AN2002 通讯系统的应用

CREATE: 2009/02/15
UPDATE : 2010/07/05
Version 1.1
GUTTA Ladder Editor Version 1.1
http://www.plcol.com
http://www.visiblecontrol.com

概述

GUTTA 系统的通讯是以 MODBUS 通讯协议为基础来实现的。PLC 程序的上载、下载、监控以及单步调试都是通过扩展 MODBUS 13 号通讯指令来实现的。使用 13 号通讯指令时，PLC 必须为从站，GUTTA 编程软件必须为主站。若 PLC 站地址未知，且只连接了一台 PLC，GUTTA 编程软件可以使用广播地址 0 来与 PLC 通讯。一般情况下，MODBUS 从站不能响应广播地址 0，而这里，PLC 对广播地址的 13 号通讯指令依然进行应答。MODBUS 13 号通讯指令数据内容可参考《UM4001 GUTTA 通讯协议》。

在没有特殊配置的情况下，PLC 总是作为 MODBUS 从站。因此只要连接没有问题，通讯设置没有问题，并且知道从站站号（或者使用广播地址 0）。PLC 总是可以被 GUTTA 编程软件找到。在某些 PLC 程序的配置下，将 PLC 的某个通讯口配置为主站，可能会导致通讯口无法与 GUTTA 编程软件通讯。若将 PLC 的所有通讯口都配置为主站，就会导致 PLC 程序无法被修改和更新。这个时候可以采用 PLC 的 FLASH 编程工具 GUTTA Flash Utility 来清除 PLC 程序。PLC 程序被清除后，所有通讯口被恢复成默认配置。之后 GUTTA 编程软件就能轻松的找到 PLC 了。

GUTTA PLC 不需要进行特殊编程，总是默认为从站。若需要 PLC 做为主站主动发起通讯，则需要通过 PLC 指令来触发，这就需要编写一段 PLC 程序。如何编写这段 PLC 程序，正是下面需要介绍的。

MODBUS 通讯协议

MODBUS 协议定义了总线上主站（Master）与从站（Slave）之间的通讯格式。MODBUS 协议有 ASCII 和 RTU 两种格式，两种格式的通讯字段含义是相同的，差别在于字段的传输方法不同、帧开始与帧结束的判断条件不同、数据校验的方法不同。目前 GUTTA PLC 支持 RTU 通讯格式（虽然在 EC20/EC30 系统块中保留了 7 位 ASCII 模式的配置，但并未实现）。

MODBUS 通讯帧

在 RTU 通讯模式下，主站需要通过时间来判断 RTU 帧的开始和结束。主站监视总线上的通讯数据，如果发现总线有超过 4.5 个字符时间（时间绝对值随字符传送波特率的变化而变化）的空闲，则认为一个帧已经结束（如果存在），同时认为下一个字符为下一帧的开始。在 EC20/EC30 的系统功能块中，可以手动设置这个时间，使其适应某些特殊的通讯场合。

完整的 RTU 帧应该是下面的格式（不论是主站发起还是从站应答）：

空闲	地址	功能码	数据	CRC 校验	空闲时间
	1 字节	1 字节	N 字节	2 字节	

地址

MODBUS 协议的站地址由一个字节组成，站地址用来指示哪个从设备来应答主站的通

讯报文。在总线上，每个从设备必须指定一个唯一的站地址，只有当通讯报文中地址与该从设备地址相同时，该从设备才能应答主站的通讯报文。从设备应答的通讯报文也必须包含该地址，以告知主站，这个通讯报文是哪个从设备应答的。广播报文的地址是零，所有的从站可以根据广播报文进行相应的动作，但是一般不能应答该广播报文。

功能码

功能码指示从设备应该执行什么动作。若应答的功能码最高位被置位，则表示从设备不能够正确执行此功能码。若一致，则表示从设备能够正确执行此功能，并能够返回功能码所需要的数据（如果有）。做为从站，目前 GUTTA PLC 只实现了部分常用的 MODBUS 通讯指令：

- 01 - 读保持线圈状态（Read coil status）
- 02 - 读输入线圈状态（Read input status）
- 03 - 读保持寄存器（Read holding register）
- 04 - 读输入寄存器（Read input register）
- 05 - 写单个线圈（Force single coil）
- 06 - 写单个寄存器（Preset single register）
- 15 - 写多个线圈（Force multiple coils）
- 16 - 写多个寄存器（Preset multiple registers）

MODBUS 通讯地址和 GUTTA PLC 地址的对应关系，请参考文档《IN1001 GUTTA 内存使用》。新版的 GUTTA Ladder Editor 自带了 MODBUS 地址查询工具。

使用 MODBUS 地址查询工具，需要选择一个数据宽度，然后在右边输入一个符合该数据宽度的 PLC 变量，该变量实际占用的内存在下方以绿色的叉叉来标示。例如在上面输入变量 MB31 后，下方对应的图示含义为，MB31 占用了 8 个位，位于 MODBUS 变量 400024 的高 8 位。

在 MODBUS 通讯协议中，除了位（Coil），协议认为所有的寄存器都是 16 位的字（Register），这可能与早期 MODICON 的 PLC 都采用 16 位处理器以及 PLC 指令相关。EC20/EC30 系统中，PLC 的变量宽度可以是 8 位、16 位、32 位；而且不要求对齐。例如 MW0、MW1、MD2、MD3 都是合法的 PLC 变量。这将导致这些 PLC 数据不能够简单的被 MODBUS 主站访问。在这里建议尽量使用长度对齐的变量地址，例如 MW 变量的偏移尽量为 0、2、4、6 等。MD 变量的偏移尽量为 0、4、8、12 等。这样做一方面可以简化连接人机设备（触摸屏）的难度，另一方面，可以优化某些 PLC 类型的程序执行速度（Cortex-M3 内核在访问长度对齐数据要快于非对齐数据）。

数据

数据是主站需要发送给从站的数据，或者是从站需要返回给主站的数据。数据的具体含义由功能码来定义。特别的，有些功能码不包含数据区，数据区大小 N 可以为 0。

校验码

校验码让接收数据方来检查通讯的传输过程中是否有错误发生。有时因为干扰使得数据传输过程中发生了错误，而校验码使得数据接收方能够判断这种错误并忽略此通讯报文。校验码极大的增加了 MODBUS 系统的安全性。具体方式是，发送方根据所发送的数据，采用特定的算法生成一个校验码，并将校验码放在发送数据的后面一起发送。接收方接收到通讯报文后，根据前面的数据部分，采用同样的算法也生成一个校验码，然后比较自己生成的校验码和通讯报文中的校验码是否一致，若一致，则表示通讯报文是有效的。在通讯过程中，不论是数据发生了传输错误还是校验码发生了传输错误，都会导致检验不一致。当然，数据和校验码同时发生了传输错误且刚好校验一致的可能性是有的，概率却微乎其微。

在 MODBUS 中，RTU 模式必须采用 CRC16 校验码。在单片机中实现一般有两种方法，查表法或者运算法，有兴趣的读者可以从网上找到实现的源代码。出于速度的考虑，EC20/EC30 绝大部分系统采用查表方式运算 CRC16。

MODBUS 功能码

01 读保持线圈状态（Read coil status）

描述

本指令读取从设备的离散量输出（0X）状态（ON 或者 OFF）。不支持广播。

发送

发送数据必须包含需要读取线圈的起始地址和线圈个数。注意线圈地址是从 0 开始的。线圈 1-16 在这里用 0-15 来寻址。

下面是一个从 17 号从设备读取线圈 00020-00056 的例子：

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#01
开始地址（高字节）	16#00
开始地址（低字节）	16#13
数量（高字节）	16#00
数量（低字节）	16#25
校验码（LRC 或者 CRC）	--

应答

线圈的状态通过数据中的位来传送。数据位 1 表示线圈为 ON，数据位 0 表示线圈为 OFF。第一个数据字节的小端位（LSB）为第一个需要查询的线圈状态，其余的线圈状态紧跟其后。若线圈个数不是 8 的倍数，多余的位需要被 0 填充，即最后一个数据字节的大端位（MSB）。同时，报文中包含字节数，用来指示一共有多少个数据字节需要被传送。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#01
字节数	16#05
数据（线圈 00027-00020）	16#CD
数据（线圈 00035-00028）	16#6B
数据（线圈 00043-00036）	16#B2
数据（线圈 00051-00044）	16#0E
数据（线圈 00056-00052）	16#1B
校验码（LRC 或者 CRC）	--

02 读输入线圈状态（Read input status）

描述

本指令读取从设备的离散量输入（1X）状态（ON 或 OFF）。不支持广播。

发送

发送数据必须包含需要读取线圈的起始地址和线圈个数。注意线圈地址是从 0 开始的。线圈 1-16 在这里用 0-15 来寻址。

下面是一个从 17 号从设备读取线圈 10197-10218 的例子：

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#02
开始地址（高字节）	16#00
开始地址（低字节）	16#C4
数量（高字节）	16#00
数量（低字节）	16#16
校验码（LRC 或者 CRC）	--

应答

线圈的状态通过数据中的位来传送。数据位 1 表示线圈为 ON，数据位 0 表示线圈为 OFF。第一个数据字节的小端位（LSB）为第一个需要查询的线圈状态，其余的线圈状态紧跟其后。若线圈个数不是 8 的倍数，多余的位需要被 0 填充，即最后一个数据字节的大端

位（MSB）。同时，报文中包含字节数，用来指示一共有多少个数据字节需要被传送。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#02
字节数	16#03
数据（线圈 10204-10197）	16#AC
数据（线圈 10212-10205）	16#DB
数据（线圈 10218-10213）	16#35
校验码（LRC 或者 CRC）	--

03 读保持寄存器（Read holding register）

描述

本指令读取从设备保持寄存器（4X）的二进制值。不支持广播。

发送

发送数据必须包含需要读取寄存器的起始地址和寄存器个数。注意寄存器地址是从 0 开始的。寄存器 1-16 在这里用 0-15 来寻址。

下面是一个从 17 号从设备读取寄存器 40108-40110 的例子：

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#03
开始地址（高字节）	16#00
开始地址（低字节）	16#6B
数量（高字节）	16#00
数量（低字节）	16#03
校验码（LRC 或者 CRC）	--

应答

寄存器的二进制值通过两个字节来传送。对于每个寄存器：第 1 个字节为寄存器的高位字节（MSB），第 2 个字节为寄存器的低位字节（LSB）。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#03
字节数	16#06
高位字节（寄存器 40108 高字节）	16#02

低位字节（寄存器 40108 低字节）	16#2B
高位字节（寄存器 40109 高字节）	16#00
低位字节（寄存器 40109 低字节）	16#00
高位字节（寄存器 40110 高字节）	16#00
低位字节（寄存器 40110 低字节）	16#64
校验码（LRC 或者 CRC）	--

