Controlador de percusión electrónica

Autores:
Felipe Alberto Delgado Jiménez
Esteban Arley Cortes Rincon

Sistemas Embebidos

Universidad Nacional de Colombia

24 de julio de 2025

Índice general

Capítulo 1: Introducción al Sistema

1.1 Descripción General

El sistema implementa un controlador de percusión electrónica que utiliza paneles equipados con sensores piezoeléctricos para capturar los impactos, los cuales son convertidos a señales digitales mediante dos módulos ADS1115 (convertidores analógico-digital de 16 bits). Estos controlan un sintetizador de sonidos desarrollado en Faust, capaz de generar diferentes timbres percusivos en tiempo real.

Adicionalmente, el sistema incorpora una interfaz gráfica controlada mediante un D-Pad que permite al usuario reasignar dinámicamente los sonidos a cada panel físico, ofreciendo así una personalización inmediata del set de percusión durante la ejecución.

1.2 Arquitectura del Sistema

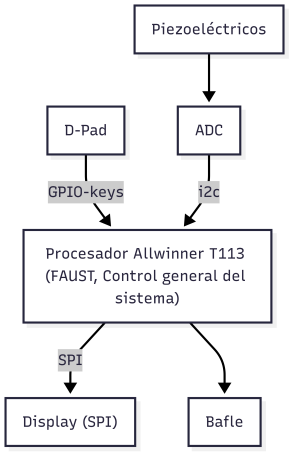

Figura 1.1: Diagrama de Bloques Proyecto

Capítulo 2: Componentes del Sistema

2.1 ADS1115

El ADS1115 es un convertidor analógico-digital (ADC) de 16 bits de precisión que se comunica a través del protocolo I2C. Este dispositivo es capaz de realizar conversiones en cuatro canales diferenciales o en cuatro canales simples, con un rango de voltaje programable desde $\pm$6.144V hasta $\pm$0.256V.

Se implementaron 5 funciones basicas, considerando que tenemos 2 ADS1115 disponibles para un total de 8 canales, uno con una dirección i2c por defecto de 0x48 y otro condirección 0x49.

2.1.1 Funciones Principales

- ADS1115_init(int channel, int file)

  Abre el bus I2C en `/dev/i2c-3` y devuelve el descriptor de archivo. Encaso de que este bus no este inicializado retorna -1.

  En este archivo, el kernel escribe los eventos que ocurran en el bus i2c, y sobre este se escriben los comandos y datos que se quieran usar.

  Por ejemplo,

  ```
  ioctl(file, I2C_SLAVE, addr)
  ```

  indica al bus i2c que se va a interactuar con un dispositivo esclavo en la direccion *addr*.

  ```
  write(file, config, 3)
  ```

  escribira los 3 bits que estan almacenados en la variable *config* sobre el bus i2c, el cual ya debe estar apuntando a la direccion del dispositivo.

```
read(file, config_status, 2
```

leera 3 bits del dispositivo i2c al que se esté apuntando, y los guardara en la variable *config_status*.

- ADS1115_exit()

  Cierra la comunicación I2C:

- ADS1115_start_reading

  Envia al ADS1115 los dos bits de configuracion requeridos para comenzar una conversión en el canal especificado.

```
uint16_t config_value =
    ADS1115_CONFIG_OS_SINGLE |
    ADS1115_CONFIG_MUX_SINGLE  |
    ADS1115_CONFIG_MODE_SINGLE |
    ADS1115_CONFIG_DR_860SPS |
    ADS1115_CONFIG_CQUE_NONE |
    (channel << 12);
```

  La funcion acepta canales del 0 al 7, siendo los canales 0-3 en la direccion i2C 0x48 y los canales 4-7 en la direccion 0x49.

  El ADC permite variar la velocidad de escritura, modo de adquisicion (unico o continuo), o activar el modo comparador. En este caso usamos modo unica lectura, dado que cada vez se toman datos de un canal distinto, a 860SPS y con el comparador desactivado.

  Una vez se envia este comando, el dispositivo empieza la lectura en el canal indicado.

- ADS1115_get_result(int channel, int file)

  Esta función retorna el resultado de la lectura previamente iniciada con la función *start reading*.

