Sistemas Embebidos

Laboratorio 2: TouchPad, WIFI y Webserver

Integrantes del grupo:

- Sebastián Berardi
- Gervasio Hernández
- Joaquín Cuitiño

Repo: https://github.com/epicgerva/Lab1IoT

Tag: https://github.com/epicgerva/Lab1IoT/tree/Lab_2

Docentes: Nicolás Calarco y Pablo Alonso

29/05/2025

Objetivo

Ya con algunas nociones básicas del IDE y la placa de desarrollo del curso, se seguirá avanzando en el control del hardware, se conectará el dispositivo a una red WiFi y generará su propia red, así como también se montará un pequeño servidor web en el dispositivo.

Se comenzará a utilizar la herramienta brindada por Espressif para la configuración de registros de periféricos a través de sus APIs (Interfaz de Programación de Aplicaciones). Con esta herramienta, se configurarán puertos de entrada para el uso de botones, comunicación WiFi y servidor web.

Desarrollo

Primera Parte

En esta primera parte haremos uso de la placa de extensión ESP-LyraP-TouchA v1.1 que cuenta con 6 botones capacitivos y un “anillo de guarda” para aplicaciones “resistentes al agua”.

Lectura de botones mediante polling:

a. Lea las referencias brindadas por Espressif para botones touch y haga una breve descripción de su funcionamiento.

inicialización: se inicia el controlador de pads táctiles con “`touch_pad_init()`”, que establece parámetros predeterminados y elimina la información acerca de que pads han sido tocados previamente y desactiva las interrupciones.

Configuración de pads: se asignan los GPIOs deseados como pads táctiles mediante “`touch_pad_config()`”. Usando la función “`touch_pad_set_fsm_mode()`” se selecciona si las medidas del touch pad se inician automáticamente con el timer del hardware o con el del software a través de “`touch_pad_set_fsm_mode(TOUCH_FSM_SW) + touch_pad_sw_start()`”.

Lectura de Medidas: para obtener valores sin procesar sobre los pulsos se utiliza la función “`touch_pad_read_raw_data()`”, útil para elegir el umbral apropiado a la hora de detectar el pulso.

Detección de toque: el hardware compara lectura cruda vs. umbral reservado en cada ciclo y consulta el estado con “`touch_pad_get_status()`”. Tras procesarlo, para limpiar la bandera, “`touch_pad_clear_status()`”. Si el entorno es muy ruidoso, se recomienda filtrar y hacer la lógica de detección en software.

Interrupciones por toque: tras fijar el umbral, hay que habilitar el ISR con “`touch_pad_isr_register()`” para el registro de rutina y “`touch_pad_intr_enable()`” para habilitarla. En la ISR para saber qué pad disparó la interrupción leer “`touch_pad_get_status()`” y luego “`touch_pad_clear_status()`” para limpiar. Finalmente, para desactivarla “`touch_pad_intr_disable()`” y “`touch_pad_isr_deregister()`”.

En caso de ser necesario usar filtros, se pueden definir distintos parámetros con “`touch_pad_set_voltage()`” define rango de voltaje de carga/descarga; “`touch_pad_set_cnt_mode()`” regula la pendiente de carga; y “`touch_pad_set_meas_time()`” fija duración de cada medición.

b. Revise el esquemático de la placa base y el esquemático del TouchPad ¿Hay que modificar algo en la placa base para poder utilizar la placa de expansión? ¿Qué problemas (known issues) existen para esta placa de expansión?

En principio no habría que modificar nada mas que conectar el TouchPad a través del cable correspondiente, lo que si es importante es verificar que los no haya configuraciones de pines que interfieran con los GPIOs del TouchPad.

