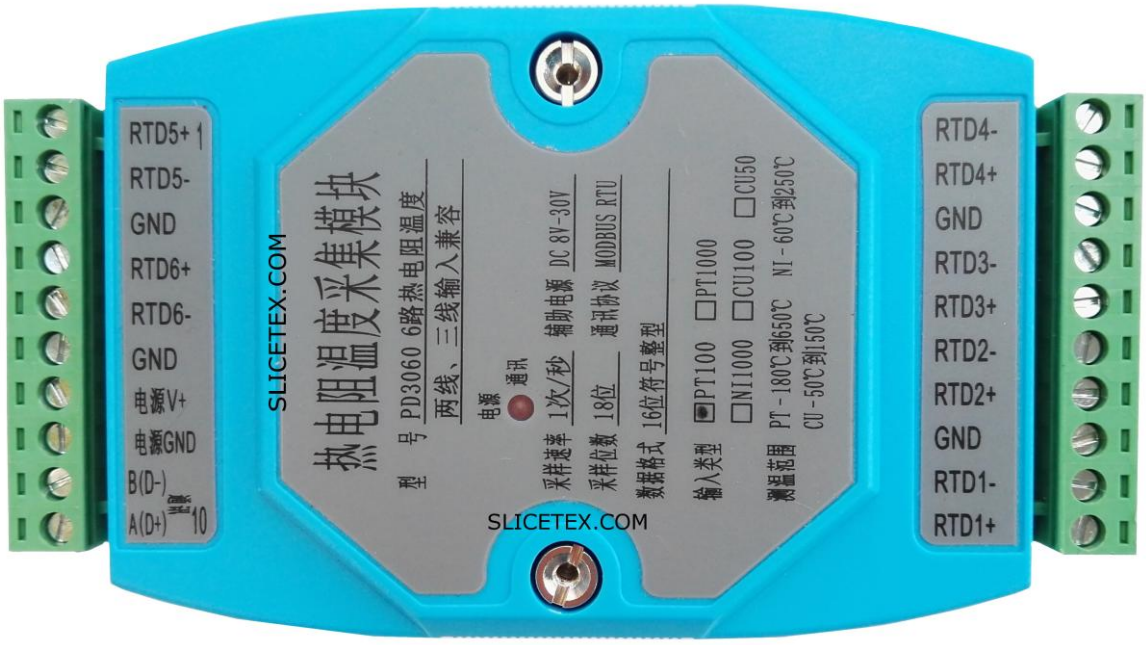
PD3060-PT100
Lector de 6 canales para sensores de temperatura PT100, conexión RS-485 (ModBus RTU)

Hoja de Datos

Autor: Ing. Boris Estudiez



1. Descripción General

El PD3060-PT100 es un módulo lector de sensores de temperatura PT100 (dispositivo termo-resistivo o RTD) que admite hasta 6 entradas para sensores. Puede conectar sensores PT100 de 2 y 3 cables. La lectura de temperatura se obtiene utilizando la interfaz de comunicación RS-485 mediante el protocolo ModBus RTU. Ideal para conectar a un PLC, HMI o sistema de monitoreo con soporte de dicha interfaz de comunicación.

2. Características de Hardware Principales

- Rango de Alimentación: 8 - 30 VDC
- Consumo: 0.6 a 1.6 Watts
- Entradas para 6 sensores PT100.
- Soporta conexión de PT100 con 2 y 3 cables.
- Rango de temperatura de lectura: -99.9 a 650 °C
- Resolución: 0.1 °C
- Exactitud: 0.5 °C
- Interfaz de comunicación: RS-485, half-duplex.
- Protocolo: ModBus RTU, módulo es esclavo.
- Velocidad máxima de comunicación: 19200 bps (por defecto 9600, 8N1).
- Gabinete para montaje en riel DIN de 35 mm.
- Terminales de conexión enchufables/móviles.
- Dimensiones: 88 x 72 x 59 mm
- Temperatura de operación del módulo: -35 a 50 °C
- Led indicación de energía / conexión.
- Ejemplos disponibles para PLC marca Slicetex Electronics.
- Compatible con la mayoría de PLC, HMI y sistemas con soporte ModBus RTU por RS-485
- Origen del producto: China.

2.1 Modelos Disponibles

Tabla 1: Modelos Disponibles para Ordenar

Modelo Numero de Parte (P/N)	Descripción
PD3060-PT100	Lector sensor PT100 de 6 canales, conexión ModBus RTU por RS-485.

2.2 Requerimientos

Para utilizar este módulo requiere un PLC o dispositivo con soporte RS-485 y ModBus RTU.

Para los PLC de Slicetex Electronics, además debe actualizar el PLC al último firmware disponible y actualizar el entorno de programación **StxLadder** a la última versión. De lo contrario los ejemplos descriptos en este documento pueden no funcionar correctamente.

3. Conexionado

3.1 Localización de Terminales, Controles e Indicadores

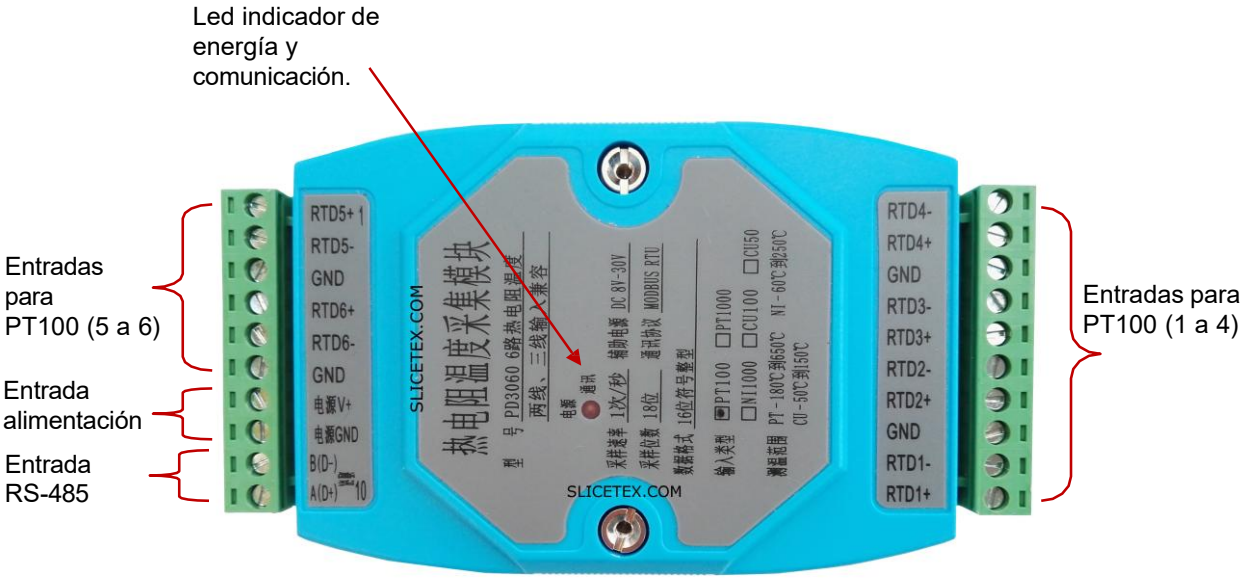


Figura 1: Localización de terminales e indicadores

Figura 2: Vista Lateral

Nota: Led del dispositivo parpadea cuando recibe una petición ModBus.