  El argumento *channel* se usa para saber de cual de los dos dispositivos se quiere obtener la lectura.

Esta función aprovecha un bit del registro de configuración, el bit OS, que es el bit mas significativo. Este indica si se esta realizando una conversión en el momento. Cuando este pasa a ser 0 la conversión esta completa.

```
if (config_status[0] & 0x80) {
    break; // Conversion complete
}
```

En caso de que el bit OS, no pase a ser 0 en 100ms la función retorna -1 e imprime errot en consola.

- ADS1115_read(int channel, int file)

  Combina ADS1115_start_reading con ADS1115_get_result(int channel, int file)

  Retorna los dos bits del resultado de la conversion.

2.1.2 Adaptación de Impedancia

Al conectar los piezoelectricos directamente a un ADC, el resto de canales sufren de interferencia, esto debido a la capacitancia que presentan los piezoelectricos. Por lo cual se tuvo que realizar un pequeño circuito seguidor, usando un amplificador operacional para cada piezoelectrico.

2.2 Faust

Faust (Functional Audio Stream) es un lenguaje de programación de alto nivel diseñado específicamente para la síntesis y el procesamiento de señales de audio en tiempo real. Se basa en el paradigma funcional, lo que permite describir procesos de audio como funciones matemáticas. Faust está optimizado para generar código altamente eficiente en C++, lo cual facilita su integración en plataformas como Linux, microcontroladores, VSTs, entornos embebidos, y más.

El enfoque principal de Faust es permitir a desarrolladores, músicos e ingenieros de audio construir algoritmos de procesamiento de señal (DSP) de manera compacta, modular y reutilizable. Además, su arquitectura permite generar interfaces gráficas automáticas y exportar el código a distintos entornos de ejecución

Se utilizó Faust para simular 6 elementos: bombo, caja, hit-hat, tom, y platillo crash. Cada componente fue modelado utilizando funciones que sintetizan el sonido correspondiente, emulando características acústicas básicas como el ataque, la resonancia, la duración, y el timbre.

El diseño en Faust se organizó en módulos independientes para cada parte de la batería, permitiendo un control individual de sus parámetros (frecuencia base, envolvente, ganancia, etc.). Los sonidos fueron generados combinando osciladores (como ruido blanco para la caja o charles, y senoides amortiguadas para el bombo y toms) con filtros paso banda y envolventes ADSR (Attack, Decay, Sustain, Release).

El siguiente fragmento de código Faust representa, por ejemplo, un generador básico de sonido

```
// Parámetros de control OSC
level = hslider("/noise/level", 0, 0, 2, 0.001) : si.smoo;
gain = hslider("gain", 0.8, 0, 1, 0.01);
trigger_threshold = hslider("trigger_threshold", 0.1, 0, 1, 0.001);

// Detección de trigger basada en nivel
trigger = level > trigger_threshold : ba.impulsify;
// Generador de kick drum sintético
kick_freq = 60; // Frecuencia base del kick
kick_decay = 0.3; // Tiempo de decay
kick_env = trigger : en.ar(0.001, kick_decay);
kick_pitch_env = trigger : en.ar(0.001, 0.05) * 30;
kick_osc = os.osc(kick_freq + kick_pitch_env) * kick_env;
kick_noise = no.noise * kick_env * 0.3;
kick_sound = (kick_osc + kick_noise) * 0.7;
```

Cada uno de los 6 sonidos se puede activar mediante señales de control (como pulsos digitales, teclas o MIDI), lo que permite tocar la batería de forma interactiva o automática.

2.3 Display TFT con controlador st7738

Para la integracion de la interfaz, se uso un display de 1.9’ con resolución 160x128 pixeles. Este se controla mediante la interfaz SPI. La cual en nuestra placa no tiene soporte por

Hardware en los pines disponibles. Por lo cual se tuvo que habilitar el SPI bitbanging, que se hace en la configuración del Kernel.

