela García Ríos, Estefany Cuervo Suárez, Juan Felipe Arroyave Zapata**

2. Resumen

El proyecto consiste en un sistema de medición de reflejos basado en ESP32 que genera estímulos visuales (LEDs) y sonoros (buzzer) aleatorios. Los jugadores deben responder presionando el botón correspondiente al estímulo activado lo más rápido posible. El sistema mide los tiempos de reacción, asigna puntuaciones y determina al ganador según diferentes modos de juego.

3. Descripción del hardware

- **ESP32** (modelo con ADC y pines GPIO)
- **Pulsadores conectos a los pines:** 25, 26, 27, 14, , 34, 33, 35, 32, 12, 13, 16
- **Led Rojo** conectado al pin 22
- **Led Multicolor** conectado al pin 21
- **Led Verde** conectado al pin 23
- **Buzzer** conectado al pin 19
- **Resistencia conectadas a:** leds, pulsadores y buzzer

Diagrama de conexión:

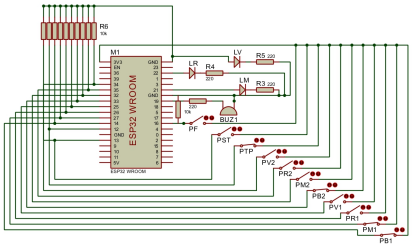
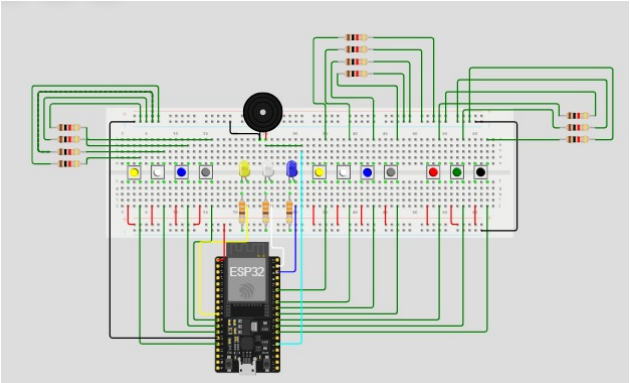


Diagrama de Conexión



Esquema de Wokwi

4. Descripción del software

- **Lenguaje usado:** MicroPython
- **Librerías:** `machine`, `time`, `random`
- **IDE:** Thonny

Flujo general del programa:

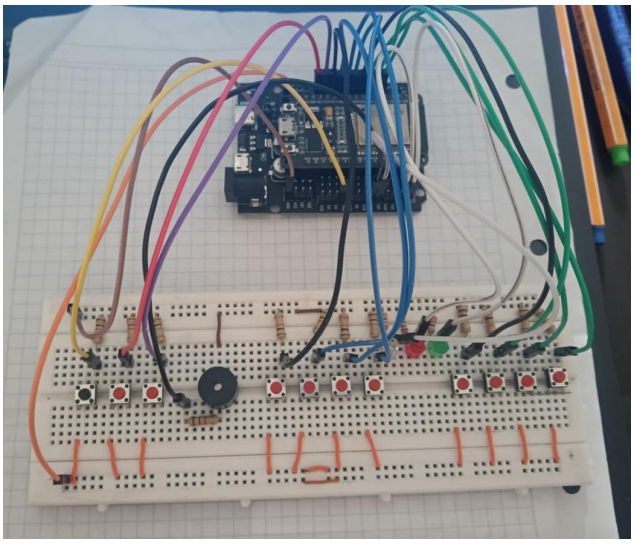
1. Configuración de pines y estados iniciales
2. Lectura de entradas, verificación de botones de jugadores y control
3. Generación de estímulos, activación aleatoria de LEDs o buzzer
4. Medición de tiempos, cálculo de tiempos de reacción
5. Evaluación de respuestas, asignación de puntuaciones según aciertos/errores
6. Activación de salidas, retroalimentación visual y auditiva según resultados

5. Estructura del código

- Archivo principal: `main.py`
- No se usan módulos adicionales.
- Código lineal dentro de un ciclo infinito con lectura y decisión en tiempo real.

6. Explicación de funciones principales

- `Pin.value()`: Lee el estado de un pin digital (0 o 1)
- `PWM.duty()`: Establece el ciclo de trabajo del PWM (0-1023)
- `PWM.freq()`: Establece la frecuencia del PWM en Hz
- `Pin.on() / Pin.off()`: Activa/desactiva salidas digitales
- `time.ticksms()`: Obtiene el tiempo actual en milisegundos
- `time.ticksdiff()`: Calcula la diferencia entre dos tiempos



Evidencia visual del montaje en protoboard

7. Funciones de lógica del juego

- `generarEstimulo()`: Genera un número aleatorio (0-3) para seleccionar estímulo
- `activarEstimulo()`: Establece el ciclo de trabajo del PWM (0-1023)
- `botonPresionadoDebounce()`: Detecta pulsaciones de botones con antirrebote
- `jugarRondaNormal()`: Ejecuta una ronda completa en modo normal
- `jugarRondaFest())`: Ejecuta una ronda completa en modo fest
- `calcularPromedioTiempos()`: Calcula el promedio de tiempos de respuesta

8. Funciones de retroalimentación

- `beep()`: Emite sonidos de confirmación o error
- `mostrarPuntuacion()`: Muestra el marcador actual por terminal
- `mostrarResultadosFinales()`: Presenta los resultados finales del juego

9. Procedimiento de Prueba

1. Conectar el ESP32 al computador mediante cable USB
2. Cargar el código en el ESP32 usando Thonny IDE
3. Abrir consola serial para visualizar mensajes del sistema
4. Verificar que todos los LEDs estén apagados al inicio
5. Presionar botones para confirmar funcionamiento correcto
6. Iniciar juego y responder a estímulos para validar medición de tiempos
7. Verificar asignación correcta de puntuaciones en consola
8. Probar modo fest y validar bonificación de puntos

10. Manejo de Errores y Seguridad

10.1. Prevención de Errores

- Antirrebote implementado en software para todas las entradas
- Validación de selección de jugadores (solo se permiten 1 o 2 jugadores)
- Manejo de excepciones en entrada por teclado
- Uso de `time.ticks_diff()` para evitar desbordamiento de tiempos
- Límites establecidos en valores aleatorios (1-10 segundos en modo normal, 0.5-2 segundos en modo fest)

10.2. Recuperación de Errores

- Función `inicializar_salidas()` para restablecer estado conocido
- Reinicio de variables al comenzar nueva partida
- Botón STOP para finalización segura en cualquier momento
- Implementación de antirrebote en interrupción de modo fest

10.3. Limitaciones y Consideraciones de Seguridad

- **Protección de hardware**:
  - Se utilizan resistencias limitadoras para LEDs y buzzer
  - Configuración PULL_DOWN en todas las entradas para evitar estados flotantes
  - Los pines GPIO del ESP32 tienen protección limitada contra cortocircuitos
- **Estabilidad del sistema**:
  - Uso de delays no bloqueantes en loops principales
  - Gestión adecuada de memoria para evitar desbordamientos
  - No se utilizan llamadas recursivas problemáticas
- **Consideraciones importantes**:
  - No se implementa protección por watchdog timer
  - No hay mecanismo de reconexión automática en caso de reinicio
  - El sistema depende de una alimentación estable por USB
  - Se asume que los botones pueden presentar rebotes; se implementa antirrebote por software

10.4. Monitorización

- Mensajes de depuración en terminal para diagnóstico
- Indicación de estado del sistema en tiempo real
- Registro de eventos importantes durante la ejecución
- Visualización continua de puntuaciones en terminal