MODBUS Organization, Inc.

MODBUS Serial Master Driver

サポートバージョン	OS	V4.0 以上
	XDesignerPlus	4.0.0.0 以上

CONTENTS

㈱ミスミの“Touch Operation Panel(M2I GX7) Series”をご使用になるお客様に感謝申し上げます。本マニュアルを読んで“GX7-外部装置”の接続方法および手順を熟知してください。

1. システム構成 **2ページ**

接続に必要な機器、各機器の設定、ケーブル、構成可能なシステムについて説明します。
本節を参照して適切なシステムを選定してください。

2. **GX7機種と外部装置選択** **3ページ**

GX7機種と外部装置を選択します。

3. システム設定例題 **4ページ**

本機器と該当外部端末の通信接続のための設定例題を説明します。
“1. システム構成”で選択したシステムに従って例題を選択してください。

4. 通信設定項目 **7ページ**

GX7の通信設定をする方法について説明します。
外部装置の設定が変わる場合、本章を参照してGX7の設定も外部装置と同じように設定してください。

5. ケーブル表 **10ページ**

接続に必要なケーブル仕様について説明します。
“1. システム構成”で選択したシステムに従って適したケーブル仕様を選択してください。

6. サポートアドレス **14ページ**

本節を参照して外部装置と通信可能なアドレスを確認してください。

APPENDIX A. MODBUS Protocol **15ページ**

本機器の“MODBUS Serial Master Driver”がサポートするMODBUSプロトコルのコマンドおよびデバイスについて説明します。

1. システム構成

本ドライバは“MODBUS Organization, Inc.”の“MODBUS Protocol”のうち“Serial Master Driver”です。

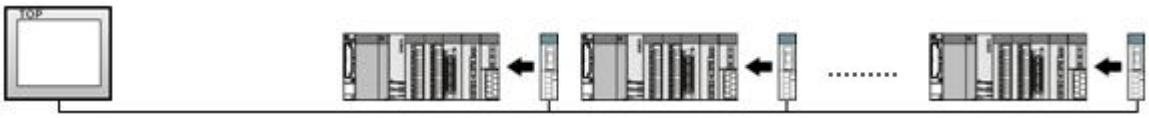
外部装置(MODBUS Slave Protocolサポート)に従ってドライバの“コマンドコード”、“プロトコルフレーム形式”などを別途に設定しなければなりません。この場合、通信方式による細部設定事項を外部装置側に合わせて設定してください。
本ドライバがサポートする外部装置とのシステム構成は下記のとおりです。

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	3.1 設定例題 1 (4ページ)	5.1 ケーブル表 1 (10ページ)
			RS-422 (4 wire)	3.2 設定例題 2 (5ページ)	5.2 ケーブル表 2 (11ページ)
			RS-485 (2 wire)	3.3 設定例題 3 (6ページ)	5.3 ケーブル表 3 (12ページ)

■ 接続構成ＭＯＤＢＵＳ

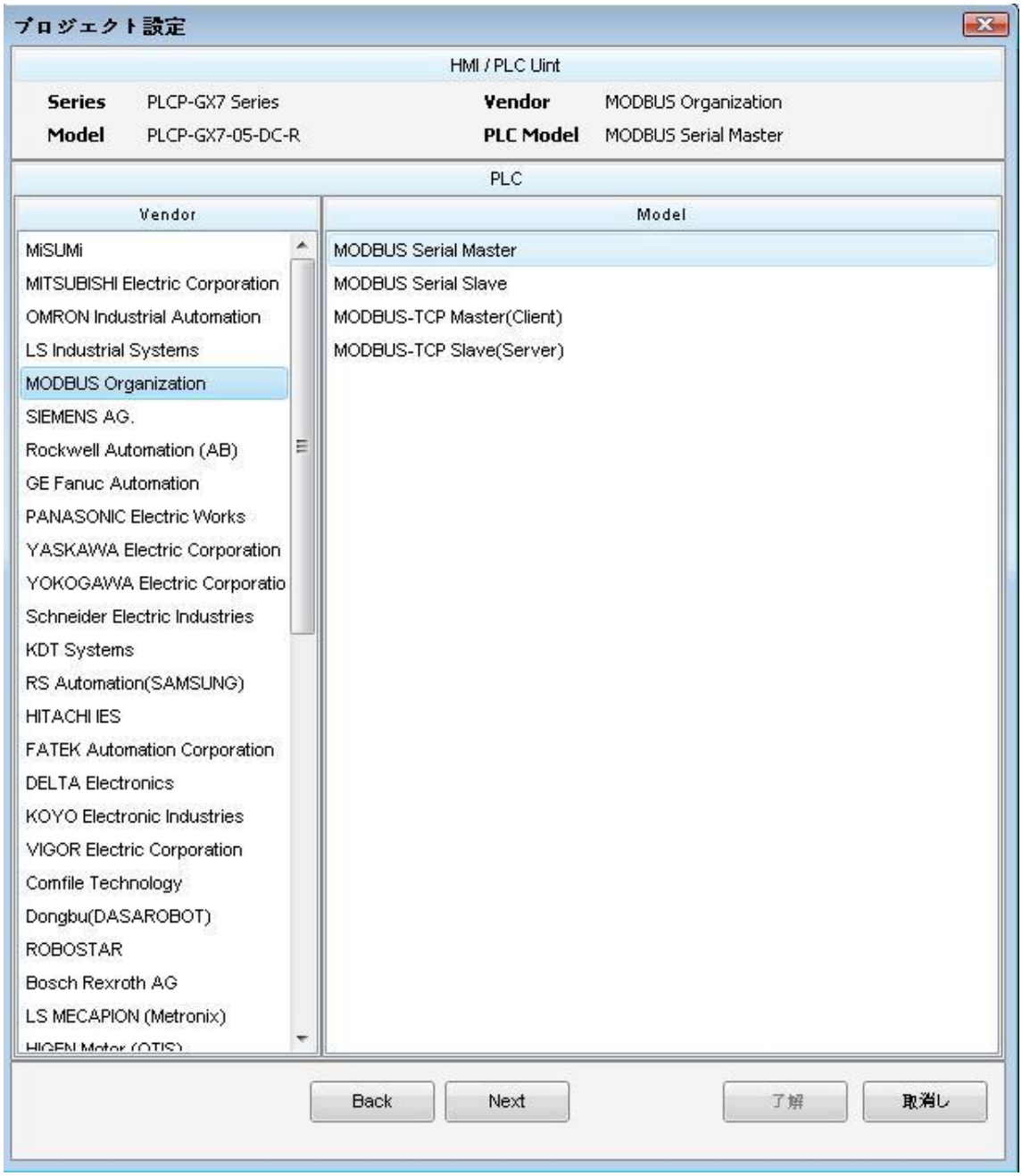
・1：1(GX7 1台と外部装置1台)接続 – RS232C/422/485通信で可能な構成です。

