Chapter 82

Using the AXN/UBM/401

TEC/NOT/088

This paper discusses the following topics:

82.1 RS-232

RS-232 is a single-ended physical layer where the data line is referenced to the signal ground. To overcome noise and signal degradation the line swings between +/- 12V with respect to GND. With RS-232 as with most serial protocols, an idle line is always high, which in RS-232 is classed as -12V (logical one) while a low (logical zero) is classed at +12V, that is, the logical and voltage sense are inverted.The receiver accepts lower than +/-12V. RS-232 is often used for lower data rates, also known as baud rates, typically from 9600 bits per second to 115200 bps.

82.2 RS-422/485

RS-422 is a differential serial bus where two data lines are used—sometimes called Data+, Data- or DataA, DataB—which switch between 0 to 3.3 volts or 0 to 5 volts. A one is when A is high and B is low while a zero is when B is high and A is low. Because the signals are differential more noise can be tolerated and they can also accept a small common mode voltage, however, a common ground is still required. RS-422 operates at a higher data rate than RS-232 and because of this signal reflections can occur if the bus not properly terminated by a 120-ohm resistor at the receiver end. RS-422 operates as a point-to-point signal where data only ever flows in one direction. RS-485 shares the same electrical voltage as RS-422 but is a bi-directional bus where data can flow both directions on the same wires with appropriate handshaking. Because any node on bus can drive it, termination is required at both ends of the bus. RS-422 operates over the same rates as RS-232 but is also suited to higher data rates and harsher environments. The AXN/UBM/401 can receive data at up to 10 Mbps.

82.3 Universal asynchronous receiver-transmitter protocol (UART)

Universal asynchronous receiver-transmitter is a common protocol that sends serial data between computers and devices. An idle line is always considered high. Transmission is initiated by driving the bus low using what is known as start a bit. This start bit is followed by eight or seven bits of data, where the data is either sent Most Significant Bit (MSB) first or last. An optional parity bit can also be sent using either odd or even parity. Odd parity is where the number of 1s including the parity bit is an odd number. Transmission ends with the bus being driven high.

82.4 Serial messages

Serial messages can be made up of a single byte or a group or string of bytes that may contain binary data or text data. They might start with a fixed set of data, such as GPS NMEA navigation data that always begins the ASCII code for $GP and might end with a fixed character or gap in transmission of one or more bytes.

82.5 Module overview

The AXN/UBM/401 is a 16-channel RS-232, RS-422, RS-485 which can parse and/or packetize each channel at the same time.

You can select the Signal Type (RS-232, RS-422, RS485), Baud Rate (0.3-10 Mbps), Data Bits Per Word (7/8 bits per word), and Parity (None, Even, Odd) on a channel-by-channel basis.

Screen shots and descriptions of settings shown in this technical note are from DAS Studio 3 software. DAS Studio 3 is used to create a configuration, which contains the various elements that make up your data acquisition system. You then use this configuration file to manage and program these elements.

To see how hardware is represented in the DAS Studio 3 graphical user interface, refer to the *DAS Studio 3 User Manual*.

82.5.1 Key features

- Monitors up to sixteen independent input RS-422/485/232 busses
- Bit-rates from 300 bps to 10,000,000 bps
- 7/8 bits per word with odd, even or no parity
- Programmable start sequence (1 to 8 bytes), stop sequence (1 byte or by fixed length) and message gap (idle time)
- Coherently parses traffic and tags for up to 1,024 messages per module. Each message can be 4 to 1024 bytes long
- Aperiodic transmission of packetized serial messages including tags as iNET-X parser-aligned payload structure or Chapter 10 UART Data Packets
- Message wide stale and skipped indication

82.6 Parser operation

82.6.1 How parsing works

Like other Curtiss-Wright bus monitors, the AXN/UBM/401 uses a triple buffer for parsing. The following figure illustrates the triple buffering of data words (green) and time message tags (white) used for each bus in the AXN/UBM/401's parser.

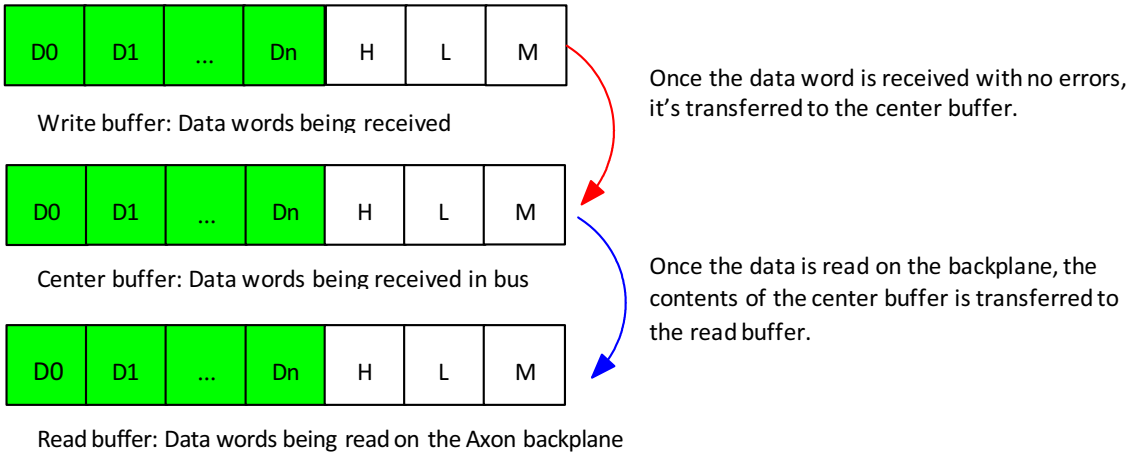


Figure 82-1: Triple buffering of traffic and associated message tags

In the previous figure, D0, D1, D2, Dn corresponds to the traffic data words with n < 512.

