Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Escuela de Ingeniería Electrónica

Sistemas Digitales II

# Trabajo Práctico Nº 3

## Protocolos de comunicación serie: Módulos UART y protocolo RS-485. Acceso directo a memoria

Autores:

| Grupo N° 3 | |
|---|---|
| Nombres y Apellido | N° de Legajo |
| Bellini Valentin | B-6127/1 |
| Iván Saitta | S-5435/6 |

| Corrigió | Calificación |
|---|---|
| | |

2024

# Tabla de contenido

# 1. INTRODUCCIÓN

Este trabajo práctico aplica los contenidos temáticos de la asignatura para utilizar el microcontrolador Cortex M0+ de la placa de desarrollo FRDM-KL46Z con el fin de establecer una comunicación serie RS-485 con otro dispositivo. El funcionamiento del sistema se modela utilizando el formalismo de Maquina de Estado Finito / Statecharts UML y el código C debe reflejar el modelo propuesto. El intercambio de mensajes entre los dos dispositivos se implementa utilizando estructuras de datos de tipo cola circular en la recepción y transacciones a cargo de DMA en la transmisión. El desarrollo de la aplicación incorpora funciones de biblioteca provistas por el fabricante. La aplicación se programará y depurará utilizando el ambiente MCUXpresso y las bibliotecas asociadas.

# 2. OBJETIVOS

*Objetivos cognitivos:*

Se espera que los alumnos sean capaces de:

1. Especificar el comportamiento del sistema utilizando el modelo de Máquina de Estado Finito / Statecharts UML.
2. Aplicar los conocimientos adquiridos sobre la arquitectura de la familia de microcontroladores KL46 para desarrollar una aplicación basada en la placa FRDM-KL46Z.
3. Aplicar los conocimientos adquiridos sobre protocolos off-board y los módulos UART del microcontrolador.
4. Diferenciar las tareas relacionadas a la detección de tramas del procesamiento de las mismas, en la solución propuesta
5. Utilizar las funciones de biblioteca suministrada por el fabricante para soportar el desarrollo de la aplicación software.
6. Aplicar el criterio de reutilización de código al definir la estructura del proyecto.
7. Aplicar los conocimientos adquiridos para el desarrollo de drivers de dispositivos y modificarlos para la utilización de DMA en circunstancias donde represente una ventaja.
8. Aplicar el criterio de reutilización de código al definir la estructura del proyecto. Dividiendo las distintas funcionalidades o MEFs en archivos separados.

*Objetivos actitudinales:*

1. Promover el trabajo en equipo para obtener la solución a un problema.
2. Promover la habilidad de realizar una defensa de la solución propuesta para el problema planteado.
3. Promover la habilidad de elaborar un reporte escrito sobre el trabajo realizado.

# 3. PAUTAS PARA LA ENTREGA

*Material a entregar:*

- El modelo completo de la solución del problema planteado. El mismo deberá ser claro y legible.
- El informe de las tareas realizadas en base a la plantilla oportunamente subida al campus.
- El código de la aplicación desarrollada.

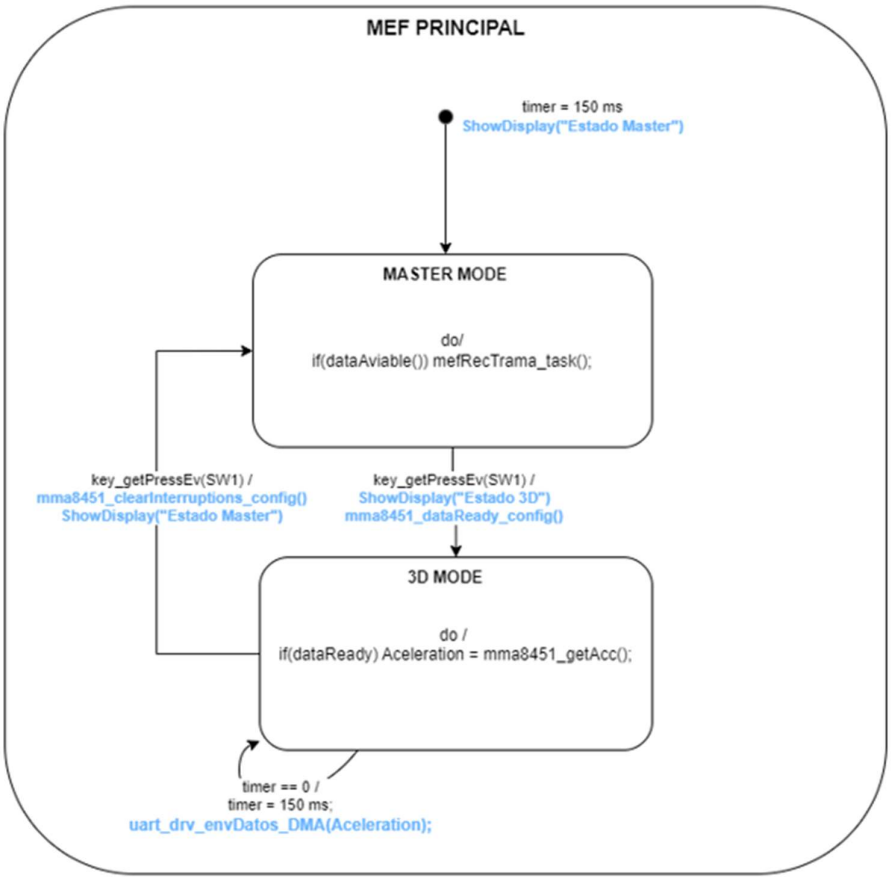
*Aspectos a tener en cuenta para la entrega:*