・1：N(GX7 1台と外部装置複数台)接続 – RS422/485通信で可能な構成です。



2. GX7機種と外部装置選択

GX7と接続される外部装置を選択します。



設定事項		内容
GX7	Series	PLCと接続するGX7のシリーズ名称を選択します。 設定内容をDownloadする前にGX7のシリーズによって下の表に明示されたバージョンのOSをインストールしてください。 シリーズ：XGX7 / HGX7 バージョン名称：V4.0
	Name	GX7製品のモデル名を選択します。
外部装置	製造会社	GX7と接続する外部装置の製造会社を選択します。 “MODBUS Organization, Inc.”を選択してください。
	PLC	GX7に接続される外部装置のモデルシリーズを選択します。 “MODBUS Serial Master Driver”を選択してください。 接続する外部装置がシステム構成可能な機種なのか1章のシステム構成で確認してください。

3. システム設定例題

GX7と"MODBUS Slave Device"の通信インターフェース設定を下記のとおり推奨します。

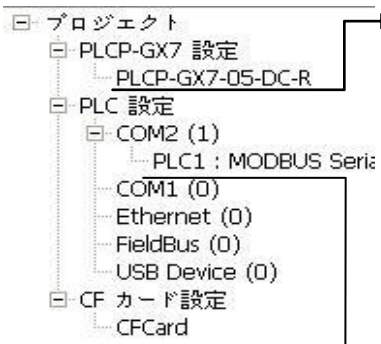
3.1 設定例題 1

構成したシステムを下記のとおり設定します。

項目		GX7	Slave Device	備考
シリアルレベル(ポート/チャンネル)		RS-232C (COM2)	RS-232C	ユーザー設定
局番(PLC Address)		—	0	ユーザー設定
シリアルボーレート	[BPS]	38400		ユーザー設定
シリアルデータビット	[Bit]	8		ユーザー設定
シリアル停止ビット	[Bit]	1		ユーザー設定
シリアルパリティビット	[Bit]	NONE		ユーザー設定

(1) XdesignerPlus設定

[プロジェクト > プロジェクト設定]で下の内容を設定後、GX7機器に設定内容をダウンロードします。



■ [プロジェクト > プロジェクトプロパティ > プロジェクト > 設定 > GX7 Name].
GX7機器の通信インターフェースを設定します。

– 右ウインドウで [HMI設定 > HMI設定使用チェック > 装置管理者]

■ 外部装設定

"MODBUS Serial Master Driver" 通信ドライバのオプションを設定します。

– PLC局番(PLC) : 外部装置設定局番

– プロトコルモード : Protocol Frame形式上でMODBUS RTU/ASCIIモードを選択します。

– 書き込み命令 : "4"デバイスの書き込みコマンドを選択します。(選択書き込み:06、連続書き込み:10)

– 読み込みビット単位 : "0"、"1"デバイスのcoil、Input接点処理単位を選択します。

– デバイス読み込み方式 : Register読み込み単位を選択します。(16ビット/32ビット/最大128ワード連続処理)

– 実数型データ(32ビット)ワードスワップ使用 : 実数型データのHigh/ Low Wordの順序をLow/High Wordの順に設定します。

(2) 外部装置設定

外部装置のユーザーマニュアルを参照して外部機器I/Fに"MODBUS Serial Slave Driver"を設定してください。

- Protocol Frame形式上でRTU/ASCⅡのモード選定に注意してください。
- 外部装置側のアドレスマップの内容を確認し、その内容に従って通信アドレスを使ってください。

