**Encendido del sistema**

1. Al encender el sistema, la RPi envía una solicitud para actualizar el tiempo del Master con la hora del GPS, red o RTC.
2. El Master activa la interrupción externa para incrementar la hora del reloj.
   1. El Master envía la hora a la RPi.
   2. El Master envía la hora a los nodos.
   3. El nodo activa una bandera para indicar que recibió la hora.
   4. El Master genera los pulsos en los nodos para que incrementen la hora.
3. La RPi envía una solicitud de inicio de muestreo al Master y este lo reenvía a todos los nodos.
4. Los nodos responden con la dirección del sector donde empezaron la escribir.
5. Los nodos empiezan el muestreo y almacenan los datos en la SD.

**Conexión de un nuevo Nodo**

1. El nodo inicia con la interrupción externa encendida.
2. Dentro de la interrupción revisa el estado de la bandera que indica si recibió la hora.
   1. Si la bandera es igual a 0, envía una petición de actualización de tiempo por RS485.
3. El Master recibe la petición a través de una interrupción por UART.
4. El Master envía la trama de tiempo al nodo por RS485.
5. El Master activa el pulso de interrupción en el nodo correspondiente.
6. El nodo empieza a incrementar su tiempo interno.
7. El Master envía una solicitud de inicio de muestreo al nodo.
8. El nodo empieza el muestreo y almacena los datos en la SD.

**Bajada de datos de los nodos**

1. La RPi envía una solicitud de lectura al Master especificando el id del nodo, el sector, la fecha, la hora y la duración del evento en segundos.
2. El Master reenvía la solicitud al nodo.
3. El nodo recibe la solicitud y la procesa.
4. El nodo responde la solicitud al Master la solicitud.
   1. Si la respuesta del nodo es que el dato no esta disponible, el Master reenvía la respuesta a la RPi y esta toma la decisión de que hacer (pedir de nuevo la información o pedir la información de otro nodo)
   2. Si la respuesta del nodo es que el dato esta disponible, el Master se prepara para recibir la información del primer sector y envía una respuesta a la RPi para que cree un archivo para almacenar los datos pedidos.
5. El Master recibe los sectores y los reenvía a la RPi uno por uno.
6. La RPi guarda todos los sectores y cierra el archivo.

**Funciones y parámetros**

1. **Muestreo:**

**Función:** 0xF1

**Subfunciones:** Iniciar, detener, leer (0xD1, 0xD2, 0xD3)

**Iniciar:**

Parámetros = Escribir, sobrescribir

Pyload = [0xD1, 0|1]

NumDatos = 2

**Detener:**

Parámetros = ninguno

Pyload = [0xD2]

NumDatos = 1

**Leer:**

Parámetros = sector, tiempo, duración

Pyload = [0xD3, SMSB, S2, S1, SLSB, aa, mm, dd, hh, mm, ss, duracionSegundos]

NumDatos = 12

1. **Tiempo:**

**Función:** 0xF2

**Subfunciones:** Establecer, leer (0xD1, 0xD2)

**Establecer:**

Parámetros = tiempo

Pyload = [0xD1, aa, mm, dd, hh, mm, ss]

Numdatos = 7

**Leer:**

Parámetros = ninguno

Pyload = [0xD2]

NumDatos = 1

1. **Comprobar comunicación:**

**Función:** 0xF3

**Ejemplos de solicitudes**

**Constantes:**

***Cabecera*** = [Inicio, Dirección, Función, #datos]

***Fin de trama*** = [0x0D, 0x0A]

**Sincronización de nodos (Función F1, subfunción D1):**

***Master envía:***

Cabecera= [0x3A, 255, 0xF1, 7]

Pyload = [Subfunción, aa, mm, dd, hh, mm, ss]

Pyload = [0xD1, 0x20, 0x6, 0x18, 0x15, 0x0, 0x0]

***Nodo procesa:***

tramaCabeceraRS485 = [0x3A, 255, 0xF2, 7]

tramaPyloadRS485 = [0xD1, 0x20, 0x6, 0x18, 0x15, 0x0, 0x0, 0x0D, 0x0A]

función = tramaCabeceraRS485[2]

subfunción = tramaPyloadRS485[0]

tiempo = tramaPyloadRS485[1:8]

***Nodo responde:***

La hora del master.