- Se deben incluir la/las MEF que modelan la solución propuesta. Si utiliza más de una MEF se debe explicar con claridad como colaboran entre sí y de qué forma se comunican, detallando los recursos con los que se plasma esta colaboración en el código.
- Si intervienen IRQs de periféricos, indicar cuales y que tareas se llevan adelante en sus rutinas de servicio. Especificar si han tenido algún cuidado especial en su manejo y las razones que lo justifican.
- Si se definen funciones, explicar de qué tareas son responsables.
- Explicar donde se ubican en el código las declaraciones de variables y funciones y donde se las invoca.
- Explicar las fases de funcionamiento del sistema y como se configuran los periféricos que intervienen, para cumplir con las tareas que tienen asignadas.
- Detallar y justificar la estructuración que se adoptó para el programa teniendo en cuenta las recomendaciones para su implementación.
- Tener en cuenta las actividades que se solicitan en la plantilla e incluirlas en el informe.

# 4. TAREAS DESARROLLADAS

## MEF Principal

El primero modelo de funcionamiento es llamado “MEF Principal”. En el mismo, cuyo diagrama se ve en la figura 1.1, se selecciona el modo de funcionamiento del programa.



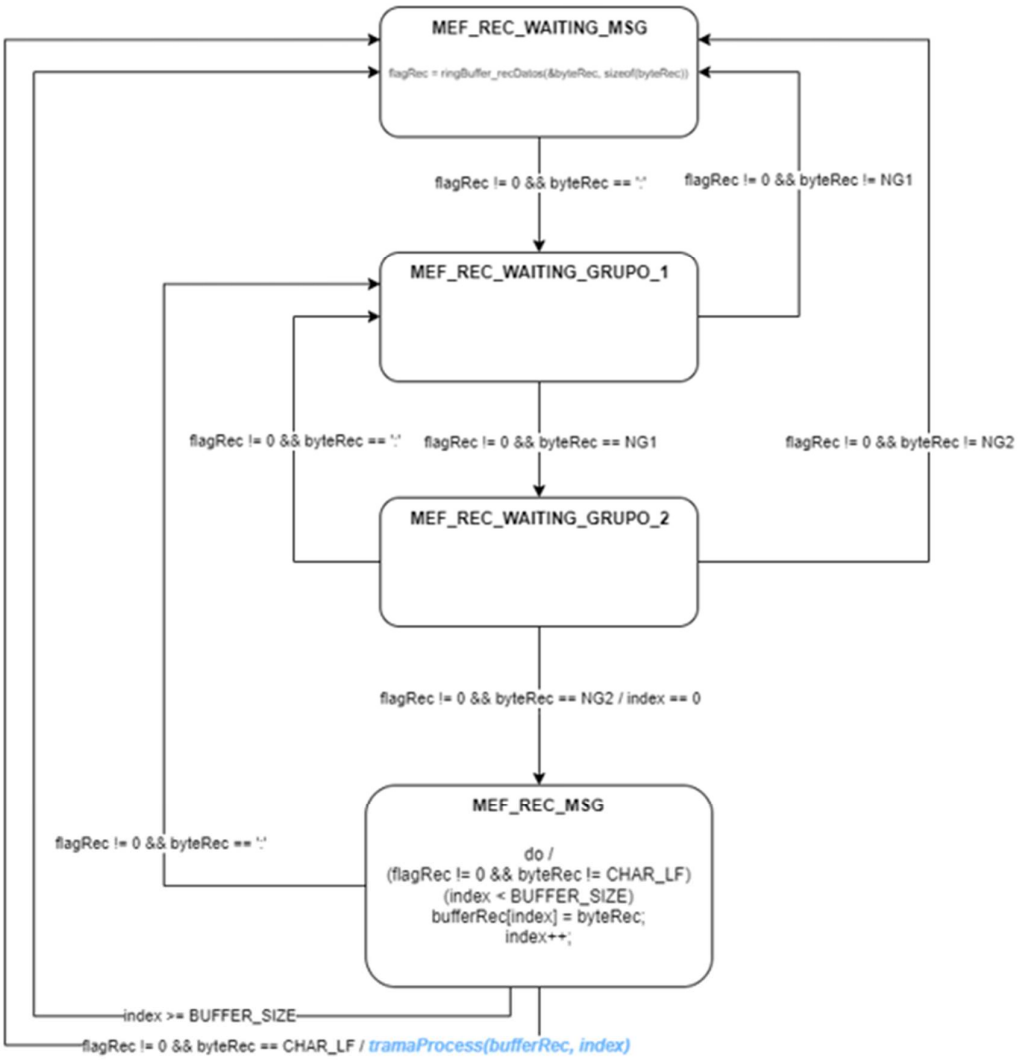

Esta MEF es sencilla, consta de 2 estados: Modo Master y Modo 3D. El intercambio entre ellos se da mediante el pulsado del switch 1 integrado en la placa de desarrollo FRDM-KL43Z. El manejo del pulsado del Switch no es motivo de este trabajo y fue tratado anteriormente en la materia (puede encontrarse la solución en el archivo “key.c”).

En ambos modos de funcionamiento se realiza una comunicación entre la PC y la placa de desarrollo mediante UART1, con un conversor UART/RS485 por un lado, y un conversor USB/RS485 por el otro. En el modo de funcionamiento master, se recibe tramas mediante el protocolo desde la PC, se procesa en otra MEF llamada “MEF REC TRAMA” donde se toman las decisiones de accionar sobre la placa y luego se imprime en una pantalla OLED, el estado actual del programa (La comunicación con el display es por SPI y tampoco es tema de este trabajo). En el modo 3D, el micro envía continuamente, cada 150 ms, los datos de aceleración tomados del acelerómetro MMA8451 integrado en la placa. Los datos son enviamos a la PC donde se utiliza un software para simular la

posición de la placa según los comandos recibidos. Nuevamente, la obtención de datos del acelerómetro no es motivo de este trabajo, habiendo sido realizado en el trabajo anterior. Se utilizó la misma librería desarrollada para el MMA8451, llamando a las funciones con configuraciones necesarias para este trabajo.

