Informe practica de laboratorio 4

Universidad Sergio Arboleda
Sistemas Embebidos

Estudiantes:

- Sharay Valentina Galvis Prieto.
- Juan Diego Moreno Cardenas.

Introducción:

El presente informe describe el desarrollo e implementación de un sistema embebido utilizando el microcontrolador STM32F411. El objetivo principal del laboratorio fue diseñar un sistema capaz de recibir y decodificar comandos infrarrojos (IR), establecer una comunicación USB mediante el puerto COM, tener un conteo en tiempo real de las vueltas que realizaba la rueda y medir las revoluciones por minuto (RPM) de un motor DC.

Para cumplir con estos objetivos, se trabajó en la integración de distintos módulos, como el receptor IR, el encoder y la pantalla OLED. Además, se configuró la comunicación USB para la transmisión de datos en tiempo real. Este laboratorio permitio reforzar conceptos sobre protocolos de comunicación, manejo de timers y sincronización de señales digitales en sistemas embebidos.

Materiales utilizados:

- Microcontrolador STM32F4.
- Puente h TB6612FNG
- Receptor IR TSOP 38230.
- Pantalla OLED.
- Cable USB.
- Motor DC.
- Control SONY RM-Y173.
- Encoder de herradura.
- Protoboard.
- Montaje de rueda.
- Cables de conexión.

Softwares utilizados:

- Stmcube IDE.
- CodeBlocks.
- Excel.
- Hercules.

Procedimiento:

El procedimiento seguido para la implementación del sistema fue el siguiente:

1. Se llevó a cabo la conexión y prueba del receptor IR utilizando un osciloscopio, lo que permitió un análisis detallado de la estructura del protocolo de comunicación del control *Fig 1.0* . Esta fase fue crucial para determinar con precisión los tiempos de bit y las variaciones en las señales *Fig 2.0*, ya que al conocer estos parámetros pudimos recibir los datos que realmente necesitábamos del control; este envía un tren de 13 pulsos, de los cuales solo nos interesan 7 ya que los demás son de dirección. El uso del osciloscopio proporcionó visualizaciones claras que facilitaron la identificación de patrones en las señales recibidas, lo cual se comparó con el protocolo de comunicación SONY y era correcto.

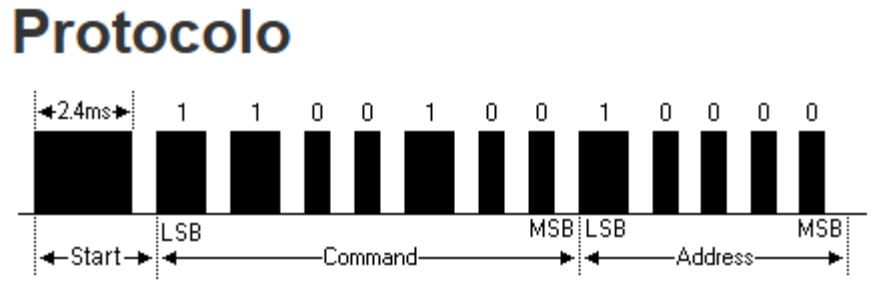

Fig 1.0 Visualización de protocolo control SONY. [1]

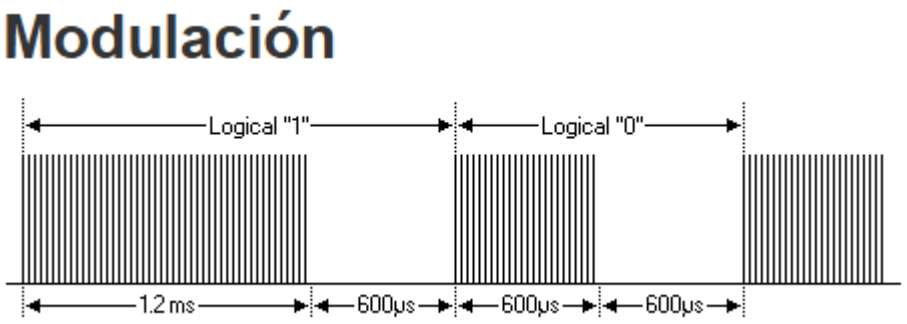

Fig 2.0 Visualización de la modulación del protocolo control SONY. [1]

