MODBUS Organization
MODBUS Master Series
MODBUS Serial Master Driver

2023/10/12　　バージョン　GX8 Design Studio　V1.4 以上

CONTENTS

当社(株)ミスミのTouch Operation Panel(ミスミ GX8) Seriesをご使用いただきまして誠にありがとうございます。本マニュアルを読み、「GX8–外部機器」の接続方法と手順をご確認ください。

1. システム構成

本ドライバは、MODBUS OrganizationのMODBUS Protocolのなか、Serial Master Driverです。
外部機器（MODBUS Slave Protocolサポート）に応じて、ドライバの「コマンドコード」、「プロトコルのフレーム形式」などを別途設定する必要があります。
この場合、通信方式に応じた詳細な設定を外部機器側に合わせて設定してください。
本ドライバがサポートする外部機器のシステム構成は以下の通りです。

シリーズ	CPU	Link I/F	通信方式	システム設定	ケーブル
MODBUS Slave Device			RS-232C	3. GX8通信設定 4. 外部機器の設定	5. ケーブル表
			RS-422 (4 wire)		
			RS-485 (2 wire)		

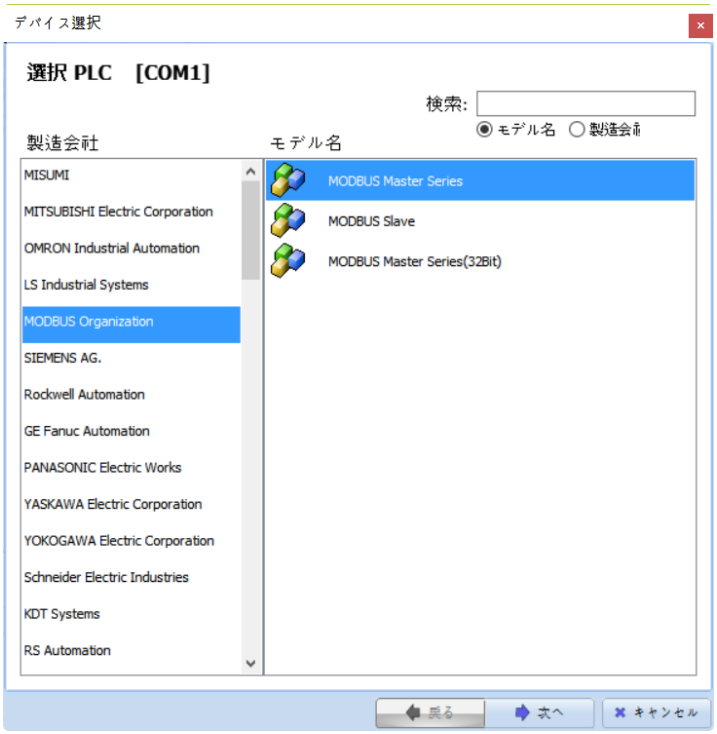
■接続構成

・ 1：1（GX8 1台と外部機器1台）接続 – RS232C/422/485通信で可能な構成です。

・ 1：N（GX8 1台と外部機器数台）接続 - RS422/485通信で可能な構成です。

2. 外部機器の選択

■ GX8モデル及びポートを選択した後、外部機器を選択します。



設定事項		内容
GX8	モデル	GX8のディスプレイとプロセスを確認し、タッチモデルを選択します。
外部機器	メーカー	GX8と接続する外部機器のメーカー「MODBUS Organization」を選択します。
	PLC	GX8と接続する外部機器を選択します。 モデル: MODBUS Master Series / インターフェイス: Serial / プロトコル: ユーザー設定 サポートしているプロトコル: MODBUS RTU, MODBUS ASCII 接続したい外部機器が、システムの構成可能な機種であることを1項のシステム構成で確認してください。

3. GX8通信設定

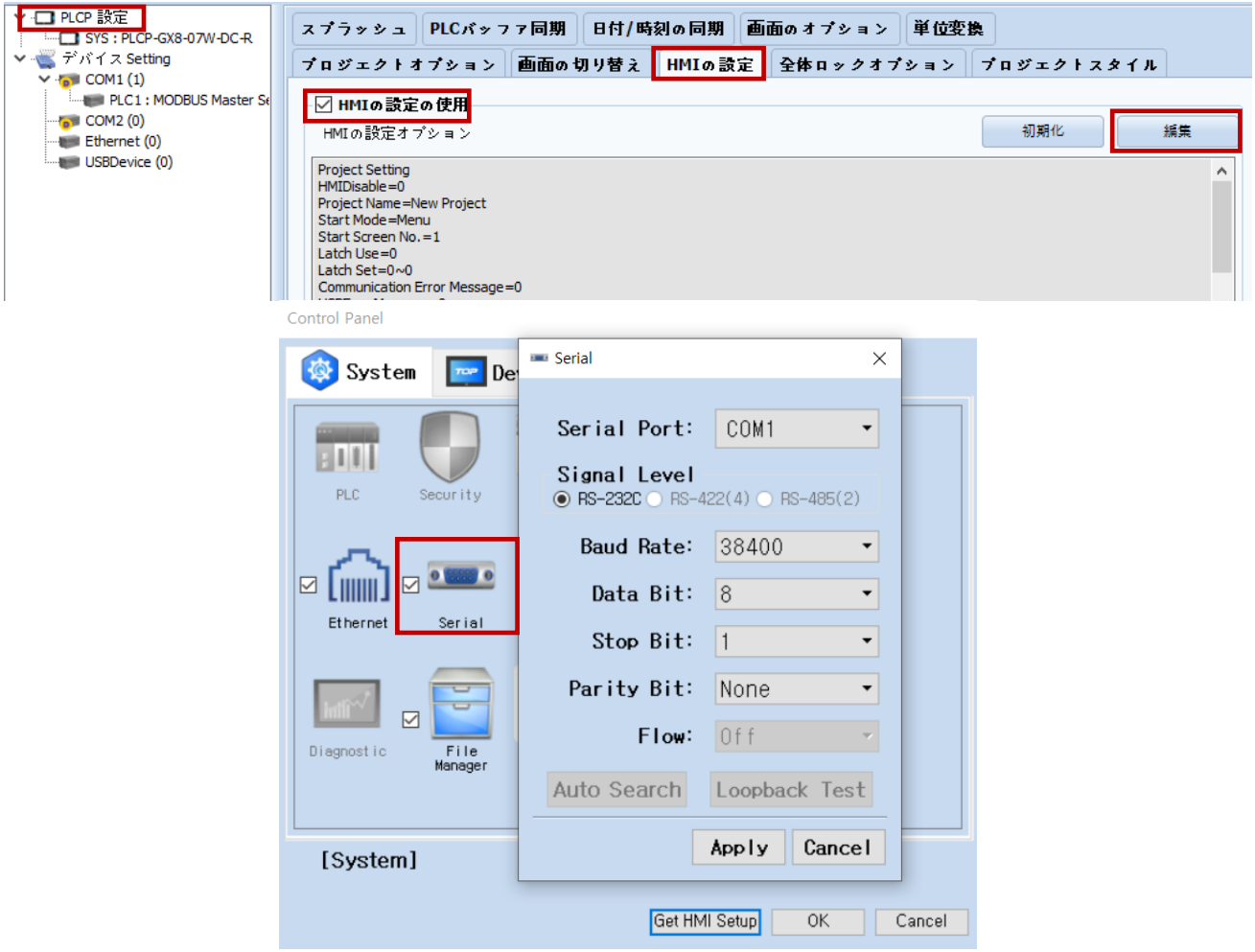
通信設定はGX8 Design Studio、またはGX8メインメニューから設定可能です。通信設定は外部機器と同一に設定する必要があります。

