

Trabajo de Grado – Bitácora

Título: “Desarrollo e implementación del módulo de control y diagnóstico de la remota para la supervisión y monitoreo de pozos y casetas de válvulas en poliductos en PDVSA”.

Este documento describe las actividades realizadas en el desarrollo del trabajo de grado, así como los sub-proyectos creados para la programación de las diferentes tareas y funcionalidades.

Fecha de inicio: 24-06-2023

Sub-proyecto: SPI Communication

Trabajo realizado antes del 19-07-2023:

- ✓ Programas de prueba para la comunicación SPI entre un ESP32S3 y un ESP32, configurando el ESP32S3 como maestro y el ESP32 como esclavo.
- ✓ Programa para la comunicación SPI con el módulo I/O en el ESP32S3.
- ✓ Programa para la comunicación SPI con el módulo I/O en el ESP32.
- ✓ Creada estructura de memoria asignada dinámicamente para el almacenamiento de:
 - Variables analógicas de entrada: anTbl[0][j]. (tamaño: 16 registros de 16bits)
 - Variables digitales de entrada: digTbl[0][j]. (tamaño: 2 registros de 16bits)
 - Variables analógicas de salida: anTbl[1][j] (tamaño: 16 registros de 16bits)
 - Variables digitales de salida: digTbl[1][j] (tamaño: 2 registros de 16bits)
 - Registros de configuración: configTbl[0][j] (tamaño: 50 registros de 16bits)
 - Registros auxiliares: auxTBL[0][j] y auxTBL[1][j] (tamaño: 50 registros de 16bits)
- ✓ Creada función para inicialización de tablas en memoria: (Maestro y Esclavo)

```
esp_err_t tablesInit(varTables_t *tables,
                    uint8_t numAnTbIs,    //Tablas de variables analógicas
                    uint8_t numDigTbIs,   //Tablas de variables digitales
                    uint8_t numConfigTbIs, //Tablas de configuración
                    uint8_t numAuxTbIs,    //Tablas auxiliares
                    uint8_t anSize,        //Tamaño de tablas analógicas
                    uint8_t digSize,       //Tamaño de tablas digitales
                    uint8_t configSize,    //Tamaño de tablas de configuración
                    uint8_t auxSize);      //Tamaño de tablas auxiliares

esp_err_t tablePrint(uint16_t *table, uint8_t size);
esp_err_t tablesUnload(varTables_t *tables);
```

- ✓ Creado protocolo para envío de comandos por SPI, para el reconocimiento de las diferentes peticiones enviadas al esclavo.
- ✓ Creadas funciones para envío y recepción de datos por SPI en half dúplex y full dúplex, las funciones fueron integradas en la biblioteca SPI_IO_Master.h en el ESP32S3 y en SPI_IO_Slave.h en el ESP32.
- ✓ Creadas funciones para envío y recepción de datos, tanto tablas enteras como datos individuales de una tabla: (Maestro)

```

esp_err_t readAnalogTable(varTables_t *Tables, uint8_t tbl);
esp_err_t readDigitalTable(varTables_t *Tables, uint8_t tbl);
esp_err_t readConfigTable(varTables_t *Tables, uint8_t tbl);
esp_err_t readAuxTable(varTables_t *Tables, uint8_t tbl);
esp_err_t readAllTables(varTables_t *Tables);
esp_err_t readAnalogData(varTables_t *Tables, uint8_t tbl, uint8_t dataIndex);
esp_err_t readDigitalData(varTables_t *Tables, uint8_t tbl, uint8_t dataIndex);
esp_err_t readConfigData(varTables_t *Tables, uint8_t tbl, uint8_t dataIndex);
esp_err_t readAuxData(varTables_t *Tables, uint8_t tbl, uint8_t dataIndex);
esp_err_t writeAnalogTable(varTables_t *Tables, uint8_t tbl);
esp_err_t writeDigitalTable(varTables_t *Tables, uint8_t tbl);
esp_err_t writeConfigTable(varTables_t *Tables, uint8_t tbl);
esp_err_t writeAuxTable(varTables_t *Tables, uint8_t tbl);
esp_err_t writeAnalogData(uint8_t tbl, uint8_t dataIndex, uint16_t payload);
esp_err_t writeDigitalData(uint8_t tbl, uint8_t dataIndex, uint16_t payload);
esp_err_t writeConfigData(uint8_t tbl, uint8_t dataIndex, uint16_t payload);
esp_err_t writeAuxData(uint8_t tbl, uint8_t dataIndex, uint16_t payload);

```

- ✓ Creado código en el esclavo para la recepción de los comandos correspondientes a las diferentes funciones; reconociendo la petición y respondiendo a cada una según su código en el protocolo creado.
- ✓ Incorporado CRC16 para calcular el checksum, haciendo uso de la biblioteca esp_crc.h
- ✓ Funcionalidad del checksum incorporada a las funciones de las bibliotecas SPI_IO_Master.h en el ESP32S3 y en SPI_IO_Slave.h en el ESP32. En cada transacción, antes de enviar datos se realiza el cálculo del checksum y se añade al buffer de envío; asimismo al recibir datos, se recupera el checksum recibido y se compara con el checksum calculado permitiendo la detección de errores. Las funciones de la biblioteca retornan ESP_OK o ESP_FAIL según sea el caso.
- ✓ Agregado manejador de error en las funciones que leen y escriben datos en las tablas, en caso de errores de CRC y en caso de que el comando enviado no haya sido reconocido por el esclavo.
- ✓ Agregada función exchangeData(), que permite enviar las tablas analógicas y digitales de salida al tiempo que se reciben las tablas analógicas y digitales de entrada en comunicación full dúplex.

Cambios realizados hasta el 19-07-2023:

- ✓ Agregadas macros para registros de tabla auxTbl[1] (Tabla auxiliar para registros propios del ESP32S3):

```

#define SPI_TRANSACTION_COUNT s3Tables.auxTbl[1][0]
#define SPI_ERROR_COUNT s3Tables.auxTbl[1][1]

```

- ✓ Agregado código en el maestro para probar la comunicación al inicializar y bloquear hasta que se reciba un eco de lo que se envió (código de prueba).
- ✓ Agregado caso en el esclavo para el código de prueba, que recibe el comando y responde con un eco de lo que recibió.
- ✓ Agregado código en función exchangeData() para contabilizar la transacción, contabilizar si hubo error y calcular el error ratio. Se usaron los registros aux[1][0] y aux[1][1], definidos en las macros.

- ✓ Agregado código de contabilización de errores en todas las funciones.
- ✓ El comando de prueba de la comunicación se usa ahora para recuperar la comunicación en caso de detectarse un error de CRC. Los errores generados por la pérdida de sincronización (en caso de reset) son contabilizados como uno solo al recuperar la comunicación.
- ✓ Agregada función para imprimir reporte de errores `print_spi_stats()`.
- ✓ Agregado conteo de errores en esclavo y reinicia el microcontrolador al exceder la cantidad de 10 errores seguidos.

NOTA: El código para el conteo de errores, así como el cálculo de la tasa de error, fue modificado en el proyecto **Remota_IO_Modbus_Slave**; a fin de solucionar el problema del overflow en el contador de 16 bits del número de transacciones. Fue optimizado para el uso de dos registros de la tabla auxiliar: `SPI_TRANSACTION_COUNT_L` y `SPI_TRANSACTION_COUNT_H`. Ver la sección de la bitácora correspondiente al proyecto **Remota_IO_Modbus_Slave** para más detalles. (05-08-2023).

Cambios realizados hasta el 25-07-2023:

- ✓ Incluida la línea que entrega el semáforo luego que se inicializa el spi con `init_spi()`. Lo cual solucionó muchos errores de sincronización.
- ✓ Cambiada función `spi_task()` para ejecutar únicamente `exchangeData()` en el bucle infinito.
- ✓ Agregadas las macros para medición de los tiempos de ciclo de ejecución y transferencia de variables de I/O:

```
#define SPI_EXCHANGE_TIME s3Tables.auxTbl[1][2]
#define SPI_CYCLE_TIME s3Tables.auxTbl[1][3]
```

- ✓ Incorporada la medición de tiempo del intercambio de datos y de ejecución entre ciclos dentro de la misma función `spi_task()`.
- ✓ Cambiado `vTaskDelay` esclavo por `taskYIELD()`.
- ✓ Modificada la prioridad de la tarea `spi_task()` a prioridad de nivel 1, lo cual aceleró enormemente la ejecución de la misma.
- ✓ Cambiado tiempo antirebote en la función de interrupción, de 100us a 50us; lo cual solucionó muchos problemas de tiempos excesivamente largos; logrando tiempos de transferencia y ciclos de ejecución del orden de 200us.
- ✓ Agregado semáforo `spiTaskSem` para controlar el acceso concurrente al puerto SPI. Al intentar tomar el semáforo, el código bloquea la tarea mediante una sentencia `while()` `continue`;
- ✓ Agregada rutina `spi_test()` a la biblioteca `SPI_IO`, que permite el envío repetido del comando de eco hasta recibir una respuesta válida del esclavo. Utilizada en `init_spi()` y en `exchangeData()`. Permite recuperarse de un error al producirse un reset en el esclavo o el maestro.

Sub-proyecto: Modbus-Scratch

Cambios realizados hasta el 16-07-2023:

- ✓ Creado proyecto Modbus scratch
- ✓ Incorporada la biblioteca `JRC_WiFi.h` para hacer pruebas de conectividad WiFi y acceso a la red LAN.

- ✓ Incorporado el componente mbcontroller.h para agregar la funcionalidad de protocolo Modbus.
- ✓ Definida la estructura de diccionario de datos o características de un esclavo Modbus, para realizar pruebas con programas de simulación de esclavos Modbus (Modsim32 y Modbus Slave).
- ✓ Creada rutina de inicialización de la pila Modbus TCP/IP en modo maestro (Cliente).
- ✓ Verificada la conectividad y funcionamiento de la pila Modbus TCP/IP al conectar esclavos.
- ✓ Pruebas satisfactorias utilizando el programa Modbus Slave, no obstante, se obtienen errores al escribir características en el esclavo utilizando Modsim32. Posiblemente el error se debe a que la trama Modbus se está efectuando como una trama Modbus RTU encapsulada en TCP/IP.

Sub-proyecto: Modbus-Scratch-Slave

Cambios realizados hasta el 16-07-2023:

- ✓ Creado proyecto Modbus scratch slave TCP/IP para implementar la comunicación vía Modbus esclavo (servidor).
- ✓ Incorporada la biblioteca JRC_WiFi.h para hacer pruebas de conectividad WiFi y acceso a la red LAN.
- ✓ Incorporado el componente mbcontroller.h para agregar la funcionalidad de protocolo Modbus.
- ✓ Definido el mapa de memoria con las áreas de registro correspondientes, definiendo un área de registros Holding de 100 registros de 16bits y un área de registros de entrada Input de 100 registros de 16 bits.
- ✓ Creada rutina de inicialización de la pila Modbus TCP/IP en modo esclavo (Servidor).
- ✓ Verificada la conectividad y funcionamiento de la pila Modbus TCP/IP al conectar maestros.
- ✓ Pruebas satisfactorias obtenidas tanto con el programa Modscan32 como con el programa Modbus Poll. Probando conexiones concurrentes de dos simuladores desde direcciones IP distintas en la misma LAN.

Sub-proyecto: Modbus-Slave-ENC28J60

Cambios realizados hasta el 27-07-2023:

- ✓ Documentación sobre los módulos ENC28J60 y W5500; advirtiendo las desventajas del ENC28J60 frente al W5500. Se sugiere la adquisición y utilización del W5500.
- ✓ Establecido el puerto SPI a utilizar, mediante los GPIO nativos del ESP32S3: (sin usar la matriz de GPIO):
 - CS 10
 - SCLK 12
 - MISO 13
 - MOSI 11
 - Interrupt 14
- ✓ Realizada la conexión del módulo ENC28J60 al ESP32S3, utilizando una fuente de alimentación externa de 3.3V (Regulador AMS1117-3.3V). Ensamblaje del circuito regulador en baquelita perforada, utilizando componentes SMD.

Cambios realizados hasta el 30-07-2023:

- ✓ Probado código de ejemplo de espressif para el ENC28J60, comprobando su funcionamiento y efectuando pruebas de ping a la dirección IP obtenida por el microcontrolador a través de DHCP. Las pruebas al inicio fueron poco satisfactorias dada la alta latencia obtenida al hacer ping.
- ✓ Creado proyecto Modbus-Slave-ENC28J60, para implementar la comunicación vía Modbus TCP/IP en modo esclavo (servidor), utilizando el adaptador de ethernet ENC28J60.
- ✓ Incorporado el componente esp_eth_enc28j60.h desarrollado por espressif y utilizado en el ejemplo.
- ✓ Creada la rutina de inicialización del módulo ethernet y la pila TCP/IP, se obtiene la negociación de la IP automáticamente vía DHCP.
- ✓ Verificado el código necesario para establecer una conexión punto a punto con configuración de IP estática (manual). Ambas formas de conexión se incluyeron en el código. Para IP estática, quitar los comentarios de las líneas correspondientes.
- ✓ Incorporada la rutina de inicialización del protocolo Modbus TCP/IP modo esclavo (servidor) del proyecto Modbus-Scratch-Slave. Se configuró para utilizar la interfaz de red proporcionada por el ENC28J60 en lugar de la interfaz WiFi anterior.
- ✓ Definido mapa de memoria igual al del proyecto Modbus-Scratch-Slave, para efectuar pruebas.
- ✓ Verificada la conectividad y funcionamiento de la pila Modbus TCP/IP al conectar maestros.
- ✓ Las pruebas al inicio fueron poco satisfactorias dada la alta latencia y errores de timeout en la conexión y las solicitudes, además de reinicios aleatorios del microcontrolador.
- ✓ Solucionado problema de alta latencia al identificar un error en la conexión del pin de interrupción del ENC28J60. Lo cual solventó los problemas de latencia, llegando a tener tiempos del orden de 1ms en las pruebas de ping.
- ✓ Pruebas satisfactorias obtenidas tanto con el programa Modscan32 como con el programa Modbus Poll. Probando conexiones concurrentes de dos simuladores desde direcciones IP distintas en la misma LAN.

Cambios realizados hasta el 04-08-2023:

Sub-proyecto: Remota IO Modbus Slave - (Primer avance de integración)

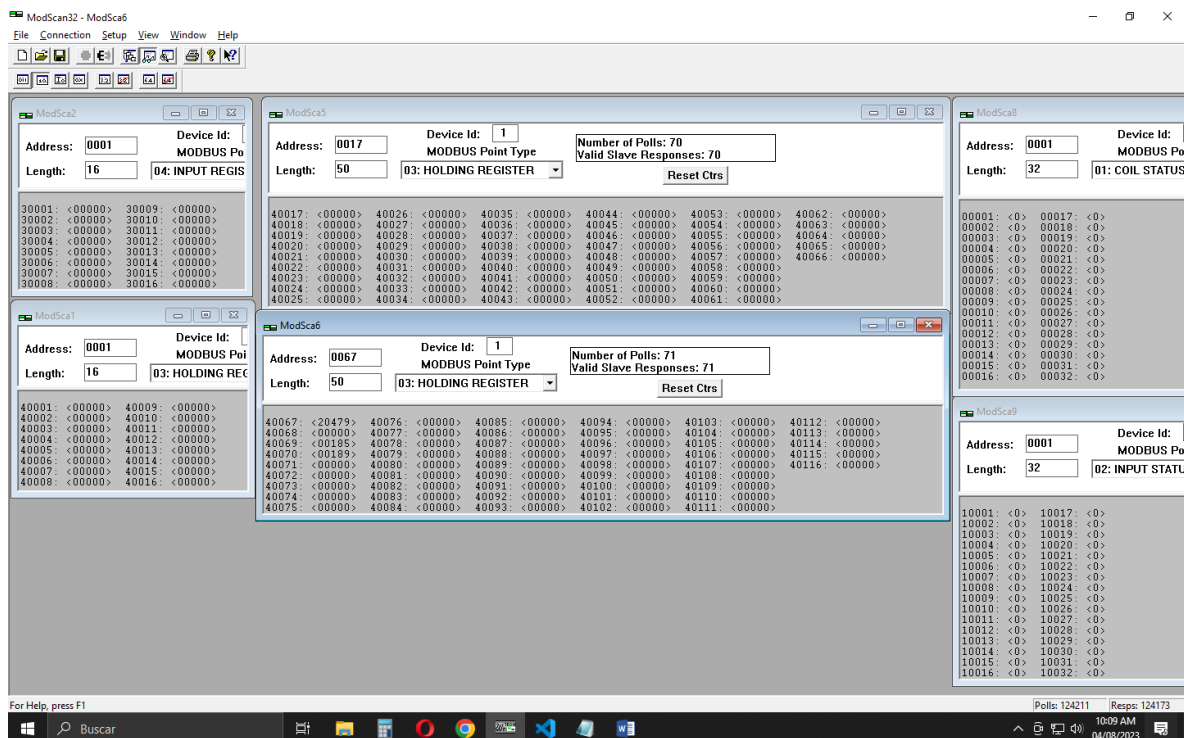
- ✓ Creado proyecto Remota_IO_Modbus_Slave, para integrar la comunicación SPI con el módulo I/O, la conectividad de red a través del módulo ethernet ENC28J60 y la implementación de la pila Modbus TCP/IP en modo esclavo (servidor).
- ✓ Agregadas las bibliotecas necesarias para ambos proyectos (SPI_Communication y Modbus_Slave_ENC28J60) en el nuevo proyecto.
- ✓ Agregados los códigos de ambos proyectos al programa main.c.
- ✓ Asignación del SPI_HOST:
 - `#define IO_SPI_HOST SPI2_HOST //En SPI_IO_Master.h`
 - `#define ENC28J60_SPI_HOST SPI3_HOST //En main.c`
- ✓ Creación del mapa de memoria para el esclavo: Fueron asignados los registros de las tablas de memoria según los siguientes criterios:

- INPUT REGISTERS:
 - Tabla analógica de entrada s3Tables.anTbl[0]
 - Direcciones Modbus: 30001 a 30016 (PLC Style addresses) (de 0 a 15 en notación zero based).
 - Offset: 0
 - Apunta a: s3Tables.anTbl[0][0]
 - Tamaño: s3Tables.anSize * 2 (o s3Tables.anSize << 1 en C)
- INPUT STATUS (o DISCRETE INPUTS):
 - Tabla digital de entrada s3Tables.digTbl[0]
 - Direcciones Modbus:
 - 10001 a 10016 para el registro de 16 bits s3Tables.digTbl[0][0]
 - 10017 a 10032 para el registro de 16 bits s3Tables.digTbl[0][1]
 - Offset: 0
 - Apunta a: s3Tables.digTbl[0][0]
 - Tamaño: 4 bytes (2 bytes para cada registro)
- COIL STATUS (o DISCRETE OUTPUTS):
 - Tabla digital de salida s3Tables.digTbl[1]
 - Direcciones Modbus:
 - 00001 a 00016 para el registro de 16 bits s3Tables.digTbl[1][0]
 - 00017 a 00032 para el registro de 16 bits s3Tables.digTbl[1][1]
 - Offset: 0
 - Apunta a: s3Tables.digTbl[1][0]
 - Tamaño: 4 bytes (2 bytes para cada registro)
- HOLDING REGISTERS
 - Tabla analógica de salida s3Tables.anTbl[1]
 - Direcciones Modbus:
 - 40001 a 40016
 - Offset: 0
 - Apunta a: s3Tables.anTbl[1][0]
 - Tamaño: s3Tables.anSize * 2 (o s3Tables.anSize << 1 en C)
 - Tabla de configuración s3Tables.configTbl[0]
 - Direcciones Modbus:
 - 40019 a 40068
 - Offset: 16
 - Apunta a: s3Tables.configTbl[0][0]
 - Tamaño: s3Tables.configSize * 2 (o s3Tables.configSize << 1 en C)
 - Tabla auxiliar s3Tables.auxTbl[0]
 - Direcciones Modbus:
 - 40069 a 40118
 - Offset: 66
 - Apunta a: s3Tables.auxTbl[0][0]

- Tamaño: $s3Tables.auxSize * 2$ (o $s3Tables.auxSize < 1$ en C)

Nota: El mapa de memoria del esclavo está sujeto a la configuración actual referente al número de tablas de cada tipo (2 tablas analógicas de 16 registros, 2 tablas digitales de 2 registros, 1 tabla de configuración de 50 registros y 1 tabla auxiliar de 50 registros). (Hard-coded Style). Se debe realizar la configuración del mapa de memoria nuevamente si el número de tablas cambia, para adaptarlo a la nueva estructura de registros.

- ✓ Realizadas pruebas de rendimiento de SPI al añadir la funcionalidad del módulo Ethernet y la Pila Modbus TCP/IP; obteniendo tiempos de intercambio de variables del orden de 200us (sin cambios) y tiempos de ciclos en el mismo orden; salvo que cuando la tarea `app_main()` toma el semáforo para usar el puerto SPI, en ese caso el tiempo de ciclo se incrementa a 700 – 900 us, debido a que el sistema operativo tarda un poco más en volver a ejecutar la tarea `spl_task()` dado que también está atendiendo las tareas de TCP/IP y modbus. Las pruebas de rendimiento son aceptables.
- ✓ Realizadas las pruebas de funcionalidad de la pila Modbus TCP/IP, mediante el programa Modscan32, mostrando pruebas satisfactorias con tiempos de Delay between polls: 250ms y Slave response timeout 2000ms. La configuración para la prueba fue la siguiente:



- ✓ Modificado el programa del ESP32 WROOM (módulo I/O):
 - Creada tarea para gestionar la comunicación SPI con nivel de prioridad 1.
 - Creada tarea para medición del canal ADC 5 (conectado a un potenciómetro)
 - El valor leído del ADC es asignado a los registros de la tabla analógica de entrada.
 - Eliminados los contadores que modificaban los valores de los registros de las tablas.

- ✓ Verificado que el valor de medición de voltaje del potenciómetro se refleja en la salida de la consola del programa del ESP32S3 y en el Modscan32.

Cambios realizados hasta el 04-08-2023:

- ✓ Modificado el programa del ESP32 WROOM (módulo I/O):
 - Asignados los pines GPIO para conectar leds indicadores del estado de salidas digitales:

```
#define GPIO_LED_0 18
#define GPIO_LED_1 19
#define GPIO_LED_2 21
#define GPIO_LED_3 2
#define GPIO_LED_4 4
#define GPIO_LED_5 5
```
 - Configurados los pines GPIO anteriores como salidas y conectados 6 LEDS indicadores.
 - Creada tarea asociada a la función `update_outputs()`, la cual actualiza el estado de las salidas digitales correspondientes a los LEDS según el estado de los bits menos significativos de los registros de la tabla de salidas digitales, a fin de visualizar el cambio de las salidas al modificar el estado de los registros COIL correspondientes desde el simulador Modscan32.
 - Verificado el funcionamiento correcto al modificar los registros de salidas COIL desde el cliente Modbus remoto.
- ✓ Modificado el sistema de conteo de transacciones SPI, para evitar que el contador de error vuelva a cero al producirse el desborde en el contador de transacciones:
 - Empleado registro adicional de la tabla `auxTbl[0]`, a fin de utilizar dos registros de 16 bits para el conteo de transacciones, quedando de la siguiente manera:

```
#define SPI_TRANSACTION_COUNT_L s3Tables.auxTbl[0][0]
#define SPI_TRANSACTION_COUNT_H s3Tables.auxTbl[0][1]
#define SPI_ERROR_COUNT s3Tables.auxTbl[0][2]
```
 - Creada la función `spi_transaction_counter()` para efectuar la cuenta de transacciones, teniendo en cuenta el desborde del registro `SPI_TRANSACTION_COUNT_L` y utilizando `SPI_TRANSACTION_COUNT_H` para ampliar a 32 bits la capacidad del contador.

Cambios realizados hasta el 06-08-2023:

- ✓ Creado nuevo proyecto **Remota IO Module ESP32**, para contener el programa de prueba del módulo IO (ESP32 WROOM) con los cambios más recientes implementados. Este proyecto está diseñado para ser utilizado en conjunto con el proyecto **Remota IO Modbus Slave**. Ambos proyectos se encuentran en el repositorio github creado para el trabajo de grado: (<https://github.com/jrctechULA?tab=repositories>)

Cambios realizados hasta el 09-08-2023:

- ✓ Agregado código para manejar los niveles de log en consola, mediante las funciones de la biblioteca `esp_log.h`. Los mensajes de error, warning, info, debug y verbose, ahora se muestran de acuerdo al valor de configuración correspondiente al último registro de la tabla de

configuración: s3Tables.configTbl[0][49]. Ese valor se puede cambiar desde el simulador ModScan32 y el comportamiento de la consola cambia de acuerdo al valor establecido.

Sub-proyecto: Remota Integration - (Segundo avance de integración)

- ✓ Creado proyecto **Remota Integration** para contener la integración de todo el programa, con los siguientes cambios:
 - Creado el archivo comm_services.c, dentro de la carpeta main del proyecto; a fin de contener todo el código concerniente a la iniciación de los servicios de comunicaciones.
 - Movido el código que inicia la interfaz TCP/IP a través del módulo ENC28J60 al archivo comm_services.c. Tanto las funciones necesarias (inicializador, manejadores de eventos) como las macros correspondientes, están ahora en dicho archivo.
 - Agregada la macro para definir el registro usado para establecer el nivel de log en la consola serial:

```
▪ #define CFG_REMOTA_LOG_LEVEL s3Tables.configTbl[0][49]
```

Cambios realizados hasta el 12-08-2023:

- ✓ Agregado soporte para el módulo Ethernet W5500; para utilizar dicho módulo, se dispuso de la siguiente macro:

```
//Comment out this line to use ENC28J60 Ethernet Module

#define ETHERNET_USE_W5500

// Important Note:
// Enable support for W5500 via menu-config:
// Activate "Use W5500 (MAC RAW)" option along with
// "Support SPI to Ethernet Module" in the Ethernet section!
```

- ✓ Si se comenta la definición de la macro ETHERNET_USE_W5500, el programa se compila con el código necesario para usar el módulo ENC28J60; de lo contrario se compilará con el código necesario para usar el módulo W5500.
- ✓ Agregada configuración para utilizar dirección IP estática o dirección IP dinámica (vía servidor DHCP), esto se implementó mediante las siguientes macros:

```
//Comment out this line to use dynamic IP, via DHCP Server:

//#define ETHERNET_USE_STATIC_IP

#ifndef ETHERNET_USE_STATIC_IP

#define ETHERNET_IP_ADDR      "192.168.1.20"
#define ETHERNET_GATEWAY     "192.168.1.10"
#define ETHERNET_SUBNET_MASK  "255.255.255.0"

#endif //ETHERNET_USE_STATIC_IP
```

- ✓ Al comentar la macro `ETHERNET_USE_STATIC_IP`, el programa se compilará con el código necesario para usar servidor DHCP; de lo contrario, se asignará una configuración de IP estática, basada en las macros `ETHERNET_IP_ADDR`, `ETHERNET_GATEWAY` y `ETHERNET_SUBNET_MASK`.