## MEF Rec_Trama

La modelización del modo master se lleva a cabo mediante una MEF anidada. En esta, se hace el procesamiento de la trama. En la figura 1.2 se puede ver la MEF diseñada.




El formato de la trama recibido es similar a MODBUS ASCII. Cada mensaje o trama comienza con el carácter de inicio ":" (dos puntos). Luego, con dos caracteres ASCII hexadecimales, representamos la dirección del dispositivo, en este caso, el número de grupo. (03). Una vez indicado el inicio y el grupo correcto, viene el código de función, es decir, un byte que indica el tipo de operación que se desea realizar. En este caso no habrá verificación de integridad con LRC, pero si tenemos un mensaje de finalización de trama "LF" (Line Feed). A continuación, se explicará la máquina de estado diseñada.

Como se puede observar, funciona como un proceso sincrónico por etapas. En el primer estado se espera el mensaje de inicio de la trama ":", en el segundo estado se espera el primer carácter de número de grupo "NG1" y en el tercer estado se espera el segundo carácter de número de grupo "NG2". En número del grupo es "03", por lo que en nuestro caso se espera que NG1 = 0 y NG2 = 3. Cuando hay algún error en la trama recibida, como se puede ver en la MEF, el estado cambia entre estos tres primeros estados continuamente según la trama recibida. Por ejemplo, si se recibe ":02", el carácter de inicio de trama y el primer número de grupo es correcto, pero al recibir NG2 = 2, se descarta la trama y se espera a recibir una nueva que comience con ":".

Una vez que se ha recibido correctamente el comando ":03", se pasa al estado de la trama donde se recibe el mensaje para actuar sobre la placa de desarrollo. En este estado, mientras no llegue el fin de control LF, se va almacenando la trama en un buffer circular. Si llega un ":" o el índice supera al buffer size, se descarta la trama y se vuele al inicio, mientras que si llega el mensaje de CHAR_LF indicando el final de la trama, entonces se ejecuta una función que procesa la trama, enviándole el buffer recibido. En esta función, como se verá a continuación, se procesa la trama recibida desde la PC para accionar sobre la placa, ya sea para accionar sobre Leds, Switches o Acelerómetro.

## Procesamiento de Trama

Para el procesamiento de la trama recibida en el modo master se implementa, como hemos mencionado, una función llamada *tramaProcess(),* que recibe como parámetros el buffer de datos y la longitud del mismo y se encarga de procesar esa trama, accionar sobre la placa y hacer él envió de datos por UART vía DMA. Para ello, la función implementa un switch-case para el primer bit de la trama, que indica sobre qué grupo de periférico se quiere actuar (led, switch o acelerómetro). Una vez decidido esto, se procede de diferentes formas:

- LED: Si el primer dato de la trama es un "0", entonces se indica acción sobre los leds. Se realiza un nuevo switch-case con el tercer elemento de la trama (buffer[2]), que indica si se desea encender, apagar o togglear el led. Entonces, para cada caso, se envía la acción correspondiente y el led deseado (buffer[1]) como parámetros a la función *handleLedAction*. Si el resultado de buffer[2] no coincide con lo deseado, se envía y registra un error. Por último, se da el formato de la trama a enviar y se almacena en una variable de buffer.
- SWITCH: Si el primer dato de la trama es un "1", entonces se llama a una función, que recibe como parámetro el switch deseado (buffer[1]), y devuelve un valor booleano de acuerdo a si el switch se encuentra o no pulsado. Luego se formatea la trama a enviar y se almacena en la variable.
- ACELERÓMETRO: Si el primer dato de la trama es un "2" y el segundo dato es un "1", se configura el acelerómetro mma8451 para que haga interrupciones por data ready. Cuando la bandera de la conversión finalice, se toma la medida de los tres ejes, se desactiva las interrupciones del acelerómetro y se formatea la trama a enviar según las aceleraciones obtenidas. En cualquier caso, que no se reciba lo deseado en la trama de entrada a la función, se registra un error.

La declaración de la función es la siguiente:

```
void tramaProcess(char *buf, int length)
```

y la definición de la misma se encuentra en el anexo del documento.

## Configuración Transceiver UART/RS485

Según se detalla en el enunciado, la comunicación entre el microcontrolador y la PC, debe realizarse mediante el protocolo UART. Para la recepción de datos (del punto de vista del microcontrolador) se debe implementar por cola circular (ring buffer) y la transmisión de datos debe hacerse mediante DMA (Direct Memory Access).

La primera versión del programa fue implementada en UART0, debido a que la misma se encuentra conectada directamente al OpenSDA y no se requiere conexionado externo. La configuración de UART0 es más sencilla y nos sirvió para poder corroborar el correcto funcionamiento de las MEF implementadas. Una vez que la transmisión y recepción de trama en ambos modos de funcionamientos era la deseada, se procedió a hacer la comunicación por UART1, con un conversor de UART/RS485 y otro de RS485/USB. En este punto se realizó una nueva configuración y es la que veremos a continuación.

Vamos a dividir la configuración en 4 partes: Inicialización, recepción de datos, envío de datos y el handler de la interrupción. Luego dentro de cada una se llaman a otras funciones y estructuras de datos que se explican brevemente a continuación:

- **`void 7ransceiver_init(void)`**

  En la inicialización del transceiver UART/RS485 se inicializa un ring buffer, los pines de RX y TX para UART1 con su alternativa de mux y se crea una configuración de UART con baud rate en 9600 bps, sin paridad y un bit de stop, además de habilitarse Tx y Rx. Además se habilitan las interrupciones necesarias del protocolo y se hacen las inicializaciones del DMA eligiendo el canal 0 del mismo para la transmisión de datos por UART.
  Se debe resaltar que la inicialización de los pines RE y DE no se realiza en esta función, sino que se inicializa junto con todos los GPIO en la función board_init().

- **`int32_t uart_ringBuffer_recDatos(uint8_t *pBuf, int32_t size)`**

  Esta función se llama cada vez que se solicita leer datos del ring buffer. Dentro de la misma, se deshabilitan todas las interrupciones mientras se cargan los datos del ring buffer en un buffer definido donde se llama la función. La lógica de la función es sencilla, mientras el ring buffer de recepción no esté vacio, se obtiene el dato del índice y se mueve el índice a la próxima posición.

- **`int32_t uart_drv_envDatos_DMA(uint8_t *pBuf, int32_t size)`**

  A la función para envio de datos por UART mediante DMA se le pasan dos parámetros, el puntero al buffer de transmisión y el tamaño (size) del mismo. La función crea una variable de transferencia lpuart y luego verifica si hay una transacción en curso (txOnGoin). Si esto último es verdadero, define el size de la transferencia en 0 para que no se realice. En caso de no haber una transmisión en

curso, limita el size de la transmisión en caso de ser necesario, habilita las líneas de control de RS485 para transmisión y hace una copia del buffer enviado como parámetro en un buffer llamada "txBuffer_dma" por seguridad. Luego, carga los datos de la transferencia con la copia del buffer y el tamaño del mismo y coloca la transmisión "txOnGoin" en verdadero. Por último, envía el dato por DMA y activa las interrupciones de transmisión completa para saber cuando la misma ha terminado.

- `void LPUART1_IRQHandler(void)`

  El handler de la interrupción de UART chequea las banderas de interrupción. En primer lugar, chequea la interrupción por recepción con la bandera *RxDataRegFullFlag* y de ser verdadera, lee el dato disponible en UART1 y lo coloca en el ring buffer antes de limpiar la bandera de interrupción.
  La segunda bandera que chequea es *TransmissionCompleteFlag*, que indica que la transmisión por DMA ha sido finalizada. Cuando esto sucede, deshabilita dicha interrupción (que volverá a ser activada en la función para envío de datos), limpia la bandera y habilita las líneas de control de RS485 para recepción.
  Por último, el handler llama a una función *"checkUartErrors()",* que chequea las tres banderas de error posibles y limpia las banderas en caso de ser verdaderas. Luego muetra el error por pantalla y lo registra.