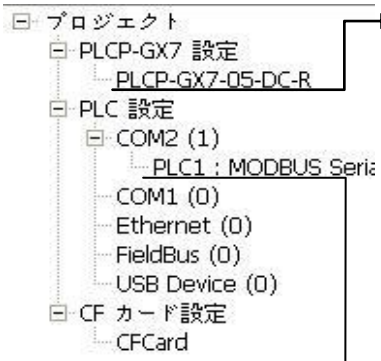
3.2 設定例題 2

構成したシステムを下記のとおり設定します。

項目		GX7	Slave Device	備考
シリアルレベル(ポート/チャンネル)		RS-422 (4 wire, COM2)	RS-422	ユーザー設定
局番(PLC Address)		—	0	ユーザー設定
シリアルボーレート	[BPS]	38400		ユーザー設定
シリアルデータビット	[Bit]	8		ユーザー設定
シリアル停止ビット	[Bit]	1		ユーザー設定
シリアルパリティビット	[Bit]	NONE		ユーザー設定

(1) XdesignerPlus設定

[プロジェクト > プロジェクト設定]で下の内容を設定後、GX7機器に設定内容をダウンロードします。



■ [プロジェクト > プロジェクトプロパティ > プロジェクト > 設定 > GX7 Name].
GX7機器の通信インターフェースを設定します。

– 右ウインドウで [HMI設定 > HMI設定使用チェック > 装置管理者]

■ 外部装設定

"MODBUS Serial Master Driver" 通信ドライバのオプションを設定します。

– PLC局番(PLC) : 外部装置設定局番

– プロトコルモード : Protocol Frame形式上でMODBUS RTU/ASCIIモードを選択します。

– 書き込み命令 : "4"デバイスの書き込みコマンドを選択します。(選択書き込み:06、連続書き込み:10)

– 読み込みビット単位 : "0"、"1"デバイスのcoil、Input接点処理単位を選択します。

– デバイス読み込み方式 : Register読み込み単位を選択します。(16ビット/32ビット/最大128ワード連続処理)

– 実数型データ(32ビット)ワードスワップ使用 : 実数型データのHigh/ Low Wordの順序をLow/High Wordの順に設定します。

(2) 外部装置設定

外部装置のユーザーマニュアルを参照して外部機器I/Fに"MODBUS Serial Slave Driver"を設定してください。

- Protocol Frame形式上でRTU/ASCIIのモード選定に注意してください。
- 外部装置側のアドレスマップの内容を確認し、その内容に従って通信アドレスを使ってください。

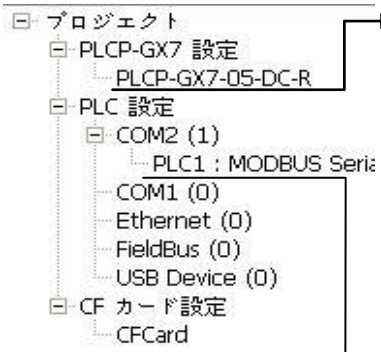
3.3 設定例題 3

構成したシステムを下記のとおり設定します。

項目		GX7	Slave Device	備考
シリアルレベル(ポート/チャンネル)		RS-485 (2 wire, COM2)	RS-485	ユーザー設定
局番(PLC Address)		—	0	ユーザー設定
シリアルボーレート	[BPS]	38400		ユーザー設定
シリアルデータビット	[Bit]	8		ユーザー設定
シリアル停止ビット	[Bit]	1		ユーザー設定
シリアルパリティビット	[Bit]	NONE		ユーザー設定

(1) XDesignerPlus設定

[プロジェクト > プロジェクト設定]で下の内容を設定後、GX7機器に設定内容をダウンロードします。



■ [プロジェクト > プロジェクトプロパティ > プロジェクト > 設定 > GX7 Name].
GX7機器の通信インターフェースを設定します。

– 右ウインドウで [HMI設定 > HMI設定使用チェック > 装置管理者]

■ 外部装設定

"MODBUS Serial Master Driver" 通信ドライバのオプションを設定します。

– PLC局番(PLC) : 外部装置設定局番

– プロトコルモード : Protocol Frame形式上でMODBUS RTU/ASCIIモードを選択します。

– 書き込み命令 : "4"デバイスの書き込みコマンドを選択します。(選択書き込み:06、連続書き込み:10)

– 読み込みビット単位 : "0"、"1"デバイスのcoil、Input接点処理単位を選択します。

– デバイス読み込み方式 : Register読み込み単位を選択します。(16ビット/32ビット/最大128ワード連続処理)

– 実数型データ(32ビット)ワードスワップ使用 : 実数型データのHigh/ Low Wordの順序をLow/High Wordの順に設定します。

(2) 外部装置設定

外部装置のユーザーマニュアルを参照して外部機器I/Fに"MODBUS Serial Slave Driver"を設定してください。

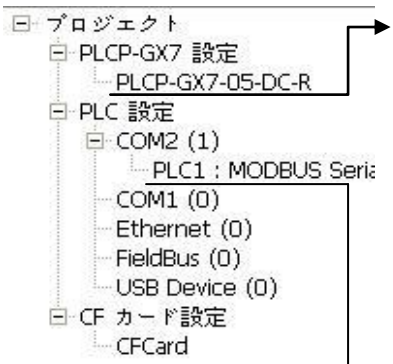
- Protocol Frame形式上でRTU/ASCIIのモード選定に注意してください。
- 外部装置側のアドレスマップの内容を確認し、その内容に従って通信アドレスを使ってください。

4. 通信設定項目

通信設定はXDesignerPlusもしくはGX7メインメニューで設定できます。通信設定は外部装置と同じように設定してください。

4.1 XDesignerPlus設定項目

下の画面の内容を示すために[プロジェクト > プロジェクトプロパティ]を選択してください。



■ [プロジェクト > プロジェクトプロパティ > プロジェクト > 設定 > GX7 Name].
GX7機器の通信インターフェースを設定します。

– 右ウインドウで [HMI設定 > HMI設定使用チェック > 装置管理者]

