UART Connection Orientated Communication Protocol

This document describes the software of the UART communication protocol for the EIVE PLOC project.

Content

[1. Introduction 4](#_Toc42868097)

[1.1. S 4](#_Toc42868098)

[1.2. Overview 4](#_Toc42868099)

[2. Packet Structure 5](#_Toc42868100)

[2.1. Header 5](#_Toc42868101)

[2.1.1. … ID 5](#_Toc42868102)

[2.1.2. CRC-8 6](#_Toc42868103)

[2.1.3. Data Size 6](#_Toc42868104)

[2.1.4. Flags 7](#_Toc42868105)

[3. Flow Charts 8](#_Toc42868106)

[3.1. Receiving data 8](#_Toc42868107)

[3.2. Transmitting Data 9](#_Toc42868109)

[4. Functions Overview 10](#_Toc42868110)

[4.1. Functions Provided By Xilinx 10](#_Toc42868111)

# **Introduction**

# **S**

# **Overview**

# **Packet Structure**

The FIFO of the UART controller present on the PS has a maximum size of 64 bytes, meaning 32 bytes per direction.

Since the FIFO allowed the packet size to be a maximum of 32 bytes, only a few bytes could be used for the header, so only the most important information could be included there. Here 4 bytes were used for the header, so that still 28 bytes of the packet can be used for the actual user data.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Figure : Packet structure

# **Header**

It is important for the other partner to know to which subsystem the commands are addressed or from which subsystem the data comes, whether the transmission of the packet was without errors and how many valid bytes are in the packet. Various status information about the connection-oriented mode of operation is also necessary.

The following information is transmitted in the header of the packets:

* .. ID
* CRC-8
* Data Size
* Flags

# **… ID**

Two identification numbers are used for each subsystem, one for TCs and one for TMs. This way the respective other side can be informed who is to be addressed or from whom the user data comes.

Currently, the following subsystems each have two IDs:

* Camera
  + TC: 0b00000000 (hex: 00, dec: 0)
  + TM: 0b11110000 (hex: F0, dec: 240)
* BRAM
  + TC: 0b00000101 (hex: 05, dec: 5)
  + TM: 0b11110101 (hex: F5, dec: 245)
* Downlink
  + TC: 0b00001001 (hex: 09, dec: 9)
  + TM: 0b11111001 (hex: F9, dec: 249)
* UART
  + TC: 0b00001010 (hex: 0A, dec: 10)
  + TM: 0b11111010 (hex: FA, dec: 250)
* CPU
  + TC: 0b00001111 (hex: 0F, dec: 15)
  + TM: 0b11111111 (hex: FF, dec: 255)
* DAC
  + TC: 0b00000110 (hex: 06, dec: 6)
  + TM: 0b11110110 (hex: F6, dec: 246)

# **CRC-8**

As error detection for the packets a Cyclic Redundancy Check algorithm was taken.

Since only 1 byte of the header is to be used for the CRC value, a CRC-8 algorithm with the name CRC8\_SAE\_J1850\_ZERO was used in the MSB version. This algorithm is characterized by a generator polynomial 0x1D and an initial and final XOR value of 0x00.

To check the correct course of the packets, the CRC-8 algorithm was slightly modified instead of a sequence number. The last received or last sent value is taken as the new initial value of the CRC calculation in the modified mode of operation, so that the correct sequence can be checked.

The following is a sample calculation of the CRC algorithm:

# **Data Size**

For commands or user data that are either smaller than or not an exact multiple of 28 bytes, it is necessary to specify the number of valid bytes in the data field. Therefore, the valid size of the data field is also passed in the header as additional information.

As example a command of the length 1.3kByte:

Here you can see that in the last packet only 12 bytes are valid instead of 28 bytes.

# **Flags**

For various status information that can be signaled with one bit, one byte is occupied for so-called flags. Currently only 6 of these are in use.