在这个应答中：寄存器 40108 的值为 555（16#022B）、寄存器 40109 的值为 0（16#0）、寄存器 40110 的值为 100（16#0064）。

04 读输入寄存器（Read input register）

描述

本指令读取从设备输入寄存器（3X）的二进制值。不支持广播。

发送

发送数据必须包含需要读取寄存器的起始地址和寄存器个数。注意寄存器地址是从 0 开始的。寄存器 1-16 在这里用 0-15 来寻址。

下面是一个从 17 号从设备读取寄存器 30009 的例子：

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#04
开始地址（高字节）	16#00
开始地址（低字节）	16#08
数量（高字节）	16#00
数量（低字节）	16#01
校验码（LRC 或者 CRC）	--

应答

寄存器的二进制值通过两个字节来传送。对于每个寄存器：第 1 个字节为寄存器的高位字节，第 2 个字节为寄存器的低位字节。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#03
字节数	16#02
高位字节（寄存器 30009 高字节）	16#00
低位字节（寄存器 30009 低字节）	16#0A
校验码（LRC 或者 CRC）	--

在这个应答中：寄存器 30009 的值为 10（16#000A）。

05 写单个线圈（Force single coil）

描述

本指令将从设备的某个保持线圈（0X）状态设置为 ON 或者 OFF。在广播时，与总线相连的所有从设备相同地址上的线圈状态被设置。

发送

发送数据必须包含需要设置线圈的地址。注意线圈地址是从 0 开始的。线圈 1 在这里用 0 来寻址。线圈需要被设置的状态在数据中：数据 16#FF00 表示需要将线圈设置为 ON；数据 16#0000 表示需要将线圈设置为 OFF。其余的值将被忽略。

下面是一个把 17 号从设备线圈 00173 设置为 ON 的例子：

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#05
开始地址（高字节）	16#00
开始地址（低字节）	16#AC
数据（高字节）	16#FF
数据（低字节）	16#00
校验码（LRC 或者 CRC）	--

应答

若线圈设置成功，应答是发送数据的一份拷贝。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#05
开始地址（高字节）	16#00
开始地址（低字节）	16#AC
数据（高字节）	16#FF
数据（低字节）	16#00
校验码（LRC 或者 CRC）	--

06 写单个寄存器（Preset single register）

描述

本指令将从设备的某个保持寄存器（4X）设置为指定值。在广播时，与总线相连的所有从设备相同地址上的寄存器值被设置。

发送

发送数据必须包含需要设置寄存器的地址。注意寄存器地址是从 0 开始的。寄存器 1 在这里用 0 来寻址。寄存器需要被设置的值在数据中。

下面是一个把 17 号从设备寄存器 40002 设置为 16#0003 的例子：

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#06
开始地址（高字节）	16#00
开始地址（低字节）	16#01
数据（寄存器 40002 高字节）	16#00
数据（寄存器 40002 低字节）	16#03
校验码（LRC 或者 CRC）	--

应答

若寄存器设置成功，应答是发送数据的一份拷贝。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#06
开始地址（高字节）	16#00
开始地址（低字节）	16#01
数据（寄存器 40002 高字节）	16#00
数据（寄存器 40002 低字节）	16#03
校验码（LRC 或者 CRC）	--

15 写多个线圈（Force multiple coils）

描述

本指令将从设备的一段连续保持线圈（0X）状态设置为指定值。在广播时，与总线相连的所有从设备相同地址上的线圈状态被设置。

发送

发送数据必须包含需要设置线圈的地址。注意线圈地址是从 0 开始的。线圈 1 在这里用 0 来寻址。线圈需要被设置的状态在数据中：数据位 1 表示需要将线圈设置为 ON；数据位 0 表示需要将线圈设置为 OFF。

例如我们需要设置 17 号从设备从 00020 开始的连续 10 个线圈的值。数据区第 1 个被传送的字节（16#CD）表示线圈 00020-00027 的设置值。小端位表示低地址线圈 00020，大端位表示高地址线圈 00027。第 2 个被传送的字节（16#01）表示线圈 00028-00029 的设置值，小端位表示低地址线圈 00028，大端不需要使用的位保留为 0。

	MSB			第 1 个字节			LSB			MSB			第 2 个字节			LSB	
位	1	1	0	0	1	1	0	1		0	0	0	0	0	0	0	1
线圈	27	26	25	24	23	22	21	20		-	-	-	-	-	-	29	28

发送	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#0F
开始地址（高字节）	16#00
开始地址（低字节）	16#13
数量（高字节）	16#00
数量（低字节）	16#0A
字节数	16#02
数据（线圈 00020-00027）	16#CD
数据（线圈 00028-00029）	16#01
校验码（LRC 或者 CRC）	--

应答

若线圈设置成功，应答包括从站地址、功能码、开始地址、数量。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#0F
开始地址（高字节）	16#00
开始地址（低字节）	16#13
数量（高字节）	16#00
数量（低字节）	16#0A
校验码（LRC 或者 CRC）	--

16 写多个寄存器（Preset multiple registers）

描述

本指令将从设备的一段连续保持寄存器（4X）设置为指定值。在广播时，与总线相连的所有从设备相同地址上的寄存器值被设置。

发送

发送数据必须包含需要设置寄存器的地址。注意寄存器地址是从 0 开始的。寄存器 1 在这里用 0 来寻址。寄存器需要被设置的值在数据中。

下面是一个把 17 号从设备寄存器 40002、40003 设置为 16#000A、16#0102 的例子：

发送

段名	数据（16 进制格式）
从站地址	16#11
功能码	16#10
开始地址（高字节）	16#00
开始地址（低字节）	16#01
数量（高字节）	16#00
数量（低字节）	16#02
字节数	16#04
数据（寄存器 40002 高字节）	16#00
数据（寄存器 40002 低字节）	16#0A
数据（寄存器 40003 高字节）	16#01
数据（寄存器 40003 低字节）	16#02
校验码（LRC 或者 CRC）	--

应答

若寄存器设置成功，应答包括从站地址、功能码、开始地址、数量。

下面是一个应答的例子：

应答	
段名	数据（16 进制格式）
从站地址	16#11
功能码	16#10
开始地址（高字节）	16#00
开始地址（低字节）	16#01
数量（高字节）	16#00
数量（低字节）	16#02
校验码（LRC 或者 CRC）	--

EXCH 指令介绍

EC20/EC30 系统默认支持 MODBUS 协议。由于 PLC 程序的上传和下载就是使用的 MODBUS 协议，故只要能上传或下载程序，那么就能进行 MODBUS 通讯。这里 PLC 总是做为从站，根据主站发出的 MODBUS 通讯请求，返回正确的应答。例如连接触摸屏和 PLC，让触摸屏显示和修改 PLC 中变量，是不需要编写任何通讯程序的。只要正确的设置了 PLC 的通讯参数，并在触摸屏中正确的配置变量的 MODBUS 地址即可。

如果需要 PLC 作为主站主动去访问其他 PLC 从站，则需要使用 EXCH 指令。

串口数据交换（EXCH）指令根据输入 TBL 中给出的通讯参数表格进行一次串口通讯操作，输入 PORT 标识通讯串口号。您可以在程序中保持任意数目的 EXCH 指令，但在任何时间最多只能有 8 条 EXCH 被放入通讯队列。若通讯队列满，EXCH 指令将不进行任何操作（亦不设置错误号）。EXCH 指令的使能位输入能流可以是脉冲，EXCH 指令将尝试进行单次通讯。EXCH 指令的使能位输入能流也可以一直保持，EXCH 指令会检查自己是否在通讯队列中，若不在指令尝试添加自己到通讯队列中，否则等待通讯结果。

EXCH 指令有两个操作数。

- TBL - 参数表格的第一个字节。

- PORT - 进行通讯操作的串口号。

偏移量	含义	格式	类型	说明
0.0	A 队列	BIT	输出	当通讯功能进入队列，A 设置为 1。当一个通讯功能移出队列（通讯完成或者出错），A 设置为 0。
0.1	D 完成	BIT	输出	当通讯功能进入队列，D 设置为 0。当一个通讯功能完成，D 设置为 1。
0.2	E 错误	BIT	输出	当通讯功能进入队列，E 设置为 0。当一个通讯功能错误，E 设置为 1。
1	错误号	BYTE	输出	当 E 被设置为 1 时，错误号同时被指定。
2	发送数据长度	UINT	输入	发送数据的长度，以字节记。
4	发送数据偏移量	UINT	输入	发送数据的首地址偏移量。发送数据必须在 M 区。
6	接收数据长度	UINT	输入	接收数据缓冲的长度，以字节记。
8	接收数据偏移量	UINT	输入	接收数据缓冲的首地址偏移量。接收数据必须在 M 区。

错误号可以是下面的值：

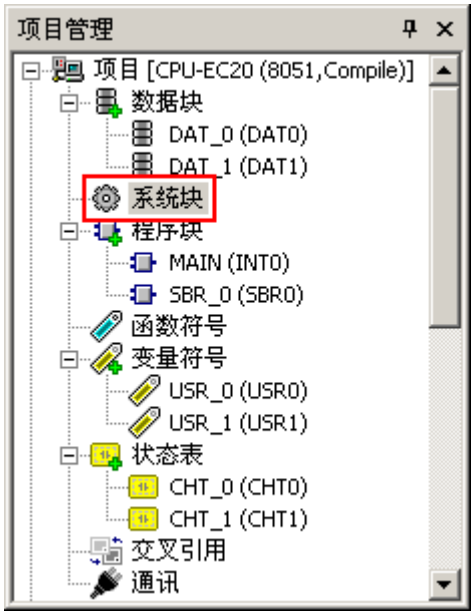
- 0 - 无错。
- 1 - 表格参数错误。
- 2 - 超时错误；远程站不应答。
- 3 - 接收错误；远程站应答中存在校验错误。
- 4 - 接收错误；接收缓冲区长度不够。