– 右ウインドウで [HMI設定 > HMI設定使用チェック > PLC設定]

■ 外部装置設定

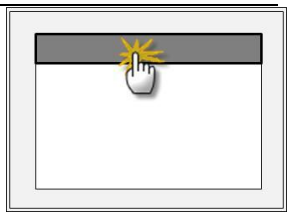
"MODBUS Serial Master Driver" 通信ドライバのオプションを設定します。

■ 通信インターフェース設定

項目	内容
信号レベル	外部装置 – GX7間のシリアル通信方式を選択します。(COM 1はRS-232Cだけを提供します。)
ボーレート	外部装置 – GX7間のシリアル通信速度を選択します。
データビット	外部装置 – GX7間のシリアル通信データビットを選択します。
停止ビット	外部装置 – GX7間のシリアル通信停止ビットを選択します。
パリティビット	外部装置 – GX7間のシリアル通信パリティビット確認方式を選択します。
タイムアウト[x100 mSec]	GX7が外部装置からの応答を待機する時間を[0 – 5000] x 1 mSecに設定します。
送信遅延時間[x10 mSec] 受信待機時間[x10 mSec]	GX7が外部装置からの応答受信 – 次のコマンド要請転送の間に待機する時間を[0 – 5000] x 1 mSecに設定します。
PLC局番[0~65535]	相手機器の局番です。[0 – 65535]までの値を選択します。

4.2 GX7メインメニュー設定項目

- 電源をリセット中にブザー音が鳴るとき、LCD上端1点をタッチして"GX7管理メイン"画面に移動します。
- GX7でドライバインターフェース設定は下記の**Step1 → Step2**の内容に従って設定します。
  (**Step 1.** で"GX7 COM 2/1設定"を押せば、**Step2.**で設定を変えることができます。)



Step 1. [PLC設定] – ドライバインターフェースを設定します。

PLC設定	
PLC局番 : 00 タイムアウト : 1000 [mSec] 送信前の遅延時間 : 0 [mSec] GX7 COM 2/1 : RS - 232C , 38400 , 8 , 1 , NONE GX7 COM 2/1 設定 通信診断	通信インターフェース設定

Step 1–Reference.

項目	内容
PLC局番 [0~65535]	相手機器の局番です。[0 – 65535]までの値を選択します。
タイムアウト [x1 mSec]	GX7が外部装置からの応答を待機する時間を[0 – 5000] x 1 mSecに設定します。
送信前の遅延時間 [x1 mSec]	GX7が外部装置からの応答受信 – 次のコマンド要請転送の間に待機する時間を[0 – 5000] x 1 mSecに設定します。
GX7 COM 2/1	GX7が外部装置に対するインターフェース設定です。

Step 2. [PLC設定] >[GX7 COM2/COM1設定] – 該当ポートのシリアルパラメータを設定します。

ポート設定	
* シリアル通信 + COM-1 Port - ボーレート : 38400 [BPS] - データビット : 8 [BIT] - 停止ビット : 1 [BIT] - パリティビット : NONE [BIT] - 信号レベル : RS – 232C	COM 1ポート 通信インターフェース設定
+ COM-2 Port - ボーレート : 38400 [BPS] - データビット : 8 [BIT] - 停止ビット : 1 [BIT] - パリティビット : NONE [BIT] - 信号レベル : RS – 232C	COM 2ポート 通信インターフェース設定

Step 2-Reference.

項目	内容
ボーレート	外部装置 – GX7間のシリアル通信速度を選択します。
データビット	外部装置 – GX7間のシリアル通信データビットを選択します。
停止ビット	外部装置 – GX7間のシリアル通信停止ビットを選択します。
パリティビット	外部装置 – GX7間のシリアル通信パリティビット確認方式を選択します。
信号レベル	外部装置 – GX7間のシリアル通信方式を選択します。

4.3 通信診断

■ GX7 – 外部装置間のインターフェース設定状態を確認

- GX7の電源をリセットし、LCD画面の上端をクリックしてメニュー画面に移動する。
- [通信設定]で使用するポート[COM 2 or COM 1]設定が外部機器の設定内容と同じか確認する。

■ ポート通信の異常有無診断

- PLC設定 > GX7 [COM 2もしくはCOM 1] "通信診断"のボタンをクリックする。
- 画面上にDiagnosticsダイアログボックスがポップアップされ、ボックスの3番項に表示された内容によって診断状態を判断する。

OK!	**通信設定　正常**
Time Out Error!	通信設定　非正常 - ケーブルおよびGX7/外部装置の設定状態エラー **(参照 : 通信診断シート)**

■ 通信診断シート

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

Designer Version		O.S Version	

項目	内容				確認	
システム構成	CPU名称				OK	NG
	通信相手のポート名称				OK	NG
	システム接続方法	1:1	1:N	N:1	OK	NG
接続ケーブル	ケーブル名称				OK	NG
PLC設定	設定局番				OK	NG
	Serial baud rate	[BPS]			OK	NG
	Serial data bit	[BIT]			OK	NG
	Serial SGX7 bit	[BIT]			OK	NG
	Serial parity bit	[BIT]			OK	NG
	アドレス割当範囲				OK	NG
GX7設定	設定ポート	COM 1	COM 2		OK	NG
	ドライバ名称				OK	NG
	相手局番	Project Property設定			OK	NG
		通信診断時			OK	NG
	Serial baud rate	[BPS]			OK	NG
	Serial data bit	[BIT]			OK	NG
	Serial SGX7 bit	[BIT]			OK	NG
	Serial parity bit	[BIT]			OK	NG