3.1 GX8 Design Studioでの通信設定

(1) 通信インタフェースの設定

■[プロジェクト>プロジェクトのプロパティ>GX8設定]→[プロジェクトオプション>「HMIの設定使用」チェック>編集>シリアル]

– GX8通信インタフェースをGX8 Design Studioで設定します。



項目	GX8	外部機器	備 考
信号レベル（ポート）	RS-232C RS-422/485	RS-232C RS-422/485	
ボーレート	38400		
データビット	8		
ストップビット	1		
パリティビット	無		

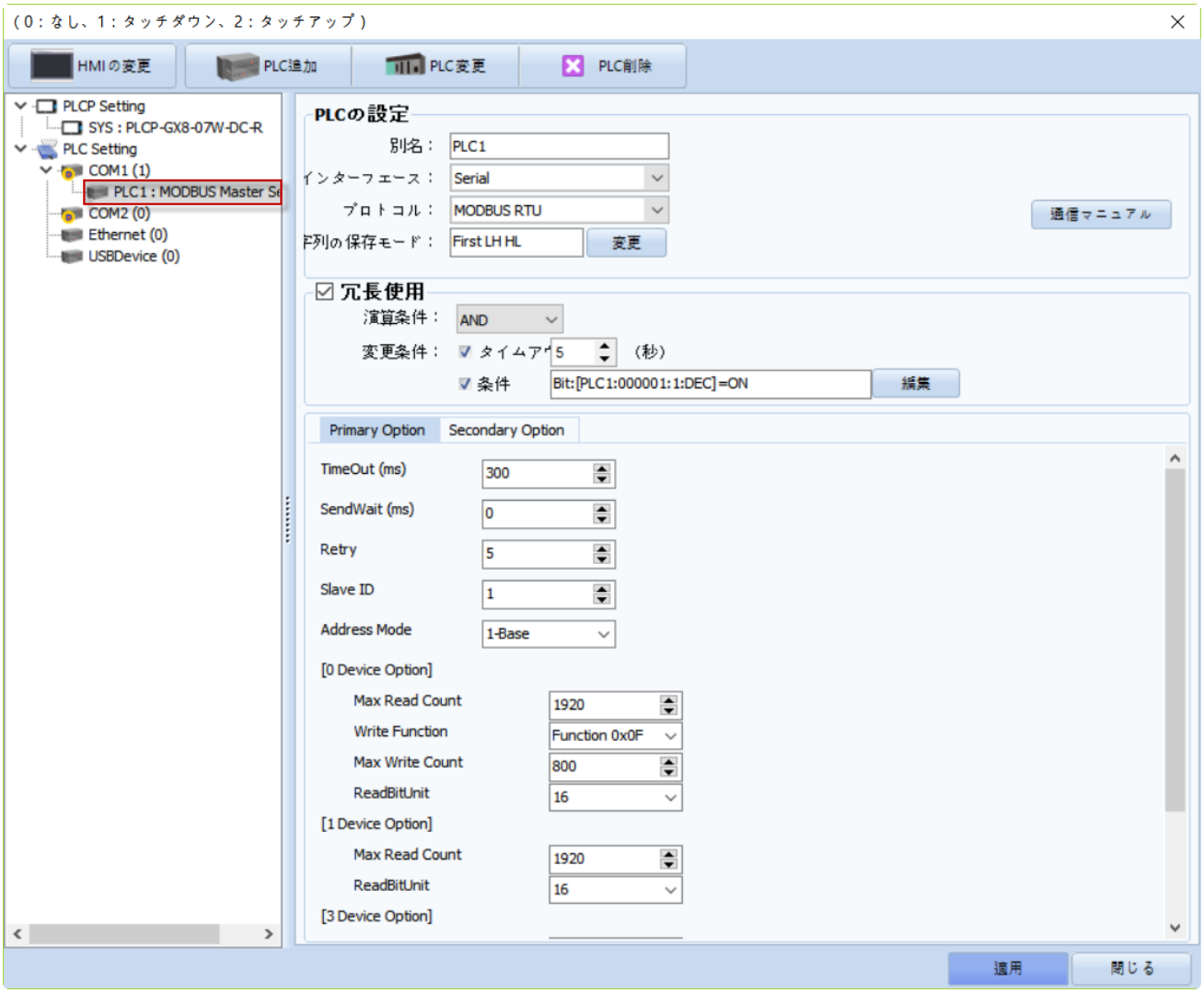
※上記の内容は、当社が推奨する設定例です。

項目	説明
信号レベル	GX8 - 外部機器間のシリアル通信方式を選択します。
ボーレート	GX8 - 外部機器間のシリアル通信速度を選択します。
データビット	GX8 - 外部機器間のシリアル通信のデータビットを選択します。
ストップビット	GX8 - 外部機器間のシリアル通信ストップビットを選択します。
パリティビット	GX8 - 外部機器間のシリアル通信のパリティビットを確認する方法を選択します。

(2) 通信オプションの設定

■[プロジェクト > プロジェクトのプロパティ > PLC設定 > COM1 > PLC1 : MODBUS Master Series]

