The iowad Project

1. Introduction 2
2. Types of ports 2
3. Layout of D16 ports 3
4. Layout of D8 ports 3
5. Layout of Flags port 4
6. AD00 through AD15 4
7. DA00 through DA15 5
8. MTR00 through MTR15 5
9. MTS00 through MTS15 6
10. RF00 through RF15 6
11. DP00 through DP15 7
12. ADBank port 7
13. DABank port 7
14. RFBank port 8
15. LCD ports 8
16. MTRBank port 8
17. RFBank port 9
18. StepT flag 9
19. Reset flag 9

The iowad protocol 10

1. Introduction 10
2. Poll command 10
3. IAmHere response 11
4. ReadD16 command 11
5. ReadD8 command 11
6. ReadFlag command 12
7. WriteD16 command 12
8. WriteD8 command 12
9. WriteFlag0, WriteFlag1 command 13
10. WriteMultiD8 command 13
11. NotSupported response 13
12. Acknowledge response 14
13. Data16 response 14
14. Data8 response 14
15. DataFlag response 15

1. Introduction

The iowad project defines a protocol and port layout for connecting an I/O processor to a host processor using a communications link. The iowad protocol describes the exchange of data between the I/O processor and the host processor. The iowad port layout defines the various virtual ports controlled by the I/O processor. These virtual ports may be accessed by the host processor through data exchanged over the iowad protocol.

The protocol and port layout are intended to be as platform-independent as possible. The I/O processor must support a minimal version of the protocol and a minimum set of ports, but any microcontroller (MCU) that supports these minimal requirements can be used.

The protocol does not support any error checking or correction. It is assumed that the connection between the host processor and the I/O processor will be short and reliable. In those instances where the connection is unreliable, a higher-level protocol, such as BACnet or TCP/IP can be used to add the needed reliability.

The physical connection between the I/O and host processors may use any suitable medium, including RF link, RS-232 serial, or Ethernet. The protocol defines the contents of the data stream, not the medium.

2. Types of ports

The iowad protocol supports three different types of virtual ports. D16 ports are 16-bit digital ports, used for exchanging 16-bit data, such as A/D results, D/A commands, or motor positioning commands.

D8 ports are 8-bit digital ports, used for exchanging bytes of data, such as LCD data, bank selection, or file contents.

Flag ports are 1-bit digital ports, used for controlling or sensing digital I/O ports, selecting operational characteristics, or monitoring operations on the I/O processor.

The protocol supports 256 of each type of port. Note that most ports do not have a physical equivalent; for example, the I/O processor is not required to support 256 16-bit digital ports. Most of the ports will be represented by function results, converted data, or will not even be supported by the I/O processor.

The iowad protocol reserves the first 128 ports of each type. The functionality of these ports is either reserved, or is described fully in this document. The second 128 ports of each type are available for custom use by the system designer. Obviously, functionality of such ports is not platform independent.

In the following paragraphs, a shorthand notation is used to describe specific ports. The notation is the port type, an underscore, and the port number. For example, port 8 of the D16 port layout is noted as D16_8.

Many sections of this document list behaviors of ports as implementation-specific. In those instances, the I/O processor firmware has freedom to implement functionality not described here. Obviously, such functionality will not be platform- or host-independent, but it does permit the designer to extend the firmware's capability.

3. Layout of D16 ports

The following table shows the assignment of each of the 128 D16 ports.

D16	+0	+1	+2	+3	+4	+5	+6	+7
0	AD00	AD01	AD02	AD03	AD04	AD05	AD06	AD07
8	AD08	AD09	AD10	AD11	AD12	AD13	AD14	AD15
16	DA00	DA01	DA02	DA03	DA04	DA05	DA06	DA07
24	DA08	DA09	DA10	DA11	DA12	DA13	DA14	DA15
32	MTR00	MTR01	MTR02	MTR03	MTR04	MTR05	MTR06	MTR07
40	MTR08	MTR09	MTR10	MTR11	MTR12	MTR13	MTR14	MTR15
48	MTS00	MTS01	MTS02	MTS03	MTS04	MTS05	MTS06	MTS07
56	MTS08	MTS09	MTS10	MTS11	MTS12	MTS13	MTS14	MTS15
64	RF00	RF01	RF02	RF03	RF04	RF05	RF06	RF07
72	RF08	RF09	RF10	RF11	RF12	RF13	RF14	RF15
80								
88								
96								
104								
112								
120								
128								

4. Layout of D8 ports

The following table shows the assignment of each of the 128 D8 ports.

