Sistemas Basados en Microprocesador

Integración y desarrollo de una aplicación: Medidor de Luminosidad

Alumno:
Lukas Gdanietz de Diego
Puesto Nº: N/A

2021-22

Departamento de Ingeniería Telemática y Electrónica
Universidad Politécnica de Madrid

Índice del documento

1 OBJETIVOS DE LA PRÁCTICA

1.1 Resumen de los objetivos de la práctica realizada

Se deben enumerar los objetivos que ha alcanzado al realizar la práctica. Enumérelos de forma precisa y sencilla.

1.2 Acrónimos utilizados

Identifique los acrónimos usados en su documento.

USART	Universal Synchronous/Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit (Communication protocol)
O.S.	Operating System, refiriéndose a RTOSv2
HAL	Hardware Abstraction Layer
LCD	Liquid Cristal Display
PWM	Pulse Width Modulation
RGB	Red Green Blue
CS	Chip Select
MOSI	Master Out Slave In
CMSIS	Córtex Microcontroller Software Interface Standard

1.3 Tiempo empleado en la realización de la práctica.

Debe realizar una descripción sencilla del tiempo que ha dedicado a la realización de las actividades relacionadas con la práctica.

Bloque	Tiempo de desarrollo	Tiempo asignado a debug y corrección de errores
USART	4-6h	3h
I2C-TSL2591	6-8h	3h
Clock	1h	0h
RGB	2h	0,5h
Joystick	4h	1h
Principal	6h	8h
Lcd	2h	1h
Informe	10h	

1.4 Bibliografía utilizada

[RD1] Pagina de fabricante stm32
[RD2] Pagina de mbed ST_NUCLEO-F429
[RD3] DOCS NUCLEO F429
[RD4] DeepBlue Tutoriales sobre la capa HAL
[RD5] Beginning STM32
[RD6] UPV STM32 HAL Docs
[RD7] CMSIS RTOS2 Docs
[RD8] CMSIS USART Driver Interface Docs
[RD9] CMSIS I2C Driver Interface Docs
[RD10] C online compiler
[RD11] Tutorialspoint.com

1.5 Autoevaluación.

Mi mayor dificultad en el desarrollo de la practica ha sido no tener un ordenador con Windows. En sí, ha sido mi mayor dificultad al cursar la asignatura, ya que básicamente me impedía realizar las practicas, ni entregarlas en tiempo y forma. Aun así, las realice en STM32CubeIDE (Eclipse con una máscara creada por STM32 para utilizar su debug tool y herramientas de configuración)
Así mismo me gustaría agradecer al personal en el SICO como a Francisco Aznar Ballesta que me han permitido prestar un ordenador de la universidad para poder realizar la práctica.

Ahora en cuanto a las competencias adquiridas en la asignatura:

CODIGO	Objetivo	Adquisición	Anotaciones
RA971	Manejar temporizadores hardware para gestionar la temporización y sincronización de una aplicación	✓	Ya había utilizado timers previamente a la asignatura
RA970	Establecer y gestionar las comunicaciones entre dos sistemas utilizando diferentes interfaces	✓	Ya había utilizado estos periféricos, y otros más, como CAN-BUS. También escribí una implementación de RS-485, aunque al final me decante por MODBUS
RA907	Desarrollo de aplicaciones en grupos de trabajo	✓	
RA736	Interpretar las especificaciones de funcionamiento de un sistema basado en microcontrolador de mediana complejidad	✓	No es mi primer rodeo
RA737	Escribir el código necesario para desarrollar una aplicación basada en microcontrolador de mediana complejidad	✓	No es mi primer rodeo
RA730	Conectar un periférico a un microcontrolador utilizando interfaces basadas en protocolos estándar	✓	
RA733	Aprender a manejar cualquier periférico de mediana complejidad de un microcontrolador a partir de la documentación proporcionada por el fabricante	✓	
RA968	Manejar instrumentación electrónica específica para el desarrollo de sistemas basados en microprocesador	✓	
RA738	Elaborar el informe que justifica y describe la toma de decisiones adoptadas en el desarrollo de un proyecto y defenderlo oralmente con precisión y detalle	✓	
RA735	Analizar la arquitectura software y hardware de sistemas basados en microcontrolador de mediana complejidad	✓	
RA734	Manejar entornos de CAD para la codificación, la compilación y la depuración de aplicaciones basadas en microcontrolador	✓	Al final no estuvo tan mal utilizar KEIL, aunque al final no lo pueda utilizar en el futuro.

