**Application Control and Debug Protocol**

|  |  |
| --- | --- |
| Version | 1.0 |

Overview:

The Application Control and Debug Protocol (abbreviated as **ACDP**) is a medium speed protocol with physical layer based on UART protocol operating with one stop bit and no parity bit configuration and at a maximum baud rate of 115200 bauds per second. It consists of a single host node (PC) connected to a maximum of 16 device nodes via a full-duplex RS-485 bus. nodes Each data frame consists of 9 bytes irrespective of a frame type. Each byte in a message frame takes value from *0Bh* to *FFh* except for the last byte (byte 8 (0-indexing)) which takes a value of *0Ah*. The occurrence of *0Ah* indicates the end of a message frame. Since this protocol uses a fixed size data frame, the data packets can be handled easily by processors supporting UART DMA. The minimum data transmission time for each message frame is given by the equation:

For a baud rate of 115200, the minimum frame transmission time is

**NOTE: Frames / Packets are used interchangeably through the documentation and C/C++/LabVIEW APIs.**

Message frames:

This protocol uses the following message frames: connection management (CMF) frame, error status frame (ESF), data frame (DF), remote request frame (RRQF). The type of frame is identified by the frame type (FT) segment in the frame descriptor (FD) byte in the message frame. The frame descriptor consists of three segments: frame priority (FPRI), frame type (FT) and node ID (NID).

*Table 1: Frame descriptor byte*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BIT** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Name** | Frame  Priority  (**FPRI**) | Frame Type (**FT**) | | | Node ID (**NID**) | | | |

|  |  |
| --- | --- |
| **Segment** | **Description** |
| FPRI | Indicates the priority of the message frame. A higher priority means that the message frame / packet will be placed in front of the message queue in the receiving node. A lower priority means that the message frame received by the receiving node will be placed at the end of the device node’s receiving queue. |
| FT | Indicates the type of the message frame / packet. This protocol supports the following message frames:  0 – Data frame (DF).  1 - Connection Management frame (CMF).  2 – Remote request frame (RRQF).  3 – Error/status frame (ESF). |
| NID | Node ID of the device node to which the frame is sent to/received from. |

This protocol consists of two main frame / packet formats: un-processed/raw frame (bytes are prefixed by ***R***) and processed frame (bytes are prefixed by ***P***). The host and device(s) maintain the frames in raw format in their memory but send the frame in processed format over the network. Tables 2 and 3 shows the structure of a generalized raw frame and a processed frame respectively.

*Table 2: Generalized raw frame.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| R\_FD | R\_PB0 | R\_PB1 | R\_PB2 | R\_PB3 | R\_PB4 | R\_CS |

|  |  |
| --- | --- |
| **Byte** | **Description** |
| R\_FD | Frame descriptor |
| R\_PB0 | Frame Packet Byte 0 |
| R\_PB1 | Frame Packet Byte 1 |
| R\_PB2 | Frame Packet Byte 2 |
| R\_PB3 | Frame Packet Byte 3 |
| R\_PB4 | Frame Packet Byte 4 |
| R\_CS | Frame Checksum |

*Table 3: Generalized processed frame*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P\_CY | P\_FD | P\_PB0 | P\_PB1 | P\_PB2 | P\_PB3 | P\_PB4 | P\_CS | 0x0A |

|  |  |
| --- | --- |
| **Byte** | **Description** |
| R\_CY | Carry byte |
| P\_FD | Processed frame descriptor |
| P\_PB0 | Processed frame packet byte 0 |
| P\_PB1 | Processed frame packet byte 1 |
| P\_PB2 | Processed frame packet byte 2 |
| P\_PB3 | Processed frame packet byte 3 |
| P\_PB4 | Processed frame packet byte 4 |
| P\_CS | Processed checksum |
| 0x0A | End of frame byte (line feed character) |

The following equations describe the translation of a raw frame to a processed frame.