– MODBUS Serial Master 通信ドライバのオプションをGX8 Design Studioで設定します。



項目	設定	備考
インタフェース	Serialを選択します。	「2. 外部機器の選択」参考
プロトコル	GX8 - 外部機器間の通信プロトコルを選択します。 MODBUS ASCIIとMODBUS RTUの2種類があります。	
文字列の保存モード	文字列を使用時に文字列の配列を指定します。	
冗長使用	二種類のオプションを使うときに使用します。 変更条件は：Primary – Secondary変更条件を設定します。 演算条件で変更条件に対する演算条件を設定します。 AND : チェックされた変更条件が全て満足すると Primary–Secondary 変更 OR : チェックされた変更条件の１つでも満足するとPrimary–Secondary 変更	
TimeOut (ms)	GX8の外部機器からの応答を待つ時間を設定します。	
SendWait (ms)	GX8の外部機器からの応答を受信し、次のコマンド要求の送信までの待機時間を設定します。	
Retry	通信障害時の再試行の回数を設定します。	
Slave Station Num	外部機器の局番を入力します。	
Address Mode	アドレスの方式を選択します。 1-base : 装備のメモリアドレスが１から始まる。登録されたアドレス-1にデータ要請 0-base : 装備のメモリアドレスが0から始まる。登録されたアドレス-データ要請	
Address Notation	アドレス表記方式を選択します。	
[0 Device Option]	Coil (R/W)	
Max Read Count	Coilを読み取り要請時１回に要請できる最大数を設定します。	*注1) *注2)
Write Function	Coil書き込み要請命令語を設定します。	*注3)

	0x05 : Force Single Coil (1ビット単位で書き込み。ビット単位動作のみ可能) 0x0F : Force Multiple Coils (16ビット単位で書き込み) Auto : データ数によって0x05/0x0F要請	
Max Write Count	Coilの書き込み要請時最大要請数を設定します。	*注2)
Read Bit Unit	Coil読み取り要請時要請するビット数を設定します。 設定値が16で画面に続くアドレスが登録された場合１回に最大"Max Read Count"数分データを要請します。	
[1 Device Option]	Discrete Input (READ ONLY)	
Max Read Count	Discrete Input読み取り要請時１回に要請できる最大数を設定します。	*注1) *注2)
Read Bit Unit	Discrete Input読み取り要請時要請するビット数を設定します。 設定値が16でで画面に続くアドレスが登録された場合一回に最大"Max Read Count"数分データを要請します。	
[3 Device Option]	Input Register(READ ONLY)	
Max Read Count	Input Register読み取り要請時１回に要請できる最大数を設定します。	*注1) *注2)
[4 Device Option]	Holding Register(R/W)	
Max Read Count	Holding Register読み取り要請時１回に要請できる最大数を設定します。	*注1)
Write Function	Holding Register書き込み要請命令語を設定します。 0x06 : Preset Single Register (1個書き込み) 0x10 : Preset Multiple Registers (N個書き込み) Auto : データ数によって0x06/0x10に要請	*注3)
Max Write Count	命令語を0x10にHolding Registerデータ書き込み要請時１回に要請できる最大数を設定します。	*注2)

*注1)
- 各デバイスのMax Read Countは画面に登録されたアドレスたちが連続していない通信を南海することではなく１回に要請するアドレス範囲としても使用されます。
例1)画面に数字オブジェクトで400001, 400002, 400003, 400004, 400005, 400120を登録し4デバイスのMax Read Countを120に設定する場合400001から400120までを連続したアドレスに仮定し400001から120ワードを一回の要請で読み取ります。
例２) 画面に数字オブジェクトで400001, 400002, 400003, 400004, 400005, 400120を登録し4デバイスのMax Read Countを3に設定する場合400001から400003まで3ワード、400004から400005まで２ワード、400120の１ワードとして３回の要請でデータを読み取ります。
例３) 画面に数字オブジェクトで400001, 400010, 400011, 400021, 400031, 400041を登録し4デバイスのMax Read Countを10に設定した場合400001から400010まで10ワード400011 1ワード、 400021 1ワード、400031 1ワード、 400041 1ワードとして５回の要請でデータを読み取ります。

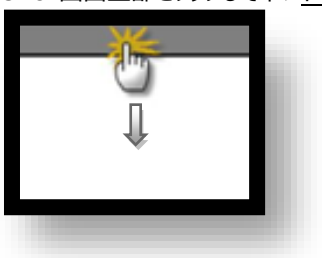
*注2)
- 外部装置のマニュアルを参考し登録したアドレスか一回に何個のデータの読み取り/書き込みが可能か確認してください。
外部装置が支援している範囲より大きく設定する場合通信が正常的に行割れられません。
例)外部装置のHolding Register(4デバイス)が１回の通信に最大１０ワードのみ応答可能な場合、GX8の通信設定の4デバイスMax Read Countを外部装置に合わせ10に設定してください。

*注3)
- 外部装置のマニュアルを参考し支援する書き込み命令語に合わせて設定してください。
支援していない書き込み命令語を設定するとデータの書き込み動作が行われません。

3.2 GX8での通信設定

※「3.1 GX8 Design Studioでの通信設定」の「HMIの設定を使用する」をチェックしていない場合の設定方法です。

■GX8 画面上部をタッチして下にドラッグします。ポップアップウィンドウの「EXIT」をタッチして、メイン画面に移動します。



(1) 通信インタフェースの設定

■[メイン画面 > コントロールパネル > シリアル]