2. Una vez obtenidos los datos del receptor, se procedió a la configuración de los timers del microcontrolador. Esto permitió medir con precisión los tiempos de los pulsos IR y clasificar los comandos recibidos de acuerdo con su duración. Se configuraron los siguientes pines en la STM32:

➔ **PIN B10**: Este pin se asignó como INPUT, es la entrada del receptor IR, lo que permite recibir y procesar las señales provenientes del control remoto.
➔ **PIN B4 y PIN B5**: Estos pines se utilizaron para controlar el sentido de giro del motor.
➔ **PIN A15 y PIN B15**: Estos pines se configuraron para la lectura de los encoders, utilizados para obtener datos sobre la dirección y revoluciones de la rueda.

- ➔ **TIM2, TIM3 y TIM5**: Estos timers fueron configurados como contadores para la medición de las revoluciones por minuto (RPM) del motor, para la captura de las señales IR y para la captura de los pulsos enviados por el encoder.
- ➔ **Interfaz USB**: Se habilitó la comunicación con la PC por el puerto COM4. Por medio de la interfaz de Hercules, se reciben los respectivos datos de las RPM.

3. En el siguiente paso, se desarrolló el programa para la decodificación de 5 comandos diferentes que provienen de un control remoto IR. Los comandos programados son los que se observan en la *Tab.01.* Para la comparación de los comandos recibidos con los que se esperaban, se utilizo el condicional if, que compara la variable que almacena el comando decodificado por el control remoto con los valores que se esperan, en caso de no cumplir con ninguna de las condiciones simplemente no ejecutaba ninguna instrucción y esperaba el siguiente comando desde el control.

Boton control	comando	Instrucción
Power	84	Derecha
Sleep	54	Detenerse
Multing	20	Izquierda
Enter	104	Transmision de datos activada
Reset	52	Transmision de datos desactivada

Tab 1.0 Comandos programados para control SONY RM-Y173.

4. Después, conectamos el motor y el microcontrolador a un puente H (tb6612fng) con el esquema proporcionado por el profesor, este módulo nos permite controlar la dirección del motor. De esta manera, cuando el microcontrolador asigne los comandos correspondientes, girará hacia donde se desee.

5. Además, incorporamos sensores de encoder para medir las revoluciones por minuto (RPM) y contar cuántas vueltas da el motor. Con el uso de un TIM conectado al internal clock del microcontrolador tomamos valores de 0 a 1 segundo, y en este instante obtenemos el valor de otro TIM configurado en modo entrada de GPIO, el cual va conectado a la salida del encoder. De esta manera, tenemos la cantidad de pulsos por segundo (RPS) y al multiplicarlo por 60, obtenemos las revoluciones por minuto (RPM). En el momento que capturamos los valores, reiniciamos los dos timers para evitar que se pierdan datos en la lectura.

6. Al ya tener configurados los dos encoders como entradas hacia el microcontrolador, realizamos la codificación necesaria de tal manera que detectara el sentido de giro hacia el cual estaba orientándose la rueda; dentro de el ciclo principal del código (While) establecimos que tomara lectura del estado de ambos sensores (para que actualizara el contador independientemente si se encontrara encendido o no el motor), de esta manera si detectaba que eran diferentes, corresponde a que estaban pasando por una de las guías de la rueda, al ingresar en esta condición, se comparó nuevamente el valor de cada encoder para saber cual estaba dentro y fuera de la guia, asi determinamos hacia donde rota la rueda y se incrementaba o decrementaba un contador que almacena la cantidad de giros respectivamente.