应用实例

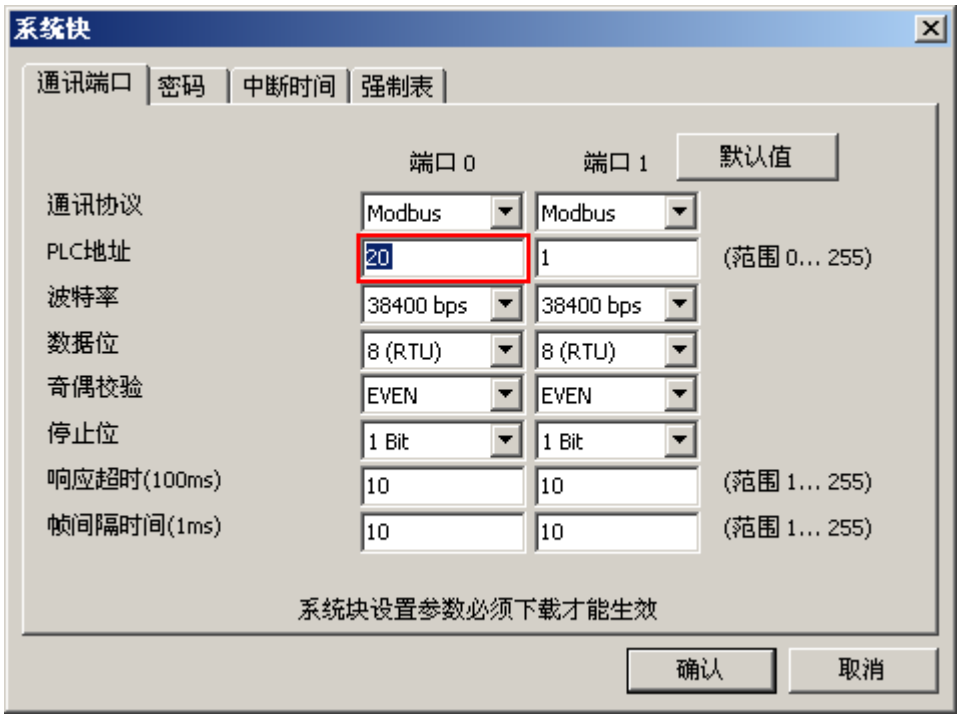
使用 ModScan32 工具

ModScan32 是 Win-Tech 推出的 MODBUS 通讯工具，这个软件可以通过电脑 COM 端口发送标准的 MODBUS 主站请求与从站进行通讯。使用 ModScan32，需要至少一个 MODBUS 从站。可以使用 GUTTA Simulator 软件模拟器模拟一个 MODBUS 从站，只需要给 GUTTA Simulator 软件模拟器绑定一个计算机串口。这里我们使用 PLC 实验板：CPU-EC20 (8051) 来完成这个实验。其它类型的试验板和模拟器的使用基本类似，不再重复介绍。

前面说过，使用 EC20/EC30 系统，PLC 默认就是从站，无需编程，只需要配置通讯。运行 GUTTA Ladder Editor 软件，双击项目管理窗口的项目，选择 PLC 类型为 CPU-EC20 (8051)。新建一个项目，双击系统块：



编辑系统块中的通讯端口：

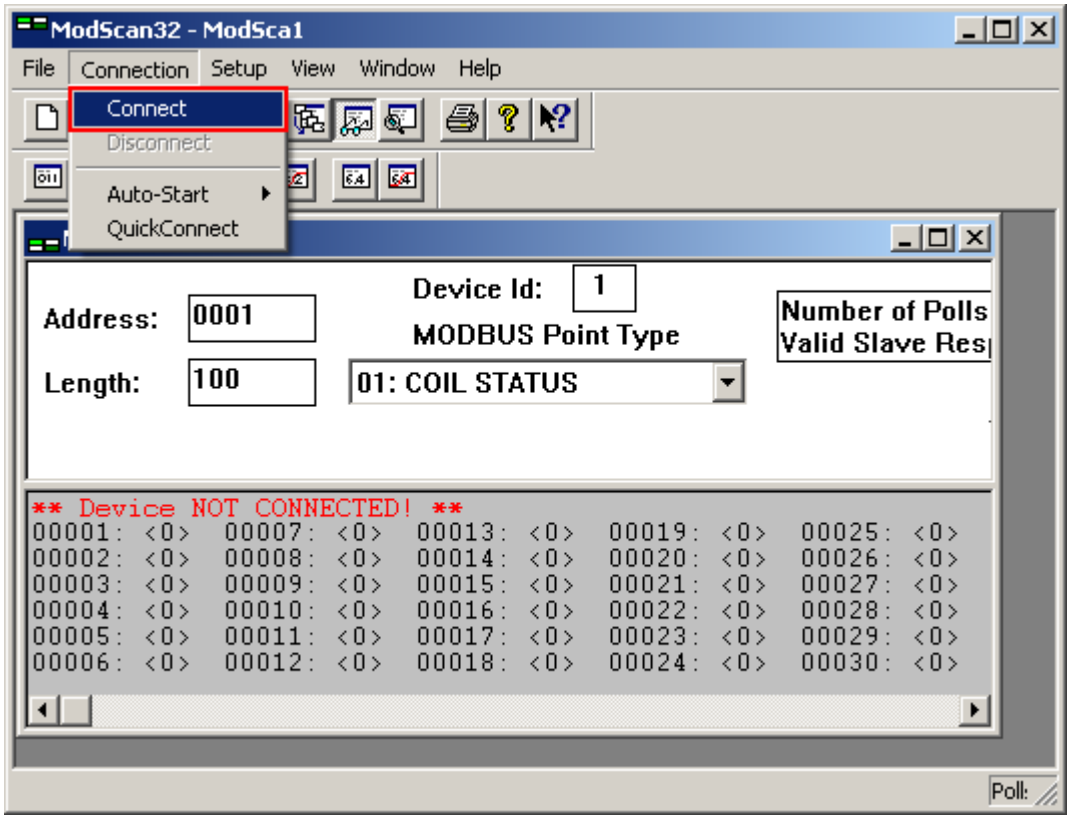


由于 CPU-EC20 (8051)只有 1 个串行通讯口，端口 1 可以忽略，我们只需要配置端口 0。

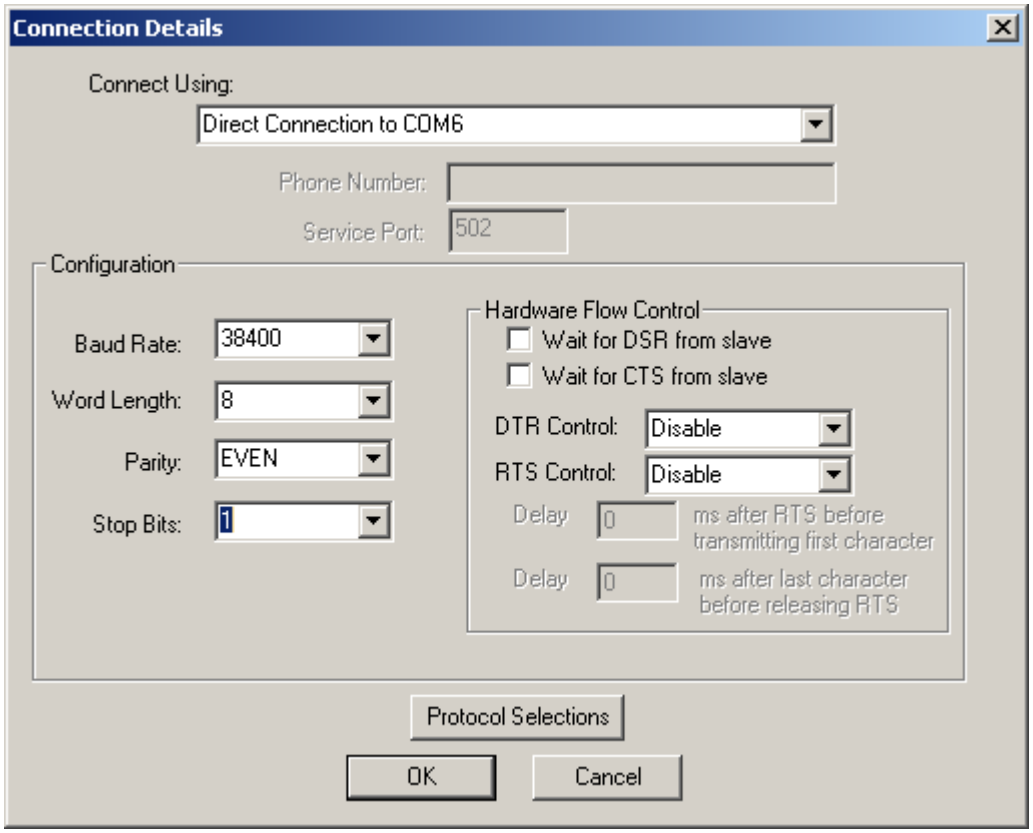
- 通讯协议： 必须选择 Modbus（Free Port 自由口在 EC20/EC30 上并没有实现）。
- PLC 地址： 这里我们修改成 20。
- 波特率： 这里我们使用默认的 38400 bps。
- 数据位： 这里我们使用默认的 8 RTU（7 位的 ASCII 模式在 EC20/EC30 上并没有实现）。
- 奇偶校验： 这里我们使用默认的偶校验 EVEN。
- 停止位： 这里我们使用默认的 1 Bit。
- 响应超时： 这里我们使用默认的 10（10×100ms = 1000ms）。
- 通讯帧： 这里我们使用默认的 10（10×1ms = 10ms）。

点击确认，保存系统块配置，将这个空白的程序下载到 PLC 试验板 CPU-EC20 (8051)。

启动 ModScan32，单击 Connection 菜单下的 Connect 命令。



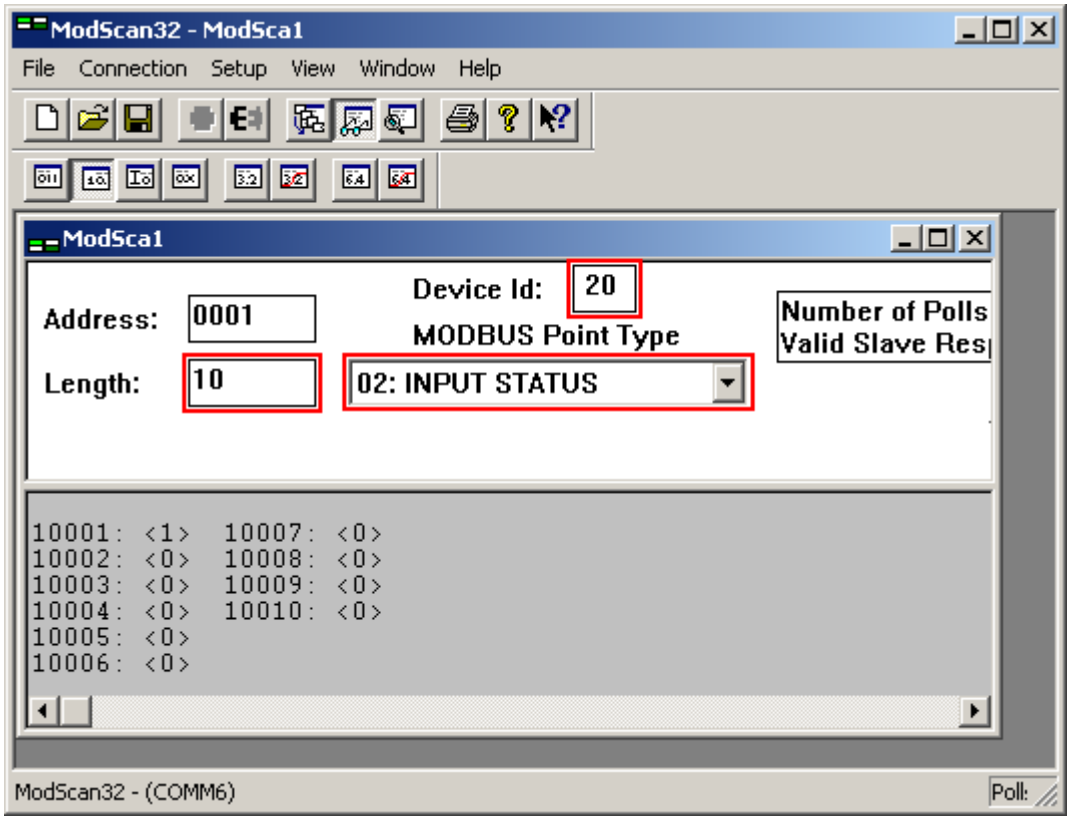
出现通讯端口配置对话框：



- Connect Using：选择通讯端口，可以使用 CPU-EC20 (8051)下载程序的通讯端口。

- Configuration：
  - Baud Rate：波特率，修改为 38400 bps。
  - Word Length：数据位，使用 8 位 RTU 模式。
  - Parity：效验位，修改为偶校验 EVEN。
  - Stop Bits：停止位，使用 1 Bit。

