**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 numAnTbls,     //Tablas de variables analógicas

                     uint8\_t numDigTbls,    //Tablas de variables digitales

                     uint8\_t numConfigTbls, //Tablas de configuración

                     uint8\_t numAuxTbls,    //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 matríz 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.

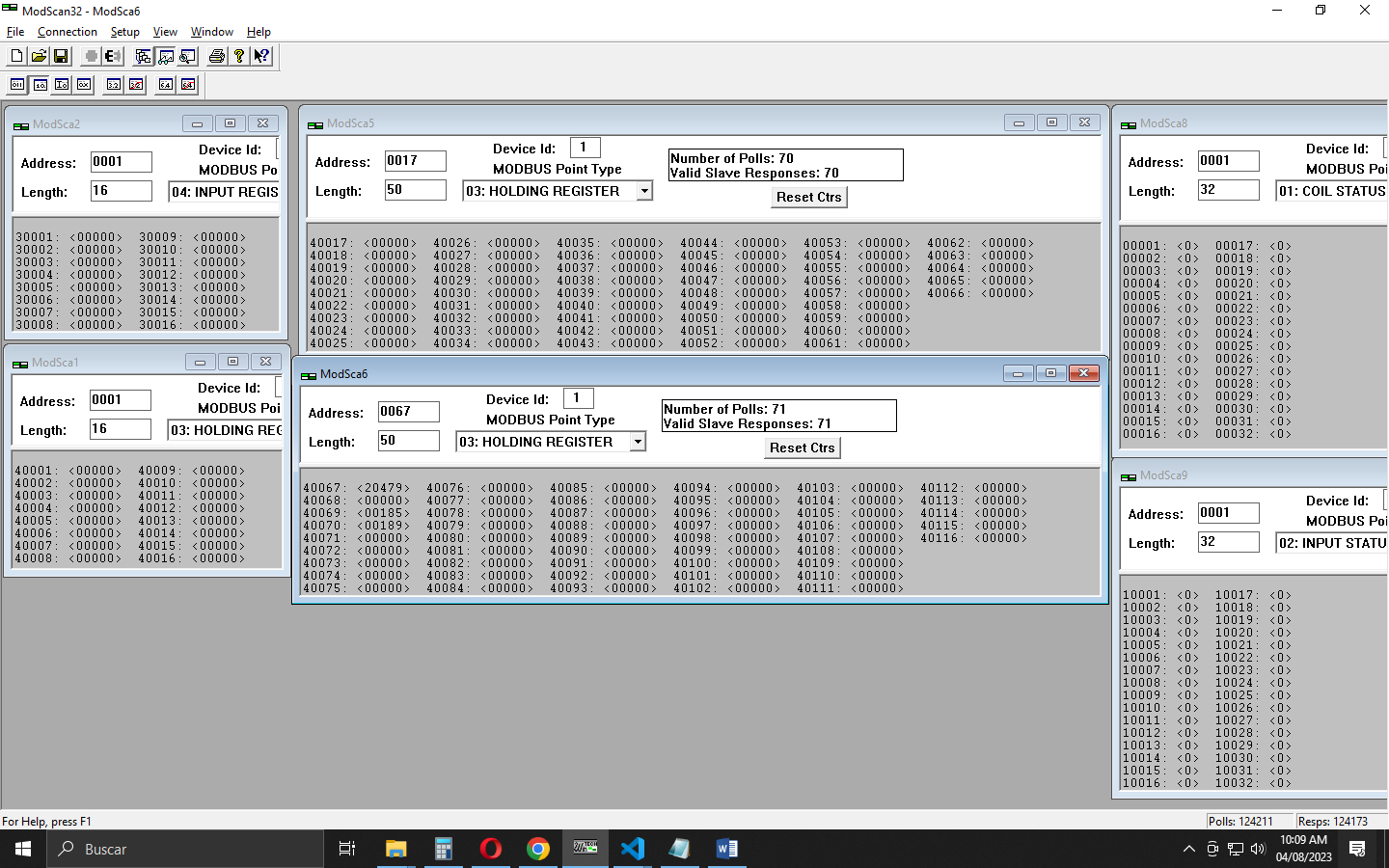
**Cambios realizados hasta el 04-08-2023:**

**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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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 spi\_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**](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

#ifdef 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 numAnTbls;     // Número de vectores analógicos

    uint8\_t numDigTbls;    // Número de vectores digitales

    uint8\_t numConfigTbls; // Número de vectores de configuración

    uint8\_t numAuxTbls;    // 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){

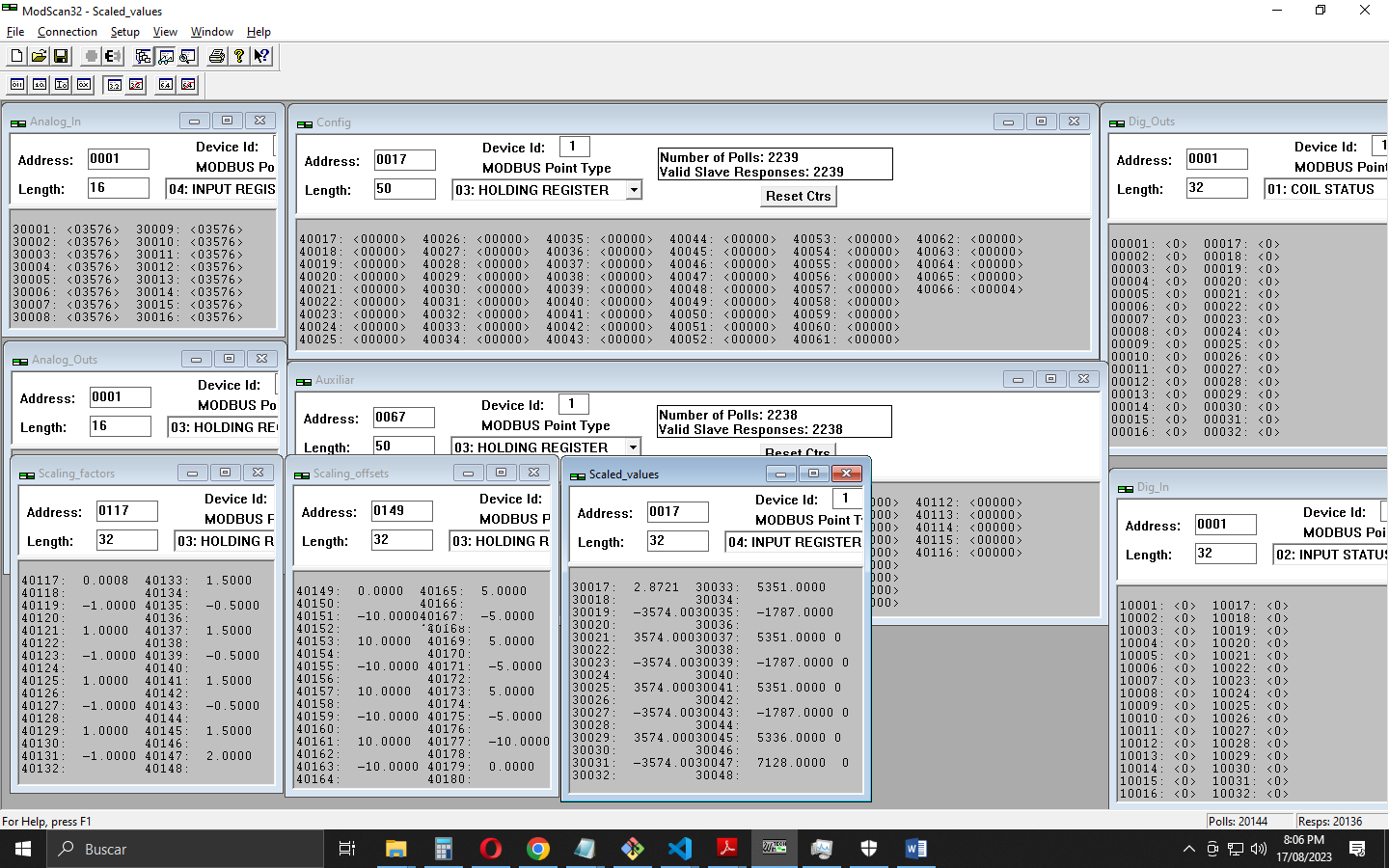
        ESP\_LOGE(TAG, "Error al asignar memoria!\n");

        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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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]
      * Direcciónes 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 extraida 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

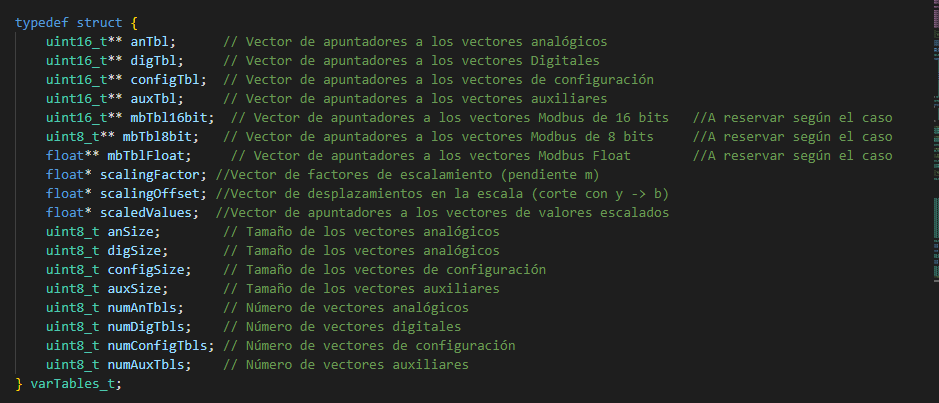
Los octetos más significativos en \*CFG\_IP0: 172.16 --> 0xAC10 (0xAC -> 172 y 10 -> 16)

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 restiutuye 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 ehecución.
        + 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:

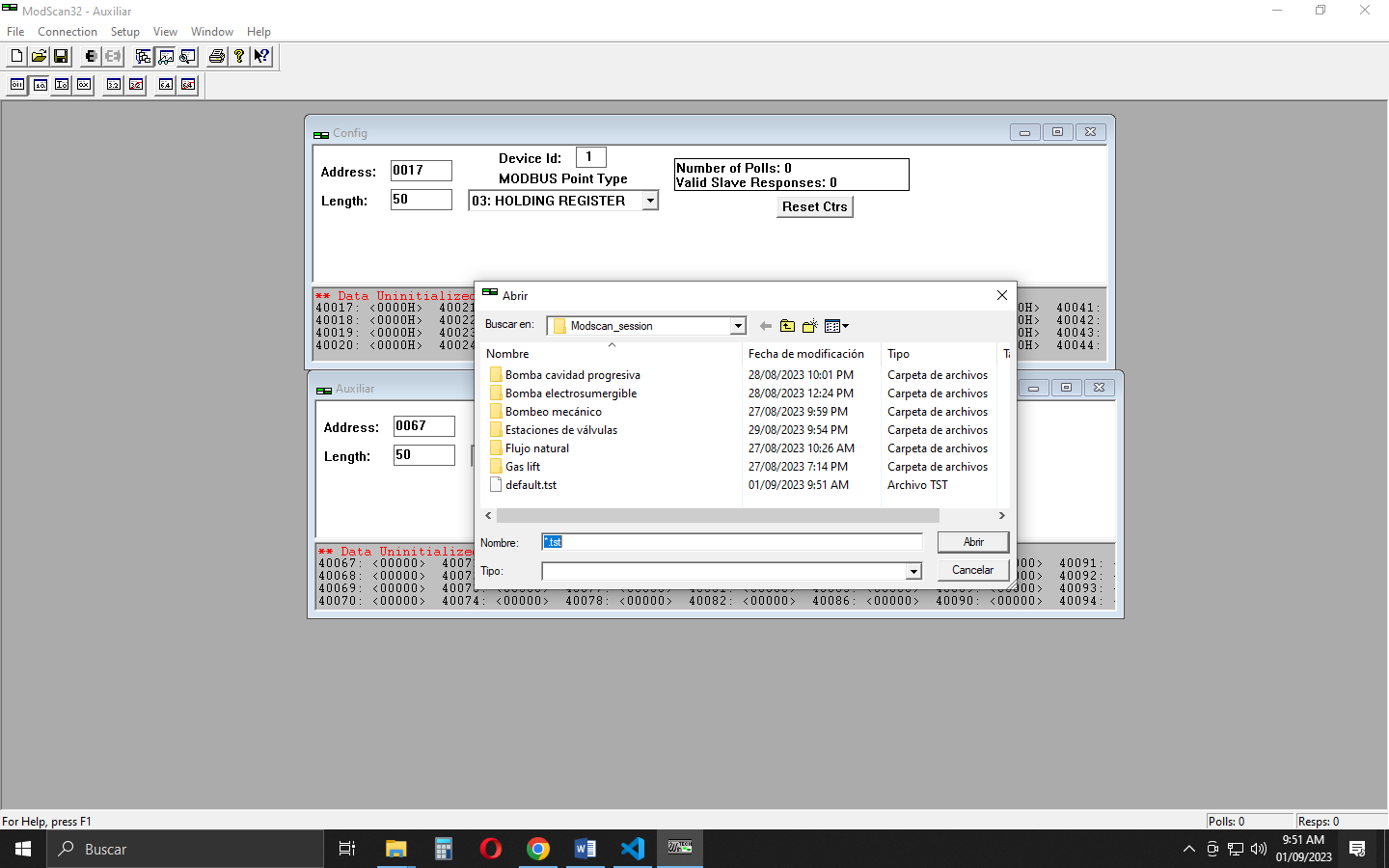


* Se creó la función: create\_modbus\_map(), la cual genera el mapa de memoria modbus dependiendo del modo de operación. Dicha función contempla la creación de un mapa modbus básico (que contiene las tablas comunes para todos los casos de operación, como son la tabla de configuración, la tabla auxiliar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

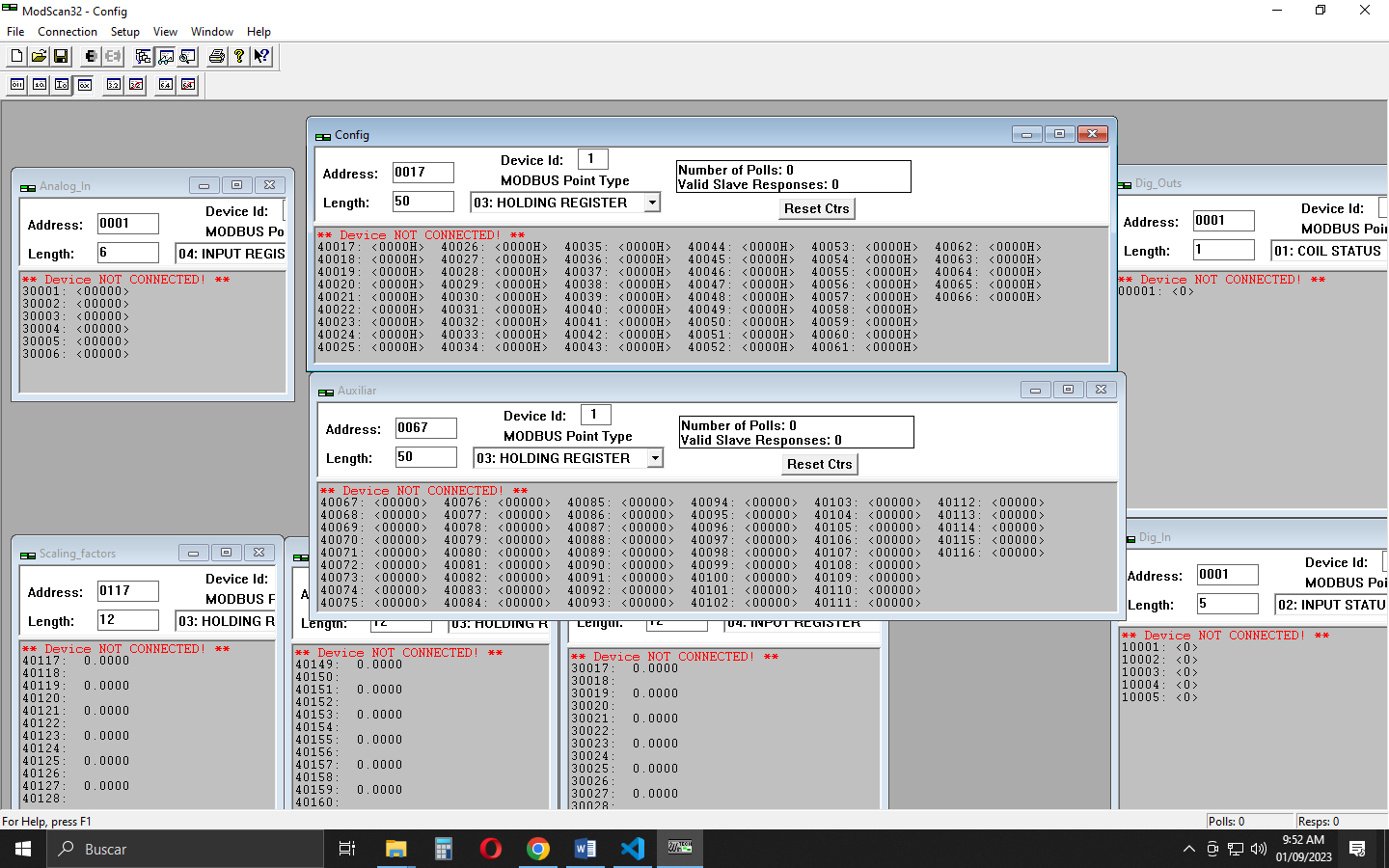
La creación del mapa modbus para cada uno de los 5 métodos de producción y para el caso de estaciones de válvulas, se realizó siguiendo las especificaciones del documento “Informe Técnico Filosofía de Control A2SCP Rev0\_Tesis\_ULA”. Para ello se generó el documento “Variables a considerar dependiendo 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 Electrosumergible 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 leidas y/o escritas a partir de la comunicación modbus master.