7. Configuramos la condición enviada por control remoto para activar para la transmisión de datos con el computador por medio del USB, además se asignó una condición adicional para que los datos se transmitieran únicamente cada 200 ms y no saturara el puerto. Utilizamos la función **CDC_Transmit_FS()** con la cual se envía una cadena de caracteres que se declara instantes antes de la transmisión; con la función **SPRINTF()** se toma el valor actualizado de la variable que almacena las RPM y la asigna a otra variable de tipo CHAR, la cual es la transmitida.

8. Continuamos con la programación de una pantalla OLED SSD 1603 que actualiza en tiempo real (5ms) las RPM, la cantidad de vueltas que ha realizado la rueda y los comandos que se reciben desde el control. Se establece este tiempo para que sea acorde a el funcionamiento de la rueda, y no retrase el programa, ya que realizar este proceso toma una cantidad de tiempo considerable debido a la estructura de las funciones utilizadas. De igual manera que en el inciso anterior, utilizamos la función **SPRINTF()** para convertir las variables en strings, carácter a carácter. y con la función **ssd1306_WriteString()** se toma esa cadena de caracteres y se registra en una posición de la pantalla. Cuando se realiza este proceso con todas las variables, se utiliza la función **ssd1306_UpdateScreen()** que actualiza la pantalla con los datos actualizados.

9. Luego, procedimos a guardar los datos obtenidos por la comunicación USB para poder graficar la curva de la aceleración y desaceleración del motor, como se evidencia en la *Fig. 03.* Seguimos las instrucciones dadas en la guía para crear un archivo de texto que guardará los datos transmitidos por el puerto COM. Con el control remoto, se envió la instrucción para encender el motor y esperamos hasta que llegara a su máxima velocidad, luego se envio la instrucción para apagarlo y esperamos que se detuviera completamente. Así obtuvimos los datos tanto de aceleración como de desaceleración del sistema, que posteriormente se graficaron por medio del software Excel tomando estos datos respecto al tiempo transcurrido.

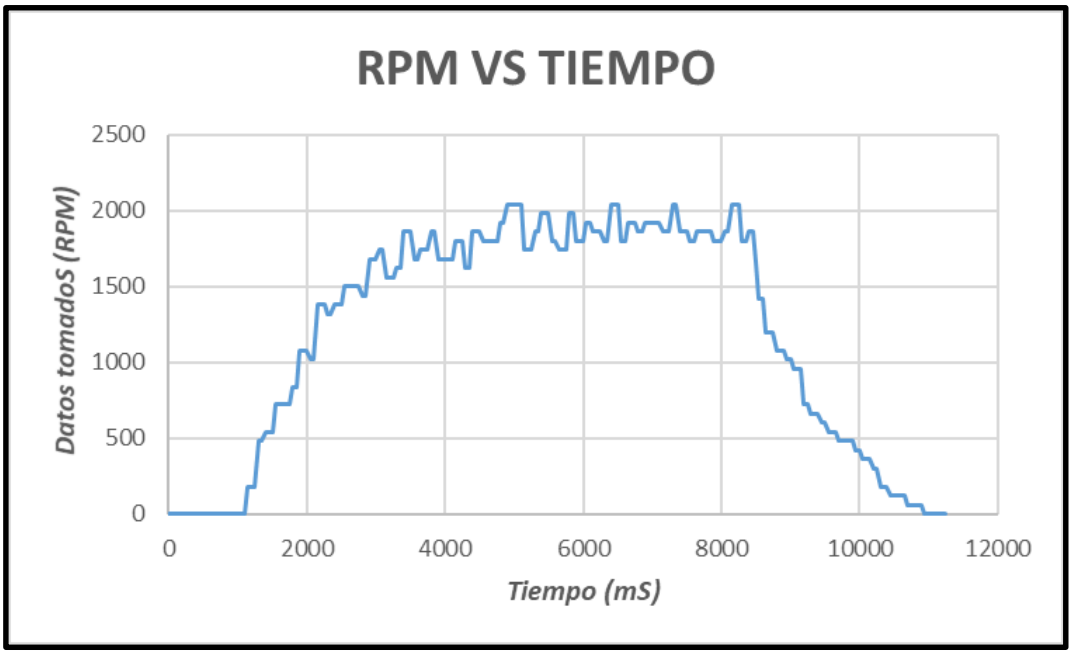

Fig 3.0 Gráfica de aceleración y desaceleración del motor .