D8	+0	+1	+2	+3	+4	+5	+6	+7
0	DP00	DP01	DP02	DP03	DP04	DP05	DP06	DP07
8	DP08	DP09	DP10	DP11	DP12	DP13	DP14	DP15

16	ADBank	DABank	MTR Bank	RFBank				
24	LCDC0	LCDD0	LCDC1	LCDD1	LCDC2	LCDD2	LCDC3	LCDD3
32								
40								
48								
56								
64								
72								
80								
88								
96								
104								
112								
120								
128								

5. Layout of Flags port

The following table shows the assignment of each of the 128 flag ports.

Flag	+0	+1	+2	+3	+4	+5	+6	+7
0	Reset	StepT						
8								
16								
24								
32								
40								
48								
56								
64								
72								
80								
88								
96								
104								
112								
120								
128								

6. AD00 through AD15

Ports AD00 through AD15 are read/write D16 ports connected to A/D converters.

When read, each provides the current value of an associated A/D converter. The value in this port is right-adjusted. Based on implementation, this value may be signed or unsigned.

When written, each provides an implementation-specific function. For example, writes to an AD port might change A/D filtering, scanning mode, or select single/double-ended operation.

These ports correspond to the currently selected bank of AD ports, based on the current value of the A/D bank selection port, ADBank. Changes to ADBank will map a different block of AD ports into this port space.

Implementations are not required to support all or any AD ports. Accesses to AD ports with no associated A/D converter must return NotSupported responses.

7. DA00 through DA15

Ports DA00 through DA15 are read/write D16 ports connected to D/A converters.

When written, each causes a corresponding voltage or current to appear at the output of the associated D/A converter. The value in this port is right-adjusted. Based on implementation, this value may be signed or unsigned.

When read, each provides implementation-specific information about the associated port. For example, reads of a D/A port might return filter settings or flags indicating single/double-ended operation.

These ports correspond to the currently selected bank of DA ports, based on the current value of the D/A bank selection port, DABank. Changes to DABank will map a different block of DA ports into this port space.

Implementations are not required to support all or any DA ports. Accesses to DA ports with no associated D/A converter must return NotSupported responses.

8. MTR00 through MTR15

Ports MTR00 through MTR15 are read/write D16 ports connected to motor controllers. These motor controllers may control any type of motor, including servo, DC gearhead, or stepper.

When written, each assigns a corresponding motor duty cycle and direction for the associated motor controller. The value of this port is signed. A value of 0 indicates full stop. Positive values select a scaled motor speed or position in an implementation-specific direction, with a value of $7fff indicating full speed or throw. Negative values select a scaled motor speed or position in the opposite direction, with a value of $8000 indicating full speed or throw.

When read, each returns the value previously written.

Note that writing a value to a MTR port is not enough to trigger the start of motor motion. The motor will not begin moving until a write to the StepT flag is done.
These ports correspond to the currently selected bank of MTR ports, based on the current value of the MTR bank selection port, MTRBank.

Implementations are not required to support all or any MTR ports. Accesses to MTR ports with no associated motor must return NotSupported responses.

9. MTS00 through MTS15

MTS00 through MTS15 are read/write D16 ports, each associated with the corresponding MTR port. The MTS ports provide step (distance) targets for an associated motor.

When read, each provides current step information for the associated motor. This value is unsigned, and indicates the number of steps the motor has moved in the direction selected by the corresponding MTR port since the motor began moving.

When written, the value determines the number of steps a motor must move in the direction selected by the corresponding MTR port. Motor motion will not start until the flag StepT changes from 0 to 1. For each motor, motion stops automatically when the target number of motor steps has been reached.

These ports correspond to the currently selected bank of MTS ports, based on the current value of the MTR bank selection port, MTRBank. Changes to MTRBank will map a different block of MTS ports into this port space.

Implementations are not required to support all or any MTS ports. Accesses to MTS ports with no associated motor must return NotSupported responses.

