MASB
Proyecto MASBstat

#stm32 #c #biomedical #microcontroladores #programacion

Albert Álvarez Carulla
hello@thealbert.dev
https://thealbert.dev/

7 de mayo de 2020

Índice

1. Proyecto MASB-POT-S

1.1. Introducción

En este proyecto programaréis un potenciostato. El potenciostato está formado por un *front-end* específicamente diseñado para la asignatura y un *back-end* basado en la *Evaluation Board* (EVB) NUCLEO-F401RE de STMicroelectronics.

El objetivo del proyecto es poder realizar dos tipos de mediciones electroquímicas: voltammetría cíclica y cronoamperometría. El dispositivo será validado haciendo mediciones con una muestra de ferricianuro de potasio a diferentes concentraciones en un tampón/*buffer* de cloruro de potasio.

1.1.1. Voltammetría cíclica

Una Voltammetría Cíclica (CV) es un tipo de medición electroquímica potenciodinámica en la que se aplica un potencial variable a una celda electroquímica mientras se mide la corriente que esta celda proporciona. El potencial entre el electrodo de trabajo (WE) y el de referencia (RE) de la celda varía con el tiempo hasta que alcanza un valor potencial establecido, luego cambia de dirección, realizando lo que se denomina barrido triangular de potencial. Este proceso se repite durante un número establecido de ciclos. El resultado se representa en un voltamograma cíclico, que representa la corriente a través de la celda frente al voltaje aplicado en esta. La CV es una técnica ampliamente utilizada para estudiar las propiedades electroquímicas de un analito en una solución. Una CV proporciona gran cantidad de información sobre el comportamiento químico y físico de un sistema. Además, se pueden observar diferentes fenómenos físicos realizando voltammetrías a diferentes velocidades de exploración (modificando la velocidad de cambio de voltaje con el tiempo).

1.1.2. Cronoamperometría

Una Cronoamperometría (CA) es una técnica electroquímica que transduce la actividad de las especies biológicas de una celda electroquímica en una señal de corriente que cuantifica la concentración del analito de interés. En esta técnica se aplica una señal escalón y se mide la corriente a través de la celda en función del tiempo. Una de las ventajas de esta técnica es que no requiere etiquetado de analito o biorreceptor. El experimento comienza manteniendo la celda a un potencial en el que no ocurre ningún proceso faradaico. Entonces, el potencial se eleva a un valor en el cual ocurre una reacción redox.

1.2. Objetivos

- Programar un potenciostato portable.
- Controlar la *Power Management Unit* (PMU) del módulo *front-end* del potenciostato.
- Comunicarse con la aplicación viSens-S instalada con el *host* u ordenador mediante el protocolo MASB-COMM-S.
- Realizar una voltammetría cíclica.
- Realizar una cronoamperometría.

1.3. Requerimientos

El dispositivo debe de cumplir con una serie de requerimientos una vez programado. Esos requerimientos y sus códigos son:

1.3.1. Power Management Unit

- **[PMU.R001]** La PMU debe de habilitarse al incio y no volverse a deshabilitar.

1.3.2. Comunicación

- **[COMM.R001]** El microcontrolador debe de comunicarse con el *host* mediante una comunicación série asíncrona con configuración 115200 8N1.
- **[COMM.R002]** La comunicación debe de estar codificada en COBS utilizando como *term char* el carácter `0x00`.
- **[COMM.R003]** El microcontrolador actúa como esclavo del master (*host*). Debe de atender las instrucciones indicadas en el documento referido al juego de instrucciones de la comunicación MASB-COMM-S.
- **[COMM.R004]** Cuando el microcontrolador reciba la instrucción para iniciar una medición, el microcontrolador debe de iniciarla inmediatamente.
- **[COMM.R005] (Opcional)** Cuando el microcontrolador reciba la instrucción de `stop`, el microcontrolador debe de detener inmediatamente la medida que esté realizando en ese momento. El carácter opcional de este requerimiento hace que su no implementación no comporte penalización en la evaluación del proyecto. Por otro lado, implementarlo supondrá una valoración adicional de 1.5 puntos siempre y cuando el resto de requerimientos también se cumplan.