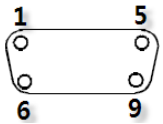
5. ケーブル表

本ChapterはGX7と該当機器との正常な通信のためのケーブルダイアグラムを紹介します。(本節で説明するケーブルダイアグラムは外部装置側の製造会社の推奨事項と異なることがあります)

5.1 ケーブル表 1

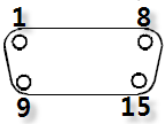
■ 1：1接続

(A) XGX7 COM 2ポート(9ピン)

XGX7 COM2			ケーブル接続	PLC
ピン配列*注1)	信号名	ピン番号		信号名
	CD	1		
	RD	2		SD
	SD	3		RD
	DTR	4		DTR
	SG	5		SG
通信ケーブルコネクタ前面基準、D-SUB 9 Pin male(オス、凸)	DSR	6		DSR
	RTS	7		RTS
	CTS	8		CTS
		9		

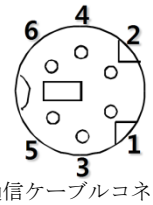
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

(B) XGX7 COM 2ポート(15ピン)

XGX7 COM2			ケーブル接続	PLC
ピン配列*注1)	信号名	ピン番号		信号名
	CD	1		
	RD	2		SD
	SD	3		RD
	DTR	4		DTR
	SG	5		SG
通信ケーブルコネクタ前面基準、D-SUB 15 Pin male(オス、凸)	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。

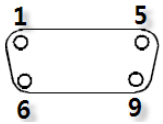
(C) XGX7/AGX7 COM 1ポート(6ピン)

XGX7/AGX7 COM 1ポート			ケーブル接続	PLC
ピン配列*注1)	信号名	ピン番号		信号名
		1		
	RD	2		SD
	SG	3		RD
		4		DTR
		5		SG
通信ケーブルコネクタ前面基準、D-SUB 6 Pin male(オス、凸)	SD	6		DSR
				RTS
				CTS

*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。

■ 1：1接続

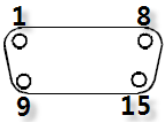
(A) XGX7 COM 2ポート(9ピン)

XGX7 COM2			ケーブル接続	PLC
ピン配列*注1)	信号名	ピン番号		信号名
	RDA	1		SDA
		2		SDB
		3		RDA
	RDB	4		RDB
	SG	5		SG
	SDA	6		
		7		
		8		
	SDB	9		

通信ケーブルコネクタ前面基準、D-SUB 9 Pin male(オス、凸)

*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。

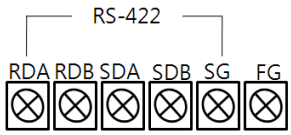
(B) XGX7 COM 2ポート(15ピン)

XGX7 COM2			ケーブル接続	PLC
ピン配列*注1)	信号名	ピン番号		信号名
	–	1		SDA
	(생략)			SDB
				RDA
	–	10		RDB
	RDA	11		SG
	RDB	12		
	SDA	13		
	SDB	14		
	SG	15		

通信ケーブルコネクタ前面基準、D-SUB 15 Pin male(オス、凸)

*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。

(C) AGX7 COM 2ポート(ターミナルブロック5 pin)

AGX7 COM2		ケーブル接続	PLC
ピン配列*注1)	信号名		信号名
	RDA		SDA
	RDB		SDB
	SDA		RDA
	SDB		RDB
	SG		SG

通信ケーブルコネクタ前面基準 ターミナルブロック5 Pin

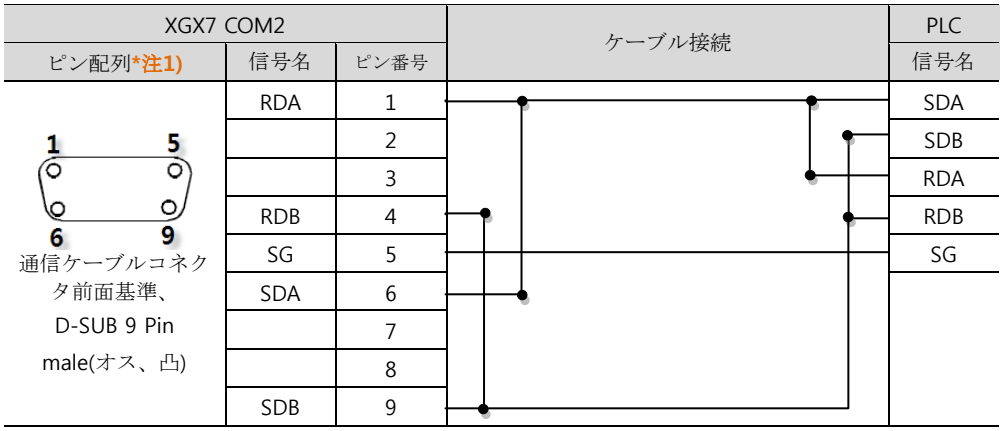
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