10. RF00 through RF15

RF00 through RF15 are read-only D16 ports, each associated with a range finder. The technology for performing the range finding is not specified, and need not be consistent across all RF ports.

When read, each port provides the distance to a target object, if any, in units of hundredths of an inch. The value returned is always unsigned, permitting a range measurement of 0.00 to 655.30 inches (values $0000 through $FFFA). If a sonar range finder does not have an object within range, it returns a value of 655.35 inches ($FFFF). The remaining range values ($FFFA through $FFFE) are reserved for error conditions.

Writing to an RF port may be used to trigger a ranging operation for an individual port, though support for this is not required by the protocol. In response to such a write to an existing device, the I/O processor should respond with an Acknowledge packet.

Triggering of a range-finder reading is normally handled in background by the I/O processor. The host processor should assume that when a range value is returned, it is the most recent value for the selected device.

These ports correspond to the currently selected bank of RF ports, based on the current value of the RF bank selection port, RFBank. Changes to RFBank will map a different block of RF ports into this port space.

Implementations are not required to support all or any RF ports. Accesses to RF ports with no associated device must return NotSupported responses.

11. DP00 through DP15

DP00 through DP15 are read/write D8 ports, each associated with an eight-bit digital port.

When written, each bit of a DP port appears on a designated output line of the I/O processor.

When read, each bit of a DP port reflects the voltage, translated to a logic level, on a designated input line of the I/O processor.

Implementations are not required to support all or any DP ports. All bits of implemented DP ports must be supported within the protocol, though some DP ports may be partially implemented. For example, a DP port may have physical outputs or inputs for only four of the bits, but reads or writes to all eight bits must be supported. Reading of unimplemented bits within a DP port will return 0. Writing of unimplemented bits within a DP port will have no effect externally.

12. ADBank port

The ADBank port is a D8 port that selects one of 256 different banks of 16 AD ports. The selected AD ports appear in the port layout as AD00 through AD15.

All I/O processor implementations are required to support the ADBank port. The default value of ADBank on power-up or reset is 0. Implementations are not required to support ADBank values other than 0. If an unsupported value is written to ADBank, the I/O processor must return a NotSupported response and leave the current value in ADBank unchanged.

13. DABank port

The DABank port is a D8 port that selects one of 256 different banks of 16 DAPorts. The selected DA Ports appear in the port layout as DA00 through DA15.

All I/O processor implementations are required to support the DABank port. The default value of DABank on power-up or reset is 0. Implementations are not required to support DABank values other than 0. If an unsupported value is written to DABank, the I/O processor must return a NotSupported response and leave the current value in DABank unchanged.

14. RFBank port

The RFBank port is a D8 port that selects one of 256 different banks of 16 RFPorts. The selected RF Ports appear in the port layout as RF00 through RF15.

All I/O processor implementations are required to support the RFBank port. The default value of RFBank on power-up or reset is 0. Implementations are not required to support RFBank values other than 0. If an unsupported value is written to RFBank, the I/O processor must return a NotSupported response and leave the current value in RFBank unchanged.

15. LCD ports

The LCD ports are D8 ports that provide control of up to four alphanumeric LCDs. The ports are named LCDCn and LCDDn, corresponding to the command and data registers, respectively.

Reads of the data and command ports return implementation-specific data. Not all implementations will provide read access to these registers. If read access is not provided, reads of these registers should return a NotSupported response.

Writes to the command registers can be used to control characteristics of the LCD, such as cursor positioning, character selection, or reset functions.

Writes to the data registers transfer data to the LCD.

LCD characteristics, such as number of rows and columns of characters, are implementation-specific.

Implementations are not required to support any or all LCD ports. Accesses to LCD ports with no supported LCD must return a NotSupported response.

If any LCDs are supported, the I/O processor is responsible for power-on reset of the LCDs, as this is usually a timing-critical operation. The host processor may assume that all accesses to the LCDs via the above registers are timing-insensitive. The I/O processor must handle all timing-sensitive transactions with the phyxical LCDs.

16. MTRBank port

The MTRBank port is a D8 port that selects one of 256 different banks of 16 MTR ports. The selected MTR ports appear in the port layout as MTR00 through MTR15. The value in MTRBank also selects a corresponding bank of 16 MTS ports, which appear in the port layout as MTS00 through MTS15.