## 5. EQUIPAMIENTO UTILIZADO

- Placa de desarrollo FRDM-KL43Z
- Pantalla OLED 128x64 SSD1306 SPI
- Conversor RS485/UART
- Conversor UART/USB

## 6. RESULTADOS OBTENIDOS

Muchas de las funciones, como las funciones para la utilización del acelerómetro MMA8451 o el display OLED, fueron reutilizadas del trabajo práctico anterior, simplificando la solución de esa parte del trabajo. En cuanto a la comunicación, se desarrolló en un principio para UART0, de forma que no se utilizaron los conversores ya que se conectó directamente la placa de desarrollo a la PC mediante Open SDA.

Una vez que el programa funcionaba correctamente con UART0, fue trasladada la implementación para UART1, siendo esto quizás uno de los principales desafíos del trabajo.

El desarrollo de la aplicación propiamente dicha fue planteado con MEF jerárquica y no presentó mayores problemas, de igual manera, fue de las primeras cosas chequeadas con mucho detalle para poder abstraerse de esto cuando se pasaba de uart0 a uart1.

# 7. CONCLUSIONES

El trabajo presentó un desafío para el equipo y una posibilidad de explorar nuevos protocolos de comunicación. Nos pareció interesante la posibilidad de comunicarse con otro dispositivo (PC) y además trabajar con un modelo de trama similar a MODBUS ASCII utilizado en la industria.

El principal problema se presentó con el software Master Series proporcionado por la cátedra. Al conectar el convertidor UART/USB, la PC con Windows 11 reconocía el dispositivo y contaba con los controladores apropiados. Sin embargo, aunque el software también identificaba el puerto, no lograba establecer conexión. En contraste, al utilizar el software Processing con la misma configuración y puerto, no se presentaba este inconveniente. Finalmente, la solución fue emplear una PC con Windows 10, ya que no pudimos resolver el problema en Windows 11.

# 8. BIBLIOGRAFÍA

[1] Gunther Gridling, Bettina Weiss, “Introduction to Microcontrollers”, Vienna University of Technology, Institute of Computer Engineering, Embedded Computing Systems Group February 26, 2007, Version 1.4.

[2] NXP Semiconductors, “KL43 Sub-Family Reference Manual”, Document Number: KL43P64M48SF6RM, July 2016, Rev. 5.1.

[3] NXP Semiconductors “MCUXpresso IDE User Guide”, 26 October, 2023, Rev. 11.8.0.

[4] Freescale Semiconductor. Application Note: “Motion and Freefall Detection Using the MMA8451, 2, 3Q”. Document Number: AN4070 Rev 1, 10/2011.

# 9. ANEXO

## Código

Main.c

```
int main(void) {

    /* Inicialización de clocks a máxima frecuencia y micro en modo RUN a 48MHz */
    init_clocks_and_power_mode();
    /* Inicialización FSL debug console. */
    BOARD_InitDebugConsole();
    /* Inicialización de GPIOS (LED, SW, OLED, RS485 */
    board_init();
    /* Inicialización de SPI y display OLED */
    board_configSPI0();
    oled_init();
    oled_setContrast(16);
    /* Inicialización del I2C */
    SD2_I2C_init();
    /* Inicialización MEF de switches*/
    key_init();
    /* Se inicializa UART1, DMA y RingBufferRx */
    transceiver_init();
    /* Se configura interrupción de systick cada 1 ms */
    SysTick_Config(SystemCoreClock / 1000U);
    /* INIT de la APP */
    mef_principal_init();

    while(1) {
      mef_principal();
    }
    return 0;
}

void SysTick_Handler(void){
    key_periodicTask1ms();
    mef_principal_task1ms();
}
```

Mef_principal_init()

```
void mef_principal_init(){
      estado_mef_principal = Est_Master;
      timer = UART_TRANSMISSION_DELAY;
      oled_clearScreen(OLED_COLOR_BLACK);
      oled_putString(30, 29, (uint8_t*)"Estado Master" ,
OLED_COLOR_WHITE, OLED_COLOR_BLACK);
}
```