Respecto a los problemas conocidos, puede que las lecturas de los sensores táctiles saen algo inconsistentes, la humedad puede interferir con también con la precisión de lo que recibe el touch. Hay que asegurarse de elegir un umbral de detección correcto, de lo contrario podrían darse falosos positivos o no detectar las pulsaciones. Además puede haber problemas de compatibilidad usando otras placas simultáneamente, por ejemplo la pantalla LCD es sabido que causa inconvenientes y que se requiere ajustes en la configuración de los pines para evitarlos.

c. Diseñe un algoritmo (en pseudocódigo) indicando cómo utilizar el TouchPad y cómo detectar el estado de sus botones (presionado/no presionado).

```
Inicializar el sistema TouchPad con configuración global
Instalar controlador de botones táctiles

Para cada botón (de 0 a 13):
    Asignar canal capacitivo correspondiente
    Asignar sensibilidad (por ejemplo, 0.1)
    Crear el botón táctil con la configuración anterior
    Configurar tiempo de long-press (opcional, ej. 2000 ms)

Iniciar el sistema de elementos táctiles

Loop principal de detección:

Repetir periódicamente:
    Para cada botón (de 0 a 13):
        Leer el valor raw del canal capacitivo correspondiente
        Si el valor supera un umbral definido (ej. 30000):
            Registrar el evento (debug/log)
        Si el valor actual >= umbral y el valor anterior < umbral:
            Marcar el botón como "presionado"
            Registrar el evento (debug/log)
        Guardar el valor actual como valor anterior
```

d. Cree una librería nueva inicializando el TouchPad e implementando la detección de los botones y modifique el código anterior para que al presionar un botón cambie el estado del LED (color, brillo, on/off, etc)

Segunda Parte

En esta oportunidad se creará otro proyecto (Laboratorio_2b, se puede mantener todo en un mismo código pero recuerde mantener la prolijidad del mismo) y se implementará un Access Point (AP) y una estación (STA) WiFi y se debe poder conectar tanto al equipo como el equipo a una red ya existente.

1. Interfaz de Red y Loop de eventos

a. Para poder llevar a cabo la implementación tanto del punto de acceso Wifi como la estación WiFi es necesario comprender que es el ESP-NETIF y que es el Event Loop.

ESP-NETIF es la capa de abstracción de interfaz de red del ESP-IDF. Es responsable de gestionar las conexiones de red (como WiFi o Ethernet), proporcionando una forma unificada de manipular interfaces como Access Point (AP) o Station (STA). Permite configurar IP, manejar eventos relacionados con la red, y enlazarse con el stack TCP/IP (LWIP).
El **Event Loop** es el sistema de manejo de eventos asincrónicos del ESP-IDF. Funciona como un despachador que recibe eventos (por ejemplo, "una estación se conectó al AP" o "el STA obtuvo una IP") y los envía a funciones registradas (handlers) para su procesamiento.

b. Comente que hace la función 'esp_netif_init()' y si cree que es necesario para la implementación de redes WiFi.

La función esp_netif_init() inicializa el módulo ESP-NETIF. Es fundamental porque configura internamente las estructuras necesarias para gestionar interfaces de red. **Es esencial para implementar redes WiFi**, ya que, sin esta inicialización, no se podrían crear las interfaces esp_netif_t requeridas para STA o AP.

c. Comente que hace la función 'esp_event_loop_create_default()' y qué utilidad podría tener para la implementación de redes WiFi.

esp_event_loop_create_default() crea un *event loop* por defecto para manejar eventos del sistema. Es esencial para trabajar con WiFi, ya que muchos de los eventos importantes (como conexión, desconexión, obtención de IP, etc.) se notifican a través de este mecanismo. **Permite que nuestro programa escuche y reaccione ante los eventos del stack de red.**

2. implementación AP

a. Comente con qué opciones de configuración cuenta el API de WiFi del equipo basado en la guía de referencia.