All I/O processor implementations are required to support the MTRBank port. The default value of MTRBank on power-up or reset is 0. Implementations are not required to support MTRBank values other than 0. If an unsupported value is written to MTRBank, the I/O processor must return a NotSupported response and leave the current value in MTRBank unchanged.

17. RFBank port

The RFBank port is a D8 port that selects one of 256 different banks of 16 RF ports. The selected RF ports appear in the port layout as RF00 through RF15.

All I/O processor implementations are required to support the RFBank port. The default value of RFBank on power-up or reset is 0. Implementations are not required to support RFBank values other than 0. If an unsupported value is written to RFBank, the I/O processor must return a NotSupported response and leave the current value in RFBank unchanged.

18. StepT flag

The StepT flag is a Flag port used to synchronize starts of motors, such as steppers. After one or more motor motions have been setup, using writes to MTR ports, the host initiates the motion sequences by setting the StepT flag using a WriteFlag1 command. The I/O processor then starts the motor sequences and clears the StepT flag.

Support of the StepT flag is required if the implementation supports one or more motor ports (MTRPorts). If the StepT flag is not supported, the I/O processor should respond to all references to the StepT flag with a NotSupported response.

Reads of the StepT flag returns a DataFlag response of 0, as this flag will always be cleared.

Writes of the StepT flag should return an Acknowledge response. Note that writing a 0 to the StepT flag has no effect.

19. Reset flag

Upon power-up reset, the target processor sets this flag. The flag stays set until the host processor reads it. When the host processor reads the reset flag, the I/O processor returns the current state of the reset flag, using the appropriate DataFlag response, then immediately clears the reset flag.

This mechanism allows the host processor to determine when the I/O processor has been reset, either through power-cycling or under host direction.

All I/O processor implementations are required to support the reset flag.

Reads of the reset flag return the flag's current state and automatically clear the flag.

Writes to the reset flag are not supported; the I/O processor is required to return a NotSupported response.

The iowad protocol

1. Introduction

The iowad protocol defines the syntax for exchaning information between the host processor and the I/O processor. The protocol definition covers sequencing of information, but intentionally excludes definition of medium. The protocol can be implemented over Ethernet, RS-232 serial, RF link, or any other suitable medium.

The protocol is a single-master, single-slave protocol, with the host processor acting as the master. All exchanges begin with a command from the host processor; the I/O processor cannot initiate an exchange. The I/O processor must respond to all valid commands, though sometimes those responses are simply an acknowledgement or an error response. The I/O processor ignores all packets that begin with an unknown byte.

In the following paragraphs, all bytes associated with packets are given in hexadecimal. Any spaces shown within a sequence of bytes are for formatting only; the spaces are not part of the packet.

2. Poll command

The host processor queries the I/O processor for its presence using a poll command. The poll command is a one-byte packet of the form:

A1

If the I/O processor receives this command, it should immediately return an IAmHere response.

In some implementations, such as RS-232 serial, the host processor can use the Poll command to determine the baud rate of the I/O processor. By issuing the Poll command at different baud rates, then looking for a response from the I/O processor, the host processor can deduce the proper baud rate.

Alternatively, the I/O processor can implement auto-bauding, by listening for a poll command and, if the byte received is incorrect, using the invalid data to estimate the correct baud and changing the serial port's baud rate. If auto-bauding is used between the host and I/O processors, the details of the handshaking is implementation specific.

3. IAmHere response

The I/O processor returns an IAmHere response when it receives a poll command. The IAmHere response is a one-byte packet of the form:

E0

This response must be returned to any and all poll commands immediately after the poll command is received.

4. ReadD16 command

The host processor issues the ReadD16 command to retreive the value of a specific D16 port. This command is a packet of the form:

C0 <D16_port>

where D16_port is the number, from 0 to 255, of the selected port, expressed as a single byte. For example, to request the value of D16_20, the host processor would send the packet:

C0 14

The I/O processor immediately responds with a suitable packet, either a Data16 response or a NotSupported response.

