ASI Communications Protocol

Modbus

USER GUIDANCE DOCUMENT NO. 1.21
ACCELERATED SYSTEMS INC.

Communications between the ASI display and motor controller is based on the MODBUS application layer messaging protocol. Not all of the MODBUS functionality is supported. MODBUS is a request/response protocol and offers services specified by function codes. MODBUS function codes are elements of MODBUS request/response packets. The objective of this document is to describe the function codes used within the ASI implementation of MODBUS.

VERSION 1.22
FEBRUARY 2022

[Type here]

ASI ACCELERATED SYSTEMS INC.

CONTENTS

OVERVIEW

Communications between the ASI display and motor controller is based on the MODBUS application layer messaging protocol. Not all of the MODBUS functionality is supported. MODBUS is a request/response protocol and offers services specified by function codes. MODBUS function codes are elements of MODBUS request/response packets. The objective of this document is to describe the function codes used within the ASI implementation of MODBUS

1.0 Modbus Functions Supported

Function	Function Code
Read Holding Registers	0x03
Write Multiple Registers	0x10

Table 1 - Function codes

Byte Definition

Feature	Description
Byte Format	8-bit binary 1 Start bit 1 Stop bit No parity
Bit Order	Least Significant Bit First

Table 2 - Byte definitions

Timing

Feature	Value
Baud Rate	115200 bits per second
Byte time	10bits/115200bps = 86.8 usec
Minimum time between frames	3.5 byte times (304 usec)
Maximum time between bytes within a frame	<= 1.5 byte times (130 usec)

Table 3 - Communication Timing

Logic Levels

State	Level
Low	0 V
High	5 V

Table 4 - Logic level definitions

2.0 DEFINITIONS

Item	Description
Slave ID	Unique identifier assigned to each slave device on the Modbus. By default, the Slave ID = 1
Function Code	Code used to identify the type of action to be performed by the slave device.
Error Code	Code contained in the slave device response packet to indicate that the last request from the master was invalid.
Exception Code	Code used to indicate why an Error code was returned by the slave.
Start Address	Register Address of first register to be read or written.
Number of Registers	Number of registers to read or write from a single Request packet.
Register #	Data byte containing high or low order byte from a register.
CRC	Cyclic Redundancy Check. See calculation below.
Master	The device on the bus which initiates all communication events. Example: the display device.
Slave	Devices on the bus which reply to communication requests from a Master device. Example: the motor controller.

Table 5 - Modbus definitions

3.0 Frame Descriptions

3.1 Expected Responses

When a master device sends a request to a slave device it expects a normal response. One of four possible events can occur from the master's query:

- If the slave device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the slave does not receive the request due to a communication error, no response is returned. The master will eventually process a timeout condition for the request.
- If the slave receives the request, but detects a communication error (parity, LRC, CRC, etc), no response is returned. The client program will eventually process a timeout condition for the request.
- If the slave receives the request without a communication error, but cannot handle it (for example, if the request contains an invalid register address), the slave will return an exception response informing the master of the nature of the error.

3.2 Read Holding Registers (Function 0x03)

This function is used to read the values stored in selected registers within the controller.

The valid **Number of Registers** that can be read using a single read request packet is from 1 to 125. If the number of registers requested is outside of this range, an error response will be returned by the controller.

The valid address range is from 0 to 511. When reading multiple registers with a single request, the **Start Address** + **Number of Registers** must not result in the reading of an address above 511. Attempts to do so will result in an error response from the controller.

Below is a byte wise description of the packet frames used to perform a Function 0x03 read operation.

