Laboratorio 1: Conociendo el Microcontrolador

ICC4200 - Sistemas Embebidos 202510

March 17, 2025

Fecha de entrega: 14 de abril de 2024 hasta las 6:00pm.
Modalidad: Laboratorio **individual**, sin embargo está permitido que conversen con sus compañeros sobre los ejercicios.

El equipo de este curso ha probado distintas alternativas para que el toolchain sea lo más estable posible, sin embargo siempre surgen problemas. Hemos probado las instrucciones con distintos OS. En caso de algún problema se recomienda seguir en **estricto** orden:

1. Realizar rubber-ducking- debugging (tenemos algunos patos para regalar en caso de ser necesario)
2. Revisar documentación + google + ChatGPT (o LLM de su preferencia)
3. Preguntar a un compañero
4. Preguntar a algún integrante del equipo del curso.

Al principio del curso cada alumno debe compartir con el profesor y los ayudantes un GitHub donde se deben incluir los proyectos del curso. Para cada laboratorio se debe crear una carpeta específica llamada Lab_X que debe contener cada uno de los ejercicios con la estructura: Ejercicio 1, Ejercicio 2, etc. Dentro de cada ejercicio debe haber un *README* con el comando necesario para correr el código asumiendo que todo está bien instalado. Preferente todos los ejercicios debería poder correrse con el comando:

```
source ~/.zprofile && source ~/.zshrc && get_esp32 && idf.py build
   && idf.py flash -p /dev/<PORT> monitor
```

Además del *README*, en cada Ejercicio debe haber un archivo *output.pdf* con los resultados que se piden. En cada ejercicio se define un **entregable** en que se define específicamente lo que se espera ver en el documento *output.pdf*. En resumen, cada repositorio debe verse de la siguiente forma:

```
Rep_SE_202501_Jorge_Gomez
|-Lab_1
|   |-Ejercicio_1
|   |    |-README.md
|   |    |-output.pdf
|   |    |-codigo del ejercicio
|   |-Ejercicio_2
```

```
|   |-Ejercicio_3
|-Lab_2
|-Lab_3
|-Proyecto
```

Notar que el último *PUSH* a cada laboratorio tiene que ser antes de la fecha de entrega, en caso de haber un *PUSH* posterior a la fecha de entrega, se realizará un *checkout* del último *commit* previo a la fecha de entrega.

Usted debe compartir su repositorio del curso con: *jtgomezmir89, caromanini, Vicente-Giaconi, FabianSaavedraE, DomingoFuenzalida*

1 Instalar toolchain de ESP-IDF

Primero debemos instalar ESP-IDF. Las instrucciones de instalación pueden variar dependiendo de su OS, se recomienda buscar tutoriales específicos para su *setup*. A continuación se entregan comandos que han sido probados en macOS y Windos utilizando WSL.

Preparar estructura de carpetas:

```
mkdir ~/esp && mkdir ~/esp/idf && mkdir ~/esp/idf-tools && mkdir ~/
   esp/backup && mkdir ~/esp/projects
```

Obtener ESP-IDF:

```
cd ~/esp/idf
git clone --recursive https://github.com/espressif/esp-idf.git
```

Por conveniencia, es bueno agregar un alias a .bashrc. (O .zshrc en macOS)

```
cd
vim ~/.bashrc
```

Agregar las siguientes líneas:

```
alias get_esp32='. $HOME/esp/idf/esp-idf/export.sh'
```

Lo mismo para .profile (O .zprofile en macOS):

```
vim ~/.profile
```

Agregar las siguientes líneas para definir variables de entorno:

```
export IDF_TOOLS_PATH="$HOME/esp/idf-tools"
export PATH="$HOME/esp/idf/esp-idf/tools:$PATH"
export IDF_PATH="$HOME/esp/idf/esp-idf"
```

Podemos hacer un source para aplicar los cambios (también sirve reiniciar el terminal de WSL).

```
source ~/.profile && source ~/.bashrc
```

Instalar ESP-IDF:

```
cd ~/esp/idf/esp-idf
./install.sh
. ./export.sh
```

Si estas usando WSL puedes seguir este tutorial para comunicarte con el MCU.