5. ReadD8 command

The host processor issues the ReadD8 command to retreive the value of a specific D8 port. This command is a packet of the form:

C1 <D8_port>

where D8_port is the number, from 0 to 255, of the selected port, expressed as a single byte. For example, to request the value of D8_20, the host processor would send the packet:

C1 14

The I/O processor immediately responds with a suitable packet, either a Data8 response or a NotSupported response.

6. ReadFlag command

The host processor issues the ReadFlag command to retreive the value of a specific Flag port. This command is a packet of the form:

C2 <Flag_port>

where Flag_port is the number, from 0 to 255, of the selected port, expressed as a single byte. For example, to request the value of Flag_20, the host processor would send the packet:

C2 14

The I/O processor immediately responds with a suitable packet, either a DataFlag response or a NotSupported response.

7. WriteD16 command

The host processor issues the WriteD16 command to change the value of a specific D16 port. This command is a packet of the form:

C8 <D16_port> <datah> <datal>

where D16_port is the number, from 0 to 255, of the selected port, expressed as a single byte, datam is high half of the 16-bit value to write, and datal is the low half of the 16-bit value to write. For example, to write the value of $1234 to D16_30, the host processor would send the packet:

C8 1E 12 34

The I/O processor immediately responds with a suitable packet, either an Acknowledge response or a NotSupported response.

8. WriteD8 command

The host processor issues the WriteD8 command to change the value of a specific D8 port. This command is a packet of the form:

C9 <D8_port> <data>

where D8_port is the number, from 0 to 255, of the selected port, expressed as a single byte, and data is 8-bit value to write. For example, to write the value of $12 to D8_30, the host processor would send the packet:

C9 1E 12

The I/O processor immediately responds with a suitable packet, either an Acknowledge response or a NotSupported response.

If it is necessary to write multiple bytes to the same D8 port, consider using the WriteMultiD8 command instead of using several WriteD8 commands.

9. WriteFlag0, WriteFlag1 command

The host processor issues the WriteFlag commands to change the value of a specific Flag port. This command is a packet in one of two forms:

CA <Flag_port>	**Clear Flag_port to 0**
CB <Flag_port>	**Set Flag_port to 1**

where Flag_port is the number, from 0 to 255, of the selected port, expressed as a single byte. For example, to set Flag_30, the host processor would send the packet:

CB 1E

The I/O processor immediately responds with a suitable packet, either an Acknowledge response or a NotSupported response.

10. WriteMultiD8 command

The host processor issues a WriteMultiD8 command to write multiple consecutive bytes to a single D8 port. This command is a packet of the form:

CC <D8_port> <num> <data_bytes>

where D8_port is the selected D8 port, num is a single byte containing the length of the data_bytes field, and data_bytes is the stream of bytes to write to the selected port.

This command can be used to write a stream of data to ports such as an LCD port, without having to resort to multiple WriteD8 commands.

11. NotSupported response

The I/O processor issues a NotSupported response whenever the host processor attempts to access an unsupported port. This response is a packet of the form:

F0

The host processor must assume that any write commands that generate a NotSupported response were ignored.

12. Acknowledge response

The I/O processor issues an Acknowledge response whenever the host processor issues a command that writes to a port or initiates a function on the I/O processor. This response indicates to the host processor that the requested action has occurred. This response is a packet of the form:

A0

The host processor must assume that the write command or function requrest that generated an Acknowledge response was successful.

13. Data16 response

The I/O processor issues a Data16 response whenever the host processor issues a ReadD16 command referencing a supported D16 port. The format of the Data16 reponse is:

A4 <datah> <datal>

where datah is the high byte of the port data and datal is the low part of the port data. For example, if the requested port contained the value $1234, a Data16 response to a read of that port would return:

A4 12 34

14. Data8 response

The I/O processor issues a Data8 response whenever the host processor issues a ReadD8 command referencing a supported D8 port. The format of the Data8 response is:

A3 <data>

where data is the byte of the port data. For example, if the requested port contained the value $34, a Data8 response to a read of that port would return:

A3 34

15. DataFlag response

The I/O processor issues a DataFlag response whenever the host processor issues a ReadFlag command referencing a supported Flag port. The DataFlag response takes one of two forms, depending on the value of the Flag port:

A1	**if Flag port value is TRUE or 1**
A2	**if Flag port value is FALSE or 0**