2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR

2.1 Diagrama de bloques hardware del sistema.

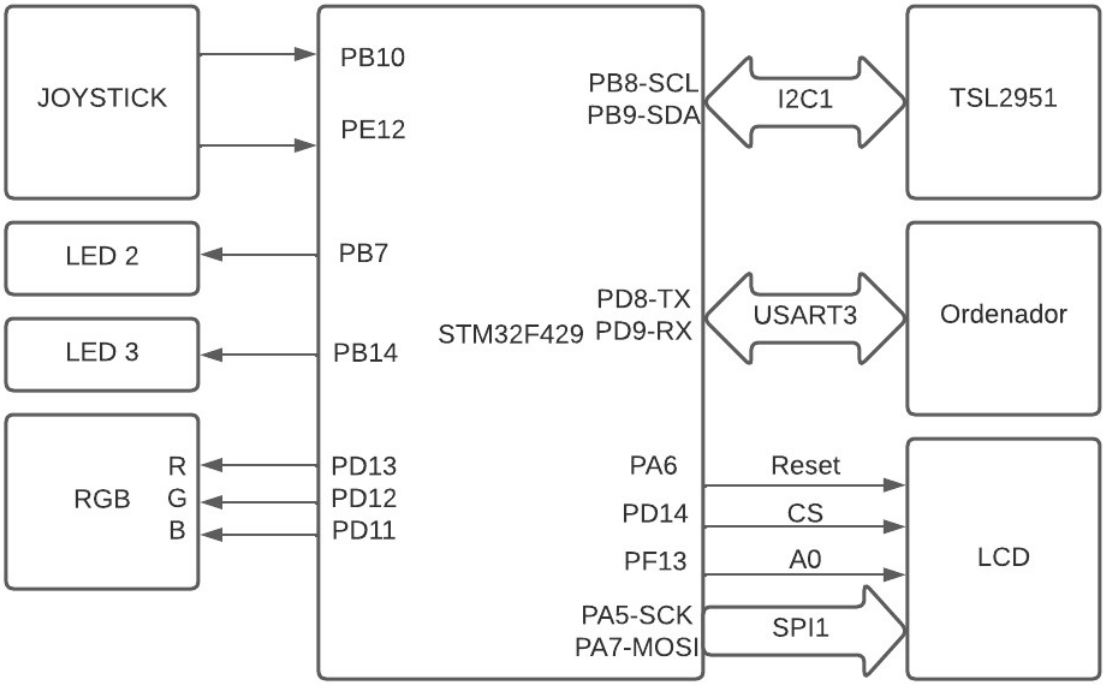

Ilustración 1 Interconexión en dispositivo

2.2 Cálculos realizados y justificación de la solución adoptada.

2.2.1 USART

El STM32 se configura para que utilice el periférico USART3

Se configura el periférico para que funcione de forma asíncrona, con 8 bits de datos, 0 bits de paridad, 1 bit de stop, y una velocidad de transmisión de 19200 Baudios. Además, se activan tanto la recepción como la transmisión, con salida en los pines PD8 (TX) & PD9 (RX).

2.2.2 I2C

El STM32 se configura para que utilice el periférico I2C1

El periférico se inicializa para que opere a 400Khz.

2.2.3 TIMER

El STM32 se configure para utilizar el TIMER 4.

La funcionalidad del TIMER 4 será la de un PWM. Para obtener un resultado óptimo sin parpadeos visibles, se desea obtener una frecuencia de operación de 40 KHz. Para conseguir esta frecuencia, se establece un preescaler de 20. Con ello se consigue dividir la frecuencia de APB1, de 84Mhz a 4,2MHz.
Se configura el registro de Periodo del timer 4 a 100. El motivo para configurarlo a 100, es que de esta forma se vuelve a dividir la frecuencia de 4.2Mhz a 42Khz. Por tanto, conseguimos una frecuencia fija de 42 KHz.
Se activa la auto carga del registro de periodo, para que una vez finalizada la cuenta de 0 a 100 en el registro de periodo, vuelva a comenzar, y se establece **inicialmente** en el registro de comparación un valor de 50, para tener un duty cycle de 50%. Esto quiere decir que inicialmente, el 50% de λ estará el pulso en alto, y a la otra parte en bajo. Este valor del registro de comparación ira variando con una frecuencia de 2 segundos (más adelante) para conseguir el efecto dimming.