La API permite configurar múltiples parámetros a través de la estructura wifi_config_t:

- ssid: nombre de la red.
- password: clave de acceso (puede estar vacía si es abierta).
- ssid_hidden: booleano para ocultar el SSID.
- authmode: método de autenticación (WPA2, WPA3, etc.).
- max_connection: número máximo de estaciones conectadas al AP.
- channel: canal de radiofrecuencia.
- beacon_interval: intervalo entre beacons.
- pmf_cfg: manejo de Management Frame Protection.

b. Implemente en pseudocódigo, y utilizando las funciones que aparecen en la referencia (se recomienda investigar bien los distintos ejemplos para llevar a cabo dicha implementación), como implementar un AP con el ESP32-S2.

```
funcion modo_ap():
    esp_netif_init()
    esp_event_loop_create_default()
    esp_netif_create_default_wifi_ap()
    esp_event_handler_instance_register(wifi_handler)
    esp_wifi_set_mode(WIFI_MODE_AP)
    wifi_config = {
        "ssid": "MiAP",
        "password": "12345678",
    }
    esp_wifi_set_config(wifi_config)
    esp_wifi_start()


funcion wifi_handler(evento):
    si evento == WIFI_EVENT_AP_START:
        print("AP iniciado")
    si evento == WIFI_EVENT_AP_STOP:
        print("AP detenido")
```

c. Lleve a cabo dicha implementación e intente conectar un dispositivo al ESP32-S2. ¿Qué datos del equipo que se conecta muestra el equipo? ¿Qué datos de red relevantes puede obtener en el monitor del idf?

Se puede obtener del equipo conectado:

- **MAC del cliente**: 76:02:cd:dc:94:71
- **Asignación de IP por DHCP**: 192.168.4.2
- **Asociación de cliente (AID)**: AID=1
- **Mensajes de join/leave del cliente**
- **Eventos HTTP (si se sirve una página web)**

Del lado del ESP32-S2, se puede ver:

- **Conexión/desconexión del cliente**
- **Dirección MAC de cada dispositivo**
- **Asignación de IP por el servidor DHCP interno**

d. Pruebe distintas configuraciones (WiFi abierto/con contraseña, SSID oculto, etc.) y comente su resultado en la implementación.
WiFi abierto: basta con establecer authmode = WIFI_AUTH_OPEN y dejar password vacío. El AP funciona sin autenticación.

SSID oculto: ssid_hidden = true. El SSID no se muestra al escanear, pero aún se puede conectar si se conoce el nombre.

Cambio de canal o beacon interval: puede mejorar la estabilidad o reducir interferencias.

3. implementación STA

   a. Recordando las funciones provistas en la guía de referencia, implemente, en pseudocódigo (se recomienda investigar bien los distintos ejemplos para llevar a cabo dicha implementación), una STA con el ESP32-S2 y conectarlo a alguna red WiFi que tenga disponible.

```
funcion modo_sta():
    esp_netif_init()
    esp_event_loop_create_default()
    esp_netif_create_default_wifi_sta()
    esp_event_handler_instance_register(wifi_handler)
    esp_wifi_set_mode(WIFI_MODE_STA)
    wifi_config = {
        ssid: "caliope",
        password: "sinlugar",
        authmode: WIFI_AUTH_WPA2_PSK,
        scan_method: WIFI_FAST_SCAN,
    }
    esp_wifi_set_config(wifi_config)
    esp_wifi_start()


funcion wifi_handler(evento):
    si evento == WIFI_EVENT_STA_START:
        print("Modo estación iniciado")
    si evento == WIFI_EVENT_STA_CONNECTED:
        print("Conectado a la red WiFi")
    si evento == WIFI_EVENT_STA_DISCONNECTED:
        print("Desconectado de la red WiFi")
```

   b. Lleve a cabo dicha implementación e intente conectar el equipo a una red. ¿Qué datos de red relevantes puede obtener en el monitor del idf? ¿Qué diferencias hay comparado con la implementación del AP?

   Log:
   I (1718) esp_netif_handlers: sta ip: 10.4.97.146, mask: 255.255.240.0, gw: 10.4.100.1
   I (1718) WIFI: STA: Got IP address:10.4.97.146

   Datos clave obtenidos:
   - Máscara de subred
   - Dirección IP asignada
   - Gateway
   - Confirmación de conexión exitosa

   Diferencias con el AP:
   - El STA obtiene IP de un servidor DHCP externo (el del router), no lo asigna.
   - Se conecta a una red existente, en lugar de crearla.
   - No hay control sobre los clientes (como en AP).

   c. Pruebe distintas configuraciones y comente su efecto en la implementación.