Mef_principal()

```
void mef_principal(){
	switch(estado_mef_principal){
		case Est_Master:
			if (board_rs485_isDataAvailable()){
				mefRecTrama_task();
			}

			/* Transición a ESTADO 3D */
			if(key_getPressEv(BOARD_SW_ID_1)){
				estado_mef_principal = Est_3D;
				oled_clearScreen(OLED_COLOR_BLACK);
				oled_putString(35, 29, (uint8_t*)"Estado 3D"
, OLED_COLOR_WHITE, OLED_COLOR_BLACK);
				mma8451_dataReady_config();
			}
			break;

		case Est_3D:
			uint8_t buffer_modo_3d[32];
			if(IS_DATA_MMA8451_READY){
				mma8451_acel_reading.x = mma8451_getAcX();
				mma8451_acel_reading.y = mma8451_getAcY();
				mma8451_acel_reading.z = mma8451_getAcZ();
			}
			if(timer <= 0) // Para no sobrecargar el puerto UART
			{
				DEBUG_PRINT("Eje X: %d | ", mma8451_acel_reading.x);
				DEBUG_PRINT("Eje Y: %d | ", mma8451_acel_reading.y);
				DEBUG_PRINT("Eje Z: %d\n", mma8451_acel_reading.z);

				timer = UART_TRANSMISSION_DELAY;

				/* Se formatea el buffer y se envia por uart via DMA: */
				snprintf((char*)buffer_modo_3d,
sizeof(buffer_modo_3d), "%d %d %d\n",  mma8451_acel_reading.x,
mma8451_acel_reading.y, mma8451_acel_reading.z);
				uart_drv_envDatos_DMA(buffer_modo_3d,
strlen((char*)buffer_modo_3d));
			}

			/* Transición a ESTADO MASTER */
			if(key_getPressEv(BOARD_SW_ID_1)) {
				mma8451_clearInterruptions_config();
				estado_mef_principal = Est_Master;
				oled_clearScreen(OLED_COLOR_BLACK);
				oled_putString(30, 29, (uint8_t*)"Estado
Master" , OLED_COLOR_WHITE, OLED_COLOR_BLACK);
			}
	}
}
```

Mef_principal_task1ms()

```
void mef_principal_task1ms(){
	if(timer && estado_mef_principal == Est_3D) timer--;
}
```

mefRecTrama_task()

```
void mefRecTrama_task(void){
	static mefRecTrama_estado_enum estado = MEF_EST_ESPERANDO_INICIO;
	uint32_t flagRec;
	uint8_t byteRec;
	static uint8_t indexRec;  /* Indice del buffer */
	flagRec = uart_ringBuffer_recDatos(&byteRec, sizeof(byteRec));

	switch (estado){

		case MEF_EST_ESPERANDO_INICIO:

			if (flagRec != 0 && byteRec == ':'){
				estado = MEF_EST_ESPERANDO_GRUPO_1;
			}
			break;
		case MEF_EST_ESPERANDO_GRUPO_1:

			if (flagRec != 0 && byteRec == NUM_GRUPO_A){
				estado = MEF_EST_ESPERANDO_GRUPO_2;
			}
			if (flagRec != 0 && byteRec == ':'){
				estado = MEF_EST_ESPERANDO_GRUPO_1;
			}
			else if (flagRec != 0 && byteRec != NUM_GRUPO_A){
				estado = MEF_EST_ESPERANDO_INICIO;
			}
			break;
		case MEF_EST_ESPERANDO_GRUPO_2:

			if (flagRec != 0 && byteRec == NUM_GRUPO_B){
				estado = MEF_EST_RECIBIENDO_TRAMA;
				indexRec = 0;
			}
			if (flagRec != 0 && byteRec == ':'){
				estado = MEF_EST_ESPERANDO_GRUPO_1;
			}
			else if (flagRec != 0 && byteRec != NUM_GRUPO_B){
				estado = MEF_EST_ESPERANDO_INICIO;
			}
			break;
		case MEF_EST_RECIBIENDO_TRAMA:

			if (flagRec != 0 && byteRec != CHAR_LF){
				if (indexRec < BUFFER_SIZE){
					bufferRec[indexRec] = byteRec;
					indexRec++;
				}
			}
			if (flagRec != 0 && byteRec == ':'){
				indexRec = 0;
				estado = MEF_EST_ESPERANDO_GRUPO_1;
			}
			if (flagRec != 0 && byteRec == CHAR_LF){
				tramaProcess(bufferRec, indexRec);
				estado = MEF_EST_ESPERANDO_INICIO;
			}
			if (indexRec >= BUFFER_SIZE){
				estado = MEF_EST_ESPERANDO_INICIO;
			}
			break;
	}
}
```