**Lectura de reloj del nodo (Función F1, subfunción D2):**

***Master envía:***

Cabecera= [0x3A, 5, 0xF1, 1]

Pyload = [Subfunción]

Pyload = [0xD2]

***Nodo procesa:***

tramaCabeceraRS485 = [0x3A, 5, 0xF1, 1]

tramaPyloadRS485 = [0xD2]

función = tramaCabeceraRS485[2]

subfunción = tramaPyloadRS485[0]

***Nodo responde:***

Cabecera= tramaCabeceraRS485 = [0x3A, 5, 0xF2, 6] => modifico la dirección y el número de datos

Pyload = tramaTiempo

**Iniciar muestreo (Función F2, Subfunción D1):**

***Master envía:***

Cabecera= [0x3A, 255, 0xF2, 2]

Pyload = [Subfunción, sobrescribir1/0]

Pyload = [0xD1, 0] => sobrescribir = no

***Nodo procesa:***

tramaCabeceraRS485 = [0x3A, 255, 0xF2, 2]

tramaPyloadRS485 = [0xD1 0]

función = tramaCabeceraRS485[2]

subfunción = tramaPyloadRS485[0]

***Nodo responde:***

\*\*Nada por ahora

**Detener muestreo (Función F2, Subfunción D2):**

***Master envía:***

Cabecera= [0x3A, 255, 0xF2, 1]

Pyload = [Subfunción]

Pyload = [0xD2]

***Nodo procesa:***

tramaCabeceraRS485 = [0x3A, 255, 0xF2, 1]

tramaPyloadRS485 = [0xD2]

función = tramaCabeceraRS485[2]

subfunción = tramaPyloadRS485[0]

***Nodo responde:***

\*\*Nada por ahora

**Inspeccionar sector (Función F3, Subfunción D2):**

El Master envía como parámetro la dirección del nodo y el sector que desea analizar. El nodo responde con la hora y fecha almacenada en ese sector, o mejor dicho el contenido de la ubicación donde se supone que esta almacena esa información.

***Master envía:***

Cabecera = [0x3A, 5, 0xF3, 5]

Pyload = [Subfunción, SMSB, S2, S1, SLSB]

Lectura de 100 segundos del sector 1000:

Sector = 1000d = 0x03EB

Pyload = [0xD3, 0x0, 0x0, 0x03, 0xEB]

***Nodo procesa:***

tramaCabeceraRS485 = [0x3A, 5, 0xF1, 1]

tramaPyloadRS485 = [0xD3, 0x0, 0x0, 0x03, 0xEB]

***Nodo responde:***

Cabecera= tramaCabeceraRS485 => modifico la dirección y el número de datos

Pyload = datos tiempo sector

**Programas en C:**

**Importante:**

B1 => Tiempo

B2 => Muestreo

B3 => Lectura de sectores

1. **SincronizarTiempoSistema(r):** Iguala el reloj del master y TODOS los nodos.

***Parámetros de entrada:*** Fuente de reloj

* r: Hora de red
* g: Hora de GPS

***Retorna:*** La hora del Master

***Función:*** F1, D1

1. **LeerTiempoNodo(255):** Lee la hora y la fecha de un nodo especifico.

***Parámetros de entrada:*** Dirección del nodo. En caso de poner la dirección de broadcast hace un barrido de las direcciones de los nodos teniendo en cuenta un slot de tiempo para evitar interferencias.

***Retorna:*** La dirección del nodo con su fecha y hora correspondiente.

***Función:*** F1, D2

1. **IniciarMuestreo(255, 1):** Inicia el muestreo en los nodos.

***Parámetros de entrada:*** Dirección del nodo. Permiso para sobrescribir la SD del nodo. En caso de poner la dirección de broadcast hace un barrido de las direcciones de los nodos teniendo en cuenta un slot de tiempo para evitar interferencias.

* 0: No sobrescribe la SD
* 1: Sobrescribe la SD

***Retorna:*** La dirección del nodo, el sector donde empezó la escritura y la hora/fecha del nodo.

***Función:*** F2, D1

1. **DetenerMuestreo(255):** Detiene el muestreo de los nodos.