其余参数使用默认值，点击确认，保存设置。



在 ModSca1 窗口中，做如下修改：

- Device Id：修改为 20，即前面配置的 PLC 站号。
- MODBUS Point Type：修改为 INPUT STATUS，读取输入线圈的值。
- Length：EC20/EC30 的数字量输入区间都很短，I0.0 ~ I7.7 或 I0.0 ~ I15.7，不足 100 个，如果使用默认的 100，EC20/EC30 会按照 MODBUS 协议标准返回错误代码。

按照上面的说明修改后，应该就能建立正常的 MODBUS 通讯了，ModSca1 窗口下的红字** Device NOT CONNECTED! **应该会消失。按下或弹起试验板 CPU-EC20 (8051)的 I0.0 开关，观察 I0.0 对应的 MODBUS 变量 10001 是否有变化。

工具软件：**ModScan32.zip**

http://www.plcol.com/technologies/anindex/an2002/ModScan32.zip

PLC 程序：**Sample1_Lad.vcw**

http://www.plcol.com/technologies/anindex/an2002/Sample1_Lad.vcw

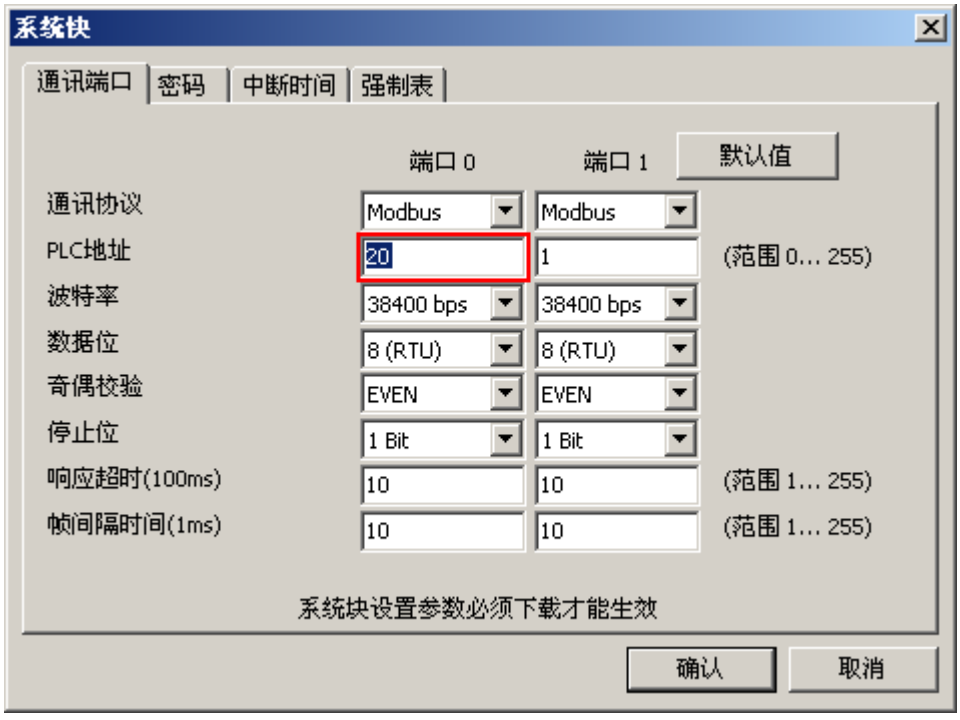
使用触摸屏

由于 EC20/EC30 默认支持 MODBUS 从站协议，用触摸屏连接 EC20/EC30 系统，关键是配置好控件变量的 MODBUS 通讯地址。这里我们实现一个简单的例子，用触摸屏显示 PLC 的 2 个模拟量输入。改变 PLC 模拟量值，触摸屏上显示的值对应的发生变化。触摸屏上还有一个按键，用于决定当前显示哪个模拟量。例如即按下此按键，显示模拟量通道 0；松开此按键，显示模拟量通道 1。

为了大家试验方便，这里使用斯耐德的触摸屏开发软件 Vijeo-Designer Version 4.10。这个软件包含一个非常优秀的模拟系统，可以在电脑上模拟触摸屏的很多功能。我们将使用 Vijeo-Designer Version 4.10 的模拟器，效果和使用实际的触摸屏一致。

配置 MODBUS 从站：PLC

基于最开始提出的要求，我们先给出 CPU-EC20 (8051)的 PLC 程序。和上一节类似，运行 GUTTA Ladder Editor 软件，双击项目管理窗口的项目，选择 PLC 类型为 CPU-EC20 (8051)。新建一个项目，双击系统块，编辑系统块中的通讯端口：



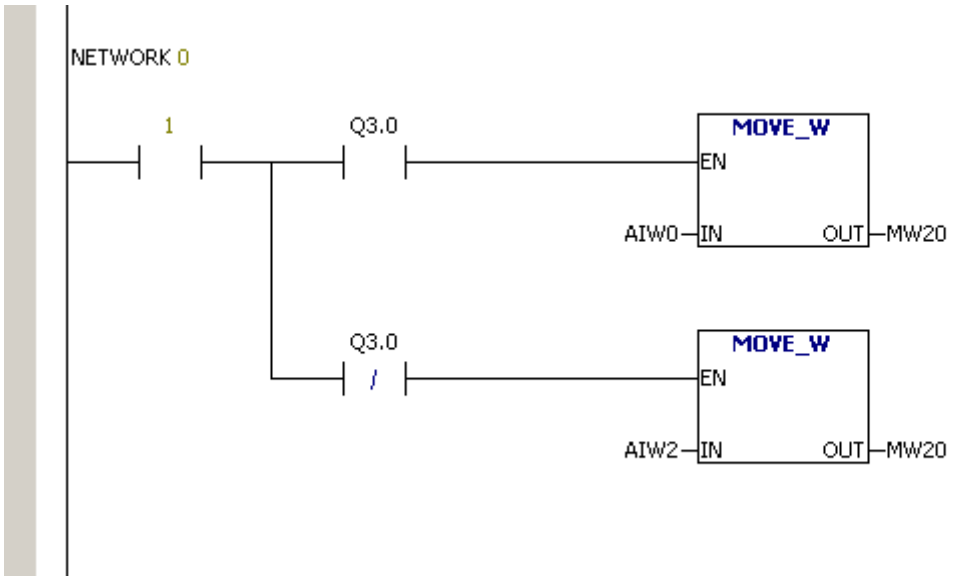
由于 CPU-EC20 (8051)只有 1 个串行通讯口，端口 1 可以忽略，我们只需要配置端口 0。

- 通讯协议： 必须选择 Modbus（Free Port 自由口在 EC20/EC30 上并没有实现）。
- PLC 地址： 这里我们修改成 20。
- 波特率： 这里我们使用默认的 38400 bps。
- 数据位： 这里我们使用默认的 8 RTU（7 位的 ASCII 模式在 EC20/EC30 上并没有实现）。
- 奇偶校验： 这里我们使用默认的偶校验 EVEN。
- 停止位： 这里我们使用默认的 1 Bit。
- 响应超时： 这里我们使用默认的 10（10×100ms = 1000ms）。

- 通讯帧： 这里我们使用默认的 10（10×1ms = 10ms）。

点击确认，保存系统块配置，然后根据需要编写 PLC 程序：

MAIN（INT0）



这个程序很简单，判断当 Q3.0 为 1 时，将模拟量通道 1（AIW2）的值拷贝给 MW20；判断当 Q3.1 为 1 时，将模拟量通道 0（AIW0）的值拷贝给 MW20。