The time tags H, L, M correspond to High time, Low time and Micro time, which is the time of the first transmitted bit of the message with a 1-µsec resolution.

The way triple buffering works is as follows:

Time message tags are added to each message received and stored in separate buffers for each of the four busses. As soon as a message is received with no errors, the contents of the write buffer is transferred to the center buffer. If the data in the center buffer has not been transferred to a read buffer, a skipped flag is set.

As soon as the last parameter of interest has been read from the buffer being read by the backplane, the contents of the center buffer (if new) are transferred to the read buffer. If no new data word has been received, the stale flag is set. A center and read buffer exist for every message ID (parser slot). Skipped and stale bits can be found in the Message Info register to indicate whether messages are lost or repeated (undersampling or oversampling situations).

Additional tags such as Message Count, Message Size, and Message Info registers are also available as additional information and can be added from DAS Studio's Serial Builder application as explained in "82.8.5 Adding parameters to the package" on page 8. For further information regarding these registers, refer to the *AXN/UBM/401* data sheet.

82.7 Module Settings tab

82.7.1 Parser Data Endianness

In the parser, a total of up to 1024 complete messages are triple buffered so that the stale indication is message-wide. Each message can be up to 1024 characters (bytes) long (including start and stop characters). Each message is tagged to 1 us resolution; a message is considered found when a start sequence up to 8 user-defined characters are received. The end-of-message delimiter is determined by either a user-defined stop character or a specific number of bytes. A parsed message is not updated/parsed if the start sequence does not match.

NOTE: To view the screen shots shown in this section in DAS Studio 3, ensure the AXN/UBM/401 module is in context and the Settings tab is selected.

To configure the parser, first you must decide on the endianness of the data you wish to receive.

As shown in the following figure, there are two choices available for Parser Data Endianness.

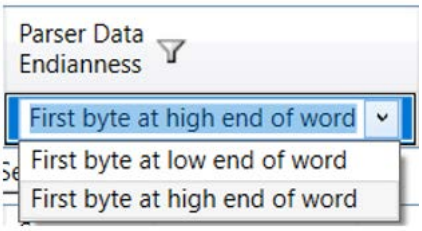


Figure 82-2: Parser Data Endianness settings

Considering the input data ABCD:

First byte at high end of word returns the hex values 0x4142, 0x4344, which corresponds to the decimal values 65, 66, 67, and 68, the ASCII codes for ABCD.

First byte at low end of word returns the hex values 0x4241 0x4443, which corresponds to the decimal values 66, 65, 68, and 67, the ASCII codes for BADC.

82.7.2 Setting up the incoming signal

In this section you define the signal type of the incoming data for that channel.

As shown in the following figure, there are three choices available for Signal Type.

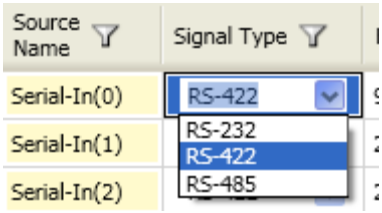


Figure 82-3: Signal Type settings on channel 0

RS-232 is a single ended input signal. For this mode, the data source must be connected to the positive input pin of the channel pair, while the negative input is connected to ground.

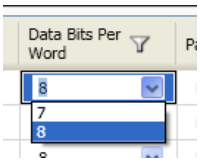
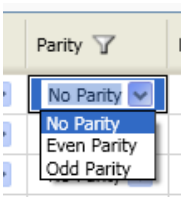
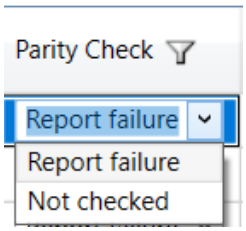
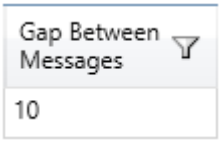
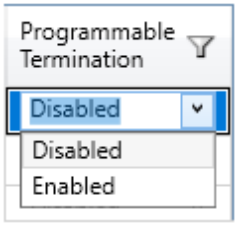
RS-422 and RS-485 are both differential inputs, so the source positive signal must be connected to the positive input, while the source negative input must be connected to the negative input.

For each input pair, there are individual termination pins. To use these, the negative input must be connected to the correct termination pin for that input.

The remaining channel settings to be configured for the parser are Baud Rate, Data Bits Per Word, and Parity as shown in the following figure and the table that follows.