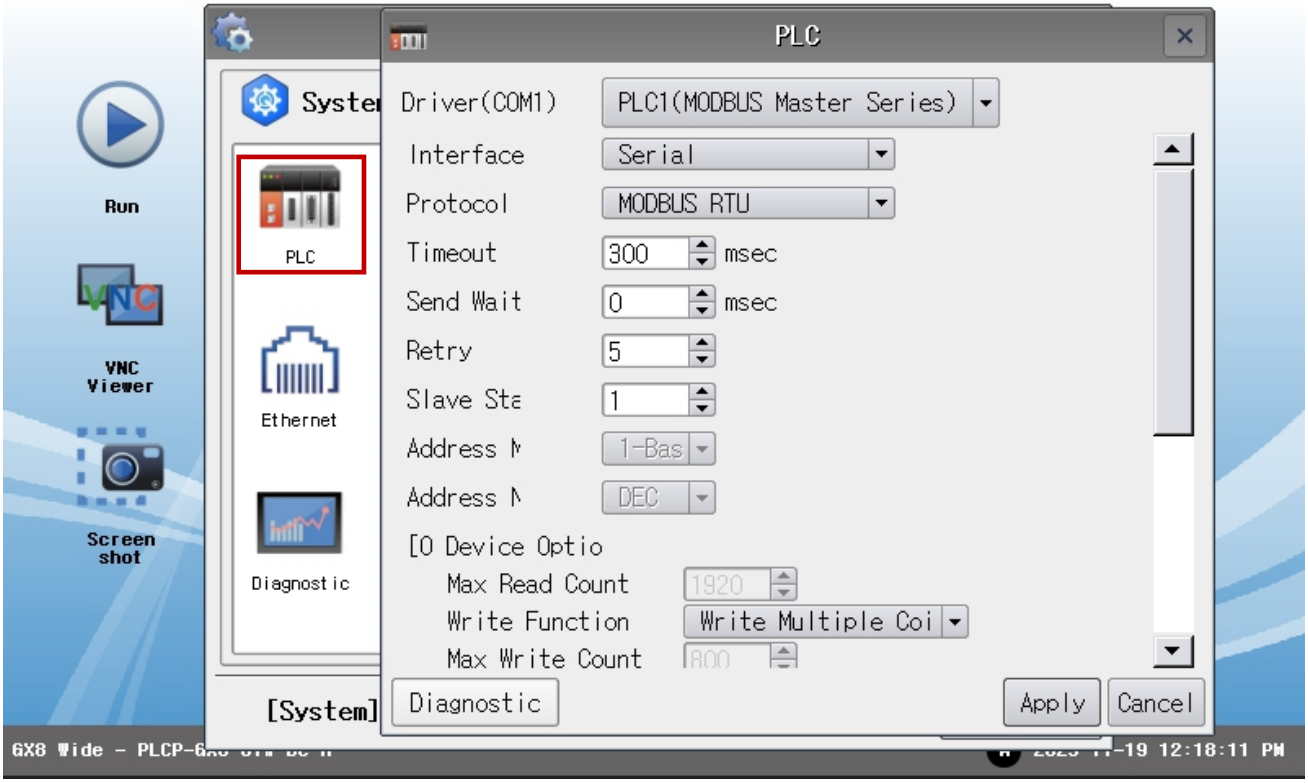
項目	GX8	外部機器	備 考
信号レベル（ポート）	RS-232C RS-422/485	RS-232C RS-422/485	
ボーレート	38400		
データビット	8		
ストップビット	1		
パリティビット	無		

※上記の内容は、当社が推奨する設定例です。

項目	説明
信号レベル	GX8 - 外部機器間のシリアル通信方式を選択します。
ボーレート	GX8 - 外部機器間のシリアル通信速度を選択します。
データビット	GX8 - 外部機器間のシリアル通信のデータビットを選択します。
ストップビット	GX8 - 外部機器間のシリアル通信ストップビットを選択します。
パリティビット	GX8 - 外部機器間のシリアル通信のパリティビットを確認する方法を選択します。

(2) 通信オプションの設定

■[メイン画面 > コントロールパネル > PLC]



項目	設定	備考
インタフェース	Serialを選択します。	「2. 外部機器の選択」参考
プロトコル	GX8 - 外部機器間の通信プロトコルを選択します。 MODBUS ASCIIとMODBUS RTUの2種類があります。	
文字列の保存モード	文字列を使用時に文字列の配列を指定します。	
冗長使用	二種類のオプションを使うときに使用します。 変更条件は：Primary – Secondary変更条件を設定します。 演算条件で変更条件に対する演算条件を設定します。 AND ： チェックされた変更条件が全て満足すると Primary–Secondary 変更 OR ： チェックされた変更条件の１つでも満足するとPrimary–Secondary 変更	
TimeOut (ms)	GX8の外部機器からの応答を待つ時間を設定します。	
SendWait (ms)	GX8の外部機器からの応答を受信し、次のコマンド要求の送信までの待機時間を設定します。	
Retry	通信障害時の再試行の回数を設定します。	
Slave Station Num	外部機器の局番を入力します。	
Address Mode	アドレスの方式を選択します。 1-base ： 装備のメモリアドレスが１から始まる。登録されたアドレス-1にデータ要請 0-base ： 装備のメモリアドレスが0から始まる。登録されたアドレス-データ要請	
Address Notation	アドレス表記方式を選択します。	
[0 Device Option]	Coil (R/W)	
Max Read Count	Coilを読み取り要請時１回に要請できる最大数を設定します。	*注1) *注2)
Write Function	Coil書き込み要請命令語を設定します。 0x05 ： Force Single Coil (1ビット単位で書き込み。ビット単位動作のみ可能) 0x0F ： Force Multiple Coils (16ビット単位で書き込み) Auto ： データ数によって0x05/0x0F要請	*注3)
Max Write Count	Coilの書き込み要請時最大要請数を設定します。	*注2)
Read Bit Unit	Coil読み取り要請時要請するビット数を設定します。 設定値が16で画面に続くアドレスが登録された場合１回に最大"Max Read Count"数分データを要請します。	
[1 Device Option]	Discrete Input (READ ONLY)	

Max Read Count	Discrete Input読み取り要請時１回に要請できる最大数を設定します。	*注1) *注2)
Read Bit Unit	Discrete Input読み取り要請時要請するビット数を設定します。 設定値が16でで画面に続くアドレスが登録された場合一回に最大"Max Read Count"数分データを要請します。	
[3 Device Option]	Input Register(READ ONLY)	
Max Read Count	Input Register読み取り要請時１回に要請できる最大数を設定します。	*注1) *注2)
[4 Device Option]	Holding Register(R/W)	
Max Read Count	Holding Register読み取り要請時１回に要請できる最大数を設定します。	*注1)
Write Function	Holding Register書き込み要請命令語を設定します。 0x06 : Preset Single Register (1個書き込み) 0x10 : Preset Multiple Registers (N個書き込み) Auto : データ数によって0x06/0x10に要請	*注3)
Max Write Count	命令語を0x10にHolding Registerデータ書き込み要請時１回に要請できる最大数を設定します。	*注2)

*注1)
- 各デバイスのMax Read Countは画面に登録されたアドレスたちが連続していない通信を南海することではなく１回に要請するアドレス範囲としても使用されます。
例1)画面に数字オブジェクトで400001, 400002, 400003, 400004, 400005, 400120を登録し4デバイスのMax Read Countを120に設定する場合400001から400120までを連続したアドレスに仮定し400001から120ワードを一回の要請で読み取ります。
例２) 画面に数字オブジェクトで400001, 400002, 400003, 400004, 400005, 400120を登録し4デバイスのMax Read Countを3に設定する場合400001から400003まで3ワード、400004から400005まで２ワード、400120の１ワードとして３回の要請でデータを読み取ります。
例３) 画面に数字オブジェクトで400001, 400010, 400011, 400021, 400031, 400041を登録し4デバイスのMax Read Countを10に設定した場合400001から400010まで10ワード400011 1ワード、 400021 1ワード、400031 1ワード、 400041 1ワードとして５回の要請でデータを読み取ります。