2.2.4 SPI

El STM32 se configura para utilizar el periférico SPI1

Se configura para que funcione en modo master, en modo 3 (Lógica en 1, y polaridad en 1). El tamaño de palabra es 8, y el sentido es del bit más significativo al bit menos significativo. Además, se configura para que opere a 2 MHz.

3 SOFTWARE

3.1 Descripción de cada uno de los módulos del sistema

- **PRINCIPAL:** Gestiona la máquina de estados del dispositivo y es el orquestador entre los distintos hilos de tareas. Se ocupa de la comunicación con los hilos de *"lcd"*, "*com*", "*TSL2591"*, *"RGB", "Joystick"* y "*clock".* Se ocupa de leer los comandos que recibe por "*joystick"* y *"COM"* y modifica el estado de la máquina de estados según la programación interna. Además, según el estado y los comandos que recibe de "joystick" y "COM" controla con el uso de Flags, colas de mensaje y funciones directas el funcionamiento de los demás hilos.
  Por último, gestiona internamente un buffer circular con el cual guarda las últimas medidas más recientes.
- **LCD:** Controla independientemente de los demás hilos la pantalla LCD incluida en el dispositivo. Primero realiza una operación de reset y configuración de la pantalla LCD, a continuación, prepara un buffer, lo rellena según lo que reciba por su cola de mensajes *mid_MsgQueue_lcd* y lo transmite por el protocolo SPI a la pantalla LCD.
- **COM:** Recibe datagramas de longitud no definida por el periférico USART, hace comprobación de errores en el datagrama, si el datagrama es no valido, lo descarta, si es válido lo transmite, por la cola de mensajes *mid_MsgQueue_com_rx*
  Entre tanto, espera durante 1 segundo (para no bloquear el proceso indefinidamente) una respuesta de parte del proceso *principal* por la cola de mensajes *mid_MsgQueue_com_tx* y la retransmite, introduciendo una cabecera, un final de datagrama, el comando transmitido y el tamaño total del datagrama.
- **TSL2591:** Gestiona la comunicación con el sensor TSL2591 a través del protocolo I2C. Realiza la configuración inicial del sensor, y luego en cada medida comprueba primero si los datos que se encuentran en los registros de datos del sensor son válidos. Si lo son, los lee, y realiza los cálculos para convertiros en una luminosidad relativa porcentual.
- **RGB:** Se ocupa de encender y apagar los leds según los comandos que reciba por la cola de mensajes *mid_MsgQueue_rgb* y además se ocupa de cambiar el duty cycle del generador PWM, para reproducir un efecto de dimming cada 2 segundos.
- **JOYSTICK:** Se ocupa de leer y gestionar los rebotes producidos en el joystick del dispositivo. Una vez trata el joystick, cada vez que hay un evento significativo (Pulso corto, pulso largo en ambas direcciones), lo retransmite por la cola de mensajes *mid_MsgQueue_joy*
- **CLOCK:** Mantiene la hora gracias a un timer del sistema operativo que llama un callback que incrementa la hora por 1 segundo. Además, incluye una función para cambiar la hora. La hora está disponible globalmente en todo el proyecto.

3.2 Descripción global del funcionamiento de la aplicación. Descripción del autómata con el comportamiento del software (si procede)