■ 1：N接続 – 1:1接続を参考にして下記の方式で接続してください。

GX7	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

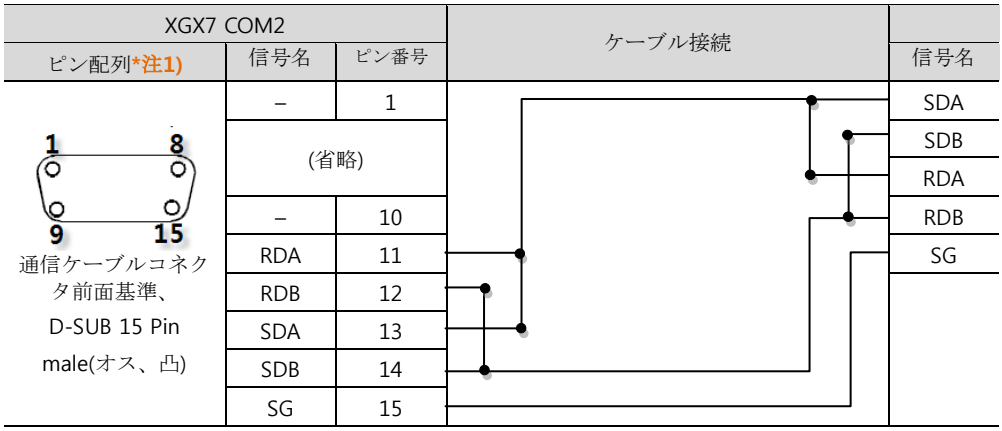
■ 1：1接続 – TYPE A

(A) XGX7 COM 2ポート(9ピン)



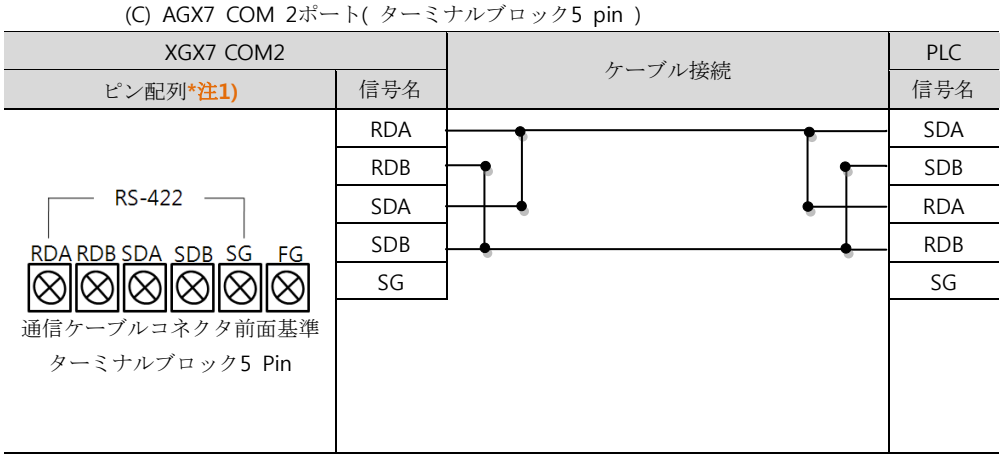
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

(B) XGX7 COM 2ポート(15ピン)



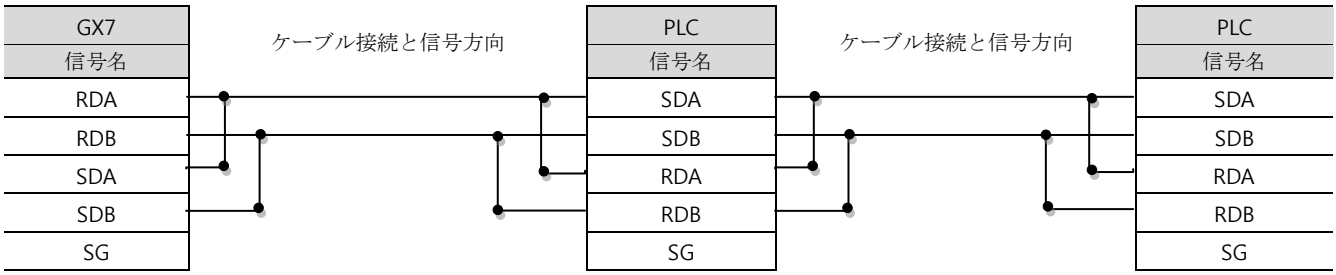
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

(C) AGX7 COM 2ポート(ターミナルブロック5 pin)



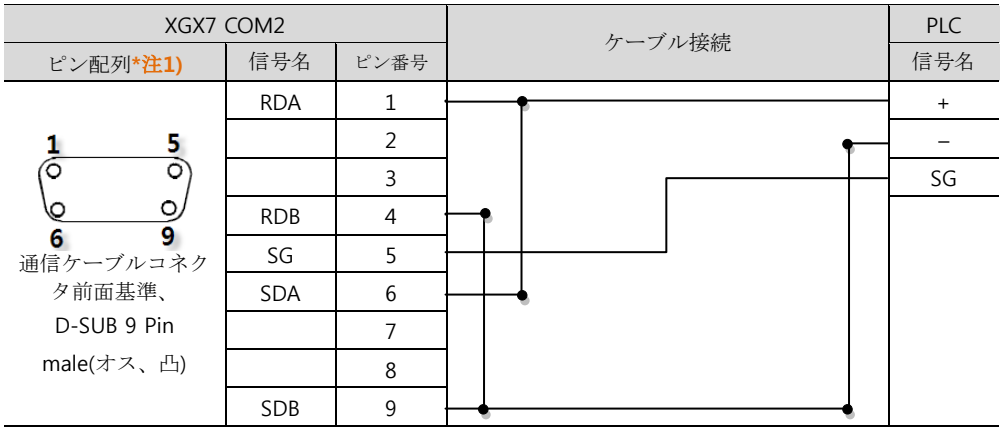
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

