Chain Protocol V0.2

Document describes communication protocol used between IO modules (Modules) and controller (Controller)

Table of Contents

Index of Tables

Topology

Network consist of: one Controller and one or more Modules.

Controller has one communication port, and each Module has two communication ports. All the ports are bidirectional.

One of the Modules is directly connected to the Controller and the remaining Modules are connected in chain-like fashion. The communication ports in Module are dedicated: one of the ports (H) has to be connected with the Controller or with the preceding Module, the other port (T) can be used for connection of subsequent Modules. Port T of the last Module is left unconnected.

Controller – (port H) Module (port T) – ... – (port H) Module (port T, unconnected)

After initialisation procedure is completed every Module has unique ID number assigned. Module ID is used for addressing commands to specific Module, and for identifying Module responses. The Module directly connected to the Controller has ID = 0, and the remaining Modules are numbered in sequence.

Example, Controller with 3 Modules connected:

Controller – Module 0 – Module 1 – Module 2

Up to 250 Modules may be connected in the chain. Note, that communication delay grows with the increased number of Modules.

The Chain Protocol allow for dynamic Chain modification. Some of the modules can be removed, or new modules can be added. Note, that hot-plug support depends on the particular HW implementation.

Physical Layer

- RS422, Asynchronous serial transmission,
- Rate: 115200 bit/sec
- 8 data bits, LSB first, no parity, 1 stop bit

(for information about tunnelling over TCP/IP or USB see Appendix)

Frame Structure

All the data exchanged between Controller and Modules is organised into frames. Frames follow frame structure defined below.

Table 1: Frame Structure

Field	Position in frame	Length (bytes)	Description
CP_START	0	1	Frame start indicator, fixed value: 0x4D
CP_FIRST	1	1	Determines type of the frame (Command/Response) and length of the data in CP_BODY (LEN)
CP_BODY	2 .. 2+LEN	LEN	Frame body, this field is not present if LEN=0
CP_CHECKSUM	3+LEN	1	8-bit one's complement sum of all the bytes in CP_FIRST and CP_BODY (Example C implementation of checksum calculation is given in Appendix)

Three main frame types are defined.

Table 2: Frame Types

Frame type	Direction	CP_FIRST value	CP_BODY LEN
Module Response	Module → Controller	0x00 .. 0x7F	CP_FIRST
Long Command	Controller → Module	0x80 .. 0xBF	CP_FIRST & 0x3F
Short Command	Controller → Module(s)	0xC0 .. 0xFF	CP_FIRST & 0x3F

Short Command

Short Command frame can be sent from Controller to selected Module or to all the Modules (Broadcast).

Table 3: Short Commands

Command	CP_FIRST	CP_BODY	Description
GetEvents	0xC0	None	Get any pending events (Broadcast)
RollCall	0xC1	0x00	Probe modules & assign IDs (Broadcast) (also disables chain terminator)
GetInput	0xC2	ID, Select	Get Input state from Module *ID* (interpretation of *Select* is module dependent)

Long Command

Long Command frame can be sent from Controller to selected Module.

All Long Command frames follow structure defined below.

Table 4: Long Command Structure

Frame Field	Contents	Notes
CP_FIRST	0x80 + LEN	LEN >= 2
CP_BODY	ID	Target Module ID
	CMD	Secondary Command code
	(data)	Any additional data (present if LEN > 2)

Table 5: Long Commands

Command	Secondary Command Code	Additional data	Description
GetModule	0x00	-	Get Module type (string)
GetVersion	0x01	-	Get SW version (string)
Terminator	0x02	-	Enable chain terminator
GetPStat	0x03	...	Get communication / protocol status (see GetPStat Response table)
GetHW	0x04	...	Get HW status (module dependent)
GetConfig	0x40	Select	Get IO config Use of Select field is Module dependent.
SetConfig	0x41	Select, ...	Set IO config data is Module dependent
SetOutput	0x50	Select, Value	Set Output state data is Module dependent

Module Response

All Module Response frames follow structure defined below.

Table 6: Module Response Structure

Frame Field	Contents	Notes
CP_FIRST	LEN	LEN >= 1
CP_BODY	ID	Module ID
	RSP	Response code (present if LEN > 1)
	(data)	Any additional data (present if LEN > 2)

Table 7: RollCall Response

Frame Field	Contents	Notes
CP_FIRST	0x01	LEN = 1
CP_BODY	ID	Module ID – newly assigned

RollCall Response is sent by every Module in the chain in response to RollCall command.

Table 8: ChecksumError Response

Frame Field	Contents	Notes
CP_FIRST	LEN	LEN >= 4
CP_BODY	ID	Module ID
	0x00	ChecksumError response
	(data)	Start of the received (invalid) frame (at least 2 bytes are included, starting from CP_FIRST)

ChecksumError Response maybe sent by Module if Frame Checksum error is detected.