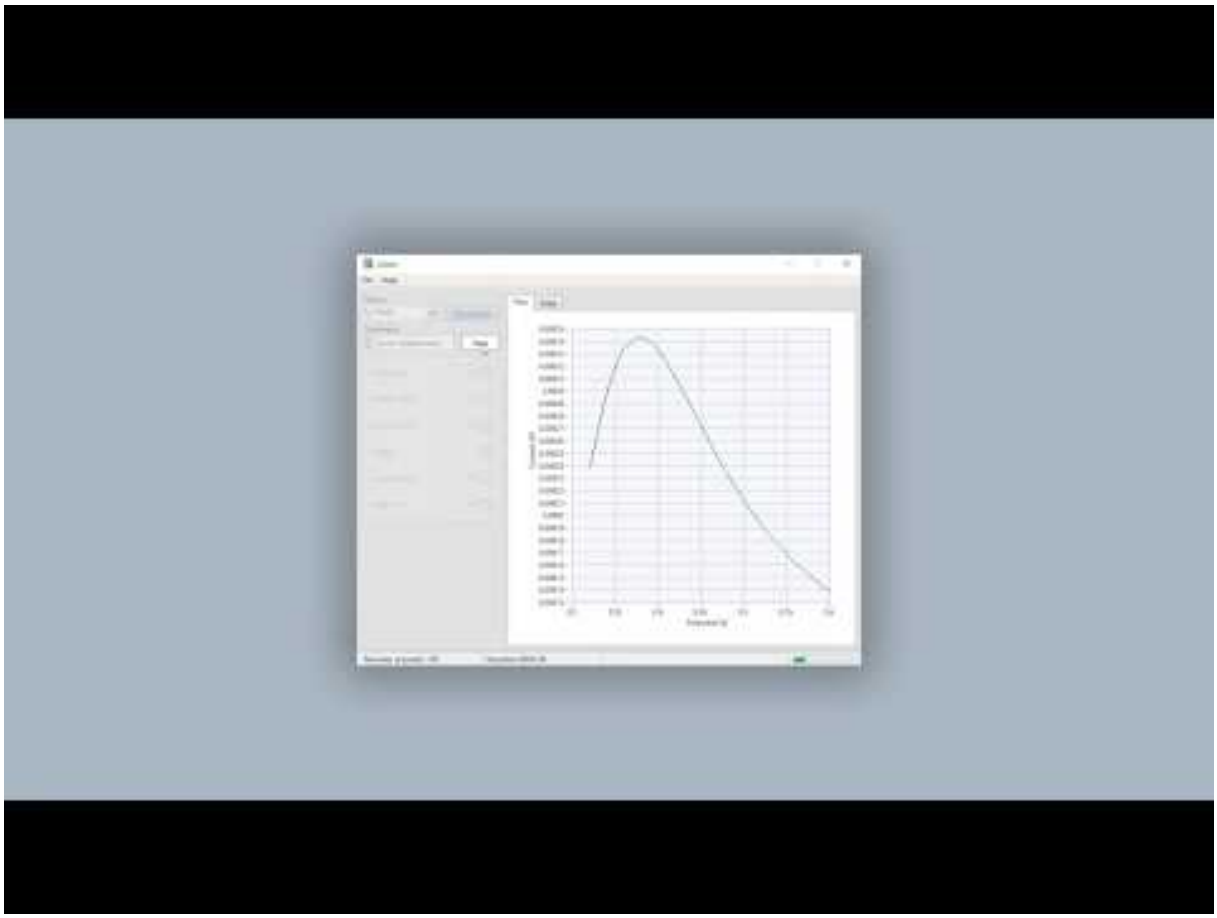
1.3.3. Otros

- **[OTR.R001]** Cuando el dispositivo no esté realizando ninguna medición, el relé que conecta el electrodo de *Counter Electrode* (CE) debe de quedar abierto.
- **[OTR.R002]** Se debe de emular el comportamiento de Arduino creando una función `setup` y `loop` con el objetivo de modificar lo menos posible el archivo `main.c`.

1.4. Aplicación de escritorio para el *host*: viSens-S

Podéis descargaros la aplicación que debéis de utilizar en vuestro ordenador en el siguiente enlace. Es el archivo `viSens.Installer.zip`.

Una vez instalada, la operación de la aplicación es sencilla. A continuación tenéis un video donde se muestra su funcionamiento.



> Se recomienda que en el repositorio de la aplicación, arriba del todo, activéis el seguimiento de las *releases* des de el icono Watch y cliquéis en la estrella. De este modo, si se corrige algún *bug*

> en la aplicación y se publica una nueva versión, os llegará una notificación.

> Las indicaciones del siguiente párrafo se dieron en el contexto del año pasado en el que estabamos confinados y no teniamos acceso al laboratorio. Igualmente, usadlo como indicaciones para ir haciendo pruebas hasta que podáis disponer del dispositivo.

Aunque no podamos hacer la prueba en el laboratorio, sí que podéis hacer pruebas utilizando el potenciometro que tenéis de prácticas. Haced un divisor de tensión con él y conectad el terminal variable a las entradas analógicas. Es importante también que comentéis el código relativo al DAC (I2C). Puesto que no está el DAC, el microcontrolador se quedaría congelado a la espera de recibir respuesta de él y el programa no funcionaría.

1.5. *Workflow* en Git

El flujo de trabajo o *workflow* utilizado será el que se ha utilizado en anteriores desarrollos. El resposi-torio debe de seguir el siguiente árbol de ramas:

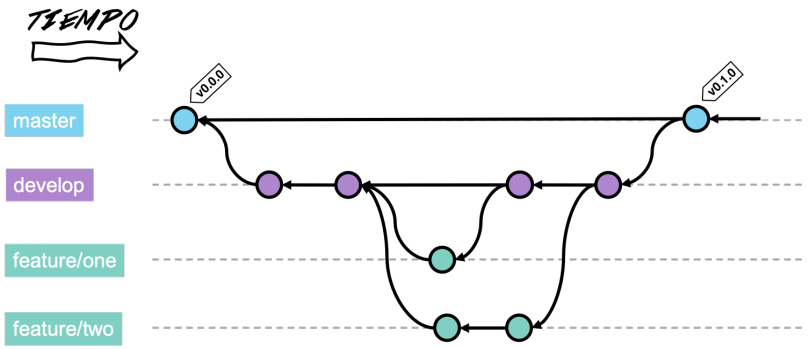


Figura 1: Git tree del repositorio.