通过软件自带的 MODBUS 地址查询工具可以知道：

- PLC 变量 Q3.0 对应的 MODBUS 地址为 000025（000017+8=00025）。

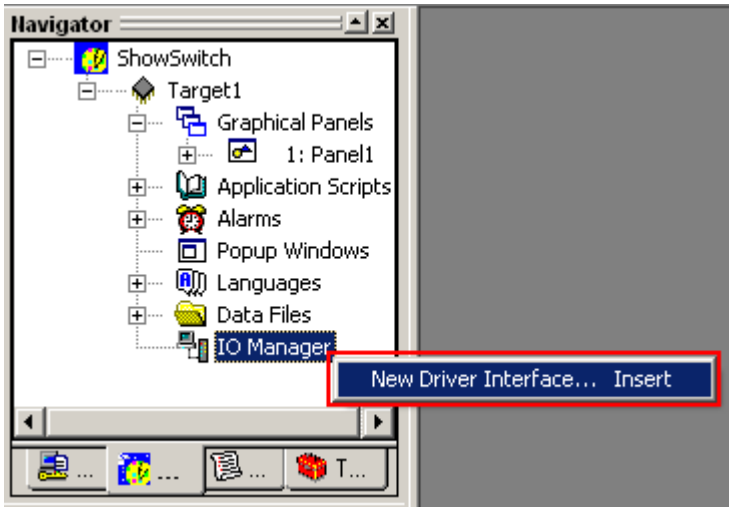
- PLC 变量 MW20 对应的 MODBUS 地址为 400019。

配置 MODBUS 主站：触摸屏

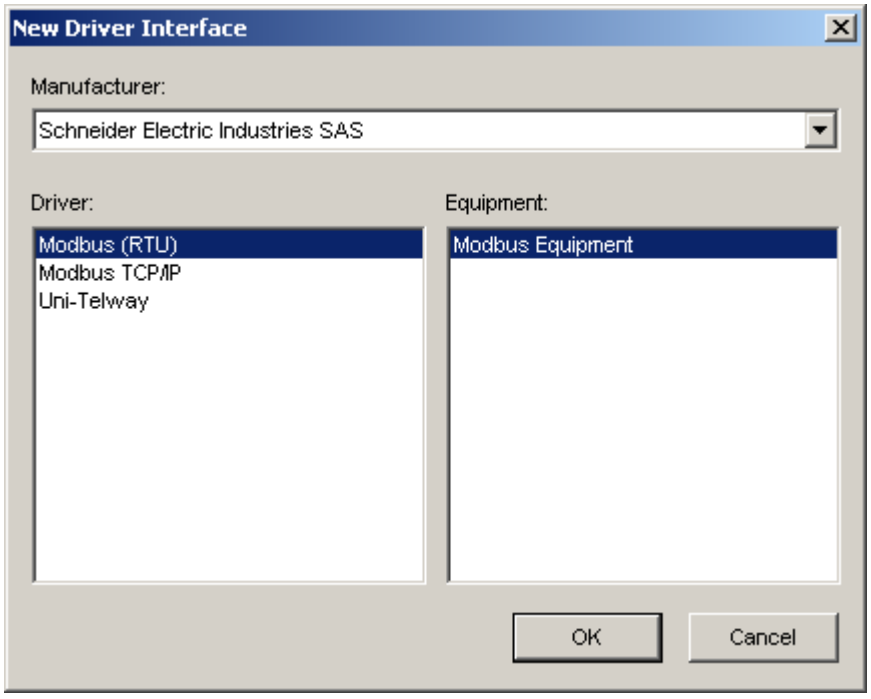
基于最开始提出的要求，我们只知道，触摸屏需要将 400019（MW20）显示出来，同时通过按键控制 000025（Q3.0）即可，PLC 逻辑会根据 Q3.0 的值，决定 400019（MW20）等于通道 0（AIW0）还是通道 1（AIW2）。

配置总线

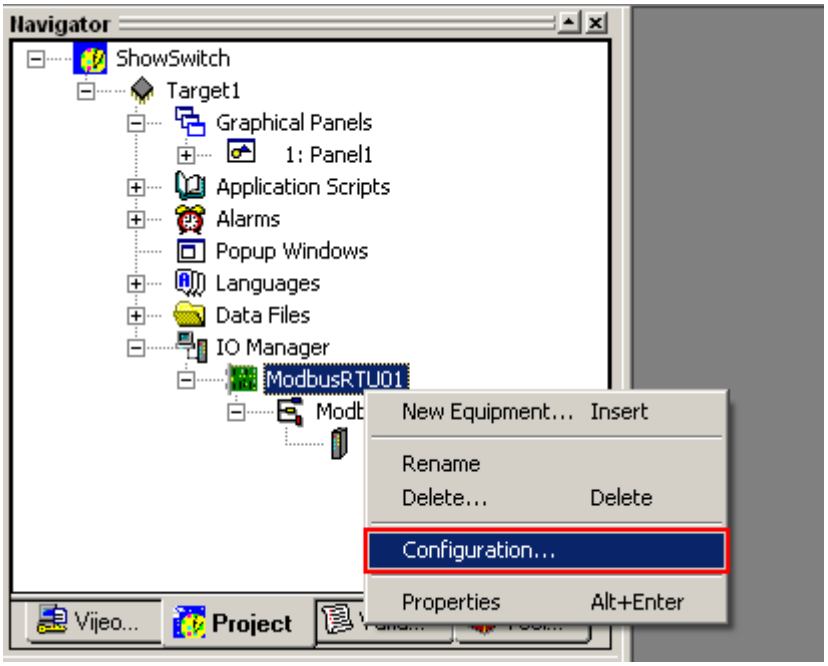
在 Vijeo-Designer Version 4.10 中，新建一个项目 ShowSwitch。要连接外部设备，首先要创建 MODBUS 总线，在 Navigator 中，右键单击 IO Manager，弹出菜单。



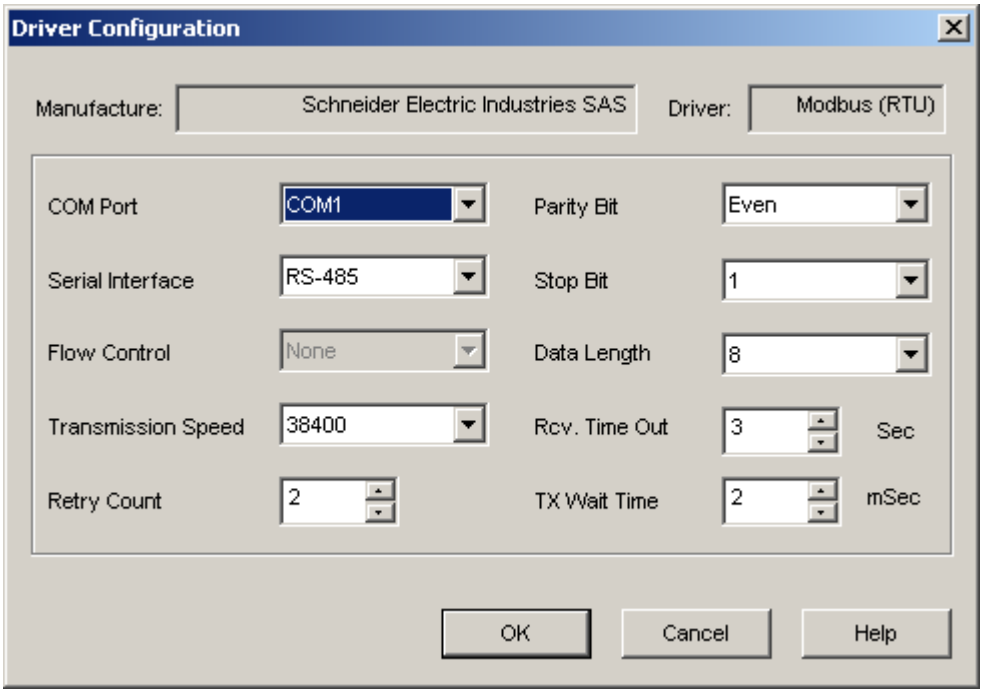
单击 New Device Interface...，创建总线，此时出现总线类型选择对话框。



默认配置就是 Modbus (RTU)，不用修改，直接点击 OK 按键保存配置。此时 Navigator 结构发生变化，IO Manager 下面多出一些子节点。在 Navigator 中，右键单击 ModbusRTU01，弹出菜单。



单击 Configuration...，此时出现总线配置对话框。

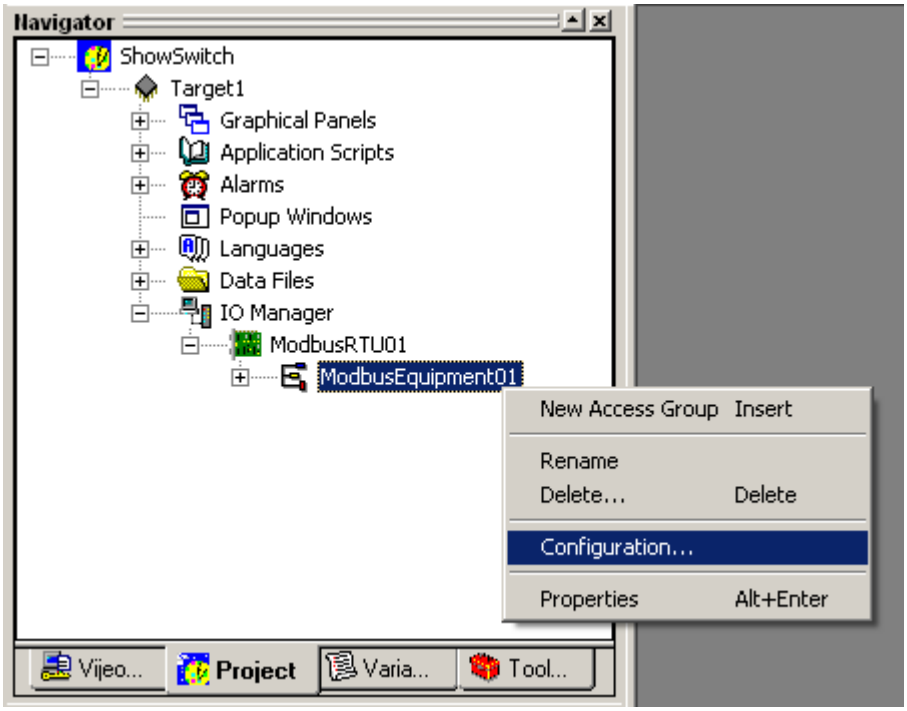


根据上前面 PLC 的配置，做出相应的修改：

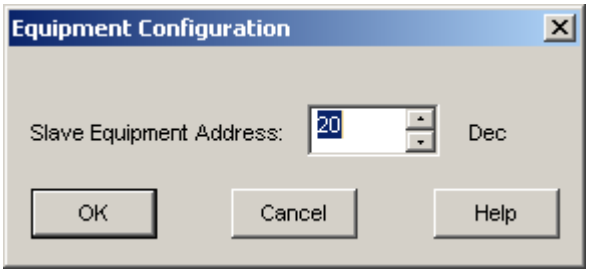
- COM Port：　　必须使用 COM1，因为目标触摸屏只有两个 COM 口，COM1 和 COM2，而 COM2 不能用 RS485 通讯，故必须使用 COM1。触摸屏模拟器运行后会很机械的将电脑上的 COM1 当作触摸屏的 COM1 来使用，这个行为不可配置。万一电脑的 COM1 不存在或者是存在硬件故障而无法使用怎么办呢？可以通过 Windows 的设备管理器解决这个问题。在设备管理器中，可以手动设定硬件的端口号。如果 COM1 已经分配了硬件，而此硬件是我们不希望使用的硬件，将此硬件的 COM 号修改为其他值即可。然后将我们希望模拟器使用的串口硬件（例如 CPU-EC20 (8051)试验板上自带的 USB 虚拟串口 Prolific USB-to-Serial Comm Port），分配到 COM1 上即可。

- Serial Interface： 这里使用 RS485。我们实际的传输是 USB 或者 RS232、当然也可以使用 CPU-EC20 (8051)试验板上自带的 RS485。不论实际的物理链路是什么，这里都选择 RS-485。毕竟我们不是连真实的屏，模拟器也模拟不出 RS485 来，它只是向 Windows 系统中的 COM 口发送和接收数据而已。根据我们的实验，模拟器使用 RS485 较为稳定。
- Transmission Speed： 必须和 PLC 设定一致，38400 bps。
- Retry Count： 使用默认的 2。
- Parity Bit： 必须和 PLC 设定一致，偶校验 Even。
- Stop Bit： 必须和 PLC 设定一致，1 Bit。
- Data Length： 必须和 PLC 设定一致，8 位 RTU 模式。
- Rcv. Time Out： 使用默认值。
- TX Wait Time： 使用默认值。