3.1 Descripción de Terminales

Tabla 2: Descripción de terminales en borneras

Símbolo	Tipo	Terminal Número	Descripción
V+	E	7	Entrada de tensión CC (corriente continua) para alimentación, terminal positivo. Rango 8 a 30 VCC. Ver Figura 1.
馈送 / GND	E	8	Retorno, tierra o masa de alimentación. Terminal negativo. Ver Figura 1. No confundir con GND para sensores PT100. También utilizado para masa de comunicación.
A (D+)	E/S	9	Terminal D+ (no inversor) para conexión RS-485.
B (D-)	E/S	10	Terminal D- (inversor) para conexión RS-485.
RTD1+	S	11	Conexión a PT100 canal 1. Alimenta sensor, 0.5 mA.
RTD1-	E	12	Conexión a PT100 canal 1. Retorno de sensor.
RTD2+	S	14	Conexión a PT100 canal 2. Alimenta sensor, 0.5 mA.
RTD2-	E	15	Conexión a PT100 canal 2. Retorno de sensor.
RTD3+	S	16	Conexión a PT100 canal 3. Alimenta sensor, 0.5 mA.
RTD3-	E	17	Conexión a PT100 canal 3. Retorno de sensor.
RTD4+	S	19	Conexión a PT100 canal 4. Alimenta sensor, 0.5 mA.
RTD4-	E	20	Conexión a PT100 canal 4. Retorno de sensor.
RTD5+	S	1	Conexión a PT100 canal 5. Alimenta sensor, 0.5 mA.
RTD5-	E	2	Conexión a PT100 canal 5. Retorno de sensor.
RTD6+	S	4	Conexión a PT100 canal 6. Alimenta sensor, 0.5 mA.
RTD6-	E	5	Conexión a PT100 canal 6. Retorno de sensor.
GND	E	3, 6, 13, 18	Punto común de sensores PT100. Para sensores de 3 cables, conectar tercer cable del sensor a un GND disponible. Para sensores de 2 cables, conectar también el terminal RTD(-) del sensor en cuestión a un GND disponible. No confundir con GND para alimentación. Ver Figura 1.

3.2 Conexión Típica al PLC o Dispositivo

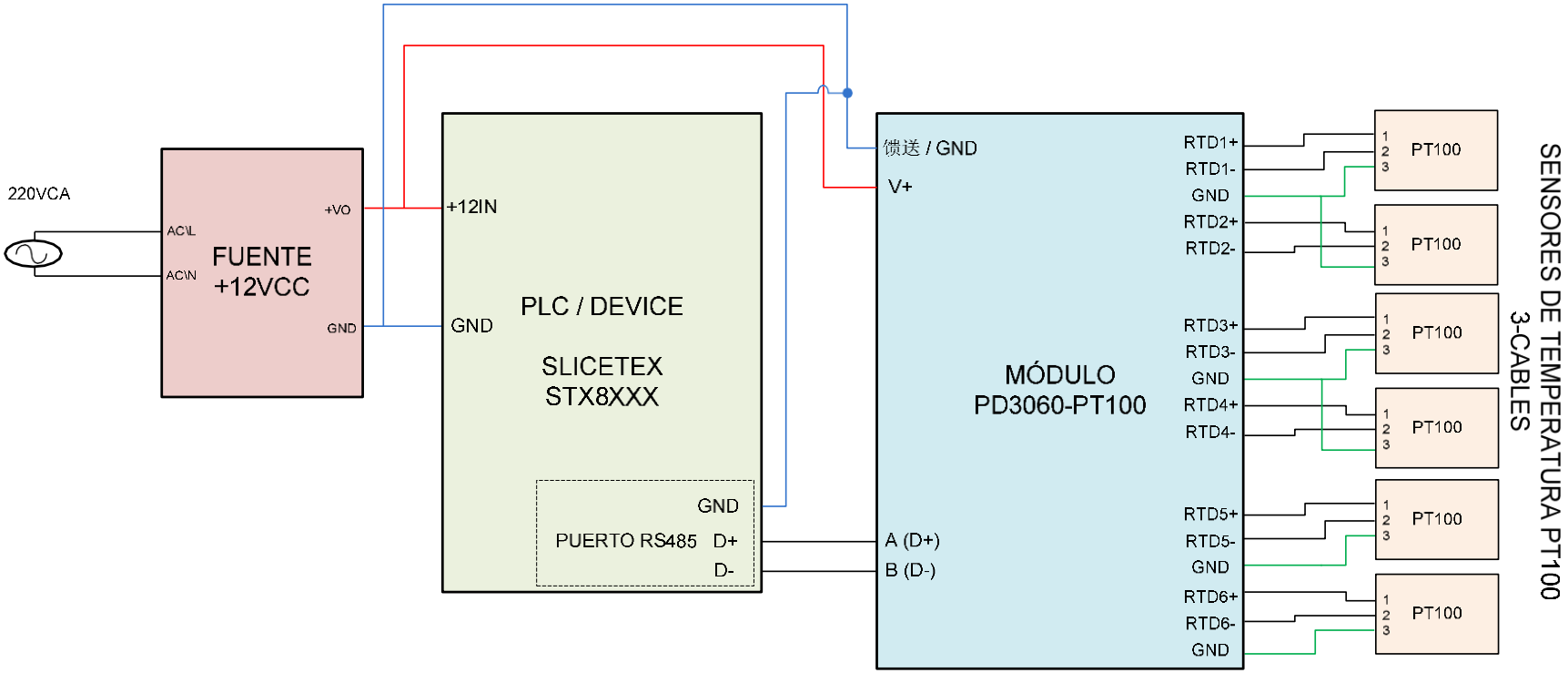


Figura 3: Ejemplo de conexión típica con PLC y PT100 de 3 cables.

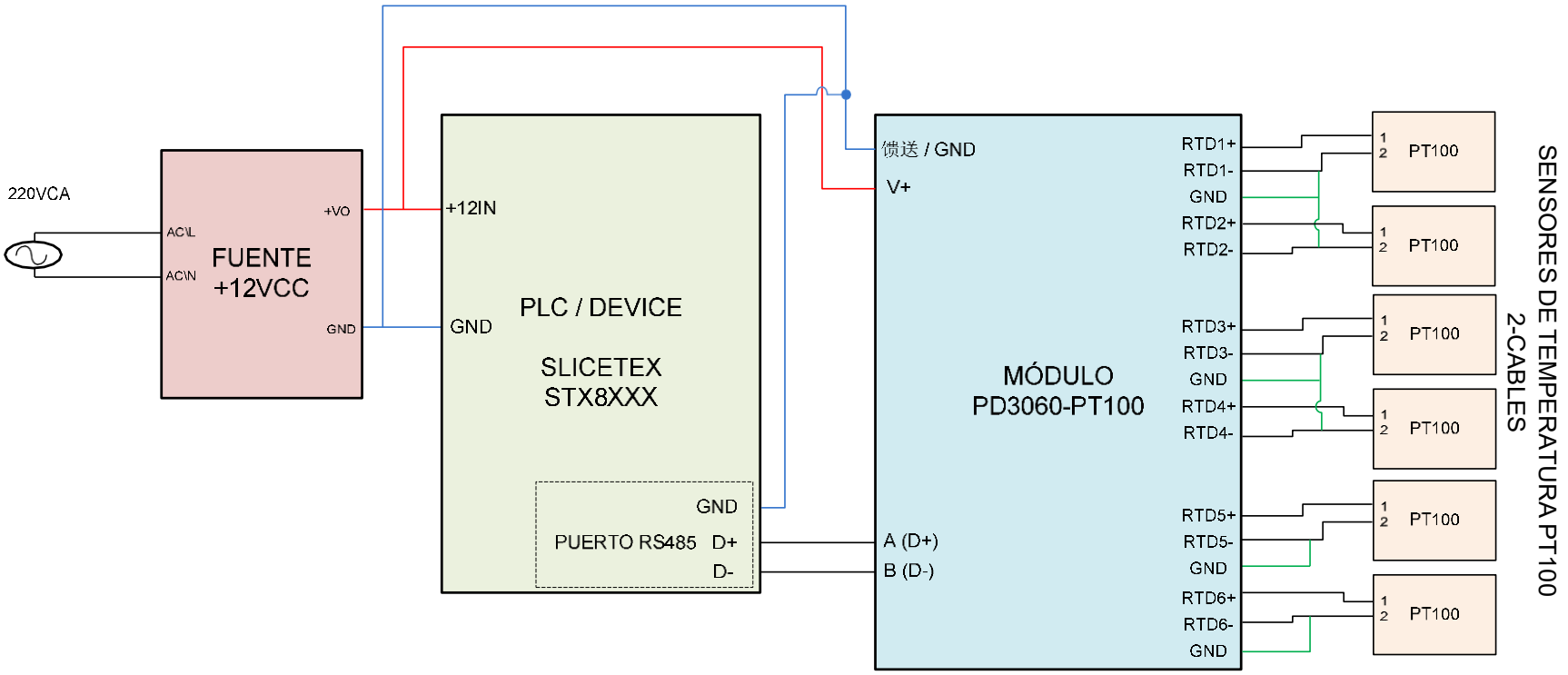


Figura 4: Ejemplo de conexión típica con PLC y PT100 de 2 cables.