- **master:** rama que contiene el código de producción. Se entiende como código de producción aquel que se le puede entregar a un cliente y funciona 100 %.
- **develop:** rama que contiene nuestro desarrollo. En esta rama se agrupan todos los desarrollos de los miembros del equipo y se testean. Una vez validado su correcto funcionamiento, los contenidos de la rama `develop` se vuelcan a la rama `master`, mediante un `Pull Request`, para entregar al cliente.
- **feature/***:** rama que contiene el desarrollo individual o colectivo de una funcionalidad. Los asteriscos deben de ser sustituidos por un término descriptivo de la funcionalidad desarrollada en esa rama. Los contenidos de esa rama se vuelcan en la rama `develop`, mediante un `Pull Request`, una vez han sido testeados.

- **hotfix/***:** rama que contiene la corrección de un *bug* o comportamiento no deseado. Los asteriscos deben de ser sustituidos por un término descriptivo del *bug* o comportamiento no deseado corregido. En función de la gravedad del *bug* o comportamiento no deseado a corregir, los contenidos de esa rama se vuelcan en la rama `master` (si es urgente su correción) o `develop` (si su corrección puede esperar), mediante un `Pull Request`, una vez han sido testeados.

Todos los *Pull Request* requieren poner como revisor a los otros miembros del equipo con tal de asegurar que todo el equipo está de acuerdo con los cambios propuestos.

Cada vez que queramos iniciar el desarrollo de una funcionalidad nueva, lo haremos creando una rama `feature/***` desde a rama `develop`. En esta rama `feature/***` **solo modificaremos archivos que hayamos creado nosotros y no generados automáticamente por STM32CubeMX**. **Esto último es muy importante**. Una vez finalizado el desarrollo y testeado, lo incorporamos a la rama `develop` con un *Pull Request*.

Si necesitamos modificar el archivo de configuración `.ioc`, lo haremos directamente en la rama `develop`. **En esta rama solo modificaremos el archivo `.ioc` y los archivos generados automáticamente por STM32CubeIDE, incluido `main.c`. Eso último es muy importante.**

Solo una persona puede hacer modificaciones/*commits* en la rama `develop` a la vez. Cuando se tenga que modificar el archivo `.ioc` o un archivo generado por STM32CubeIDE, **deberéis de sentaros juntos delante de un mismo ordenador (ya sea presencial o virtualmente) y editarlo conjuntamente** (que uno le diga al otro qué es lo que necesita que se configure).

Si habéis creado la rama `feature` antes de realizar la configuración o si habéis cambiado la configuración en medio del desarrollo de una funcionalidad para corregir un error, ese cambio de configuración no estará en vuestra rama `feature`. Tendréis que hacer un `git rebase`. A continuación tenéis el *snippet* para hacerlo:

```
1 git checkout develop
2 git pull
3 git checkout feature/***
4 git rebase develop
5 git push --force-with-lease
```

Una vez todas las funcionalidades deseadas se han incorporado a la rama `develop` y se ha comprobado que **todo funciona 100 %**, se hace un *Pull Request* a la rama `master`.

1.6. Pinout

A continuación se detallan los pines utilizados del microcontrolador para interaccionar con el *front-end* del potenciostato.

Pin	Alias	Tipo	Descripción
PA0	VREF	Entrada analógica	Tensión absoluta del *Reference Electrode (RE)* VREF utilizada para la medición de VCELL.
PA1	ICELL	Entrada analógica	Tensión de salida del TIA utilizada para la medición de ICELL.
PB8	SCK	I2C (SCK)	Señal SCK del I2C. Con el bus I2C se controla el DAC del *front-end*.
PB9	SDA	I2C (SDA)	Señal SDA del I2C. Con el bus I2C se controla el DAC del *front-end*.
PA5	EN	Salida digital	Señal de (des)habilitación de la PMU.`0`: PMU deshabilitada. `1`: PMU habilitada.
PB5	RELAY	Salida digital	Señal de control del relé.`0`: Relé abierto. `1`: Relé cerrado.

> De ahora en adelante, en el guión se hará referencia a los pines por su alias. Cabe diferenciar que cuando se utilicen subindices se estará haciendo referencia a la magnitud física en cuestión. Si está todo en mayúsculas, se estará haciendo referencia al pin. Por ejemplo, ICELL hace referencia al pin, ICELL a la corriente que pasa por la celda.

1.7. Modulos del *front-end* controlados por el microcontrolador

A continuación se describirán los diferentes módulos del *front-end* que controla el microcontrolador.