Table 9: GetInput Response

Frame Field	Contents	Notes
CP_FIRST	LEN	LEN > 2
CP_BODY	ID	Module ID
	0x01	GetInput response
	(data)	Requested input state – module dependent

GetInput Response is sent in response to GetInput command.

Table 10: Long Command Response

Frame Field	Contents	Notes
CP_FIRST	LEN	LEN >= 3
CP_BODY	ID	Module ID
	0x02	LongCommand response
	CMD	Original Long Command code
	(data)	Any additional data – command/module dependent

Long Command Response is sent in response to Long Command.

Table 11: Event Response

Frame Field	Contents	Notes
CP_FIRST	LEN	LEN >= 3
CP_BODY	ID	Module ID
	EVT	Event Code & Number
	(data)	Event data
		More then one event may be sent in a frame, in this case EVT & data fields are repeated Set Event chapter for more detail

Event Response is sent in response to GetEvents command.

Table 12: GetPStat Response

Field	Length (bytes)	Description
PStat_Flags	1	Bit 0 – Terminator state
PStat_Overflow	2	Number of bytes dropped due to input overflow
PStat_ChkErr	2	Number of frames received with checksum error
PStat_StartErr	2	Number of frame start errors
PStat_RcvdFrames	2	Number of valid frames received
		Additional data may be present

Communication Protocol

Chain Initialisation

After Power On or after Chain modification (adding or removing Modules) the following operations need to be completed:

- Detection of connected Modules
- Module ID assignment
- Chain termination – the last module in the chain has to be informed that there are no subsequent modules, and any messages have to be sent backward (towards the Controller)

Procedure:

1) Controller sends RollCall frame
2) Modules, in sequence:
   a. Assign Module ID (sequential numbers starting from 0)
   b. Respond with RollCall Response, using the newly assigned ID
   c. Pass the RollCall command to the next Module in chain
3) From the responses Controller finds the number of the Modules present, and sends Terminate command (as defined in Long Command List) to the last Module in chain (the Module with highest ID number). If no responses were received see below for error recovery.
4) The addressed Module activates Chain Termination, and responds with acknowledge (as defined in Long Command Response)
5) Now the Chain is initialised and ready for communication. Controller will usually proceed with querying the Type and status of all the Modules.

Example

Controller sends RollCall frame

T 4D C1 00 C1

RollCall response from Module 0 & Module 1

R 4D 01 00 01 4D 01 01 02

Controller sends Terminate command to Module 1

T 4D 82 01 02 85

Module 1 responds to Terminate command

R 4D 03 01 02 02 08

Initialisation Error Recovery

If no responses were received after RollCall, Controller may attempt to recover modules from potential frame error state by sending several short frames.

For example:

T 4D 00 00

T 4D 00 00

T 4D 00 00

Basic Functionality

The following commands must be supported by all Modules.

Table 13: Mandatory Commands

Command	Response	Notes
RollCall	RollCall Response	
Terminator	Long Command Response	No additional data in response frame
GetModule	Long Command Response	Data = Module type string
GetVersion	Long Command Response	Data = SW Version string
GetPStat	Long Command Response	Flags are mandatory, counters are optional

All the remaining commands are optional, their implementation depends on the actual Module Type.

If a frame (command) is not supported by Module it is returned to Controller unmodified. This feature may by used by Controller to probe which commands are supported.

Error Handling in Modules

When an invalid frame structure is detected (wrong CP_START byte), Module discards invalid bytes, and waits for the start of a valid frame. The discarded bytes may be counted.

When a frame checksum is detected, Module may:

- Pass the invalid frame unmodified (if all the Modules follow this option, the frame arrives to Controller);
- Indicate frame error with ChecksumError Response, and drop the invalid frame.

Events

...

Module-Dependent Functionality

Module Types

Module Type string	Description
'DI6 V01'	Digital Input Module PCB: DI6-01
'AI4 AO1 V01'	Analog Input & Analog Output Module PCB: xxx

'DI6 V01' Module

GetInput

Supported Select values: 0x00, 0xFF

Table 14: DI6, GetInput Response, Select=0x00

Frame Field	Contents	Notes
CP_FIRST	0x03	LEN
CP_BODY	ID	Module ID
	0x01	GetInput response
	Digital Inputs	length = 1 byte Bit 7 = Input 0 Bit 6 = Input 1 Bit 5 = Input 2 Bit 4 = Input 3 Bit 3 = Input 4 Bit 2 = Input 5 Bit 1..0 = not defined

Table 15: DI6, GetInput Response, Select=0xFF

Frame Field	Contents	Notes
CP_FIRST	0x03	LEN
CP_BODY	ID	Module ID
	0x01	GetInput response
	Virtual_IO	1 byte