点击 OK 按键保存配置。在 Navigator 中，右键单击 ModbusEquipment01，弹出菜单。



单击 Configuration...，此时出现设备配置对话框。

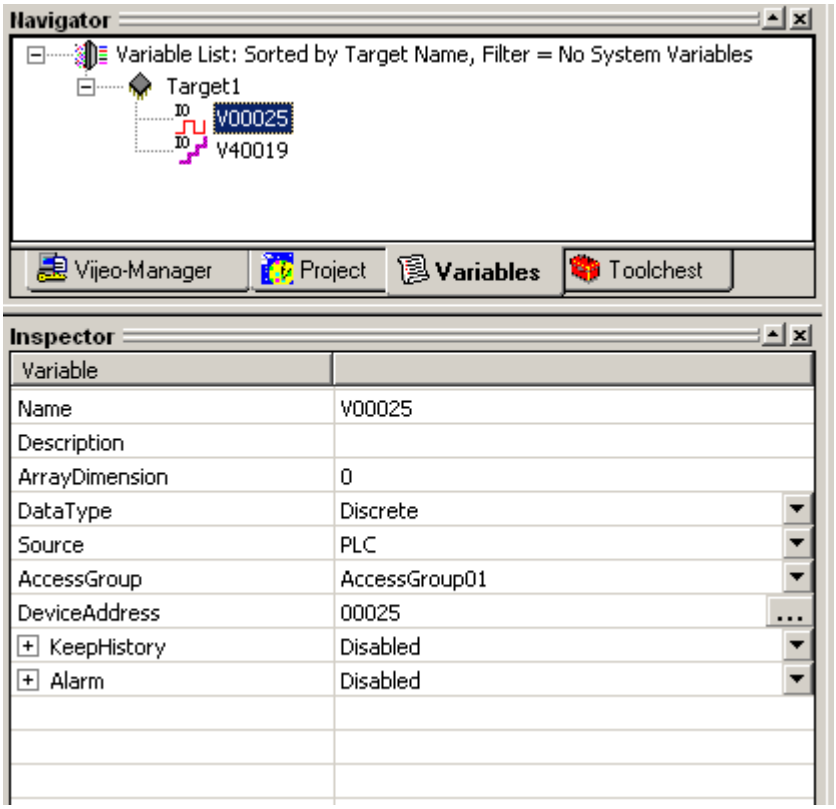


由于我们的 PLC 子站地址是 20，故这里填入 20，点击 OK 按键保存配置。

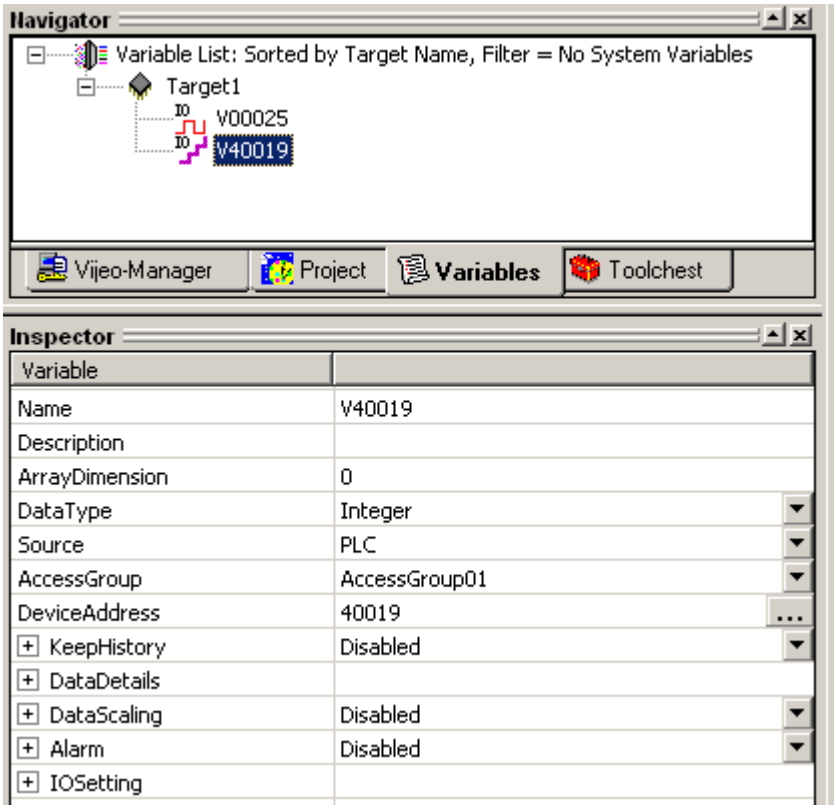
配置变量

在 Navigator 的 Variables 页面中，添加两个变量：

- 添加 I/O 变量 V00025，对应的配置如下：



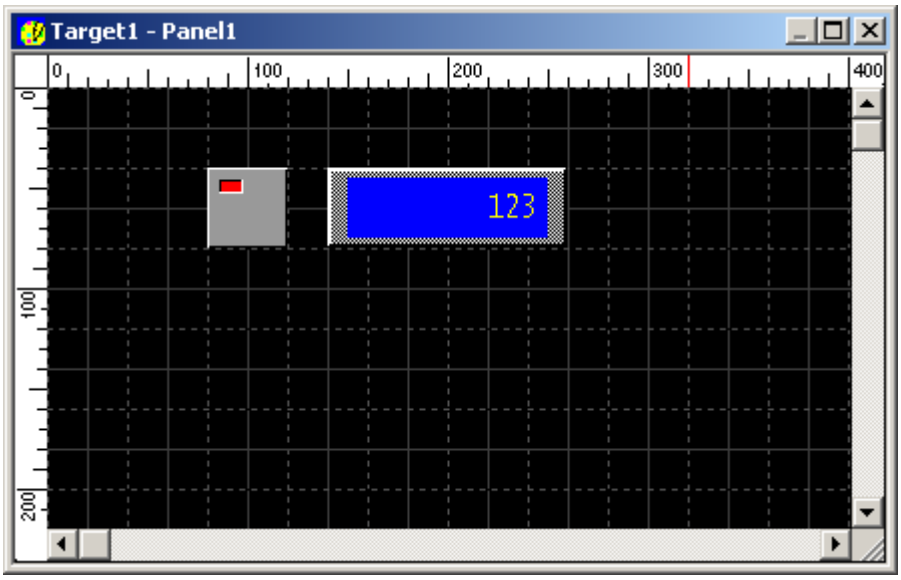
- 添加 I/O 变量 V40019，对应的配置如下：



配置页面

在项目的第一个显示页面 1:Panel1 中，插入两个控件，一个用于显示模拟量，一个用

于切换两个通道：

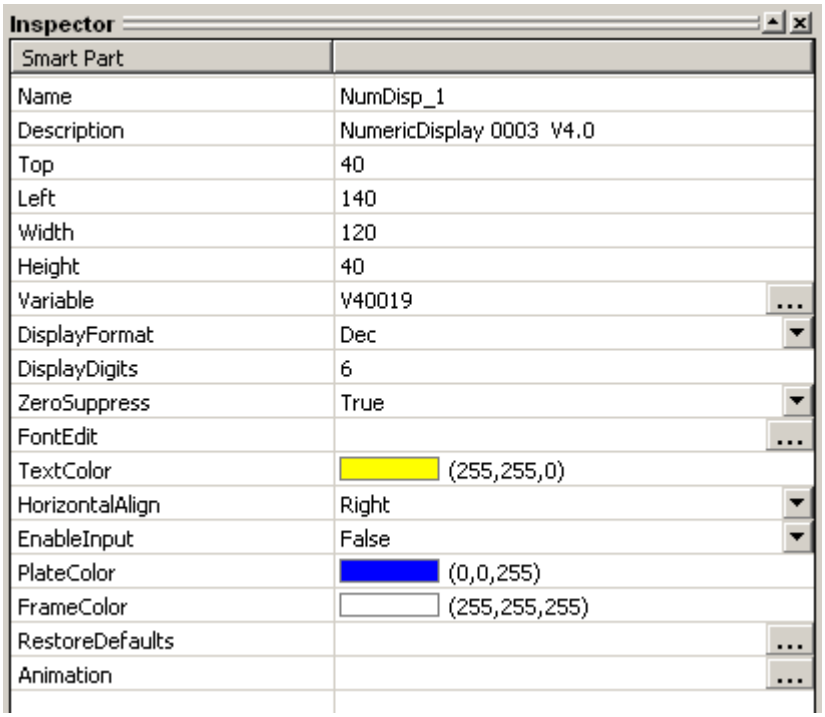


按钮 SwBitO_1 的配置如下：

Inspector

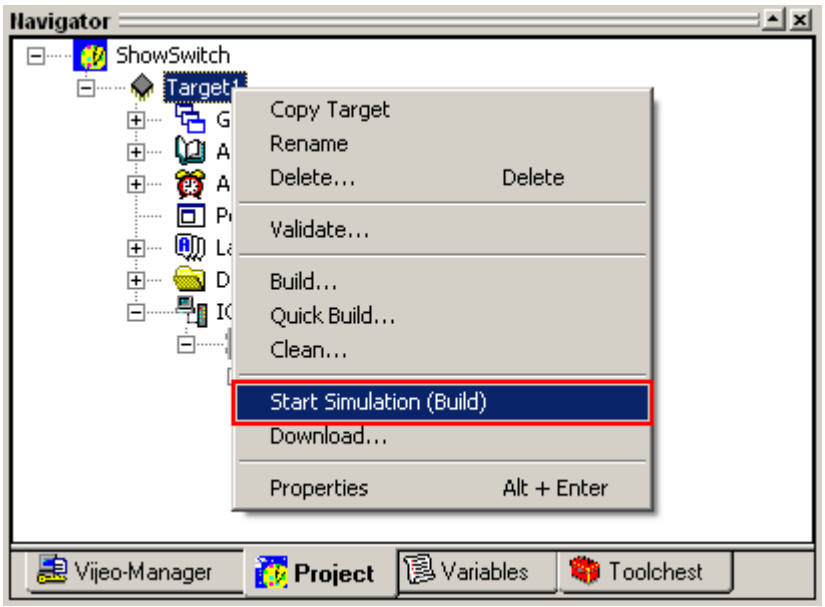
Smart Part	
Name	SwBitO_1
Description	SwBit 0015 V4.0
Top	40
Left	80
Width	40
Height	40
Operation	Toggle
ControlVariable	V00025
MonitorVariable	V00025
ForeColor(OFF)	(0,255,0)
ForeColor(ON)	(255,0,0)
Interlock	False
Buzzer	True
Pattern	1:SOLID
BackColor	(0,0,0)
FrameColor	(153,153,153)
RestoreDefaults	
Animation	

