INFORME PRACTICA DE LABORATORIO 4

Santiago Nomesque – Jair Munevar

Universidad Sergio Arboleda

¿QUE USAMOS?

- Microcontrolador STM32F411CEU.
- ST LINK V2.
- Programa STMCUBE IDE.
- Programa PYTHON.
- Cable USB-C

PROTOCOLO PARA ENVIAR RPM Y TIEMPO

Para este laboratorio se implementó un protocolo de comunicación básico creado por nosotros, donde se utilizó un array de siete posiciones, donde la primera indica el inicio del envió de datos con un 0x0F, luego en la segunda se establece el tamaño del array según la información enviada, de siete bytes, para los datos que deseamos enviar se usan las posiciones tres, cuatro y cinco, de modo que en la tres y cuatro se envían las RPM y en la cinco el valor del TIEMPO. La posición seis se usa para el CHK, el cual establecimos una operación XOR entre los datos de la posición tres, cuatro y cinco y el numero decimal 15. Para finalizar nuestra comunicación en la posición siete se envía el valor 0xFF que funciona como indicativo de que el array de información llego a su fin.

USO DEL USB PARA ENVIAR DATOS

Para poder usar el puerto USB de nuestra STM, primero se configuro en el apartado de “Connectivity”, seleccionando el “USB_OTG_FS” y estableciendo su modo en “Device_Only”. Posteriormente vamos al apartador “Middleware and Software Packs” y seleccionamos la opción “USB_DEVICE” y en la parte de “Class For FS IP” seleccionamos la opción “Communication Device Class (Virtual Port Com)” y de esta manera ya queda en funcionamiento nuestro puerto USB de la tarjeta STM. Ahora para poder hacer uso de él, utilizamos la función **CDC_Transmit_FS,** en esta función, utilizamos como primer argumento el array que usaremos para enviar la información y el segundo parámetro es el indicador de cuanta información se enviara.

¿QUE SE REALIZÓ?

Se creo un protocolo para enviar información desde la STM a nuestro PC por medio de USB, y se usó la función CDC_Transmit_FS para enviar la información. Para los datos que se agregaban al arreglo de datos, se dividió el valor de RPM en dos partes ya que esta variable es de tipo UINT16 es decir de dos BYTES, logrando tener el valor de RPM en dos BYTES aparte pero que al unirlos en el orden correcto representa el valor de RPM real. La variable Tiempo también se envió en el array, sin embargo, esta variable al ser UINT8 no se separó, únicamente se envió tal cual se leyó en el programa. Para la gráfica, se creó un programa en Python que nos permitió conectarnos al puerto COM de nuestra PC y leer el array de datos, se almacena en otro arreglo en Python y luego aplicamos el protocolo utilizado, pero a la inversa, se verificaba que el valor de CHK fuera igual al realizar la operación, también se unificaron los bytes de RPM y se identificó el inicio y final del protocolo. Luego de convertir los datos estos se usaron para realizar una gráfica de RPM vs TIEMPO en Python.

¿QUÉ SUCEDIÓ?

Primero en Python, no reconocía el puerto COM, esto se debía a que no se había establecido el correcto, luego se estableció el correcto, pero no lo cerramos en el HERCULES, entonces no se lograba conectar en Python, corregimos esto y ya se leían datos, pero en forma extraña, esto se debía a que teníamos la tasa de BAUDRATE en menos de115200. Ya se logró mostrar bien la gráfica, pero al cabo de un tiempo los datos más antiguos se borraban, esto se debía a que únicamente teníamos una lista que capturaba 50 datos, esto se arregló poniendo una lista sin un tamaña determinado para almacenar los datos. Otro problema que tuvimos fue que cuando se había graficado las RPM del motor girando hacia un lado y luego queríamos graficar estas, pero cuando giraba hacia el lado contrario, la gráfica no se borraba y sobrescribía la que ya existía, para ello se puso un condicional que cada vez que el tiempo fuese 1, la gráfica se reiniciara y arrancara de nuevo. Con esta solución se nos presentó el problema de que ahora generaba muchas gráficas sobre sí mismas y muchas etiquetas, teníamos infinitas gráficas del mismo par de datos, entonces, cuando se entrara al condicional, solo se borraran los datos de las listas que almacenaban los datos y así no se borraba la interfaz de la gráfica.