Se recomienda utilizar VScode para programar su microcontrolador con la extensión de C/C++.

Para probar la correcta instalación del ESP-IDF podemos modificar y correr un ejemplo simple.

```
cd ~/esp/projects
cp -r $IDF_PATH/examples/get-started/hello_world .
```

Necesitamos compilar (idf.py build) el código, *flashearlo* al MCU (idf.py flash) y monitorer el resultado de forma serial (idf.py -p PORT monitor). En VScode se recomienda definir un **Action button** con todas estas operaciones. En el json de configuración se puede definir:

```
    "actionButtons": {
        "commands": [
            {
                "name": " ESP Build and Run",
                "color": "white",
                "command": "source ~/.zprofile && source ~/.zshrc &&
                    get_esp32 && idf.py build && idf.py flash -p /
                   dev/cu.usbserial-1120 monitor"
            }

        ],
        "defaultColor": "white",
        "reloadButton": "",
        "loadNpmCommands": false
    }
```

Un muy buen tutorial en que se muestran varios de los puntos anteriores se puede ver aquí.

1.1 Ejercicio_1: (0.2 Puntos)

Modificar ejemplo *Hello World* para que imprima su nombre. **Entregable:** Screenshot de la salida serial obtenida con su nombre.

2 Midiendo desempeño

En esta sección vamos a crear el primer proyecto desde cero, configurar el microcontrolador y hacer un profiling de nuestro código.

Vaya a la carpeta en Github que compartió con el profesor y los ayudantes del cursov y cree un nuevo proyecto (Ejercicio_2).

```
source ~/.zprofile && source ~/.zshrc && get_esp32
idf.py create-project -p . <nombre>
```

En menuconfig puede definir el esp32 que está utilizando y modificar tamaño memoria y clock. De aquí en adelante **está permitido realizar cualquier modificación que quiera en menuconfig para obtener un mejor desempeño de cu MCU**.

```
idf.py menuconfig
```

2.1 Ejercicio_2: (1 Puntos)

Realice un código que implemente lo que aparece en el siguiente **pseudo-código**:

```
var_1 = 233
var_2 = 128
init_time()
init_instruction_count()
init_cycle_count()
for i=0, i=X, i++:
    result_0 = var1 + var_2
    result_1 = var1 + 10
    result_2 = var1%var_2
    result_3 = var1*var_2
    result_4 = var1/var_2
end_time()
end_instruction_count()
end_cycle_count()
check_results(PASS, FAIL)
calculate_Time()
calculate_CPI()
calculate_Cycles()
calculate_time_using_cycles()
```

1. Determine como evoluciona la medidad CPI cuando incrementa el valor de `X`, analice el resultado. **Entregable:** Gráfico + Análisis (0.33 puntos)

2. Realice un profiling del programa y determine cuál es la operación que utiliza más ciclos de reloj. Determine cuál es el bottleneck de su programa. **Entregable:** Gráfico de barras + Análisis (0.33 puntos)

3. Utilizando `menuconfig` modifique la frecuencia del MCU y analice el impacto en tiempo y ciclos de reloj. **Entregable:** Gráfico + Análisis (0.33 puntos)

Todos los gráficos y análisis deben estar en el documento *output.pdf*

3 DSP

Una de las ventajas de dsp-idf es que nos entrega varias funciones de dsp optimizadas en este repositorio.

3.1 Ejercicio_3: (1 Punto)

En este ejercicio usted debe preparar una imagen utilizando google colab. Específicamente debe tomar cualquier imagen, hacerla monocromática y escalarla a 96x96. En el esp32 usted debe aplicar un operador Sobel horizontal y vertical e imprimir a través del monitor serial el resultado. Finalmente utilizando google colab debe graficar la salida. **Entregable:** imagen de entrada, imagen de salida y link a colab utilizado.