Source Name	Signal Type	Baud Rate	Data Bits Per Word	Parity	Parity Check	Gap Between Messages	Programmable Termination
Serial-In(0)	RS-232	9600	8	No Parity	Not checked	10	Disabled

Figure 82-4: Baud Rate, Data Bits Per Word, and Parity settings

Setting	Description
Baud Rate (or bit rate)	Specifies the number of symbols (0s or 1s) transmitted per second. The AXN/UBM/401 can coherently parse messages at any bit rate in the range 300 bps to 10 Mbps.
Data Bits Per Word 	Can be either 7 or 8 bits. This does not include Start, Stop, or Parity bits.
Parity 	Parity configures whether a parity bit is present in incoming data. It can be set to No Parity, Even Parity, or Odd Parity. **No Parity** means no parity bit is expected to be present in the incoming data words. **Even Parity** means a parity bit is expected to be present at the end of the incoming data word. This bit is set to either 1 or 0 so that the total number of 1s in the word adds up to an even number. Similarly, **Odd Parity** means this bit is set to either 1 or 0 so that the total number of 1s in the word adds up to an odd number.
Parity Check 	Parity error detection can be enabled by choosing an option from the Parity Check drop-down list as shown in the following figure. The following two Parity Check options are available: • **Not checked** results in all parity errors being ignored. • **Report failure** results in a parity error being indicated in the parser block header and Report Word parameter.
Gap Between Messages 	This represents the time gap (represented in characters) between consecutive characters required before starting a new message. The value is expressed in units of character periods at the configured bit rate. This setting can be used as both a packetizer and parser. When using as a packetizer, set to 0 to ignore all gaps and create a message filled with all incoming messages. Refer to “82.9 Packetizer operation” on page 10 for more information. When using as a parser, the message gap can be used to parse messages which do not have a start sequence and/or the start sequence appears within the message. Refer to “82.11.3 Message gap” on page 14 for more information.
Programmable Termination 	This is an optional setting to enable an internal 120-ohm termination resistance. This termination is only active when the module power is powered on. Use wiring selectable termination instead if termination is required at all times. Note: Implementing wiring termination and enabling programmable termination by mistake reduces the value of the termination resistor by half, making termination less efficient. Refer to “82.11.1 Termination” on page 13 for more information.

82.7.3 Global parameters settings

Like most bus monitors, the AXN/UBM/401 has global parameters, which can be used for debugging. The main global parameters are Report word and several counters such as module, message, error and byte counters as described in the following table.

Source Name	Parameter Type	Parameter Name
MyAXN_UBM_401	Report	***P_MyAXN_UBM_401_Report***
MyAXN_UBM_401	ModuleMessageCount	***P_MyAXN_UBM_401_ModuleMessageCount***
Serial-In(0)	ChannelMessageCount	***P_MyAXN_UBM_401_Serial-In(0)_ChannelMessageCount***
Serial-In(0)	ChannelByteCount	***P_MyAXN_UBM_401_Serial-In(0)_ChannelByteCount***
Serial-In(0)	ChannelErrCount	***P_MyAXN_UBM_401_Serial-In(0)_ChannelErrCount***

Figure 82-5: Global parameters in DAS Studio 3 Settings tab

Setting	Description
Report	The Report word is a 16-bit register, which provides information regarding errors detected in the card and the bus. The Report word is recommended to be monitored as a debug register when abnormal conditions are detected. Refer to the *AXN/UBM/401* data sheet and “82.7.3 Global parameters settings” on page 5 for further information.
ModuleMessageCount	ModuleMessageCount increments by one each time the parser logic detects a complete message. Note that the module does not know which bytes form a message until it is parsed. This register counts how many messages the module has parsed by any of the channels.
ChannelMessageCount (0 to 15)	Increments by one each time the parser logic detects a complete message. Note that messages which are not defined in the parser are just considered bytes, which can be tracked with ChannelByteCount. Note: Do not confuse this register with MessageCount, which corresponds to a counter added to each message (see “82.8.5 Adding parameters to the package” on page 8).
ChannelByteCount (0 to 15)	Count the overall bytes received on this bus, regardless whether they are defined in the parser or not.
ChannelErrCount (0 to 15)	Count of errors detected on this bus. See “82.7.4 Errors” on page 5 for error definitions.
ModuleTemperature	Temperature of the AXN/UBM/401 module. Refer to the *AXN/UBM/401* data sheet.

82.7.4 Errors

There are several errors reported by the AXN/UBM/401. As explained in the previous section, the Report word provides an indicator of the different errors detected in any of the busses, and ChannelErrCount provides an indicator of the number of errors detected on each bus. Additional packetizer headers contains an error flag Er (1 bit) and a 6 bits Error Code indicator.

The supported errors are:

- **Parity Error**: value 0x1 - This error can be ignored if Parity Check is set to Not checked.
- **Bad Stop Bit**: value 0x2 - Refer to “82.11.5 Bad Stop Bit” on page 15.
- **Too many data words**: value 0x4 - Stop character has not been found in 1024 characters. This error is not reported in the packetizer error code.

82.8 Serial Builder

For the following sections, use the Serial Builder application in DAS Studio 3. Refer to the “Builder application GUI overview” section of the *DAS Studio 3 User Manual* for a brief overview of navigating the application.

“82.8.1 Defining parsing rules” on page 6

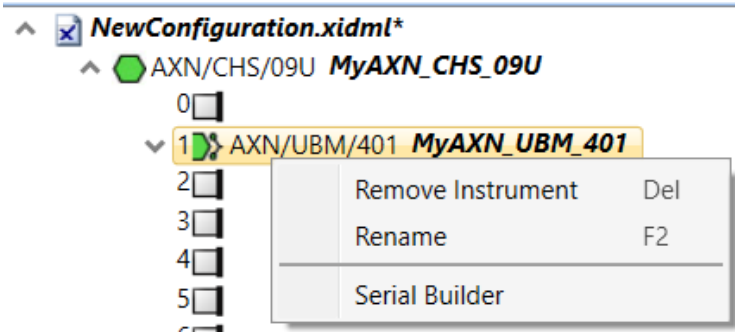
“82.8.2 Parsing Mode” on page 7

“82.8.3 Start/Stop Sequence format” on page 7

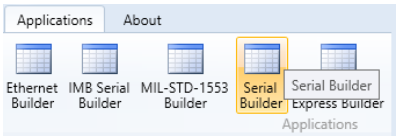
82.8.1 Defining parsing rules

After you have all channel settings configured, refer to the following to define rules to identify messages.