*注2)
- 外部装置のマニュアルを参考し登録したアドレスか一回に何個のデータの読み取り/書き込みが可能か確認してください。
外部装置が支援している範囲より大きく設定する場合通信が正常的に行割れられません。
例)外部装置のHolding Register(4デバイス)が１回の通信に最大１０ワードのみ応答可能な場合、GX8の通信設定の4デバイスMax Read Countを外部装置に合わせ10に設定してください。

*注3)
- 外部装置のマニュアルを参考し支援する書き込み命令語に合わせて設定してください。
支援していない書き込み命令語を設定するとデータの書き込み動作が行われません。

3.3 通信診断

■GX8 - 外部機器間のインタフェースの設定の状態を確認

- GX8画面上部をタッチして下にドラッグ。ポップアップウィンドウの「EXIT」をタッチして、メイン画面に移動する
- [コントロールパネル>シリアル]で使用するポート（COM1/COM2/COM3）の設定が、外部装置の設定内容と同じであることを確認する。

■ポートの通信異常の有無の診断

- [コントロールパネル> PLC]で「通信診断」をタッチする。
- 画面上にDiagnosticsダイアログボックスがポップアップし、診断の状態を表示する。

OK	**通信設定の正常**
Time Out Error	**通信設定の異常** -ケーブルとGX8、外部機器の設定状態を確認する。**（参照：通信診断シート）**

■通信診断シート

-外部端末との通信接続に問題がある場合は、以下のシートの設定内容を確認してください。

項目	内容		確認		参考
システム構成	システムの接続方法		OK	NG	1．システム構成
	接続ケーブルの名称		OK	NG	
GX8	バージョン情報		OK	NG	2．外部機器の選択 3．GX8通信設定
	使用ポート		OK	NG	
	ドライバ名称		OK	NG	
	その他の詳細設定情報		OK	NG	
	相手局番	プロジェクトの設定	OK	NG	
		通信診断	OK	NG	
	シリアルパラメータ	転送速度	OK	NG	
		データビット	OK	NG	
		ストップビット	OK	NG	
		パリティビット	OK	NG	
外部機器	CPUの名称		OK	NG	4．外部機器の設定
	通信ポートの名称（モジュール名）		OK	NG	
	プロトコル（モード）		OK	NG	
	設定局番		OK	NG	
	その他の詳細設定情報		OK	NG	
	シリアルパラメータ	転送速度	OK	NG	
		データビット	OK	NG	
		ストップビット	OK	NG	
		パリティビット	OK	NG	
	アドレス範囲チェック		OK	NG	6．サポートアドレス (詳細については、PLCメーカーのマニュアルを参照してください。)

4. 外部機器の設定

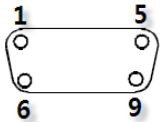
外部機器のユーザーマニュアルを参照し、外部機器I / Fに「MODBUS Serial Slave Driver」を設定してください。

- Protocol Frame形式でRTU/ ASCIIモード選定に注意してください。
- 外部機器側のアドレスマップの内容を確認し、その内容に応じて通信アドレスを使用してください

5. ケーブル表

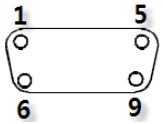
本ChapterはGX8と該当機器の間、通常の通信のためのケーブル図を紹介します。
（本項で説明されているケーブルの図は、外部機器メーカーの推奨事項とは異なる場合があります）

■ **RS-232C**（1:1接続）

COM1 / COM2			ケーブル接続	PLC	
ピン配列*注1)	信号名	ピン番号		信号名	
 ケーブルコネクタ前面基準 D-SUB 9 Pin male(数、凸)	CD	1			
	RD	2		SD	
	SD	3		RD	
	DTR	4		DTR	
	SG	5		SG	
	DSR	6		DSR	
	RTS	7		RTS	
	CTS	8		CTS	
		9			

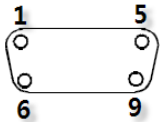
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

■ **RS-422**（1:1接続）

COM1 / COM2			ケーブル接続	PLC	
ピン配列*注1)	信号名	ピン番号		信号名	
 ケーブルコネクタ前面基準 D-SUB 9 Pin male(数、凸)	RDA(+)	1		SDA(+)	
		2		SDB(-)	
		3		RDA(+)	
	RDB(-)	4		RDB(-)	
	SG	5		SG	
	SDA(+)	6			
		7			
		8			
	SDB(-)	9			

*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

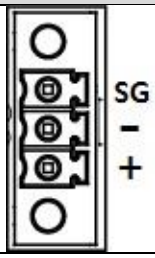
■ **RS-485**（1:1接続）

COM1 / COM2			ケーブル接続	PLC	
ピン配列*注1)	信号名	ピン番号		信号名	
 ケーブルコネクタ前面基準 D-SUB 9 Pin male(数、凸)	RDA(+)	1		+	
		2		-	
		3			
	RDB(-)	4			
	SG	5			
	SDA(+)	6			
		7			
		8			
	SDB(-)	9			

*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

☞ 次ページに続く

■ **RS-485** （1:1接続）

COM3		ケーブル接続	PLC	
ピン配列	信号名		信号名	
	+		SDA(+)	
	-		SDB(-)	
	SG		RDA(+)	
			RDB(-)	
			SG	

■ **RS-422** （1:N接続） – 1:1接続を参考にして、以下の方法で接続してください。

GX8	ケーブル接続と信号方向	PLC	ケーブル接続と信号方向	PLC
信号名		信号名		信号名
RDA(+)		SDA(+)		SDA(+)
RDB(-)		SDB(-)		SDB(-)
SDA(+)		RDA(+)		RDA(+)
SDB(-)		RDB(-)		RDB(-)
SG		SG		SG

■ **RS-485** （1:N/N:1接続）1：1接続を参考にして、以下の方法で接続してください。

GX8	ケーブル接続と信号方向	PLC	ケーブル接続と信号方向	PLC
信号名		信号名		信号名
RDA(+)		+		+
RDB(-)		-		-
SDA(+)				
SDB(-)				
SG				

6. サポートアドレス

GX8で使用可能な機器は、以下の通りです。

CPUモジュールシリーズ/タイプに応じて、機器の範囲（アドレス）の差があることがあります。GX8シリーズは、外部機器シリーズが使用する最大アドレス範囲をサポートします。使用する機器がサポートしているアドレス範囲を超えないように各CPUモジュールユーザーマニュアルを参照して/注意してください。