■ 1：N接続 – 1:1接続を参考にして下記の方式で接続してください。



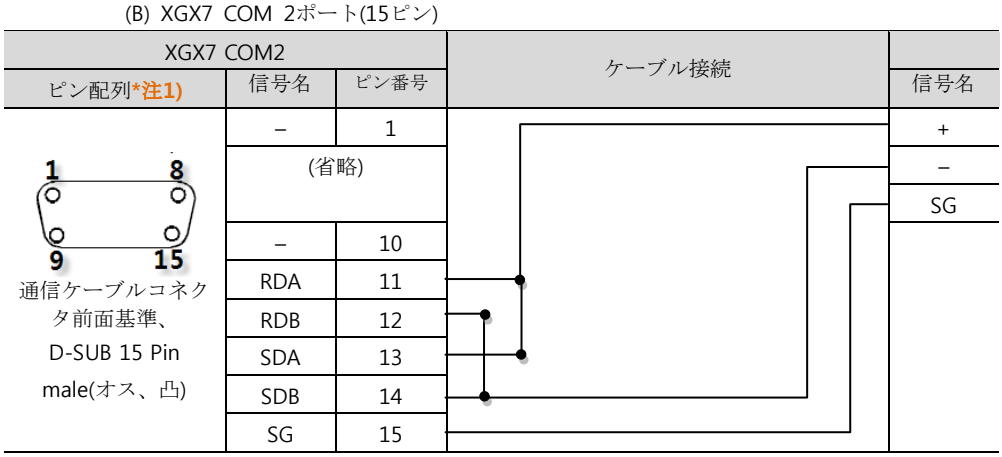
■ 1：1接続 – TYPE B

(A) XGX7 COM 2ポート(9ピン)



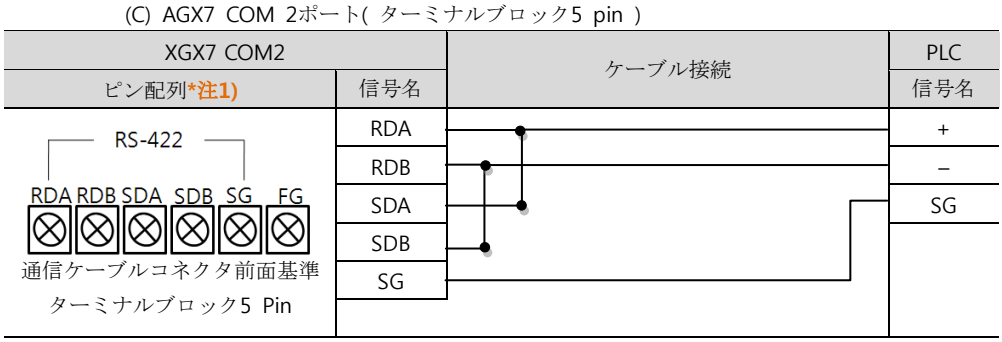
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

(B) XGX7 COM 2ポート(15ピン)



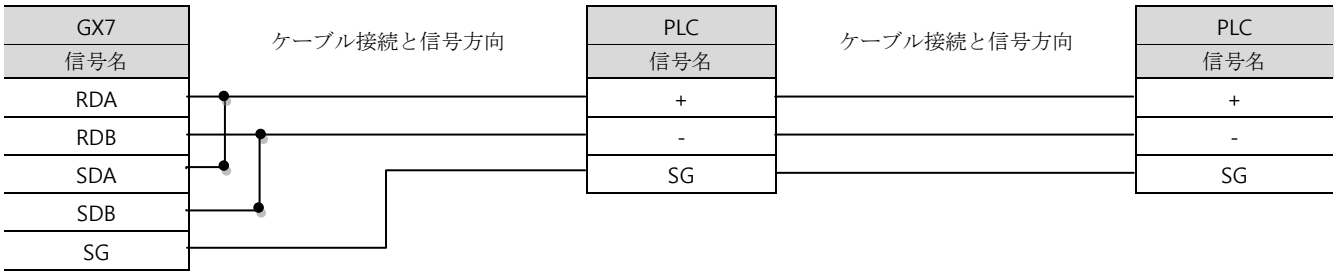
*注1) ピン配列はケーブル接続コネクタの接続面から見たものです。

(C) AGX7 COM 2ポート(ターミナルブロック5 pin)



*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。

■ 1：N / N：1接続 – 1:1接続を参考にして下記の方式で接続してください。



6. サポートアドレス

GX7で使用可能なデバイスは下記のとおりです。

CPUモジュールシリーズ/タイプによってデバイス範囲(アドレス)に差があることがあります。GX7シリーズは外部装置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。使用する装置がサポートするアドレス範囲から出ないように各CPUモジュールユーザーマニュアルを参照/注意してください。