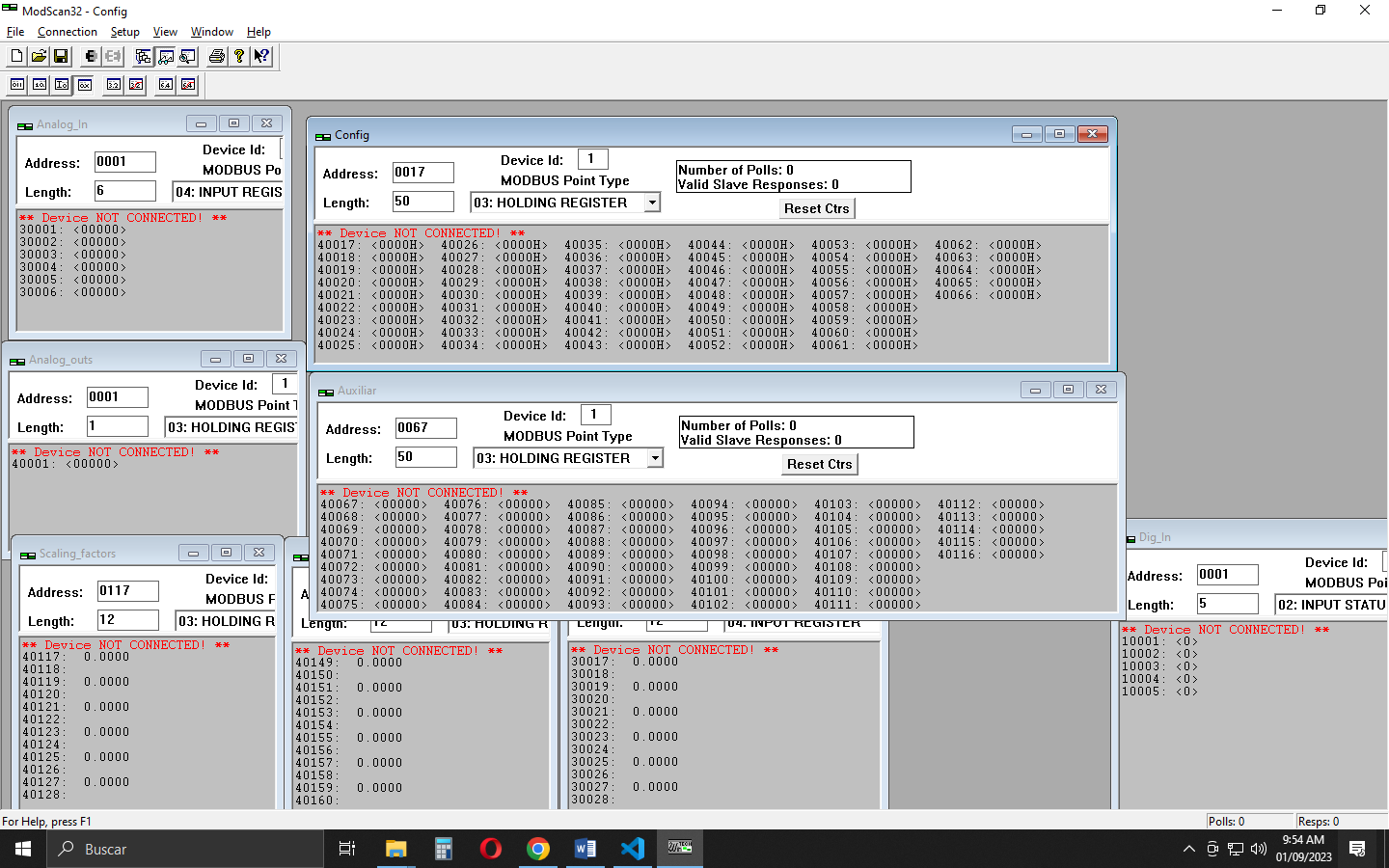
* La función create\_modbus\_map(), es llamada al ejecutarse la función modbus\_slave\_init(), que se ejecuta durante el inicio de la tarea app\_main().
* Se utilizó el software ModScan32, para crear las configuraciones necesarias para leer y escribir las variables modbus dependiendo del mapa creado 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.



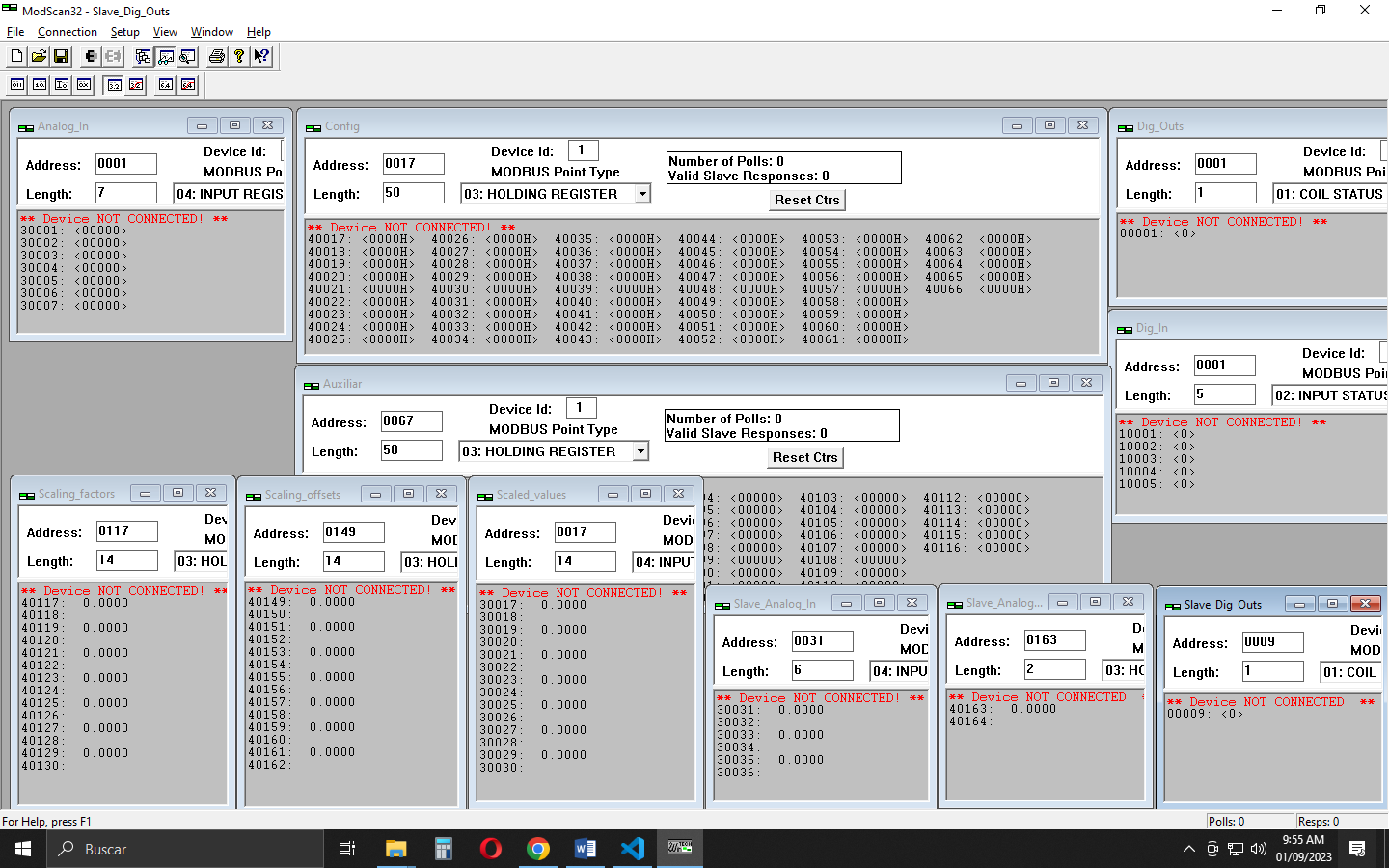
Flujo Natural:



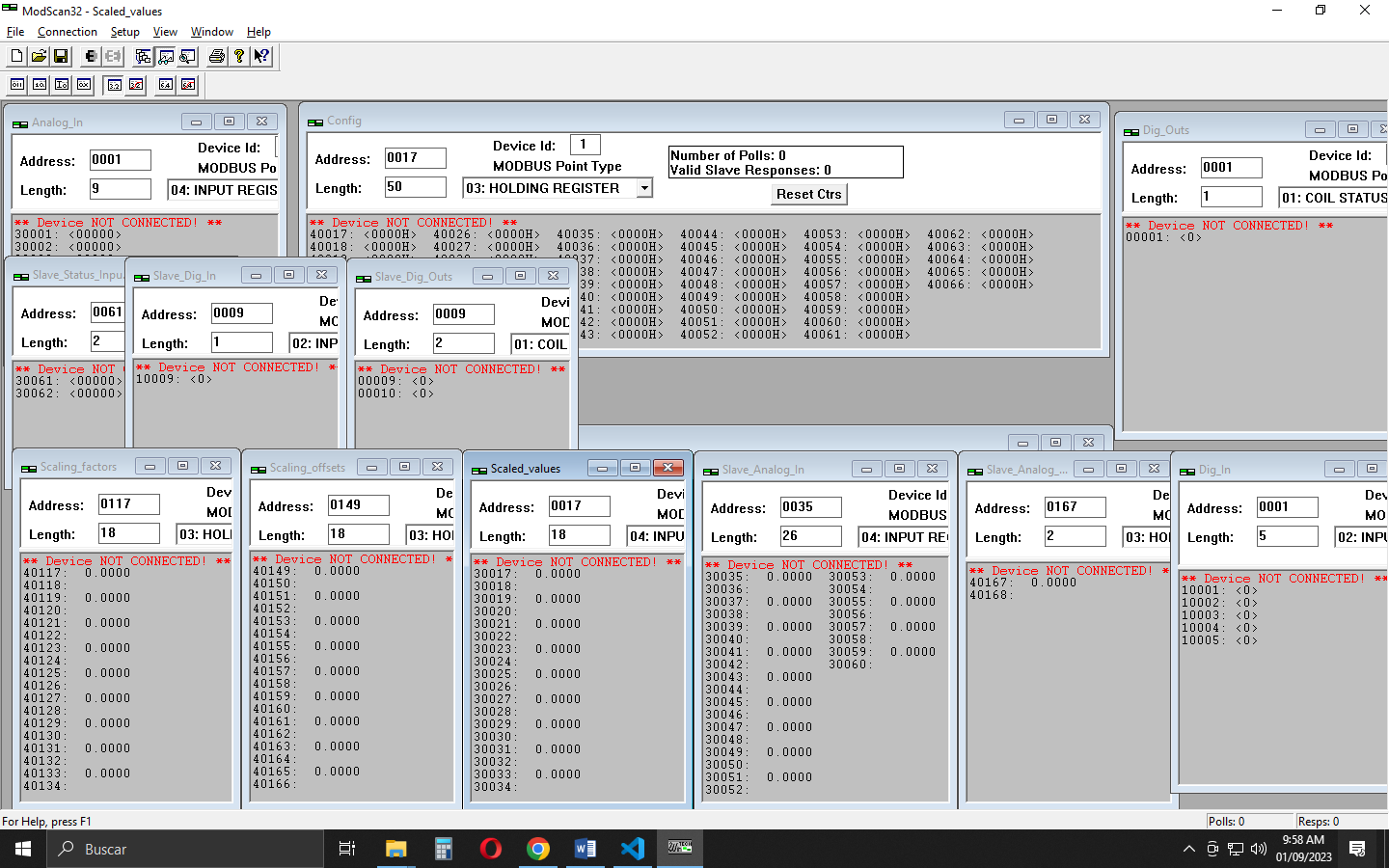
Gas Lift:



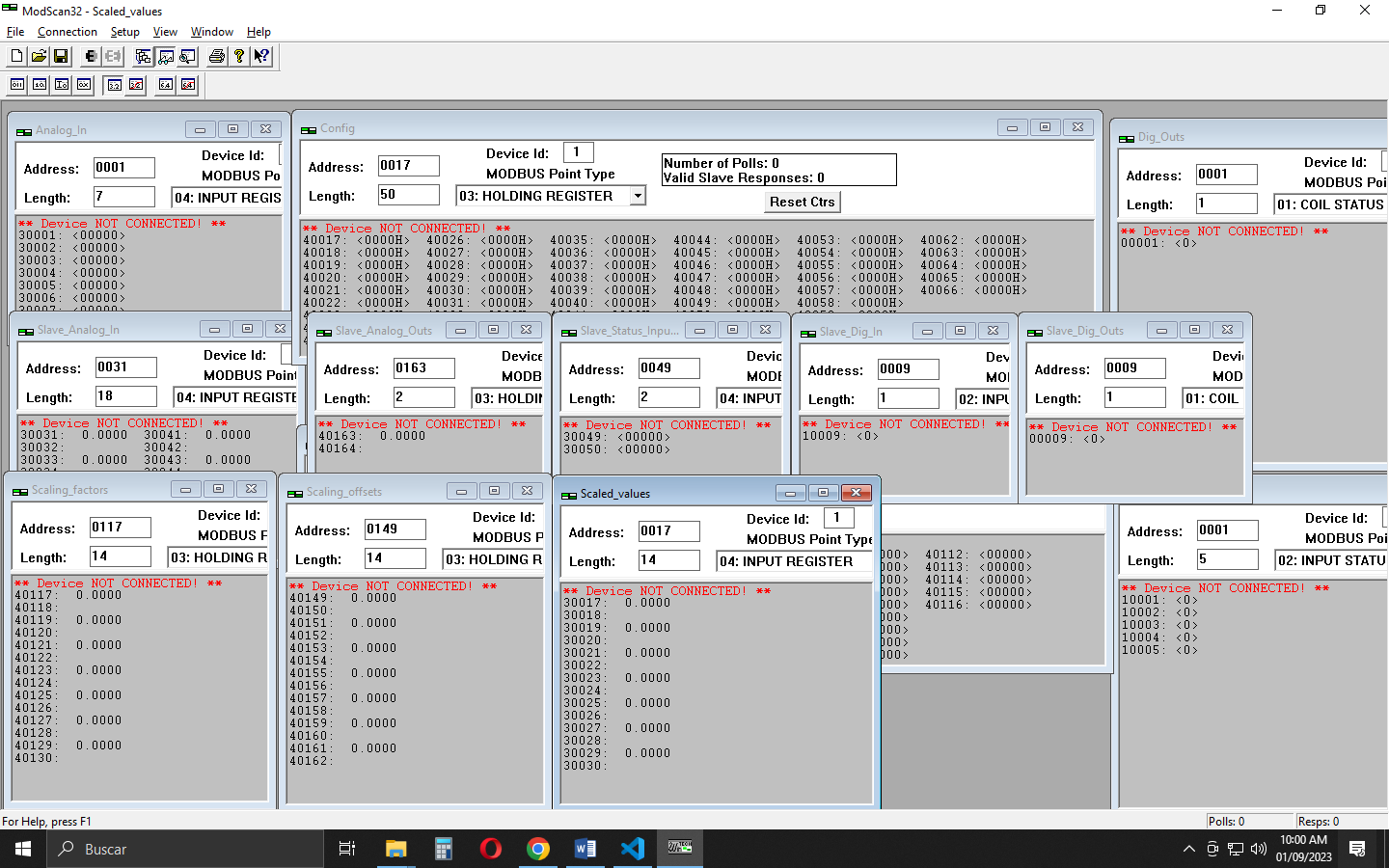
Bombeo Mecánico:



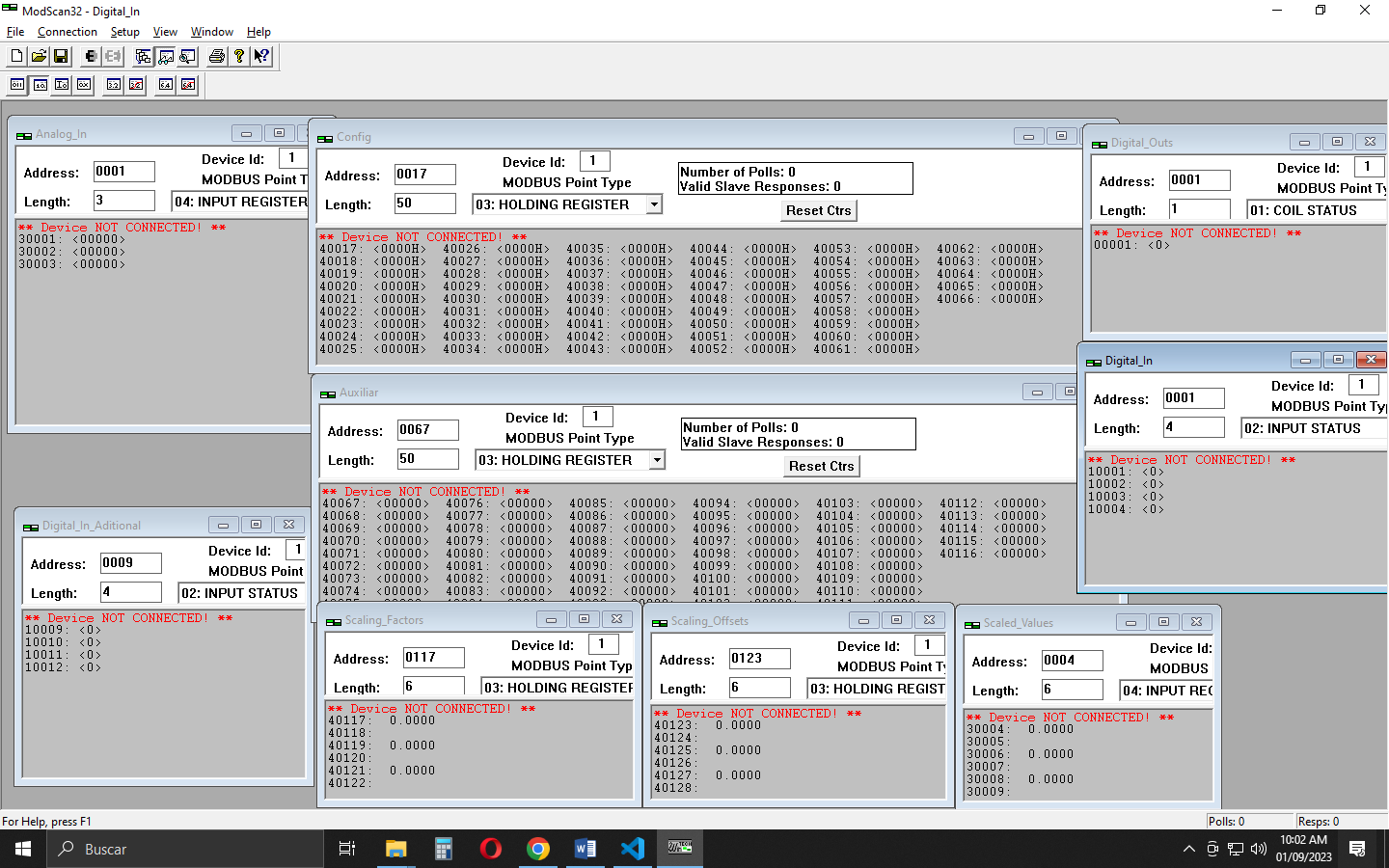
Bomba Electrosumergible:



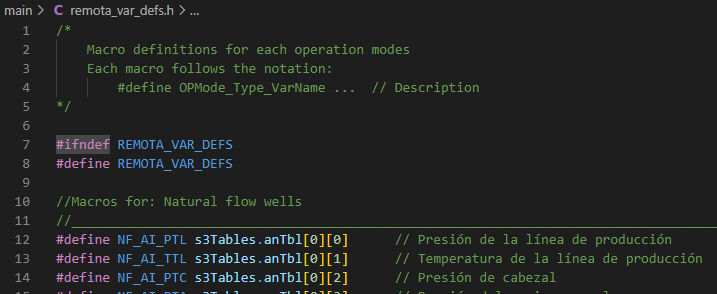
Bomba de Cavidad Progresiva:



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:

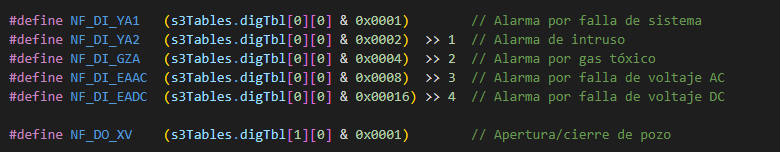


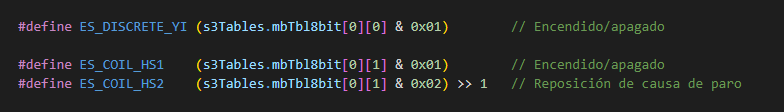
VarName

Type

OPMode

* 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:

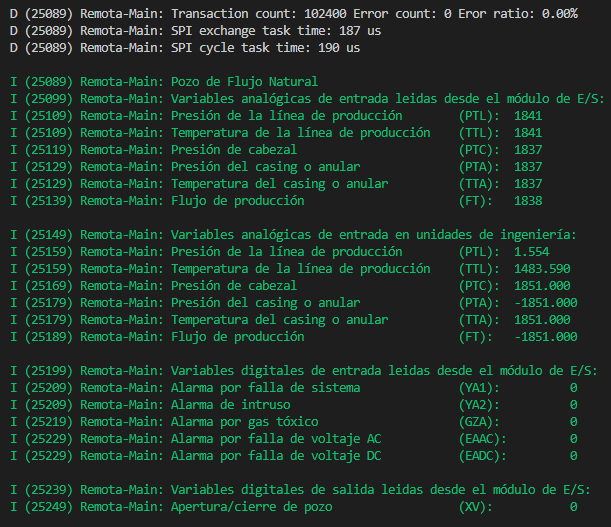




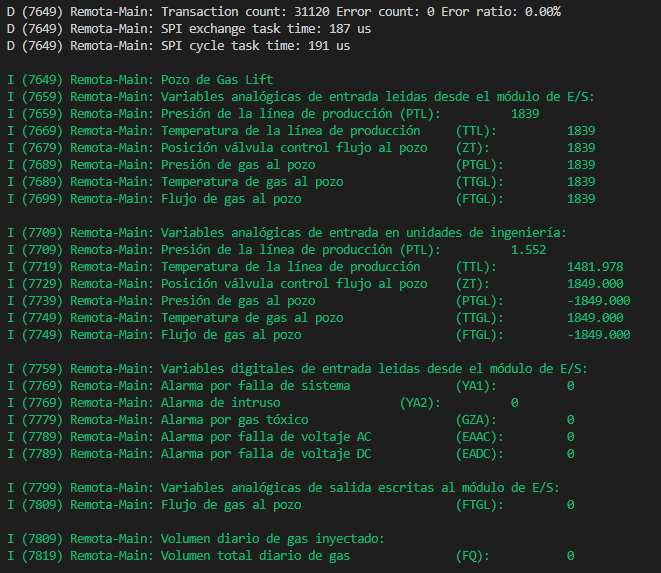
* Se reescribió parte del código del lazo infinito de la tarea app\_main(), particularmente donde se hace la selección (switch) para el modo de operación (CFG\_OP\_MODE) y así seleccionar las acciones a realizar dependiendo del método de producción. Para cada caso se escribió el código necesario para 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:

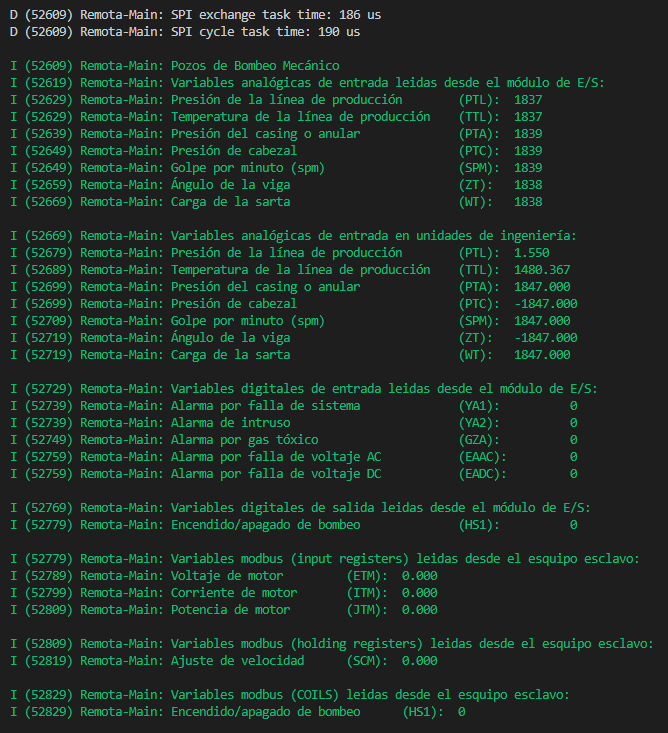
Flujo natural:



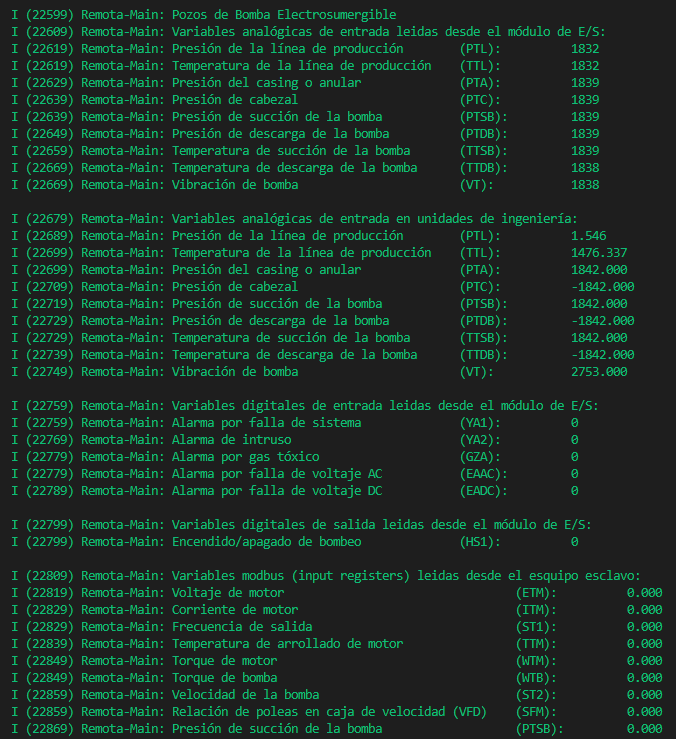
Gas Lift:

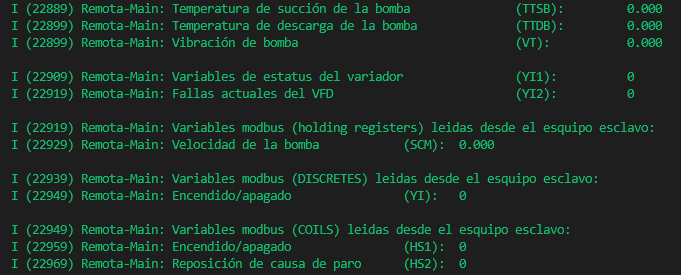


Bombeo Mecánico:

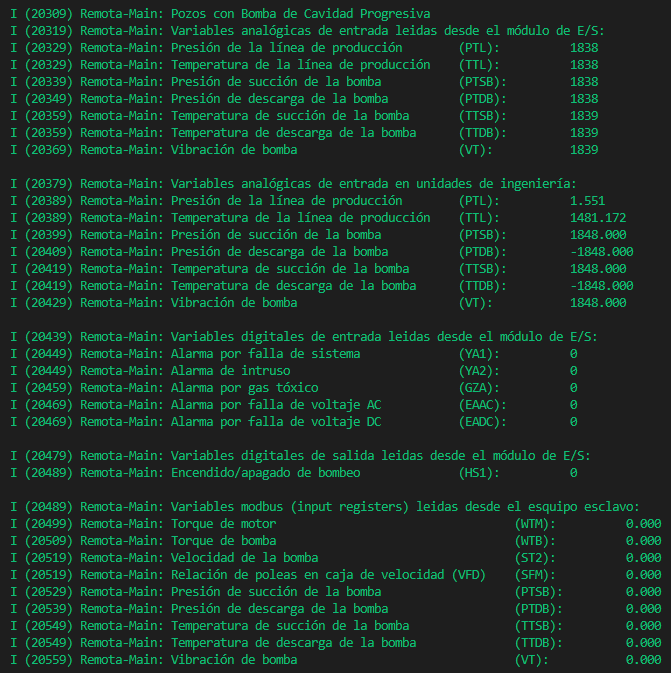


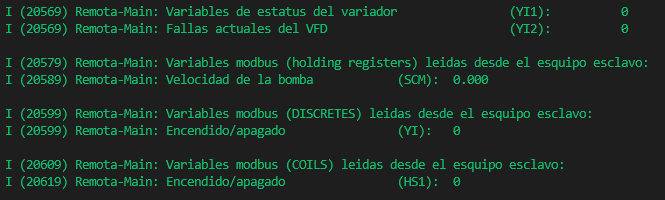
Bomba Electrosumergible:



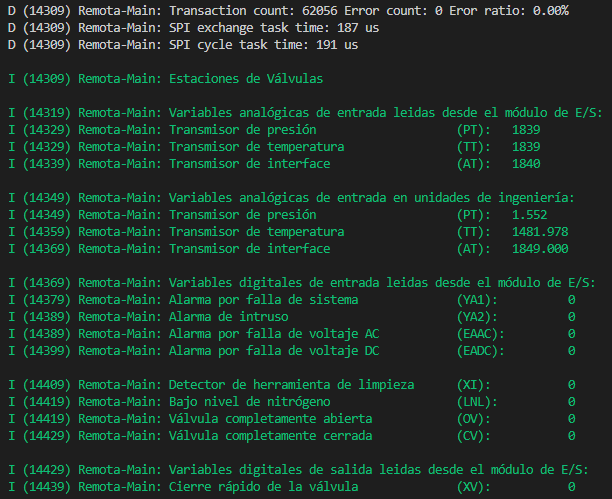


Bomba de Cavidad Progresiva:



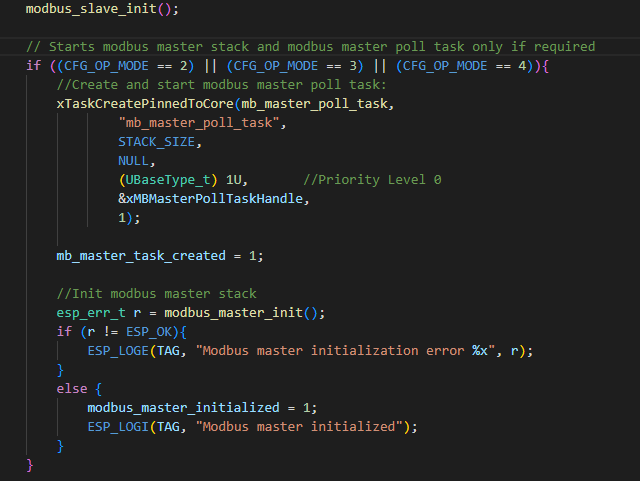


Estaciones de Válvulas:



**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 un 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 electrosumergible y pozos con bomba de cavidad progresiva.