En *buildroot* con el comando *linux-menuconfig*, habilitando las opciones:

- Device Drivers → SPI support [*]
- Device Drivers → SPI support → Utilities for Bitbanging SPI masters [*]
- Device Drivers → SPI support → GPIO-based bitbanging SPI Master[*]

La asignación de los pines de GPIO usados para esto, se hace en el árbol de dispositivos. Añadiendo el siguiente bloque:

```
\{
    compatible = "allwinner,t113";

    spi_gpio: spi-gpio {
        compatible = "spi-gpio";
        gpio-sck = <&pio 4 11 GPIO_ACTIVE_HIGH>;
        gpio-mosi = <&pio 4 10 GPIO_ACTIVE_HIGH>;
        gpio-miso = <0>; // unused
        cs-gpios = <&pio 4 8 GPIO_ACTIVE_LOW>;
        num-chipselects = <1>;
        status = "okay";
        spi-delay-us = <1>;
        display@0 {
            compatible = "sitronix,st7735r";
            reg = <0>;
            spi-max-frequency = <16000000>;
                txbuflen = <32768>;
                bpp = <8>;
                dc = <&pio 4 9 GPIO_ACTIVE_HIGH>;
                reset = <0>;  // (RST, unused)
            buswidth = <8>;
            width = <128>;
```

```
            height = <160>;
            rotate = <270>;
            debug = <1>;
            status = "okay";
        };
    };
};
```

Aunque se configuro a una frecuencia maxima de 16MHz, la velocidad real llega a menos de 1MHz. Obteniendo una velocidad de transferencia de tan solo 70kB/s que permiten una tasa de fotogramas muy baja.

2.4 Interfaz Grafica con LVGL

Para el diseño de la interfaz gráfica, se uso SquareLine Studio. Esta interfaz consiste en sencillamente 6 paneles distribuidos en una rejilla 3x2, cada uno representando un canal del ADC, es decir, un panel físico. Y un Slider que representa el volumen actual del sistema.

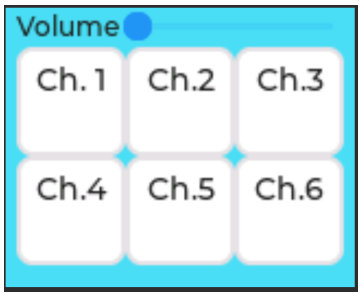

Figura 2.1: Pantalla 1 Interfaz

Se generó también otra pantalla, que es sencillamente un slider en el cual van a estar las distintas opciones de sonido para cada canal.

Figura 2.2: Pantalla 2 Interfaz

SquareLine genera varios archivos, el principal ui.h, que contiene la funcion ui_init(); que inicializa la interfaz, ademas de que importa el resto de archivos de la interfaz. Y ui_Screen1.c y ui_Screen2.c, que crean todos los componentes de la interfaz, que luego referenciaremos en el codigo principal.

La libreria usada se tomo de un repositorio en Github, llamado lv_port_linux. En esta librería se trabajo el resto del proyecto, aprovechándonos de su Makefile.

2.5 D-Pad

Se usaron 6 botones conectados a pines GPIO de la placa simulando un D-Pad. Este tipo de interfaz esta soportado directamente por el kernel compilado en buildroot, se activa desde la configuracion linux-menuconfig, habilitando las opciones:

- Device Drivers → Input device support → Event Interface [*]
- Device Drivers → Input device support → Keyboards [*] → GPIO Buttons [*]

Se añadió el siguiente bloque en el arbol de dispositivos indicando los pines a utilizar:

```
/ {
    gpio-keys {
        compatible = "gpio-keys";
        button-up {
            label = "UP";
            gpios = <&pio 4 6 (GPIO_ACTIVE_HIGH|GPIO_PULL_DOWN)>;
            linux,code = <KEY_UP>;
            debounce-interval = <20>;
```

```
        };
        button-down {
            label = "DOWN";
            gpios = <&pio 4 1 (GPIO_ACTIVE_HIGH|GPIO_PULL_DOWN)>;
            linux,code = <KEY_DOWN>;
            debounce-interval = <20>;
        };
        button-left {
            label = "LEFT";
            gpios = <&pio 4 0 (GPIO_ACTIVE_HIGH|GPIO_PULL_DOWN)>;
            linux,code = <KEY_LEFT>;
            debounce-interval = <20>;
        };
        button-right {
            label = "RIGHT";
            gpios = <&pio 4 5 (GPIO_ACTIVE_HIGH|GPIO_PULL_DOWN)>;
            linux,code = <KEY_RIGHT>;
            debounce-interval = <20>;
        };
        button-select {
            label = "SELECT";
            gpios = <&pio 4 4 (GPIO_ACTIVE_HIGH|GPIO_PULL_DOWN)>;
            linux,code = <KEY_ENTER>;
            debounce-interval = <20>;
        };
        button-return {
            label = "RETURN";
            gpios = <&pio 4 7 (GPIO_ACTIVE_HIGH|GPIO_PULL_DOWN)>;
            linux,code = <KEY_ESC>;
            debounce-interval = <20>;
        };
    };
};
```