In the above equation, each ***i*** th byte of the raw frame is added with 0x0B and the carry is then stored in carry byte of the processed frame by left shifting the computed carry by ***i*** bits and ORing with the previous value of the processed carry byte. The above equation can be further simplified as:

The bytes following the carry bit are encoded according to the equation:

The C-code for translation of raw frame to processed frame is as follows.

uint16\_t carry = 0;

/\* Obtain carry byte by adding each byte of raw frame \*/

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[0] ) >> 8 ) << 0 );

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[1] ) >> 8 ) << 1 );

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[2] ) >> 8 ) << 2 );

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[3] ) >> 8 ) << 3 );

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[4] ) >> 8 ) << 4 );

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[5] ) >> 8 ) << 5 );

carry |= ( ( ( (uint16\_t)0x0B + (uint16\_t)raw->byte[6] ) >> 8 ) << 6 );

carry += 0x0B;

processed->byte[0] = carry; // Load carry byte.

/\* Compute other processed bytes \*/

for(int i = 1 ; i <= 7 ; i++)

{

/\* If byte value is greater than or equal to 0xFF-0x0A = 0xF5, then when 0x0B is added to it,

a carry is generated. In that case, first, the raw value must be added with 0x0B, the leftover

must be taken ( 0x00 <= leftover <= 0x0A ) and the leftover is added with 0x0B. \*/

if(raw->byte[i-1] >= 0xF5)

{

processed->byte[i] = (((uint16\_t)raw->byte[i-1] + (uint16\_t)0x0B) & (uint16\_t)0xFF) + 0x0B;

}

else

{

processed->byte[i] = (raw->byte[i-1] + 0x0B);

}

}

processed->byte[8] = 0x0A; // Termination byte (0x0A).

Similarly, a processed frame can be calculated using the following C-code.

void processed\_to\_raw(uint8\_t\* raw,uint8\_t\* processed)

{

/\* Calculate carry byte. \*/

uint8\_t carry = (processed[0] – 0x0B);

raw[0] = (processed[1] – 0x0B) + ( ( ( carry & ( 1 << 0 ) ) >> 0 ) \* 0xf5 );

raw[1] = (processed[2] – 0x0B) + ( ( ( carry & ( 1 << 1 ) ) >> 1 ) \* 0xf5 );

raw[2] = (processed[3] – 0x0B) + ( ( ( carry & ( 1 << 2 ) ) >> 2 ) \* 0xf5 );

raw[3] = (processed[4] – 0x0B) + ( ( ( carry & ( 1 << 3 ) ) >> 3 ) \* 0xf5 );

raw[4] = (processed[5] – 0x0B) + ( ( ( carry & ( 1 << 4 ) ) >> 4 ) \* 0xf5 );

raw[5] = (processed[6] – 0x0B) + ( ( ( carry & ( 1 << 5 ) ) >> 5 ) \* 0xf5 );

raw[6] = (processed[7] – 0x0B) + ( ( ( carry & ( 1 << 6 ) ) >> 6 ) \* 0xf5 );

}

Data Frame (DF)

This frame is sent by the target device node to the master node. The format of the data frame is shown in Table 4. Each data frame consists of 4 data bytes followed by a checksum byte.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bytes | 0 | | | 1 | 2 | 3 | 4 | 5 | 6 |
| Bit | 7 | 6-4 | 3-0 | 7-0 | 7-0 | 7-0 | 7-0 | 7-0 | 7-0 |
| Value/Symbol | PRI | HC(0) | NID | FID | DB3 | DB2 | DB1 | DB0 | CS |

|  |  |
| --- | --- |
| **Segment /Symbol** | **Description** |
| PRI | Priority of the data frame. If priority is set to 0, the data frame is stored at the end of the message queue of device node. If priority is set to 1, the data frame is stored at the beginning of the message queue of the device node. |
| NID | Target node ID. |
| FID | The ID of the data frame transmitted by the host/device node. |
| DB3 – DB0 | Data bytes from MSB to LSB |
| CS | Checksum of the entire frame |

Connection manager frame (**CMF**)

This frame is used to manage the connection between the host node and a device node. The connection manager frame structure is represented in Table 4. This frame is sent by the host node to the device node to request for opening/closing connection and vice-versa (device🡪host) to acknowledge the status of the request. This frame is also used to synchronize the connection between the host node and device node.