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 appropiate 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) {

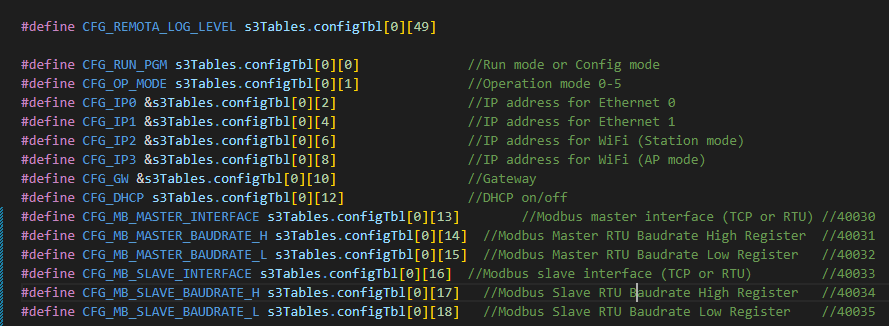
        ESP\_LOGE(TAG, "mb controller start fail, err=%x.", err);

    }

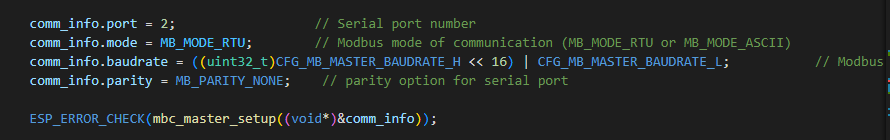
    return err;

}

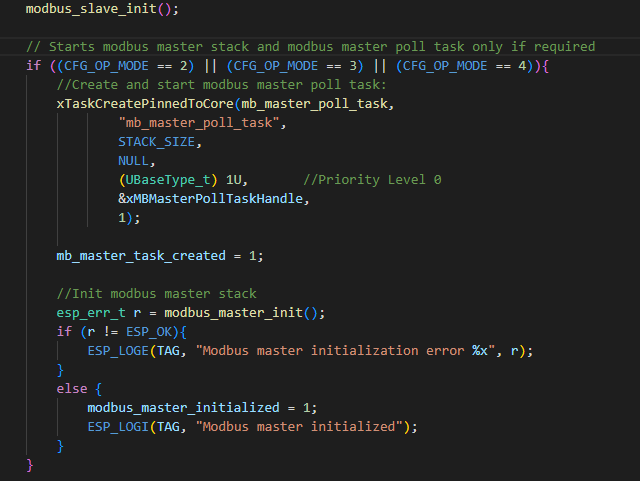
* + Se asignaron tres registros adicionales de la tabla de configuración, a fin de poder configurar la interfaz para el dispositivo esclavo (TCP/IP o RTU) y configurar la velocidad de la comunicación serial para la UART2. Dichas asignaciones se hicieron en el archivo remota\_globals.h:



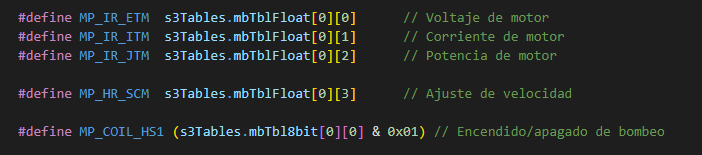
* + La función modbus\_master\_init() tiene en cuenta el registro CFG\_MB\_MASTER\_INTERFACE para decidir si inicia la pila modbus maestro TCP o la pila modbus maestro RTU, dependiendo del valor del registro; así, si:
    - CFG\_MB\_MASTER\_INTERFACE = 1, se inicia modbus master TCP
    - CFG\_MB\_MASTER\_INTERFACE = 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:



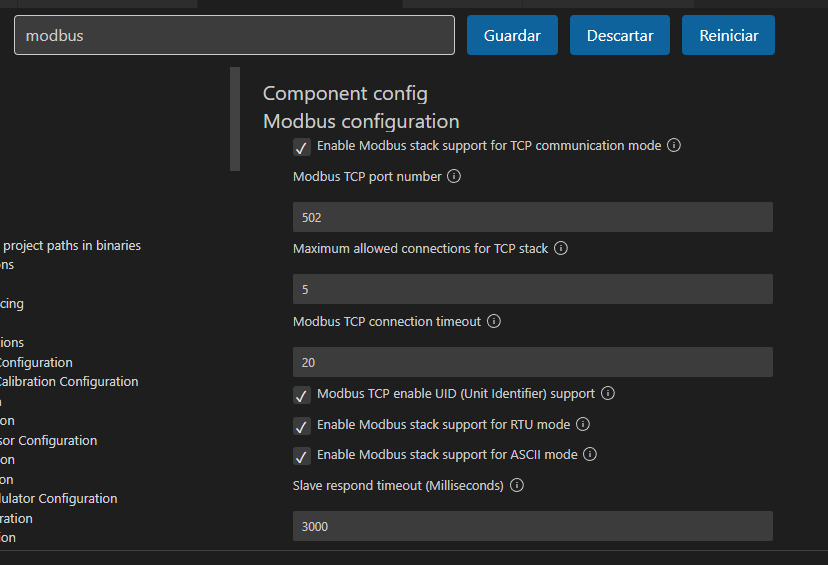
* + Hasta el momento el programa no tiene en cuenta el cambio de las demás configuraciones de la comunicación serial, como son los bits de parada y la paridad.
  + Para realizar la configuración, se incorporaron los casos necesarios en la tarea mb\_check\_event\_task(), de manera que al cambiarse los 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 Electrosumergible y pozos con Bomba de Cavidad 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.



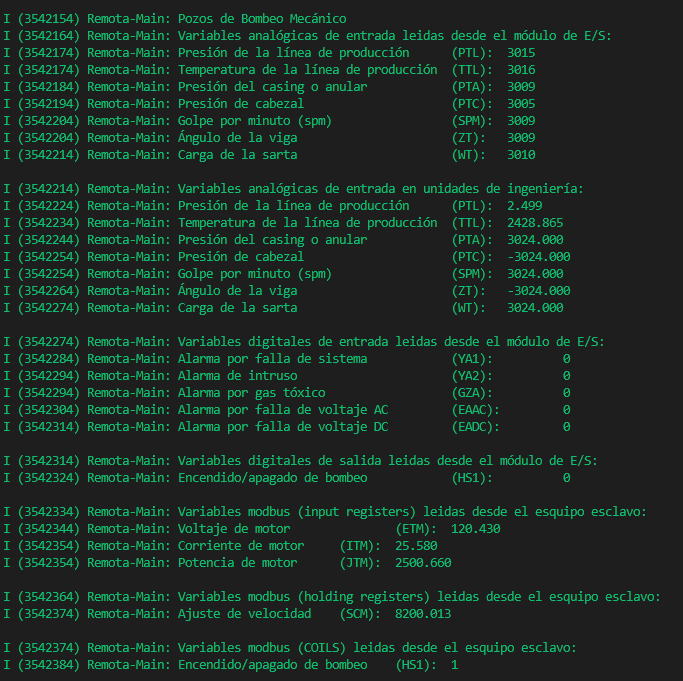
* + Dicha tarea utiliza la información del diccionario de características correspondiente al dispositivo esclavo que se está utilizando según el modo de operación para acceder a los datos del esclavo y realizar la lectura y escritura de los registros; los valores leídos (y escritos), se guardan en (o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



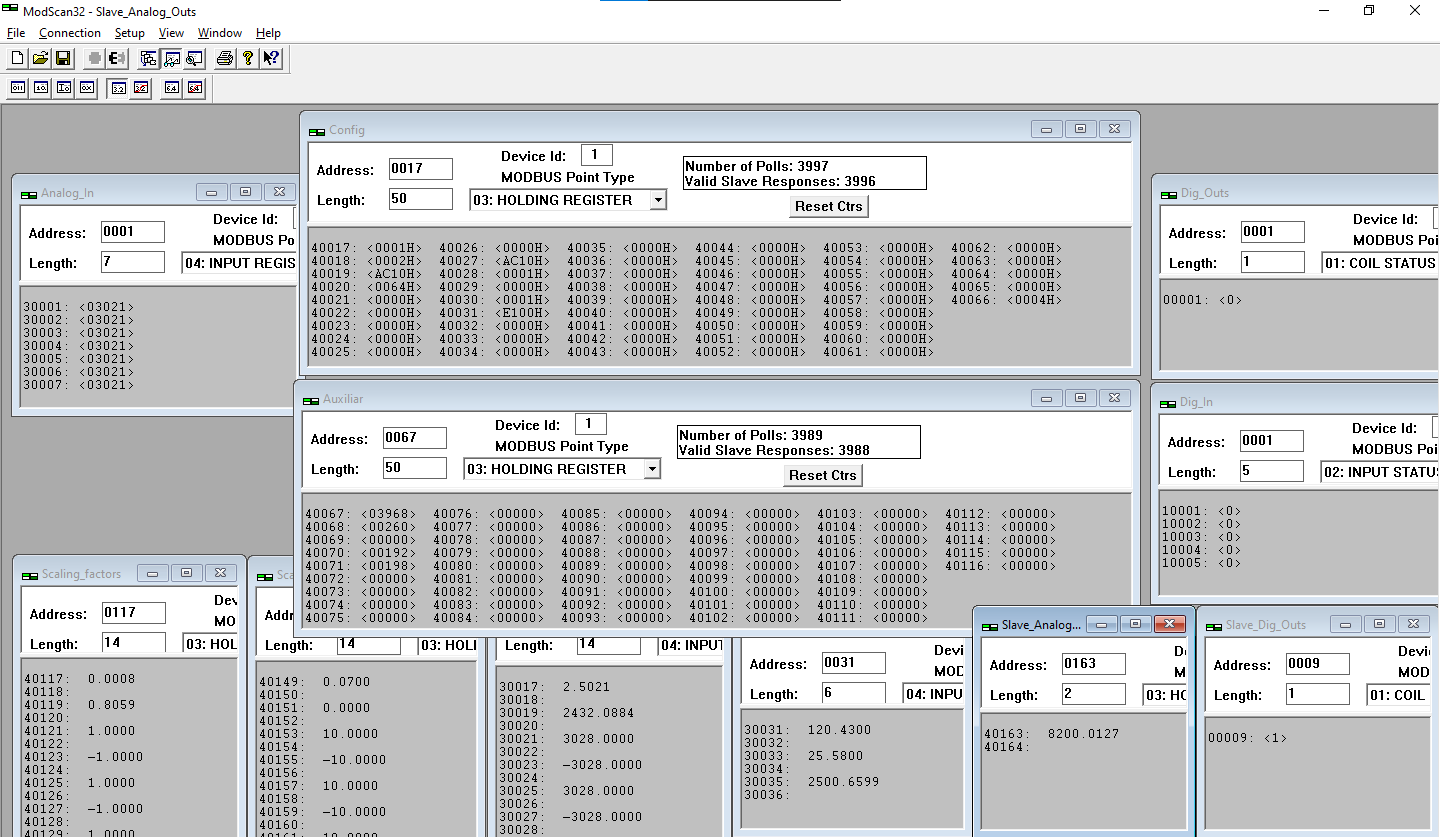
* + La tarea decide qué consultas hacer y qué datos escribir al esclavo, dependiendo del modo de operación. Dicha selección se hace a través de una sentencia switch.
  + Agregada la lógica para suspender o reanudar la nueva tarea al cambiar el estado del registro CFG\_RUN\_PGM. La suspensión o reactivación 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.