Capítulo 3: Integración del Sistema

3.1 Procesamiento de Entradas Analógicas

El sistema emplea dos módulos **ADS1115** conectados mediante I2C para capturar las señales provenientes de seis sensores piezoeléctricos. Cada lectura es evaluada en el bucle principal y procesada por la función `process_ads_triggers`, que detecta *rising/falling edges* respecto a un umbral global `ADS_THRESHOLD = 500`.

Para cada canal se mantienen dos estructuras de estado:

- `prev_ads_values[6]`: almacena el último valor leído para detección de bordes.
- `ads_triggered[6]`: bandera booleana para evitar múltiples activaciones durante un mismo impacto.

Cuando el valor de un canal cruza el umbral hacia arriba y no se encontraba activado (*rising edge*), se envía un mensaje OSC con valor `1.0`. Cuando desciende nuevamente por debajo del umbral (*falling edge*), se envía el valor `0.0`. El mapeo entre canales físicos (ADS) y canales lógicos (Faust) se realiza en tiempo de ejecución, permitiendo flexibilidad en el ruteo de eventos.

3.2 Manejo de la Interfaz Gráfica

La interfaz gráfica fue desarrollada con **LVGL** y maquetada en **SquareLine Studio**. Los seis paneles de pads están dispuestos en una grilla de 3×2 y son navegables mediante un **D-Pad** físico. El sistema cuenta con dos pantallas principales:

1. **Pantalla de ejecución** (`ui_Screen1`): muestra los paneles activos. El D-Pad permite moverse por la grilla y resaltar el panel seleccionado.

2. **Pantalla de configuración** (`ui_Screen2`): permite reasignar dinámicamente el sonido asociado a cada panel usando un control tipo `lv_roller`.

El cambio entre pantallas se realiza con la tecla `ENTER`, mientras que `ESC` se utiliza para *triggerear* manualmente el canal seleccionado (enviando los mensajes OSC correspondientes).

3.3 Comunicación OSC

Se utiliza **liblo** para la comunicación **OSC** con el sintetizador implementado en **Faust**. El destino se crea con:

```
lo_address t = lo_address_new("localhost", "5510");
```

Cada canal lógico se asocia a una ruta OSC del tipo:

```
/drumkit/<nombre_sonido>
```

donde `nombre_sonido` proviene de la enumeración `SoundType` y su tabla `sound_names`. La función `set_channel_trigger` encapsula el envío de los mensajes `ON/OFF` (1.0 / 0.0), permitiendo la abstracción del hardware y simplificando la integración con Faust.

Bibliografía

[1] GRAME-CNCM (s.f.) *FAUST (Functional Audio Stream) Programming Language*. Recuperado de `https://faust.grame.fr/`

[2] Texas Instruments (s.f.) *Hoja de datos del convertidor ADC ADS1115*. Recuperado de `https://www.alldatasheet.es/datasheet-pdf/view/345945/TI/ADS1115.html`

[3] LVGL (s.f.) *lv_port_linux: LVGL configured to work with a standard Linux framebuffer*. Recuperado de `https://github.com/lvgl/lv_port_linux`

[4] ETC1 (s.f.) *Hoja de datos del inversor buffer GD54HCT113*. Recuperado de `https://www.alldatasheet.com/datasheet-pdf/view/126593/ETC1/GD54HCT113.html`