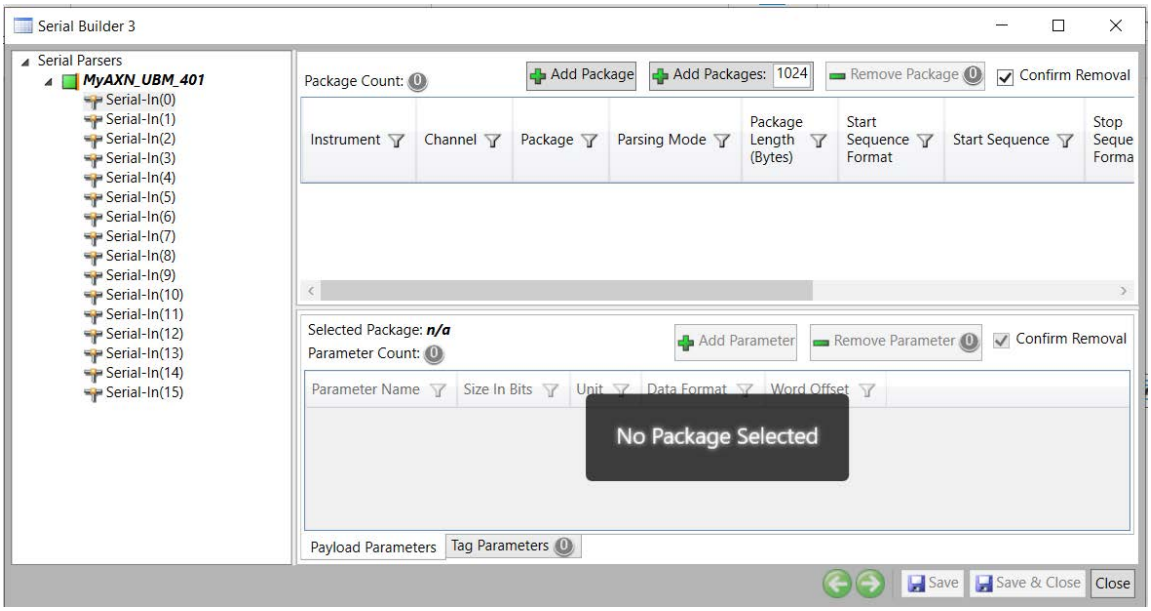
1. Do one of the following.
   - In the Navigator, right-click the AXN/UBM/401 module and then click **Serial Builder**.



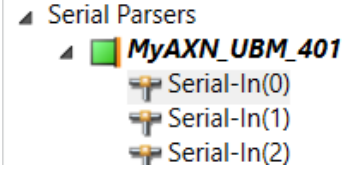
   - On the **Applications** tab click **Serial Builder**.



   The **Serial Builder** application opens.



2. In the Navigator (left pane), select the channel on the AXN/UBM/401 that you want to parse data off.

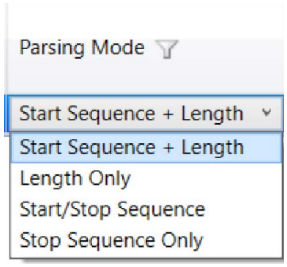


3. Click the **Add Package** button to add a single package. To add multiple packages (up to 1024), click the **Add Packages** button (typing the number of packages in the field).

82.8.2 Parsing Mode

Now you must define the rules to identify or parse the desired message.

4. The first rule to define is **Parsing Mode**. In the **Parsing Mode** field, open the drop-down list.



Choose **Start Sequence+Length** if you know the start pattern of the message and the number of bytes of data that make up the entire message.

Choose **Length Only** if there is no unique start sequence and you know the number of bytes of the entire message. A message gap needs to be known for it. Refer to “82.11.3 Message gap” on page 14 for further information.

Choose **Start/Stop Sequence** if you know a fixed sequence of 1 to 8 characters that uniquely identify this message, and the stop character that indicates the end of the message.

Choose **Stop Sequence Only** if there is no unique start sequence and you only know the stop character that indicates the end of the message. A message gap will need to be known for it. Refer to “82.11.3 Message gap” on page 14 for further information.

When **Start/Stop Sequence** is chosen, the **Package Length** field is not available and the relevant **Start/Stop Sequence** fields are made available.

82.8.3 Start/Stop Sequence format

Now you define the **Start Sequence Format** as **ASCII**, **Hex** or **Binary**.

The **Start Sequence** can be up to eight characters long. Binary and Hex can be used when wildcards are required to identify the message. See “82.11.4 Wildcard in Start Sequence” on page 15 for more information.