En la
Figura 3 y
Figura 4 se aprecia un diagrama de la conexión de 6 sensores PT100 al módulo PD3060-PT100, el cual a su vez se conecta por RS-485 al PLC. Los cables del PT100 deben ser identificados antes de conexión para correcta lectura. No genera ningún daño su mal conexionado, solo obtendrá una lectura errónea fuera de escala. Recomendamos comenzar probando un sensor y luego conectar el resto.

El cable RS-485 puede ser armado por cuenta propia, solo hay que respetar los terminales de conexión, que coincidan del lado PLC y lado módulo. Algunos modelos de PLC pueden requerir selección o configuración extra de hardware para RS-485, consulte hoja de datos del dispositivo particular. Tenga en cuenta conectar el cable de masa para la comunicación, es decir, se utilizan tres cables para RS-485, dos de señales y uno de masa.

Si el cableado es largo, o la velocidad alta, se recomienda utilizar un resistor de terminación de 120 Ohms / 0.5Watt al principio de la conexión RS485 (lado PLC) y al final del último nodo (en este caso el módulo). Pero si hay varios módulos o dispositivos en el bus RS485, colocar el resistor en el último nodo. La conexión con varios dispositivos debe ser en cadena (Daisy-chain), es decir, del PLC conectar al primer módulo, del primer módulo al segundo, del segundo al tercero, y así. No conectar en estrella.

La alimentación del sistema es de acuerdo con los requerimientos, en este caso se utilizó una fuente de +12 a +24 VCC para alimentar PLC y módulo, pero puede ser otra de diferente valor, siempre en cuando no pase los limites erétricos de ambos equipos.

3.3 El sensor PT100

Un PT100 es un dispositivo termo-resistivo (RTD) cuyo valor a 0 °C es 100 ohms. Normalmente están fabricados en platino. Al aumentar la temperatura, aumenta su resistencia. Dicho incremento no es lineal, pero si creciente, por lo que mediante tablas es posible encontrar su temperatura exacta, como se ilustra en la siguiente figura.

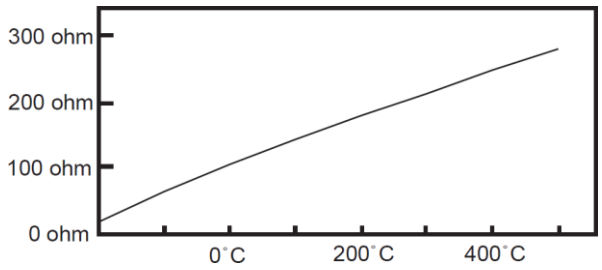


Figura 5: Curva resistiva del PT100

El módulo PD3060-PT100 automáticamente corrige la no-linealidad del sensor y nos entrega la temperatura en grados Celsius sin necesidad de utilizar tablas de corrección extras.

Normalmente las PT100 son más estable que los termistores NTC/PTC, pero más costosos. Se consiguen generalmente en formato industrial encapsuladas dentro de un tubo de acero inoxidable u otro material (vaina). En un extremo está el elemento sensible (alambre de platino) y en el otro está el terminal eléctrico (cables).

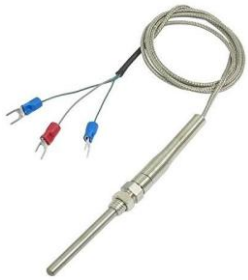


Figura 6: PT100 industrial de 3 terminales

Las PT100 pueden entregar fácilmente una exactitud en la medición de 0.1 °C, con la ventaja que las PT100 no se descomponen gradualmente con el tiempo entregando lecturas erróneas.

Si se daña, se abre el alambre resistivo y el lector normalmente entrega una lectura indicativa de error. En el caso del módulo PD3060-PT100 entrega la lectura de -100.00 °C.

Además la PT100 puede ser colocada a cierta distancia del medidor sin mayor problema (hasta 30 metros) utilizando cable de cobre convencional para hacer la extensión. Se recomienda utilizar cables gruesos para disminuir su resistencia. Sin embargo, tenga en cuenta que esto puede acarrear en lecturas menos exactas. Como regla general, mientras menos extensión, mejor.

El módulo PD3060-PT100 soporta dos tipos de conexión de PT100, las de dos-cables y las de tres-cables.

No se debe montar en general un PT100 en lugares sometidos a mucha vibración porque pueden fracturarse.

3.3.1 PT100 de dos y tres cables

El PT100 de dos-cables es el más sencillo (pero **menos recomendado**), en este caso un extremo del cable se conecta al terminal **RTD+** del módulo y el otro extremo a los terminales **RTD-** y **GND** del módulo, ver
Figura 4. Este tipo de conexión suele introducir errores en la medición debido a que a la resistencia del alambre PT100, se suma la resistencia del cable de conexión (algunas aplicaciones pueden tolerarlo, más aún si los cables son cortos).

El PT100 de tres-cables (el más común) resuelve bastante bien el problema del error generado por los cables, y se considera mejor opción para mediciones que requieran mayor exactitud.

El único requisito es que los tres cables tengan la misma resistencia eléctrica, ya que el sistema de medición se basa normalmente en el puente de Wheatstone.

El conexionado al módulo de PT100 de tres-cables puede verse en la
Figura 3.

Finalmente existen los PT100 de 4-cables, más exactos, los 4 cables pueden ser diferente (distinta resistencia eléctrica), pero son más costosos y no están soportados por el módulo PD3060-PT100.

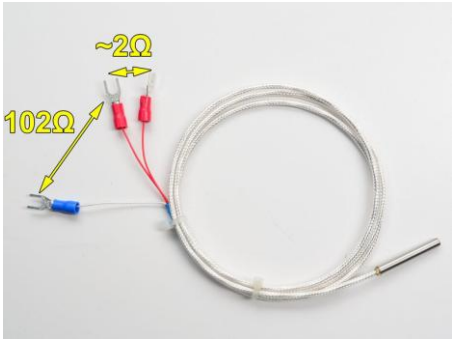


Figura 7: Resistencia de PT100+Cable a 0 °C en sensores de 3-cables

4. Registros ModBus RTU

El módulo actúa como esclavo en una conexión ModBus RTU a través de RS-485, por lo que un PLC u otro dispositivo, debe actuar como master o cliente, para indagar mediante peticiones ModBus los valores de temperatura en los registros internos del módulo.

Por defecto, el módulo utiliza una velocidad de 9600 BPS, formato 8N1 (sin paridad), dirección "1" de esclavo ModBus RTU. Los valores de temperatura están multiplicados por 10.

Tabla 3: Registros ModBus código de función 3 (lectura holding) y función 6 (escritura holding)

Dirección Registro Decimal (hexadecimal) Nota[1]	Byte	Descripción	Parámetro	Acceso (Read / Write)
16 (0x10)	Byte menos significativo	Configuración de comunicación Valor inicial: 00	BIT[7:5] = reservado, sin uso. Formato: BIT[4:3] = 00 No parity BIT[4:3] = 01 Even parity BIT[4:3] = 10 Odd parity BIT[4:3] = 11 Odd parity Velocidad: BIT[2:0] = 000 (9600 BPS) BIT[2:0] = 001 (1200 BPS) BIT[2:0] = 010 (2400 BPS) BIT[2:0] = 011 (4800 BPS) BIT[2:0] = 100 (9600 BPS) BIT[2:0] = 101 (14400 BPS) BIT[2:0] = 110 (19200 BPS)	R/W
	Byte más significativo	Dirección ModBus Valor inicial: 01	Rango: 1 a 250 Dirección 0: Broadcast	R/W
17 (0x11)	Byte menos significativo	Reservado. Sin uso.		R/W
	Byte más significativo	Round display switching interval Valor inicial: 05	Rango: 0 a 5	R/W

Tabla 4: Registros ModBus código de función 3 (lectura holding)