TramaProcess()

```
void tramaProcess(char *buf, int length)
{
	ASSERT(length >= 3);
	DEBUG_PRINT("Input buffer: %s\n", buf);
	uint8_t buffer[BUFFER_SIZE];
	bool swPressed;
	buffer[0]='\0';
	wrongTrama = false;

	switch (buf[0]){
		case '0': // Caso de los leds
			switch (buf[2]){
				case 'A':
				  handleLedAction(buf[1], BOARD_LED_MSG_OFF);
				  break;
				case 'E':
				 handleLedAction(buf[1], BOARD_LED_MSG_ON);
				  break;
				case 'T':
				  handleLedAction(buf[1], BOARD_LED_MSG_TOGGLE);
				  break;
				default:
				  setErrorAndLog("Trama incorrecta.");
			break;
			}

			snprintf((char*)buffer, sizeof(buffer), ":%c%c0%c%c\n",
NUM_GRUPO_A, NUM_GRUPO_B, buf[1], buf[2]);
			break;

		case '1': // Caso de los switchs
			swPressed = isSwitchPressed(buf[1]);
			snprintf((char*)buffer, sizeof(buffer), ":%c%c1%c%c\n",
NUM_GRUPO_A, NUM_GRUPO_B, buf[1], (swPressed ? 'P' : 'N'));
			break;

		case '2': // Caso de los acelerometro
			if (buf[1] == '1'){
				mma8451_dataReady_config();
				while(!mma8451_getDataReadyInterruptStatus()){
					__NOP();
				}
				int16_t x = mma8451_getAcX();
				int16_t y = mma8451_getAcY();
				int16_t z = mma8451_getAcZ();
				snprintf((char*)buffer, sizeof(buffer),
":%c%c21%+04d%+04d%+04d\n", NUM_GRUPO_A, NUM_GRUPO_B, x, y, z);
				mma8451_clearInterruptions_config();
			} else {
		    setErrorAndLog("Trama incorrecta para lectura de acc");
		  }
		 break;
		  default:
			setErrorAndLog("Trama incorrecta. buf[0] error");
			break;
	}

	DEBUG_PRINT("Output buffer: %s\n", buffer);

	/* Envia datos por UART mediante DMA */
	uart_drv_envDatos_DMA(buffer, strlen((char*)buffer));
}
```

setErrorAndLog()

```
void setErrorAndLog(const char *errorMessage) {
    wrongTrama = true;
    DEBUG_PRINT("Error: %s\n",errorMessage);
    LOG_ERROR(errorMessage);
}
```

Debug.h

```
#ifndef DEBUG_H
#define DEBUG_H

#include "fsl_debug_console.h"
#include <assert.h>

#ifdef DEBUG
    #define DEBUG_PRINT(...) PRINTF(__VA_ARGS__)
    #define ASSERT(condition) assert(condition)
      #define LOG_ERROR(message) do { \
            FILE *error_log_file = fopen("error.log", "a"); \
            if (error_log_file != NULL) { \
                  fprintf(error_log_file, "[ERROR] File: %s, Line: %d:
%s\n", __FILE__, __LINE__, message); \
                  fclose(error_log_file); \
            } else { \
                  DEBUG_PRINT("[ERROR] Failed to open error log file.\n");
\
                  DEBUG_PRINT("[ERROR] File: %s, Line: %d: %s\n",
__FILE__, __LINE__, message); \
            } \
      } while(0)
#else
    #define DEBUG_PRINT(...) (void)0
    #define ASSERT(condition) (void)0
    #define LOG_ERROR(message) (void)0
#endif

#endif // DEBUG_H
```

LPUART_UserCallBack()

```
static void LPUART_UserCallback(LPUART_Type *base,
lpuart_dma_handle_t *handle, status_t status, void *userData)
{
    if (kStatus_LPUART_TxIdle == status)
    {
        txOnGoing = false;
    }
}
```

Transceiver_init()

```
void transceiver_init(void){

	lpuart_config_t lpuart_config;
	txOnGoing = false;
	 pRingBufferRx = ringBuffer_init(RX_RING_BUFFER_SIZE);

	CLOCK_SetLpuart1Clock(0x1U);
	PORT_SetPinMux(PORTE, 1U, kPORT_MuxAlt3);	/* RX: PORTE PIN 1 */
	PORT_SetPinMux(PORTE, 0U, kPORT_MuxAlt3);	/* TX: PORTE PIN 0 */

	LPUART_GetDefaultConfig(&lpuart_config);

	lpuart_config.baudRate_Bps = TR_UART_BAUD_RATE;	// 9600
	lpuart_config.parityMode = kLPUART_ParityDisabled;
	lpuart_config.stopBitCount = kLPUART_OneStopBit;
	lpuart_config.enableTx = true;
	lpuart_config.enableRx = true;

	LPUART_Init(TR_UART, &lpuart_config,
	CLOCK_GetFreq(kCLOCK_CoreSysClk));

	dataAvailable = false;

	/* Habilitación de interrupciones */
	LPUART_EnableInterrupts(TR_UART,kLPUART_RxDataRegFullInterruptEnable);
	LPUART_EnableInterrupts(TR_UART,kLPUART_TransmissionCompleteInterruptE
nable);
	LPUART_EnableInterrupts(TR_UART, kLPUART_RxOverrunInterruptEnable);
	LPUART_EnableInterrupts(TR_UART, kLPUART_FramingErrorInterruptEnable);
	LPUART_EnableInterrupts(TR_UART, kLPUART_ParityErrorInterruptEnable);

	EnableIRQ(RS485_UART_IRQn);

	/* Init DMAMUX */
	DMAMUX_Init(DMAMUX0);

	/* Set channel for LPUART  */
	DMAMUX_SetSource(DMAMUX0, LPUART_TX_DMA_CHANNEL, DMA_REQUEST_SRC);
	DMAMUX_EnableChannel(DMAMUX0, LPUART_TX_DMA_CHANNEL);

	/* Init the DMA module */
	DMA_Init(DMA0);
	DMA_CreateHandle(&LPUARTTxDmaHandle, DMA0, LPUART_TX_DMA_CHANNEL);

	/* Create LPUART DMA handle. */
	LPUART_TransferCreateHandleDMA(
			TR_UART,
			&LPUARTDmaHandle,
			LPUART_UserCallback,
			NULL,
			&LPUARTTxDmaHandle,
			NULL);

}
```