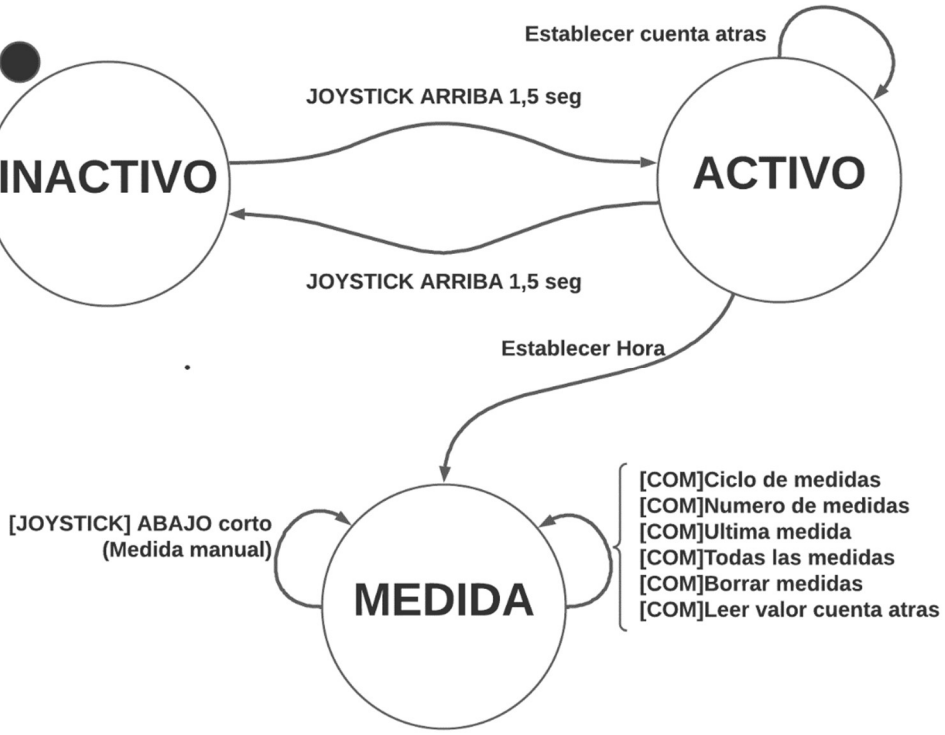

Ilustración 2: Diagrama de bolas de maquina de estados

La aplicación central del dispositivo es un sistema de muestreo y logueado de la luminosidad. Para cumplir con este propósito dispone de un hilo orquestador llamado **principal** cuya función es la de controlar la máquina de estados del propio programa y controlar los hilos secundarios según las tareas a realizar en el momento de ejecución. El sistema dispone de 3 estados:

1. **ESTADO INACTIVO:** En este estado el hilo principal espera la recepción de un mensaje desde la cola de mensajes del joystick, que indique la activación del sistema. Una vez recibe la confirmación por un mensaje diciéndole que el joystick ha estado en pulsación arriba durante 1,5 segundos (Pulsación larga), este pasa al siguiente estado: ESTADO ACTIVO
2. **ESTADO ACTIVO:** En este estado, el sistema espera varios comandos, que actúan de diferente manera:
    a. **[JOYSTICK] Arriba largo:** Devuelve la máquina de estados al estado inactivo
    b. **[COM] Establecer cuenta atrás:** Mantiene la máquina de estados en el mismo estado.
    c. **[COM] Establecer hora:** Pasa la máquina de estados al estado medida.

    Además, todos estos comandos (menos el de Arriba largo) configuran el sistema de una manera u otra
3. **ESTADO MEDIDA:** En este estado, las medidas están habilitadas, y todos los comandos que recibe son operaciones relacionadas en cuanto a las medidas. Entre ellas se encuentran
    a. **[COM]Ciclo de medidas**
    b. **[COM]Numero de medidas**
    c. **[COM]Ultima medida**
    d. **[COM]Todas las medidas**
    e. **[COM]Borrar Medidas**
    f. **[COM]Leer valor cuenta atrás**
    g. **[JOYSTICK]Abajo corto**

    De este estado no se puede volver a otro estado.

3.3 Descripción de las rutinas más significativas que ha implementado.

3.3.1 COM

Al ser la comunicación con el microcontrolador uno de los pilares del proyecto, se comienza el proyecto por el paquete com. Se fija una estructura predefinida de datagramas UART. Un datagrama UART tiene la siguiente estructura:

CABECERA	COMANDO	LONGITUD	DATOS	FINAL
0X01	Type(HEX)	Length(HEX)	/-/-/-/	0xFE

Como se puede observar, la estructura del datagrama es fija, aunque varía de tamaño y comando. Por tanto, el paquete com se construye de tal forma que el driver UART de cmsis, primero espera 3 bytes con la que consigue un inicio de datagrama, el tipo de comando y el tamaño del paquete total.

Una vez recibe estos 3 bytes, reajusta el tamaño del buffer de recepción para alocar el resto del datagrama, de tamaño variable. De esta manera se consigue utilizar de una mejor manera los recursos del microcontrolador sin tener que estar haciendo comprobaciones por cada byte que se recibe en el periférico USART.

Al finalizar la recepción de todo el datagrama, se comprueba si el final de datagrama es el correcto. En caso de no serlo, se desecha el datagrama, y se comienza de nuevo desde el principio con la recepción de 3 bytes de un nuevo datagrama.

Si el final de datagrama es el correcto, se manda un mensaje por una cola de mensajes al programa *"principal",* y se espera respuesta por parte de esté durante un segundo. La respuesta debe contener información como el tipo de respuesta, la sección de datos, el tamaño de los datos y si le siguen otros mensajes o no.