Cambios realizados hasta el 17-08-2023:

- ✓ **Incorporada la tarea de escalamiento de señales analógicas de entrada:**

```
void scaling_task(void *pvParameters){
    float y, m, x, b;

    while (1)
    {
        for (int i = 0; i < s3Tables.anSize; i++){
            m = s3Tables.scalingFactor[i];
            x = s3Tables.anTbl[0][i];
            b = s3Tables.scalingOffset[i];
            y = m * x + b;
            s3Tables.scaledValues[i] = y;
        }
        taskYIELD();
    }
}
```

- ✓ Esta tarea se colocó en el núcleo 0 con nivel de prioridad 0.
- ✓ Se modificó la estructura en la definición del tipo `varTables_t` para agregar las siguientes tablas adicionales. (Estas tablas son vectores de valores `float`):
 - `s3Tables.scalingFactor[]`
 - `s3Tables.scalingOffset[]`
 - `s3Tables.scaledValues[]`

```
typedef struct {
    uint16_t** anTbl;      // Vector de apuntadores a los vectores analógicos
    uint16_t** digTbl;     // Vector de apuntadores a los vectores Digitales
    uint16_t** configTbl;  // Vector de apuntadores a los vectores de configuración
    uint16_t** auxTbl;     // Vector de apuntadores a los vectores auxiliares
    float* scalingFactor;  //Vector de factores de escalamiento (pendiente m)
    float* scalingOffset;  //Vector de desplazamientos en la escala (corte con y -> b)
    float* scaledValues;   //Vector de apuntadores a los vectores de valores escalados
    uint8_t anSize;        // Tamaño de los vectores analógicos
    uint8_t digSize;       // Tamaño de los vectores analógicos
    uint8_t configSize;    // Tamaño de los vectores de configuración
    uint8_t auxSize;       // Tamaño de los vectores auxiliares
    uint8_t numAnTbIs;     // Número de vectores analógicos
    uint8_t numDigTbIs;    // Número de vectores digitales
}
```

```
uint8_t numConfigTbIs; // Número de vectores de configuración
uint8_t numAuxTbIs;    // Número de vectores auxiliares
} varTables_t;
```

- ✓ Se modificó la función `tablesInit()` para reservar el espacio necesario en memoria e inicializar los valores de dichas tablas. (Los factores de escalamiento se inicializan en 1 y los offsets en 0):

```
//Create scaling tables: (The size of these tables are the same as anSize)
//Scaling factors (m factors in y=mx+b)
tables->scalingFactor = (float*)malloc(tables->anSize * sizeof(float));
if (tables->scalingFactor == NULL){
    ESP_LOGE(TAG, "Error al asignar memoria!\n");
    return ESP_FAIL;
}
//Default value is 1 (no scaling)
for (size_t i = 0; i < tables->anSize; i++)
{
    tables->scalingFactor[i] = 1;
}

//Scaling offsets (b offset in y=mx+b)
tables->scalingOffset = (float*)malloc(tables->anSize * sizeof(float));
if (tables->scalingOffset == NULL){
    ESP_LOGE(TAG, "Error al asignar memoria!\n");
    return ESP_FAIL;
}
memset(tables->scalingOffset, 0, tables->anSize * 4); //Default value is 0 (no offset)

//Scaled values (Calculated values after scaling factors applied)
tables->scaledValues = (float*)malloc(tables->anSize * sizeof(float));
if (tables->scaledValues == NULL){
    ESP_LOGE(TAG, "Error al asignar memoria!\n");
    return ESP_FAIL;
}
memset(tables->scaledValues, 0, tables->anSize * 4);
```

- ✓ Se agregaron las tablas de escalamiento al mapa de memoria Modbus para el acceso al mismo desde el sistema SCADA, ahora el mapa modbus tiene la siguiente configuración:
 - INPUT REGISTERS:
 - Tabla analógica de entrada `s3Tables.anTbl[0]`
 - Direcciones Modbus: 30001 a 30016 (PLC Style addresses) (de 0 a 15 en notación zero based).
 - Offset: 0
 - Apunta a: `s3Tables.anTbl[0][0]`

- Tamaño: $s3Tables.anSize * 2$ (o $s3Tables.anSize < 1$ en C)

▪ Tabla de valores escalados `s3Tables.scaledValues`

- Direcciones Modbus:
 - 3017 a 3048
- Offset: 16
- Apunta a: `s3Tables.scalingFactor[0]`
- Tamaño: $s3Tables.anSize * 4$

○ INPUT STATUS (o DISCRETE INPUTS):

- Tabla digital de entrada `s3Tables.digTbl[0]`
 - Direcciones Modbus:
 - 10001 a 10016 para el registro de 16 bits `s3Tables.digTbl[0][0]`
 - 10017 a 10032 para el registro de 16 bits `s3Tables.digTbl[0][1]`
 - Offset: 0
 - Apunta a: `s3Tables.digTbl[0][0]`
 - Tamaño: 4 bytes (2 bytes para cada registro)

○ COIL STATUS (o DISCRETE OUTPUTS):

- Tabla digital de salida `s3Tables.digTbl[1]`
 - Direcciones Modbus:
 - 00001 a 00016 para el registro de 16 bits `s3Tables.digTbl[1][0]`
 - 00017 a 00032 para el registro de 16 bits `s3Tables.digTbl[1][1]`
 - Offset: 0
 - Apunta a: `s3Tables.digTbl[1][0]`
 - Tamaño: 4 bytes (2 bytes para cada registro)

○ HOLDING REGISTERS

- Tabla analógica de salida `s3Tables.anTbl[1]`
 - Direcciones Modbus:
 - 40001 a 40016
 - Offset: 0
 - Apunta a: `s3Tables.anTbl[1][0]`
 - Tamaño: $s3Tables.anSize * 2$ (o $s3Tables.anSize < 1$ en C)
- Tabla de configuración `s3Tables.configTbl[0]`
 - Direcciones Modbus:
 - 40017 a 40066
 - Offset: 16
 - Apunta a: `s3Tables.configTbl[0][0]`
 - Tamaño: $s3Tables.configSize * 2$ (o $s3Tables.configSize < 1$ en C)
- Tabla auxiliar `s3Tables.auxTbl[0]`
 - Direcciones Modbus:
 - 40067 a 40116
 - Offset: 66

- Apunta a: s3Tables.auxTbl[0][0]
- Tamaño: s3Tables.auxSize * 2 (o s3Tables.auxSize << 1 en C)

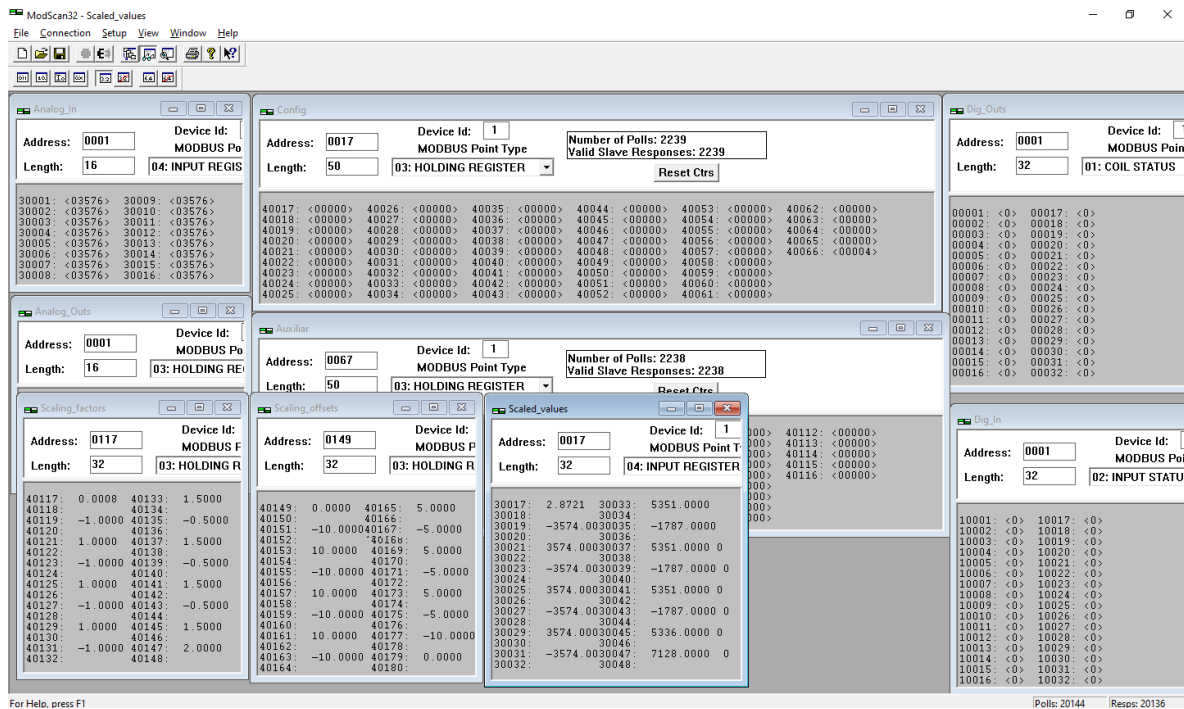
■ Tabla de factores de escalamiento s3Tables.scalingFactor

- Direcciones Modbus:
 - 40117 a 40148
- Offset: 116
- Apunta a: s3Tables.scalingFactor[0]
- Tamaño: s3Tables.anSize * 4

■ Tabla de offsets de escalamiento s3Tables.scalingOffset

- Direcciones Modbus:
 - 40149 a 40180
- Offset: 148
- Apunta a: s3Tables.scalingOffset[0]
- Tamaño: s3Tables.anSize * 4

- ✓ Verificado el funcionamiento correcto al escribir datos en las tablas de escalamiento (factores y offsets) desde ModScan32 y visualizar que se mostraban correctamente los valores calculados en los registros correspondientes de la tabla de valores escalados. Tanto los valores de las tablas de escalamiento como los valores escalados, se trabajan en formato de punto flotante, utilizando 2 registros de 16 bits para cada valor de la tabla. Una captura de pantalla de ModScan32 permite ilustrar el mapa de memoria actualizado:



- ✓ Incorporada la funcionalidad de escritura de datos en la memoria flash, a través de la biblioteca “nvs.h” y “nvs_flash.h”.

- Inicialización de la partición nvs a través de la función init_nvs():

```

esp_err_t init_nvs(void){
    esp_err_t r;
    nvs_flash_init();
    r = nvs_open("Main_Namespace", NVS_READWRITE, &app_nvs_handle);
    return r;
}

```

- La función abre el namespace denominado "Main_Namespace" para lectura y escritura.
- Creadas funciones para lectura y escritura de datos en la memoria flash read_nvs() y write_nvs(). Estas funciones reciben una clave (string) y un valor (uint_16t).
- Se crearon funciones que permitieran crear tablas en la memoria flash (ya que no es posible almacenar vectores, sino datos en forma de diccionario: "clave": valor).
 - Agregada la función create_table_nvs(), que toma como parámetros una cadena como prefijo de las claves y un valor correspondiente al tamaño de la tabla que se creará. De esta forma, si se hace esta llamada a la función:

```
create_table_nvs("C", s3Tables.configSize); //For config table
```

Se creará en la memoria flash el siguiente conjunto de pares "clave": valor

"C0":0	"C8":0	(---)	"C34":0	"C42":0
"C1":1	"C9":0	(---)	"C35":0	"C43":0
"C2":0	"C10":0	(---)	"C36":0	"C44":0
"C3":0	"C11":0	(---)	"C37":0	"C45":0
"C4":0	"C12":0	(---)	"C38":0	"C46":0
"C5":0	"C13":0	(---)	"C39":0	"C47":0
"C6":0	"C14":0	(---)	"C40":0	"C48":0
"C7":0	"C15":0	(---)	"C41":0	"C49":0

- Agregada la función create_float_table_nvs(), la cual crea igualmente una tabla en memoria flash, con la diferencia de que se utilizan datos en punto flotante.
 - Las funciones create_table_nvs() y create_float_table_nvs(), verifican primero si la tabla existe en la memoria flash, para evitar sobrescribir los valores en caso de que existan.
- ✓ Nuevo código en app_main() que inicializa la flash mediante la llamada a la función init_nvs(), procede a la creación de tablas espejo (en caso de que no existan), para la tabla de configuración, la tabla auxiliar, la tabla de factores de escalamiento y la tabla de offsets de escalamiento:

```

//Create tables in the nvs namespace (if they don't exist):
create_table_nvs("C", s3Tables.configSize); //For config table
create_table_nvs("A", s3Tables.auxSize); //For aux table
create_float_table_nvs("SF", s3Tables.anSize); //For Scaling factors table
create_float_table_nvs("SO", s3Tables.anSize); //For Scaling offsets table

```

- ✓ A continuación, se agregaron rutinas para leer los datos de las tablas en memoria flash y cargarlas en la estructura creada en memoria RAM (s3Tables). Esto permite que se recupere la configuración y el estado del sistema en caso de un apagado o reinicio del mismo.

```
for (int i = 0; i < s3Tables.configSize; i++)
{
    char key[5] = {'\0'};
    sprintf(key, "C%i", i);
    read_nvs(key, &s3Tables.configTbl[0][i]);
}

for (int i = 0; i < s3Tables.auxSize; i++)
{
    char key[5] = {'\0'};
    sprintf(key, "A%i", i);
    read_nvs(key, &s3Tables.auxTbl[0][i]);
}

for (int i = 0; i < s3Tables.anSize; i++)
{
    char key[6] = {'\0'};
    sprintf(key, "SF%i", i);
    size_t size = sizeof(float);
    esp_err_t r = nvs_get_blob(app_nvs_handle, key, &s3Tables.scalingFactor[i], &size);
    ESP_ERROR_CHECK(r);
}
```

- ✓ Creada la tarea `mb_event_check_task` para chequear si ocurrió algún evento de escritura en los Holding Registers del mapa Modbus. Dicha tarea se asignó al núcleo 1, con nivel de prioridad 2.
 - Se utiliza la función `mbc_slave_check_event(MB_EVENT_HOLDING_REG_WR)` para bloquear la tarea hasta que se produzca un evento de escritura en un holding register.
 - Una vez detectado el evento, se utiliza la función `mbc_slave_get_param_info()` para obtener la información de los eventos en la cola de notificaciones del controlador Modbus. Se leen tantos eventos como el tamaño de la cola de notificaciones. (Por defecto este tamaño es 20, aunque pudiese disminuirse a 1 para evitar notificaciones no deseadas).
 - Una vez extraída la información de cada evento de la cola, se verifica el tipo de evento; en caso de ser un evento `MB_EVENT_HOLDING_REG_WR` se determina la tabla a la cual pertenece el registro, así como el índice del elemento de dicha tabla (a partir del offset entregado en la notificación).
 - Habiendo determinado la tabla y el índice, se procede a guardar el dato de dicha tabla en la correspondiente tabla espejo en la memoria flash, para asegurar su almacenamiento persistente.

- Únicamente se escriben (por el momento) los datos que han sido cambiados desde el ModScan32 en las tablas de configuración, auxiliar y las tablas de escalamiento.
- ✓ Verificado el funcionamiento de la tarea en conjunto con las funciones de escritura en la memoria flash, a través del simulador ModScan32; se escribieron valores en las diferentes tablas y se comprobó que los valores fueron guardados en la nvs y recuperados al producirse un reinicio o un apagado.
- ✓ La comprobación de funcionamiento fue verificada también utilizando la herramienta de depuración a través de la interfaz de depuración JTAG mediante el puerto USB del ESP32-S3.

Cambios realizados hasta el 20-08-2023:

- ✓ Probada la asignación de IP Estática y el funcionamiento con conexión al router.
- ✓ Asignados los siguientes registros de la tabla de configuración para controlar las siguientes características:

Descripción	Dirección Modbus	Registro asociado	Macro
Modo Run/Program	40017	configTbl[0][0]	CFG_RUN_PGM
Mode de operación	40018	configTbl[0][1]	CFG_OP_MODE
Dirección IP Ethernet 0	40019 40020	configTbl[0][2] - configTbl[0][3]	*CFG_IP0
Dirección IP Ethernet 1	40021 40022	configTbl[0][4] - configTbl[0][5]	*CFG_IP1
Dirección IP WiFi (Station)	40023 40024	configTbl[0][6] - configTbl[0][7]	*CFG_IP2
Dirección IP WiFi (AP)	40025 40026	configTbl[0][8] - configTbl[0][9]	*CFG_IP3
Gateway	40027 40028	configTbl[0][10] - configTbl[0][11]	*CFG_GW
DHCP Mode	40029	configTbl[0][12]	CFG_DHCP

Las macros CFG_IPx son punteros al primer registro de la IP correspondiente. Así:

CFG_IP0 apunta a configTbl[0][2] y

CFG_IP0+1 apunta a configTbl[0][3]

- ✓ Los handles de las tareas de freertos ahora son distintos para cada una y son globales.
- ✓ En app_main() se chequea el registro CFG_RUN_PGM y se ponen las tareas spi_task y scaling_task en pausa o en ejecución.
- ✓ Movidas las definiciones de macros, variables globales y prototipos de funciones a remota_globals.h.
- ✓ La dirección IP es almacenada en 2 registros de 16bit de la siguiente forma, ej: 172.16.0.100

Los octetos más significativos en *CFG_IP0: 172.16 --> 0xAC10 (0xAC -> 172 y 10 -> 16)

Los octetos menos significativos en *(CFG_IP0+1): 0.100 --> 0x0064 (0x00 -> 0 y 0x64 -> 100)

- ✓ La misma forma de almacenamiento es usada para el gateway CFG_GW. También se implementará para las futuras direcciones IP (eth1, wifi station y wifi AP).

- ✓ Desde la tarea `mb_event_check_task()` al cambiar el registro `CFG_DHCP` se llama a la función `set_DHCP()` definida en `comm_services.h`.
- ✓ La función `set_DHCP()` verifica el valor del registro `CFG_DHCP`, si vale 1 inicia el servicio dhcp, si vale 0, detiene el servicio dhcp y ejecuta la función `set_ip_eth0()`.
- ✓ La función `set_ip_eth0()` configura la IP, gateway y máscara de subred para la interfaz `eth0`
 - Se toman los registros de la tabla de configuración `*CFG_IP0` y `*(CFG_IP0+1)` para reconstruir la dirección IP.
 - Se toman los registros de la tabla de configuración `*CFG_GW` y `*(CFG_GW+1)` para reconstruir la puerta de enlace.
 - Se guardan las nuevas direcciones en la estructura correspondiente y se aplica la configuración.
 - El cambio tiene lugar inmediatamente, siendo posible una configuración 'en caliente'.
- ✓ En la función `ethernetinit`, se toma en cuenta el estado del registro `CFG_DHCP` para decidir si se inicia con IP dinámica o estática:
 - Si `CFG_DHCP` vale 1, se inicia con IP dinámica
 - Si `CFG_DHCP` vale 0, se detiene el servicio dhcp y se llama a la función `set_ip_eth0()` para configurar la IP estática.
- ✓ Dado que los valores de los registros que almacenan la IP, así como el registro `CFG_DHCP` están respaldados en flash, cuando se produce un reinicio, se arranca con la última configuración.
- ✓ Si se quiere cambiar la IP estática y aplicar los cambios desde el ModScan32, se debe seguir el siguiente procedimiento:
 - Desde ModScan32 modificar los registros correspondientes a la dirección IP (`*CFG_IP0 -> 40019` y `*(CFG_IP0+1) -> 40020`)
 - Para aplicar los cambios escribir el valor 0 en `CFG_DHCP` (40029).
 - Los cambios serán aplicados inmediatamente.
- ✓ Si se quiere cambiar la IP estática mediante código de programación, (futura implementación IU):
 - Escribir los valores en los registros correspondientes a la dirección IP. (y guardar los valores en la tabla espejo de la flash).
 - Escribir 0 en el registro `CFG_DHCP` y guardar el correspondiente valor en la tabla espejo de la flash.
 - Llamar a la función `set_DHCP()`, la cual se encargará de aplicar los cambios.
 - Al haber respaldado los valores, la configuración es persistente en caso de un reinicio.
- ✓ Cuando se selecciona el modo run, no se permiten cambios a la tabla de configuración excepto en el registro `CFG_RUN_PGM` (dirección 40017) y en los registros 40062 al 40066, (reservados para hacer alguna configuración en tiempo de ejecución). Este rango de memoria podría cambiarse según la necesidad.
- ✓ Tampoco en modo RUN se permiten cambios a las tablas de escalamiento. Al cambiarse un dato no permitido, vía modbus en cualquiera de las tablas, el cambio es detectado por la tarea `mb_check_event_task`, y al verificar el modo RUN, el valor modificado en RAM es restituido al valor espejo en memoria flash.
- ✓ Cuando se selecciona el modo program, se permiten los cambios en cualquiera de las tablas, verificando los valores válidos para los registros `CFG_RUN_PGM` y `CFG_OP_MODE`. También se verifican valores válidos para el registro `CFG_REMOTA_LOG_LEVEL`.

- ✓ La modificación del modo de operación, a través del registro CFG_OP_MODE, (la cual solo es posible desde el modo program), ocasiona que el lazo principal discrimine el modo actual de operación (tipo de pozo / caseta de válvulas).
- ✓ Se colocó la estructura switch que permitirá aplicar la lógica necesaria en cada caso.

Cambios realizados hasta el 31-08-2023:

- ✓ Se reescribió el código de la tarea mb_check_event_task(), para optimizar su funcionamiento y mejorar su legibilidad. El código ahora realiza las siguientes tareas:
 - Se chequea que la pila modbus slave esté inicializada (a través de la bandera global mb_slave_initialized), en caso de no estar inicializada, se espera a que lo esté.
 - Esperar la ocurrencia de un evento de tipo holding register write
 - Se inicia una iteración para cada uno de los elementos de la cola de notificaciones modbus, en busca de un evento de tipo HOLDING_REG_WR.
 - Al encontrar dicho evento en la cola, se chequea a partir del offset (dirección modbus) del registro, para identificar a cuál tabla pertenece. (Tabla de salidas analógicas, tabla de configuración, tabla auxiliar o tablas de escalamiento).
 - En caso de pertenecer a la tabla de salidas analógicas, sólo se imprime un mensaje (verbose) en consola.
 - En caso de pertenecer a la tabla auxiliar, se calcula el índice del vector y se genera la 'key' correspondiente para la tabla espejo en flash. Una vez hecho esto, el valor escrito a través de modbus, se respalda en memoria flash en la posición correspondiente.
 - En caso de que el registro pertenezca a alguna de las tablas de escalamiento (factores u offsets), se calcula el índice del vector correspondiente y se genera la 'key' para la tabla espejo en flash. Además, se chequea el modo de ejecución (CFG_RUN_PGM):
 - Si el modo actual es RUN: Se lee el valor en flash y se restaura en la correspondiente tabla en RAM, de manera que no se permite el cambio en modo de ejecución.
 - Si el modo actual es PROGRAM: El nuevo valor en RAM se escribe en memoria flash.
 - En caso de que el registro pertenezca a la tabla de configuración, se calcula el índice del vector y se genera la 'key' correspondiente, se establece una bandera writeFlag = 1, que se utilizará para controlar cuando se desea escribir el valor en flash; luego se efectúan las siguientes comprobaciones:
 - Si el modo de ejecución es RUN y el índice es mayor que 0 y menor que 45, se prohíbe la escritura, dado que es una zona de memoria prohibida en modo de ejecución. Para esto se escribe writeFlag=0. En caso de que el índice del registro sea 0 ó sea mayor o igual a 45 (últimos 5 registros de la tabla), la escritura en ellos se permite en modo de ejecución.
 - Se verifica el índice del registro mediante una estructura switch, que permite ejecutar diferentes bloques de código para cada caso en particular; a fin de establecer las configuraciones apropiadas de acuerdo al valor establecido para cada registro. Así, se contemplan hasta ahora los siguientes casos:

- Caso 0 (registro CFG_RUM_PGM): Se valida si el dato recibido por modbus es mayor que 1, en cuyo caso se restituye el valor anterior en memoria flash, para no permitir valores incorrectos. Si el valor es válido, el programa continúa y el registro se escribe en flash, dado que la bandera writeFlag está en 1.
 - Caso 1 (registro CFG_OP_MODE): Se valida igualmente si el valor no es mayor que 5, en cuyo caso se restituye el valor anterior, para no permitir entradas incorrectas. Si la entrada es correcta, se toma en cuenta el modo de ejecución para tomar una de las siguientes acciones:
 - Si el modo actual es RUN: se establece writeFlag=0, de manera que no se escriba el valor en flash.
 - Si el modo actual es PROGRAM, se establece la bandera resetRequired=1, para controlar que una vez que se haya guardado el dato en flash, el sistema se reinicie automáticamente para aplicar la nueva configuración. Es necesario efectuar el reinicio, dado que al cambiar el modo de operación es preciso redefinir completamente el mapa de memoria modbus, creando las tablas adicionales necesarias en memoria. Se intentó realizar una des-inicialización de la pila modbus slave, pero a pesar de haber funcionado, al cambiar el modo de operación varias veces seguidas, el programa fallaba debido a problemas de fragmentación de la memoria. De manera que se optó por provocar un reinicio del microcontrolador para aplicar la nueva configuración desde el arranque.
 - Caso 12 (registro CFG_DHCP): Se valida que el valor establecido no sea mayor que 1, en cuyo caso se hace writeFlag=0. En caso de que el valor sea válido, se chequea el modo de ejecución actual:
 - Si el modo actual es RUN: se establece writeFlag=0, para no permitir el cambio en modo de ejecución.
 - Si el modo actual es PROGRAM: se llama a la función set_DHCP(), la cual establecerá la configuración para el servicio DHCP y la dirección IP estática, dependiendo del valor establecido en el registro CFG_DHCP.
 - Caso 49 (registro CFG_REMOTA_LOG_LEVEL): Se valida que el valor no sea superior a 5, en cuyo caso se prohíbe la escritura a través de writeFlag=0. En caso de una entrada válida, se establece el valor para el nivel de log de la consola, en cada una de las etiquetas o TAGS del programa.
 - *Los demás casos se irán agregando conforme se vaya desarrollando el código que corresponda para cada opción de configuración.*
- Al salir de la estructura switch, el programa continúa chequeando el estado de la bandera writeFlag:

- Si writeFlag=1: el valor de configuración se escribe en la posición correspondiente de la flash. Además se comprueba si resetRequired=1 y el índice es 0 (CFG_RUN_MODE) y el modo actual es RUN; sólo en este caso se ejecuta esp_restart(), produciendo un reinicio si se estableció el modo de ejecución a RUN y la bandera resetRequired=1. Mediante éste código, se produce el reinicio que aplica los cambios al modo de operación seleccionado previamente.
 - Si writeFlag=0: el valor es restaurado en RAM por el correspondiente valor leído desde flash.
- ✓ Se modificó la estructura definida para las tablas en memoria, para añadir la creación de tablas adicionales, a fin de almacenar los registros modbus, leídos desde los dispositivos esclavos:

```
typedef struct {
    uint16_t** anTbl;      // Vector de apuntadores a los vectores analógicos
    uint16_t** digTbl;     // Vector de apuntadores a los vectores Digitales
    uint16_t** configTbl;  // Vector de apuntadores a los vectores de configuración
    uint16_t** auxTbl;     // Vector de apuntadores a los vectores auxiliares
    uint16_t** mbTbl16bit; // Vector de apuntadores a los vectores Modbus de 16 bits //A reservar según el caso
    uint8_t** mbTbl8bit;   // Vector de apuntadores a los vectores Modbus de 8 bits //A reservar según el caso
    float** mbTblFloat;    // Vector de apuntadores a los vectores Modbus Float //A reservar según el caso
    float* scalingFactor;  // Vector de factores de escalamiento (pendiente m)
    float* scalingOffset;  // Vector de desplazamientos en la escala (corte con y -> b)
    float* scaledValues;   // Vector de apuntadores a los vectores de valores escalados
    uint8_t anSize;        // Tamaño de los vectores analógicos
    uint8_t digSize;       // Tamaño de los vectores analógicos
    uint8_t configSize;    // Tamaño de los vectores de configuración
    uint8_t auxSize;       // Tamaño de los vectores auxiliares
    uint8_t numAnTbIs;     // Número de vectores analógicos
    uint8_t numDigTbIs;    // Número de vectores digitales
    uint8_t numConfigTbIs; // Número de vectores de configuración
    uint8_t numAuxTbIs;    // Número de vectores auxiliares
} varTables_t;
```

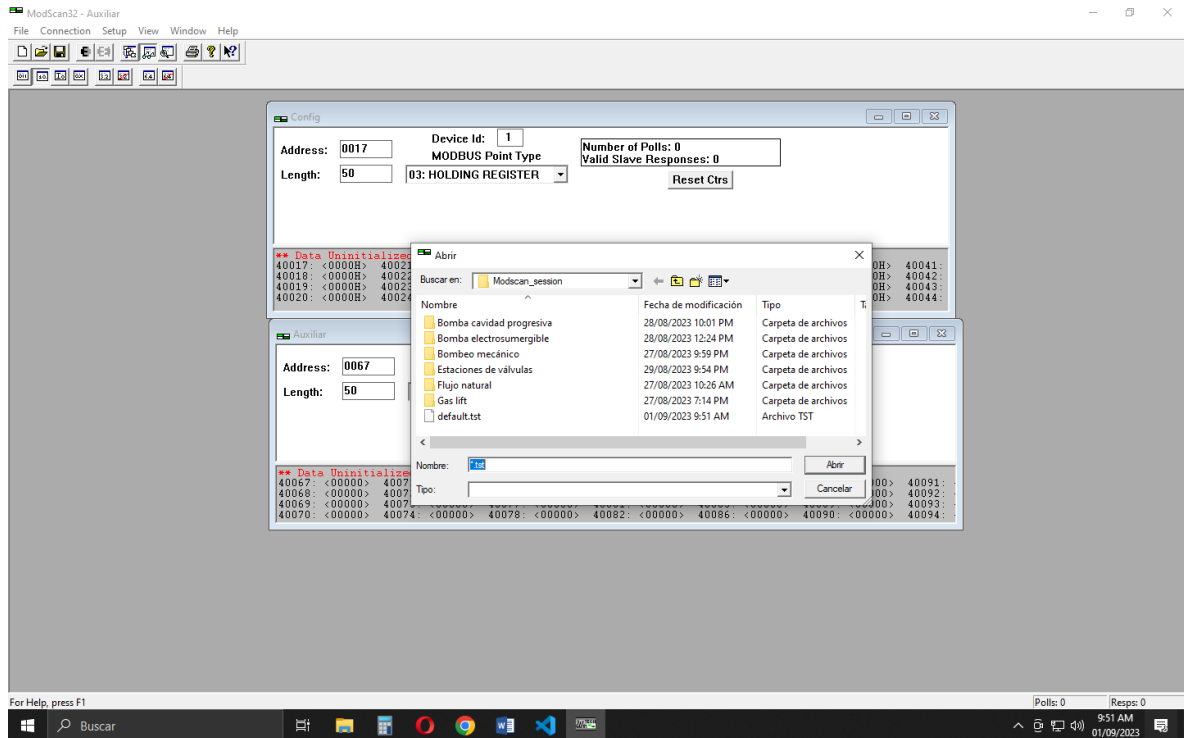
- ✓ Se creó la función: create_modbus_map(), la cual genera el mapa de memoria modbus dependiendo del modo de operación. Dicha función contempla la creación de un mapa modbus básico (que contiene las tablas comunes para todos los casos de operación, como son la tabla de configuración, la tabla auxiliar y la tabla de entradas digitales); además del mapa modbus extendido, el cual será particularizado dependiendo de cada caso de operación.