SetOutput

Supported Select values: 0xFF

Table 16: DI6, SetOutput Command

Frame Field	Contents	Notes
CP_FIRST	0x84	LEN = 4
CP_BODY	ID	Target Module ID
	SetOutput	Secondary Command code
	0xFF	Select
	Virtual_IO	

Table 17: DI6, SetOutput Response, Select=0xFF

Field	Length (bytes)	Description
Virtual_IO	1	Virtual_IO

GetHW

Table 18: DI6, GetHW Response

Field	Length (bytes)	Description
HW_reserved	1	reserved
HW_Temp	2	Temperature
HW_Vcc	2	Vcc Voltage
HW_Temp30	2	Temperature sensor calibration for 30°C
HW_Temp85	2	Temperature sensor calibration for 85°C
HW_Tick	2	Internal timer

'AI4 AO1 V01' Module

GetInput

Supported Select values: 0x00

Table 19: AI4AO1, GetInput Response, Select=0x00

Frame Field	Contents	Notes
CP_FIRST	0x0E	LEN
CP_BODY	ID	Module ID
	0x01	GetInput response
	Input 1	length = 3 bytes
	Input 2	length = 3 bytes
	Input 3	length = 3 bytes
	Input 4	length = 3 bytes

SetOutput

Supported Select values: 0x00

Table 20: AI4AO1, SetOutput Command

Frame Field	Contents	Notes
CP_FIRST	0x85	LEN = 5
CP_BODY	ID	Target Module ID
	SetOutput	Secondary Command code
	0x00	Select
	Output 1	length = 2 bytes

Table 21: AI4AO1, SetOutput Command – used to sample current output value

Frame Field	Contents	Notes
CP_FIRST	0x83	LEN = 3
CP_BODY	ID	Target Module ID
	SetOutput	Secondary Command code
	0x00	Select

Table 22: AI4AO1, SetOutput Response, Select=0x00

Field	Length (bytes)	Description
Output 1	2	current value of Output 1

Embedded Controller

Table 23: Set Embedded Controller state

Frame Field	Contents	Notes
CP_FIRST	0x85	LEN = 5
CP_BODY	ID	Target Module ID
	SetConfig	Secondary Command code
	0x40	Set Embedded Controller state
	State	0 – disable, 1 – enable, 2 – reset
	InputChannel	Input Channel number

Table 24: Get Embedded Controller state

Frame Field	Contents	Notes
CP_FIRST	0x83	LEN = 3
CP_BODY	ID	Target Module ID
	GetConfig	Secondary Command code
	0x40	Get Embedded Controller state

Table 25: Set Embedded Controller parameters

Frame Field	Contents	Notes
CP_FIRST	0x83 + x	LEN = 3 + x
CP_BODY	ID	Target Module ID
	SetConfig	Secondary Command code
	0x50	Set Embedded Controller parameters
	Parameters	length = x

Table 26: Get Embedded Controller parameters

Frame Field	Contents	Notes
CP_FIRST	0x83	LEN = 3
CP_BODY	ID	Target Module ID
	GetConfig	Secondary Command code
	0x50	Get Embedded Controller parameters

Appendix

Frame Checksum calculation example

```
uint8_t CP_checksum(const uint8_t * data, unsigned len)
{
  unsigned chk = 0;
  while(len--)
    chk += *(data++);
  chk += (chk >> 8);
  return chk;
}
```

Controller Log Format

Line	Description
T aa bb cc dd<EOL>	data sent by Controller
R aa bb cc dd<EOL>	data received by Controller

aa, bb, cc, dd - data bytes (in hex format)

Number of data bytes per line: 1 .. 255

Tunneling over TCP/IP

There is no change to the data format.

TCP port number: 58003

Tunneling over USB

There is no change to the data format.

FTDI is setup in UART mode.

Communication Session Example

Controller sends RollCall frame

T 4D C1 00 C1

Module 0: RollCall response

R 4D R 01 R 00 R 01

Module 1: RollCall response

R 4D R 01 R 01 R 02

Controller → Module 1: Terminate

T 4D 82 01 02 85

Module 1: Response: len=03, ID=01, ACK CMD=Terminator

R 4D R 03 R 01 R 02 R 02 R 08

Controller → Module 0: GetModule

T 4D 82 00 00 82

Controller → Module 0: GetVersion

T 4D 82 00 01 83

Module 0: Response: len=0A, ID=00, ACK CMD=GetModule, 'DI6 V01'

R 4D R 0A R 00 R 02 R 00 R 44 R 49 R 36 R 20 R 56 R 30 R 31 R A7

Module 0: Response: len=1C, ID=00, ACK CMD=GetVersion, '0.2A Feb 10 2011 21:21:55'