1.7.1. *Power Management Unit (PMU)*

La PMU, la unidad de gestión de potencia, es la responsable de alimentar todo el *front-end*. La PMU está deshabilitada por defecto para evitar quel el *front-end* consuma corriente hasta que la EVB negocie con el controlador USB de nuestro ordenador un consumo máximo. Cuando esto ocurra, el microcontrolador será alimentado y podemos habilitar la PMU. La PMU se habilita a través del pin `EN`. Nada más se inicie el programa en el microcontrolador, se debe de llevar a nivel alto el pin `EN`. Una vez habilitada la PMU, esperar 500 ms hasta proseguir con la ejecución del programa para asegurar que todos los componentes del dispositivo se encuentran correctamente alimentados.

1.7.2. Relé

El relé se encarga de abrir y cerrar el circuito entre el *front-end* y el sensor electroquímico. De este modo, cuando el relé está abierto, no hay conexión electrica entre el sensor y el *front-end*. Cuando el

relé está cerrado, el sensor se conecta al *front-end*. El estado por defecto del relé debe de ser abierto. Al realizarse una medición, el relé debe de cerrarse previamente. Una vez finalizada la medición, el relé debe de volver a abrirse. El relé se controla con el pin `RELAY`.

1.7.3. Potenciostato

El potenciostato es el responsable de polarizar la celda electroquímica a una tensión VCELL y leer la corriente que circula por ella ICELL.

Para establecer la tensión VCELL se dispone de un DAC modelo MCP4725 con dirección I2C `1100000`. El DAC puede generar una tensión de salida de 0 a 4 V. A la salida del DAC se ha añadido una etapa que convierte esa señal unipolar de 0 a 4 V a una señal bipolar de -4 a 4 V con tal de poder polarizar la celda tanto con tensiones negativas como positivas.

Pese a que controlamos la tensión de polarización de la celda, no la podemos dar por conocida. Por ello, utilizamos el ADC del microcontrolador para leer una tensión VADC que corresponde a la tensión del RE VREF previamente pasando por un circuito conversor de señal bipolar a unipolar. A su vez, a partir de esta tensión VREF, podemos obtener la tensión de la celda VCELL.

Finalmente, para leer la corriente a través de la celda se utiliza un amplificador de transimpedancia o TIA. La resistencia del TIA es de 50 kΩ. La señal también pasa por un conversor de señal bipolar a unipolar.

Se ha incorporado una librería `formulas` para realizar la conversión entre los valores leídos por el ADC a los niveles de tensión/corriente pertinente.

1.7.4. Potenciometro digital

La ganancia del TIA está implementada mediante un potenciometro digital. La librería ya está incluída en el proyecto. A continuación, puede verse el código de inicalización de la librería:

```
1     I2C_init(&hi2c1);
2
3     AD5280_Handle_T hpot = NULL;
4
5     hpot = AD5280_Init();
6
7     // Configuramos su direccion I2C de esclavo, su resistencia total (
          hay
8     // diferentes modelos; este tiene 50kohms) e indicamos que funcion
          queremos que
9     // se encargue de la escritura a traves del I2C. Utilizaremos la
          funcion
10    // I2C_Write de la libreria i2c_lib.
```

```
11      AD5280_ConfigSlaveAddress(hpot, 0x2C);
12      AD5280_ConfigNominalResistorValue(hpot, 50e3f);
13      AD5280_ConfigWriteFunction(hpot, I2C_write);
14
15      // Fijamos la resistencia de 50 kohms.
16      AD5280_SetWBResistance(hpot, 50e3f);
```

> Tanto el potenciometro como el DAC utilizan el I2C para comunicarse. Este periférico debe de configurarse en el fichero `.ioc`.

1.7.5. Convertidor digital-analógico

El proyecto también incorpora una librería para controlar el DAC. La inicialización del mismo ser realiza mediante el siguiente código:

```
1 // Declarar esta variable como global para poder
2 // acceder a ella desde diferentes archivos
3 static MCP4725_Handle_T hdac = NULL;
4
5 hdac = MCP4725_Init();
6 MCP4725_ConfigSlaveAddress(hdac, 0x66); // DIRECCION DEL ESCLAVO
7 MCP4725_ConfigVoltageReference(hdac, 4.0f); // TENSION DE REFERENCIA
8 MCP4725_ConfigWriteFunction(hdac, I2C_write); // FUNCION DE ESCRITURA (
     libreria I2C_lib)
```

Por otro lado, cada vez que queramos generar un nivel de tensión, lo hacemos con el código:

```
1 MCP4725_SetOutputVoltage(hdac, vDac); // NUEVA TENSION
```

1.8. Flujos de operación

A continuación se expondrán los flujos de ejecución a diferentes niveles de abstracción. Estos diagramas de flujo son de alto nivel, por lo que no indican todos los detalles u operaciones que tiene que hacer el microcontrolador. Son solo de referencia.

> Los diagramas de flujo que se muestren en los informes finales deberán de estar hechos con mermaid.

1.8.1. Aplicación

El siguiente flujo corresponde a la operativa del usuario con el dispositivo y la aplicación de escritorio instalada en el *host*.