La creación del mapa modbus para cada uno de los 5 métodos de producción y para el caso de estaciones de válvulas, se realizó siguiendo las especificaciones del documento “Informe Técnico Filosofía de Control A2SCP Rev0_Tesis_ULA”. Para ello se generó el documento “Variables a considerar dependiendo del método de producción”; el cual detalla las variables a considerar para cada caso y su tipo, de acuerdo a las tablas diseñadas en el programa.

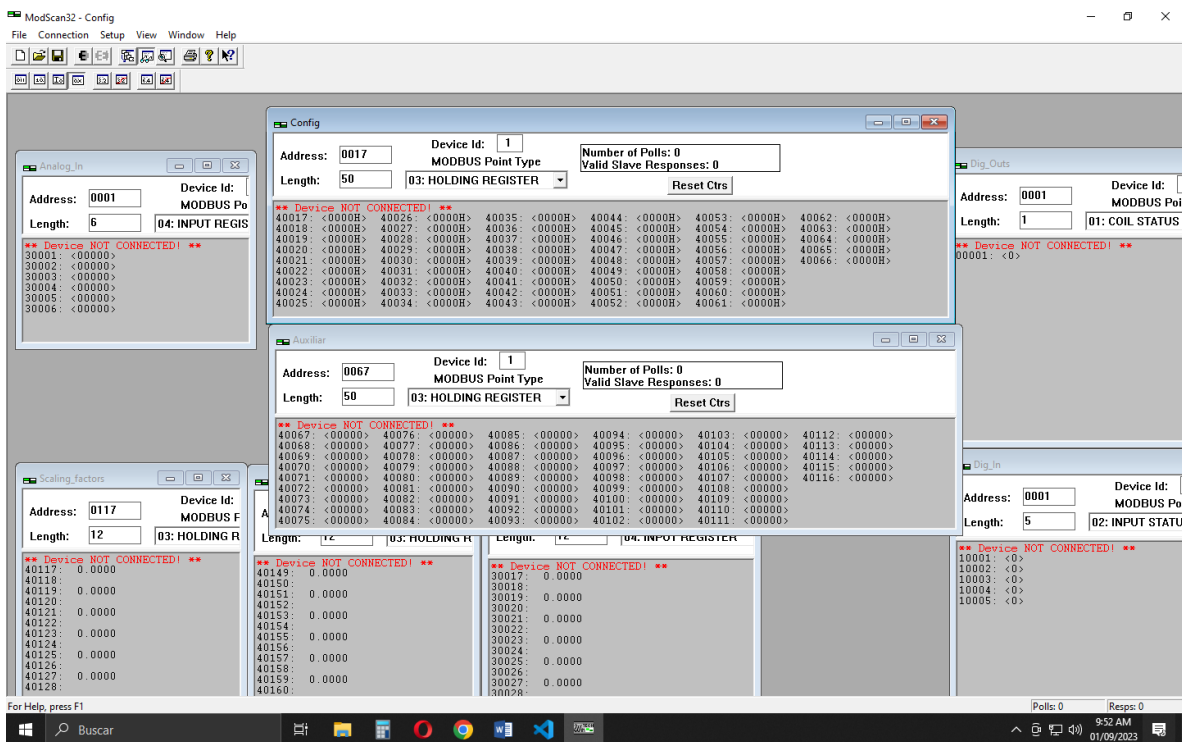
En la creación del mapa modbus extendido, se dispone de una estructura switch, que selecciona el mapa a crear de acuerdo al registro CFG_OP_MODE, además para los casos particulares de Pozos de Bombeo Mecánico, Pozos con Bomba Electrosomergible y Pozos con Bomba de Cavidad Progresiva; en los cuales es necesaria la comunicación modbus con equipos en campo, se crearon las tablas adicionales en memoria para almacenar apropiadamente las variables asociadas a los mismos, así como las áreas de memoria modbus para cada una de ellas. Dichas variables serán leídas y/o escritas a partir de la comunicación modbus master.

- ✓ La función create_modbus_map(), es llamada al ejecutarse la función modbus_slave_init(), que se ejecuta durante el inicio de la tarea app_main().

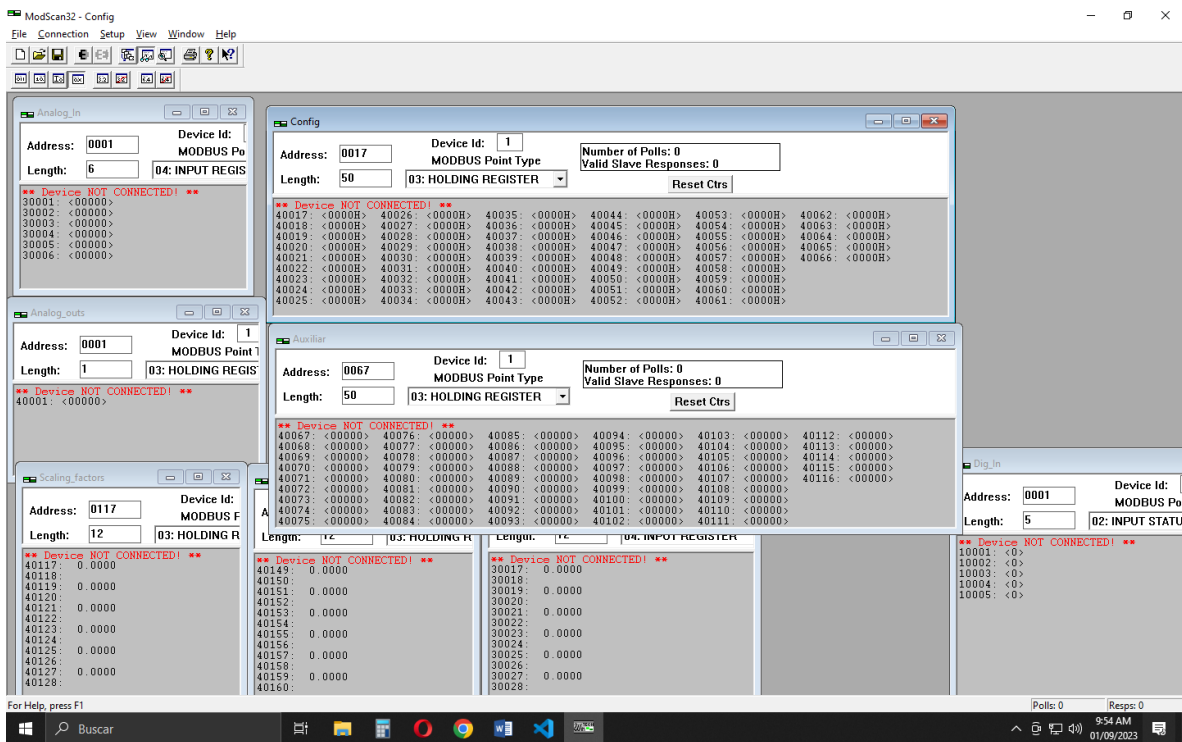
- ✓ Se utilizó el software ModScan32, para crear las configuraciones necesarias para leer y escribir las variables modbus dependiendo del mapa creado para cada método de producción, para ello, se crearon carpetas para contener tanto los archivos de las tablas del programa, como un archivo que almacena la configuración de las ventanas abiertas en el programa. Así, para cada método de producción se cuenta con una carpeta y con un conjunto de archivos que permiten restaurar de forma rápida la configuración de las ventanas necesarias de acuerdo al mapa modbus de cada caso.



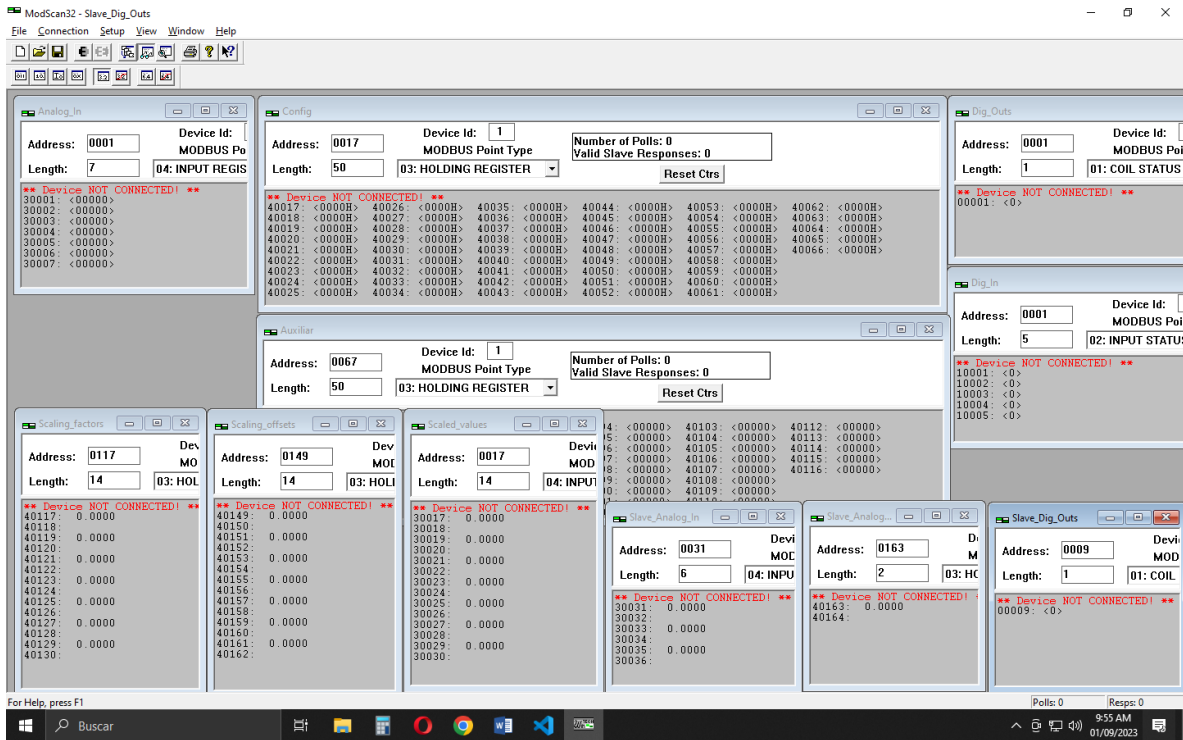
Flujo Natural:



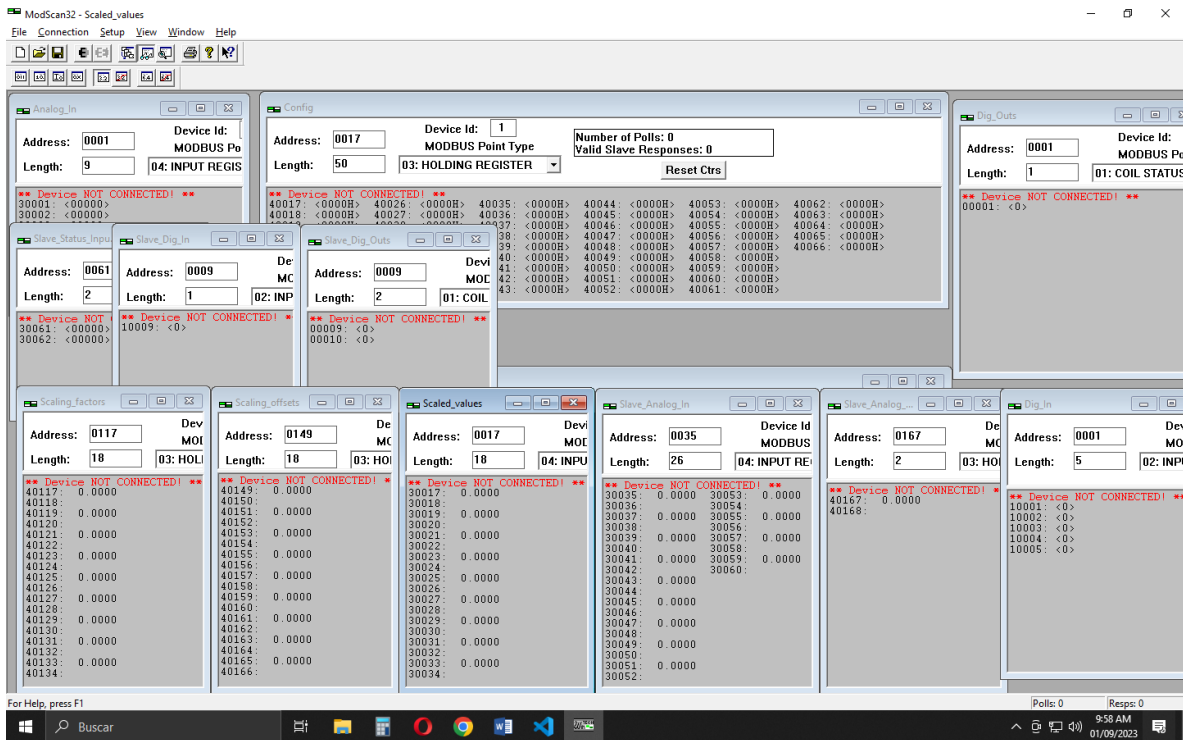
Gas Lift:



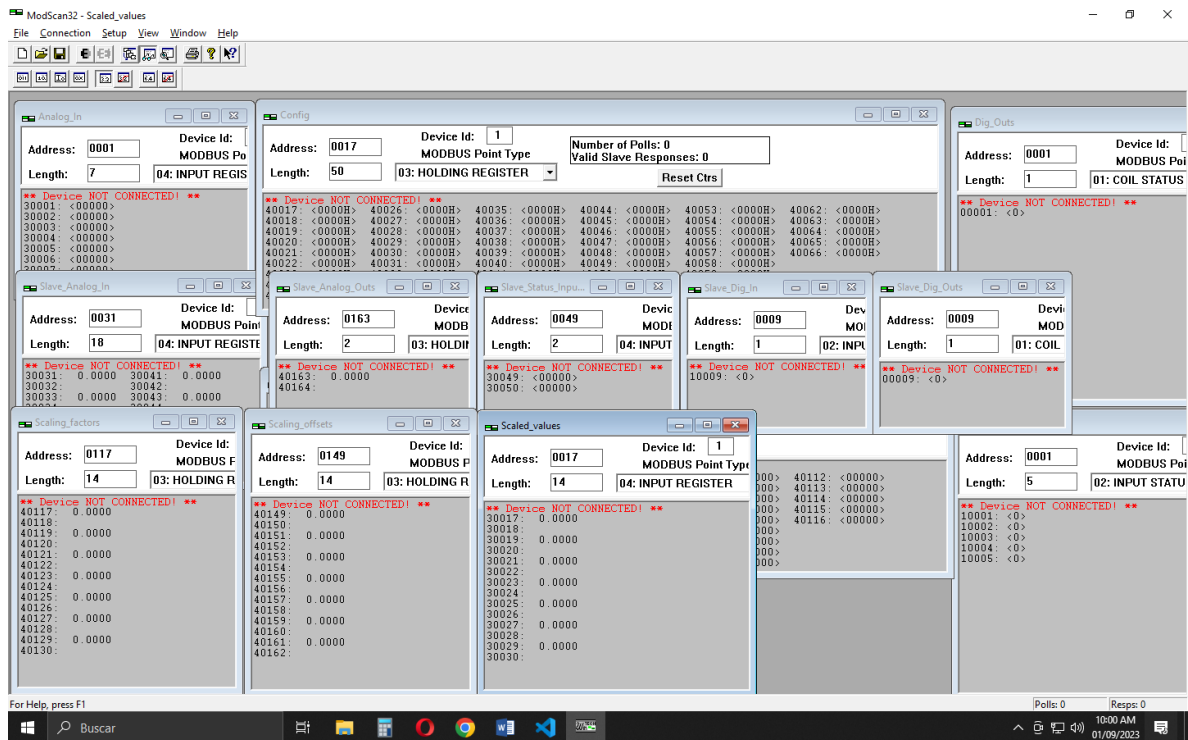
Bombeo Mecánico:



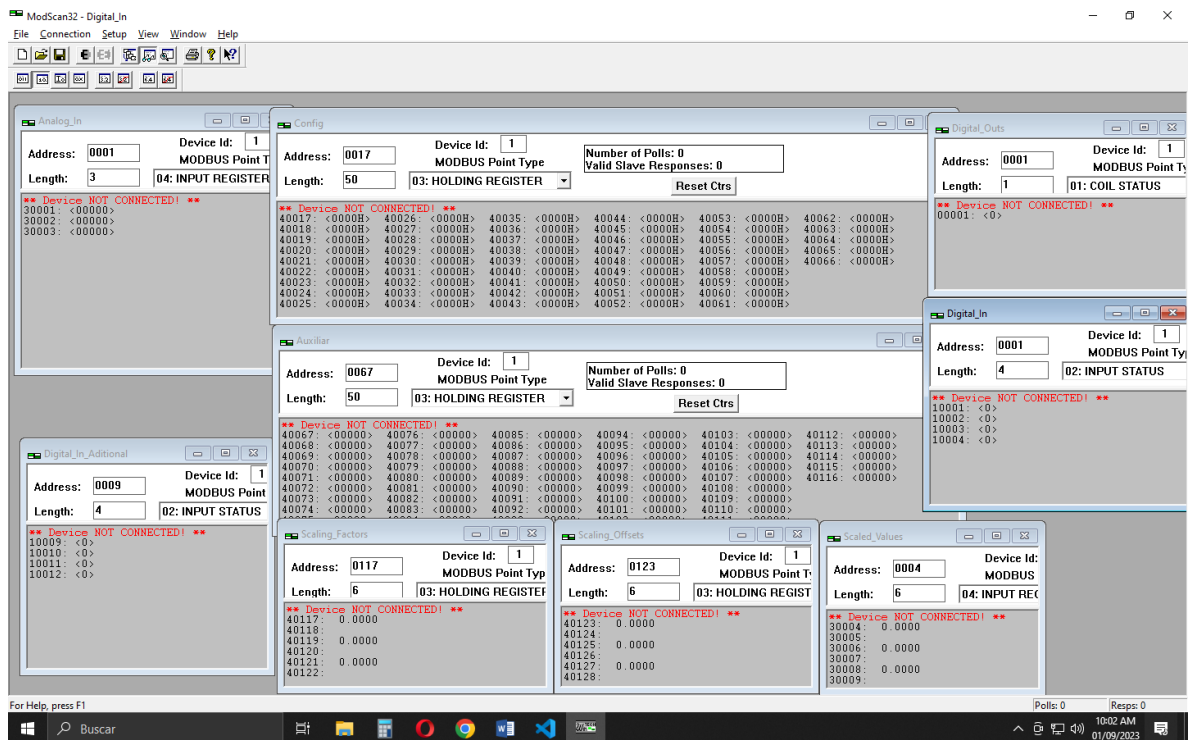
Bomba Electrosumergible:



Bomba de Cavidad Progressiva:



Estaciones de Válvulas:



- ✓ Se creó el archivo "remota_var_defs.h" el cual contiene las definiciones de las macros para cada una de las variables de los diferentes métodos de producción; en dicho archivo se hacen las

asociaciones de los registros correspondientes a las variables con un nombre nemotécnico que pueda identificar plenamente al registro en cuestión. Para esto, se utilizó la siguiente nomenclatura:

```
main > C remota_var_defs.h > ...
1  /*
2      Macro definitions for each operation modes
3      Each macro follows the notation:
4      |   #define OPMode_Type_VarName ... // Description
5  */
6
7  #ifndef REMOTA_VAR_DEFS
8  #define REMOTA_VAR_DEFS
9
10 //Macros for: Natural flow wells
11 //
12 #define NF_AI_PTL s3Tables.anTbl[0][0] // Presión de la línea de producción
13 #define NF_AI_TTL s3Tables.anTbl[0][1] // Temperatura de la línea de producción
14 #define NF_AI_PTC s3Tables.anTbl[0][2] // Presión de cabezal
15 #define NF_AI_PTA s3Tables.anTbl[0][3] // Presión de cabezal
```

- ✓ Para OPMode, se tiene:
 - NF: Flujo natural
 - GL: Gas Lift
 - MP: Bombeo Mecánico
 - ES: Bomba Electrosumergible
 - PC: Cavidad Progresiva
 - VS: Estaciones de Válvulas
- ✓ Para Type:
 - AI: Analógica de entrada
 - AO: Analógica de salida
 - DI: Digital de entrada
 - DO Digital de salida
 - SF: Factor de escalamiento
 - SO: Offset de escalamiento
 - SV: Valor escalado
 - IR: Input register (variables del esclavo en campo)
 - HR: Holding register (variables del esclavo en campo)
 - DISCRETE: Input status (variables del esclavo en campo)
 - COIL: Coil status (variables del esclavo en campo)
- ✓ Para VarName: Se utilizaron los mismos nombres definidos en el documento “Informe Técnico Filosofía de Control A2SCP Rev0_Tesis_ULA”.
- ✓ Para el caso de las variables digitales, se tiene la siguiente particularidad, ya que desde Modbus, dichas variables son DISCRETES o COILS y son manejadas un bit a la vez, mientras que en el programa de la remota, dichas variables son almacenadas como registros de 8 bits ó de 16 bits. Entonces, en las definiciones de las macros, se incluyó el código necesario para hacer referencia al bit particular de dicha variable en el registro donde se almacena, por ejemplo:

```
#define NF_DI_YA1    (s3Tables.digTbl[0][0] & 0x0001)      // Alarma por falla de sistema
#define NF_DI_YA2    (s3Tables.digTbl[0][0] & 0x0002) >> 1 // Alarma de intruso
#define NF_DI_GZA    (s3Tables.digTbl[0][0] & 0x0004) >> 2 // Alarma por gas tóxico
#define NF_DI_EAAC    (s3Tables.digTbl[0][0] & 0x0008) >> 3 // Alarma por falla de voltaje AC
#define NF_DI_EADC    (s3Tables.digTbl[0][0] & 0x0016) >> 4 // Alarma por falla de voltaje DC

#define NF_DO_XV     (s3Tables.digTbl[1][0] & 0x0001)      // Apertura/cierre de pozo
```

```
#define ES_DISCRETE_YI (s3Tables.mbTbl8bit[0][0] & 0x01)    // Encendido/apagado

#define ES_COIL_HS1    (s3Tables.mbTbl8bit[0][1] & 0x01)    // Encendido/apagado
#define ES_COIL_HS2    (s3Tables.mbTbl8bit[0][1] & 0x02) >> 1 // Reposición de causa de paro
```

- ✓ Se reescribió parte del código del lazo infinito de la tarea app_main(), particularmente donde se hace la selección (switch) para el modo de operación (CFG_OP_MODE) y así seleccionar las acciones a realizar dependiendo del método de producción. Para cada caso se escribió el código necesario para mostrar en consola la salida apropiada, considerando la escritura de mensajes particulares especificando la descripción, nombre y valor actual de cada variable asociada a dicho método.