R 4D R 1C R 00 R 02 R 01 R 30 R 2E R 32 R 41 R 20 R 46 R 65 R 62 R 20 R 31 R 30 R 20 R 32 R 30 R 31 R 31 R 20 R 32 R 31 R 3A R 32 R 31 R 3A R 35 R 35 R 4B

Controller → Module 1: GetModule

T 4D 82 01 00 83

Controller → Module 1: GetVersion

T 4D 82 01 01 84

Module 1: Response: len=0A, ID=01, ACK CMD=GetModule, 'DI6 V01'

R 4D R 0A R 01 R 02 R 00 R 44 R 49 R 36 R 20 R 56 R 30 R 31 R A8

Module 1: Response: len=1C, ID=01, ACK CMD=GetVersion, '0.2A Feb 10 2011 21:21:55'

R 4D R 1C R 01 R 02 R 01 R 30 R 2E R 32 R 41 R 20 R 46 R 65 R 62 R 20 R 31 R 30 R 20 R 32 R 30 R 31 R 31 R 20 R 32 R 31 R 3A R 32 R 31 R 3A R 35 R 35 R 4C

Controller → Module 0: GetPStat

T 4D 82 00 03 85

Module 0: Response: len=0C, ID=00, ACK CMD=GetPStat, T=0, Ovf=0, Chk=0, Str=0, Rcvd=8

R 4D R 0C R 00 R 02 R 03 R 00 R 00 R 00 R 0A R 00 R 00 R 00 R D7 R 01 R F3

Controller → Module 1: GetPStat

T 4D 82 01 03 86

Module 1: Response: len=0C, ID=01, ACK CMD=GetPStat, T=1, Ovf=0, Chk=0, Str=0, Rcvd=9

R 4D R 0C R 01 R 02 R 03 R 01 R 00 R 00 R 00 R 00 R 00 R 00 R E2 R 01 R F6

Controller → Module 0: GetInput(00)

T 4D C2 00 00 C2

Module 0: Response: len=03, ID=00, GetInput, 80

R 4D R 03 R 00 R 01 R 80 R 84

Controller → Module 0: GetInput(FF)

T 4D C2 00 FF C2

Module 0: Response: len=03, ID=00, GetInput, 00

R 4D R 03 R 00 R 01 R 10 R 14

Controller → Module 1: GetInput(00)

T 4D C2 01 00 C3

Module 1: Response: len=03, ID=01, GetInput, 80

R 4D R 03 R 01 R 01 R 80 R 85

Controller → Module 1: GetInput(FF)

T 4D C2 01 FF C3

Module 1: Response: len=03, ID=01, GetInput, 00

R 4D R 03 R 01 R 01 R 20 R 25

Controller → Module 0: SetOutput(FF, 10)

T 4D 84 00 50 FF 10 E4

Module 0: Response: len=04, ID=00, ACK CMD=50, 10

R 4D R 04 R 00 R 02 R 50 R 10 R 66

Controller → Module 1: SetOutput(FF, 20)

T 4D 84 01 50 FF 20 F5

Module 1: Response: len=04, ID=01, ACK CMD=50, 20

R 4D R 04 R 01 R 02 R 50 R 20 R 77

Controller → Module 0: GetInput(00)

T 4D C2 00 00 C2

Module 0: Response: len=03, ID=00, GetInput, 80

R 4D R 03 R 00 R 01 R 80 R 84

Controller → Module 0: GetInput(FF)

T 4D C2 00 FF C2
Module 0: Response: len=03, ID=00, GetInput, 10
R 4D R 03 R 00 R 01 R 10 R 14

Controller → Module 1: GetInput(00)
T 4D C2 01 00 C3
Module 1: Response: len=03, ID=01, GetInput, 80
R 4D R 03 R 01 R 01 R 80 R 85

Controller → Module 1: GetInput(FF)
T 4D C2 01 FF C3
Module 1: Response: len=03, ID=01, GetInput, 20
R 4D R 03 R 01 R 01 R 20 R 25

Controller → Module 0: GetPStat
T 4D 82 00 03 85
Module 0: Response: len=0C, ID=00, ACK CMD=GetPStat, T=0, Ovf=0, Chk=0, Str=0, Rcvd=20
R 4D R 0C R 00 R 02 R 03 R 00 R 00 R 00 R 0A R 00 R 00 R 00 R E3 R 01 R FF

Controller → Module 1: GetPStat
T 4D 82 01 03 86
Module 1: Response: len=0C, ID=01, ACK CMD=GetPStat, T=1, Ovf=0, Chk=0, Str=0, Rcvd=21
R 4D R 0C R 01 R 02 R 03 R 01 R 00 R 00 R 00 R 00 R 00 R 00 R EE R 01 R 03