Modbus RTU Driver Library for Coolmay FX3G PLC

Terminology

- Register - an area of the memory in Modbus address stack.

Requirements

This library require other libraries to be installed:

- Utils.sul
- TimeControl50.sul

> **!!! Important**
>
> - This library required TCO Ticker to bet set up in Time Control 50 library. Please setup it first.
> - Works only with GXW2 v1.91. Please install updates from `coolmay/soft` folder.

Chanelog

v1.1 [29.09.24]

- change `R` registers to `D` because some panels do not support `R` over Modbus protocol such as OP320.

v1.0 [22.09.24]

- Initial library with all features planned.

Description

This library helps you setup PLC as Modbus Slave or Modbus Master (on ports 2 and 3) to read and write data from Modbus devices over RTU. It requires minimum efforts to setup a master.

Coolmay PLCs have 2 RS485 ports. One is terminal connected and that one is port 2, and another is on DB9 connector, that is one port 3.

MB_PORT_SETTINGS

This function create a variable with correct bits for port initialization as master or slave.

Variable	Scope	Type	Description
`Parity`	INPUT	(Word[Signed])	Use constants `MB_PARITY_NONE` or `MB_PARITY_ODD` or `MB_PARITY_EVEN`
`StopBit`	INPUT	(Word[Signed])	Use constants `MB_STOPBIT_1` or `MB_STOPBIT_2`
`Baudrate`	INPUT	(Word[Signed])	Use constants `MB_BPS_600` or `MB_BPS_1200` or `MB_BPS_2400` or `MB_BPS_4800` or `MB_BPS_9600` or `MB_BPS_19200` or `MB_BPS_38400` or `MB_BPS_57600` or `MB_BPS_115200`

Declare local Unsigned Double Word variable in your program such as `PortSettings`. And the call function.

```
PortSettings := MB_PORT_SETTINGS(MB_PARITY_NONE, MB_STOPBIT_1, MB_BPS_9600);
```

Modbus Slave

MB_SLAVE_INIT_PORT2, MB_SLAVE_INIT_PORT3

This function initialize Modbus slave on port 2 or port 3 of a PLC.

Variable	Scope	Type	Description
`xInit`	INPUT	BIT	Signal to init data. It initialize every time there is rising edge on this parameter.
`iAddress`	INPUT	(Word[Signed])	What will be the address of this PLC in a Modbus network
`PortSettings`	INPUT	(Double word[Signed])	Result of MB_PORT_SETTINGS function

Example setup of slave with address 1 on port 2.

```
PortSettings := MB_PORT_SETTINGS(MB_PARITY_NONE, MB_STOPBIT_1, MB_BPS_9600);
M0 := MB_SLAVE_INIT_PORT2(TRUE, 1, PortSettings);
```

This is all you need to make your PLC a slave on port 2. Same example for port 3. `M0` is not used anywhere. It is just requirement for calling functions. It have to have left side.

Modbus Master

MB_MASTER_INIT_PORT2

This function initialize Modbus Master on port 2 (terminal connector).

Variable	Scope	Type	Description
`xInit`	INPUT	BIT	Signal to init data. It initialize every time there is rising edge on this parameter.
`PortSettings`	INPUT	(Double word[Signed])	Result of MB_PORT_SETTINGS function

Example setup of master on port 2.

```
PortSettings := MB_PORT_SETTINGS(MB_PARITY_NONE, MB_STOPBIT_1, MB_BPS_9600);
M0 := MB_MASTER_INIT_PORT2(TRUE, PortSettings);
```

MB_PROCESS_50

This function writes and reads from slave devices.

> Before you use it, install TimeControl50 library and setup `TCO_TICKER_50`. It is a ticker with 50ms increment intervals.

Variable	Scope	Type	Description
`mb_xEnable`	INPUT	BIT	Start processing channels.
`mb_arRegs`	IN_OUT	ARRAY(0..29) OR MB_REG_50	List of channels
`mb_iBuffer`	INPUT	(Word[Signed])	Number of `D` device for registers and `M` device for coils to use as a buffer. For instance if 100, then `D100` will be used as first device for buffer or `M100` for coils. It will take as many devices as many registers or coils are read from a single channel (`iNum`).
`mb_Timeout`	OUTPUT	(Word[Signed])	Number of device that timed out.

Create an array on 30 elements of `MB_REG_50` structure. This array is a list of channels you will process. Every channel may read\write up to 125 registers. After you created this array, you have to setup it once. For instance when PLC starts. You can use `M8002` flag.

This is the list of parameter of every `MB_REG_50` structure.