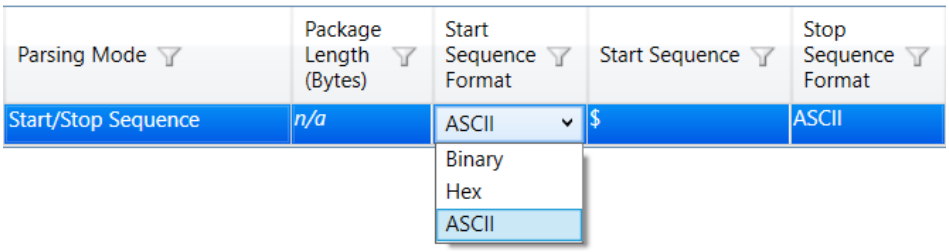


Figure 82-6: Start Sequence Format options

A common start sequence example is the one used with NMEA messages such as ASCII characters $GPSZDA.

Stop Sequence is a single byte used to identify the end of the message. By default, this is the ASCII character for line feed (\l).

A typical NMEA message ends with ASCII line feed (0xD) and stop sequence of ASCII Carriage Return (0x0A).

Similarly when **Stop Sequence Only** is selected, only the **Stop Sequence** field is available.

When **Start Sequence+Length** is chosen, the **Package Length** and **Start Sequence** fields are made available and the **Stop Sequence** field is not available.

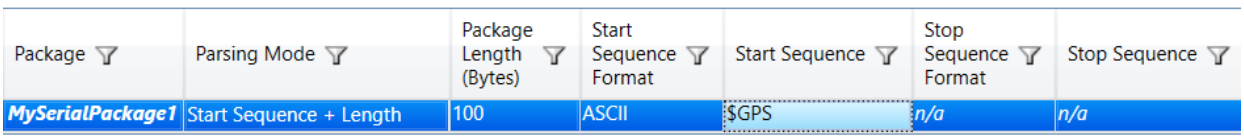


Figure 82-7: Start Sequence + length option

Similarly when **Length Only** is selected, only the **Package Length** field is available.

Instrument	Channel	Package	Parsing Mode	Package Length (Bytes)	Start Sequence Format	Start Sequence	Stop Sequence Format	Stop Sequence	Packetizer Filter Rule Enabled	Packetizer Filter Mode	Payload Count
MyAXN_UBM_401	Serial-In(0)	***MySerialPackage***	Start Sequence + Length	10	ASCII	$	*n/a*	*n/a*	☐	Pass All	0
MyAXN_UBM_401	Serial-In(0)	***MySerialPackage1***	Length Only	12	*n/a*	*n/a*	*n/a*	*n/a*	☐	Pass All	0
MyAXN_UBM_401	Serial-In(0)	***MySerialPackage2***	Start/Stop Sequence	*n/a*	ASCII	$GPZDA	ASCII	\n	☐	Pass All	0
MyAXN_UBM_401	Serial-In(0)	***MySerialPackage3***	Stop Sequence Only	*n/a*	*n/a*	*n/a*	ASCII	A	☐	Pass All	0

Figure 82-8: Example of the setup of each message type available in the AXN/UBM/401

82.8.4 Packetizer rule per message

The module supports filtered packetizer per channel. To configure the messages you want to filter you need to set the two fields: **Packetizer Filter Rule Enabled** and **Packetizer Filter Mode**.

Select the **Packetizer Filter Rule Enabled** check box to allow the message to be either **Blocked** or **Passed** in the packetizer (depending on the channel **Packetizer Filter Mode** setting in the **Settings** tab) when they meet the packetizer filtering condition referenced by this process.

NOTE: **Packetizer Filter Rule Enabled** and **Packetizer Filter Mode** are only applicable when **Packetization** is enabled.

Packetizer Filter Mode can then be specified:

Block By Rule means that messages that match defined filter rules are blocked and all other messages are passed through in the packetizer.

Pass By Rule means that messages that match defined filter rules are passed through and all other messages are blocked in the packetizer.

Pass All means that all messages are passed through the packetizer.

Packetizer Filter Mode is a channel setting; this field must be set to the same rules in both the Settings tab and Serial Builder. To avoid a discrepancy between Serial Builder and the channel settings, DAS Studio automatically changes Packetizer Filter Mode for all messages in the same channel and in the Settings tab as shown in the following example.

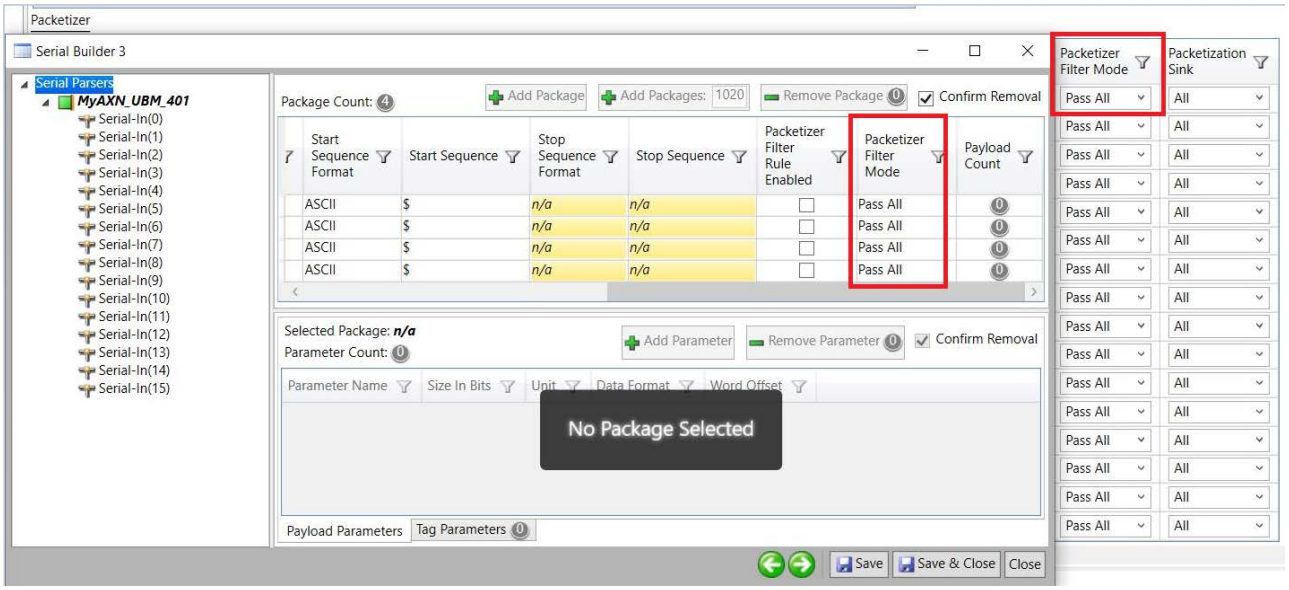
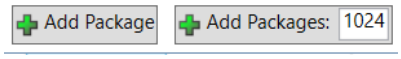