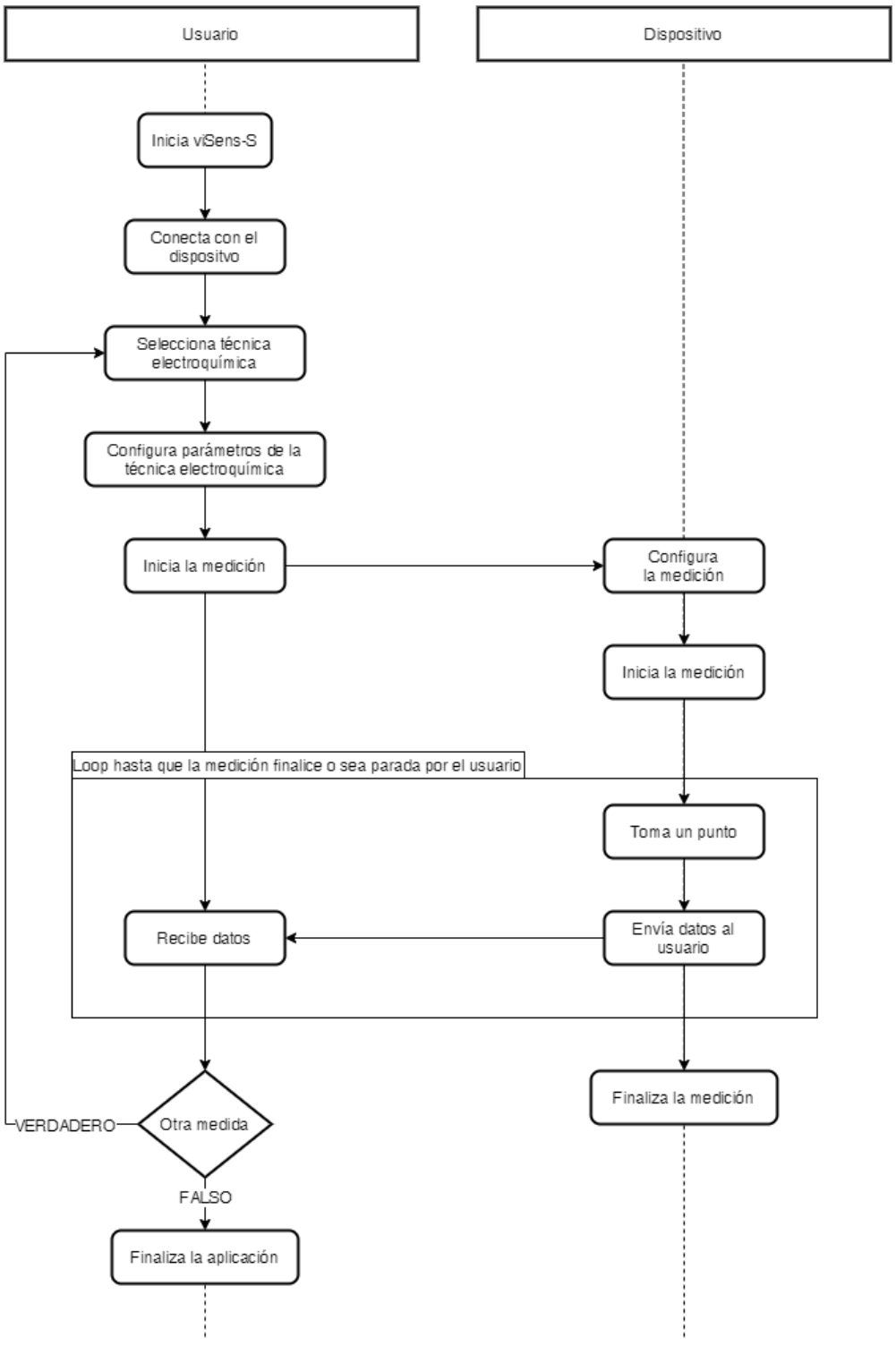


Figura 2: Diagrama de flujo de la aplicación.

1.8.2. Microcontrolador

El siguiente flujo corresponde a la operativa del microcontrolador en función de la instrucción recibida.