Dirección Registro Decimal (hexadecimal) Nota[1]	Descripción	Parámetro	Acceso (Read / Write)
32 (0x20)	Temperatura de canal 1	Formato: entero 16-bits con signo. Rango: -1000 a 6500 (multiplica por 10) Valor -1000 = Sin sensor conectado.	R
33 (0x21)	Temperatura de canal 2	Formato: entero 16-bits con signo. Rango: -1000 a 6500 (multiplica por 10) Valor -1000 = Sin sensor conectado.	R
34 (0x22)	Temperatura de canal 3	Formato: entero 16-bits con signo. Rango: -1000 a 6500 (multiplica por 10) Valor -1000 = Sin sensor conectado.	R
35 (0x23)	Temperatura de canal 4	Formato: entero 16-bits con signo. Rango: -1000 a 6500 (multiplica por 10) Valor -1000 = Sin sensor conectado.	R
36 (0x24)	Temperatura de canal 5	Formato: entero 16-bits con signo. Rango: -1000 a 6500 (multiplica por 10) Valor -1000 = Sin sensor conectado.	R
37 (0x25)	Temperatura de canal 6	Formato: entero 16-bits con signo. Rango: -1000 a 6500 (multiplica por 10) Valor -1000 = Sin sensor conectado.	R

Notas:

1. La dirección de los registros en las tablas es absoluta. En algunos dispositivos o PLC, puede tener que sumar 40001 a estos valores. Por ejemplo, el registro 32 le quedaría en 32+40001 = 40033. En otros puede que tenga que sumar una unidad, es decir el registro 32 le quedaría en 33. Recomendamos experimentar leyendo un valor por defecto y deducir el funcionamiento correcto. Esto se debe a que el protocolo ModBus no especifica una estándar claro sobre el direccionamiento y cada fabricante puede adoptar diferentes convenciones.

2. Las configuraciones realizadas no se pierden al quitar la energía.

5. Ejemplo de Programación

A continuación brindamos ejemplos para conectarse al módulo PD3060-PT100 utilizando un PLC de la marca Slicetex Electronics.

El código completo puede descargarlo desde nuestra página web, visitando la página específica del módulo PD3060-PT100:

www.slicetex.com/modules/temperatura/pd3060-pt100

Para otros dispositivos o PLC, podemos asesorarlo en conceptos generales sobre la comunicación por ModBus RTU (RS485) pero deberá consultar con la documentación del fabricante del equipo para realizar el cableado, la programación y los ensayos de prueba. Recuerde que el protocolo ModBus es universal, por lo que podrá encontrar información disponible en internet.

5.1 Lenguaje Pawn

En lenguaje Pawn hemos preparado el proyecto **PT100_PD3060_Pawn1.zip** que puede descargar para luego abrir con el entorno de programación StxLadder, compilar y transferir al PLC.

En el ejemplo en cuestión, el PLC lee 6 canales del módulo **PD3060-PT100** y muestra la temperatura de los sensores PT100 conectados al módulo en la pantalla del software VirtualHMI (software disponible para descargar desde nuestra página) cada 0.5 segundos aproximadamente, como se muestra a continuación:

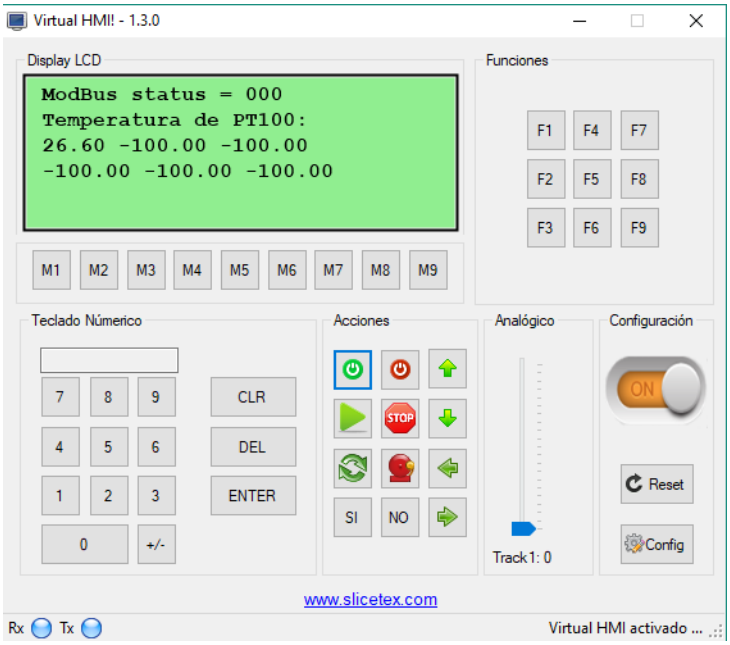


Figura 8: Temperatura de PT100 en VirtualHMI (Pawn)

En la Figura 8 se puede observar cómo se muestran 6 valores de temperatura. El valor "**26.6**" es la temperatura en grados Celsius del canal 1 del módulo que tiene conectado un sensor PT100. El resto de los canales permanecen desconectados, por lo que la lectura es "**-100.00**" (fuera de escala).

También en la primera línea, muestra el estado de la transacción ModBus, que en este caso es "**ModBus status = 000**", es decir, ningún error de conexión. Puede usar este ejemplo para probar el módulo.

Para comprender el programa, descargue el proyecto y abrirlo con StxLadder. Posteriormente abra el archivo **PD3060_PT100.p** del proyecto y busque la siguiente función:

```
Float: PD3060_PT100_ConvertToTemperature(Register)
{
   // Convertir registro, valor entero 16-bits de registro a 32-bits (extender
   // signo si es negativo) y luego escalar valor dividiendo por 10.

   return Int16ToInt(Register) / 10.0
}
```

La función **PD3060_PT100_ConvertToTemperature()** es muy simple, toma el valor del argumento "**Register**", es decir un registro de 16-bits con signo proveniente del módulo, lo convierte a 32-bits con signo y luego lo divide por 10. La división es necesaria porque el módulo entrega el valor de temperatura con un factor de 10.

Luego se retorna el valor como una variable con punto flotante que permite punto decimal. Este valor es la temperatura obtenida.

El próximo paso consiste en realizar la conexión ModBus RTU como maestro o cliente con el módulo PD3060-PT100.

La nota de aplicación **AN026** disponible en nuestro sitio web explica en detalle todas las funciones ModBus RTU como cliente en lenguaje Pawn para referencia:

www.slicetex.com/docs/an/an026

Para el ejemplo, abra el archivo **PlcMain.p** en el proyecto y busque el código:

```
// Inicializar Cliente ModBus RTU.
// Velocidad 9600 bps, formato 8N1, timeout 10 segundos, interface RS485.

if((FuncStat=MbRtuClInit(9600, SERIAL_8N1, 10, MB_RTU_INTERFACE_RS485)) < 0)
{
      nLcdPrintf(0,1, LCD_CLRALL, "ModBus init err: %d", FuncStat)
}
```

El código anterior utiliza la función **MbRtuInit()** e inicializa el puerto RS-485 del PLC para una conexión con velocidad 9600 bps y formato 8N1 (sin paridad). Tal como lo requiere el módulo PD3060-PT100 según configuración por defecto de fábrica (ver **Tabla 3**).

A continuación, el programa activa el evento "**OnMbRtuClientRx**" con la siguiente instrucción:

```
MbRtuClSetRxEvent()
```

Recordar que un evento "interrumpe" la lógica principal y llama a la función correspondiente para "manejarlo". La respuesta desde el servidor/esclavo (sensor) es leída en el evento "**@OnMbRtuClientRx()**", que se llamará cuando finalice la transacción ModBus RTU. Ver archivo **OnMbRtuClient.p**.