Ahora la salida en consola del programa es mucho más detallada y particularizada para el modo de operación, como se muestra a continuación:

Flujo natural:

```
D (25089) Remota-Main: Transaction count: 102400 Error count: 0 Error ratio: 0.00%
D (25089) Remota-Main: SPI exchange task time: 187 us
D (25089) Remota-Main: SPI cycle task time: 190 us

I (25089) Remota-Main: Pozo de Flujo Natural
I (25099) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (25109) Remota-Main: Presión de la línea de producción      (PTL): 1841
I (25109) Remota-Main: Temperatura de la línea de producción (TTL): 1841
I (25119) Remota-Main: Presión de cabezal                    (PTC): 1837
I (25129) Remota-Main: Presión del casing o anular           (PTA): 1837
I (25129) Remota-Main: Temperatura del casing o anular      (TTA): 1837
I (25139) Remota-Main: Flujo de producción                  (FT): 1838

I (25149) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (25159) Remota-Main: Presión de la línea de producción      (PTL): 1.554
I (25159) Remota-Main: Temperatura de la línea de producción (TTL): 1483.590
I (25169) Remota-Main: Presión de cabezal                    (PTC): 1851.000
I (25179) Remota-Main: Presión del casing o anular           (PTA): -1851.000
I (25179) Remota-Main: Temperatura del casing o anular      (TTA): 1851.000
I (25189) Remota-Main: Flujo de producción                  (FT): -1851.000

I (25199) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (25209) Remota-Main: Alarma por falla de sistema          (YA1): 0
I (25209) Remota-Main: Alarma de intruso                   (YA2): 0
I (25219) Remota-Main: Alarma por gas tóxico                (GZA): 0
I (25229) Remota-Main: Alarma por falla de voltaje AC       (EAAC): 0
I (25229) Remota-Main: Alarma por falla de voltaje DC       (EADC): 0

I (25239) Remota-Main: Variables digitales de salida leídas desde el módulo de E/S:
I (25249) Remota-Main: Apertura/cierre de pozo             (XV): 0
```

Gas Lift:

```
D (7649) Remota-Main: Transaction count: 31120 Error count: 0 Error ratio: 0.00%
D (7649) Remota-Main: SPI exchange task time: 187 us
D (7649) Remota-Main: SPI cycle task time: 191 us

I (7649) Remota-Main: Pozo de Gas Lift
I (7659) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (7659) Remota-Main: Presión de la línea de producción (PTL): 1839
I (7669) Remota-Main: Temperatura de la línea de producción (TTL): 1839
I (7679) Remota-Main: Posición válvula control flujo al pozo (ZT): 1839
I (7689) Remota-Main: Presión de gas al pozo (PTGL): 1839
I (7689) Remota-Main: Temperatura de gas al pozo (TTGL): 1839
I (7699) Remota-Main: Flujo de gas al pozo (FTGL): 1839

I (7709) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (7709) Remota-Main: Presión de la línea de producción (PTL): 1.552
I (7719) Remota-Main: Temperatura de la línea de producción (TTL): 1481.978
I (7729) Remota-Main: Posición válvula control flujo al pozo (ZT): 1849.000
I (7739) Remota-Main: Presión de gas al pozo (PTGL): -1849.000
I (7749) Remota-Main: Temperatura de gas al pozo (TTGL): 1849.000
I (7749) Remota-Main: Flujo de gas al pozo (FTGL): -1849.000

I (7759) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (7769) Remota-Main: Alarma por falla de sistema (YA1): 0
I (7769) Remota-Main: Alarma de intruso (YA2): 0
I (7779) Remota-Main: Alarma por gas tóxico (GZA): 0
I (7789) Remota-Main: Alarma por falla de voltaje AC (EAAC): 0
I (7789) Remota-Main: Alarma por falla de voltaje DC (EADC): 0

I (7799) Remota-Main: Variables analógicas de salida escritas al módulo de E/S:
I (7809) Remota-Main: Flujo de gas al pozo (FTGL): 0

I (7809) Remota-Main: Volumen diario de gas inyectado:
I (7819) Remota-Main: Volumen total diario de gas (FQ): 0
```

Bombeo Mecánico:

```
D (52609) Remota-Main: SPI exchange task time: 186 us
D (52609) Remota-Main: SPI cycle task time: 190 us

I (52609) Remota-Main: Pozos de Bombeo Mecánico
I (52619) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (52629) Remota-Main: Presión de la línea de producción (PTL): 1837
I (52629) Remota-Main: Temperatura de la línea de producción (TTL): 1837
I (52639) Remota-Main: Presión del casing o anular (PTA): 1839
I (52649) Remota-Main: Presión de cabezal (PTC): 1839
I (52649) Remota-Main: Golpe por minuto (spm) (SPM): 1839
I (52659) Remota-Main: Ángulo de la viga (ZT): 1838
I (52669) Remota-Main: Carga de la sarta (WT): 1838

I (52669) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (52679) Remota-Main: Presión de la línea de producción (PTL): 1.550
I (52689) Remota-Main: Temperatura de la línea de producción (TTL): 1480.367
I (52699) Remota-Main: Presión del casing o anular (PTA): 1847.000
I (52699) Remota-Main: Presión de cabezal (PTC): -1847.000
I (52709) Remota-Main: Golpe por minuto (spm) (SPM): 1847.000
I (52719) Remota-Main: Ángulo de la viga (ZT): -1847.000
I (52719) Remota-Main: Carga de la sarta (WT): 1847.000

I (52729) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (52739) Remota-Main: Alarma por falla de sistema (YA1): 0
I (52739) Remota-Main: Alarma de intruso (YA2): 0
I (52749) Remota-Main: Alarma por gas tóxico (GZA): 0
I (52759) Remota-Main: Alarma por falla de voltaje AC (EAAC): 0
I (52759) Remota-Main: Alarma por falla de voltaje DC (EADC): 0

I (52769) Remota-Main: Variables digitales de salida leídas desde el módulo de E/S:
I (52779) Remota-Main: Encendido/apagado de bombeo (HS1): 0

I (52779) Remota-Main: Variables modbus (input registers) leídas desde el equipo esclavo:
I (52789) Remota-Main: Voltaje de motor (ETM): 0.000
I (52799) Remota-Main: Corriente de motor (ITM): 0.000
I (52809) Remota-Main: Potencia de motor (JTM): 0.000

I (52809) Remota-Main: Variables modbus (holding registers) leídas desde el equipo esclavo:
I (52819) Remota-Main: Ajuste de velocidad (SCM): 0.000

I (52829) Remota-Main: Variables modbus (COILS) leídas desde el equipo esclavo:
I (52829) Remota-Main: Encendido/apagado de bombeo (HS1): 0
```

Bomba Electrosumergible:

```
I (22599) Remota-Main: Pozos de Bomba Electrosumergible
I (22609) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (22619) Remota-Main: Presión de la línea de producción (PTL): 1832
I (22619) Remota-Main: Temperatura de la línea de producción (TTL): 1832
I (22629) Remota-Main: Presión del casing o anular (PTA): 1839
I (22639) Remota-Main: Presión de cabezal (PTC): 1839
I (22639) Remota-Main: Presión de succión de la bomba (PTSB): 1839
I (22649) Remota-Main: Presión de descarga de la bomba (PTDB): 1839
I (22659) Remota-Main: Temperatura de succión de la bomba (TTSB): 1839
I (22669) Remota-Main: Temperatura de descarga de la bomba (TTDB): 1838
I (22669) Remota-Main: Vibración de bomba (VT): 1838

I (22679) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (22689) Remota-Main: Presión de la línea de producción (PTL): 1.546
I (22699) Remota-Main: Temperatura de la línea de producción (TTL): 1476.337
I (22699) Remota-Main: Presión del casing o anular (PTA): 1842.000
I (22709) Remota-Main: Presión de cabezal (PTC): -1842.000
I (22719) Remota-Main: Presión de succión de la bomba (PTSB): 1842.000
I (22729) Remota-Main: Presión de descarga de la bomba (PTDB): -1842.000
I (22729) Remota-Main: Temperatura de succión de la bomba (TTSB): 1842.000
I (22739) Remota-Main: Temperatura de descarga de la bomba (TTDB): -1842.000
I (22749) Remota-Main: Vibración de bomba (VT): 2753.000

I (22759) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (22759) Remota-Main: Alarma por falla de sistema (YA1): 0
I (22769) Remota-Main: Alarma de intruso (YA2): 0
I (22779) Remota-Main: Alarma por gas tóxico (GZA): 0
I (22779) Remota-Main: Alarma por falla de voltaje AC (EAAC): 0
I (22789) Remota-Main: Alarma por falla de voltaje DC (EADC): 0

I (22799) Remota-Main: Variables digitales de salida leídas desde el módulo de E/S:
I (22799) Remota-Main: Encendido/apagado de bombeo (HS1): 0

I (22809) Remota-Main: Variables modbus (input registers) leídas desde el equipo esclavo:
I (22819) Remota-Main: Voltaje de motor (ETM): 0.000
I (22829) Remota-Main: Corriente de motor (ITM): 0.000
I (22829) Remota-Main: Frecuencia de salida (ST1): 0.000
I (22839) Remota-Main: Temperatura de arrollado de motor (TTH): 0.000
I (22849) Remota-Main: Torque de motor (WTM): 0.000
I (22849) Remota-Main: Torque de bomba (WTB): 0.000
I (22859) Remota-Main: Velocidad de la bomba (ST2): 0.000
I (22859) Remota-Main: Relación de poleas en caja de velocidad (VFD) (SFH): 0.000
I (22869) Remota-Main: Presión de succión de la bomba (PTSB): 0.000

I (22889) Remota-Main: Temperatura de succión de la bomba (TTSB): 0.000
I (22899) Remota-Main: Temperatura de descarga de la bomba (TTDB): 0.000
I (22899) Remota-Main: Vibración de bomba (VT): 0.000

I (22909) Remota-Main: Variables de estatus del variador (VI1): 0
I (22919) Remota-Main: Fallas actuales del VFD (VI2): 0

I (22919) Remota-Main: Variables modbus (holding registers) leídas desde el equipo esclavo:
I (22929) Remota-Main: Velocidad de la bomba (SCM): 0.000

I (22939) Remota-Main: Variables modbus (DISCRETES) leídas desde el equipo esclavo:
I (22949) Remota-Main: Encendido/apagado (VI): 0

I (22949) Remota-Main: Variables modbus (COILS) leídas desde el equipo esclavo:
I (22959) Remota-Main: Encendido/apagado (HS1): 0
I (22969) Remota-Main: Reposición de causa de paro (HS2): 0
```

Bomba de Cavidad Progresiva:

```
I (20309) Remota-Main: Pozos con Bomba de Cavidad Progresiva
I (20319) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (20329) Remota-Main: Presión de la línea de producción (PTL): 1838
I (20329) Remota-Main: Temperatura de la línea de producción (TTL): 1838
I (20339) Remota-Main: Presión de succión de la bomba (PTSB): 1838
I (20349) Remota-Main: Presión de descarga de la bomba (PTDB): 1838
I (20359) Remota-Main: Temperatura de succión de la bomba (TTSB): 1839
I (20359) Remota-Main: Temperatura de descarga de la bomba (TTDB): 1839
I (20369) Remota-Main: Vibración de bomba (VT): 1839

I (20379) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (20389) Remota-Main: Presión de la línea de producción (PTL): 1.551
I (20389) Remota-Main: Temperatura de la línea de producción (TTL): 1481.172
I (20399) Remota-Main: Presión de succión de la bomba (PTSB): 1848.000
I (20409) Remota-Main: Presión de descarga de la bomba (PTDB): -1848.000
I (20419) Remota-Main: Temperatura de succión de la bomba (TTSB): 1848.000
I (20419) Remota-Main: Temperatura de descarga de la bomba (TTDB): -1848.000
I (20429) Remota-Main: Vibración de bomba (VT): 1848.000

I (20439) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (20449) Remota-Main: Alarma por falla de sistema (YA1): 0
I (20449) Remota-Main: Alarma de intruso (YA2): 0
I (20459) Remota-Main: Alarma por gas tóxico (GZA): 0
I (20469) Remota-Main: Alarma por falla de voltaje AC (EAAC): 0
I (20469) Remota-Main: Alarma por falla de voltaje DC (EADC): 0

I (20479) Remota-Main: Variables digitales de salida leídas desde el módulo de E/S:
I (20489) Remota-Main: Encendido/apagado de bombeo (HS1): 0

I (20489) Remota-Main: Variables modbus (input registers) leídas desde el equipo esclavo:
I (20499) Remota-Main: Torque de motor (WTM): 0.000
I (20509) Remota-Main: Torque de bomba (WTB): 0.000
I (20519) Remota-Main: Velocidad de la bomba (ST2): 0.000
I (20519) Remota-Main: Relación de poleas en caja de velocidad (VFD) (SFM): 0.000
I (20529) Remota-Main: Presión de succión de la bomba (PTSB): 0.000
I (20539) Remota-Main: Presión de descarga de la bomba (PTDB): 0.000
I (20549) Remota-Main: Temperatura de succión de la bomba (TTSB): 0.000
I (20549) Remota-Main: Temperatura de descarga de la bomba (TTDB): 0.000
I (20559) Remota-Main: Vibración de bomba (VT): 0.000

I (20569) Remota-Main: Variables de estatus del variador (YI1): 0
I (20569) Remota-Main: Fallas actuales del VFD (YI2): 0

I (20579) Remota-Main: Variables modbus (holding registers) leídas desde el equipo esclavo:
I (20589) Remota-Main: Velocidad de la bomba (SCM): 0.000

I (20599) Remota-Main: Variables modbus (DISCRETES) leídas desde el equipo esclavo:
I (20599) Remota-Main: Encendido/apagado (VI): 0

I (20609) Remota-Main: Variables modbus (COILS) leídas desde el equipo esclavo:
I (20619) Remota-Main: Encendido/apagado (HS1): 0
```

Estaciones de Válvulas:

```
D (14309) Remota-Main: Transaction count: 62056 Error count: 0 Error ratio: 0.00%
D (14309) Remota-Main: SPI exchange task time: 187 us
D (14309) Remota-Main: SPI cycle task time: 191 us

I (14309) Remota-Main: Estaciones de Válvulas

I (14319) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (14329) Remota-Main: Transmisor de presión (PT): 1839
I (14329) Remota-Main: Transmisor de temperatura (TT): 1839
I (14339) Remota-Main: Transmisor de interface (AT): 1840

I (14349) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (14349) Remota-Main: Transmisor de presión (PT): 1.552
I (14359) Remota-Main: Transmisor de temperatura (TT): 1481.978
I (14369) Remota-Main: Transmisor de interface (AT): 1849.000

I (14369) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (14379) Remota-Main: Alarma por falla de sistema (YA1): 0
I (14389) Remota-Main: Alarma de intruso (YA2): 0
I (14389) Remota-Main: Alarma por falla de voltaje AC (EAAC): 0
I (14399) Remota-Main: Alarma por falla de voltaje DC (EADC): 0

I (14409) Remota-Main: Detector de herramienta de limpieza (XI): 0
I (14419) Remota-Main: Bajo nivel de nitrógeno (LNL): 0
I (14419) Remota-Main: Válvula completamente abierta (OV): 0
I (14429) Remota-Main: Válvula completamente cerrada (CV): 0

I (14429) Remota-Main: Variables digitales de salida leídas desde el módulo de E/S:
I (14439) Remota-Main: Cierre rápido de la válvula (XV): 0
```

Cambios realizados hasta el 03-09-2023:

- ✓ Se resolvió un problema de rendimiento que se había notado desde que se agregó una mayor cantidad de líneas de salida por consola; dicho problema de rendimiento afectaba los tiempos de comunicación SPI con el módulo E/S, produciendo retardos no deseados en la comunicación al momento de hacer la escritura de datos por consola. El problema fue resuelto al asignar la tarea `spi_task()` al núcleo 1 y elevar su prioridad a nivel 2.
- ✓ Incorporada la funcionalidad de protocolo Modbus TCP Maestro, a través de la biblioteca `esp_modbus_master.h`. Para esto se realizaron los siguientes cambios:
 - Se agregó una nueva función para la inicialización de la pila modbus maestro, dicha función es llamada `modbus_master_init()`, y se ejecuta al inicio de la tarea principal `app_main()`.
 - La ejecución de dicha función está condicionada al modo de operación, y solo se ejecuta en caso de pozos de bombeo mecánico, pozos con bomba electrosomergible y pozos con bomba de cavidad progresiva.

```

modbus_slave_init();

// Starts modbus master stack and modbus master poll task only if required
if ((CFG_OP_MODE == 2) || (CFG_OP_MODE == 3) || (CFG_OP_MODE == 4)){
    //Create and start modbus master poll task:
    xTaskCreatePinnedToCore(mb_master_poll_task,
        "mb_master_poll_task",
        STACK_SIZE,
        NULL,
        (UBaseType_t) 1U,          //Priority Level 0
        &xMBMasterPollTaskHandle,
        1);

    mb_master_task_created = 1;

    //Init modbus master stack
    esp_err_t r = modbus_master_init();
    if (r != ESP_OK){
        ESP_LOGE(TAG, "Modbus master initialization error %x", r);
    }
    else {
        modbus_master_initialized = 1;
        ESP_LOGI(TAG, "Modbus master initialized");
    }
}
}

```

```

esp_err_t modbus_master_init(void){

    esp_err_t err;
    mb_communication_info_t comm_info;
    memset(&comm_info, 0, sizeof(mb_communication_info_t));

    if (CFG_MB_MASTER_INTERFACE){
        void* master_handler = NULL; // Pointer to allocate interface structure
        // Initialization of Modbus master for TCP/IP
        err = mbc_master_init_tcp(&master_handler);
        if (master_handler == NULL || err != ESP_OK) {
            ESP_LOGE(TAG, "mb controller initialization fail.");
        }

        const char* slave_ip_address_table[3] = {
            "172.16.0.4",    // Address corresponds to UID1 and set to predefined value by user
            NULL              // end of table
        };

        comm_info.ip_port = 502; // Modbus TCP port number (default = 502)
    }
}

```

```

        comm_info.ip_addr_type = MB_IPV4;                // version of IP protocol
        comm_info.ip_mode = MB_MODE_TCP;                // Port communication mode
        comm_info.ip_addr = (void*)slave_ip_address_table; // assign table of IP addresses
        comm_info.ip_netif_ptr = eth_netif;             // esp_netif_ptr pointer to the
corresponding network interface

        ESP_ERROR_CHECK(mbc_master_setup((void*)&comm_info));

    }
    else {
        void* master_handler = NULL; // Pointer to allocate interface structure
        // Initialization of Modbus master for serial port
        err = mbc_master_init(MB_PORT_SERIAL_MASTER, &master_handler);
        if (master_handler == NULL || err != ESP_OK) {
            ESP_LOGE(TAG, "mb controller initialization fail.");
        }

        comm_info.port = 2;                // Serial port number
        comm_info.mode = MB_MODE_RTU;       // Modbus mode of communication (MB_MODE_RTU or
MB_MODE_ASCII)
        comm_info.baudrate = ((uint32_t)CFG_MB_MASTER_BAUDRATE_H << 16) |
CFG_MB_MASTER_BAUDRATE_L;                // Modbus communication baud rate
        comm_info.parity = MB_PARITY_NONE; // parity option for serial port

        ESP_ERROR_CHECK(mbc_master_setup((void*)&comm_info));

        // Set UART pin numbers
        ESP_ERROR_CHECK(uart_set_pin(2, 41, 42, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE));
    }

    switch (CFG_OP_MODE) //Select appropriate dictionary depending on OP Mode
    {
        case 2: //Mechanical Pump Wells
            ESP_ERROR_CHECK(mbc_master_set_descriptor(&MP_device_parameters[0],
num_MP_device_parameters));
            break;
        case 3: //Electro Submersible Pump Wells
            ESP_ERROR_CHECK(mbc_master_set_descriptor(&ES_device_parameters[0],
num_ES_device_parameters));
            break;
        case 4: //Progressive Cavity Pump Wells
            ESP_ERROR_CHECK(mbc_master_set_descriptor(&PC_device_parameters[0],
num_PC_device_parameters));
            break;
    }

```



```

err = mbc_master_start();
if (err != ESP_OK) {
    ESP_LOGE(TAG, "mb controller start fail, err=%x.", err);
}
return err;
}

```

- Se asignaron tres registros adicionales de la tabla de configuración, a fin de poder configurar la interfaz para el dispositivo esclavo (TCP/IP o RTU) y configurar la velocidad de la comunicación serial para la UART2. Dichas asignaciones se hicieron en el archivo remota_globals.h:

```

#define CFG_REMOTA_LOG_LEVEL s3Tables.configTbl[0][49]

#define CFG_RUN_PGM s3Tables.configTbl[0][0] //Run mode or Config mode
#define CFG_OP_MODE s3Tables.configTbl[0][1] //Operation mode 0-5
#define CFG_IP0 &s3Tables.configTbl[0][2] //IP address for Ethernet 0
#define CFG_IP1 &s3Tables.configTbl[0][4] //IP address for Ethernet 1
#define CFG_IP2 &s3Tables.configTbl[0][6] //IP address for WiFi (Station mode)
#define CFG_IP3 &s3Tables.configTbl[0][8] //IP address for WiFi (AP mode)
#define CFG_GW &s3Tables.configTbl[0][10] //Gateway
#define CFG_DHCP s3Tables.configTbl[0][12] //DHCP on/off

#define CFG_MB_MASTER_INTERFACE s3Tables.configTbl[0][13] //Modbus master interface (TCP or RTU) //4003
#define CFG_MB_MASTER_BAUDRATE_H s3Tables.configTbl[0][14] //Modbus Master RTU Baudrate High Register //4003
#define CFG_MB_MASTER_BAUDRATE_L s3Tables.configTbl[0][15] //Modbus Master RTU Baudrate Low Register //4003
#define CFG_MB_SLAVE_INTERFACE s3Tables.configTbl[0][16] //Modbus slave interface (TCP or RTU) //4003
#define CFG_MB_SLAVE_BAUDRATE_H s3Tables.configTbl[0][17] //Modbus Slave RTU Baudrate High Register //4003
#define CFG_MB_SLAVE_BAUDRATE_L s3Tables.configTbl[0][18] //Modbus Slave RTU Baudrate Low Register //4003

```

- La función `modbus_master_init()` tiene en cuenta el registro `CFG_MB_MASTER_INTERFACE` para decidir si inicia la pila modbus maestro TCP o la pila modbus maestro RTU, dependiendo del valor del registro; así, si:
 - `CFG_MB_MASTER_INTERFACE = 1`, se inicia modbus master TCP
 - `CFG_MB_MASTER_INTERFACE = 0`, se inicia modbus master RTU.
- Para el caso del modbus RTU, se utiliza la UART2, a través de los pines 41 para TX y 42 para RX. (Hard-coded, se definirán macros posteriormente). Además, para configurar la velocidad, se utilizan los valores de la tabla de configuración `CFG_MB_MASTER_BAUDRATE_H` y `CFG_MB_MASTER_BAUDRATE_L`. La configuración para la velocidad se hace utilizando la máscara siguiente:

```

comm_info.port = 2; // Serial port number
comm_info.mode = MB_MODE_RTU; // Modbus mode of communication (MB_MODE_RTU or MB_MODE_ASCII)
comm_info.baudrate = ((uint32_t)CFG_MB_MASTER_BAUDRATE_H << 16) | CFG_MB_MASTER_BAUDRATE_L; // Modbus
comm_info.parity = MB_PARITY_NONE; // parity option for serial port

ESP_ERROR_CHECK(mbc_master_setup((void*)&comm_info));

```

- Hasta el momento el programa no tiene en cuenta el cambio de las demás configuraciones de la comunicación serial, como son los bits de parada y la paridad.
- Para realizar la configuración, se incorporaron los casos necesarios en la tarea `mb_check_event_task()`, de manera que al cambiarse los valores de estos registros, se

realiza el procesamiento necesario para guardar dichos valores en la memoria flash. Además, en caso de cambiarse el tipo de interfaz, RTU o TCP, se coloca la bandera `resetRequired = 1`; lo que ocasiona que el microcontrolador se reinicie una vez que se cambia `CFG_RUN_MODE = 1`; es decir, al configurar la remota en modo RUN, si `resetRequired = 1`, se realiza un reinicio para aplicar la nueva configuración.

- La función utiliza uno de tres diccionarios de características de los dispositivos esclavos, los cuales están definidos en un nuevo archivo: `mb_dictionaries.h`; existen tres diccionarios diferentes, en función de proporcionar las capacidades de conexión particulares con los dispositivos esclavos de los métodos de producción de pozos con Bombeo Mecánico, pozos con Bomba Electrosurgible y pozos con Bomba de Cuidad progresiva. La selección del diccionario apropiado se realiza mediante una sentencia `switch`, la cual verifica el registro `CFG_OP_MODE` y selecciona el diccionario apropiado según el caso.
- Se creó una nueva tarea del sistema operativo, `mb_master_poll_task()`; la cual se encargará de efectuar las consultas (polls) y escrituras, desde y hacia el dispositivo esclavo. La creación de esta tarea, también está condicionada al modo de operación, para crearla solamente en caso de ser necesaria la comunicación con el dispositivo esclavo. También se dispuso de una bandera global `mb_master_task_created`, para guardar si la tarea ha sido creada y poder pausar o reanudar la misma.

```
modbus_slave_init();

// Starts modbus master stack and modbus master poll task only if required
if ((CFG_OP_MODE == 2) || (CFG_OP_MODE == 3) || (CFG_OP_MODE == 4)){
    //Create and start modbus master poll task:
    xTaskCreatePinnedToCore(mb_master_poll_task,
        "mb_master_poll_task",
        STACK_SIZE,
        NULL,
        (UBaseType_t) 1U,          //Priority Level 0
        &xMBMasterPollTaskHandle,
        1);

    mb_master_task_created = 1;
}
```

- Dicha tarea utiliza la información del diccionario de características correspondiente al dispositivo esclavo que se está utilizando según el modo de operación para acceder a los datos del esclavo y realizar la lectura y escritura de los registros; los valores leídos (y escritos), se guardan en (o se toman de) los correspondientes registros de las tablas creadas en memoria RAM, los cuales, para el caso de Bombeo Mecánico, por ejemplo, son:

```

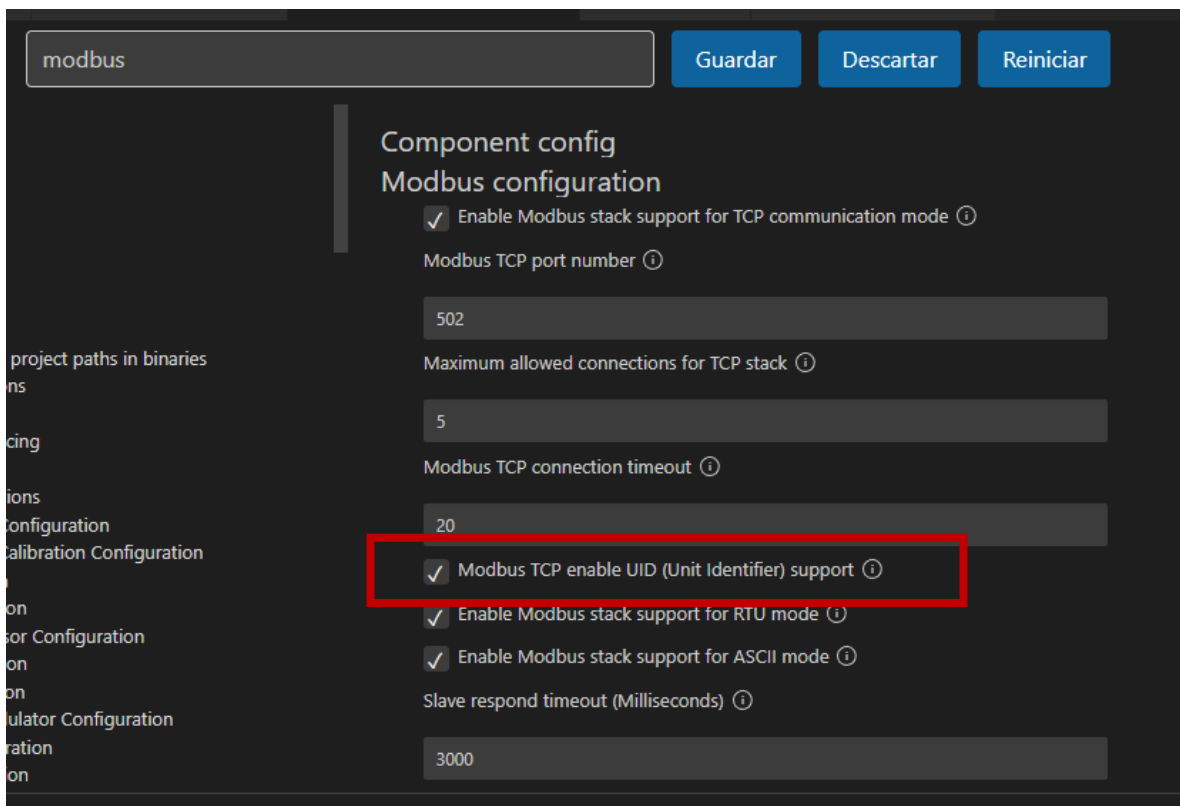
#define MP_IR_ETM  s3Tables.mbTblFloat[0][0]      // Voltaje de motor
#define MP_IR_ITM  s3Tables.mbTblFloat[0][1]      // Corriente de motor
#define MP_IR_JTM  s3Tables.mbTblFloat[0][2]      // Potencia de motor

#define MP_HR_SCM  s3Tables.mbTblFloat[0][3]      // Ajuste de velocidad

#define MP_COIL_HS1 (s3Tables.mbTbl8bit[0][0] & 0x01) // Encendido/apagado de bombeo

```

- La tarea decide qué consultas hacer y qué datos escribir al esclavo, dependiendo del modo de operación. Dicha selección se hace a través de una sentencia switch.
- Agregada la lógica para suspender o reanudar la nueva tarea al cambiar el estado del registro CFG_RUN_PGM. La suspensión o reactivación de la tarea toma en cuenta la bandera mb_master_task_created para determinar si la tarea ha sido creada.
- Desde el menú de configuración (menú-config), se activó la opción: Modbus TCP enable UID (Unit Identifier) support. A fin de que el programa tome en cuenta el ID del esclavo en modbus TCP; de lo contrario, el programa no es capaz de comunicarse de manera efectiva con el simulador ModSim32.