DIAGRAMAS DE FLUJO

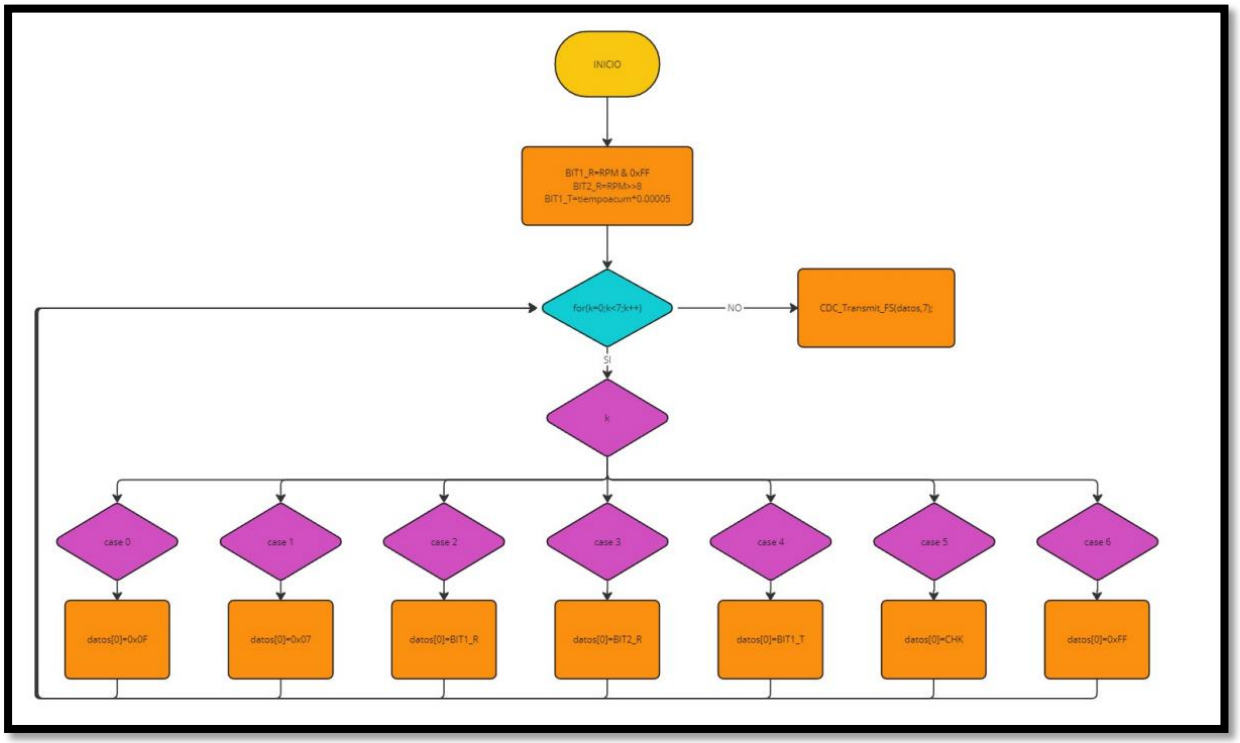

Imagen 1: Diagrama de flujo del código de la STM32 para él envió de datos

En el anexo 1 se presenta el diagrama de flujo para el programa de Python

CONCLUSIONES

- El laboratorio demostró la viabilidad de integrar un sistema embebido con un entorno de software en la PC para la visualización en tiempo real, lo que abre posibilidades para el monitoreo de sistemas de control en diversas aplicaciones industriales y educativas.
- La solución para limpiar y reiniciar correctamente la gráfica al cambiar la dirección del motor, evitando múltiples sobreescrituras, resalta la importancia de gestionar correctamente la interfaz gráfica en tiempo real para asegurar una representación clara de los datos.
- Durante el proceso se identificaron y resolvieron diversos errores como la incorrecta configuración del puerto COM y la tasa de BAUDRATE. Esto subraya la importancia de realizar una depuración exhaustiva en sistemas de comunicación serial.
- El uso de listas dinámicas en Python permitió almacenar un número ilimitado de muestras, lo que fue importante para visualizar la gráfica completa de RPM vs. TIEMPO, mejorando el muestreo de datos a largo plazo.