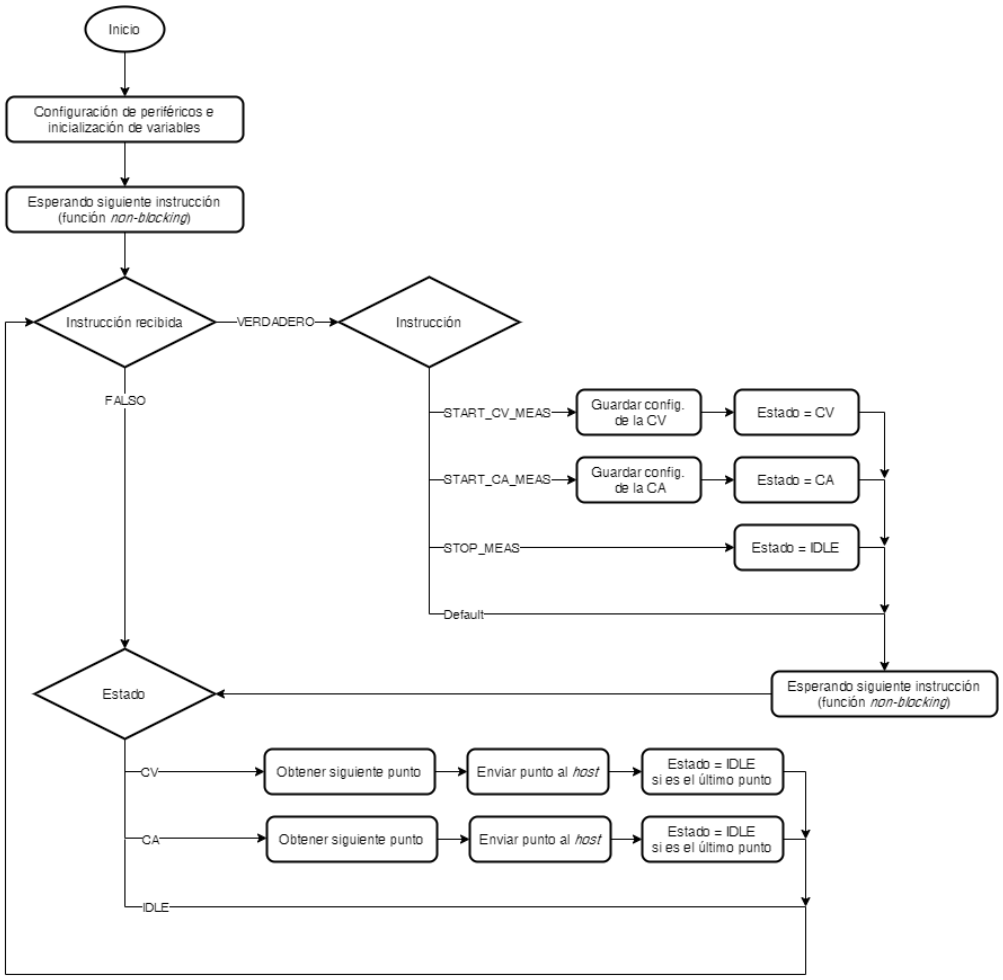


Figura 3: Diagrama de flujo del microcontrolador.

1.8.3. Voltammetría cíclica

El siguiente flujo corresponde a la operativa del microcontrolador a la hora de realizar una voltametría cíclica.