Además, adicionalmente, para evitar que haya perdida de paquetes a transmitir (Ya que el periférico USART está configurado para operar a 19200 baudios, haciendo que las transmisiones tarden múltiples ms, mientras que el resto de procesos operan mucho más rápido, y pueden resolver tareas más rápido), se implementa un sistema anti escritura en el driver USART mientras que la anterior transmisión USART no haya finalizado, que bloquea la operación del resto de paquete com.

3.3.2 TSL2591

El paquete TSL2591 se encarga de la comunicación con el sensor de luminosidad TSL2591. Para conseguir este objetivo, hace uso del periférico I2C1. Su configuración inicial es descrita en el apartado 2.2.2.

Una vez inicializado el periférico I2C1, (y ha arrancado el **O.S.**) se procede a configurar el sensor. Se configuran los registros *"Registro de Habilitación" (0x00) (***Enable Register**) para activar ALS y el Power On. Consecuentemente (en una única escritura) se configura el *"Registro de Control" (0x01) (***Control Register**) para una ganancia del sensor media "*Medium gain mode"*
El hilo que se dedica a la comunicación con el sensor, utiliza un sistema de Flags, tanto como para funcionamiento interno como para la comunicación del hilo *"principal"* con el hilo del sensor TSL2591.

Los Flags del hilo TSL2591 con id tid_Th_I2C tiene 2 secciones, la primera está reservada para el hilo *"principal"* para indicar una única lectura o múltiples lecturas, y en el último caso, tiempo entre medidas y numero de medidas. La segunda parte de los Flags están reservada para llamadas de sistemas y control del periférico I2C (Devolución de códigos de funcionamiento y error)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Start 1	Start m	FREE		Numero de ciclos								Tiempo entre ciclos			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Tiempo entre ciclos				FREE			*RESERVED* I2C DRIVER FLAGS								

Reparto de bits en gestor de Flags del hilo TSL2591

Esta gestión de los bits libres permite al hilo *principal* comunicar y ordenar al hilo TSL2591 cuando realizar una medida o el número y tiempo entre medidas.

El hilo TSL2591 realiza según como se coloquen los Flags, una operación de medida única o múltiples medidas. Al realizar una medida, siempre se comprueba antes en el "*registro de status* "del sensor el bit *"AVALID"* para comprobar si los registros de datos contienen datos válidos. En caso de no tener datos válidos, la lectura no se efectúa. Si *AVALID* se encuentra a '1', se efectúa la medida, se procesan los datos para convertir la medida en un porcentaje relativo según una luminosidad mínima y máxima preestablecidos y por último se escribe en una cola de mensajes con ID *mid_MsgQueue_I2C*.

3.3.3 RGB

El paquete RGB se ocupa de encender los leds de estado y de controlar el efecto dimming del led verde del RGB.

Se utiliza un timer hardware (TIMER 4) para conseguir un PWM por hardware por el canal 1. Para cambiar el duty cycle del PWM, se utiliza un timer gestionado por el O.S. Este timer llama un callback cada 25ms de tal forma que se produzcan 80 pasos de variación cada 2 segundos en sentido positivo y en sentido negativo. De esta forma el cambio de intensidad varía entre un 10% y un 90% cada 2 segundos.
Además, contiene 2 timers más:

- Timer cada 1 segundo re-disparable, que enciende y apaga un led (*led2, Azul*) para señalizar el estado activo.
- Timer de 350 ms mono-disparo, que apaga un led (*led3, Rojo*) para señalizar una lectura.

3.3.4 JOYSTICK

El paquete joystick se ocupa la lectura del joystick conectado a los pines PE12 (Abajo) y PB10 (Arriba).

Este paquete debe eliminar cualquier posible rebote que introduzcan los actuadores del joystick ya que pueden dar falsas instrucciones al microcontrolador. Para lograr eliminar estos rebotes se utiliza un time-lock.

Los pines del microcontrolador que están conectados al joystick están configurados de tal manera que produzcan una interrupción. Cuando se produce una interrupción, se activa un Flag del hilo joystick llamado "*IRQ_FLAG"*

Este hace que el hilo active un timer del sistema de 50ms para esperar un "*tiempo muerto"* Una vez pasado ese tiempo muerto, el hilo vuelve a reactivarse y lee el estado del pin que desato la interrupción en primer lugar. Si sigue activo, se activa otro timer para comprobar en el futuro si la pulsación es una pulsación larga u corta.