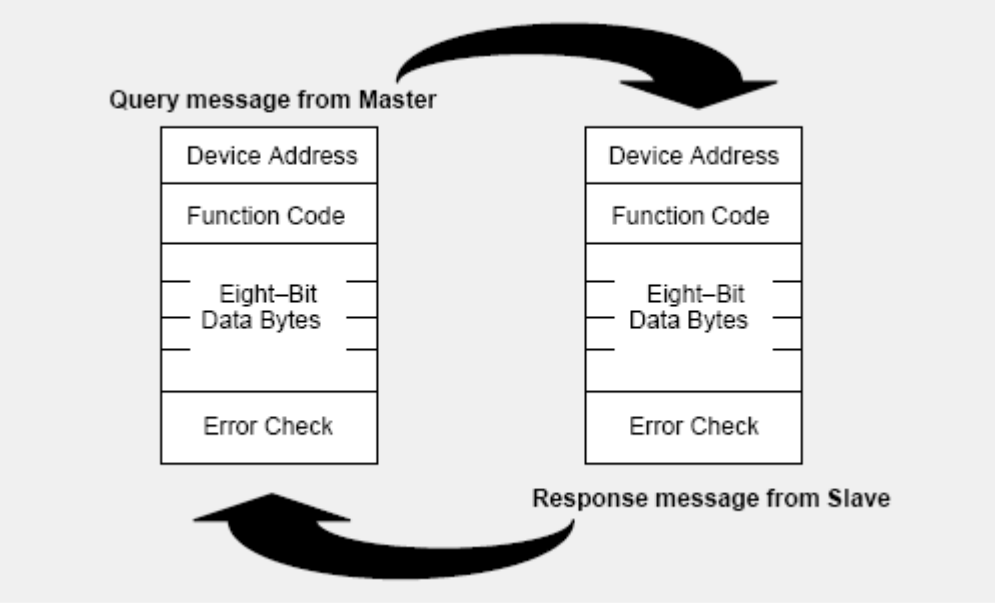
	Bit Address	Word Address	32 bits	Remarks
Coil	000001 – 065536	000001 – 065521	L/H	
Discrete Input	100001 – 165536	100001 – 165521		*注1)
Input Register	300001.00 – 365536.15	300001 – 365536		*注1)
Holding Register	400001.00 – 465536.15	400001 – 465536		

***注1）**書き込み不可（読み取り専用）

Appendix A. Standard MODBUS Protocol

本機の「MODBUS Serial Master Driver」がサポートしているMODBUSプロトコルコマンドと機器について説明します。

At the message level, the MODBUS protocol still applies the master–slave principle even though the network communication method is peer–GX8eer. If a controller originates a message, it does so as a master device, and expects a response from a slave device. Similarly, when a controller receives a message it constructs a slave response and returns it to the originating controller.



The Query: The function code in the query tells the addressed slave device what kind of action GX8erform. The data bytes contain any additional information that the slave will need GX8erform the function. For example, function code 03 will query the slave to read holding registers and respond with their contents. The data field must contain the information telling the slave which register to start at and how many registers to read. The error check field provides a method for the slave to validate the integrity of the message contents.

The Response: If the slave makes a normal response, the function code in the response is an echo of the function code in the query. The data bytes contain the data collected by the slave, such as register values or status. If an error occurs, the function code is modified to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the message contents are valid.

Read Single Coil : 01

MASTER機器のSlave機器側（局番：17番）の「000020-000056 Coil」データを読み取る例により、「01」コマンドフレームを説明します。

■ RTU Mode

（Master → Slave： 要求フレーム）

Comment	Slave 局番	コマンド	先頭機器		機器点数		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	01	00	13	00	25	--	--

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	データ数(byte)	データ					チェックコード（CRC）	
				Coils 27~20	Coils 35~28	Coils	Coils	Coils	L	H
Hex	11	01	05	CD	6B	B2	0E	1B	--	--

■ Coils データの状態

Coils	27	26	25	24	23	22	21	20
on/off	1	1	0	0	1	1	0	1
Coils	35	34	33	32	31	30	29	28
on/off	0	1	1	0	1	0	1	1
Coils	43	42	41	40	39	38	37	36
on/off	1	0	1	1	0	0	1	0
Coils	51	50	49	48	47	46	45	44
on/off	0	0	0	0	1	1	1	0
Coils	59	58	57	56	55	54	53	52
on/off	–	–	–	1	1	0	1	1

0: OFF / 1:ON

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				機器点数				チェックコード（LRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	1	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	31	33	30	30	32	35	--	--	0D	0A

（Slave → Master： 応答フレーム）

Comment	Header	Slave 局番		コマンド		データ数(byte)		データ										チェックコード（LRC）		Tail	
								Coils 27~20		Coils 35~28		Coils 43~36		Coils 51~44		Coils 56~52					
								H	L	H	L	H	L	H	L	H	L	L	H		
ASCII	:	1	1	0	1	0	5	C	D	6	B	B	2	0	E	1	B			CR	LF
Hex	3A	31	31	30	31	30	35	43	44	36	42	42	32	30	45	31	42	--	--	0D	0A

Force Single Coil : 05

MASTER機器のSlave機器側のCoil **0**00173にFORCE「ON」にした例により、「05」コマンドフレームを説明します。

■ RTU Mode

（Master → Slave： 要求フレーム ）

Comment	Slave 局番	コマンド	先頭機器		Force data		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	05	00	AC	FF	00	--	--

■ Force Data

	High	Low
Force ON	FF_H	00_H
Force OFF	00_H	00_H

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	先頭機器		Force data		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	05	00	AC	FF	00	--	--

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				Force data				チェックコード（LRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	5	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	41	43	45	45	30	30	--	--	0D	0A

（Slave → Master： 応答フレーム）

comment	Header	Slave 局番		コマンド		先頭機器				Force data				チェックコード（LRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	5	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	41	43	45	45	30	30	--	--	0D	0A

Read Input Status : 02

MASTER機器のSlave機器側（局番：17番）の「100197-100218 Input」データを読み取る例により、「02」コマンドフレームを説明します。

■ RTU Mode

（Master → Slave： 要求フレーム ）

Comment	Slave 局番	コマンド	先頭機器		機器点数		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	02	00	C4	00	16	--	--

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	データ数(byte)	データ(Inputs)			チェックコード（CRC）	
				10204~1019	10212~1020	10218~1021	L	H
Hex	11	02	03	A C	D B	35	--	--