- En la función de inicialización de la pila modbus esclavo (mb_slave_init()), en la definición de la estructura que contiene la información de la comunicación, se agregó el campo .slave_uid = 1; a fin de configurar el UID del dispositivo esclavo y lograr que el simulador ModScan32 pueda comunicarse con la remota. Esto se hizo necesario al activar el soporte del UID desde el menú-config.

- Se realizó la conexión física del dispositivo a la PC a través de un nuevo módulo USB-Serial, verificándose la capacidad de conexión a través de modbus RTU con el simulador ModSim32.
- Se verifica la funcionalidad del código incorporado a través de los simuladores ModScan32 y ModSim32 simultáneamente; obteniéndose resultados satisfactorios en la comunicación para ambos protocolos y comprobando la lectura y escritura de datos desde el ModScan32 hasta el ModSim32 y viceversa. De esta manera se comprueba la capacidad del sistema de transmitir información desde y hacia el sistema SCADA y los equipos en campo.
- Los datos que se intercambian con el equipo esclavo, también aparecen ahora reflejados en la salida por consola:

```

I (3542154) Remota-Main: Pozos de Bombeo Mecánico
I (3542164) Remota-Main: Variables analógicas de entrada leídas desde el módulo de E/S:
I (3542174) Remota-Main: Presión de la línea de producción      (PTL): 3015
I (3542174) Remota-Main: Temperatura de la línea de producción (TTL): 3016
I (3542184) Remota-Main: Presión del casing o anular            (PTA): 3009
I (3542194) Remota-Main: Presión de cabezal                    (PTC): 3005
I (3542204) Remota-Main: Golpe por minuto (spm)                (SPM): 3009
I (3542204) Remota-Main: Ángulo de la viga                      (ZT): 3009
I (3542214) Remota-Main: Carga de la sarta                      (WT): 3010

I (3542214) Remota-Main: Variables analógicas de entrada en unidades de ingeniería:
I (3542224) Remota-Main: Presión de la línea de producción      (PTL): 2.499
I (3542234) Remota-Main: Temperatura de la línea de producción (TTL): 2428.865
I (3542244) Remota-Main: Presión del casing o anular            (PTA): 3024.000
I (3542254) Remota-Main: Presión de cabezal                    (PTC): -3024.000
I (3542254) Remota-Main: Golpe por minuto (spm)                (SPM): 3024.000
I (3542264) Remota-Main: Ángulo de la viga                      (ZT): -3024.000
I (3542274) Remota-Main: Carga de la sarta                      (WT): 3024.000

I (3542274) Remota-Main: Variables digitales de entrada leídas desde el módulo de E/S:
I (3542284) Remota-Main: Alarma por falla de sistema            (YA1): 0
I (3542294) Remota-Main: Alarma de intruso                      (YA2): 0
I (3542294) Remota-Main: Alarma por gas tóxico                 (GZA): 0
I (3542304) Remota-Main: Alarma por falla de voltaje AC         (EAAC): 0
I (3542314) Remota-Main: Alarma por falla de voltaje DC         (EADC): 0

I (3542314) Remota-Main: Variables digitales de salida leídas desde el módulo de E/S:
I (3542324) Remota-Main: Encendido/apagado de bombeo           (HS1): 0

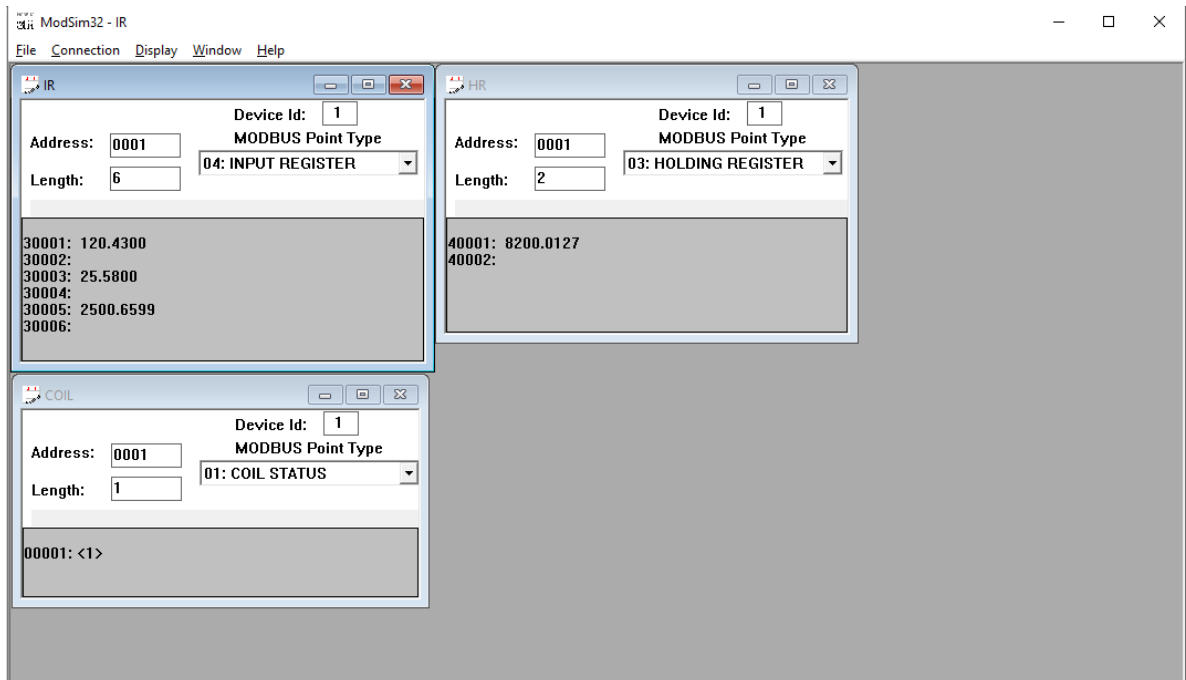
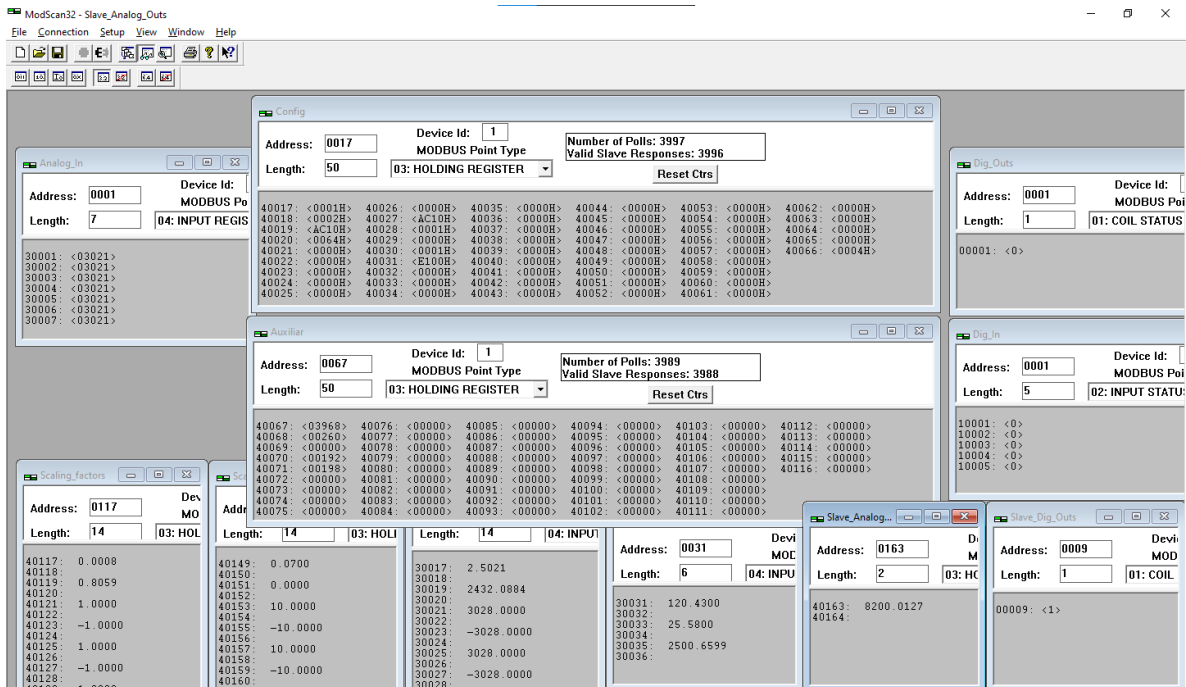
I (3542334) Remota-Main: Variables modbus (input registers) leídas desde el equipo esclavo:
I (3542344) Remota-Main: Voltaje de motor                       (ETM): 120.430
I (3542354) Remota-Main: Corriente de motor                     (ITM): 25.580
I (3542354) Remota-Main: Potencia de motor                      (JTM): 2500.660

I (3542364) Remota-Main: Variables modbus (holding registers) leídas desde el equipo esclavo:
I (3542374) Remota-Main: Ajuste de velocidad                   (SCM): 8200.013

I (3542374) Remota-Main: Variables modbus (COILS) leídas desde el equipo esclavo:
I (3542384) Remota-Main: Encendido/apagado de bombeo           (HS1): 1

```

- Se comprobó la funcionalidad del protocolo modbus maestro utilizando cualquiera de las dos interfaces (TCP o RTU).



- Se comprobó que tanto la inicialización de la pila modbus maestro, como la tarea mb_master_poll_task, se ejecutan solo en los casos correctos.

Cambios realizados hasta el 09-09-2023:

- ✓ Agregado soporte para la comunicación con el sistema SCADA a través del protocolo Modbus RTU, utilizando la UART1 a través de los pines TX=15 y RX=16.
 - Los pines originalmente asignados eran TX=17 y RX=18, sin embargo, se hizo necesario cambiarlos ya que al realizar las pruebas la comunicación no funcionó haciendo uso de los mismos, aún está pendiente por determinar la causa.
 - Para lograr incorporar esta funcionalidad, se asignaron nuevos registros de la tabla de configuración, para selección de la interfaz a utilizar para la comunicación Modbus slave, así como para configurar la velocidad de la transmisión serial.

```
#define CFG_REMOTA_LOG_LEVEL s3Tables.configTbl[0][49]

#define CFG_RUN_PGM s3Tables.configTbl[0][0]           //Run mode or Config mode
#define CFG_OP_MODE s3Tables.configTbl[0][1]           //Operation mode 0-5
#define CFG_IP0 &s3Tables.configTbl[0][2]              //IP address for Ethernet 0
#define CFG_IP1 &s3Tables.configTbl[0][4]              //IP address for Ethernet 1
#define CFG_IP2 &s3Tables.configTbl[0][6]              //IP address for WiFi (Station mode)
#define CFG_IP3 &s3Tables.configTbl[0][8]              //IP address for WiFi (AP mode)
#define CFG_GW &s3Tables.configTbl[0][10]              //Gateway
#define CFG_DHCP s3Tables.configTbl[0][12]             //DHCP on/off
#define CFG_MB_MASTER_INTERFACE s3Tables.configTbl[0][13] //Modbus master interface (TCP or RTU) //40030
#define CFG_MB_MASTER_BAUDRATE_H s3Tables.configTbl[0][14] //Modbus Master RTU Baudrate High Register //40031
#define CFG_MB_MASTER_BAUDRATE_L s3Tables.configTbl[0][15] //Modbus Master RTU Baudrate Low Register //40032
#define CFG_MB_SLAVE_INTERFACE s3Tables.configTbl[0][16] //Modbus slave interface (TCP or RTU) //40033
#define CFG_MB_SLAVE_BAUDRATE_H s3Tables.configTbl[0][17] //Modbus Slave RTU Baudrate High Register //40034
#define CFG_MB_SLAVE_BAUDRATE_L s3Tables.configTbl[0][18] //Modbus Slave RTU Baudrate Low Register //40035
```

- La utilización de estos registros se maneja de la misma forma que para el caso de Modbus master, utilizando el registro CFG_MB_SLAVE_INTERFACE para decidir el tipo de interfaz y los registros CFG_MB_SLAVE_BAUDRATE_H y CFG_MB_SLAVE_BAUDRATE_L para establecer la velocidad.
- Para implementar esta nueva funcionalidad se modificó la función modbus_slave_init(), la cual tiene en cuenta las dos posibilidades para el registro CFG_MB_SLAVE_INTERFACE.
- Se implementó la lógica necesaria en el chequeador de eventos modbus, para que cuando se cambie alguno de estos valores en los registros, se active la bandera resetRequired, produciéndose un reinicio al volver del modo PROGRAM al modo RUN y aplicando la nueva configuración.
- Se realizó la conexión física del dispositivo a la PC a través de un nuevo módulo USB-Serial, verificándose la capacidad de conexión a través de modbus RTU con el simulador ModScan32.
- Se verificó la operatividad de la nueva funcionalidad con diferentes configuraciones de velocidad y realizando los cambios entre una interfaz y otra.
- Las pruebas preliminares arrojaron una alta latencia en la comunicación serial con el simulador ModScan32, lo cual se hace evidente al tener todas las tablas correspondientes al mapa modbus; sin embargo, a pesar de la latencia, la comunicación se realiza sin errores y ésta se hace más fluida al disminuir la cantidad de tablas modbus que se están consultando al mismo tiempo desde el simulador.

- ✓ Desarrollada la lógica concerniente al objetivo propuesto concerniente al cálculo del volumen diario de gas inyectado al pozo, para el caso particular de pozos de Gas Lift. Para esto, se trabajó con el siguiente enfoque:

- El programa deberá ser capaz de contabilizar el volumen acumulado de gas, a partir de la medición de flujo recibida desde el módulo de E/S, como una variable analógica de entrada; la cual se encuentra identificada en el programa a través de:
 - GL_AI_FTGL: (Flujo de gas al pozo en cuentas).
 - GL_SV_FTGL: (Flujo de gas al pozo en unidades de ingeniería).
- La unidad de ingeniería tomada en cuenta para realizar los cálculos es Miles de Pies Cúbicos de Gas al Día MPCGD o MCFD (en inglés). Cualquier otra unidad que se utilice, deberá ser ajustada utilizando la funcionalidad de escalamiento, asignando los valores correspondientes al factor de escalamiento (GL_SF_FTGL) y al offset de escalamiento (GL_SO_FTGL).
- Se hizo necesario aplicar un filtrado a la señal transmitida de flujo, utilizando un modelo de filtro de primer orden con seguimiento al valor instantáneo; para esto, se utilizó la siguiente implementación en tiempo discreto:

$$Vs(t) = Vs(t - 1) + \alpha * (Vin(t) - Vs(t - 1))$$

- $0 < \alpha < 1$ es la constante de tiempo del filtro y se puede utilizar para ajustar su comportamiento, para valores bajos de α , se obtiene una respuesta con mayor retardo y mayor calidad de filtrado, para valores mayores de α , se obtiene una respuesta más rápida con mayor presencia de ruido.
- Se dispuso de la fórmula del trapecio para calcular la acumulación de gas, la cual toma en cuenta el valor anterior del flujo y lo promedia con el valor actual, así:

$$Volumen_{intervalo \Delta t} = \frac{Flujo_t + Flujo_{t-1}}{2} * \Delta t$$

- El programa acumulará el volumen de gas durante un plazo de 24 horas, manteniendo un histórico de 48 horas y 72 horas; asimismo, se manejará una variable adicional para almacenar el volumen proyectado (estimado) en función del flujo instantáneo (filtrado).
- ✓ Para lograr este objetivo se dispuso la creación de un Timer de FreeRTOS, el cual se crea de forma similar a una tarea, salvo que se ejecuta a intervalos de tiempo regulares, llamando a una función al cumplirse la temporización. Dicho timer se crea durante la fase de inicio de la tarea principal app_main() y su creación está condicionada al modo de operación, de manera que se creará únicamente para el modo de operación correspondiente a pozos de Gas Lift.

```
if (CFG_OP_MODE == 1) {
    //Creates the timer for GAS Volume accumulation, only in Gas Lift OP Mode
    xGL_Timer = xTimerCreate("GL_Timer",
        pdMS_TO_TICKS(CFG_GL_TMR_INTERVAL),
        pdTRUE,
        NULL,
        GLTimerCallback);
    xTimerStart(xGL_Timer, pdMS_TO_TICKS(1000));
    ESP_LOGI(TAG, "Gas accumulator timer started with interval %u ms", CFG_GL_TMR_INTERVAL);
}
```

- ✓ Se definieron las macros siguientes para almacenar los resultados de los cálculos; la tabla correspondiente en RAM se crea al momento de crear el mapa de memoria modbus, de manera que estos registros son accesibles desde el SCADA.

```
#define GL_FQ24      s3Tables.mbTblFloat[0][0] // Volumen total diario de gas (24H)
#define GL_FQ48      s3Tables.mbTblFloat[0][1] // Volumen total de gas en 48H
#define GL_FQ72      s3Tables.mbTblFloat[0][2] // Volumen total de gas en 72H

#define GL_FQ24_PROY s3Tables.mbTblFloat[0][3] // Volumen proyectado diario de gas (24H)
```

- ✓ Se asignaron los siguientes registros de la tabla de configuración para almacenar el intervalo de temporización del timer y el valor de la constante de tiempo para el filtro de primer orden:

```
#define CFG_GL_TMR_INTERVAL s3Tables.configTbl[0][21] // Intervalo del timer para el acumulador de volumen diario de
#define CFG_GL_FILTER_ALPHA s3Tables.configTbl[0][22] // Constante de tiempo para el filtro de primer orden aplicado
#define MS24H 86400000 // Milisegundos en 24H
```

- ✓ Se creó la macro MS24H, para hacer referencia al número de milisegundos en 24H.
- ✓ El registro CFG_GL_TMR_INTERVAL hace referencia a la cantidad de milisegundos que han pasado desde la ejecución previa del timer; así que para el cálculo se efectúan los siguientes pasos (ver código en la implementación de la función GLTimerCallBack()).
 - Se acumulan los milisegundos transcurridos en la variable: msCounter24.

```
msCounter24 += CFG_GL_TMR_INTERVAL;
```

- Se aplica el filtrado sobre la entrada de flujo, mediante la implementación del filtro de primer orden:

```
// First order filter
ftglFiltered = ftglFiltered + ((float)CFG_GL_FILTER_ALPHA/1000) * (GL_SV_FTGL - ftglFiltered);
```

- Se aplica la fórmula del trapecio, considerando el valor anterior del flujo:

```
// Considering the last flow calculated and taking the average
ftglFiltered = (last_ftgl + ftglFiltered)/2;
last_ftgl = ftglFiltered;
```

- Se calculan el valor proyectado en base al flujo obtenido después de filtrar la señal:

```
// Calculate estimated volume for 24H
GL_FQ24_PROY = ftglFiltered;
```

- Se realiza la acumulación del volumen de gas durante el intervalo del timer, (integración para obtener finalmente el área bajo la curva del flujo instantáneo). Como la unidad utilizada es MCFD, la unidad de tiempo se convierte a segundos, dividiendo el valor del flujo entre el número de segundos en un día. Al hacer esta división se obtiene el valor

en MCF/s; finalmente este valor se multiplica por el número de segundos transcurridos en el intervalo del timer. El valor se acumula en el registro correspondiente:

```
// Accumulate gas volume in the timer interval
float accum = (ftglFiltered / 86400) * (float)CFG_GL_TMR_INTERVAL / 1000;
GL_FQ24 += accum;
```

- Se implementa la lógica para manejar el cierre del intervalo de integración, efectuando un corrimiento de los registros que almacenan el volumen acumulado de gas en 24H, 48H y 72H:

```
if (msCounter24 >= MS24H){
    GL_FQ72 = GL_FQ48;
    GL_FQ48 = GL_FQ24;
    GL_FQ24 = 0;
    msCounter24 = 0;
}
```

- ✓ Se implementó la lógica necesaria para que al modificar el registro CFG_GL_TMR_INTERVAL, se active la bandera resetRequired, de modo que se produce un reinicio al volver al modo RUN. Dicho reinicio es necesario debido a que, al cambiar este valor, debe volver a crearse el timer. Para el valor CFG_GL_FILTER_ALPHA, aunque es necesario parar al modo PROGRAM para modificarlo desde SCADA, no es necesario efectuar un reinicio, ya que el nuevo valor tiene efecto inmediatamente es escrito.
 - El valor de CFG_GL_TMR_INTERVAL debe ser mayor o igual que 10ms para que el programa lo considere válido (valores de temporización menores hacen que el programa falle).
 - El valor de CFG_GL_FILTER_ALPHA debe ser menor que 1000 para que el programa lo considere válido; esta constante de tiempo en la implementación del filtro se divide por 1000, así que dicho factor $1/\alpha$ es menor que la unidad.
- ✓ Agregada la lógica necesaria para detener el timer en modo PROGRAM y volverlo a iniciar en modo RUN.

Cambios realizados hasta el 10-09-2023:

- ✓ Asignados los registros de la tabla de configuración necesarios para establecer la dirección IP del equipo esclavo, utilizada en caso de comunicación modbus master TCP: (Los registros de la tabla de configuración fueron reasignados, ver la tabla actualizada más adelante)

```
#define CFG_SLAVE_IP &s3Tables.configTbl[0][19] //IP address for slave field device (Modbus TCP)
```

- De manera similar a la forma como se trabajó la dirección IP del módulo ethernet, los registros s3Tables.configTbl[0][19] y s3Tables.configTbl[0][20], contienen cada uno 2 octetos de la dirección IP del equipo esclavo.
- ✓ En la función modbus_master_init(), la dirección IP se recupera de estos registros antes de configurarse en la estructura necesaria para la configuración de la pila modbus master TCP:

```

if (CFG_MB_MASTER_INTERFACE)
{
    void* master_handler = NULL; // Pointer to allocate interface structure
    // Initialization of Modbus master for TCP/IP
    err = mbc_master_init_tcp(&master_handler);
    if (master_handler == NULL || err != ESP_OK) {
        ESP_LOGE(TAG, "mb controller initialization fail.");
    }
}

char slaveIP_str[16] = {'\0'};
uint8_t IP0[4] = {0};
IP0[0] = *CFG_SLAVE_IP >> 8;
IP0[1] = *CFG_SLAVE_IP & 0xFF;
IP0[2] = *(CFG_SLAVE_IP+1) >> 8;
IP0[3] = *(CFG_SLAVE_IP+1) & 0xFF;
sprintf(slaveIP_str, "%hhu.%hhu.%hhu.%hhu", IP0[0], IP0[1], IP0[2], IP0[3]);

const char* slave_ip_address_table[3] = {
    slaveIP_str, // Address corresponds to UID1 and set to predefined value by user
    NULL, // end of table
};

/* const char* slave_ip_address_table[3] = {
    "172.16.0.4", // Address corresponds to UID1 and set to predefined value by user
    NULL, // end of table
}; */

comm_info.ip_port = 502; // Modbus TCP port number (default = 502)
comm_info.ip_addr_type = MB_IPV4; // version of IP protocol
comm_info.ip_mode = MB_MODE_TCP; // Port communication mode
comm_info.ip_addr = (void*)slave_ip_address_table; // assign table of IP addresses
comm_info.ip_netif_ptr = eth_netif; // esp_netif_ptr pointer to the corresponding network interf.

ESP_ERROR_CHECK(mbc_master_setup((void*)&comm_info));

```

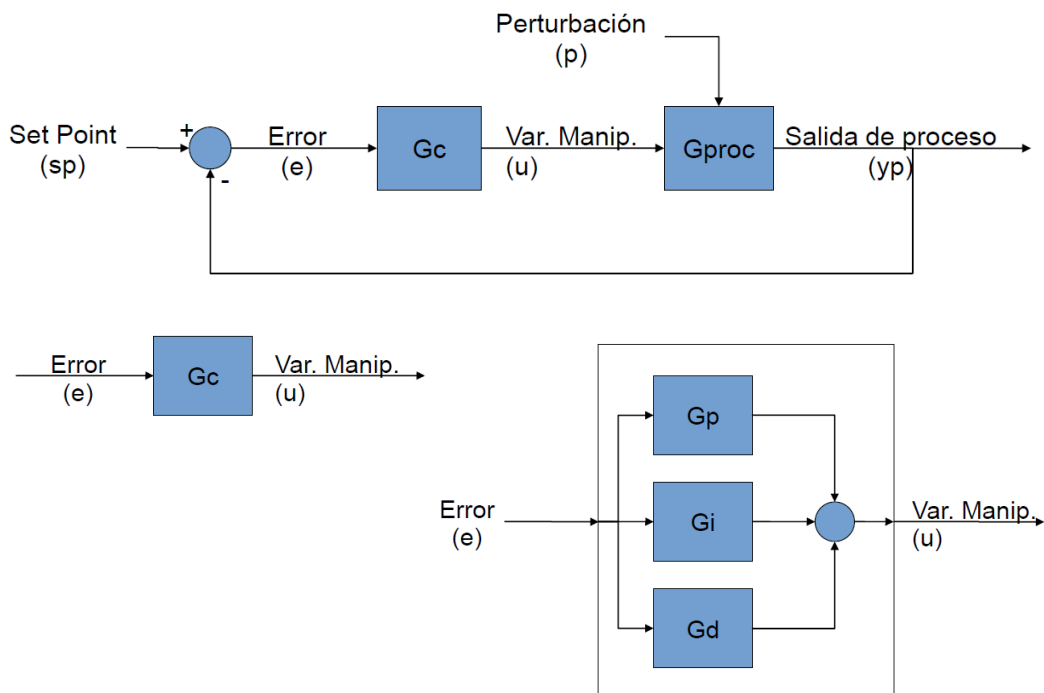
- ✓ Se agregó la lógica necesaria en el chequeador de eventos modbus para establecer la bandera resetRequired en caso de modificarse cualquiera de estos registros, produciéndose un reinicio al volver al modo RUN.
- ✓ La tabla de configuración se modificó ligeramente, los registros actualizados se muestran en la siguiente tabla:

Registros de la tabla de configuración: s3Tables.configTbl[0][i]			
Nombre	Registro (índice)	Dirección Modbus	Descripción
CFG_RUN_PGM	0	40017	Modo de ejecución o modo de programación
CFG_OP_MODE	1	40018	Selección del método de producción
CFG_IP0	2	40019	Dirección IP Ethernet 0
	3	40020	
CFG_IP1	4	40021	Dirección IP Ethernet 1
	5	40022	
CFG_IP2	6	40023	Dirección IP WiFi (Station)
	7	40024	
CFG_IP3	8	40025	Dirección IP WiFi (AP)
	9	40026	
CFG_GW	10	40027	Network Gateway
	11	40028	

Registros de la tabla de configuración: <code>s3Tables.configTbl[0][i]</code>			
Nombre	Registro (índice)	Dirección Modbus	Descripción
CFG_DHCP	12	40029	Dirección IP estática o dinámica
CFG_MB_MASTER_INTERFACE	13	40030	Modbus maestro TCP o RTU
CFG_MB_MASTER_BAUDRATE_H	14	40031	Velocidad UART2 (Modbus master)
CFG_MB_MASTER_BAUDRATE_L	15	40032	
CFG_MB_SLAVE_INTERFACE	16	40033	Modbus esclavo TCP o RTU
CFG_MB_SLAVE_BAUDRATE_H	17	40034	Velocidad UART1 (Modbus slave)
CFG_MB_SLAVE_BAUDRATE_L	18	40035	
CFG_SLAVE_IP	19	40036	Dirección IP del equipo esclavo (Modbus TCP)
	20	40037	
CFG_GL_TMR_INTERVAL	21	40038	Tiempo de ejecución timer de acumulación de gas
CFG_GL_FILTER_ALPHA	22	40039	Constante de tiempo del filtro de primer orden para señal de flujo de gas (multiplicada x 1000)
	23	40040	
	24	40041	
	25	40042	
	26	40043	
	27	40044	
	28	40045	
	29	40046	
	30	40047	
	31	40048	
	32	40049	
	33	40050	
	34	40051	
	35	40052	
	36	40053	
	37	40054	
	38	40055	
	39	40056	
	40	40057	
	41	40058	
	42	40059	
	43	40060	
	44	40061	
	45	40062	
	46	40063	
	47	40064	
	48	40065	
CFG_REMOTA_LOG_LEVEL	49	40066	Nivel de log para mensajes en consola serial

Cambios realizados hasta el 13-10-2023:

- ✓ Agregada la funcionalidad de controlador PID para flujo de gas en pozos de tipo Gas Lift.
 - Creado el timer GL_PIDTimer durante la ejecución inicial de la tarea app_main(). La creación del timer está condicionada al modo de operación CFG_OP_MODE = 1 y el intervalo de ejecución del timer está dado por la macro CFG_GL_PID_TMR_INTERVAL; la cual está asociada al registro s3Tables.configTbl[0][23] de la tabla de configuración. El valor de este registro configura el tiempo en milisegundos que tardará en llamarse periódicamente la función de callback asociada al timer: GLTimerPIDCallBack().
 - Agregadas las lógicas para iniciar o pausar el timer dependiendo del modo de ejecución (registro CFG_RUN_PGM). El timer se inicia si CFG_RUN_PGM = 1 y su estado actual es detenido, o se detiene si CFG_RUN_PGM = 0 y su estado actual es activo.
 - Agregado código dentro de la función de callback GLTimerPIDCallBack() para medir el tiempo que tarda entre una ejecución y la siguiente, dicho código utiliza el último registro de la tabla auxiliar para almacenar el valor del tiempo medido (s3Tables.auxTbl[0][49]).
 - Agregado el código necesario dentro de la función de callback para realizar los cálculos del controlador PID; este código fue desarrollado sobre el siguiente enfoque:



$$G_p = K_p$$

$$G_i = \frac{K_i}{s}$$

$$G_d = \frac{K_d * s}{\frac{1}{N} * s + 1}$$

- El cálculo de la 1ra derivada se realiza mediante una aproximación mediante diferencias finitas con un paso hacia atrás:

$$y' \cong \frac{y_{(i)} - y_{(i-1)}}{\Delta t}$$

- Después de desarrollar matemáticamente las ecuaciones, el código del controlador PID es el siguiente:

```
void GLTimerPIDCallback(TimerHandle_t pxTimer){
    time2 = esp_timer_get_time();
    s3Tables.auxTbl[0][49] = (time2 - time1)/1000;

    // Error calculation
    PID_e = CFG_GL_PID_SP - GL_AI_FTGL;

    // Proportional Gain (Gp)
    PID_up = CFG_GL_PID_KP * PID_e;

    // Integral Gain (Gi)
    PID_ui = lastPID_ui + (float)CFG_GL_PID_KI * PID_e *
((float)CFG_GL_PID_TMR_INTERVAL / 1000);

    lastPID_ui = PID_ui; // Keep track of last ui

    // Derivative Control (with N Filter) (Gd)
    PID_ud = ( PID_ud + CFG_GL_PID_KD * CFG_GL_PID_N * (PID_e - lastPID_e) )
/ ( 1 + CFG_GL_PID_N * ((float)CFG_GL_PID_TMR_INTERVAL / 1000));

    lastPID_e = PID_e; // Keep track of last error vaue

    // PID output:
    PID_u.floatValue = CFG_GL_PID_CP * PID_up + CFG_GL_PID_CI * PID_ui +
CFG_GL_PID_CD * PID_ud;
    GL_AO_FCV_L = PID_u.uint16Values.low;
    GL_AO_FCV_H = PID_u.uint16Values.high;

    time1 = esp_timer_get_time();
}
```

- Se asignaron los siguientes registros de la tabla de configuración para el funcionamiento del controlador PID: (Los registros se detallarán en la tabla de configuración actualizada)
 - CFG_GL_PID_KP (Ganancia proporcional) Valor multiplicado * 1000

- CFG_GL_PID_KI (Ganancia integral) Valor multiplicado * 1000
 - CFG_GL_PID_KD (Ganancia derivativa) Valor multiplicado * 1000
 - CFG_GL_PID_N (Constante de filtro N) Valor multiplicado * 1000
 - CFG_GL_PID_CP (Coeficiente de activación proporcional)
 - CFG_GL_PID_CI (Coeficiente de activación integral)
 - CFG_GL_PID_CD (Coeficiente de activación derivativo)
- La salida del controlador PID se calcula en precisión de punto flotante, en la variable PID_u, que es de tipo definido por usuario FloatSplit; el cual es una unión, que permite acceder a los registros alto y bajo del valor de 32 bit, así:

```
union FloatSplit {
    float floatValue;
    struct {
        uint16_t low;  // Registro bajo
        uint16_t high; // Registro alto
    } uint16Values;
};
```

De esta manera el valor es calculado en PID_u.floatValue y transmitido al módulo I/O a través de PID_u.uint16Values.low y PID_u.uint16Values.high. Se utilizaron las macros GL_AO_FCV_L y GL_AO_FCV_H para definir los registros de la tabla de valores analógicos de salida s3Tables.anTbl[1][0] y s3Tables.anTbl[1][1] respectivamente.

- Agregada la lógica para la configuración del intervalo de ejecución del timer en la tarea mb_event_check_task(); de manera que al cambiar el valor del registro CFG_GL_TMR_INTERVAL desde el SCADA, se valida que el valor ingresado sea mayor a 10ms y se establece la bandera resetRequired = 1, para provocar un reinicio al volver al modo de ejecución CFG_RUN_PGM = 1.
 - Realizada la validación de los registros CFG_GL_PID_CP, CFG_GL_PID_CI y CFG_GL_PID_CD para que solamente acepten valores 0 ó 1.
- ✓ Para efectos de prueba del controlador PID, se agregó el código correspondiente a la simulación de un pozo mediante un modelo de primer orden; que es ejecutado por el microcontrolador del módulo de I/O (ESP32 WROOM). Dicho modelo, se realizó mediante el siguiente enfoque:

$$G_{pr}^o = \frac{K}{(\tau s + 1)} = \frac{Y_p}{U}$$

En tiempo discreto, por medio de la aproximación de la derivada utilizando diferencias finitas, esta ecuación se convierte en:

% Modelo del proceso orden 1

$$yp(i) = (\text{Tau} * yp(i-1) + K_{pert} * p(i) + K * u(i) * dt) / (dt + \text{Tau});$$

- Se agregó el código de creación de un timer que controla la simulación del proceso; la creación de dicho timer se realiza en la ejecución inicial de la tarea app_main():

```
TimerHandle_t xGLProc_Timer = xTimerCreate("GL_ProcessTimer",
                                           pdMS_TO_TICKS(10),
                                           pdTRUE,
                                           NULL,
                                           GL_well_process);
xTimerStart(xGLProc_Timer, pdMS_TO_TICKS(1000));
ESP_LOGI(TAG, "Gas Lift Process simulation timer started with
interval %u ms", 10);
```

- El timer llama a la función de callback GL_well_process(); la cual realiza los cálculos pertinentes para proporcionar la salida (flujo de gas al pozo) dependiendo del valor de control recibido en las variables:

```
u.uint16Values.low = IOTables.anTbl[1][0];
u.uint16Values.high = IOTables.anTbl[1][1];
```

- El código de la función GL_well_process() es el siguiente:

```
void GL_well_process(TimerHandle_t pxTimer){
    float p = IOTables.anTbl[0][0];

    u.uint16Values.low = IOTables.anTbl[1][0];
    u.uint16Values.high = IOTables.anTbl[1][1];

    //float u = (int16_t)IOTables.anTbl[1][0];

    yp = ((Tau * yp_ant) + (Kproc * u.floatValue * 0.01) + (Kpert * p)) /
    (Tau + 0.01);

    if(yp <= 0)
        yp = 0;
    else if (yp >= 65535)
        yp = 65535;

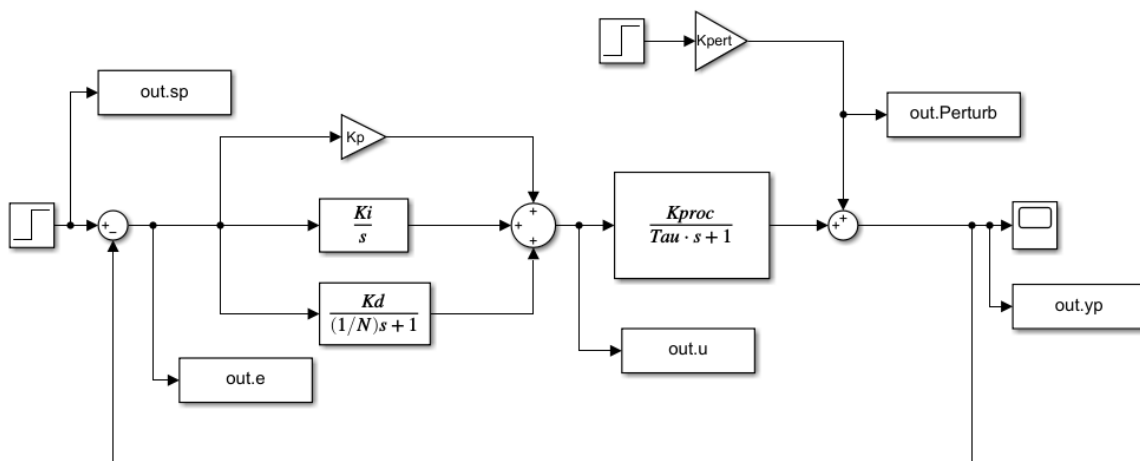
    IOTables.anTbl[0][5] = yp;

    yp_ant = yp;
}
```

- El código toma en cuenta el valor del registro IOTables.anTbl[0][0] para introducir una perturbación en el sistema; dicho valor corresponde al primer valor de la tabla de valores analógicos de entrada y proviene de la medición de tensión del canal ADC conectado al potenciómetro. De manera que al variar la posición del potenciómetro, se estará introduciendo un valor de perturbación al sistema para la simulación.
- También se han definido variables globales para establecer la ganancia del proceso, la ganancia de perturbación y la constante de tiempo del sistema de primer orden:

```
float Tau = 0.5;
float Kpert = 0.1;
float Kproc = 1;
float yp = 0;
float yp_ant = 0;
```

- La salida del proceso, es calculada en la variable yp y enviada de vuelta al microcontrolador ESP32-S3 mediante el registro IOTables.anTbl[0][5], correspondiente 6to valor analógico de entrada. Antes de ser enviada, se realiza la validación de su valor mediante un código saturador, que impide que el valor calculado sobrepase el rango de valores admitidos por un registro de 16 bit, evitando el desborde.
- Las pruebas del sistema de control PID fueron realizadas con la ayuda de una simulación previa en MATLAB, la cual se hizo mediante el paquete Simulink. Fue creado un modelo del sistema en diagrama de bloques y un código script .m que se ejecuta para definir los valores de los parámetros de la simulación y obtener gráficamente la respuesta del sistema. De esta manera, se puede verificar la entonación del controlador y validar los resultados con la respuesta obtenida mediante el controlador implementado.

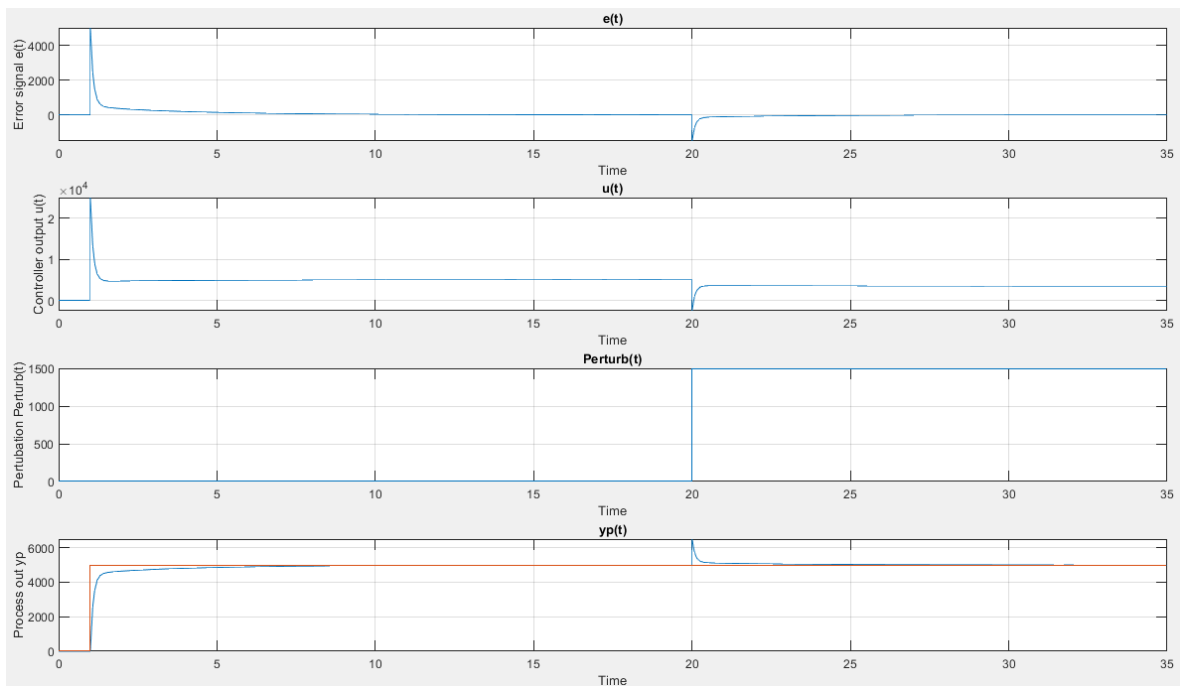



```

% PID Constants:
Kp = 5;
Ki = 2.5;
Kd = 2;
N = 1;
SP = 5000;

% Process constants:
Kproc = 1;
Tau = 0.5;
Kpert = 0.1;
Pert = 15000;

```



- Se observó un comportamiento del controlador muy similar al del modelo en MATLAB, obteniéndose diferentes tipos de respuesta (amortiguada y sobreamortiguada) para diferentes combinaciones de los parámetros del controlador (K_p , K_i y K_d), así como el filtro N y los coeficientes de activación C_p , C_i y C_d . El comportamiento similar se verificó en el tiempo de asentamiento y en el sobre disparo máximo obtenido.
- Se probaron varios valores de perturbación, a través de la variación de la posición del potenciómetro conectado al ESP32 WROOM; produciéndose el comportamiento esperado de acuerdo a la simulación y verificando la capacidad del controlador de llevar la variable nuevamente al setpoint en el tiempo de establecimiento previsto en la simulación de MATLAB.

✓ La tabla de configuración actualizada es la siguiente:

Registros de la tabla de configuración: s3Tables.configTbl[0][i]			
Nombre	Registro (índice)	Dirección Modbus	Descripción
CFG_RUN_PGM	0	40017	Modo de ejecución o modo de programación
CFG_OP_MODE	1	40018	Selección del método de producción
CFG_IP0	2	40019	Dirección IP Ethernet 0
	3	40020	
CFG_IP1	4	40021	Dirección IP Ethernet 1
	5	40022	
CFG_IP2	6	40023	Dirección IP WiFi (Station)
	7	40024	
CFG_IP3	8	40025	Dirección IP WiFi (AP)
	9	40026	
CFG_GW	10	40027	Network Gateway
	11	40028	
CFG_DHCP	12	40029	Dirección IP estática o dinámica
CFG_MB_MASTER_INTERFACE	13	40030	Modbus maestro TCP o RTU
CFG_MB_MASTER_BAUDRATE_H	14	40031	Velocidad UART2 (Modbus master)
CFG_MB_MASTER_BAUDRATE_L	15	40032	
CFG_MB_SLAVE_INTERFACE	16	40033	Modbus esclavo TCP o RTU
CFG_MB_SLAVE_BAUDRATE_H	17	40034	Velocidad UART1 (Modbus slave)
CFG_MB_SLAVE_BAUDRATE_L	18	40035	
CFG_SLAVE_IP	19	40036	Dirección IP del equipo esclavo (Modbus TCP)
	20	40037	
CFG_GL_TMR_INTERVAL	21	40038	Tiempo de ejecución timer de acumulación de gas
CFG_GL_FILTER_ALPHA	22	40039	Constante de tiempo del filtro de primer orden para señal de flujo de gas (multiplicada x 1000)
CFG_GL_PID_TMR_INTERVAL	23	40040	Tiempo de ejecución timer de PID
	24	40041	
	25	40042	
	26	40043	
	27	40044	
	28	40045	
	29	40046	
	30	40047	
	31	40048	
	32	40049	
	33	40050	
	34	40051	
	35	40052	
	36	40053	
	37	40054	

Registros de la tabla de configuración: <code>s3Tables.configTbl[0][i]</code>			
Nombre	Registro (índice)	Dirección Modbus	Descripción
	38	40055	
	39	40056	
	40	40057	
<code>CFG_GL_PID_SP</code>	41	40058	Setpoint para el controlador PID
<code>CFG_GL_PID_KP</code>	42	40059	Ganancia proporcional * 1000 (PID)
<code>CFG_GL_PID_KI</code>	43	40060	Ganancia integral * 1000 (PID)
<code>CFG_GL_PID_KD</code>	44	40061	Ganancia derivativa * 1000 (PID)
<code>CFG_GL_PID_N</code>	45	40062	Constante de filtro N * 1000 (PID)
<code>CFG_GL_PID_CP</code>	46	40063	Coeficiente de activación control proporcional (0 ó 1)
<code>CFG_GL_PID_CI</code>	47	40064	Coeficiente de activación control integral (0 ó 1)
<code>CFG_GL_PID_CD</code>	48	40065	Coeficiente de activación control derivativo (0 ó 1)
<code>CFG_REMOTA_LOG_LEVEL</code>	49	40066	Nivel de log para mensajes en consola serial

Cambios realizados hasta el 15-10-2023:

- ✓ Corregido error en la comunicación con los dispositivos esclavos, generado en modbus master TPC/IP, cuando el programa inicia sin que el dispositivo esclavo esté conectado. El programa se bloqueaba mostrando un mensaje de error repetitivo en consola hasta que se reiniciaba el microcontrolador. Para su solución, la declaración: `char slaveIP_str[16] = {'\0'};` fue movida al archivo `remota_globals.h`; ya que originalmente, al estar ubicada en la función `modbus_master_init()`, dicha variable dejaba de existir una vez que el programa salía de dicha función.
- ✓ Corregidos algunos problemas en la detección de la conexión del dispositivo esclavo; para esto se tomaron las siguientes acciones:
 - La bandera `modbus_master_initialized` ahora se coloca en 1 una vez que se ha ejecutado `mbc_master_start()`; en la función `modbus_master_init()`. Ya que en este punto es donde realmente se inicializa la pila modbus.
 - Se creó la bandera `modbus_master_connected`, para monitorear el estatus de la conexión con el dispositivo esclavo. Por defecto está establecido en 0 y se establece en 1 en la función `modbus_master_init()` siempre que la función devuelva `ESP_OK`.
 - Modificada la tarea `modbus_master_poll()`. El bucle infinito ahora está protegido por una decisión que verifica la bandera `modbus_master_connected`. En caso de valer 0, se ejecuta una solicitud modbus personalizada, con la finalidad de verificar el estado de la conexión, mediante la función `mbc_master_send_request()` y se verifica el código de error devuelto. Si el error devuelto no es `ESP_ERR_INVALID_STATE`, (significa que el dispositivo está conectado) se establece `modbus_master_connected = 1`.

- La solicitud modbus personalizada, solicita la lectura de un registro tipo Coil en la dirección 0. Aunque se recibe la respuesta devuelta por el esclavo, lo que realmente se verifica es el código de error devuelto.
- Se modificó la tarea app_main() para que en el bucle infinito, en la estructura switch que se dispuso para los diferentes modos de operación, en los casos 2, 3 y 4 (donde se requiere conexión con dispositivo esclavo), los mensajes en consola relacionados con el dispositivo esclavo, estén sujetos a la verificación de la bandera modbus_master_connected; de manera que si esta vale 0; en lugar de los mensajes relacionados al esclavo, se imprime un mensaje de error indicando que se está reintentando la conexión.
- Modificada estructura switch en la tarea modbus_master_poll(), de manera que en cada caso, cada vez que se ejecuten las funciones mbc_master_get_parameter() y mbc_master_set_parameter() y estas devuelvan un código de error diferente a ESP_OK, se establece modbus_master_connected = 0. De esta manera se puede detectar de manera práctica cuando el dispositivo esclavo ya no está presente o ha ocurrido un problema con la comunicación; forzando a que en la próxima ejecución de la tarea se realice una petición de prueba personalizada.
- ✓ Estos cambios en la funcionalidad del protocolo modbus master fueron probados utilizando tanto interfaz TCP/IP como RTU, encontrándose un desempeño óptimo en la detección y manejo de las condiciones de conexión y desconexión de dispositivos esclavos en cada uno de los casos que corresponden a los diferentes modos de operación.
- ✓ Se agregaron mensajes en consola en las funciones modbus_slave_init() y modbus_master_init(), para que al iniciar con interfaz RTU, se muestre la información de que se ha inicializado la pila modbus con el baudrate actual. Esta información es útil para realizar correctamente la conexión del dispositivo esclavo.
- ✓ Fueron removidas las funciones relacionadas con la comunicación SPI, que fueron creadas inicialmente y estaban en desuso; la lista de funciones removidas y el código correspondiente, se ha respaldado en el archivo: Funciones_removidas.c ubicado en la carpeta Documentos.

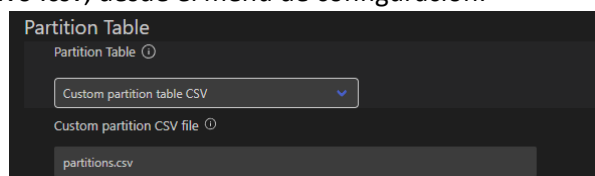
Cambios realizados hasta el 21-10-2023:

- ✓ Se añadió soporte para sistema de archivos FAT en la memoria flash, a través de las bibliotecas de ESP-IDF:

```
#include "esp_vfs.h"
#include "esp_vfs_fat.h"
```

Para realizar esto, se tomaron las siguientes acciones:

- Se seleccionó la opción correspondiente a un esquema de partición personalizado a través de un archivo .csv, desde el menú de configuración:



- Se creó el archivo partitions.csv en la carpeta raíz del proyecto, a fin de especificar manualmente el número de particiones y su tamaño, así:

```

partitions.csv
1 # ESP-IDF Partition Table
2 # Name, Type, SubType, Offset, Size, Flags
3 nvs, data, nvs, 0x9000, 0x6000,
4 phy_init, data, phy, 0xf000, 0x1000,
5 factory, app, factory, 0x10000, 4M,
6 storage, data, fat, , 2M

```

- De esta manera se seleccionó un tamaño de 4M para la partición factory (partición de la aplicación) y un tamaño de 2M para una nueva partición (storage) para crear allí el sistema de archivos FAT.
- Se creó la carpeta partitions en el directorio raíz del proyecto, a fin de contener allí una imagen de los archivos en flash, de momento el único archivo existente es sys_log.log; el cual será utilizado para el registro de eventos del sistema:

```

partition > sys_log.log
1 * Remota A2SCP System Log File: *
2 * This file contains error messages generated by diagnostic systems *
3 * Use this file for monitor or debug system performance and execution *
4
5

```

- Se agregaron las líneas 3 y 4 al archivo CMakeLists.txt de la carpeta main del proyecto, a fin de especificar el comando utilizado para que el compilador incluya la imagen de la partición al momento de programar el microcontrolador:

```

main > CMakeLists.txt
1 idf_component_register(SRCS "main.c" "comm_services.c" "main.c "
2 | | | | INCLUDE_DIRS ".")
3 # set(image ../partition)
4 # fatfs_create_spiflash_image(storage ${image} FLASH_IN_PROJECT)

```

Las líneas 3 y 4 se han comentado luego de haber programado el microcontrolador, a fin de evitar que se sobrescriba el contenido de la partición cada vez que se programe. (Se realiza una sola vez, o cuando se quiera volver a tener la imagen original de la carpeta partitions).