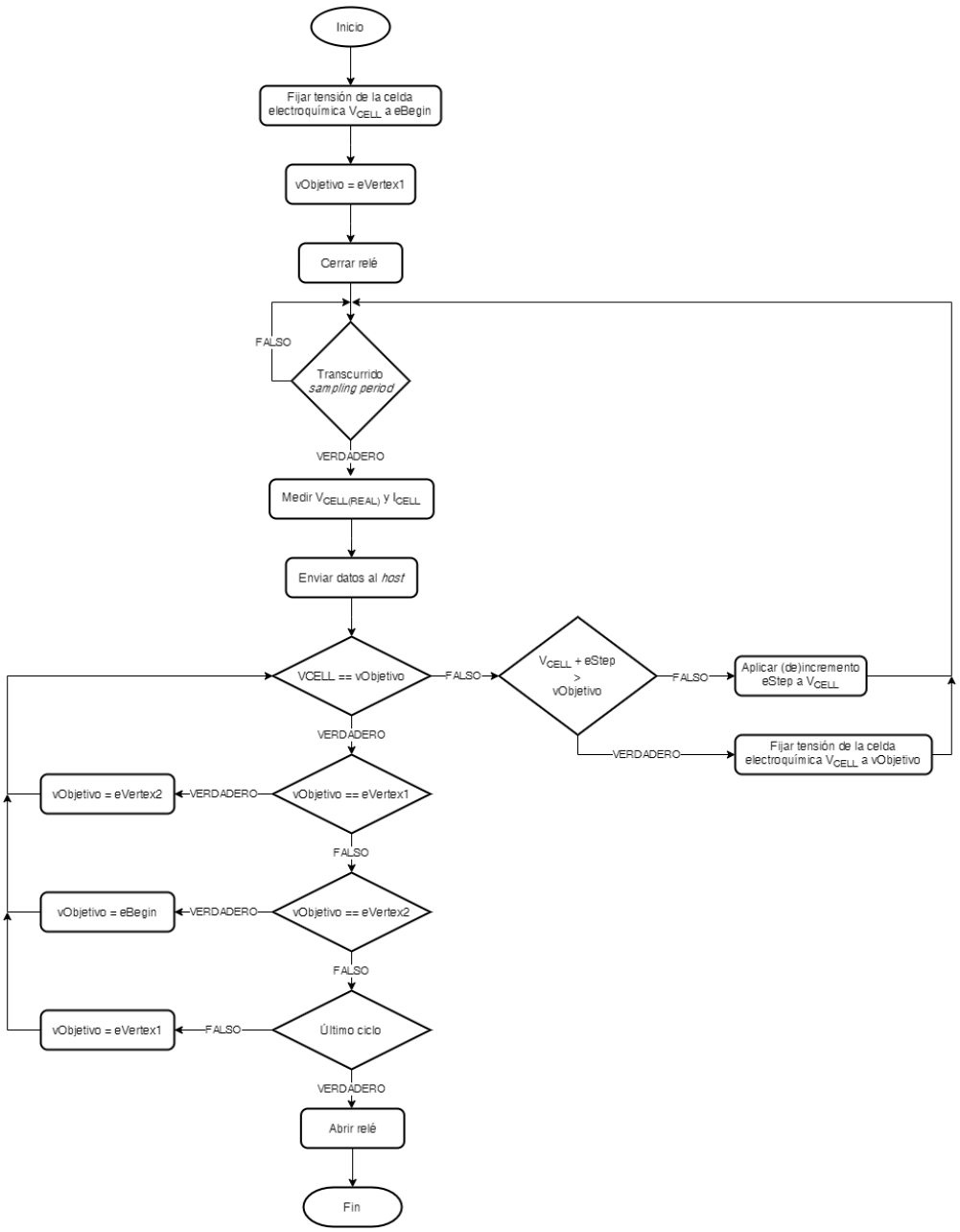


Figura 4: Diagrama de flujo de la voltammetría cíclica.

La voltammetría cíclica consiste en hacer un barrido en tensión sobre la celda electroquímica. Esto es, empezar aplicando una tensión `eBegin` a la celda e ir aplicando incrementos/decrementos de `eStep` hasta llegar a la tensión `eVertex1`. Una vez llegados a `eVertex`, realizamos el mismo proceso hasta llegar a `eVertex2`. Por último, una vez llegado a `eVertex2`, hacemos un barrido hasta `eBegin`. Esto corresponde a un ciclo. Hay que repetir el proceso tantas veces como nos lo indique `cycles`.

El periodo entre muestras nos los indican el *scan rate* y el *step*. El periodo entre muestras corresponde a:

$$t_s = \frac{eStep}{scanRate}$$

Figura 5: Fórmula del periodo entre muestras.

> En el diagrama aparece VCELL y VCELL(REAL). El primero hace referencia a la tensión de celda que fijamos con el DAC. El segundo hace referencia a la tensión de celda realmente aplicada y que obtenemos mediante el ADC.

IMPORTANTE: Para favorecer la viabilidad del proyecto, se considerará únicamente que `eVertex2 < eBegin < eVertex1`. En un potenciostato real, puede darse también el caso que `eVertex1 < eBegin < eVertex2`. Implementar el caso real, es decir, que el dispositivo funcione para ambos casos, supondrá una valoración adicional de 1 punto.

1.8.4. Cronoamperometría

El siguiente flujo corresponde a la operativa del microcontrolador a la hora de realizar una cronoamperometría.

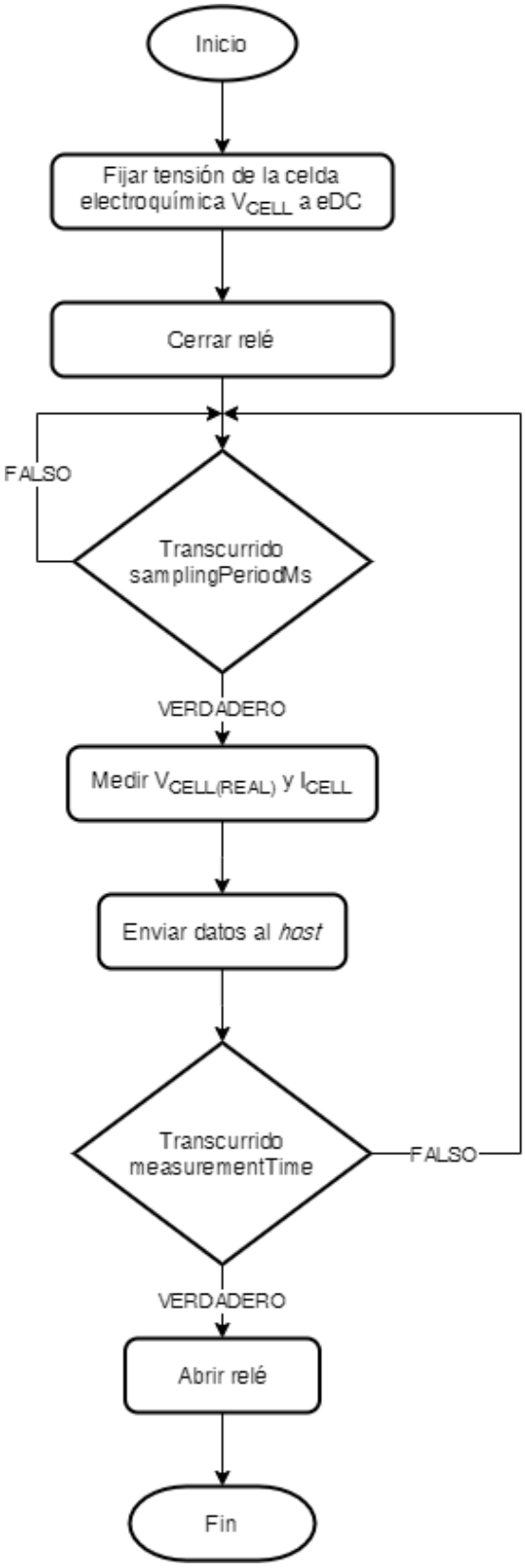


Figura 6: Diagrama de flujo de la cronoamperometría.