***Parámetros de entrada*:** Dirección del nodo. Admite dirección broadcast.

***Retorna:*** Nada. Mensaje de confirmación.

***Función:*** F2, D2

**\*\* Muestrear(255, 1, 1):** Inicia el muestreo en los nodos.

***Parámetros de entrada:*** Dirección del nodo. Muestreo (1=iniciar, 0=detener). Sobreescritura de la SD (0=No, 1=Si)

***Retorna:*** Nada

***Función:*** F2, D1

1. **InformacionSectores(255):** Recupera información acerca de los sectores de la SD.

***Parámetros de entrada:*** Dirección del nodo.

***Retorna:*** Dirección del nodo. Primer sector físico. Primer sector escrito en la última petición de muestreo. Ultimo sector escrito.

***Función:*** F3, D1

1. **InspeccionarSector(255, Sector):** Recupera la hora y fecha del sector seleccionado.

***Parámetros de entrada*:** Numero del sector que se desea inspeccionar. Dirección del nodo.

***Retorna:*** Numero del sector en el nodo, hora/fecha. Si se solicita la información de un sector vacío o con información incongruente devuelve una advertencia.

***Función:*** F3, D2

1. **LeerAceleracion(id, Sector, periodo):** Recupera los datos de aceleración de un periodo de tiempo a partir del sector especificado.

***Parámetros de entrada*:** Dirección del nodo. Sector a partir del cual se va a leer los datos de aceleración. Periodo de tiempo de lectura en segundos.

Retorna: Archivo binario con los datos de aceleración requeridos. Formato del nombre: dirección\_aammddhhmmss\_duracion.dat

***Función:*** F3, D3

**Observaciones generales.**

Lo ideal es reducir la máxima cantidad posible de trabajo a los nodos. Por ejemplo, en lugar de hacer que sean los nodos quienes busquen los datos de aceleración almacenados en la SD, se hará que sea la RPi quien le indique el sector que debe leer; de esta manera el nodo solo se limitara a comprobar primero si el dato es el correcto para luego responder con la información solicitada o el dato del tiempo del sector si no coincide con la información solicitada. Así será la RPi quien tenga que hacer un nuevo cálculo para antes de volver a solicitar la información al nodo.

La prioridad de las tareas esta en este orden:

1. Sincronizar los nodos y garantizar que se están almacenando los datos. Esto debo presentar mañana. La experiencia de lo sucedido con las estaciones de Chanlud, Labrado y la Centro Sur, indican que la memoria de 16Gb puede almacenar hasta 3 meses de datos.
2. Sincronización con el GPS.
3. Bajada de datos por sector sin revisión.
4. Bajada de datos por sector con revisión.

Los programas en red son más fáciles de implementar en Python y funcionan mejor. C es más conveniente para implementar programas que requieren velocidad y precisión. Lo más conveniente es combinar ambos lenguajes para utilizar lo mejor de cada uno.

El sistema funciona distinto al sistema de registro continuo en el sentido de que no es necesario que el sistema de control en la Raspberry Pi funcione de manera continua si no solo en los momentos de que se requiera hacer una acción especifica sobre los nodos como por ejemplo igualar el reloj, probar la comunicación o bajar los datos.

**Optimizar el código**

El primer intento de optimizar el código no funcionó. No sé por qué. Todo lo que hice lo envié a una rama nueva llamada V2 para no descartar nada. Creo que el problema fue que traté de modificar todo el sistema a la vez en lugar de trabajar por partes como hago siempre. Ahora para que funcione voy a desarrollar el ultimo método utilizando el formato que quiero implantar en todo el sistema y cuando funcione voy a implantarlo en el resto poco a poco.

Voy a mandar la subfunción como dato de la trama pyload y no como parámetro. Ese fue uno de los problemas. Primero porque cambiar eso significa hacer esa modificación en todos los dispositivos. Segundo porque parece que había problema en enviar el pyload vacío para ciertas funciones, y se tenía que enviar el pyload con un dato cero para solucionarlo lo que es un poco ineficiente.

Aunque por otro lado es más eficiente agregar el parámetro subfunción a la cabecera para organizar bien los datos.

**Lectura de datos de aceleración**

Hasta ahora los programas que he hecho han sido para que el master trasmita los mensajes de forma transparente entre la RPi y lo nodos. No sé si en el caso de la lectura de datos de aceleración tengo que hacer lo mismo. Tengo dos opciones:

1. El Master recupera la lectura de los sectores. Agrupa la información en un solo vector de 2500 muestras y lo envía a la RPi.
2. El Master traspasa los mensajes de forma directa. La Rpi los recibe y se encarga de ordenar los datos en un solo vector para luego guardarlo.