Figure 82-9: DAS Studio AXN/UBM/401 settings and Serial Builder showing the Packetizer Filter Mode settings propagation

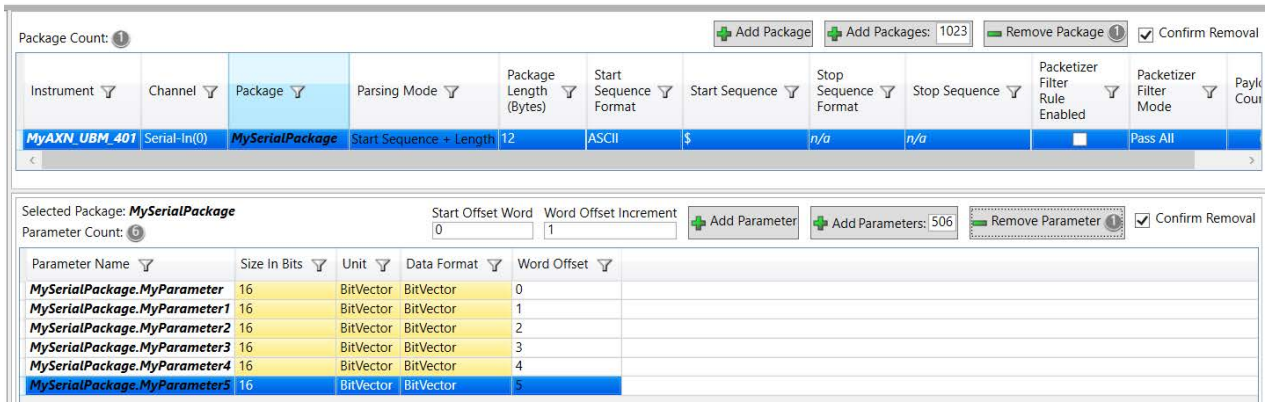
82.8.5 Adding parameters to the package

After you have defined rules to identify a message, refer to the following to select the number of bytes required to be parsed.

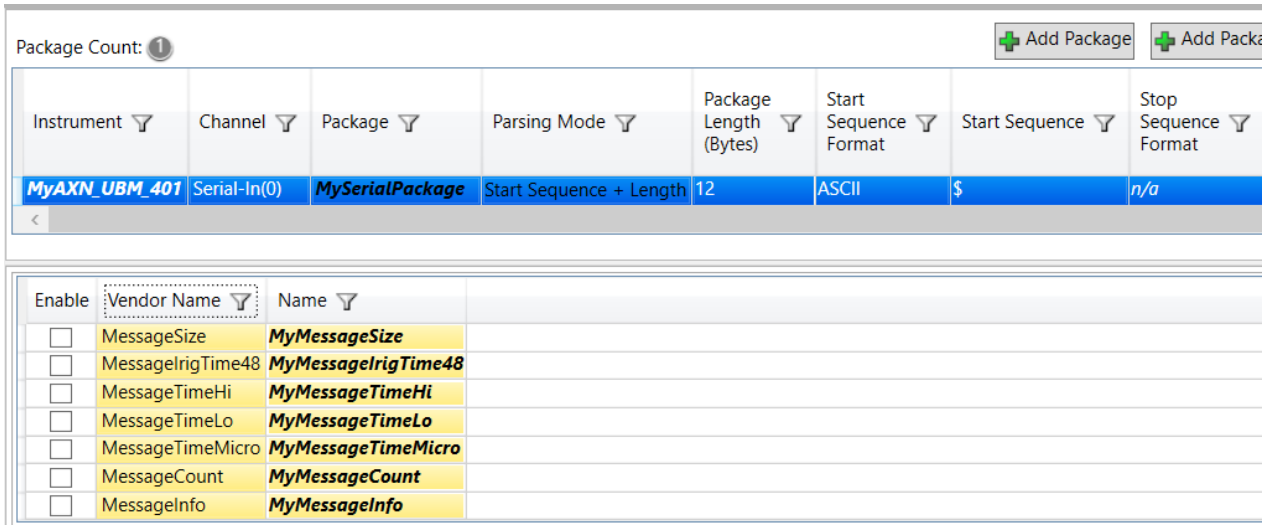
1. Click **Add Parameter** to add a single parameter. To add multiple parameters, click **Add Parameters** (typing the number of parameters in the field). Up to 1024, 16-bit parameters can be defined for every message. This corresponds to a maximum of 2048 characters per message.