数字显示 NumDisp_1 的配置如下：

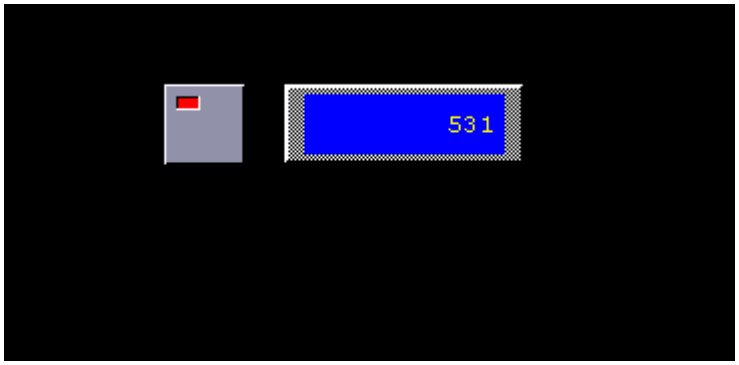


模拟测试

在 Navigator 中，右键单击 Target1，弹出菜单。



单击 Start Simulation (Build)，运行触摸屏软件模拟器：



调整试验板 CPU-EC20 (8051)上两个电位器，观察显示数字是否发生变化，从而确认数字对应的电位器是哪一个。按一次按键，调整试验板 CPU-EC20 (8051)上两个电位器，观察显示数字是否发生变化，再一次确认字对应的电位器是哪一个，看看对应关系是否发生变化。

触摸屏软件：**Vijeo-Designer Version 4.10 Trail**

http://www.plcol.com/technologies/anindex/an2002/VJDInstall.zip

PLC 程序：**Sample2_Lad.vcw**

http://www.plcol.com/technologies/anindex/an2002/Sample2_Lad.vcw

Vijeo-Designer Version 4.10 项目：**Sample2_ShowSwitch.zip**

http://www.plcol.com/technologies/anindex/an2002/Sample2_ShowSwitch.zip

使用 EXCH 指令连接另一台 PLC

使用 EXCH 指令连接另一台 PLC，意味着我们需要两台 PLC。这里的实验我们用 PLC 模拟器将电脑模拟成一台 PLC。这样对于只有一块 PLC 实验板的朋友也能进行多机通讯试验了。PLC 实验板采用 CPU-EC20 (8051)。实验板为主站，电脑模拟的 PLC 为从站。当 I0.0 按下时，主站尝试读取从站 I0.4 ~ I0.7 的值，并将从站 I0.4 ~ I0.7 的值拷贝到主站 Q0.0 ~ Q0.3 上以驱动 LED 灯。同时，主站尝试读取从站 MW0、MW2、MW4 的值，并将 MW0 的值显示在 LED 数码管上。电脑模拟器模拟的从站读取模拟量输入 AIW0 的值，并将这个值更新到 MW0 中去。

配置主站

系统块

数据块

DAT_0（DAT0）

		地址	数据类型	值	注释
	✓	MW22	UINT	6	
	✓	MW24	UINT	30	Pointer to MB30
	✓	MW26	UINT	10	
	✓	MW28	UINT	40	Pointer to MB40
			BYTE		
	✓	MB30	USINT	1	
	✓	MB31	USINT	2	
	✓	MB32	USINT	0	
	✓	MB33	USINT	4	
	✓	MB34	USINT	0	
	✓	MB35	USINT	4	
			BOOL		

根据 MB20 ~ MB28 我们知道：

- MB20 与 MB21 由 EXCH 指令维护，这里不需要我们赋值。
- MW22 初始化为 6、MW24 初始化为 30 表示需要发送的数据为 MB30 ~ MB35 这 6 个字节（CRC 校验部分由 EXCH 指令自动生成）。
- MW26 初始化为 10、MW28 初始化为 40 表示接收数据的缓冲为 MB40 ~ MB49 这 10 个字节（CRC 校验部分由 EXCH 指令自动判断）。

根据 MB30 ~ MB35 我们知道：

- MB30 初始化为 1，表示需要访问的从设备站号为 1。
- MB31 初始化为 2，即功能码为 2，执行读输入线圈状态通讯指令。
- MW32 初始化为 4，表示读取地址从 10005（I0.4）开始。
- MW34 初始化为 4，表示读取个数为 4 个 10005 ~ 10009（I0.4 ~ I0.7）。

DATA_1（DAT1）

		地址	数据类型	值	注释
	✓	MW52	UINT	6	
	✓	MW54	UINT	60	Pointer to MB60
	✓	MW56	UINT	40	
	✓	MW58	UINT	70	Pointer to MB70
			BOOL		
	✓	MB60	USINT	1	
	✓	MB61	USINT	3	
	✓	MB62	USINT	0	
	✓	MB63	USINT	8	
	✓	MB64	USINT	0	
	✓	MB65	USINT	4	
			BOOL		

根据 MB50 ~ MB58 我们知道：

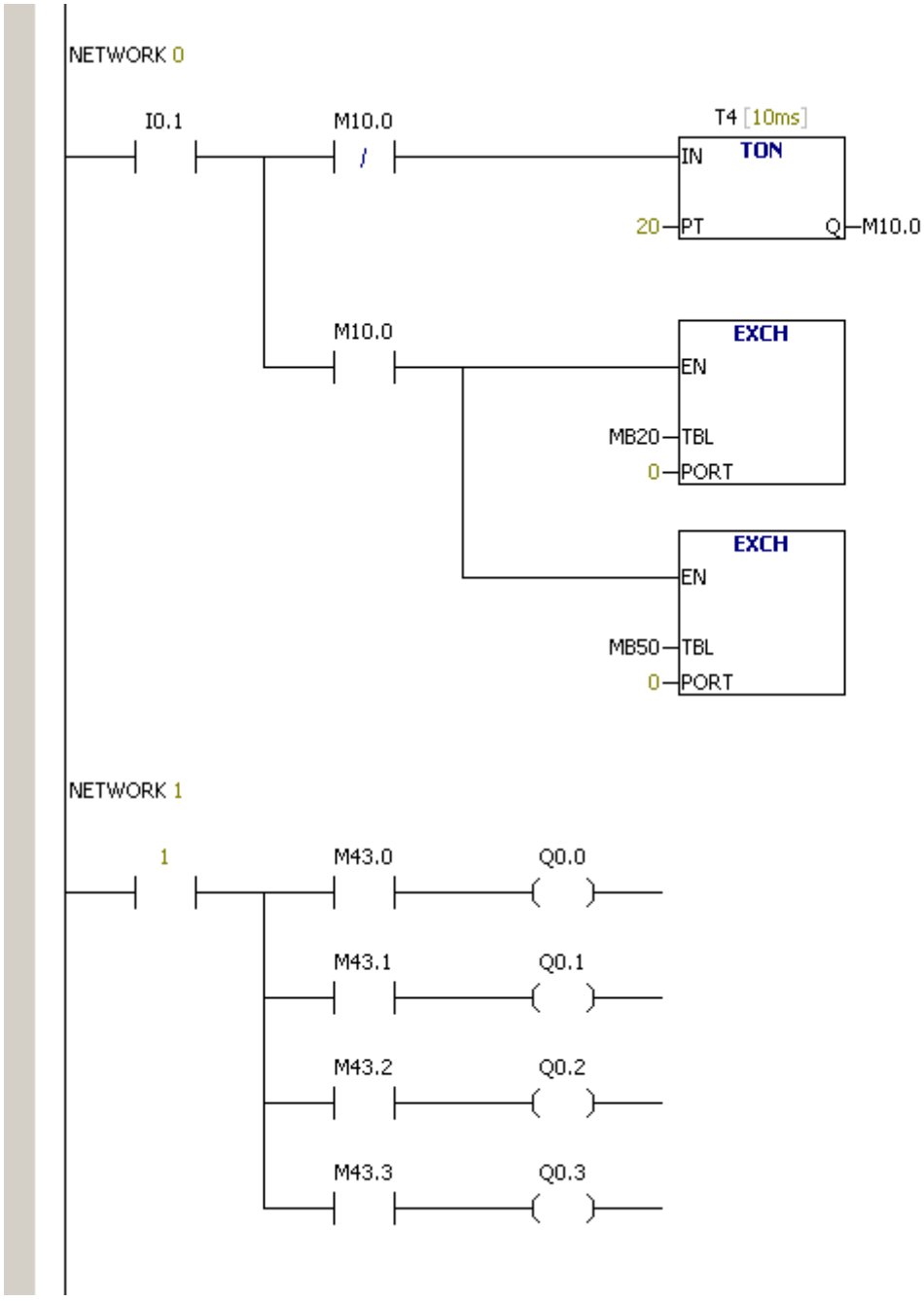
- MB50 与 MB51 由 EXCH 指令维护，这里不需要我们赋值。
- MW52 初始化为 6、MW104 初始化为 60 表示需要发送的数据为 MB60 ~ MB65 这 6 个字节（CRC 校验部分由 EXCH 指令自动生成）。
- MW56 初始化为 40、MW58 初始化为 70 表示接收数据的缓冲为 MB70 ~ MB109 这 40 个字节（CRC 校验部分由 EXCH 指令自动判断）。

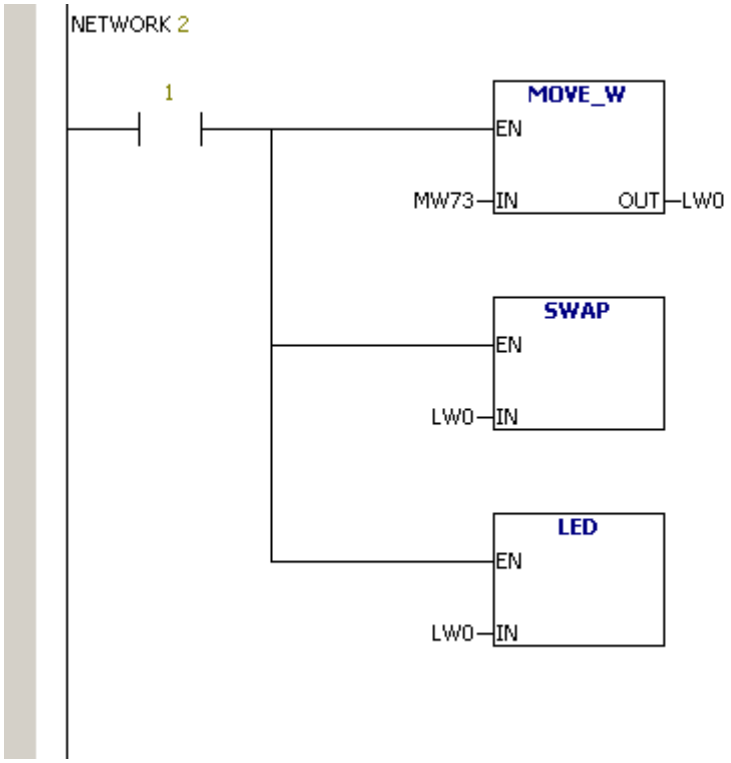
根据 MB60 ~ MB65 我们知道：

- MB60 初始化为 1，表示需要访问的从设备站号位 1。
- MB61 初始化为 3，即功能码为 3，执行读保持寄存器通讯指令。
- MW62 初始化为 8，表示读取地址从 40009（MW0）开始。
- MW64 初始化为 4，表示读取个数为 4 个，40009 ~ 40012（MW0 、MW2、MW4、MW6）。