10. Finalmente, se realizó un filtro para la señal. ya que la señal previamente obtenida presentaba picos imprecisos. Al implementar una función dentro del código principal del microcontrolador, regulamos estos valores considerablemente, se tomaron fracciones de tiempo de ejecución junto a sus respectivos valores de “rpm”, esto se promedió para tener una medida más estable en cada instante de tiempo. En la *Fig. 04* se presenta la gráfica filtrada y se evidencia como los picos ya no son tan pronunciados.

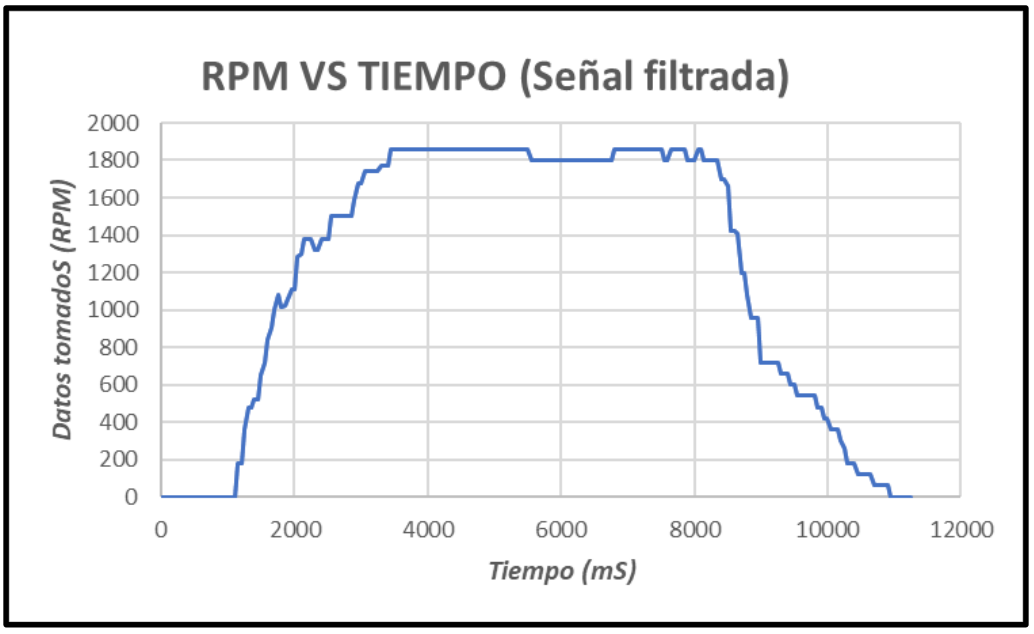

Fig 4.0 Gráfica filtrada de aceleración y desaceleración del motor .

Dificultades presentadas:

Durante el desarrollo del laboratorio, se presentaron diversas dificultades, las cuales se resolvieron de la siguiente manera:

- Lectura incorrecta del receptor IR: Inicialmente, se intentó leer los pulsos a través del pin B2 del microcontrolador, sin embargo este no captaba correctamente la señal. Para solucionar el problema se cambió la conexión al pin B10, logrando una lectura estable y precisa.

- Función **ERROR_HANDLER()**: Se presentaron dificultades al calcular las RPM ya que al programar la operación correspondiente, la ejecución del programa se redirigia a la función ERROR_HANDLER(), la cual protegía el microcontrolador de entrar a un fallo lógico. Inicialmente, el programa dejaba de hacer lo que se requería, por lo cual se empezó a compilar el código en modo debugger hasta que llegamos a esta función que desconociamos, y nos dejaba en un ciclo *While* infinito. Al consultar sobre esta función, comprendimos que la operación para calcular las RPM estaba realizando una división por cero; con lo cual modificamos el cálculo y se soluciono el problema.