Variable	Type	Description
`iRDevNum`	(Word[Signed])	Number of `D` device to store result value for registers and `M` device for coils. For instance if set to `K200` then result of read will be in `D200` if read register or `M200` if read coil.
`iNum`	(Word[Signed])	Number of registers or coils to read. By default 1.
`iReg`	(Word[Signed])	Start Modbus (holding\input) register address in decimal.
`iRF`	Word[Unsigned]/Bit String[16-bit]	Modbus read Function. Default is `H3`.
`iWF`	Word[Unsigned]/Bit String[16-bit]	Modbus write Function. Default is `H6`.
`iDev`	Word[Unsigned]/Bit String[16-bit]	Address of slave device.
`tCycle`	(Word[Signed])	How often to read\write this channel. Use integer in `50ms` increment. For instance you want channel be read once `100ms`, then `tCycle` will be `2`. If you want only manual reads through `xReadOnce` parameter set it `0`.

Variable	Type	Description
iWR	(Word[Signed])	Read write mode. Default is MB_READ_WRITE. MB_READ or MB_WRITE could be set.
iPort	(Word[Signed])	From what port to access data. L02 or PLC\HMI have 2 ports. Port 2 set - 0, Port 3 set - 1, For Ethernet Modbus TCP set - 3
xEnabled	Bit	Channel will be read or write only if enabled.
xReadOnce	Bit	Once this parameter switch from OFF to ON it will read a channel once. If you want to read this channel only manually set tCycle to 0.
xWriteOnChange	Bit	If set to TRUE, it will write changes to slave immediately as detected otherwise will wait until tCycle is reached.

Supported functions

- H1 - Read Coils
- H2 - Read Discrete Inputs
- H3 - Read Holding Register
- H4 - Read Input Register
- H5 - Write Single Coil
- H6 - Write Single Register
- HF - Write Multiple Coils
- H10 - Write Multiple Registers

iRDevNum

Note that it uses double number of devices. If iNum is 1 and iRDevNum is 200, then D200 will have a result and D201 will be used for internal use. If iNum is 5, then D200-D204 will have results and D205-D209 will be used for internal calculations. You have to keep that in mind when setup channels.

For instance if you have setup like this.

```
arRegs[0].iRDevNum := 600;
arRegs[0].iNum := 3;

arRegs[1].iRDevNum := 603;
arRegs[1].iNum := 1;
```

This setup looks logical at first because on a second channel you configure next free device, but without taking in account internal calculation devices, thus will not work.

If you read 3 registers into D600 next free device will be D606.

iWF

2 functions a supported for write registers. `H6` to write single register and `H10` to write group of registers. When you set `H6` on a multiple registers channel, when change is detected, it will write only that particular register. If you set `H10` on a multiple registers channel, even if one register in that group is changed it will update all the registers in a group in one request. For double word use `H10` function. If you have more than one register in a channel but every register is a separate variable, user `H6`. For single register channel use `H6` too.

The same rule apply for working with coils. Use `H5` even on multiple coil channels.

Example

Example program.

```
IF M8002 THEN

    (* How many times timeout on a channel to mark it as suspended: Default 2 *)
    MB_TIMEOUT_COUNT := 2;
    (* How often to retry suspended channel in 50ms increments: Default 80 (4
seconds) *)
    MB_SUSPEND_RETRY := 80;
    (* How long no response is a timeout: Default 4 (200 milliseconds) *)
    MB_TIMEOUT_TIME := 4;

    PortSettings := MB_PORT_SETTINGS(MB_PARITY_NONE, MB_STOPBIT_1, MB_BPS_9600);
    (* Multi master available for both ports for L02 *)
    M0 := MB_MASTER_INIT_PORT2(TRUE, PortSettings);
    M0 := MB_MASTER_INIT_PORT3(TRUE, PortSettings);


    arRegs[0].xEnabled := TRUE;
    arRegs[0].iRDevNum := 600;
    arRegs[0].iNum := 3;
    arRegs[0].iReg := K16384;
    arRegs[0].iRF := H3;
    arRegs[0].iWF := H6;
    arRegs[0].iDev := H1;
    arRegs[0].tCycle := 20; (* 20 * 50ms = 1000ms or 1 second *)
    arRegs[0].xWriteOnChange := TRUE;
    arRegs[0].iWR := MB_READ_WRITE;
    arRegs[0].iPort := 0; (* 0 - Port 2, 1 - Port 3 *)


    (* Minimum setup and will connect devices through Port 3*)
    arRegs[1].xEnabled := TRUE;
    arRegs[1].iRDevNum := 610;
    arRegs[1].iReg := K16385;
    arRegs[1].iDev := K16;
    arRegs[1].tCycle := 4;
    arRegs[1].iWR := MB_READ;
    arRegs[1].iPort := 1;
END_IF;
```

```
fbMbProcess(mb_xEnable := TRUE, mb_arRegs := arRegs, mb_iBuffer := 100);
```

This is all you need. Now in `D600`, `D601` and `D602` will be a results of a first channel from device address 1, updated every second and if changed immediately update slave.

On a `D610` will be result of second channel. If `D610` is changed, it will not update slave but override it with new value from slave in 200ms.

there is also a system that suspend channels. If channel will timeout to respond for 2 times consequently it is marked as suspended. Suspended channels are requested once a `MB_SUSPEND_RETRY`. As soon as it retunes values, it starts to request channel according to `arRegs[1].tCycle`.