程序块

MAIN（INT0）


NETWORK 0
I0.1
M10.0
/
T4 [10ms]
TON
IN
20
PT
Q
M10.0
M10.0
EXCH
EN
MB20
TBL
0
PORT
EXCH
EN
MB50
TBL
0
PORT
NETWORK 1
1
M43.0
Q0.0
M43.1
Q0.1
M43.2
Q0.2
M43.3
Q0.3



子程序 LED（SBR0）完成的功能就是将 LW0 的值显示到四位段码 LED 上。具体内容这里省略。

NETWORK 0 利用 100ms 定时器 T4 每 200ms 产生 1 个脉冲。在这个脉冲内，向通讯队列中加入 2 个通讯任务。这 2 个通讯任务都操作 PORT0 通讯口。第 1 个通讯任务的配置表格以 MB20 为首地址，第 2 个通讯任务的配置表格以 MB50 为首地址。

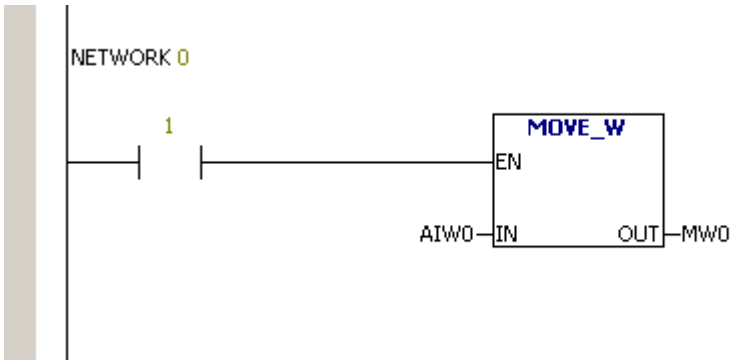
NETWORK 1 将通讯缓冲中得到的（从站）I0.4-I0.7 输出到本站的 Q0.0-Q0.3。

NETWORK 2 将通讯缓冲中得到的（从站）MW0 输出到本站的 LED 数码管上。需要注意的是：MODBUS 通讯中，高位字节（MSB）是先发送的，在缓冲中处于低地址，低位字节（LSB）是后发送的，在缓冲中处于高地址。而 EC20/EC30 的地址对起方式与之相反（小端对齐格式），需要用 SWAP 指令交换高低字节。

配置从站

由于 PLC 默认就是 MODBUS 从站，从站程序相对简单：

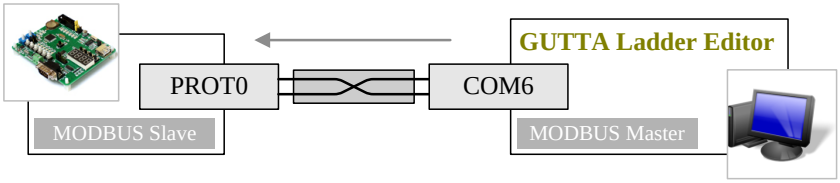
MAIN（INT0）



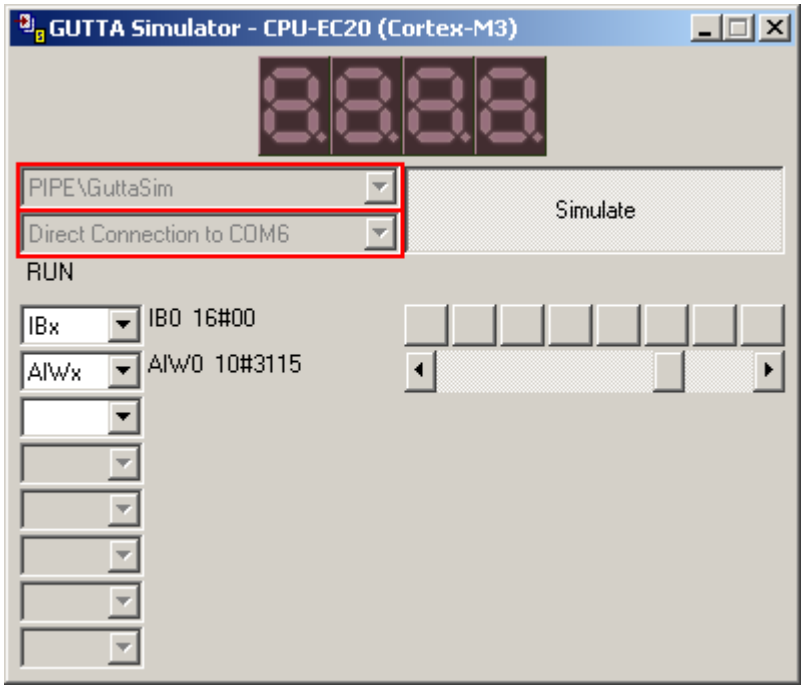
模拟测试

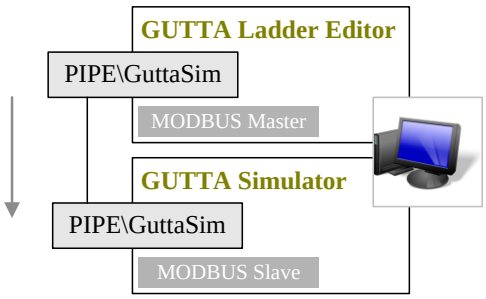
这里假设试验板 CPU-EC20 (8051)通过 USB 和电脑相连，USB 虚拟了一个串行通讯口 Prolific USB-to-Serial Comm Port (COM6)。

1. 用 USB 线连接试验板 CPU-EC20 (8051)和电脑。运行 GUTTA Ladder Editor 软件，打开主站程序 Sample3_Master_Lad.vcw，将这个程序通过 COM6 下载到试验板 CPU-EC20 (8051)，注意不要进入在线模式，以释放对 COM6 串行通讯口的占用。

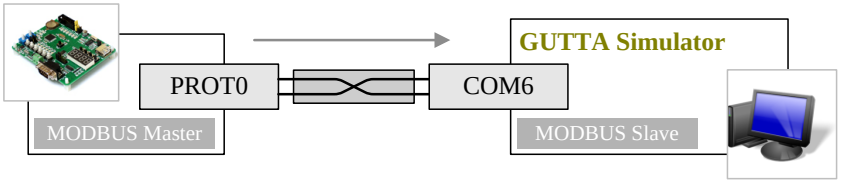


2. 运行 GUTTA Simulator 模拟器。模拟器 PORT0 通讯口配置为 PIPE\GuttaSim；PORT1 通讯口配置为 COM6；按下 Simulate 开始模拟。运行 GUTTA Ladder Editor 软件，打开从站程序 Sample3_Slave_Lad.vcw，将这个程序通过 PIPE\GuttaSim 管道下载到模拟器。





3. 按下实验板 CPU-EC20 (8051)的 I0.1，观察通讯灯 COM 是否闪烁。改变模拟器 I0.4-I0.7 和 AIW0 的值，观察实验板 CPU-EC20 (8051)数字量输出和 LED 数码管的变化。



如果松开 I0.1，实验板恢复为 MODBUS 从站（默认状态），此时可以通过 GUTTA Ladder Editor 软件再次连接到实验板，对实验板进行监控或者是编程。

AccessPort 对 COM6 通讯口的数据监控，仅供参考：

```
SUDT ACCESSPORT LOG FILE - Monitor mode


Monitor: COM6
Create Time: 2010-07-04, 03:49:18
Computer Name: WANA-001
System version: Microsoft Windows XP Professional Service Pack 3 (Build 2600)


#      Time       Function                    Data ( Hex )


1      [00000000]  IRP_MJ_CREATE                Port Opened - GuttaSimulate.exe
2      [00000000]  IOCTL_SERIAL_SET_BAUD_RATE     Baud Rate: 38400
3      [00000000]  IOCTL_SERIAL_SET_LINE_CONTROL   StopBits: 1, Parity: Even, DataBits: 8
4      [00000000]  IOCTL_SERIAL_SET_BAUD_RATE     Baud Rate: 38400
5      [00000001]  IOCTL_SERIAL_SET_LINE_CONTROL   StopBits: 1, Parity: Even, DataBits: 8
6      [00000014]  IOCTL_SERIAL_SET_BAUD_RATE     Baud Rate: 38400
7      [00000014]  IOCTL_SERIAL_SET_LINE_CONTROL   StopBits: 1, Parity: Even, DataBits: 8
8      [00000245]  IRP_MJ_READ                  Length: 0008, Data: 01 02 00 04 00 04 38 08
9      [00000248]  IRP_MJ_WRITE                 Length: 0006, Data: 01 02 01 00 A1 88
10     [00000253]  IRP_MJ_READ                  Length: 0008, Data: 01 03 00 08 00 04 C5 CB
11     [00000256]  IRP_MJ_WRITE                 Length: 0013, Data: 01 03 08 0C 2B 00 00 00 00
00 00 0E 80
12     [00000261]  IRP_MJ_READ                  Length: 0008, Data: 01 02 00 04 00 04 38 08
13     [00000264]  IRP_MJ_WRITE                 Length: 0006, Data: 01 02 01 00 A1 88
14     [00000271]  IRP_MJ_READ                  Length: 0008, Data: 01 02 00 04 00 04 38 08
15     [00000274]  IRP_MJ_WRITE                 Length: 0006, Data: 01 02 01 00 A1 88
16     [00000279]  IRP_MJ_READ                  Length: 0008, Data: 01 03 00 08 00 04 C5 CB
17     [00000282]  IRP_MJ_WRITE                 Length: 0013, Data: 01 03 08 0C 2B 00 00 00 00
00 00 0E 80
18     [00000287]  IRP_MJ_READ                  Length: 0008, Data: 01 02 00 04 00 04 38 08
19     [00000290]  IRP_MJ_WRITE                 Length: 0006, Data: 01 02 01 00 A1 88
```

主站 PLC 程序：**Sample3_Master_Lad.vcw**

http://www.plcol.com/technologies/anindex/an2002/Sample3_Master_Lad.vcw

从站 PLC 程序：**Sample3_Slave_Lad.vcw**

http://www.plcol.com/technologies/anindex/an2002/Sample3_Slave_Lad.vcw