Una vez se ha completado todo este proceso, se envía una trama por la cola de mensajes *mid_MsgQueue_joy* para que lo gestione el hilo *principal.*

3.3.5 CLOCK

El paquete clock se ocupa de mantener la hora en el dispositivo. Para cumplir con este cometido, hace uso de un timer de O.S. que llama a un callback cada segundo para que incremente por un segundo el reloj. Las variables del reloj son accesibles desde todo el programa, aunque solo el hilo *principal* haga uso de ellas.

Para modificar la hora, se provee de una función global *set_clock* que permite al sistema cambiar la hora del dispositivo.

3.3.6 LCD

El paquete LCD se ocupa de establecer comunicación con el display LCD, crear una estructura en un buffer que en el lcd se asemeje al texto introducido por la cola de mensajes del lcd.

4 DEPURACION Y TEST

4.1 Pruebas realizadas.

Descripción del método de prueba utilizado para comprobar que las rutinas funcionan adecuadamente. Resultados de los tests. Indicación explicita de si el test es correcto o incorrecto. En este punto debe hace especial hincapié en definir:

1. Cuál es el objetivo de la prueba, indicando los módulos implicados
2. Cuál es el proyecto de Keil que permite realizar la prueba
3. Cuáles son las condiciones de entrada que permiten ejecutar la prueba
4. Cuáles son los resultados que se esperan y cuáles son los realmente obtenidos.

4.1.1 DEPURACION & TESTs USART

Se fueron realizando numerosas pruebas a lo largo del proyecto, cada vez que se introducía un nuevo cambio que era vinculante con el periférico USART, o que le afectaba de algún modo. Aquí se detallan algunos de los TESTs y problemáticas encontradas.

1. Se desarrolló inicialmente el periférico siguiendo los requisitos solicitados para el proyecto. Cuando se comenzó a realizar el testeo del periférico, al principio aparentemente no funcionaba. Más tarde, gracias a un analizador lógico, se concluyó que el problema no venía del microcontrolador, ya que, sí que recibía las tramas correctamente, sino del programa utilizado para transmitir las tramas, el teraterm. Probablemente, esto se debiera al mal uso del teraterm. Aun así, se decidió cambiar de programa y se empezó a utilizar RealTerm.