Continuando en archivo **PlcMain.p,** dentro del “loop” principal **for()** del programa, busque el código:

```
if(WaitTemperature == 0)
 {
        WaitTemperature = 1

        // Enviar petición para leer 6 registros
        // "Holding Registers" en dirección 0x20.

        if(MbRtuClSendReadHoldingReg(SLAVE_ADDR, 0x20, 6) < 0)
        {
               nLcdPrintf(0,1,LCD_CLRALL, "ModBus send error...");

               WaitTemperature = 0
        }
 }
```

En el código anterior sí la variable **WaitTemperature** es 0, procedemos a realizar una transacción ModBus utilizando la función **MbRtuClSendReadHoldingReg()**, que permite leer registros del tipo “Holding Registers” utilizando la función ModBus código 3 (según estándar). Esta función realiza una transmisión por RS-485 al módulo sensor.

Los argumentos de la función son **SLAVE_ADDR** (constante que vale 1 y es la dirección del módulo en la red ModBus), valor **0x20** que representa la dirección en hexadecimal del registro de temperatura del canal 1 (ver **Tabla 4**) y el valor **6** (que indica que se solicita leer 6 registros consecutivos a partir de dirección 0x20, es decir en una sola transacción leeremos, 0x20, 0x21, 0x22, 0x23, 0x24 y 0x25, por lo tanto todos los canales de temperatura disponibles).

Si no existen errores al llamar la función, se hace **WaitTemperature = 1**, indicando que estamos esperando una respuesta ModBus del módulo de temperatura, además, esto impide iniciar nuevamente una transacción sin completar la anterior dentro del loop **for()**.

Finalmente dentro del loop **for()**, se conmuta el se realiza una pausa por 500 mS. Aquí puede hacer alguna otra actividad, pero en este ejemplo solo pausamos, es decir no hace nada el PLC mientras espera la repuesta o fin de transacción ModBus.

Cuando finalice la transacción ModBus, se "interrumpe" la lógica principal y se llama a la función del evento "**@OnMbRtuClientRx()**", en archivo **OnMbRtuClient.p** para procesar la repuesta del módulo de temperatura.

La primera instrucción en **@OnMbRtuClientRx()** es la siguiente:

```
MbStat = MbRtuClGetLibStatus()
```

Esta función permite indagar el estado de la librería ModBus en el PLC. Retorna un código de estado indicando si la transacción fue exitosa, tuvo errores o está pendiente de respuesta.

El código retornado lo guardamos en variable **MbStat**, si es igual a la constante **MB_RTU_CL_STAT_OK (**o valor 0), quiere decir que recibimos una respuesta exitosa.

Esto nos permite luego comprobar si llegaron los datos con el siguiente código con la sentencia “if”:

```
// Comprobar si llegó respuesta del Servidor o esclavo ModBus

if(MbStat == MB_RTU_CL_STAT_OK && WaitTemperature== 1)
{
   // Si, leer registros recibidos (utilizamos un offset de 1 byte para saltear
   // el primer byte de respuesta ModBus, asi leer directamente registros holding,
   // ver especificación ModBus de codigos de funciones).

   if((FuncStat=MbRtuClGetRxReg(RxData, 0, 6, 1)) == 0)
   {
      // Convertir a temperatura los 6 registros obtenidos.
      PT100_Temperature[0] = PD3060_PT100_ConvertToTemperature(RxData[0])
      PT100_Temperature[1] = PD3060_PT100_ConvertToTemperature(RxData[1])
      PT100_Temperature[2] = PD3060_PT100_ConvertToTemperature(RxData[2])
      PT100_Temperature[3] = PD3060_PT100_ConvertToTemperature(RxData[3])
      PT100_Temperature[4] = PD3060_PT100_ConvertToTemperature(RxData[4])
      PT100_Temperature[5]  =  PD3060_PT100_ConvertToTemperature(RxData[5])

      // Mostrar los 6 valores de temperatura en LCD (grados Celsius).
      nLcdPrintf(0,1,LCD_CLRLINE, "Temperatura de PT100:")
      nLcdPrintf(0,2,LCD_CLRLINE, "%f %f %f",
            PT100_Temperature[0],  PT100_Temperature[1], PT100_Temperature[2])
      nLcdPrintf(0,3,LCD_CLRLINE, "%f %f %f",
             PT100_Temperature[3],  PT100_Temperature[4],  PT100_Temperature[5])
   }
   else
   {
      // Mostrar mensaje de error.
      nLcdPrintf(0,1,LCD_CLRLINE, "ModBus data err: %d", FuncStat)
   }
}
```

Como vemos, si **MbStat = MB_RTU_CL_STAT_OK** (transacción ModBus existosa) y **WaitTemperature = 1** (estamos esperando una respuesta ModBus del sensor de temperatura) se entra al “if” y comprobamos con otro “if” a la función **MbRtuClGetRxReg()**:

```
if((FuncStat=MbRtuClGetRxReg(RxData, 0, 6, 1)) == 0)
```

La cual obtiene de buffer interno de recepción del PLC los datos recibidos por ModBus en la última respuesta. Donde usamos como argumentos, “**RxData”** (array de 6 enteros donde guardaremos los registros leídos del Módulo), “**0”** indica que se guardarán los registros a partir del elemento 0 de **RxData**, “**6”** es la cantidad de registros a copiar y “**1**” es un offset necesario para poder leer los registros posterior a una transacción ModBus de lectura de “Holding Registers”.

Una vez obtenidos los registros de temperatura del módulo, debemos convertirlos a temperatura, esto lo hacemos con el siguiente código:

```
      // Convertir a temperatura los 6 registros obtenidos.
      PT100_Temperature[0] = PD3060_PT100_ConvertToTemperature(RxData[0])
      PT100_Temperature[1] = PD3060_PT100_ConvertToTemperature(RxData[1])
      PT100_Temperature[2] = PD3060_PT100_ConvertToTemperature(RxData[2])
      PT100_Temperature[3] = PD3060_PT100_ConvertToTemperature(RxData[3])
      PT100_Temperature[4] = PD3060_PT100_ConvertToTemperature(RxData[4])
      PT100_Temperature[5]  =  PD3060_PT100_ConvertToTemperature(RxData[5])
```

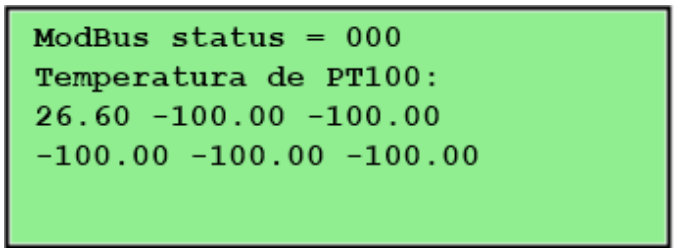
La función **PD3060_PT100_ConvertToTemperature()** la vimos con anterioridad. Tomamos los valores crudos de los registros recibidos del módulo en **RxData[ ]** y los convertimos a temperatura guardándolos en el array **PT100_Temperature[ ]** con formato punto flotante y en escala adecuada.

Luego mostramos los valores de temperatura en Virtual-HMI con el siguiente código:

```
      // Mostrar los 6 valores de temperatura en LCD (grados Celsius).
      nLcdPrintf(0,1,LCD_CLRLINE, "Temperatura de PT100:")
      nLcdPrintf(0,2,LCD_CLRLINE, "%f %f %f",
            PT100_Temperature[0], PT100_Temperature[1], PT100_Temperature[2])
      nLcdPrintf(0,3,LCD_CLRLINE, "%f %f %f",
             PT100_Temperature[3], PT100_Temperature[4], PT100_Temperature[5])
```

Donde cada valor de temperatura se imprime en decimal usando el comodín "%f" de la función **nLcdPrintf()** ver documentación de **VirtualHMI**.

Esto resulta en una impresión en pantalla igual a:



El valor "**26.6**" es la temperatura en grados Celsius del canal 1 del módulo que tiene conectado un sensor PT100. El resto de los canales permanecen desconectados, por lo que la lectura es "**-100.00**" (fuera de escala). Estos valores se imprimen leyendo la variable **PT100_Temperature[ ]**.