- Interferencia entre la comunicación USB y la pantalla OLED: En las primeras pruebas, la comunicación USB interfería con la actualización de la pantalla OLED. El problema fue que estábamos utilizando la función **HAL_Delay()**, lo cual pausaba el programa y desincronizaba diferentes variables que tenían que ejecutarse cada cierto tiempo. Reemplazamos esa función con **HAL_GetTick()** y establecimos variables que almacenarán el momento en el que el código pasará por ese punto, para posteriormente con el uso de un condicional comparar si han transcurrido 200ms desde la última iteración.

- Problema mecánico en la rueda del encoder: Inicialmente, se utilizó una rueda de plástico, pero esta presentaba movimientos inestables que afectaban la precisión de los datos obtenidos por el encoder por las vibraciones que se presentaban en el sistema. Se optó por reemplazarla con una rueda de madera, lo que mejoró considerablemente la estabilidad del sistema.

- Conteo de vueltas según dirección de giro: Al implementar el proceso de lectura y conteo sobre los dos encoders, se presentan lecturas precisas y fiables cuando se la velocidad de giro es alta; pero en el momento que se realiza el giro lentamente el contador empieza a sumar de manera descontrolada, esto sucede solamente en una dirección ya que al realizar el mismo proceso hacia el otro sentido, el contador realiza su función correctamente.

- Actualización de variables: El sistema funciona correctamente cuando se realiza el movimiento de la rueda manualmente y cuando el motor va a cierta velocidad, pero cuando llega a su velocidad máxima (con alimentación 6V) se empiezan a presentar problemas de lectura en el control, al consultar al profesor, comprendimos que al ejecutar el código se puede quedar constantemente tomando las lecturas de los encoders, por lo que a tantas revoluciones no tiene el tiempo necesario para leer los datos del control. Esto se soluciono disminuyendo el voltaje de alimentación del motor a 3.3V, con esta tensión el sistema si es capaz de leer tanto los encoders como las señales enviadas del control a la máxima velocidad.

Diagramas:

- Diagrama de flujo (*Archivo anexo ¨Diagrama de flujo del sistema¨*).

- Diagrama esquemático (*Archivo anexo ¨Laboratorio* 4¨).

Conclusiones:

1. La implementación de la pantalla OLED debe manejarse con precaución al implementarse en el código, para evitar bloqueos y problemas de sincronización en el sistema. De esta manera, comprendimos la importancia de dejar posibilidades de bucles infinitos y siempre acondicionar estos con tiempos de salida.
2. Debe evitarse el uso de funciones como HAL_Delay() que detienen la ejecución del programa y puede desincronizar tiempos, cambiar condiciones que deben actualizarse constantemente, y evitar que ingrese a ciertas condiciones que se han registrado.
3. Aprendimos que decodificar comandos infrarrojos no es solo cuestión de recibir datos, sino que requiere un análisis detallado de los tiempos de los pulsos para su correcta interpretación y decodificación. Detalles como confirmar el comando recibido con la dirección negada que igualmente se envía por la señal del control permite evitar falsas lecturas o mala clasificación de los pulsos.
4. Implementar la medición de las RPM del motor nos permitió comprender mejor el uso de los temporizadores del STM32, así como la importancia de evitar errores en los cálculos, como la división por cero; que nos llevó a analizar y corregir la lógica del código de programación.
5. Finalmente, al utilizar el Tim por el puerto GPIO del microcontrolador se garantizó una medición de datos en tiempo real, debido a que este puerto trabaja paralelamente con el programa principal, pero sin afectar en la medición general del sistema lo cual era necesario para obtener la medición correcta.

Referencias:

[1] S. *Bergmans.* (2023, DIC 31). Protocolo SIRC de Sony(2nd ed.) [Online]. Available: https://www.sbprojects.net/knowledge/ir/sirc.php