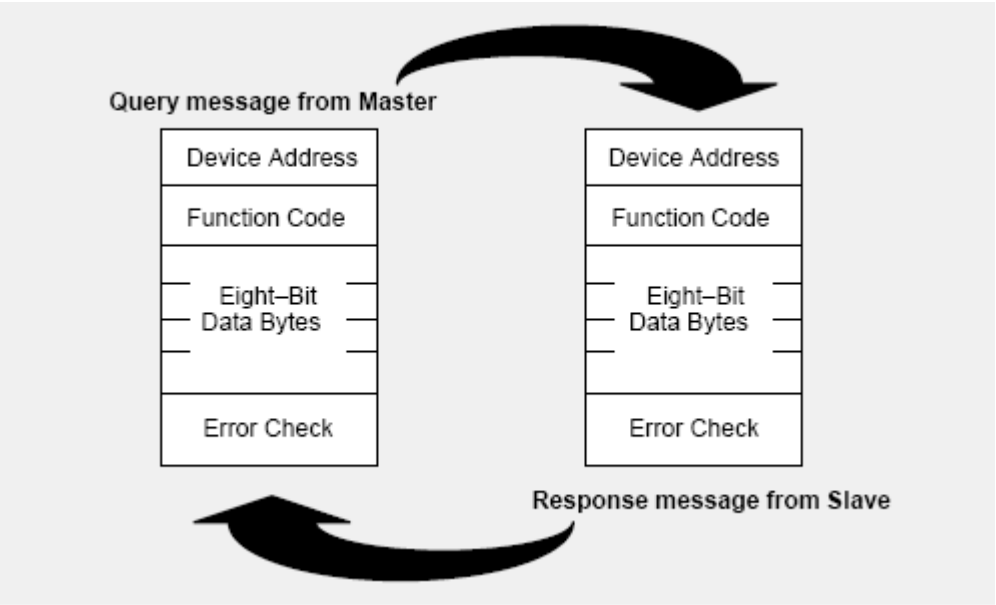
	Bit Address	Word Address	32 bits	Remarks
Coil	000001 – 065536	000001 – 065521	L/H	
Discrete Input	100001 – 165536	100001 – 165521		*注1)
Input Register	300001.00 – 365536.15	300001 – 365536		*注1)
Holding Register	400001.00 – 465536.15	400001 – 465536		

***注1)** 書き込み不可能(読み込み専用)

Appendix A. Standatd MODBUS Protocol

本機器の"MODBUS Serial Master Driver"がサポートするMODBUSプロトコルコマンドおよびデバイス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–to–peer. 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 to perform. The data bytes contain any additional information that the slave will need to perform 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機器でSlave機器側(局番:17番)の“**0**00020–**0**00056 Coil” データを読み込む例題を通じて“01”コマンドフレームを説明します。

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		デバイス点数		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	データ個数(byte)	データ					チェックコード (CRC)	
				Coils 27~20	Coils 35~28	Coils 43~36	Coils 51~44	Coils 56~52	L	H
				L	–	–	–	H		
Hex	11	01	05	CD	6B	B2	0E	1B	—	—

■ Coils データ状態

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 : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				デバイス点数				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		データ個数(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機器でSlave機器側のCoil **0**00173にFORCE "ON"する例題を通じて"05"コマンドフレームを説明します。

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		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	先頭デバイス		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 : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				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		先頭デバイス				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機器でSlave機器側(局番:17番)の”**1**00197–**1**00218 Input” データを読み込む例題を通じて“02”コマンドフレームを説明します。

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		デバイス点数		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	データ個数(byte)	データ(Inputs) 10204~10197	10212~10205	10218~10213	チェックコード (CRC)	
							L	H
Hex	11	02	03	AC	DB	35	—	—

■ Coils データ状態

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 : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				デバイス点数				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		データ個数(byte)		データ(Inputs) 10204~10197		10212~10205		10218~10213		チェックコード (LRC)		Tail	
								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機器でSlave機器側(局番:17番)の”300009 Register”データを読み込む例題を通じて”03”コマンドフレームを説明します。

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		デバイス点数 (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	データ個数(byte)	データ Register 30009		チェックコード (CRC)	
				H	L	L	H
Hex	11	04	02	00	0A	—	—

■ ASCII Mode

(Master → Slave : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				デバイス点数 (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		データ個数(byte)		データ Register 40108				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機器でSlave機器側(局番:17)の“**4**00108 – **4**00110 Register”データを読み込む例題を通じて“03”コマンドフレームを説明します。

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		デバイス点数		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	データ個数(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 : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				デバイス点数(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		データ個数(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**4**00002 Registerに00 03 (hex)データを入力する例題を通じて"06"コマンドフレームを説明します。

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		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	先頭デバイス		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 : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				Preset data				チェックコード (LRC)		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		先頭デバイス				Preset data				チェックコード (LRC)		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“00 0A (hex)”、“01 02 (hex)”連続した2つのデータを入力する例題を通じて“10”コマンドフレームを説明します。(Error Code : 90_H)

■ RTU Mode

(Master → Slave : 要請フレーム)

Comment	Slave 局番	コマンド	先頭デバイス		Quantity of Register (Word Count)		データ個数(Byte)	データ Register 40002		データ Register 40003		チェックコード (CRC)	
			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	先頭デバイス		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 : 要請フレーム)

comment	Header	Slave 局番		コマンド		先頭デバイス				Quantity of Register (Word Count)				データ個数(Byte)		データ Register 40002				データ 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 (LRC)		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		先頭デバイス				Quantity of Register (Word Count)				チェックコード (LRC)		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	—	—	0D	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), to produce the ones–complement.
3. Add 1 to 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):

Colon	Addr	Func	Data Count	Data	Data	Data	Data	LRC Hi	LRC Lo	CR	LF
								6	1		

– 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 sGX7 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):

Addr	Func	Data Count	Data	Data	Data	Data	CR C_{Lo}	CR C_{Hi}
							41	12

– 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
} ;
```