Adicionalmente, más abajo en el código comprobamos cuando el valor de **MbStat** es negativo, lo que indica un código de error en la transacción ModBus. Esto solo se hace a fines didácticos, para conocer el código de error y depurar un programa o falla, pero no es necesario para un código final.

```
// Comprobar errores (código negativo).
if(MbStat < 0)
{
      if(MbStat == MB_RTU_CL_STAT_ERR_EXCEP)
      {
         nLcdPrintf(0,1,LCD_CLRLINE, "ModBus exception: %03d", MbRtuClGetExceptionCode())
      }

      if(MbStat == MB_RTU_CL_STAT_ERR_FUNC_DIF)
      {
         nLcdPrintf(0,1,LCD_CLRLINE, "ModBUs func. dif.");
      }

      if(MbStat == MB_RTU_CL_STAT_ERR_TO)
      {
         nLcdPrintf(0,1,LCD_CLRLINE, "ModBus timeout!");
      }
}
```

Dicho error también se imprimirá en la pantalla de VirtualHMI.

Finalmente, al final de la función del evento, hacemos **WaitTemperature = 0** (ya que no esperamos más una respuesta ModBus del sensor):

```
//
// Limpiar indicador de espera de repuesta de temperatura.
//

WaitTemperature = 0
```

5.1.1 Ejemplo sin Eventos

También puede descargar el proyecto **PT100_PD3060_Pawn2.zip**, que funcionalmente es un ejemplo idéntico a **PT100_PD3060_Pawn1.zip**, solo que no utiliza eventos. Es decir, toda la transacción se realiza dentro del loop **for()** dentro de **PlcMain()**, indagando constantemente (polling) el estado de la transacción ModBus hasta determinar el fin de la misma.

El uso de ambos ejemplos, queda a criterio del programador, de acuerdo a sus necesidades, uno u otro puede ser de utilidad.

5.2 Lenguaje Ladder

En lenguaje Ladder hemos preparado el proyecto **PT100_PD3060_Ladder1.zip** que puede descargar para luego abrir con el entorno de programación StxLadder, compilar y transferir al PLC.

En el ejemplo en cuestión, el PLC lee 6 canales del módulo **PD3060-PT100** y muestra la temperatura de los sensores PT100 conectados al módulo en la pantalla del software VirtualHMI (software disponible para descargar desde nuestra página) cada 0.5 segundos aproximadamente, como se muestra a continuación:

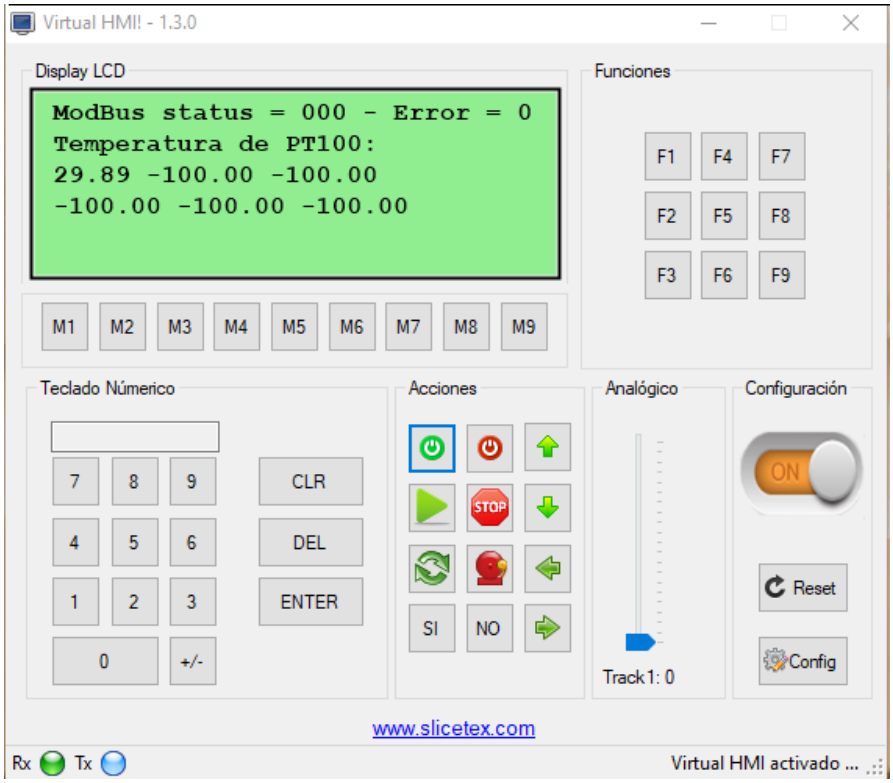
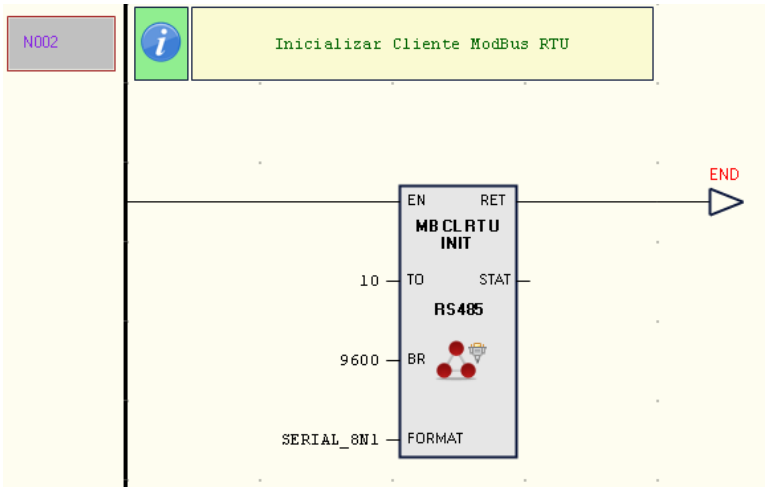


Figura 9: Temperatura de PT100 en VirtualHMI (Ladder)

En la Figura 8 se puede observar cómo se muestran 6 valores de temperatura. El valor "**26.6**" es la temperatura en grados Celsius del canal 1 del módulo que tiene conectado un sensor PT100. El resto de los canales permanecen desconectados, por lo que la lectura es "**-100.00**" (fuera de escala).

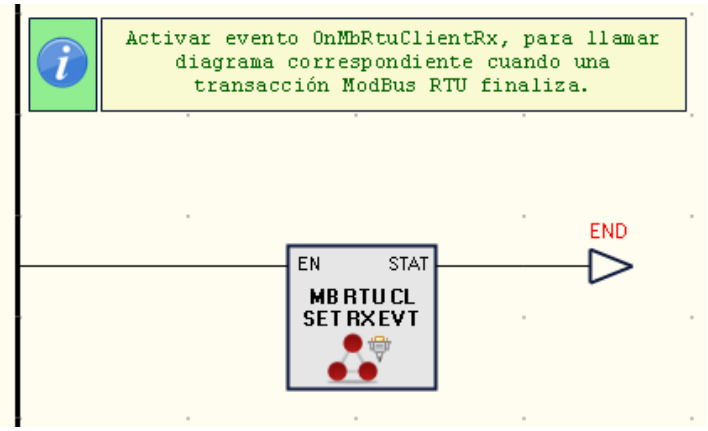
También en la primera línea, muestra el estado de la transacción ModBus, que en este caso es "**ModBus status = 000**", es decir, ningún error de conexión. Puede usar este ejemplo para probar el módulo.

Para comprender el programa, descargue el proyecto y abrirlo con StxLadder. Posteriormente abra el diagrama **Inicio.sld** del proyecto y busque el siguiente componente:



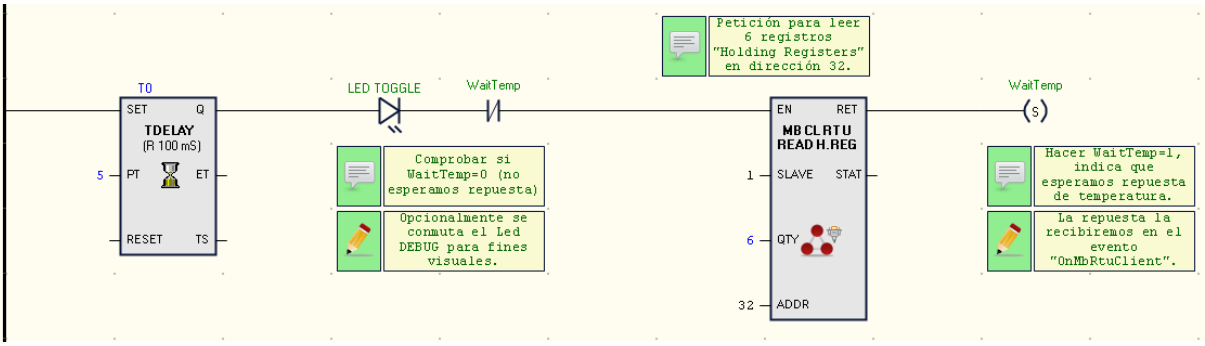
El código anterior utiliza el componente **MbRtuInit** e inicializa el puerto RS-485 del PLC para una conexión con velocidad 9600 bps y formato 8N1 (sin paridad). Tal como lo requiere el módulo PD3060-PT100 según configuración por defecto de fábrica (ver **Tabla 3**).