In the following example, 6 parameters are added.
Serial message with 6 data words starting from offset 0 and increasing by 1 position.



2. To tag a message, select the message and then click the **Tag Parameters** tab.



The tags associated with the message are described in the following table.

Setting	Description
MessageIrigTime48	MessageIrigTime48 is a 48-bit register consisting of three 16-bits time registers: TimeHi, TimeLo, and TimeMicro. Represents the time stamp of a valid parsed message.
TimeHi, TimeLo and TimeMicro	Same information as MessageIrigTime48 but split in three 16-bit registers. Note: These registers are implemented to provide compatibility with legacy systems.
MessageSize	Number of received bytes including start bytes.
MessageCount	Counter of received messages, however this counter might not increase incrementally. As explained in section “82.7.3 Global parameters settings” on page 5, ModuleMessageCount is the only counter in the module that counts messages received on all channels of the module. ModuleMessageCount stores the current value of that counter in the slot with each message, therefore ModuleMessageCount increases incrementally. MessageCount indicates where in the arrival order, the particular message was received. For this reason consecutive messages on a single bus may not have consecutive values as messages on other channels may have been received, however these messages do increase incrementally. Individual MessageCount from different channels can be correlated with ModuleMessageCount.
MessageInfo	Stale/skipped indication for each parsed message. These flags indicate whether messages are repeated or lost, that is, oversampling or undersampling situations respectively.

Refer to the AXN/UBM/401 data sheet for further information regarding these additional tags.

3. To save your changes and close **Serial Builder**, click **Save & Close**.

When package building in Serial Builder is complete, these parameters become available to be placed in a PCM stream or placed packet.

82.9 Packetizer operation

Independently of the parser, when packetizer is enabled, an iNET-X packet stream is generated for each channel. All received bytes are encapsulated in an iNET-X parser-aligned payload structure. In the context of packetizer, the programmable message gap allows the module to split the incoming bytes into shorter timestamped sequences. A block header attached to each sequence stores the channel index, length, and the time each message is received. These parser-aligned packets may be transmitted aperiodically to optimize network bandwidth utilization and memory usage when recording serial traffic.

There are many settings available to configure or tune packetizer behavior.

While the structure of the packetizer packets are what we refer to as parser-aligned packets, the AXN/UBM/401 relies on gaps between the data to identify the start/end of the parser-aligned blocks. The size of the gap between messages can be set on a channel-by-channel basis. The Gap Between Messages setting (see the following figure) is expressed in characters, and ranges from 0 to 10,000 characters.

NOTE: To view the screen shots shown in this section in DAS Studio 3, ensure the AXN/UBM/401 module is in context and the Settings tab is selected.

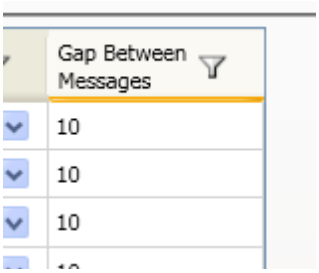


Figure 82-10: Gap Between Messages setting from Settings tab

Setting 0 characters results in all gaps being ignored and the packetizer packets being filled with bytes of data until the packet is either full or times out; thereafter the packet is transmitted.

Conversely, if a single message continues past the end of one packet with no gap, the succeeding bytes are packetized in a new parser block in the next packet. A bit in the header of the first parser block in the following packet, is set to indicate that this block is a continuation of the message in the final block of the previous packet. The location of this continuation bit is marked Cn (Continuation Indicator) in the example parser blocks shown in the following figure.

Parser blocks example

UART message parser block (10 characters of data)

MSB **LSB**

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Er	Error Code							Quad Bytes=5								Message Count								TBD				Bus ID			
Elapsed Time																															
TBD		Cn	P=0					TBD								Data #1								Data #2							
Data #3								Data #4								Data #5								Data #6							
Data #7								Data #8								Data #9								Data #10							

UART message parser block (5 characters of data with 1 byte padding)

MSB **LSB**

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Er	Error Code							Quad Bytes=4								Message Count								TBD				Bus ID			
Elapsed Time																															
TBD		Cn	P=1					TBD								Data #1								Data #2							
Data #3								Data #4								Data #5								Padding							

UART = Universal Asynchronous Receiver Transmitter)

UART Message Parser Block (8 characters of data with 2 bytes padding)

MSB **LSB**

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Er	Error Code							Quad Bytes=5								Message Count								TBD				Bus ID			
Elapsed Time																															
TBD		Cn	P=2					TBD								Data #1								Data #2							
Data #3								Data #4								Data #5								Data #6							
Data #7								Data #8								Padding								Padding							

UART Message Parser Block (13 characters of data (7 bits per character), 1 byte padding)

MSB **LSB**

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Er	Error Code							Quad Bytes=6								Message Count								TBD				Bus ID			
Elapsed Time																															
TBD		Cn	P=1					TBD								Data #1								Data #2							
	Data #3								Data #4								Data #5								Data #6							
	Data #7								Data #8								Data #9								Data #10							
	Data #11								Data #12								Data #13								Padding							

P = Number of padding bytes added to complete final quadbyte to 32 bits
Cn = 0 0 Complete message
1 0 First fragment of a message that also continues in next packet
0 1 Last fragment of the message continued from last packet and is now complete
1 1 Middle fragment, message continues from last packet and is not finished more in the next packet

Figure 82-11: Parser block formats used in packetizer

The other available packetizer settings are shown in the following figure and described in the table that follows.