Table 4: Connection manager frame

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte** | 0 | | | 1 | | | | | | 2 |
| **Bit** | 7 | 6-4 | 3-0 | 5 | 4 | 3 | 2 | 1 | 0 | 7-0 |
| **Value/Symbol** | HC(1) | HC(1) | NID | SYNC\_  RQST | SYNC\_  ACK | OC\_  RQST | OC\_  ACK | CC\_  RQST | CC\_  ACK | RSVD(0) |
| **Byte** | 3 | | | 4 | | | | | | 5 |
| **Bit** | 7-0 | | | 7-0 | | | | | | 7-0 |
| **Value/Symbol** | RSVD(0) | | | RSVD(0) | | | | | | RSVD(0) |
| **Byte** | 6 | | |  | | | | | |  |
| **Bit** | 7-0 | | |  | | | | | |  |
| **Value/Symbol** | CS | | |  | | | | | |  |

**Note:** Values represented within () indicate hardcoded values.

|  |  |
| --- | --- |
| Segment/symbol name | Description |
| NID | ID of the node to be addressed. |
| SYNC\_RQST | The synchronization request bit is set to 1 by the host node and transmitted to the device node identified by NID. This bit will be set to 0 by the addressed device node once the request is processed and sent back to the host node. |
| SYNC\_ACK | The synchronization acknowledgement bit is set to 1 by the addressed device node once the synchronization request has been processed by it. |
| OC\_RQST | The open-connection request bit is set by the host node and sent to the target device node. This bit must be cleared by the target device node. |
| OC\_ACK | The open-connection acknowledge bit is set by the target device node and is sent back to the host node. |
| CC\_RQST | The close-connection request bit works in the same way as OC\_RQST but to close the connection with the target device node. |
| CC\_ACK | The close-connection acknowledgement bit works in the same way as OC\_ACK but to close the connection with the target device node. |
| CS | Checksum |

Remote Request Frame (RREQF)

This frame is sent from the host node to the target device node to request data frames of a specific frame ID(s). The structure of a remote request frame is described in Table 5.

*Table 5: Remote Request Frame structure*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bytes** | 0 | | | 1 | | |
| **Bits** | 7 | 6-4 | 3-0 | 7 | 6 | 5-0 |
| **Symbol** | HC(1) | HC(2) | NID | RMT\_REQ | RMT\_ACK | N\_FID |
| **Bytes** | 2 | | | 3 | | |
| **Bits** | 7-0 | | | 7-0 | | |
| **Symbol** | FID\_0 | | | FID\_1 | | |
| **Bytes** | 4 | | | 5 | | |
| **Bits** | 7-0 | | | 7-0 | | |
| **Symbol** | FID\_2 | | | FID\_3 | | |
| **Bytes** | 6 | | |  | | |
| **Bits** | 7-0 | | |  | | |
| **Symbol** | CS | | |  | | |

|  |  |
| --- | --- |
| **Segment / symbol name** | **Description** |
| NID | This represents the identifier of the device node from which the data frames have to be requested. |
| RMT\_REQ | This bit is set to 1 by the master node and then sent to the target device node. This bit is set to 0 by the target device node once it has received the remote request frame. |
| RMT\_ACK | This bit is set to 1 by the target device node once it has detected a remote frame request. This bit will be polled for by the master node and will be set to 0 once the target device has set this bit to 1. |
| N\_FID | This represents the number of frames requested from the target device node. The value of the number frames must not exceed 5 and be below 1. |
| FID\_0 | Requested frame number 0 from the target device node. |
| FID\_1 | Requested frame number 1 from the target device node. |
| FID\_2 | Requested frame number 2 from the target device node. |
| FID\_3 | Requested frame number 3 from the target device node. |
| CS | Checksum. |

Error/status frame (ESF)

This frame is transmitted by the target device node to the host node in-case of any application specific error or network/communication error. Common examples of communication error include checksum error, closed connection error, collision error etc. The structure of an error/status frame is given by Table 6.