A continuación, el programa activa el evento "**OnMbRtuClientRx**" con el siguiente componente:



Recordar que un evento "interrumpe" la lógica principal y llama al diagrama correspondiente para "manejarlo". La respuesta desde el servidor/esclavo (sensor) es leída en el diagrama evento "**OnMbRtuClientRx.sld**", que se llamará cuando finalice la transacción ModBus RTU.

A continuación abra el diagrama **Principal.sld**, ver network N001:



En el código anterior hay un temporizador tipo "**TP**", que cada 500 mS ejecutará la siguiente lógica:

Sí la variable **WaitTemp** es 0, procedemos a realizar una transacción ModBus utilizando el componente **MbRtuClSendReadHoldingReg**, que permite leer registros del tipo "Holding Registers" utilizando la función ModBus código 3 (según estándar). Este componente realiza una transmisión por RS-485 al módulo sensor.

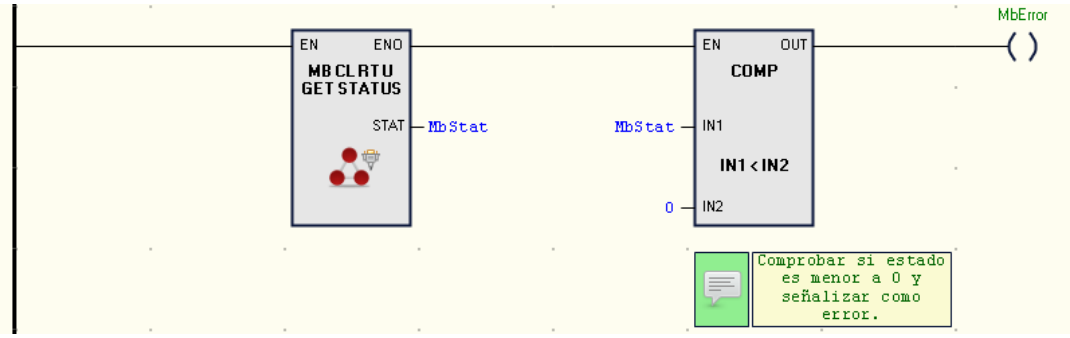
Los valores en los puertos del componente son: **SLAVE=1** (la dirección del módulo en la red ModBus), **ADDR** = **32** que representa la dirección en decimal del registro de temperatura del canal 1 (ver **Tabla 4**) y el valor **QTY** = **6** (que indica que se solicita leer 6 registros consecutivos a partir de dirección 32, es decir en una sola transacción leeremos 32, 33, 34, 35, 36 y 37, por lo tanto todos los canales de temperatura disponibles).

Si no existen errores al llamar al componente, se hace **WaitTemp = 1**, indicando que estamos esperando una respuesta ModBus del módulo de temperatura, además, esto impide iniciar nuevamente una transacción sin completar la anterior dentro del diagrama, cuando se llame nuevamente a estos componentes.

La nota de aplicación **AN026** disponible en nuestro sitio web explica en detalle todos componentes ModBus RTU como cliente en lenguaje Ladder para referencia: www.slicetex.com/docs/an/an026

Cuando finalice la transacción ModBus, se "interrumpe" la lógica principal y se llama al diagrama del evento "**OnMbRtuClientRx**", en archivo **OnMbRtuClient.sld** para procesar la repuesta del módulo de temperatura.

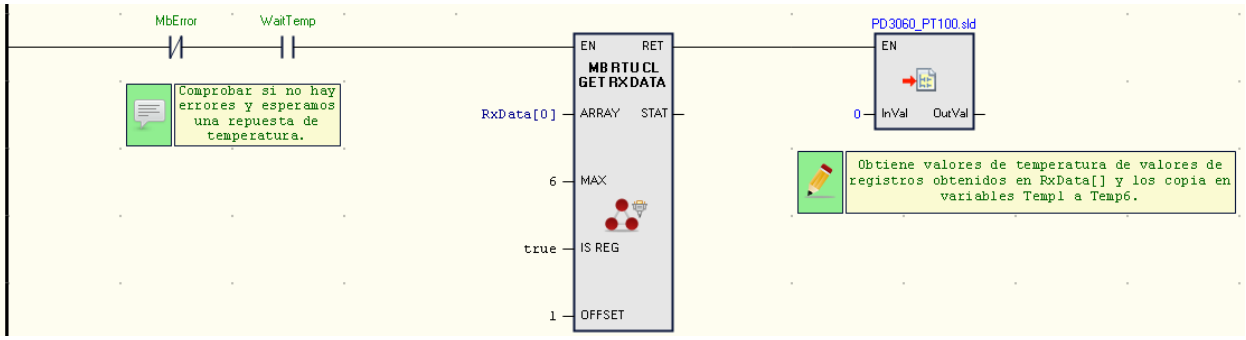
En la network N001 llamamos al componente **MbRtuClGetLibStatus**:



El componente **MbRtuClGetLibStatus** permite indagar el estado de la librería ModBus en el PLC. Retorna un código de estado indicando si la transacción fue exitosa, tuvo errores o está pendiente de respuesta.

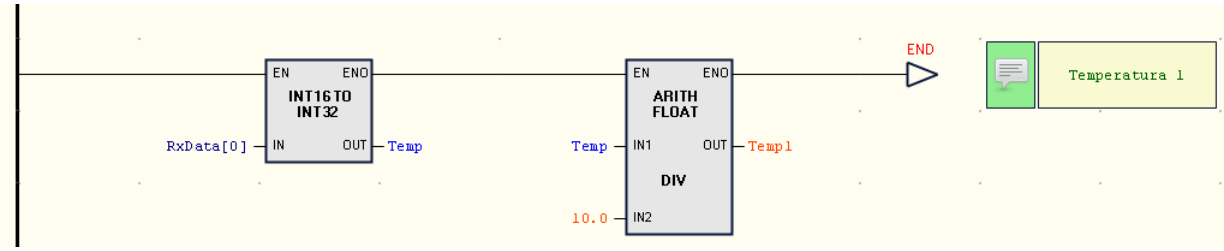
El código retornado lo guardamos en variable entera **MbStat**, y si es igual a 0, quiere decir que recibimos una respuesta exitosa. Por lo tanto luego de la comparación, colocamos en la variable **MbError** el valor 0 (sin error) o 1 (error), según corresponda.

A continuación, sí las variables tienen los valores **MbError** = 0 (sin errores) y **WaitTemp** = 1 (se espera una repuesta del módulo de temperatura), procedemos a leer los registros ModBus recibidos, como muestra el siguiente código en network N002:



El componente **MbRtuClGetRxData** obtiene de buffer interno de recepción del PLC los datos recibidos por ModBus en la última respuesta. Donde usamos como parámetros en su puertos, **ARRAY** = **RxData[0]** (array de 6 enteros donde guardaremos los registros leídos del Módulo), "**[0]"** indica que se guardarán los registros a partir del elemento 0 de **RxData[ ]**. **MAX** = **6** es la cantidad de registros a copiar y **OFFSET** = **1** es un offset necesario para poder leer los registros posterior a una transacción ModBus de lectura de "Holding Registers".