   - **Conexión a redes abiertas**: funciona dejando password vacío.

- **Conexión fallida (clave incorrecta)**: el evento de fallo se detecta en el loop.
- **Conexión a SSID oculto**: es posible, pero requiere escribir manualmente el SSID exacto.

Tercera Parte

En esta parte se hará énfasis en la creación de un servidor web en el ESP-KALUGA. En él almacenaremos una página web, a la cual accederemos para configurar el equipo. Para ello será necesario crear una página web sencilla, esta incluye un archivo de hipertexto (*.html), un archivo de "estilo" (*.css, opcional) y un archivo de ejecución de script javascript (*.js). Para comprender más el funcionamiento de estos archivos se recomienda la página W3School y los tutoriales y referencias que este brinda para crear páginas web.

a. Recordando las funciones provistas en la guía de referencia, implemente en pseudocódigo (se recomienda investigar bien los distintos ejemplos para llevar a cabo dicha implementación) un servidor web y que al acceder despliegue el mensaje de "Hola mundo" en el navegador.

```
Declarar variables para manejar archivos embebidos (HTML, JS, CSS)
Declarar buffer para datos POST

Definir función para manejar GET a "/":
      Enviar archivo HTML como respuesta

Definir función para manejar GET a "/script.js":
      Enviar archivo JS como respuesta

Definir función para manejar GET a "/style.css":
      Enviar archivo CSS como respuesta

Definir función para manejar POST a "/enviar":
      Leer datos del POST
      Si contiene "data_input":
            Mostrar en consola y devolver respuesta con el dato
      Si no:
`           Devolver mensaje de error

Función start_webserver():
      Si ya está iniciado, salir
      Configurar e iniciar el servidor
      Registrar los handlers:
            GET "/" → HTML
            GET "/script.js" → JS
            GET "/style.css" → CSS
            POST "/enviar" → datos del cliente
```

b. Usando las librerías ya implementadas genere una red WiFi (que el equipo funcione como AP) o conecte el mismo a una red ya presente (que el equipo funcione como STA).

c. Monte un servidor web basándose en los ejemplos brindados previamente. Se debe poder conectar/acceder a este desde una PC como de un dispositivo móvil (por ej. Celular).

d. ¿Cómo haría para enviar información al equipo? Implemente dicha solución.

Mediante peticiones HTTP, usando formularios HTML o peticiones de JavaScript. Algunas formas comunes son usar el método GET, el cual envía datos en la URL, o usar método POST, el cual envía datos en el cuerpo de la petición. Este último es más seguro para enviar configuraciones o datos grandes.

Nota: Puede serle de utilidad saber que puede incluir archivos en la compilación. Para ello debe agregar la siguiente línea de comando en el correspondiente archivo de CMake:

```
idf_component_register(SRCS "${srcs}"
                       INCLUDE_DIRS "${include_dirs}"
                       REQUIRES "${requires}"
                       PRIV_REQUIRES "${priv_requires}"
                       EMBED_FILES "<path-to>/index.html")
```

Y puede acceder a dicho archivo desde el ESP-IDF con las siguientes declaraciones:

```
extern const uint8_t index_html_start[] asm("_binary_index_html_start");
extern const uint8_t index_html_end[] asm("_binary_index_html_end");
```

e. ¿Qué datos de un sistema de IoT cree relevantes para persistir en el equipo?

Datos de red: Nombre y contraseña del WiFi, IP, etc.

Configuración del dispositivo: Valores de sensores, modos de funcionamiento, etc.

Identificación y seguridad: ID del equipo, claves o tokens de acceso.

Último estado: qué estaba haciendo el equipo antes de apagarse (por ejemplo, si una luz estaba encendida).

Errores o eventos importantes: por ejemplo, si falló una conexión.

Bibliografía

- *Technical Documents. Espressif:* *https://www.espressif.com/en/support/documents/technical-documents*