checkUartErrors()

```
void checkUartErrors(void) {
    if ((kLPUART_RxOverrunFlag) & LPUART_GetStatusFlags(TR_UART) &&
    (kLPUART_RxOverrunInterruptEnable)&LPUART_GetEnabledInterrupts(TR_UART))
    {
        LPUART_ClearStatusFlags(TR_UART, kLPUART_RxOverrunFlag);
        DEBUG_PRINT("UART Rx Overrun Error\n");

    }
    if ((kLPUART_FramingErrorFlag) & LPUART_GetStatusFlags(TR_UART) &&
 (kLPUART_FramingErrorInterruptEnable)&LPUART_GetEnabledInterrupts(TR_UART))
    {
        LPUART_ClearStatusFlags(TR_UART, kLPUART_FramingErrorFlag);
        DEBUG_PRINT("UART Framing Error\n");
    }
    if ((kLPUART_ParityErrorFlag) & LPUART_GetStatusFlags(TR_UART) &&
 (kLPUART_ParityErrorInterruptEnable)&LPUART_GetEnabledInterrupts(TR_UART))
    {
        LPUART_ClearStatusFlags(TR_UART, kLPUART_ParityErrorFlag);
        DEBUG_PRINT("UART Parity Error\n");
    }
}
```

Uart_ringBuffer_recDatos()

```
int32_t uart_ringBuffer_recDatos(uint8_t *pBuf, int32_t size)
{
    int32_t ret = 0;
    dataAvailable = false;

    /* Entra sección de código crítico */
    __disable_irq();

    while (!ringBuffer_isEmpty(pRingBufferRx) && ret < size)
    {
        ringBuffer_getData(pRingBufferRx, &pBuf[ret]);
        ret++;
    }

    /* Sale de sección de código crítico*/
    __enable_irq();

    return ret;
}
```

Uart_drv_envDatos_DMA()

```
int32_t uart_drv_envDatos_DMA(uint8_t *pBuf, int32_t size)
{
    lpuart_transfer_t xfer;

    if (txOnGoing || size == 0 )
    {
        size = 0;
    }
    else
    {
        /* limita size */
        if (size > TX_BUFFER_DMA_SIZE)
            size = TX_BUFFER_DMA_SIZE;

        /* Habilitar la transmisión antes de enviar los datos */
        rs485_RE(true);
             rs485_DE(true);

        /* Copio los datos a otro buff para no depender del original */
             memcpy(txBuffer_dma, pBuf, size);

             /* Configuraciones de la transferencia */
             xfer.data = txBuffer_dma;
             xfer.dataSize = size;

             /* Defino el estado de transferencia en proceso */
             txOnGoing = true;

             /* Realizo transferencia por DMA y habilito interrupción */
        LPUART_TransferSendDMA(TR_UART, &LPUARTDmaHandle, &xfer);
        LPUART_EnableInterrupts(TR_UART,
kLPUART_TransmissionCompleteInterruptEnable);
    }

    return size;
}
```

LPUART1_IRQHandler()

```
void LPUART1_IRQHandler(void){

	uint8_t data; /* variable para guardar dato de lectura de Rx */

	if ( (kLPUART_RxDataRegFullFlag) & LPUART_GetStatusFlags(TR_UART) &&
(kLPUART_RxDataRegFullInterruptEnable)& LPUART_GetEnabledInterrupts(TR_UART) )
	{
		/* obtiene dato recibido por puerto serie */
		data = LPUART_ReadByte(TR_UART);
		dataAvailable = true;
		/* pone dato en ring buffer */
		ringBuffer_putData(pRingBufferRx, data);

		LPUART_ClearStatusFlags(TR_UART,kLPUART_RxDataRegFullFlag);
	}

	if ( (kLPUART_TransmissionCompleteFlag)&LPUART_GetStatusFlags(TR_UART)
&&(kLPUART_TransmissionCompleteInterruptEnable)&LPUART_GetEnabledInterrupts(
TR_UART))
	{
	LPUART_DisableInterrupts(TR_UART,kLPUART_TransmissionCompleteInterrupt
Enable);
	LPUART_ClearStatusFlags(TR_UART, kLPUART_TransmissionCompleteFlag);

		/* Habilito recepción para esperar el dato de vuelta */
		rs485_RE(false);
		rs485_DE(false);
	}

	/* Chequeo de errores en UART leyendo banderas */
	checkUartErrors();
}
```