- El uso de un protocolo propio permitió el envío eficiente de datos entre la STM32F411 y el programa en Python, demostrando la practicidad de implementar y adaptar protocolos de comunicación según las necesidades del sistema.

ANEXOS:

Anexo 1.

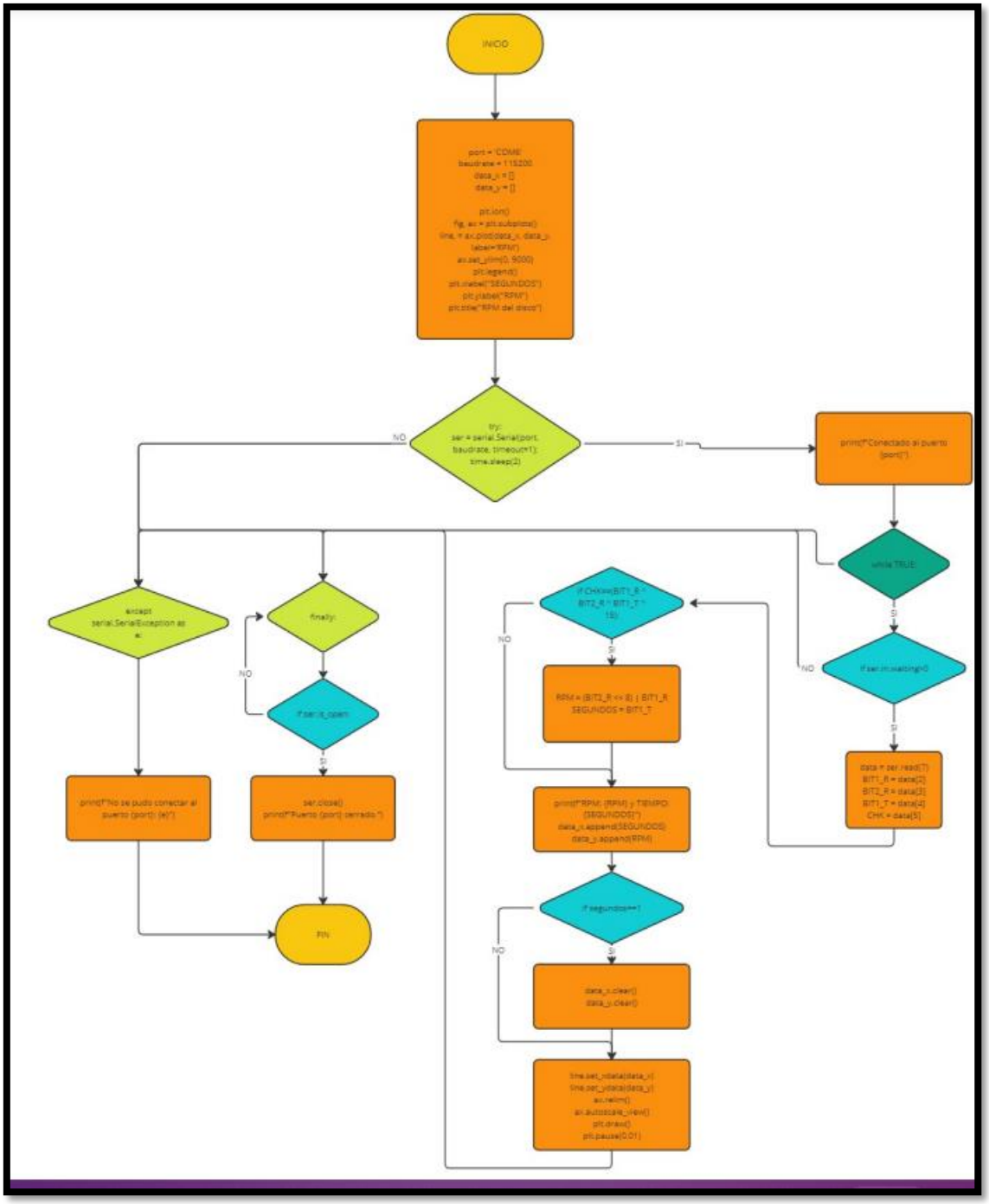

Imagen 2: Diagrama de flujo del código en Python