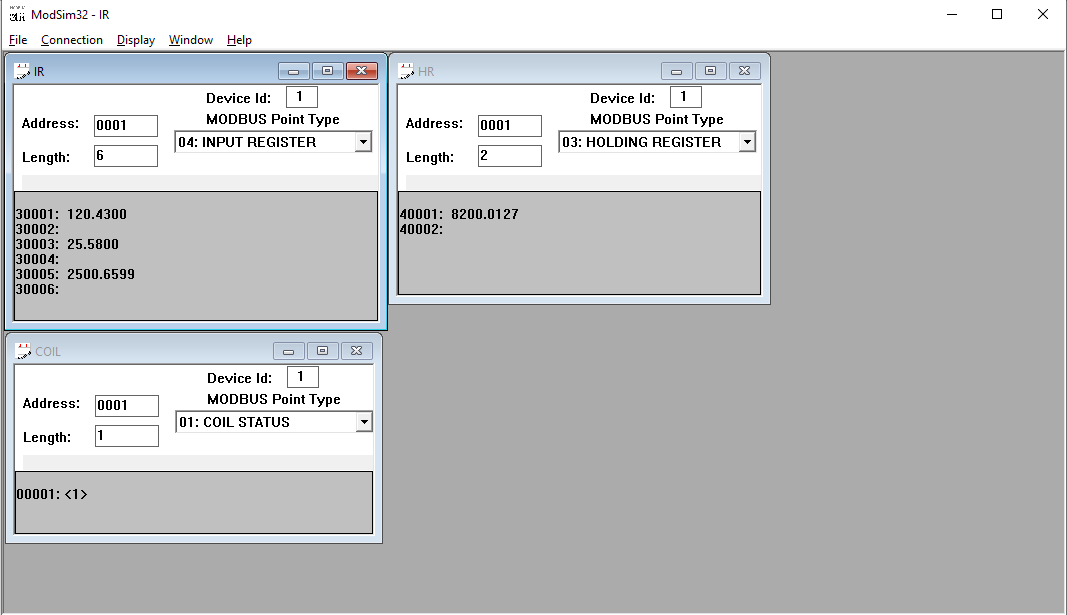


* + En la función de inicialización de la pila modbus esclavo (mb\_slave\_init()), en la definición de la estructura que contiene la información de la comunicación, se agregó el campo .slave\_uid = 1; a fin de configurar el UID del dispositivo esclavo y lograr que el simulador ModScan32 pueda comunicarse 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:



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

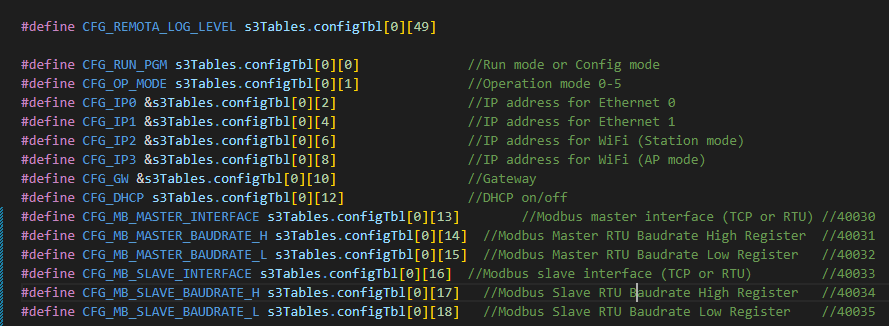




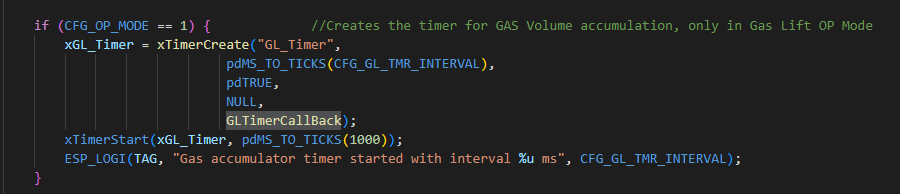
* + Se comprobó que tanto la inicialización de la pila modbus maestro, como la tarea mb\_master\_poll\_task, se ejecutan solo en los casos correctos.

**Cambios realizados hasta el 09-09-2023:**

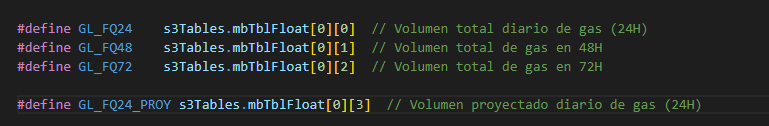
* Agregado soporte para la comunicación con el sistema SCADA a través del protocolo Modbus RTU, utilizando la UART1 a través de los pines TX=15 y RX=16.
  + Los pines originalmente asignados eran TX=17 y RX=18, sin embargo, se hizo necesario cambiarlos ya que al realizar las pruebas la comunicación no 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.



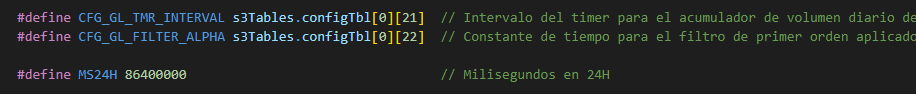
* + La utilización de estos registros se maneja de la misma forma que para el caso de Modbus master, utilizando el registro CFG\_MB\_SLAVE\_INTERFACE para decidir el tipo de interfaz y los registros CFG\_MB\_SLAVE\_BAUDRATE\_H y CFG\_MB\_SLAVE\_BAUDRATE\_L para establecer la velocidad.
  + Para implementar 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:
    - 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:
  + 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.



* 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.



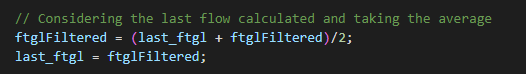
* Se asignaron los siguientes registros de la tabla de configuración para almacenar el intervalo de temporización del timer y el valor de la constante de tiempo para el filtro de primer orden:



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



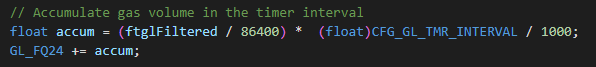
* + Se aplica la fórmula del trapecio, considerando el valor anterior del flujo:



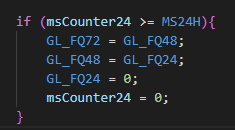
* + Se calculan el valor proyectado en base al flujo obtenido después de filtrar la señal:



* + 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:



* + 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:



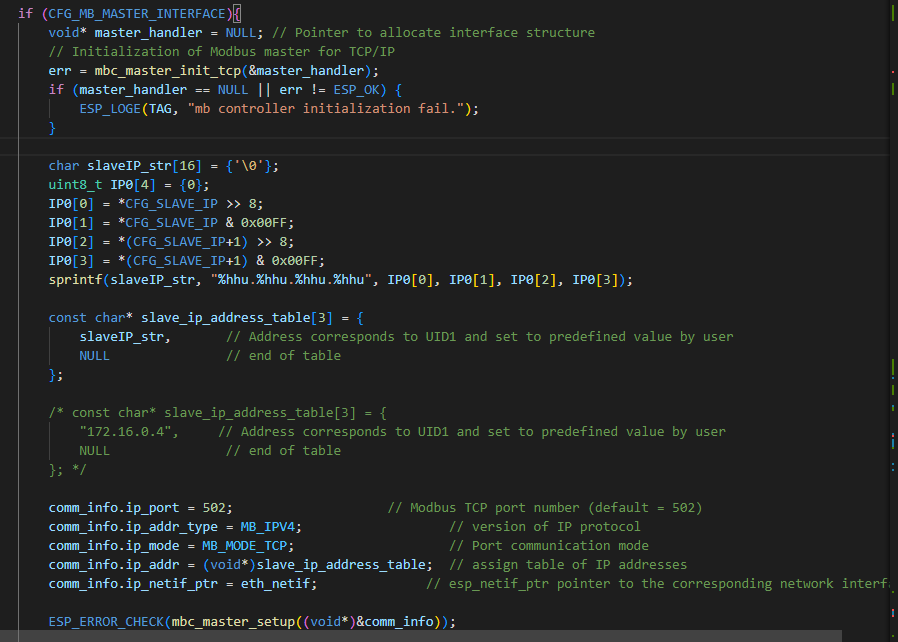
* Se implementó la lógica necesaria para que al modificar el registro CFG\_GL\_TMR\_INTERVAL, se active la bandera resetRequired, de modo que se produce un reinicio al volver al modo RUN. Dicho reinicio es necesario debido a que, al cambiar este valor, debe volver a crearse el timer. Para el valor CFG\_GL\_FILTER\_ALPHA, 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/α 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)