- Se agregó la función init_FAT_fileSystem(), cuya llamada se realiza en la tarea app_main(), al inicio de todas las operaciones:

```

void app_main(void)
{
    esp_err_t res = init_FAT_fileSystem();
    if ( res != ESP_OK) {
        ESP_LOGE(TAG, "Failed to init FAT File System! (%s)", esp_err_to_name(res));
        return;
    }
    else {
        ESP_LOGI(TAG, "FAT File System Initialized (%s)", esp_err_to_name(res));
    }
}

```

Dicha función contiene el código necesario para registrar y montar la partición, permitiendo el acceso a la misma para lectura y escritura, asimismo, se ha habilitado el algoritmo de nivelación de desgaste (wear levelling):

```

esp_err_t init_FAT_fileSystem(void){
    // Mount path for the partition
    const char *base_path = "/spiflash";
}

```

```

// Handle of the wear levelling library instance
static wl_handle_t s_wl_handle = WL_INVALID_HANDLE;

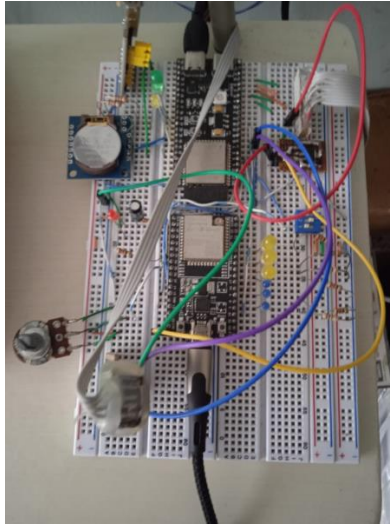
// Register and mount FAT partition:
const esp_vfs_fat_mount_config_t mount_config = {
    .max_files = 4,
    .format_if_mount_failed = true,
    .allocation_unit_size = CONFIG_WL_SECTOR_SIZE
};

esp_err_t err;
err = esp_vfs_fat_spiflash_mount_rw_wl(base_path, "storage", &mount_config, &s_wl_handle);
if (err != ESP_OK) {
    ESP_LOGE(TAG, "Failed to mount FATFS (%s)", esp_err_to_name(err));
    return err;
}

//Get info about FAT partition:
uint64_t total = 0, free = 0;
err = esp_vfs_fat_info(base_path, &total, &free);
if (err != ESP_OK) {
    ESP_LOGE(TAG, "Failed to get partition info (%s)", esp_err_to_name(err));
    return err;
}
else {
    ESP_LOGI(TAG, "Partition size: Total %llu, free %llu", total, free);
}
return ESP_OK;
}

```

- Se han incluido los manejadores de error y los mensajes correspondientes en consola para garantizar la salida apropiada del programa en caso de fallo.
- ✓ Se agregó el soporte para reloj de tiempo real RTC, utilizando el módulo ds1307, y la biblioteca ds1307 del repositorio <https://github.com/UncleRus/esp-idf-lib>. Asimismo se han incluido las dependencias i2cdev y esp_idf_lib_helpers. Todos estos componentes se han incluido en la carpeta components ubicada en el directorio raíz del proyecto.
- ✓ Se realizó el conexionado del módulo RTC al ESP32-S3, mediante los pines SDA GPIO 39 y SCLK GPIO 40, así:



- ✓ Se escribió la función `ds1307_init()`, la cual es llamada durante la ejecución inicial de la tarea `app_main()`:

```
res = ds1307_init();
if ( res != ESP_OK ) {
    ESP_LOGE(TAG, "Failed to init RTC DS1307 (%s)", esp_err_to_name(res));
    return;
}
```

```
esp_err_t ds1307_init(void){

    ESP_ERROR_CHECK(i2cdev_init());

    memset(&dev, 0, sizeof(i2c_dev_t));
    esp_err_t res;

    dev.cfg.sda_pullup_en = GPIO_PULLUP_ENABLE;
    dev.cfg.scl_pullup_en = GPIO_PULLUP_ENABLE;

    res = ds1307_init_desc(&dev, 0, 39, 40);
    return res;
}
```

En la llamada a la función, se evalúa el código de error devuelto y en caso de no ser exitoso el inicio, la tarea `app_main()` aborta su ejecución, no permitiendo que el sistema inicie si ha ocurrido un error al configurar el RTC.

- ✓ Se escribió la función `setTime_ds1307()`, que es llamada cada vez que se modifica un registro de la tabla auxiliar asociado al RTC, a fin de permitir que la hora y fecha sean configuradas en el RTC:

```
esp_err_t setTime_ds1307(void){
    // setup datetime: 2023-10-20 20:30:10
    struct tm actualTime = {
        .tm_year = AUX_RTC_YEAR - 1900, //since 1900 (2023 - 1900)
        .tm_mon  = AUX_RTC_MONTH - 1,  // 0-based
        .tm_mday = AUX_RTC_DAY,
        .tm_hour = AUX_RTC_HOUR,
        .tm_min  = AUX_RTC_MINUTE,
        .tm_sec  = AUX_RTC_SECOND
        //Agregar wday!!
    };
    esp_err_t res = ds1307_set_time(&dev, &actualTime);
    return res;
}
```

- ✓ La asignación de los registros de la tabla auxiliar para la configuración del RTC fue la siguiente:

```
#define AUX_RTC_YEAR      s3Tables.auxTbl[0][43]
#define AUX_RTC_MONTH    s3Tables.auxTbl[0][44]
#define AUX_RTC_DAY      s3Tables.auxTbl[0][45]
#define AUX_RTC_HOUR     s3Tables.auxTbl[0][46]
#define AUX_RTC_MINUTE   s3Tables.auxTbl[0][47]
#define AUX_RTC_SECOND   s3Tables.auxTbl[0][48]
```

- ✓ Se agregó el código necesario en la tarea mb_event_check_task(), para identificar un cambio en alguno de estos registros, validando la aceptación únicamente de los valores apropiados para cada caso y llamando a la función setTime_ds1307(). Los valores escritos desde el SCADA en estos registros son respaldados en la tabla espejo en la memoria NVS flash.
- ✓ Se escribió la función system_logInput(char* message); que permite generar una entrada en el archivo sys_log.log, añadiendo un evento al registro. Dicha entrada está formada por una estampa de tiempo (timestamp) y un mensaje que se pasa como una cadena de caracteres a la función. La ejecución de la función, realiza las siguientes tareas:
 - Obtener al estampa de tiempo actual del RTC.
 - Realiza una copia del archivo sys_log.log al archivo sys_log.bak, si los archivos no existen, son creados automáticamente.
 - Agrega la entrada al archivo sys_log.bak, añadiendo una nueva línea de texto que contiene la estampa de tiempo y el mensaje.
 - Borra el archivo original sys_log.log.
 - Cambia el nombre del archivo sys_log.bak a sys_log.log.
- ✓ Se escribió la función print_systemLog(), que imprime en consola el contenido del archivo sys_log.log, para efectos de visualización.
- ✓ Se realizó una prueba del funcionamiento del código desarrollado para este fin, mediante la generación de una entrada al finalizar la ejecución inicial de la tarea app_main():

```
// Generate a test input for sys_log.log and print its content:
system_logInput("Remota systems have been succesfully initialized");
print_systemLog();
```

La salida en consola del programa, correspondiente a la entrada generada y la visualización del archivo señalado, fue la siguiente: (nótese que para cada reinicio, se genera una nueva línea con la estampa de tiempo correspondiente).


```

I (545) Remota-Main: Gas accumulator timer started with interval 10 ms
I (555) Remota-Main: Gas PID timer started with interval 10 ms
V (565) Remota-Main: Creating a backup file from sys_log.log to sys_log.bak
E (645) CRC Check:: Communication CRC16 error - Bad checksum!
V (715) Remota-Main: Append info to the file sys_log.bak
E (745) CRC Check:: Communication CRC16 error - Bad checksum!
V (775) Remota-Main: Removing original file sys_log.log
V (835) Remota-Main: Original file has been removed
V (835) Remota-Main: Rename the file sys_log.bak to sys_log.log
V (865) Remota-Main: File renamed successfully
E (865) CRC Check:: Communication CRC16 error - Bad checksum!

System log (sys_log.log file):

*          Remota A2SCP System Log File:          *
* This file contains error messages generated by diagnostic systems *
* Use this file for monitor or debug system performance and execution *

(21-10-2023 11:18:52) - Remota systems have been succesfully initialized
(21-10-2023 11:18:56) - Remota systems have been succesfully initialized
(21-10-2023 11:21:26) - Remota systems have been succesfully initialized
(21-10-2023 15:09:03) - Remota systems have been succesfully initialized
(21-10-2023 18:00:39) - Remota systems have been succesfully initialized
(21-10-2023 18:05:36) - Remota systems have been succesfully initialized

```

Cambios realizados hasta el 22-10-2023:

- ✓ Debido a la especificación en la hoja de datos del RTC DS1307, se agregó el código necesario en la función ds1307_init() para que los registros del RTC sean leídos y escritos de vuelta, a fin de asegurarse que el bit CH (clock halt) sea colocado en 0 y el RTC sea activado. Al leer los registros y volverlos a escribir, no se pierden los valores de los mismos.



CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. The real time clock registers are illustrated in Figure 3. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the Binary-Coded Decimal (BCD) format. Bit 7 of Register 0 is the Clock Halt (CH) bit. When this bit is set to a 1, the oscillator is disabled. When cleared to a 0, the oscillator is enabled.

Please note that the initial power on state of all registers is not defined. Therefore it is important to enable the oscillator (CH bit=0) during initial configuration.

3 of 11

- ✓ Agregado el soporte para el conteo del día de semana, para ello, en la función setTime_ds1307() se agregó el código que calcula el día de la semana dada la información del día, mes y año y utilizando una implementación de la función de Zeller:

```

esp_err_t setTime_ds1307(void)
{
    // setup datetime: 2023-10-20 20:30:10
    uint16_t d, m, y;
    d = AUX_RTC_DAY;
    m = AUX_RTC_MONTH;
    y = AUX_RTC_YEAR;

    struct tm actualTime = {
        .tm_year = AUX_RTC_YEAR - 1900, //since 1900 (2023 - 1900)
        .tm_mon = AUX_RTC_MONTH - 1, // 0-based
        .tm_mday = AUX_RTC_DAY,
        .tm_hour = AUX_RTC_HOUR,
        .tm_min = AUX_RTC_MINUTE,
        .tm_sec = AUX_RTC_SECOND,
        .tm_wday = (d+m<3?y--:y-2,23*m/9+d+4+y/4-y/100+y/400)%7;
    };
    esp_err_t res = ds1307_set_time(&dev, &actualTime);
    return res;
}

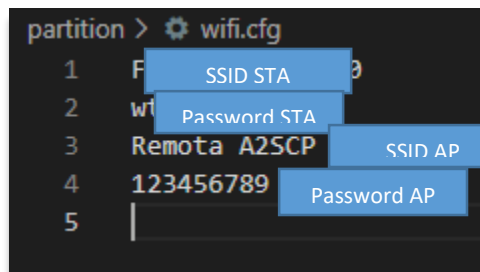
```

- ✓ Se cambió el argumento de la función system_logInput() de (char* message) a (const char* message).

- ✓ Se eliminaron las funciones para guardar los datos del RTC en la tabla auxiliar en la memoria flash; en su lugar, en la tarea app_main() en el bucle infinito, los valores de los registros de la tabla auxiliar son actualizados con la fecha y hora leídos desde el RTC, de forma que es posible obtener esta información actualizada desde el SCADA.

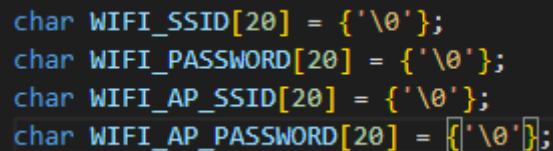
Cambios realizados hasta el 25-10-2023:

- ✓ Incorporada la funcionalidad del módulo WiFi del ESP32-S3, tanto en modo estación (STA), como en modo punto de acceso (AP), y en modo mixto (STA + AP). Para esto se realizaron las siguientes acciones:
 - Se crearon las funciones WiFi_Begin_STA(), WiFi_Begin_AP() y WiFi_Begin_STA_AP() y se agregaron al archivo comm_services.c. Dichas funciones permiten la inicialización del módulo WiFi del microcontrolador en los diferentes modos.
 - Se agregó la función WiFi_init() al archivo comm_services.c. Esta función es llamada desde la tarea app_main() y contiene todo el código necesario para iniciar el módulo WiFi. La función realiza las siguientes tareas:
 - Abre el archivo llamado wifi.cfg (ubicado en la partición FAT); dicho archivo contiene las credenciales (SSID y password) tanto para acceder al punto de acceso (router), como para configurar el punto de acceso (AP) del microcontrolador. El archivo contiene esta información en 4 líneas, de la siguiente manera:



```
partition > wifi.cfg
1 F SSID STA 0
2 w Password STA
3 Remota A2SCP SSID AP
4 123456789 Password AP
5 |
```

- La función lee cada línea del archivo, extrayendo la información del SSID y Password para ambos modos y la almacena en las variables globales:



```
char WIFI_SSID[20] = {'\0'};
char WIFI_PASSWORD[20] = {'\0'};
char WIFI_AP_SSID[20] = {'\0'};
char WIFI_AP_PASSWORD[20] = {'\0'};
```

- Luego, toma en cuenta el registro de la tabla de configuración CFG_WIFI_MODE (s3Tables.configTbl[0][24]), para elegir el modo WiFi en el que se iniciará, de manera que el modo es seleccionable desde el SCADA y/o la interfáz de usuario. La selección del modo depende del valor que tenga este registro, de la siguiente manera:

CFG_WIFI_MODE	
Valor	Configuración
0	Modo estación (STA)
1	Modo punto de acceso (AP)
2	Modo estación y punto de acceso (STA + AP)
3	Modo off (No WiFi)

- Una vez extraída la información del archivo, se llama la función apropiada dependiendo del modo WiFi seleccionado. Si el modo seleccionado es 3 (No WiFi), no se realiza ninguna inicialización del módulo.
- Se escribieron las funciones set_wifi_STA_ip() y set_wifi_AP_ip() las cuales permiten configurar las direcciones IP estáticas que se utilizarán en los diferentes modos de WiFi. Dichas funciones toman las direcciones IP almacenadas en la tabla de configuración, en los registros CFG_IP2 y CFG_IP3; se utilizó un procedimiento similar al usado para establecer la dirección IP del módulo ethernet.
- Para el caso del modo punto de acceso (AP), la dirección IP y la puerta de enlace (GW) son las mismas.
- Se modificó la función set_DHCP(), para que dependiendo del registro CFG_DHCP de la tabla de configuración, aplique la configuración de IP automática (via servidor DHCP) o configuración IP estática, llamando apropiadamente las funciones que aplican dichas configuraciones. Si el registro CFG_DHCP tiene valor 0, se aplicará configuración de IP estática tanto al módulo ethernet como al módulo WiFi, de lo contrario, se aplicará configuración de IP automática en ambos casos.
- La función set_DHCP() es llamada en las funciones WiFi_Begin_STA(), WiFi_Begin_AP() y WiFi_Begin_STA_AP(); en caso de que el registro CFG_DHCP valga 0, para aplicar la configuración de IP estática al inicio.
- Al ser cambiado el valor del registro CFG_DHCP desde el SCADA, la función set_DHCP() es llamada automáticamente, de modo que la nueva configuración es aplicada inmediatamente.

✓ La tabla de configuración actualizada es la siguiente:

Registros de la tabla de configuración: s3Tables.configTbl[0][i]			
Nombre	Registro (Índice)	Dirección Modbus	Descripción
CFG_RUN_PGM	0	40017	Modo de ejecución o modo de programación
CFG_OP_MODE	1	40018	Selección del método de producción
CFG_IP0	2	40019	Dirección IP Ethernet 0
	3	40020	
CFG_SUBNET_MASK	4	40021	Subnet Mask
	5	40022	
CFG_IP2	6	40023	Dirección IP WiFi (Station)
	7	40024	

Registros de la tabla de configuración: s3Tables.configTbl[0][i]			
Nombre	Registro (Índice)	Dirección Modbus	Descripción
CFG_IP3	8	40025	Dirección IP WiFi (AP)
	9	40026	
CFG_GW	10	40027	Network Gateway
	11	40028	
CFG_DHCP	12	40029	Dirección IP estática o dinámica
CFG_MB_MASTER_INTERFACE	13	40030	Modbus maestro TCP o RTU
CFG_MB_MASTER_BAUDRATE_H	14	40031	Velocidad UART2 (Modbus master)
CFG_MB_MASTER_BAUDRATE_L	15	40032	
CFG_MB_SLAVE_INTERFACE	16	40033	Modbus esclavo TCP o RTU
CFG_MB_SLAVE_BAUDRATE_H	17	40034	Velocidad UART1 (Modbus slave)
CFG_MB_SLAVE_BAUDRATE_L	18	40035	
CFG_SLAVE_IP	19	40036	Dirección IP del equipo esclavo (Modbus TCP)
	20	40037	
CFG_GL_TMR_INTERVAL	21	40038	Tiempo de ejecución timer de acumulación de gas
CFG_GL_FILTER_ALPHA	22	40039	Constante de tiempo del filtro de primer orden para señal de flujo de gas (multiplicada x 1000)
CFG_GL_PID_TMR_INTERVAL	23	40040	Tiempo de ejecución timer de PID
CFG_WIFI_MODE	24	40041	Selección de modo WiFi 0 --> STA, 1 --> AP, 2 --> STA+AP, 3 --> No WiFi
	25	40042	
	26	40043	
	27	40044	
	28	40045	
	29	40046	
	30	40047	
	31	40048	
	32	40049	
	33	40050	
	34	40051	
	35	40052	
	36	40053	
	37	40054	
	38	40055	
	39	40056	
	40	40057	
CFG_GL_PID_SP	41	40058	Setpoint para el controlador PID
CFG_GL_PID_KP	42	40059	Ganancia proporcional * 1000 (PID)
CFG_GL_PID_KI	43	40060	Ganancia integral * 1000 (PID)
CFG_GL_PID_KD	44	40061	Ganancia derivativa * 1000 (PID)

Registros de la tabla de configuración: <code>s3Tables.configTbl[0][i]</code>			
Nombre	Registro (índice)	Dirección Modbus	Descripción
CFG_GL_PID_N	45	40062	Constante de filtro N * 1000 (PID)
CFG_GL_PID_CP	46	40063	Coeficiente de activación control proporcional (0 ó 1)
CFG_GL_PID_CI	47	40064	Coeficiente de activación control integral (0 ó 1)
CFG_GL_PID_CD	48	40065	Coeficiente de activación control derivativo (0 ó 1)
CFG_REMOTA_LOG_LEVEL	49	40066	Nivel de log para mensajes en consola serial

- ✓ Incorporada funcionalidad de Master Reset, a través de un pulsador conectado al GPIO 20. Dicho GPIO se ha configurado como entrada y se ha habilitado la resistencia de pull up. La activación de esta funcionalidad sigue la siguiente secuencia:
 - Durante la inicialización de la Remota, luego de inicializar los GPIO y la partición NVS, se chequea el estado de la entrada pushMasterReset; si esta está en 0 (botón pulsado), luego de un algoritmo antirebote, se llama la función `set_configDefaults_nvs()`.
 - La función `set_configDefaults_nvs()`, comienza creando las tablas espejo en flash en caso de que no existan y a continuación ejecuta la escritura de todos los valores de la tabla de configuración "C", a los valores por defecto establecidos.
 - En el código de esta función, pueden observarse los valores por defecto que serán configurados, así como las direcciones modbus y nombres de registros correspondientes:

```
esp_err_t set_configDefaults_nvs(void){
    ESP_LOGW(TAG, "Setting nvs flash contents to default values...");

    //Create tables in the nvs namespace (if they don't exist):
    create_table_nvs("C", s3Tables.configSize);    //For config table
    create_table_nvs("A", s3Tables.auxSize);        //For aux table
    create_float_table_nvs("SF", s3Tables.anSize);    //For Scaling factors table
    create_float_table_nvs("SO", s3Tables.anSize);    //For Scaling offsets table

    //
    Modbus addr:  Register name:  Default value:
    write_nvs("C0", 0x0001);    //40017    CFG_RUN_PGM    (RUN)
    write_nvs("C1", 0x0000);    //40018    CFG_OP_MODE    (Natural flow)
    write_nvs("C2", 0xAC10);    //40019    CFG_IP0    (172.16)
    write_nvs("C3", 0x0064);    //40020    CFG_IP0    (.0.100)
    write_nvs("C4", 0x0000);    //40021    CFG_IP1
    write_nvs("C5", 0x0000);    //40022    CFG_IP1
    write_nvs("C6", 0xAC10);    //40023    CFG_IP2    (172.16)
    write_nvs("C7", 0x002B);    //40024    CFG_IP2    (.0.43)
    write_nvs("C8", 0xC0A8);    //40025    CFG_IP3    (192.168)
```

```

write_nvs("C9", 0x0001); //40026 CFG_IP3 (.0.1)
write_nvs("C10", 0xAC10); //40027 CFG_GW (172.16)
write_nvs("C11", 0x0001); //40028 CFG_GW (.0.1)
write_nvs("C12", 0x0000); //40029 CFG_DHCP (Static IP)
write_nvs("C13", 0x0001); //40030 CFG_MB_MASTER_INTERFACE (TCP interface)
write_nvs("C14", 0x0001); //40031 CFG_MB_MASTER_BAUDRATE_H (115200)
write_nvs("C15", 0xC200); //40032 CFG_MB_MASTER_BAUDRATE_L
write_nvs("C16", 0x0001); //40033 CFG_MB_SLAVE_INTERFACE (TCP Interface)
write_nvs("C17", 0x0001); //40034 CFG_MB_SLAVE_BAUDRATE_H (128000)
write_nvs("C18", 0xF400); //40035 CFG_MB_SLAVE_BAUDRATE_L
write_nvs("C19", 0xAC10); //40036 CFG_SLAVE_IP (172.16)
write_nvs("C20", 0x0004); //40037 CFG_SLAVE_IP (.0.4)
write_nvs("C21", 0x000A); //40038 CFG_GL_TMR_INTERVAL (10ms)
write_nvs("C22", 0x000A); //40039 CFG_GL_FILTER_ALPHA (10ms * 1000)
write_nvs("C23", 0x000A); //40040 CFG_GL_PID_TMR_INTERVAL (10ms)
write_nvs("C24", 0x0002); //40041 CFG_WIFI_MODE (2 --> STA + AP)

```

```

write_nvs("C25", 0x0000); //40042
write_nvs("C26", 0x0000); //40043
write_nvs("C27", 0x0000); //40044
write_nvs("C28", 0x0000); //40045
write_nvs("C29", 0x0000); //40046
write_nvs("C30", 0x0000); //40047
write_nvs("C31", 0x0000); //40048
write_nvs("C32", 0x0000); //40049
write_nvs("C33", 0x0000); //40050
write_nvs("C34", 0x0000); //40051
write_nvs("C35", 0x0000); //40052
write_nvs("C36", 0x0000); //40053
write_nvs("C37", 0x0000); //40054
write_nvs("C38", 0x0000); //40055
write_nvs("C39", 0x0000); //40056
write_nvs("C40", 0x0000); //40057

```

```

write_nvs("C41", 0x09C4); //40058 CFG_GL_PID_SP (2500 MPCGD)
write_nvs("C42", 0x1388); //40059 CFG_GL_PID_KP (5 * 1000)
write_nvs("C43", 0x09C4); //40060 CFG_GL_PID_KI (2.5 * 1000)
write_nvs("C44", 0x07D0); //40061 CFG_GL_PID_KD (2 * 1000)
write_nvs("C45", 0x03E8); //40062 CFG_GL_PID_N (1 * 1000)
write_nvs("C46", 0x0001); //40063 CFG_GL_PID_CP (1)
write_nvs("C47", 0x0001); //40064 CFG_GL_PID_CI (1)
write_nvs("C48", 0x0001); //40065 CFG_GL_PID_CD (1)
write_nvs("C49", 0x0004); //40066 CFG_REMOTA_LOG_LEVEL (4 --> Debug)

```

```

ESP_LOGW(TAG, "Default values has been written to flash");

```

```
return ESP_OK;  
}
```

- Esta funcionalidad es útil cuando por alguna razón se ha perdido la configuración inicial (o no se ha establecido al grabar el microcontrolador por primera vez), y también en caso de tener algún problema con valores de configuración incorrectos.
- ✓ Se ha reubicado el código de inicialización de todos los módulos y tareas de la Remota (que antes estaba en la sección de ejecución inicial de la tarea `app_main()`); dicho código se ha movido a una nueva función llamada `Remota_init()`.
- ✓ Se ha modificado la función `print_spi_stats()` para incluir dentro de la función, todas las salidas relacionadas.
- ✓ Se crearon las funciones `resume_tasks()` y `stop_tasks()`; en las cuales se ha reubicado el código necesario para pausar las tareas (en modo PROGRAM), y para reanudar las tareas, (en modo RUN).
- ✓ Con estos últimos cambios en la organización del código, se obtiene una apariencia mas limpia y ordenada, preparando y facilitando las futuras tareas de diagnóstico y autodiagnóstico.

Cambios realizados hasta el 27-10-2023:

- ✓ Se escribió toda la lógica concerniente al manejo del diagnóstico de la comunicación con el(los) equipo(s) de campo. Para esto, se tomaron en cuenta las siguientes consideraciones:
 - Se modificó la función `mb_master_init()`, con la finalidad de reescribir los manejadores de error internos de la función y manejar las posibles causas de fallo en la inicialización. La función devuelve el código de error `ESP_FAIL` si alguna de las etapas de inicialización falla; entonces en el punto de llamada a la función (durante la ejecución de `Remota_init()`) se chequea dicho código de error, dándole el manejo apropiado y generando el mensaje para el archivo de registro de eventos del sistema.
 - Si la función de inicialización falla, la tarea `app_main()` retorna y el programa se detiene, siendo necesario un reinicio del microcontrolador.
 - La función `mb_master_init()` ahora chequea las banderas `ethernet_got_ip` y `wifi_got_ip`, las cuales fueron creadas para informar cuando los modulos ethernet y wifi han establecido sus conexiones de red y tienen una dirección IP asignada. El chequeo de esta condición se utiliza para que en caso de haber seleccionado interfaz TCP/IP para modbus master, la inicialización espera 25 segundos hasta que se haya establecido una conexión IP; en caso contrario el programa abortará generando un error en consola y un reporte al archivo `sys_log.log`.
 - Se agregó la lógica necesaria en la tarea `mb_master_poll()`, para generar las entradas necesarias al archivo `sys_log.log`, a fin de reportar cuando se ha establecido la comunicación con el(los) equipo(s) esclavo(s) (de campo) y también generar el reporte

cuando se ha perdido la conexión. El programa se asegura que en caso de que la condición de pérdida de conexión sea persistente, la entrada en el archivo sys_log.log, es escrita una sola vez al perderse la conexión.