4 Tipos de Memoria

4.1 Ejercicio_4: (1 Punto)

Para este ejercicio debe asegurarse que el esp32 que está utilizando tiene PSRAM (revisar documentación). Se le pide implementar un programa que realice lo que se presenta en este pseudo-código:

```
//Definimos memorias estaticas
DRAM_ATTR static int vector_dram[VECTOR_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8,
    9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
DRAM_ATTR static int num_dram = 5;
DRAM_ATTR static int result_dram[VECTOR_SIZE];
IRAM_ATTR static int vector_iram[VECTOR_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8,
    9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
IRAM_ATTR static int num_iram = 5;
IRAM_ATTR static int result_iram[VECTOR_SIZE];
RTC_DATA_ATTR static int vector_rtc[VECTOR_SIZE] = {1, 2, 3, 4, 5, 6, 7,
    8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
RTC_DATA_ATTR static int num_rtc = 5;
RTC_DATA_ATTR static int result_rtc[VECTOR_SIZE];
const __attribute__((section(".rodata"))) int vector_flash_ext[VECTOR_SIZE
    ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
     20};
const __attribute__((section(".rodata"))) int num_int = 5;
//Definimos memorias de forma dinamica
int *vector_dram = (int *)malloc(VECTOR_SIZE * sizeof(int));
int *result_dram = (int *)malloc(VECTOR_SIZE * sizeof(int));
int *num_dram = (int *)malloc(sizeof(int));
int *vector_iram = (int *)heap_caps_malloc(VECTOR_SIZE * sizeof(int),
    MALLOC_CAP_EXEC);
int *result_iram = (int *)heap_caps_malloc(VECTOR_SIZE * sizeof(int),
    MALLOC_CAP_EXEC);
int *num_iram = (int *)heap_caps_malloc(sizeof(int), MALLOC_CAP_EXEC);
int *vector_psram = (int *)heap_caps_malloc(VECTOR_SIZE * sizeof(int),
    MALLOC_CAP_SPIRAM);
```

```
int *result_psram = (int *)heap_caps_malloc(VECTOR_SIZE * sizeof(int),
    MALLOC_CAP_SPIRAM);
int *num_psram = (int *)heap_caps_malloc(sizeof(int), MALLOC_CAP_SPIRAM);

for (cada memoria):
    init_cycle_count ()
    result_mem = multiply_vector_scalar(vector_mem, num_mem)
    end_cycle_count ()
```

Determine que memoria tiene un mayor tiempo de acceso. **Entregable:** Gráfico de barras (memoria vs número de ciclos) + Análisis.

5 Competencia Final (2.8 Puntos)

En esta competencia usted deberá tomar una señal de audio y realizar un espectrograma. Considere la misma señal de audio limpia que se ocupa en el ejemplo con NFFT de 1024 y noverlap de 512. Usted deberá guardar la señal de audio en la memoria flash de su dispositivo (puede cambiar el formato .wav si prefiere otro), posteriormente usted deberá generar un espectograma en el esp32 y guardarlo en la memoria flash en un archivo .txt. Finalmente deberá generar un google colab que lea el archivo generado y lo compare con el espectograma generado por Python. Ningún pixel del espectograma puede tener un error mayor a 10% respecto al valor obtenido utilizando Python (Hint: pueden jugar con la resolución de la info para exprimir unos ciclos de reloj). Debe medir cuántos ciclos de reloj utiliza su programa. **Entregables:** Espectograma obtenido + Error de cada pixel graficado como mapa de calor + cantidad de ciclos de su programa.

Distribución de puntaje:

1. Espectrograma de salida con error menor a 10% (1.8 punto)
2. Competencia entre los alumnos de proyecto que utilice menor cantidad de ciclos. La competencia solo se realizará entre los proyectos que entreguen un resultado correcto (menor 0 y mayor 1 punto).