La cronoamperometría consiste en fijar una tensión constante en la celda electroquímica durante el tiempo indicado por `measurementTime`. Se toma una muestra con un periodo indicado por `samplingPeriodMs`.

1.9. Reparto de tareas

El reparto de tareas es libre. Podéis repartiros las tareas como creáis conveniente. Se utilizará el usuario indicado en los cambios en GitHub como indicador de quién ha hecho ese trabajo/cambio. Esto último se puede ver desde la pestaña `Insights` de vuestro repositorio en el apartado `Contributors`.

> **Importante.** Más línias de código/*commits* no indica necesariamente más trabajo realizado. Se puede ser muy eficiente y realizar el mismo trabajo con menos línias utilizando código optimizado. Esto se tendrá en cuenta.

1.9.1. Propuesta

Si los miembros del equipo de trabajo necesitasen una propuesta inicial de reparto de tareas, se puede disponer de la siguiente:

1.9.1.1. Miembro A

- Configuración del archivo `.ioc` y generación del código.
- Creación de las funciones `setup` y `loop`, y modificación de la función `main`.
- Gestión de la cronoamperometría.
- Gestión del DAC.

1.9.1.2. Miembro B

- Control de la PMU.
- Gestión de la voltammetría cíclica.
- Gestión de *timers*.
- Gestión de ADCs.

> Es una propuesta y se puede modificar a conveniencia siempre y cuando se realice un reparto equitativo de las cargas de trabajo.

1.10. Evaluación

Los entregables del proyecto son:

- Repositorio en GitHub con el proyecto desarrollado y que compile sin errores.
- Informe en Markdown con una descripción del proyecto y el trabajo realizado.

Ambos se evaluarán siguiendo la rúbrica que encontraréis en el Campus Virtual para tal efecto. Todos los entregables son imprescindibles para poder superar el proyecto.

> En el primer entregable se especifica que **el proyecto debe de compilar sin errores**. No se puede presentar un proyecto con errores de compilación. Puede tener funcionalidades que no estén bien hechas, código o configuraciones erróneos, código sin comentarios, etc., que luego se evaluarán en consecuencia. Pero no puede contener errores de compilación. **Un proyecto con errores de compilación no será evaluado y supondrá la no superación del proyecto**.Aunque se evaluará el correcto uso de Git, GitHub y el *workflow*, el código que se evaluará es el disponible en la rama `master`. Ese es el que no tiene que tener errores de compilación.

El informe debe de cumplir con los siguientes requerimientos:

- Se crearán dos informes: uno en castellano y otro en inglés. Ambos iguales y con el mismo contenido. Simplemente en dos idiomas.
- El lenguage debe de ser formal.
- Se crearán en la misma carpeta que el presente documento con el nombre `REPORT_ES.md` y `REPORT_EN.md`.
- El objetivo final del informe es ser colgado en la página web de la asignatura, por lo que debe de seguir el formato típico de una página web (texto, hypervínculos, imágenes, medios audiovisuales, etc.). (Se verá en clase cómo proceder para subir el informe en la página).
- Mientras que los puntos/apartados del informe son libres, el informe debe de contener:
    - Una introducción al proyecto (en qué consiste, qué es un potenciostato, cuáles son los objetivos, en qué consisten las medidas electroquímicas que hay que hacer y para qué sirven,…). Git/GitHub.
    - Un diagrama de flujo y una explicación de vuestra aplicación final (que puede ser parecido o no al de este guión).
    - Los resultados obtenidos.
    - Conclusiones extraídas tanto del proyecto como de la asignatura.

    Se hace hincapié en que los puntos/apartados del informe son libres (por ejemplo, puede haber la introducción al principio del informe sin necesidad de que haya un encabezado que diga "Introducción").
- En la carpeta `Docs/assets/imgs` debe de haber una fotografía reciente de cada miembro del equipo de trabajo nombrada siguiendo el siguiente formato `NOMBRE-APELLIDO1-APELLIDO2`

`.jpg`. La foto, en formato `.jpg`, ha de ser cuadrada de tipo retrato y con un tamaño mínimo de 400 x 400 pixels y un máximo de 8 MB. Tenéis que aparecer claramente identificables.

- Los recursos utilizados en el informe (imágenes, GIFs, etc.) deben de guardarse en la carpeta `Docs/assets`.
- De manera opcional, al principio del informe, indicad correos o perfiles de redes sociales (LinkedIn, Twitter, Instagram, etc.) donde queráis que os contacten/encuentren los lectores de vuestro informe. Comentad esa porción del informe y cuando se traslade el informe a la web se añadirán esos datos como datos de contacto en la página.

1.11. Contacto

El canal de comunicación principal será el repositorio masb con el objetivo de que podáis colaborar entre vosotros y que la información esté disponible para todos.

Si es necesario, y previa cita, se puede realizar una reunión presencial/virtual entre el gestor del proyecto y uno o más equipos de trabajo.