Creo que la mejor opción es la segunda porque siempre es mejor reducir la carga de trabajo en los microcontroladores.

Lo que si estoy seguro es de utilizar el mismo formato de datos para poder usar todos los programas que ya tengo hecho para el sistema de registro continuo.

Hay más opciones:

1. La RPi hace una petición por cada sector. Cada trama de respuesta tiene 513 bytes: 1 byte de subfunción, 512 bytes de datos. Recibe el primer sector e imprime la fecha. Recibe los 4 sectores restantes, los une una sola trama de datos y los guarda en el archivo binario. Ventajas: Más control del flujo de datos, compatible con lo que he venido haciendo hasta ahora. Desventajas: Mucho tráfico de datos entre los dispositivos, más demorado.
2. El nodo recibe el número del sector que se requiere leer. Lee 5 sectores a partir de esa dirección. Los almacena en una sola trama de 2506 bytes y lo envía a la RPi. La Rpi recibe la trama, imprime la fecha y vuelve a hacer otra petición al nodo apuntando al sector 5 veces mayor. La RPi envía una petición por cada segundo que requiere. La longitud de la trama de respuesta es de 25007 bytes: 1 byte de subfunción, 2500 bytes de datos y 6 bytes de tiempo.

Después de analizar con el osciloscopio se descubrió que cada lectura de un sector toma alrededor de 7.7ms, 5ms para procesar la petición y 2.7ms para transmitir 512 bytes desde el nodo hasta el master. Con esta información se tiene que transmitir 1seg de datos utilizando las opciones 1-3 tomaría un mínimo de 38.25 ms y 100seg que sería lo máximo esperado tomaría un mínimo de 3.82seg, y digo mínimo por que no estoy considerando el tiempo que se demora en enviar la trama desde el Master hasta la RPi, por lo que el tiempo total puede estar alrededor de 7.65seg por nodo.

Voy a proceder a implementar el método utilizando la opción 4 utilizando los siguientes pasos:

1. La RPi inspecciona el sector deseado. Si recupera un dato incongruente registra los siguientes 5 sectores hasta encontrar un dato de tiempo valido. Si recupera un tiempo errónea emite una alerta. Si recupera un tiempo aproximado (fijada por el usuario) devuelve el numero del sector.
2. La RPi utiliza este dato para enviar una solicitud de lectura de aceleración de 1 segundo. La RPi recupera la trama de datos de 2500 bytes (2000 bytes de aceleración + 6 bytes de tiempo). La RPi guarda el vector recuperado en un archivo de texto cuyo nombre es el dato del tiempo de este vector.
3. La RPi inspecciona el siguiente sector (5 posiciones más a partir del sector anterior). Si recupera un tiempo valido continua con el proceso anterior hasta terminar de recuperar el total de segundos requeridos. Si es un tiempo invalido cierra el archivo y emite una alerta. Un tiempo valido además de ser legible es la continuación del tiempo anterior.

\*\* Esto es muy complejo y necesito tiempo para implementarlo bien. Para tener una versión instalable es suficiente con recuperar una trama de un segundo y guardarla.

Para guardar la trama definí el siguiente formato de nombre: CxNx\_FechaHora\_Duracion

* Cx: Código del concentrador
* Nx: Código del nodo
* FechaHora: Fecha y hora en formato aammddhhmmss
* Duración: Duración del evento recuperado en formato de 3 digitos
* Ejemplo: C1N4\_200730113230\_060

Los parámetros de entrada de la función son: Dirección del nodo, sector requerido y duración.

La fecha es recuperada de la primera trama. La función no hace ninguna validación puesto que de eso se encarga la función InspeccionarSector.c, o alguna otra que se implemente para inspeccionar el nodo antes de recuperar los datos.

La idea es que estas funciones sean invocadas desde un software principal implementado en Python. Este software se encargaría de gestionar las comunicaciones con todos los nodos. Al momento de ocurrir un evento o solicitar la información de un tiempo determinado, lo primero que se hace es inspeccionar que sector tiene la información solicitada para luego proceder a recuperarla.