Finalmente, si no hay errores se llama al diagrama función **PD3060_PT100.sld** (ver archivo) donde procesaremos los valores de registros obtenidos del módulo sensor a través de ModBus:



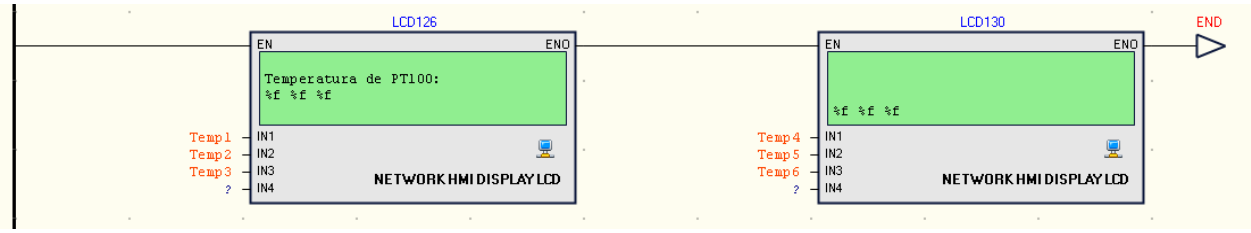
En el código anterior, se ejemplifica como se obtienen el valor de temperatura para el canal 1 o sensor 1 de PT100, pero para los 5 restantes canales el código es idéntico.

Se toma el valor del registro del módulo obtenido en variable **RxData[0]**, ya que es un array declarado globalmente**.** Primero se convierte de entero de 16-bits a entero de 32-bits (recordar que ModBus usa registros de 16-bits, por lo tanto, si hay un valor negativo, hay que extender el signo). Entonces, **RxData[0]** se convierte a variable entera **Temp** usando la conversión de 16-bits a 32-bits. Luego el valor

de **Temp** se divide por 10 y se copia su resultado a **Temp1** (variable Float, preservando decimales). La división es necesaria porque el módulo entrega el valor de temperatura con un factor de 10.

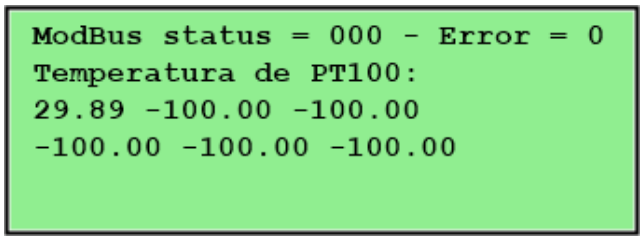
En variable **Temp1** ya tenemos la temperatura en grados Celsius del PT100 número 1, mismo procedimiento efectuamos para los restantes 5 canales, obteniendo temperatura en variables **Temp2**, **Temp3**, **Temp4**, **Temp5** y **Temp6**.

Luego mostramos los valores de temperatura en Virtual-HMI con el siguiente código:



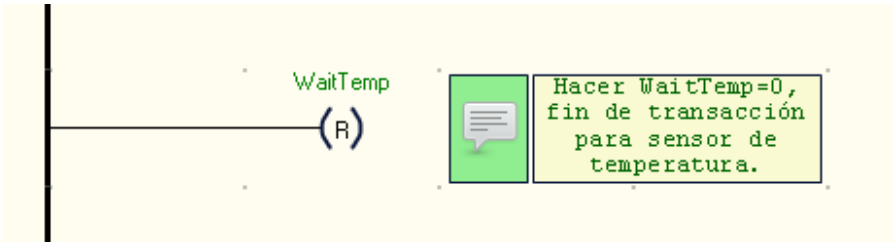
Donde cada valor de temperatura se imprime en decimal usando el comodín **"%f** del componente **NetHmiLcdPrintf** ver documentación de **VirtualHMI**.

Esto resulta en una impresión en pantalla igual a:



El valor "**29.89**" es la temperatura en grados Celsius del canal 1 del módulo que tiene conectado un sensor PT100. El resto de los canales permanecen desconectados, por lo que la lectura es "**-100.00**" (fuera de escala). Estos valores se imprimen leyendo las variables **Temp1**, **Temp2**, **Temp3**, **Temp4**, **Temp5** y **Temp6.**

Por último, volviendo al diagrama evento **OnMbRtuClientRx.sld**, se debe hacer la variable **WaitTemp=0** (ver network N003), para indicar fin de transacción o fin de espera de repuesta del módulo de temperatura:



Sí no hiciéramos **WaitTemp=0**, la lógica en diagrama **Principal.sld** no iniciaría una nueva transacción ModBus, ya que depende del valor de dicha variable.

6. Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **PT100**: Sensor termo-resistivo usado para medición de temperatura.
- **ModBus RTU**: Protocolo de comunicación que utiliza interfaz RS-232 o RS-485.
- **CA**: Corriente Alterna, o en ingles AC.
- **CC**: Corriente Continua, o en ingles DC.

7. Historial de Revisiones

Tabla 5: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
04 29/Sep/2022	1	1. Pequeñas actualizaciones en texto general en todo el documento. 2. Figura 3 y Figura 4 agregan conexionado de masa para RS-485. 3. Notas de direccionamiento de registros en sección Registros ModBus RTU.	Inicial
03 05/Abr/2018	2	1. Actualizado comportamiento de led del dispositivo. 2. Otros cambios menores	Inicial
02 24/Mar/2018	5	1. Se agrega ejemplo para Lenguaje Ladder. 2. Se cambia ejemplo Pawn para uso de eventos y nuevas funciones. 3. Se agrega conexionado para PT100 de dos cables. 4. Nuevas fotos del módulo. 5. Otros cambios y correcciones de texto.	Inicial
01 29/Dic/2017	1	1. Versión preliminar liberada.	Preliminar

8. Referencias

- **Referencia [1]:** "*PT100, su operación, instalación y tablas*", rev. A, Nota Técnica 004, ARIAN (Control & Instrumentación), www.arian.cl.

9. Información Legal

9.1 Aviso de exención de responsabilidad

General: La información de este documento se da en buena fe, y se considera precisa y confiable. Sin embargo, Slicetex Electronics no da ninguna representación ni garantía, expresa o implícita, en cuanto a la exactitud o integridad de dicha información y no tendrá ninguna responsabilidad por las consecuencias del uso de la información proporcionada.

El derecho a realizar cambios: Slicetex Electronics se reserva el derecho de hacer cambios en la información publicada en este documento, incluyendo, especificaciones y descripciones de los productos, en cualquier momento y sin previo aviso. Este documento anula y sustituye toda la información proporcionada con anterioridad a la publicación de este documento.

Idoneidad para el uso: Los productos de Slicetex Electronics no están diseñados, autorizados o garantizados para su uso en aeronaves, área médica, entorno militar, entorno espacial o equipo de apoyo de vida, ni en las aplicaciones donde el fallo o mal funcionamiento de un producto de Slicetex Electronics pueda resultar en lesiones personales, muerte o daños materiales o ambientales graves. Slicetex Electronics no acepta ninguna responsabilidad por la inclusión y / o el uso de productos de Slicetex Electronics en tales equipos o aplicaciones (mencionados con anterioridad) y por lo tanto dicha inclusión y / o uso es exclusiva responsabilidad del cliente.

Aplicaciones: Las aplicaciones que aquí se describen o por cualquiera de estos productos son para fines ilustrativos. Slicetex Electronics no ofrece representación o garantía de que dichas aplicaciones serán adecuadas para el uso especificado, sin haber realizado más pruebas o modificaciones.

Los valores límites o máximos: Estrés por encima de uno o más valores límites (como se define en los valores absolutos máximos de la norma IEC 60134) puede causar daño permanente al dispositivo. Los valores límite son calificaciones de estrés solamente y el funcionamiento del dispositivo en esta o cualquier otra condición por encima de las indicadas en las secciones de Características de este documento, no está previsto ni garantizado. La exposición a los valores limitantes por períodos prolongados puede afectar la fiabilidad del dispositivo.

Documento: Prohibida la modificación de este documento en cualquier medio electrónico o impreso, sin autorización previa de Slicetex Electronics por escrito.

10. Información de Contacto

Para mayor información, visítenos en www.slicetex.com

Consultas e información general, envíe un mail a: info@slicetex.com

Foro de soporte técnico: foro.slicetex.com

11. Contenido

11.1 Índice general

11.2 Índice de tablas

11.3 Índice de figuras