* + 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:

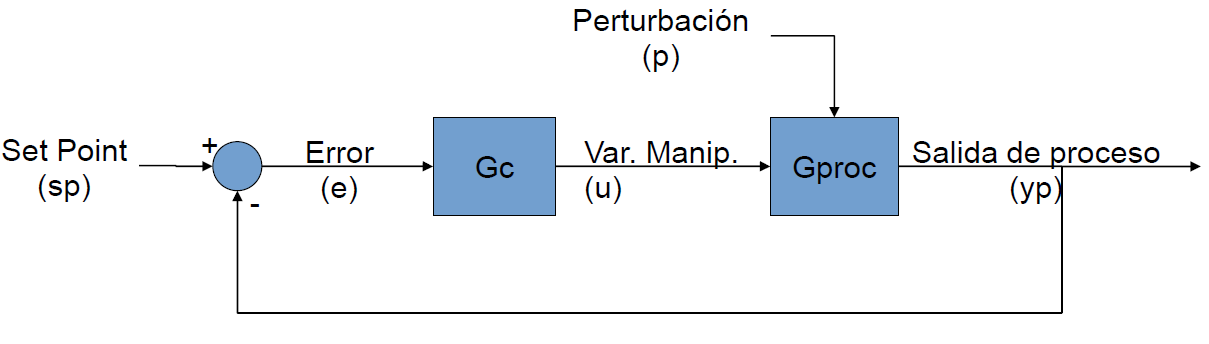


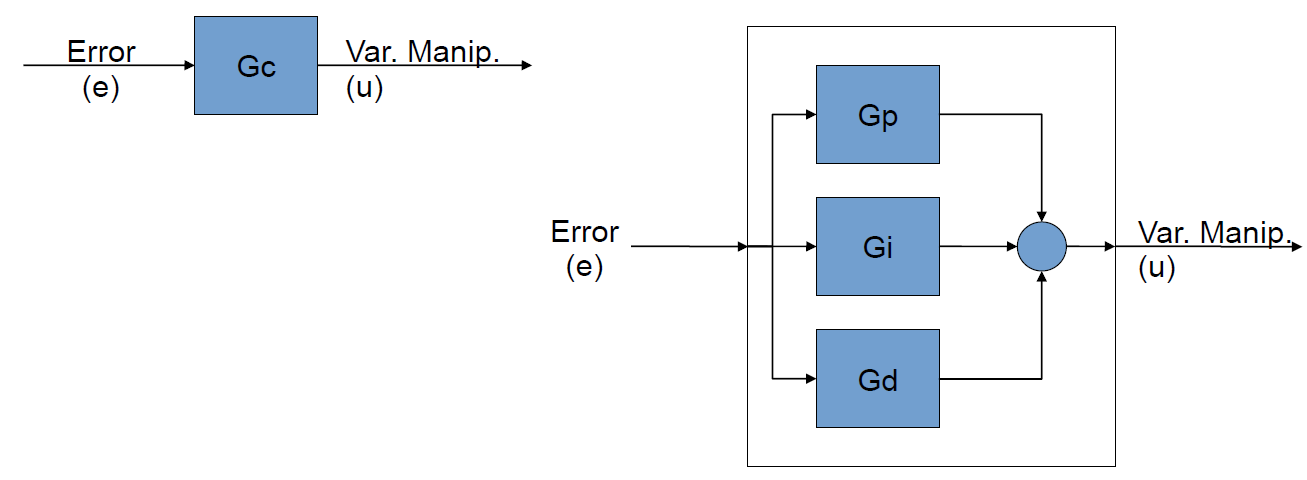
* Se agregó la lógica necesaria en el chequeador de eventos modbus para establecer la bandera resetRequired en caso de modificarse cualquiera de estos registros, produciéndose un reinicio al volver al modo RUN.
* La tabla de configuración se modificó ligeramente, los registros actualizados se muestran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

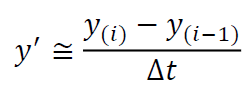
**Cambios realizados hasta el 13-10-2023:**

* 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\_RUNPGM = 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, dichop 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:





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



* + - 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

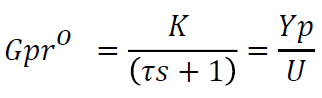
            uint16\_t high; // Registro alto

        } uint16Values;

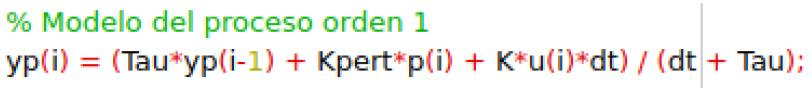
};

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 al cambiar el valor del registo 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 ejecición 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:



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



* 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;

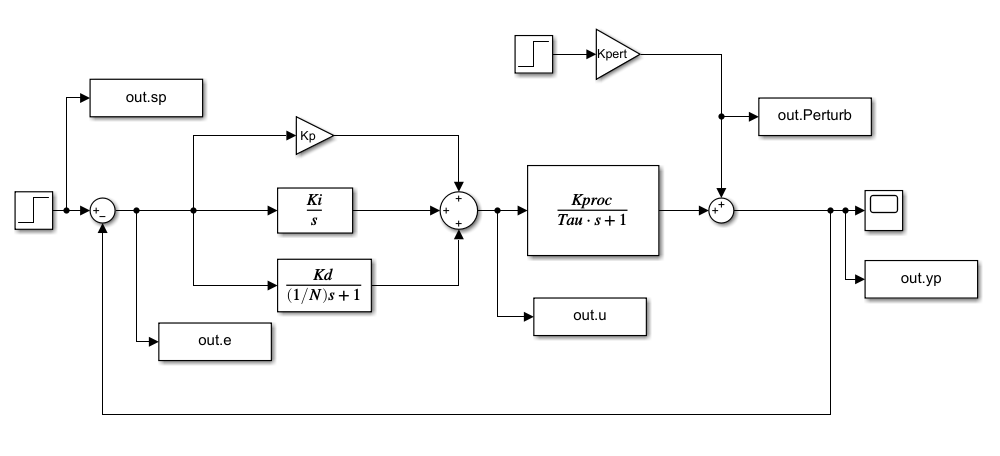
float Kpert = 0.1;

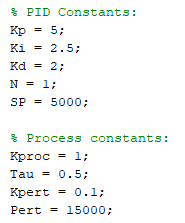
float Kproc = 1;

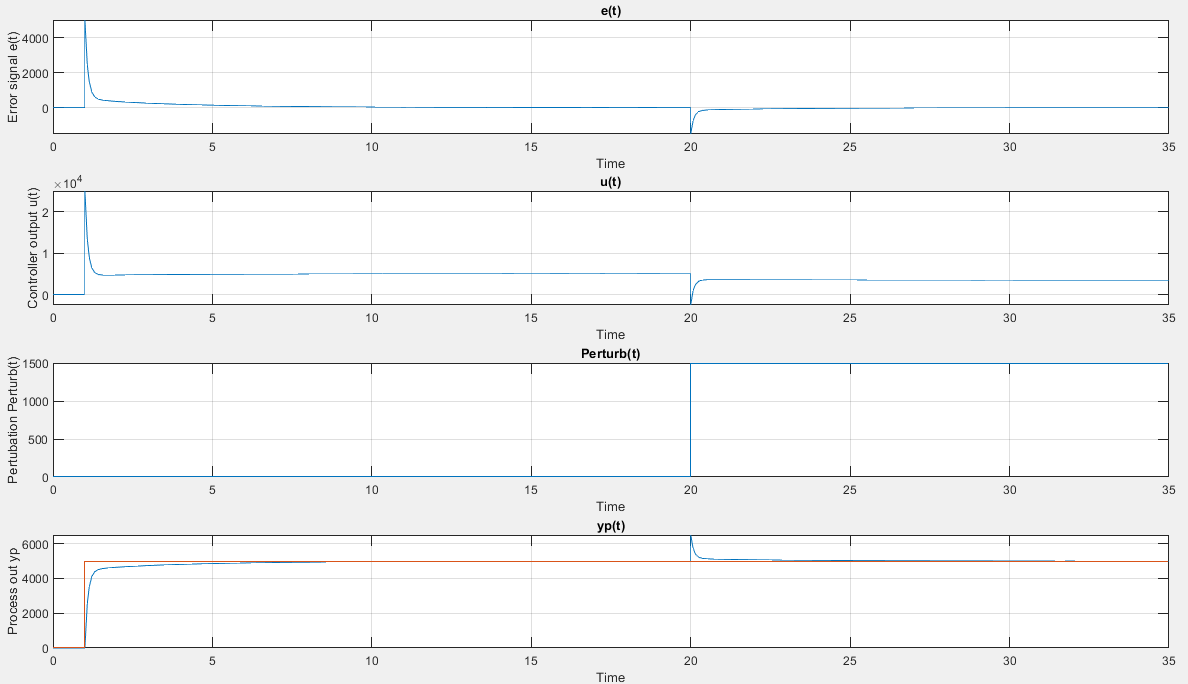
float yp = 0;

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.







* 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 (Kp, Ki y Kd), así como el filtro N y los coeficientes de activación Cp, Ci y Cd. 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 |  |
|  | 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) |
| 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 |