■ Coils データの状態

Coils	204	203	202	201	200	199	198	197
on/off	1	0	1	0	1	1	0	0
Coils	212	211	210	209	208	207	206	205
on/off	1	1	0	1	1	0	1	1
Coils	220	219	218	217	216	215	214	213
on/off	–	–	1	1	0	1	0	1

0: OFF / 1:ON

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				機器点数				チェックコード（LRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	2	0	0	C	4	0	0	1	6			CR	LF
Hex	3A	31	31	30	32	30	30	43	34	30	30	31	36	--	--	0D	0A

（Slave → Master： 応答フレーム）

Comment	Header	Slave 局番		コマンド		データ数(byte)		データ(Inputs)						チェックコード（LRC）		Tail	
								7 10204~1019		5 10212~1020		3 10218~1021					
								H	L	H	L	H	L	L	H		
ASCII	:	1	1	0	2	0	3	A	C	D	B	3	5			CR	LF
Hex	3A	31	31	30	31	30	35	41	43	44	42	33	35	--	--	0D	0A

Read Input Registers : 04

MASTER機器のSlave機器側（局番：17番）の「300009 Register」データを読み取る例により、「03」コマンドフレームを説明します。

■ RTU Mode

（Master → Slave： 要求フレーム ）

Comment	Slave 局番	コマンド	先頭機器		機器点数 (Word Count)		チェックコード (CRC)	
			H	L	H	L	L	H
Hex	11	04	00	08	00	01	--	--

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	データ数(byte)	データ 30009 Register		チェックコード (CRC)	
				H	L	L	H
Hex	11	04	02	00	0A	--	--

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				機器点数 (Word)				チェックコード (LRC)		Tail	
		H	L	H	L	H	-	-	L	H	-	-	L	L	H		
ASCII	:	1	1	0	1	0	0	0	8	0	0	0	1			CR	LF
Hex	3A	31	31	30	31	30	30	30	38	30	30	30	31	--	--	0D	0A

（Slave → Master： 応答フレーム）

Comment	Header	Slave 局番		コマンド		データ 数(byte)		データ 40108 Register				チェックコード (LRC)		Tail	
								H	-	-	L	L	H		
ASCII	:	1	1	0	4	0	2	0	0	0	A			CR	LF
Hex	3A	31	31	30	31	30	35	30	30	30	41	--	--	0D	0A

Read Holding Registers : 03

MASTER機器のSlave機器側（局番：17）の「400108 - 400110 Register」データを読み取る例により、「03」コマンドフレームを説明します。

■ RTU Mode

（Master → Slave： 要求フレーム ）

Comment	Slave 局番	コマンド	先頭機器		機器点数		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	03	00	6B	00	03	--	--

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	データ数(byte)	データ						チェックコード（CRC）	
				Register 40108		Register 40109		Register 40110			
				H	L	H	L	H	L	L	H
Hex	11	03	06	02	2B	00	00	00	64	--	--

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				機器点数 (Word)				チェックコード（LRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	1	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	31	33	30	30	32	35	--	--	0D	0A

（Slave → Master： 応答フレーム）

Comment	Header	Slave 局番		コマンド		データ数(byte)		データ										チェックコード（LRC）		Tail	
								Register 40108				Register 40109				Register 40110							
								H	–	–	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	3	0	6	0	2	2	B	0	0	0	0	0	0	6	4			CR	LF
Hex	3A	31	31	30	31	30	35	30	32	32	42	30	30	30	30	30	30	36	34	--	--	0D	0A

Preset Single Register : 06

Slave機器側の400002 Registerに0003（hex）データを入力する例により、「06」コマンドフレームを説明します。

■ RTU Mode

（Master → Slave： 要求フレーム ）

Comment	Slave 局番	コマンド	先頭機器		Preset data		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	06	00	01	00	03	--	--

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	先頭機器		Preset data		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	06	00	01	00	03	--	--

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				Preset data				チェックコード（CRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	6	0	0	0	1	0	0	0	3			CR	LF
Hex	3A	31	31	30	36	30	30	30	31	30	30	30	33	--	--	0D	0A

（Slave → Master： 応答フレーム）

comment	Header	Slave 局番		コマンド		先頭機器				Preset data				チェックコード（CRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	6	0	0	0	1	0	0	0	3			CR	LF
Hex	3A	31	31	30	36	30	30	30	31	30	30	30	33	--	--	0D	0A

Preset Multiple Register : 10

Slave機器側の400002 Registerに「000A（hex）」、「0102（hex）」の連続した二つのデータを入力する例により、「10」コマンドフレームを説明します。（Error Code : 90H）

■ RTU Mode

（Master → Slave： 要求フレーム ）

Comment	Slave 局番	コマンド	先頭機器		Quantity of Register (Word Count)		データ数(Byte)	データ				チェックコード（CRC）	
								Register 40002		Register 40003			
			H	L	H	L		H	L	H	L	L	H
Hex	11	10	00	01	00	02	04	00	0A	01	02	--	--

（Slave → Master： 応答フレーム）

Comment	Slave 局番	コマンド	先頭機器		Quantity of Register (Word Count)		チェックコード（CRC）	
			H	L	H	L	L	H
Hex	11	10	00	01	00	02	--	--

■ ASCII Mode

（Master → Slave： 要求フレーム ）

comment	Header	Slave 局番		コマンド		先頭機器				Quantity of Register (Word Count)				データ数(Byte)		データ							
																Register 40002				Register 40003			
		H	L	H	L	H	–	–	L	H	–	–	L	–	L	H	–	–	L	H	–	–	L
ASCII	:	1	1	1	0	0	0	0	1	0	0	0	2	0	4	0	0	0	A	0	1	0	2
Hex	3A	31	31	31	30	30	30	41	43	30	30	30	32	30	34	30	30	30	41	30	31	30	32

➔ 継続…

	チェックコード（CRC）		Tail	
	L	H		
ASCII			CR	LF
Hex	--	--	0D	0A

（Slave → Master： 応答フレーム）

comment	Header	Slave 局番		コマンド		先頭機器				Quantity of Register (Word Count)				チェックコード（CRC）		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	1	0	0	0	0	1	0	0	0	2			CR	LF

Hex	3A	31	31	30	31	30	30	30	31	30	30	30	32	--	--	0 D	0A

A.5 LRC/CRC Generation

(1) LRC Generation

The Longitudinal Redundancy Check (LRC) field is one byte, containing an 8–bit binary value. The LRC value is calculated by the transmitting device, which appends the LRC to the message. The receiving device recalculates an LRC during receipt of the message, and compares the calculated value to the actual value it received in the LRC field. If the two values are not equal, an error results.