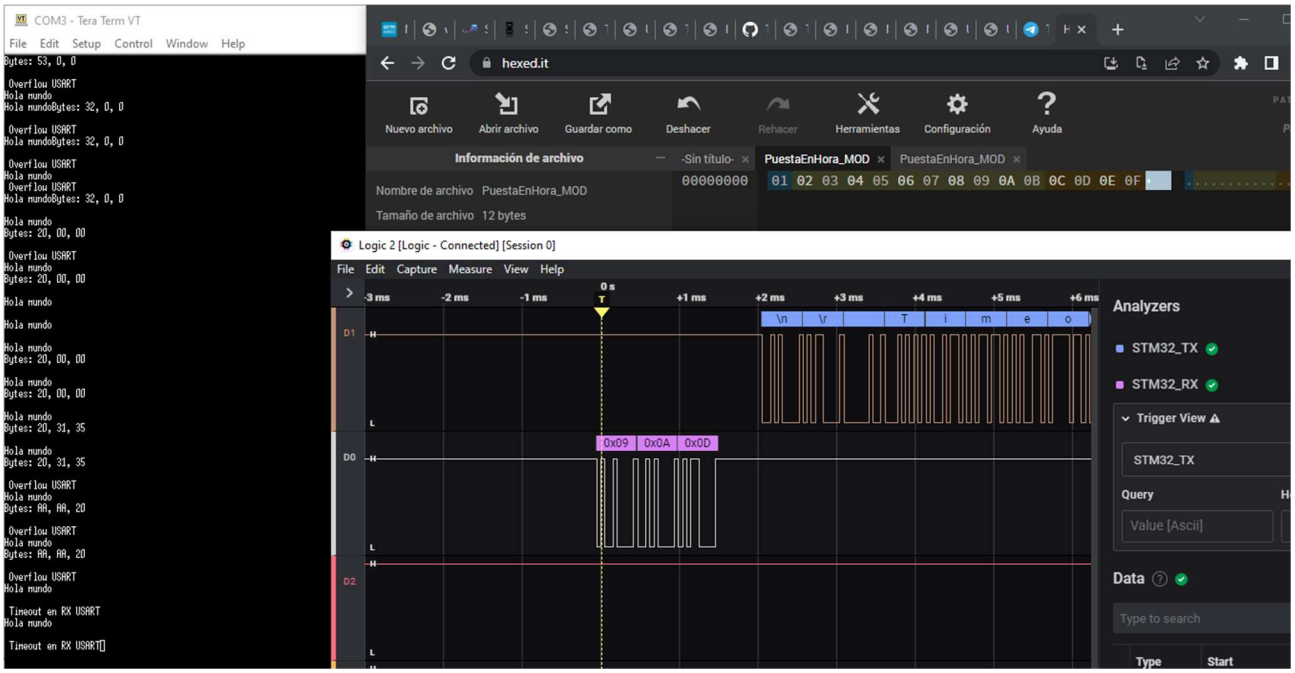

Ilustración 3 Datagrama no correctamente transmitido

2. Durante el desarrollo del dispositivo y testeo de otros módulos, se descubrió que “se perdían” datagramas. Una rápida observación en el analizador lógico, concluyo que realmente no se estaban perdiendo en la transmisión, sino más bien no se estaban transmitiendo. Se concluyó que el periférico trabajaba a una velocidad muchísimo más baja de la que el resto del microcontrolador trabajaba, y por tanto se estaba produciendo un bottleneck en las transmisiones. Se desarrolló un lock que impedía introducir nuevos mensajes en el driver USART si no había finalizado todavía la transmisión previa. Más tarde se intentó optimizar, pero no resulto en los resultados esperados.

3.

4.1.2 DEPURACION & TESTs I2C

Tras el desarrollo del módulo TSL2951, se decide testear su correcto funcionamiento, haciendo que capture la luminosidad cada segundo. Este test fue hardcoded con un *osDelay()*

1. Inicialmente, no funcionaba porque no se seteo el bit de acceso a registro de control. Una vez se modificó el código, el sensor devolvía datos tal como se esperaba.

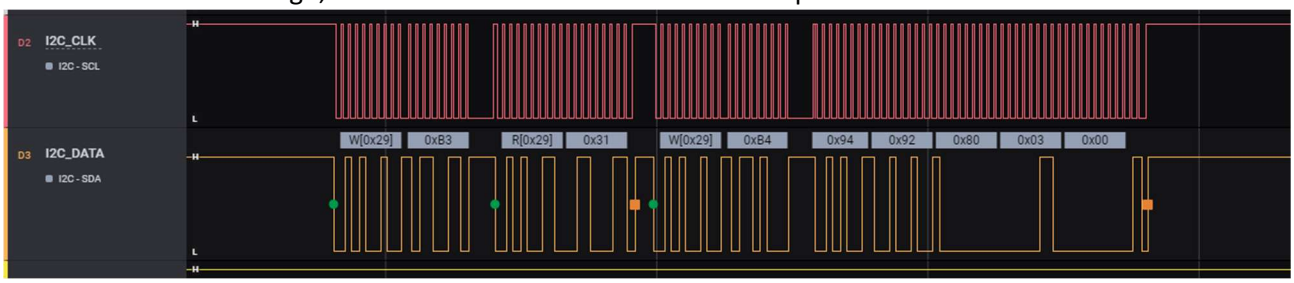

Ilustración 4 Captura de transaccion I2C

2. En un intento de optimizar el código y desalojar variables que no se volverían a utilizar más en el hilo, el uso de la función *free()* provoco que todo el microcontrolador crasheara. Encontrar el fallo fue muy difícil ya que se habían realizado muchos cambios a lo largo del día, y no había podido ser testeados (Fueron los días en los que había problemas con la licencia de keil), y además al crashear al comienzo, no era fácil de averiguar cuál de todos los hilos provocaba ese comportamiento.
   Se eliminó la función, y todo volvió a funcionar como antes.
3. Más adelante, cuando se quiso hacer que el hilo "*principal*" recogiera múltiples medidas, se vio que era relativamente ineficiente y que mayoritariamente bloqueaba al hilo "*principal*", por tanto, se decidió escribir otro sistema de múltiples medidas en el hilo TSL2591. Durante el testeo se observó que no reconocía ni el flag de arranque de una única lectura ni el flag de lectura de múltiples lecturas. Se encontró el fallo, y se corrigió. Esta búsqueda de errores fue realizada gracias a watches y Breakpoints en keil para detener, modificar variables y hacer correr el programa.