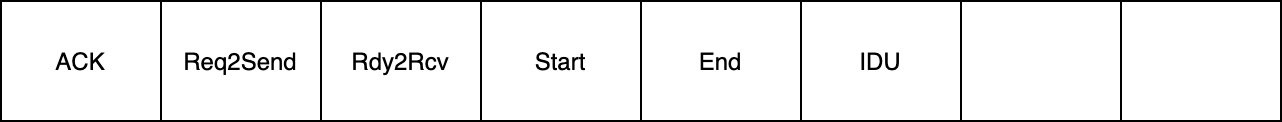


Figure :

Figure 2: Order of the flags

The flags are required for establishing the connection, they also signal the first and the last packet to be sent or the positive or negative acknowledgement of the received packet. In the following the flags are explained:

* **ACK:** This bit confirms the correctness of the last received packet. It is obtained using the CRC-8 algorithm.
* **Request to send (Req2Send):** The bit is set if one of the two devices wants to establish a connection for transmitting payload data.
* **Ready to Receive (Rdy2Rcv):** The bit is set to confirm the connection establishment. It can only be set once a connection request has been made.
* **Start:** The bit is set if it concerns the first packet of payload data to be transferred.
* **End:** The bit is set if it concerns the last packet of payload data to be transferred.
* **ID unknown (IDU):** The bit is set if the received ID does not exist. The ID is checked during the connection setup.

# **Flow Charts**

In order to better understand the flow of the respective processes, both the sending of data and the receiving of data were graphically represented in a flow chart. The reception process is divided into 2 sections: The connection establishment and the reception or transmission of the packets.

# **Receiving data**

# Ein Bild, das Text, Whiteboard enthält. Automatisch generierte Beschreibung

Figure 3: Flow chart for receiving data

# **Transmitting Data**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 4: Flow chart for transmitting data

# **Functions Overview**

# **Header *UART\_EIVE\_Protocol* for sending protocol**

# **int** UART\_Send\_Data(uint8\_t ID, uint8\_t \*databytes, **int** dataLength);

* Description: Main method of the EIVE UART protocol to send data. At first, this method uses the *connect\_()*-method to establish a connection between the transmitter and the receiver. Afterwards, once a connection was established, it uses the method *send\_data()* to send the transferred data to the receiver. It is the main method, which is called by the OBC or the PLOC to send the data
* Parameters:
  + **ID**: Identification number of the subsystem to which it refers
  + **databytes**: pointer to the data which is going to be send
  + **dataLength**: length of the data which is going to be send (number of bytes)
* Return value:
  + **XST\_SUCCESS**: if the data was sent properly
  + **XST\_FAILURE**: if the data was not sent properly
* Notes:

**int** connect\_(uint8\_t ID, uint8\_t \*databytes, uint8\_t dataLength, uint8\_t \*lastCRC\_send, uint8\_t \*lastCRC\_rcvd);

* Description: This method is used to establish a connection between the transmitter and the receiver
* Parameters:
  + **ID**: Identification number of the subsystem to which it refers
  + **databytes**: pointer to the data which is going to be send
  + **dataLength**: length of the data which is going to be send (number of bytes)
  + **lastCRC\_send**: pointer to the last CRC value, which was sent
  + **lastCRC\_rcvd**: pointer to the last CRC value, which was received
* Return value:
  + **XST\_SUCCESS**: if the connection was established properly
  + **XST\_FAILURE**: if the connection was not established properly
* Notes:

**int** send\_request\_to\_send(uint8\_t ID, uint8\_t \*temp32, uint8\_t \*lastCRC\_send, uint8\_t \*flags);

* Description: This method sends a connection request to the receiver to establish a connection
* Parameters:
  + **ID**: Identification number of the subsystem to which it refers
  + **temp32**: pointer to an array, which is located in the method *connect\_()*, and is used to buffer an entire package
  + **lastCRC\_send**: pointer to the last CRC value, which was sent
  + **flags**: pointer to the flags which are going to be send in the connection request
* Return value:
  + **XST\_SUCCESS**: if the connection request was sent properly
  + **XST\_FAILURE**: if the connection request was not sent properly
* Notes:

**int** package\_count(**int** dataLength);

* Description: Method for counting the number of packets from the transferred data length
* Parameters:
  + **dataLength**: length of the data which is going to be send (number of bytes)
* Return value: It returns the number of packages needed to transfer all the data
* Notes:

**void** get\_received\_data(uint8\_t \*header, uint8\_t \*data, uint8\_t \*flags, uint8\_t \*submittedCRC);

* Description: Method to save the submitted data and splitting it into header (CRC and flags) and data
* Parameters:
  + **header**: pointer to the array in which the submitted header from the receiver is going to be buffered
  + **data**: pointer to the array in which the submitted data from the receiver is going to be buffered
  + **flags**: pointer to the array in which the submitted flags from the receiver are going to be buffered
  + **submittedCRC**: pointer to a variable in which the submitted CRC from the receiver value is going to be buffered
* Notes:Since this is a connection-oriented protocol, it is necessary to wait for a response after each transmission. This occurs both when the connection is established and when the packages containing the data are sent. Accordingly, this method is required for these two functions and therefore the arrays are located once in the method *connect()* and another time in the method *send\_data().*

**int** send\_data(uint8\_t ID, uint8\_t \*databytes, **int** dataLength, uint8\_t \*lastCRC\_send, uint8\_t \*lastCRC\_rcvd);

* Description: Method which is used to split the data to be sent into packets and to send the packages themselves. For this purpose, the method *UART\_Send()* provided by Xilinx is used (see next chapter, functions provided by Xilinx)
* Parameters:
  + **ID**: Identification number of the subsystem to which it refers
  + **databytes**: pointer to the data which is going to be send
  + **dataLength**: length of the data which is going to be send (number of bytes)
  + **lastCRC\_send**: pointer to the last CRC value, which was sent
  + **lastCRC\_rcvd**: pointer to the last CRC value, which was received
* Return value:
  + **XST\_SUCCESS**: if the sending of the data was successful
  + **XST\_FAILURE**: if the sending of the data was not successful
* Notes:

**int** wait\_on\_answer(uint8\_t \*send\_array, uint8\_t ID, uint8\_t \*lastCRC\_send);

* Description: Method for waiting for the response of the receiver
* Parameters:
  + **send\_array**: pointer to the array containing the data to be sent
  + **ID**: Identification number of the subsystem to which it refers
  + **lastCRC\_send**: pointer to the last CRC value, which was sent
* Return value:
  + **XST\_SUCCESS**: if an answer was received
  + **XST\_FAILURE**: if an answer was not received
* Notes: This method regularly checks the Receive Buffer. To ensure that the protocol runs continuously, a counter is incremented. If the counter reaches its maximum value without filling the receive buffer, the desired package is sent to the receiver again. Another counter is responsible for counting the number of transmissions of the same package to the receiver. If this counter also reaches its maximum value, a failure is sent to the recipient and the method also returns a failure. If the buffer is filled, the method returns a success, even without the timers reaching their maximum values.

**void** fill\_packages(uint8\_t ID, **int** dataLength, uint8\_t \*databytes, uint8\_t \*temp, **int** packageCount);

* Description: This method is used to fill the packages with the submitted information. The pointer to the submitted value temp is used to buffer the temporary packages
* Parameters:
  + **ID**: Identification number of the subsystem to which it refers
  + **dataLength**: length of the data which is going to be send (number of bytes)
  + **databytes**: pointer to the data which is going to be send
  + **temp**: pointer to an array, which is located in the method *send\_data()* and is used to buffer the packets with the corresponding header and data, which are going to be sent
* Notes:

uint8\_t fill\_header\_for\_empty\_data(uint8\_t \*header, uint8\_t ID, uint8\_t flags, uint8\_t \*lastCRC\_send);

* Description: Method for filling the header of packets which are not to transmit data
* Parameters:
  + **header**: pointer to the array in which the submitted information is going to be buffered
  + **ID**: Identification number of the subsystem to which it refers
  + **flags**: array containing the flags which are filed into the header
  + **lastCRC\_send**: pointer to the last CRC value, which was sent
* Return value: The method returns the CRC value calculated for this package to be sent
* Notes: The method is used to fill the header when empty data packages are to be sent. It is used while waiting for a response if the previously received package was not acknowledged. It is also used while sending a success or a failure, because here only the ACK flag is relevant

# **Header UART\_EIVE\_Protocol for receiving protocol**

**int** UART\_Recv\_Data();

* Description: Main method for receiving data if its available. The method *UART\_Recv\_Buffer()* provided by Xilinx is used to fill the receive buffer. Furthermore the method *receive()* is used to analyse the received information and to generate a corresponding response for the transmitter
* Return value:
  + **XST\_SUCCESS**: If the receiving process has been completed successfully
  + **XST\_FAILURE**: If the receiving process has not been completed successfully
  + **XST\_NO\_DATA:** if no data is available in the receiving buffer
* Notes:

**int** receive();

* Description: This method is called in the main method; it is used to receive the data. For this purpose, it implements the algorithm for receiving data
* Return value:
  + **XST\_SUCCESS**: If the receiving was correct
  + **XST\_FAILURE**: If the receiving was not correct
* Notes: As indicated above, in this method the protocol for receiving data is implemented, which requires a connection to the transmitter. A connection is established to receive packages longer than 28 bytes. First, it is indicated that some data is to be transmitted, to which it responds that data can be received. Then the actual data transmission takes place, where each received package is acknowledged positively or negatively

**int** connection\_establishment(uint8\_t \*last\_crc\_rcv, uint8\_t \*last\_crc\_send, uint8\_t \*new\_flags, uint8\_t \*conn\_id, uint8\_t \*calc\_crc);

* Description: Method to connect with the transmitter. It checks the received data from the transmitter whether the request to send flag is set and answers with a package where the ready to receive flag is set
* Parameters:
  + **last\_crc\_rcv:** pointer to the last CRC value, which was received
  + **last\_crc\_send:** pointer to the last CRC value, which was sent
  + **new\_flags:** pointer to the flags, which are going to be sent
  + **conn\_id:** pointer to the identification number for the connection establishment
  + **calc\_crc:** pointer to the variable located in the method *recevie()*, to store the calculated CRC
* Return value:
  + **XST\_SUCCESS**: If the connection was established correctly
  + **XST\_FAILURE**: If the connection was not established correctly
* Notes:

**int** receive\_data(uint8\_t \*crc\_rcv, uint8\_t \*crc\_send, uint8\_t rcvd\_id, uint8\_t last\_sent\_flags, uint8\_t \*calc\_crc);

* Description: this method is used to receive data from the connected transmitter
* Parameters:
  + **crc\_rcv**: pointer to the last received CRC value
  + **crc\_send:** pointer to the last send CRC value
  + **last\_sent\_flags:** pointer to the last sent flags
* Return value:
  + **XST\_SUCCESS**: If the data was successful received and written in the receiving buffer
  + **XST\_FAILURE**: If the data was not successful received
* Notes:

**int** extract\_header(**const** uint8\_t \*rcvBuffer, uint8\_t \*header, uint8\_t \*data);

* Description: this method splits the received data into header and payload data
* Parameters:
  + **rcvBuffer**: pointer to the buffer with received data
  + **header**: pointer to the header array to store the extracted header
  + **data**: pointer to the data array to store the extracted header
* Return value:
  + **XST\_SUCCESS**: If the received header and data were successfully written in the corresponding arrays
  + **XST\_FAILURE**: If the received header and data were not successfully written in the corresponding arrays
* Notes: Although this method is called extract header, its functionality has been extended and the data is also extracted from the receive buffer and written to the corresponding arrays

**int** check\_ID(uint8\_t ID);

* Description: Method to check if the incoming packages have a known or unknown ID
* Parameters:
  + **ID**: Identification number of the incoming package
* Return value:
  + **1:** If the ID is known
  + **0:** If the ID is not known
* Notes:

**int** send\_failure(uint8\_t \*last\_crc, uint8\_t old\_id, uint8\_t \*calc\_crc, **int** id\_unknown);

* Description:
* Parameters:
* Return value:
* Notes:

**int** send\_success(uint8\_t \*last\_crc, uint8\_t **id**, uint8\_t flags, uint8\_t \*calc\_crc);

* Description:
* Parameters:
* Return value:
* Notes:

**int** UART\_answer(uint8\_t \*header);

* Description:
* Parameters:
* Return value:
* Notes:

**int** recv\_TC(uint8\_t \*header, uint8\_t \*databytes, **int** size\_of\_data);

* Description:
* Parameters:
* Return value:
* Notes:

**int** recv\_TM();

* Description:
* Parameters:
* Return value:
* Notes:

**void** default\_operation();

* Description:
* Parameters:
* Return value:
* Notes:

# **Functions Provided By Xilinx**