Table 6: Error/status frame structure

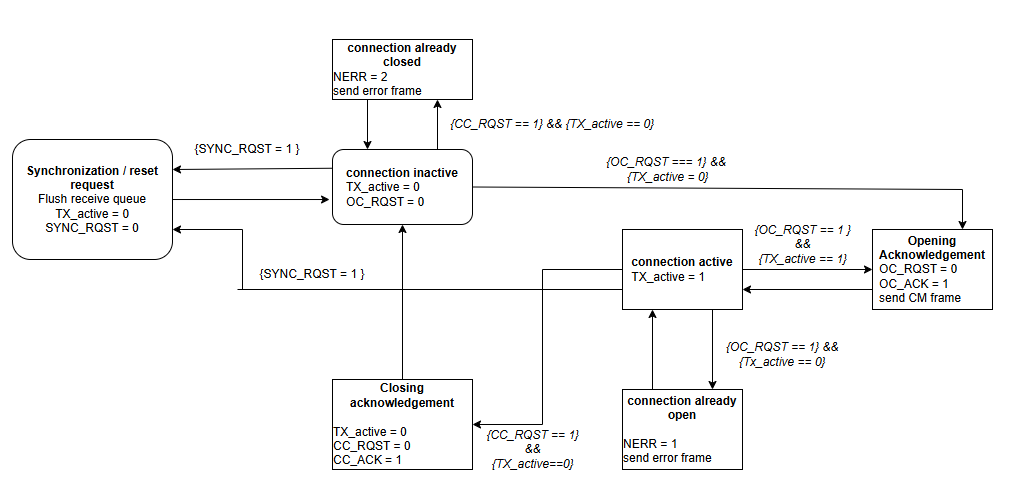
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bytes | 0 | | | 1 | | | | | 2 |
| Bits | 7 | 6-4 | 3-0 | 7-3 | 4 | 3 | 2 | 1-0 | 7-0 |
| Symbol | HC(1) | HC(3) | NID | RSVD(0) | DPWACK | SREQ | SACK | QSTAT | QSZ |
| Bytes | 3 | | | 4-5 | | | | | 6 |
| Bits | 7 | | | 7-0/7-0 | | | | | 7-0 |
| Symbol | NES | | | AERR | | | | | CS |

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| NID | Node ID of the device node to which the request is made to / is being sent from. |
| SREQ | Status request bit. This bit is set by the master node and is cleared by the target device node once the request has been processed. |
| SACK | Status request acknowledgement bit. This bit is set by the target device node once the request made by the host/master node has been processed. |
| QSTAT | Status of the target node’s receiver queue.  0🡪 Receiver queue OK  1🡪 Receiver queue empty  2 🡪 Receiver queue full |
| QSZ | Queue size. The number of elements in the device node’s receiver buffer. |
| NES | Network error/status byte. This byte indicates the error / status of the connection between a given device node and the host node.   1. 🡪 No error occured. 2. 🡪 A connection is active between the device node and the host node. 3. 🡪A connection between node and host already exists. 4. 🡪 Checksum verification of sent packet failed (NOT IMPLEMENTED). |
| AERR | Application specific error bytes. This field is 16 bits long. |
| CS | Checksum of entire frame. |
| DPWACK | Data Packet Write Acknowledgement. When the host transmits a data packet/frame to the target device node, the device node acknowledges by setting this bit and transmits the entire error & status packet back to the host. |

State machine diagram for connection management

This section describes the state machine diagram used to handle the connection between a device node and a host node. Figure 1 represents the state diagram for connection management from the device node’s end and Figure 2 represents the state diagram for connection management from the end of the host node.

*Figure 1: Connection Management state machine diagram for device node*



State machine diagram for remote frame request (device node)

A diagram of a flowchart

Description automatically generated

State machine diagram for error & status frame / packet (device node)

A white rectangular object with black text

Description automatically generated

**Revision history**

|  |  |
| --- | --- |
| Revision | Comments |
| 0.0 | Initial Revision |
| 0.1 | Added Data Packet Write Acknowledgement bit. |