4.1.3 DEPURACION & TESTs PWM

Uno de los primeros hilos que se crearon fue el de RGB ya que era una forma fácil de visualizar si el microcontrolador seguía operando adecuadamente o no. Se testeo en un osciloscopio y se observó que la implementación realizada era incorrecta ya que en vez de cambiar el duty cycle, cambiaba la frecuencia del PWM. Se corrigió el error, al descubrir que erróneamente había estado modificando el registro del periodo en vez del comparador.

4.1.4 JOYSTICK

Se realizaron múltiples pruebas para comprobar que las pulsaciones se detectaban y se transmitían de forma correcta al hilo principal. Se estuvo debugeando con watches en keil. El funcionamiento del joystick era el esperado.

4.1.5 LCD

Se realizó un test al terminar de escribir el modulo del LCD. Inicialmente todo parecía funcionar correctamente, pero más adelante en el proyecto se observó que el buffer no se ponía a 0, y por tanto si había sido escrito la pantalla delante, no se borraba. Se corrigió el problema.

Aun así, más adelante, se observó que volvía a suceder exactamente lo mismo, cuando delante de lo que se escribía, había sido escrito previamente. Se volvió a corregir el problema rellenando el buffer con 0's, donde no se rellenaba el buffer con nuevos datos

4.1.6 PRINCIPAL

El hilo principal fue extensamente testeado cada vez que se completaba un hito. En cada caso, se iban añadiendo uno por uno los módulos y comprobando que cada uno de los módulos funcionaran como tenían que funcionar y que recibieran las señalizaciones tanto por Flags como por la cola de mensajes que era requerida para el funcionamiento del módulo. Algunos de los problemas que se detectaron fueron:

1. El hilo principal al principio hacia que el lcd fuera con 1 segundo de retraso (Aproximadamente) Se concluyó que la razón de este efecto era que la cola de mensajes del LCD estaba siendo escrita múltiples veces al segundo, saturándola. Se corrigió el problema comprobando previamente a escribir la cola de mensajes si había cambios significativos. Si no los había, se evitaba escribir en la cola de mensajes.
2. Durante el testeo del comando "*DEVOLVER TODAS LAS LECTURAS"* se descubrió que el módulo COM no llegaba a mandar todas las lecturas. En ese entones ya había sido implementado un mecanismo en el módulo COM que evitaba la perdida de paquetes a transmitir, bloqueando todo el hilo cuando aparecía una transmisión pendiente mientras que la anterior no había finalizado. Se concluyó que el problema aparecía cuando la cola de mensajes del hilo principal al hilo COM se saturaba, el hilo principal simplemente las descartaba. Se cambió el tiempo de intento de escritura en vez de 0 a 10 ms. Fue un fallo no implementar un mecanismo de salvaguarda en ese punto.
3. Al principio la estructura para realizar múltiples medidas de luminosidad, por el comando *"CICLO DE MEDIDAS"* estaba implementado en el hilo principal. Esto termino siendo desventajoso ya que bloqueaba y ralentizaba la operación del hilo principal. Además, el hilo era un poco *"spaguetti code"* Se cambió el código para que las múltiples lecturas las gestionara el modulo del sensor.
4. Buffer circular. El código del buffer circular se testeo múltiples veces fuera del microcontrolador y respondió tal como se esperaba (GitHub Gist), pero en el microcontrolador la función *malloc()* no era capaz de asignar un espacio lo suficientemente grande para el buffer. Al final se terminó solucionando asignándole un array de tamaño fijo, y no dinámicamente alojado como se había esperado inicialmente.
   Por otra parte, la función *parse_data_rx()* que debía introducir datos de contexto, como la hora, los minutos y los segundos del momento de la medición, las posiciones de *lum_cen*, *lum_dec y lum_uni* se correspondían con las posiciones de las variables *hora*, *min*, y *seg*. Al escribir las variables *hora, min, seg*, los datos almacenados en las mismas posiciones de luminosidad se reescribían con la hora, reescribiendo de esta forma la medida recién realizada. El problema se solucionó al realizar un *malloc()* para un nuevo tipo de dato, tipo *time_lum_pack_t*.
   Esta solución aun así dio sus problemas, que aparecieron una vez se testeo un ciclo de lecturas de luminosidad de más de 29 medidas. A la 29 medida, *malloc()* era incapaz de alojar más memoria para el tipo de struct time_lum_pack_t.
   Este problema así mismo se solucionó al realizar una operación de *free()* una vez se había guardado el dato en el buffer circular.