The LRC is calculated by adding together successive 8–bit bytes in the message, discarding any carries, and then two's complementing the result. The LRC is an 8–bit field, therefore each new addition of a character that would result in a value higher than 255 decimal simply 'rolls over' the field's value through zero. Because there is no ninth bit, the carry is discarded automatically.

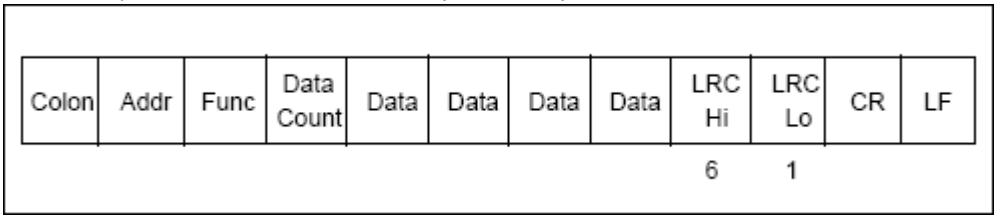
A procedure for generating an LRC is:

1. Add all bytes in the message, excluding the starting 'colon' and ending CRLF. Add them into an 8–bit field, so that carries will be discarded.
2. Subtract the final field value from FF hex (all 1's), GX8roduce the ones–complement.
3. Add 1 produce the twos–complement.

– Placing the LRC into the Message

When the 8–bit LRC (2 ASCII characters) is transmitted in the message, the high–order character will be transmitted first, followed by the low–order character.

For example, if the LRC value is 61 hex (0110 0001):



– Example

An example of a C language function performing LRC generation is shown below.

The function takes two arguments:

```
unsigned char *auchMsg ;          // A pointer to the message buffer containing
                                  // binary data to be used for generating the LRC
unsigned short usDataLen ;        // The quantity of bytes in the message buffer.
```

The function returns the LRC as a type unsigned char.

– LRC Generation Function

```
static unsigned char LRC(auchMsg, usDataLen)
unsigned char *auchMsg ;                    /* message to calculate LRC upon */
unsigned short usDataLen ;                  /* quantity of bytes in message */
{
    unsigned char uchLRC = 0 ;              /* LRC char initialized */
    while (usDataLen--)                     /* pass through message buffer */
    uchLRC += *auchMsg++ ;                  /* add buffer byte without carry */
    return ((unsigned char)(-((char)uchLRC))) ;   /* return twos complement */
}
```

(2) CRC Generation

The Cyclical Redundancy Check (CRC) field is two bytes, containing a 16–bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The receiving device recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results.

The CRC is started by first preloading a 16–bit register to all 1's. Then a process begins of applying successive 8–bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and SGX8 bits, and the parity bit, do not apply to the CRC.

During generation of the CRC, each 8–bit character is exclusive ORed with the register contents. Then the result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8–bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final contents of the register, after all the characters of the message have been applied, is the CRC value.

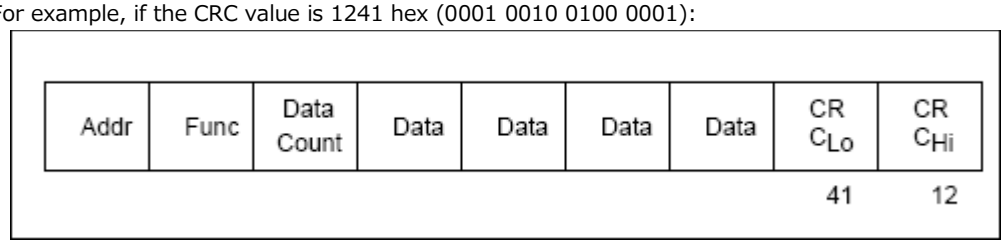
A procedure for generating a CRC is:

1. Load a 16–bit register with FFFF hex (all 1's). Call this the CRC register.
2. Exclusive OR the first 8–bit byte of the message with the low–order byte of the 16–bit CRC register, putting the result in the CRC register.
3. Shift the CRC register one bit to the right (toward the LSB), zero–filling the MSB. Extract and examine the LSB.
4. (If the LSB was 0): Repeat Step 3 (another shift). (If the LSB was 1): Exclusive OR the CRC register with the polynomial value A001 hex (1010 0000 0000 0001).
5. Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8–bit byte will have been processed.
6. Repeat Steps 2 through 5 for the next 8–bit byte of the message. Continue doing this until all bytes have been processed.
7. The final contents of the CRC register is the CRC value.
8. When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

– Placing the CRC into the Message

When the 16–bit CRC (two 8–bit bytes) is transmitted in the message, the low-order byte will be transmitted first, followed by the high-order byte.

For example, if the CRC value is 1241 hex (0001 0010 0100 0001):



– Example

An example of a C language function performing CRC generation is shown on the following pages. All of the possible CRC values are preloaded into two arrays, which are simply indexed as the function increments through the message buffer.

One array contains all of the 256 possible CRC values for the high byte of the 16–bit CRC field, and the other array contains all of the values for the low byte. Indexing the CRC in this way provides faster execution than would be achieved by calculating a new CRC value with each new character from the message buffer.

> **Note** This function performs the swapping of the high/low CRC bytes internally. The bytes are already swapped in the CRC value that is returned from the function. Therefore the CRC value returned from the function can be directly placed into the message for transmission.

The function takes two arguments:

```
unsigned char *puchMsg ;            //A pointer to the message buffer containing
                                    //binary data to be used for generating the CRC
unsigned short usDataLen ;          //The quantity of bytes in the message buffer.
```

The function returns the CRC as a type unsigned short.

– CRC Generation Function

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg ;                    /* message to calculate CRC upon */
unsigned short usDataLen ;                  /* quantity of bytes in message */
{
    unsigned char uchCRCHi = 0xFF ;          /* high byte of CRC initialized */
    unsigned char uchCRCLo = 0xFF ;          /* low byte of CRC initialized */
    unsigned uIndex ;                       /* will index into CRC lookup table */
    while (usDataLen--)                     /* pass through message buffer */
    {
        uIndex = uchCRCHi ^ *puchMsgg++ ;    /* calculate the CRC */
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex} ;
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

– High-Order Byte Table

```
/* Table of CRC values for high–order byte */
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,
0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80,
0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;
```

– Low-Order Byte Table

```
/* Table of CRC values for low–order byte */
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D,
0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB,
0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2,
0x12, 0x13, 0xD3, 0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37,
0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8,
0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24,
0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26, 0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63,
0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE,
0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F,
0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5, 0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C,
0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89,
0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C, 0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46,
0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;
```