- ✓ Se escribió la lógica relacionada al manejo de las estadísticas de comunicación con el(los) equipo(s) de campo, esto se realizó de la siguiente manera:

- Se asignaron los registros de la tabla auxiliar:

```
#define AUX_MB_MASTER_TOTAL_POLLS s3Tables.auxTbl[0][5]
#define AUX_MB_MASTER_ERR_COUNT   s3Tables.auxTbl[0][6]
#define AUX_MB_MASTER_RETRY_COUNT s3Tables.auxTbl[0][7]
```

- En la tarea mb_master_poll() se realiza el conteo de las transacciones modbus realizadas, el conteo de errores y el conteo de operaciones de reintento de comunicación. Dichos conteos se almacenan en los registros auxiliares señalados anteriormente, de manera que esta información es visible desde el SCADA.
 - Se incluyó una lógica que verifica la condición de desborde del contador de transacciones y reinicia en ese caso, todos los contadores.
 - Se escribió la función print_mb_master_stats(), la cual genera la salida por consola de la información relacionada a las estadísticas de comunicación; mostrando el número de transacciones, el conteo de errores y reintentos; así como la tasa porcentual calculada de error. Dicha función es llamada durante la ejecución del bucle infinito de app_main(), únicamente en los casos donde se utiliza comunicación con equipos de campo.
- ✓ Se incluyó un retardo en la inicialización del sistema, ya que que al finalizar la programación, se genera un reinicio desde vscode y al iniciar el monitor serial se produce un segundo reinicio desde vscode; entonces, dicho retardo evita que se generen entradas duplicadas en el archivo sys_log.log en este caso.
 - ✓ Se incluyó la generación de un mensaje en consola que reporta la fecha y hora actuales, incluyendo el día de semana, y se ejecuta en cada iteración del bucle infinito de app_main().
 - ✓ Se agregó una salida por consola al generar una entrada al archivo sys_log.log mediante la función system_logInput(), de manera que se pueda visualizar en consola el mensaje escrito al archivo.
 - ✓ Se modificó la función sys_logInput(), para que el código que escribe al archivo sys_log.log pueda desactivarse mediante la definición de la macro SYS_LOG_FILE_WRITE_DISABLE, la cual se encuentra en el archivo remota_globals.h. Esto es especialmente útil en tiempo de depuración del programa, para evitar escrituras frecuentes en la memoria flash. En caso de que

la escritura al archivo esté desactivada, solo será visible el mensaje generado en consola. Si se desea habilitar la escritura en el archivo, basta con comentar la macro correspondiente.

- ✓ La sección de código correspondiente a la escritura en el archivo `sys_log.log` fue protegida por un semáforo de FreeRTOS, dado que esta función puede ser llamada desde varias tareas de forma concurrente. Esto evita que se generen errores en tiempo de ejecución al intentar abrir más de una vez los archivos relacionados (`sys_log.log` y `sys_log.bak`). El semáforo utilizado es un semáforo binario llamado `sysLogFileSem` y es inicializado durante la ejecución de la función `Remota_init()`.
- ✓ Se agregó una llamada a la función `print_systemLog()` al ejecutar la función `stop_tasks()`; de manera que al seleccionar el modo PROGRAM desde el SCADA, se genera una visualización en consola del archivo `sys_log.log`. Esto puede ser útil, ya que la información se presenta de manera oportuna antes de realizar algún cambio en la configuración. Asimismo, la llamada a la función `print_systemLog()` durante la inicialización de la remota, fue removida.

Cambios realizados hasta el 29-10-2023:

- ✓ Se modificó la función `mb_slave_init()`, para incluir los manejadores de error necesarios y sustituir las funciones `ESP_ERROR_CHECK()` (las cuales causan un reinicio del microcontrolador al producirse un error en la función a la cual se aplica). Dichos manejadores de error se utilizan para generar la información relacionada con el error y mostrarla en consola; así como para proporcionar puntos de salida de la función retornando el código `ESP_FAIL`.
- ✓ En la función `Remota_init()`, en el punto de llamada a la función `mb_slave_init()`, se agregó la lógica que maneja el código de error devuelto por la función, de manera que se generan los mensajes apropiados tanto en consola como para las entradas correspondientes al archivo `sys_log.log`. En caso de producirse un error, la función `Remota_init()` retornará con código `ESP_FAIL`; lo cual también provocará que la función `app_main()` aborte con el código de error apropiado, en este último caso con un mensaje de error relacionado al fallo del proceso de inicialización de la remota.
- ✓ Se modificó la tarea `mb_event_check_task()`, que anteriormente chequeaba únicamente cuando ocurría un evento de tipo `HOLDING REGISTER WRITE`; para que ahora pueda chequear todos los posibles eventos de lectura y escritura a los diferentes tipos de registro.

- Esto se realizó a través de la creación de la máscara de eventos:

```
mb_event_group_t event_mask = (MB_EVENT_HOLDING_REG_WR |
                                MB_EVENT_HOLDING_REG_RD |
                                MB_EVENT_INPUT_REG_RD |
                                MB_EVENT_DISCRETE_RD |
                                MB_EVENT_COILS_RD |
                                MB_EVENT_COILS_WR );
```

```

while (1)
{
    memset(&reg_info, 0, sizeof(mb_param_info_t));
    //mb_event_group_t event = mbc_slave_check_event(MB_EVENT_HOLDING_REG_WR);
    mb_event_group_t event = mbc_slave_check_event(event_mask);

    if (event & MB_EVENT_HOLDING_REG_WR){
        AUX_MB_SLAVE_HR_WRITES++;
        for (int i = 0; i<=CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_SIZE; i++)
        {
            mbc_slave_get_param_info(&reg_info, 10 / portTICK_PERIOD_MS);
            ESP_LOGV(mbEventChkTAG, "HOLDING (%lu us), ADDR:%lu, TYPE:%lu, SIZE:%u",
                (uint32_t)reg_info.time_stamp,
                (uint32_t)reg_info.mb_offset,
            );
        }
    }
}

```

- Se asignaron los registros destinados a contener los valores correspondientes a las estadísticas de comunicación con el sistema SCADA: (Estos registros serán utilizados para llevar el conteo de solicitudes modbus recibidas tanto de lectura como escritura de los diferentes tipos de registro modbus).

```

#define AUX_MB_SLAVE_HR_READS      s3Tables.auxTbl[0][8]
#define AUX_MB_SLAVE_HR_WRITES    s3Tables.auxTbl[0][9]
#define AUX_MB_SLAVE_COIL_READS   s3Tables.auxTbl[0][10]
#define AUX_MB_SLAVE_COIL_WRITES  s3Tables.auxTbl[0][11]
#define AUX_MB_SLAVE_INPUT_READS  s3Tables.auxTbl[0][12]
#define AUX_MB_SLAVE_STATUS_READS s3Tables.auxTbl[0][13]

```

- Al producirse un evento de lectura o escritura, la función de la tarea mb_event_check_task() determina el tipo de evento y efectúa el conteo utilizando el contador apropiado; asimismo, cada vez que se produce un evento se realiza el chequeo correspondiente para evitar el desborde de los contadores, y en caso de detectarse esa condición se reinician todos a la vez, a fin de que el total de operaciones (el cual se obtiene sumando todos los contadores) sea representativo.
- Se validó el funcionamiento del código implementado al visualizar los valores de estos registros en el simulador ModScan, apreciándose que se incrementaban apropiadamente al producirse los eventos de lectura y escritura modbus. El código se probó tanto en modo TCP/IP como en modo RTU.
- ✓ Se escribió la función print_mb_slave_stats(), la cual presenta en consola las estadísticas de comunicación con el sistema SCADA, mostrando los contadores anteriores y el total de operaciones de lectura y escritura.
- ✓ A fin de determinar en qué momento se establece una comunicación con un cliente modbus (SCADA), se creó la tarea system_monitor_task(); la cual se encarga de monitorear el total de operaciones de lectura y escritura realizadas desde el cliente modbus. Si el contador de

operaciones totales se detiene durante 5 segundos, entonces se establece la bandera `mb_slave_comm_live` en 0, generando el reporte al archivo `sys_log.log` de que la comunicación con el sistema SCADA no está activa; (de manera similar si el contador de operaciones cambia, se genera el mensaje apropiado).

- ✓ La tarea lleva el registro del último estado de la comunicación (`mb_slave_comm_live_last`); a fin de que el mensaje escrito al archivo sólo se genere una vez al producirse un cambio en esta condición.
- ✓ La tarea de monitoreo de sistema, `system_monitor()` podrá utilizarse para realizar operaciones adicionales de diagnóstico y ha sido configurada en el núcleo 0 y con la mínima prioridad, a fin de que su ejecución no afecte los demás procesos que se ejecutan dentro de la remota.
- ✓ Se implementaron rutinas de **Autodiagnóstico** que se ejecutan al momento de la inicialización de cada módulo del sistema y durante la ejecución de la función `Remota_init()`, los módulos diagnosticados fueron los siguientes:
 - Módulo RTC: a través de la verificación del código de error devuelto por la función `ds1307_init()`.
 - Sistema de archivos FAT: a través de la función `init_FAT_fileSystem()`.
 - Inicialización de los GPIO: se agregó el mensaje al archivo `sys_log.log`, como verificación de que la ejecución de esa etapa se ha completado.
 - Partición NVS: a través del código de error devuelto por la función `init_nvs()`.
 - La función `init_nvs()` se modificó a fin de manejar los posibles errores relacionados con la inicialización del almacenamiento NVS en flash; como por ejemplo, la condición de espacio insuficiente. En este caso, el programa al determinar que ha fallado la inicialización de la partición NVS, procederá al borrado de la partición NVS e intentará inicializarla de nuevo.
 - En caso de que la partición NVS haya sido borrada, se generarán los mensajes de advertencia apropiados y se procederá a escribir los valores por defecto de la tabla de configuración espejo en flash; a fin de garantizar que el sistema pueda iniciar con la configuración por defecto establecida.
 - Verificación de pulsador de Master Reset: Se agregaron los mensajes apropiados para el archivo `sys_log.log`.
 - Creación de tablas en RAM y carga de datos desde la flash: Se agregó el mensaje de verificación al concluir la ejecución de esta etapa.

- Módulo Ethernet: A través del código de error devuelto por la función `ethernetInit()`.
 - Se modificó la función `ethernetInit()` para agregar los manejadores de error, sustituyendo las funciones `ESP_ERROR_CHECK()` por verificaciones de error, generando el retorno de la función con código `ESP_FAIL` en caso de error.
 - Se modificó el manejador de eventos para el módulo ethernet, escrito en la función `eth_event_handler()` del archivo `comm_services.c`. Al generarse los eventos `ETHERNET_EVENT_CONNECTED` y `ETHERNET_EVENT_DISCONNECTED`, se generan los mensajes apropiados para el archivo `sys_log.log`. Esto permite identificar si el cable de red del módulo ethernet está conectado o no.
 - Se modificó el manejador de eventos `got_ip_event_handler()`, el cual se ejecuta cuando el módulo ethernet ha negociado la dirección IP con el dispositivo al que se ha conectado. La dirección IP (y demás información relacionada con la conexión de red) se muestra en consola y se genera una entrada en `sys_log.log` que contiene la IP configurada para el módulo ethernet.
- Módulo WiFi: A través del código de error devuelto por la función `WiFi_init()`. También se tomó en cuenta el valor del registro `CFG_WIFI_MODE`, para verificar si el módulo WiFi ha sido configurado como desactivado (3--> No WiFi).
 - En `WiFi_init()` se generan los mensajes correspondientes al modo en que el módulo WiFi ha sido iniciado.
 - En la función `set_wifi_AP_ip()` se agregó el código para generar una entrada en `sys_log.log` que incluye la dirección IP del punto de acceso creado en la remota.
 - En el manejador de eventos `wifi_event_handler()`, se agregaron las entradas correspondientes al archivo `sys_log.log`, en caso de producirse los eventos relacionados con la conexión y desconexión del módulo WiFi a un punto de acceso, así como la entrada al archivo que contiene la dirección IP negociada por el módulo WiFi con el punto de acceso al que se ha conectado.
 - Al mismo manejador de eventos anterior, se agregaron los mensajes para el archivo de registro, relacionados con los eventos de conexión y desconexión de una estación externa al punto de acceso creado en la remota.
- Módulo Modbus Slave: Detallado anteriormente.
- Módulo Modbus Master: Detallado anteriormente.
- ✓ Se agregó a la tarea `system_monitor()`, la funcionalidad para medir la temperatura del microcontrolador, utilizando su sensor de temperatura integrado en el chip. Para ello se utilizó como referencia el código de ejemplo proporcionado por el fabricante y se realizó una adaptación del mismo para incorporarlo a la tarea.
 - Se asignó el registro de la tabla auxiliar:

```
#define AUX_CPU_TEMPERATURE      s3Tables.auxTbl[0][14]
```

A fin de llevar el registro del valor de la temperatura medido. En vista de que el valor de temperatura es un valor en punto flotante, se almacenó dicho valor multiplicado por 100, para poder almacenar esta cantidad con una precisión de máximo 3 cifras decimales en un registro uint16.

- Se observó que la temperatura del microcontrolador oscila entre los 40 y 50 °C, produciéndose un incremento significativo al utilizar el módulo WiFi. Aún así, los valores observados son temperaturas aceptables para un microprocesador y se encuentran dentro de los límites especificados por el fabricante.
- El valor de temperatura, es reportado junto con el resto de la información en la consola, durante el bucle infinito de app_main().

✓ A continuación se presenta la lista de mensajes de registro posibles generados por el programa para el archivo sys_log.log:

Origen	Mensaje a sys_log.log	Tipo
init_FAT_fileSystem()	Remota system powered up...	Información
init_FAT_fileSystem()	FAT file system is up and running	Información
Remota_init()	Real time clock module started up	Información
	GPIO configurations have been made	Información
	Failed to initialize NVS flash partition	Error
	NVS flash partition is up and running	Información
	Master reset button activated!	Advertencia
	WARNING! NVS has been set to it's default values	Advertencia
	RAM tables created and data loaded	Información
	Failed to initialize ethernet module	Error
	Ethernet module has been succesfully initialized	Información
	Failed to initialize WiFi module	Error
	WiFi module has been succesfully initialized	Información
	WiFi module is set to OFF in configuration table	Información
	Modbus slave stack initialization failed	Error
	Modbus slave stack succesfully initialized	Información
	Modbus master initialization error	Error
	Op. Mode: <modo de operación>	Información
app_main()	Remota systems have been succesfully initialized	Información
spi_task()	SPI Communication with I/O Module has been initialized	Información
	I/O Module communication error	Error
modbus_master_init()	Remota initialization failed - No IP for Modbus TCP master	Error
	IP address for modbus slave device: xxx.xxx.xxx.xxx	Información

	Modbus master RTU baudrate is: xxxxxx bps	Información
	Modbus master stack succesfully initialized	Información
Modbus_slave_init()	Modbus slave RTU baudrate is: xxxxxx bps	Información
mb_master_poll_task()	Modbus master is NOT connected to slave device(s)	Error
	Modbus master connected to slave device(s)	Información
system_monitor_task()	SCADA client connection is active	Información
	SCADA client connection is NOT active	Advertencia
init_nvs()	WARNING! NVS partition has been formatted!	Advertencia
	WARNING! NVS has been set to it's default values	Advertencia
eth_event_handler()	Ethernet module link up	Información
	Ethernet module link down	Advertencia
got_ip_event_handler()	Ethernet got IP: xxx.xxx.xxx.xxx	Información
wifi_event_handler()	WiFi station: Connected to AP	Información
	WiFi station got IP: xxx.xxx.xxx.xxx	Información
	WiFi station: Disconnected from AP	Información
	WiFi AP: Station disconnected	Información
	WiFi AP: Station connected	Información
set_wifi_AP_ip()	WiFi AP IP address: xxx.xxx.xxx.xxx	Información
WiFi_init()	WiFi initialization in station mode	Información
	WiFi initialization in AP mode	Información
	WiFi initialization in station + AP mode	Información

✓ La asignación de registros de la tabla auxiliar es la siguiente:

Registros de la tabla auxiliar: s3Tables.auxTbl[0][i]			
Nombre	Registro (índice)	Dirección Modbus	Descripción
SPI_TRANSACTION_COUNT_L	0	40067	Contador de transacciones SPI
SPI_TRANSACTION_COUNT_H	1	40068	
SPI_ERROR_COUNT	2	40069	Contador de errores SPI
SPI_EXCHANGE_TIME	3	40070	Tiempo de ejecución de función de intercambio de datos SPI en μ s
SPI_CYCLE_TIME	4	40071	Tiempo entre ciclos de intercambio de datos SPI en μ s
AUX_MB_MASTER_TOTAL_POLLS	5	40072	Solicitudes totales enviadas al dispositivo esclavo
AUX_MB_MASTER_ERR_COUNT	6	40073	Contador de errores en solicitudes al equipo esclavo
AUX_MB_MASTER_RETRY_COUNT	7	40074	Contador de reintentos de conexión

Registros de la tabla auxiliar: <code>s3Tables.auxTbl[0][i]</code>			
Nombre	Registro (índice)	Dirección Modbus	Descripción
AUX_MB_SLAVE_HR_READS	8	40075	Operaciones de lectura de HOLDING REGISTERS desde SCADA
AUX_MB_SLAVE_HR_WRITES	9	40076	Operaciones de escritura de HOLDING REGISTERS desde SCADA
AUX_MB_SLAVE_COIL_READS	10	40077	Operaciones de lectura de COIL REGISTERS desde SCADA
AUX_MB_SLAVE_COIL_WRITES	11	40078	Operaciones de escritura de COIL REGISTERS desde SCADA
AUX_MB_SLAVE_INPUT_READS	12	40079	Operaciones de lectura de INPUT REGISTERS desde SCADA
AUX_MB_SLAVE_STATUS_READS	13	40080	Operaciones de lectura de STATUS (DISCRETE) REGISTERS desde SCADA
AUX_CPU_TEMPERATURE	14	40081	Temperatura medida de la CPU en °C (multiplicada por 100)
ESPACIO DISPONIBLE (40082 AL 40109)			
AUX_RTC_YEAR	43	40110	Configuración/Visualización RTC (AÑO)
AUX_RTC_MONTH	44	40111	Configuración/Visualización RTC (MES)
AUX_RTC_DAY	45	40112	Configuración/Visualización RTC (DÍA)
AUX_RTC_HOUR	46	40113	Configuración/Visualización RTC (HORA)
AUX_RTC_MINUTE	47	40114	Configuración/Visualización RTC (MINUTOS)
AUX_RTC_SECOND	48	40115	Configuración/Visualización RTC (SEGUNDOS)
AUX_PID_CALLBACK_TIME	49	40116	Visualización del tiempo entre llamadas a la función callback del PID (en ms)

- ✓ Se modificó la función `system_logInput()`, para que el modo de escritura en el archivo `sys_log.log` sea configurable de acuerdo al valor de la macro:

```
#define USE_LOG_BACKUP_FILE 0
```

Si este valor vale 1, la escritura se efectúa realizando el backup de `sys_log.log` en `sys_log.bak`; lo cual es el método más seguro, pero también más lento y más agresivo para con la memoria flash. Si el valor de esta macro se establece a 0, la escritura se realiza sin hacer ningún

respaldo del archivo, lo que permite una escritura mucho más rápida, consume menos recursos, pero también es un método inseguro en caso de pérdida de energía.

Cambios realizados hasta el 31-10-2023:

- ✓ Se agregó el código necesario para realizar la configuración de la máscara de subred; para ello se reemplazó la asignación de registros reservados para la dirección IP de un segundo módulo ethernet (CFG_IP1 no implementado) y se renombraron como CFG_SUBNET_MASK:

```
#define CFG_IP0 &s3Tables.configTbl[0][2] //IP address for E
#define CFG_SUBNET_MASK &s3Tables.configTbl[0][4] //Subnet Mask
#define CFG_IP2 &s3Tables.configTbl[0][6] //IP address for W
```

- Las funciones de inicialización ethernet_init(), así como las funciones set_wifi_STA_ip() y set_wifi_AP_ip() fueron modificadas para tener en cuenta los valores almacenados en CFG_SUBNET_MASK (registros modbus 40021 y 40022) y configurar la máscara de subred para las interfaces de red.
 - Se agregó la máscara de subred predeterminada (255.255.255.0) a los valores por defecto de la tabla de configuración, los cuales son cargados mediante la función set_configDefaults(), la cual es invocada al presionar el pulsador de Master Reset.
- ✓ Se agregó la funcionalidad de borrado del archivo sys_log.log; para lograr esto se tomaron las siguientes acciones:

- Se crearon las funciones:

```
> esp_err_t clear_systemLog(void){ ...
> esp_err_t systemLog_copy2SD(void){ ...
```

- La función clear_systemLog() realiza las siguientes operaciones:
 - Realiza una llamada a la función systemLog_copy2SD(), cuyo propósito es el respaldo del archivo sys_log.log en la tarjeta micro SD. (Esta funcionalidad no está implementada actualmente y se encuentra fuera del alcance de éste trabajo, sin embargo es un prototipo de función vacío que permite agregar posteriormente el código necesario para efectuar el respaldo del archivo).
 - Borra el archivo sys_log.log de la partición FAT.
 - Crea un archivo sys_log.log nuevo, el cual contendrá la cabecera del archivo (que muestra información inicial relacionada al registro del sistema).
 - Genera un mensaje de advertencia que señala que el archivo sys_log.log ha sido borrado.
- En la tarea mb_event_check_task(), se agregó el código que permite realizar el borrado del archivo a través del SCADA. De igual manera se asignó el registro modbus 40082 de la tabla auxiliar:

```
#define AUX_SYS_LOG_CLEAR s3Tables.auxTbl[0][15]
```

De manera que al escribir el valor 1 desde el SCADA en dicho registro, se realiza una llamada a la función clear_systemLog(). Esta acción genera mensajes de advertencia en

consola, indicando que se ha recibido una solicitud de borrado del archivo y finalmente se muestra el mensaje de confirmación de borrado. El valor del registro AUX_SYS_LOG_CLEAR es restaurado a 0 después de la ejecución de esta tarea.

Este es el resumen de los cambios incorporados al proyecto hasta la fecha actual (31-10-2023). A partir de esta fecha se realiza el cierre de la versión 1.0 de la Remota A2SCP.

Las tablas de configuración y auxiliar definitivas para esta versión se muestran a continuación:

Registros de la tabla de configuración: s3Tables.configTbl[0][i]			
Nombre	Registro (índice)	Dirección Modbus	Descripción
CFG_RUN_PGM	0	40017	Modo de ejecución o modo de programación
CFG_OP_MODE	1	40018	Selección del método de producción
CFG_IP0	2	40019	Dirección IP Ethernet 0
	3	40020	
CFG_SUBNET_MASK	4	40021	Subnet Mask
	5	40022	
CFG_IP2	6	40023	Dirección IP WiFi (Station)
	7	40024	
CFG_IP3	8	40025	Dirección IP WiFi (AP)
	9	40026	
CFG_GW	10	40027	Network Gateway
	11	40028	
CFG_DHCP	12	40029	Dirección IP estática o dinámica
CFG_MB_MASTER_INTERFACE	13	40030	Modbus maestro TCP o RTU
CFG_MB_MASTER_BAUDRATE_H	14	40031	Velocidad UART2 (Modbus master)
CFG_MB_MASTER_BAUDRATE_L	15	40032	
CFG_MB_SLAVE_INTERFACE	16	40033	Modbus esclavo TCP o RTU
CFG_MB_SLAVE_BAUDRATE_H	17	40034	Velocidad UART1 (Modbus slave)
CFG_MB_SLAVE_BAUDRATE_L	18	40035	
CFG_SLAVE_IP	19	40036	Dirección IP del equipo esclavo (Modbus TCP)
	20	40037	
CFG_GL_TMR_INTERVAL	21	40038	Tiempo de ejecución timer de acumulación de gas
CFG_GL_FILTER_ALPHA	22	40039	Constante de tiempo del filtro de primer orden para señal de flujo de gas (multiplicada x 1000)
CFG_GL_PID_TMR_INTERVAL	23	40040	Tiempo de ejecución timer de PID
CFG_WIFI_MODE	24	40041	Selección de modo WiFi 0 --> STA, 1 --> AP, 2 --> STA+AP, 3 --> No WiFi

Registros de la tabla de configuración: <code>s3Tables.configTbl[0][i]</code>			
Nombre	Registro (índice)	Dirección Modbus	Descripción
ESPACIO DISPONIBLE (40042 AL 4057)			
CFG_GL_PID_SP	41	40058	Setpoint para el controlador PID
CFG_GL_PID_KP	42	40059	Ganancia proporcional * 1000 (PID)
CFG_GL_PID_KI	43	40060	Ganancia integral * 1000 (PID)
CFG_GL_PID_KD	44	40061	Ganancia derivativa * 1000 (PID)
CFG_GL_PID_N	45	40062	Constante de filtro N * 1000 (PID)
CFG_GL_PID_CP	46	40063	Coeficiente de activación control proporcional (0 ó 1)
CFG_GL_PID_CI	47	40064	Coeficiente de activación control integral (0 ó 1)
CFG_GL_PID_CD	48	40065	Coeficiente de activación control derivativo (0 ó 1)
CFG_REMOTA_LOG_LEVEL	49	40066	Nivel de log para mensajes en consola serial

Registros de la tabla auxiliar: <code>s3Tables.auxTbl[0][i]</code>			
Nombre	Registro (índice)	Dirección Modbus	Descripción
SPI_TRANSACTION_COUNT_L	0	40067	Contador de transacciones SPI
SPI_TRANSACTION_COUNT_H	1	40068	
SPI_ERROR_COUNT	2	40069	Contador de errores SPI
SPI_EXCHANGE_TIME	3	40070	Tiempo de ejecución de función de intercambio de datos SPI en μ s
SPI_CYCLE_TIME	4	40071	Tiempo entre ciclos de intercambio de datos SPI en μ s
AUX_MB_MASTER_TOTAL_POLLS	5	40072	Solicitudes totales enviadas al dispositivo esclavo
AUX_MB_MASTER_ERR_COUNT	6	40073	Contador de errores en solicitudes al equipo esclavo
AUX_MB_MASTER_RETRY_COUNT	7	40074	Contador de reintentos de conexión
AUX_MB_SLAVE_HR_READS	8	40075	Operaciones de lectura de HOLDING REGISTERS desde SCADA
AUX_MB_SLAVE_HR_WRITES	9	40076	Operaciones de escritura de HOLDING REGISTERS desde SCADA

Registros de la tabla auxiliar: s3Tables.auxTbl[0][i]			
Nombre	Registro (índice)	Dirección Modbus	Descripción
AUX_MB_SLAVE_COIL_READS	10	40077	Operaciones de lectura de COIL REGISTERS desde SCADA
AUX_MB_SLAVE_COIL_WRITES	11	40078	Operaciones de escritura de COIL REGISTERS desde SCADA
AUX_MB_SLAVE_INPUT_READS	12	40079	Operaciones de lectura de INPUT REGISTERS desde SCADA
AUX_MB_SLAVE_STATUS_READS	13	40080	Operaciones de lectura de STATUS (DISCRETE) REGISTERS desde SCADA
AUX_CPU_TEMPERATURE	14	40081	Temperatura medida de la CPU en °C (multiplicada por 100)
AUX_SYS_LOG_CLEAR	15	40082	Solicitar borrado del archivo sys_log.log (Escribir cualquier valor diferente de cero para borrar, el valor 0 es restaurado al realizar la operación)
ESPACIO DISPONIBLE (40083 AL 40109)			
AUX_RTC_YEAR	43	40110	Configuración/Visualización RTC (AÑO)
AUX_RTC_MONTH	44	40111	Configuración/Visualización RTC (MES)
AUX_RTC_DAY	45	40112	Configuración/Visualización RTC (DÍA)
AUX_RTC_HOUR	46	40113	Configuración/Visualización RTC (HORA)
AUX_RTC_MINUTE	47	40114	Configuración/Visualización RTC (MINUTOS)
AUX_RTC_SECOND	48	40115	Configuración/Visualización RTC (SEGUNDOS)
AUX_PID_CALLBACK_TIME	49	40116	Visualización del tiempo entre llamadas a la función callback del PID (en ms)

*** Fin de la Bitácora del Trabajo de Grado ***