3.3 Request (Master to Slave)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Slave ID	Function Code (0x03)	Start Address (High Byte)	Start Address (Low Byte)	Number of Registers (N*) (High Byte)	Number of Registers (N*) (Low Byte)	CRC (Low Byte)	CRC (High Byte)

Table 6 - Request (master to slave)

3.4 Valid Response (Slave to Master)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte ...	Byte 2xN+1	Byte 2xN+2	Byte 2xN+3	Byte 2xN+4
Slave ID	Function Code (0x03)	Number of data bytes (2 x N*)	Register 1 (High byte)	Register 1 (Low Byte)	…	Register N* (High byte)	Register N* (Low Byte)	CRC (Low Byte)	CRC (High Byte)

Table 7 - Valid response (slave to master)

* N = number of registers (1 to 125)

3.5 Error Responses (Slave to Master)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
Slave ID	Error Code (0x83)	Exception Code: 0x02 = Invalid Address 0x03 = Invalid Number of Registers	CRC (Low Byte)	CRC (High Byte)

Table 8 - Error responses (slave to master)

3.6 Write Multiple Registers (Function 0x10)

This function is used to write values into select registers within the controller.

The valid **Number of Registers** that can be written using a single write request is from 1 to 123. If the number of registers written to is outside of this range, an error response will be returned by the controller.

The valid address range is from 0 to 511. When writing multiple registers with a single write request packet, the **Start Address** + **Number of Registers** must not result in the writing of an address above 511. Attempts to do so will result in an error response from the controller.

Below is a byte wise description of the packet frames used to perform a Function 0x10 write operation.

3.7 Request (Master to Slave)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Slave ID	Function Code (0x10)	Start Address (High Byte)	Start Address (Low Byte)	Number of Registers (N*) (High Byte)	Number of Registers (N*) (Low Byte)	Number of data bytes (2 x N*)	Register 1 (High byte)

Byte 8	Byte …	Byte 2xN+7	Byte 2xN+8	Byte 2xN+9	Byte 2xN+10
Register 1 (Low Byte)	…	Register N* (High byte)	Register N* (Low Byte)	CRC (Low Byte)	CRC (High Byte)

Table 9 - Request (master to slave)

3.8 Valid Response (Slave to Master)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Slave ID	Function Code (0x10)	Start Address (High Byte)	Start Address (Low Byte)	Number of Registers (N*) (High Byte)	Number of Registers (N*) (Low Byte)	CRC (Low Byte)	CRC (High Byte)

Table 10 - Valid response (slave to master)

* N = number of registers (1 to 123)

3.9 Error Responses (Slave to Master)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
Slave ID	Error Code (0x90)	Exception Code: 0x02 = Invalid Address 0x03 = Invalid Number of Registers	CRC (Low Byte)	CRC (High Byte)

Table 11 - Error responses (slave to master)

4.0 CRC GENERATION

The Cyclical Redundancy Checking (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 will result.

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 stop 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 that was shifted off 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 XOR takes place. This process of shifting, extracting and XORing on an LSB value of 1 is repeated seven more times, so a total of 8 *shift/extraction/exclusive OR on '1'* steps have been performed.

After the last (eighth) repetition, the next 8–bit character is exclusive ORed with the CRC register's current value, and the process repeats for eight more shifts as described above. The final content 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. Extract the LSB of the CRC register, then shift the register one bit to the right.
4. (If the LSB was 0): Do nothing.
   (If the LSB was 1): Exclusive OR the CRC register with the polynomial value 0xA001 (1010 0000 0000 0001).
5. Repeat Steps 3 and 4 until 8 shifts, LSBs examined and XORs on 1 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 content 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.

Parameter Address List

For a complete list of parameters, please refer to the object dictionary file "ASIObjectDictionary.xml", which resides in the c:\Accelerated Systems\BacDoor\ directory, which is created when installing ASI's BacDoor configuration software.

This file specifies the name, address, and scale factor for all available configuration parameters in the BAC controller.

X.0 Document History

Version	Updated	Author	Revision Notes
1.0	08/26/2011	Terry Stone	Document Created
1.1	08/31/2011	Craig MacKinnon	Table 12 definitions change to correspond to BAC software release 4.xx and earlier
1.2	07/21/2014	Terry Stone	Remove Table 12 and refer to object dictionary file
1.21	11/09/2017	Martin Kunze	New template
1.22	2/28/2022	Nick Oudyk	Update to CRC generation section for clarity.