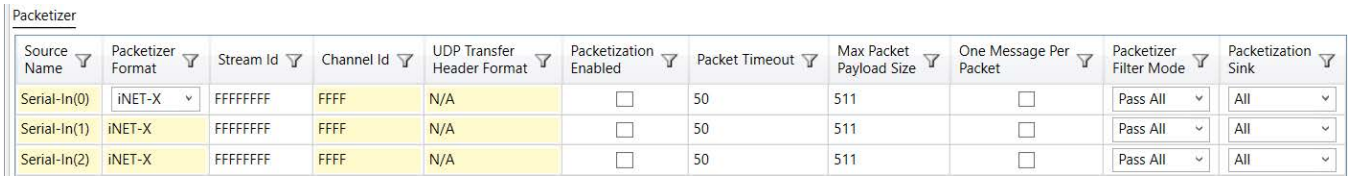


Figure 82-12: Packetizer settings

Setting	Description
Source Name	Channel to packetize.
Packetizer Format	iNET-X or Chapter 10. This setting defines the packetizer format for all channels.
Stream ID	iNET-X stream identifier for selected channel if a packet is generated via the assertion of Packetization Enabled. This is a conditional setting and is only active when the Packetizer Format is set to iNET-X.
Channel ID	Chapter 10 channel ID for selected channel if a packet is generated via the assertion of Packetization Enabled. This is a conditional setting and is only active when the Packetizer Format is set to Chapter 10.
Packetization Enabled	Enables the generation of a stream of packets containing messages from this channel. DAS Studio 3 automatically creates a packetizer packet after verification/programming.
Packet Timeout	The timeout in milliseconds before a packet is generated if insufficient messages have been received to reach the Packet Size. Packets generated due to Packet Timeout are tagged in the iNET-X header. The Packet Timeout ranges from 10 ms to 999 ms (default value is 50 ms). Reducing this value results in more frequent and generally smaller packets. Increasing the value results in less frequent, but generally bigger packets.
Max Packet Payload Size	The number of words in the packet buffer, ranges from 200 words to 511 words. The default value is 511 words; reducing this value results in smaller and therefore generally more frequent packets.
One Message Per Packet	This setting determines whether multiple or a single message should be included in a packet. When enabled, a packet is generated after a valid message has been detected. Subsequent messages are placed in separate packets. This simplifies message extraction as there is only one message per packet.
Packetization Filter Mode	Specifies the filtering mode for the channel. **Block By Rule** means that messages that match defined filter rules are blocked and all other messages are passed through. **Pass By Rule** means that messages that match defined filter rules are passed through and all other messages are blocked. **Pass All** means that all messages are passed through. This is only applicable when Packetization is enabled.
Packetization Sink	Selects which modules the packetizer package is sent to for transmission or storage. The choices are **Controller only**, **All slots**, or **Slot** in which a sink module that supports packetizer logging.

For further information regarding iNET-X Placed packets used by the packetizer refer to *TEC/NOT/067 - IENA and iNET-X packet payload formats*. Additionally, the AXN/UBM/401 data sheet provides several examples of packetizer parser blocks.

The AXN/UBM/401 data sheet explains the Chapter 10 packetizer format generated by the module.

NOTE: IADS does not currently support this Chapter 10 format.

82.10 Enabling packetizer

To turn on packetizer operation on any channel, define a unique stream ID for that channel and then select the Packetization Enabled check box for that channel as shown in the following figure. The packetizer is enabled the next time the module is programmed.

Packetizer

Source Name	Packetizer Format	Stream Id	Channel Id	UDP Transfer Header Format	Packetization Enabled
Serial-In(0)	iNET-X	1	FFFF	N/A	☑
Serial-In(1)	iNET-X	2	FFFF	N/A	☑
Serial-In(2)	iNET-X	3	FFFF	N/A	☑

Figure 82-13: Packetization Enabled setting

NOTE: DAS Studio 3 automatically creates packetizer packets on the aperiodic transmitter (such as the AXN/BCU/402). The packet rate is always 1 Hz but this value is only required for XidML. This value is not used by the Axon hardware.

Settings | Processes | Packages | Algorithms | Documentation

Channels

Instrument Name	Channel Name	Bit Rate	Connection Name	Connected Instrument	Connected Channel	Package Count
MyAXN_BCU_402	Ethernet(1)	n/a	MyAXN_CHS_06U_Packetizing			1
MyAXN_BCU_402	Ethernet(2)	n/a	MyAXN_CHS_06U_1_Packetizing			1

Package Properties

Name	Rate (Hz)	Type	Sub Type	Stream ID	Source IPA	Source UDP Port	Destination MAC	Destination IPA	Destination UDP Port	DataType
AXNUBM401_12	1	INet-X	Parser aligned	12	192.168.28.1	1023	01-00-5E-00-01-64	235.0.1.100	8010	Serial

Figure 82-14: AXN/BCU/402 packages showing a packetizer created by DAS Studio 3 after verification/programming

82.11 Appendix

82.11.1 Termination

For an RS422/485 bus, the transmission line should be terminated at the last transceiver on the line (bus). Transmission line termination schemes is a topic of discussion beyond the scope of this technical note. The section gives a basic overview of the termination schemes.

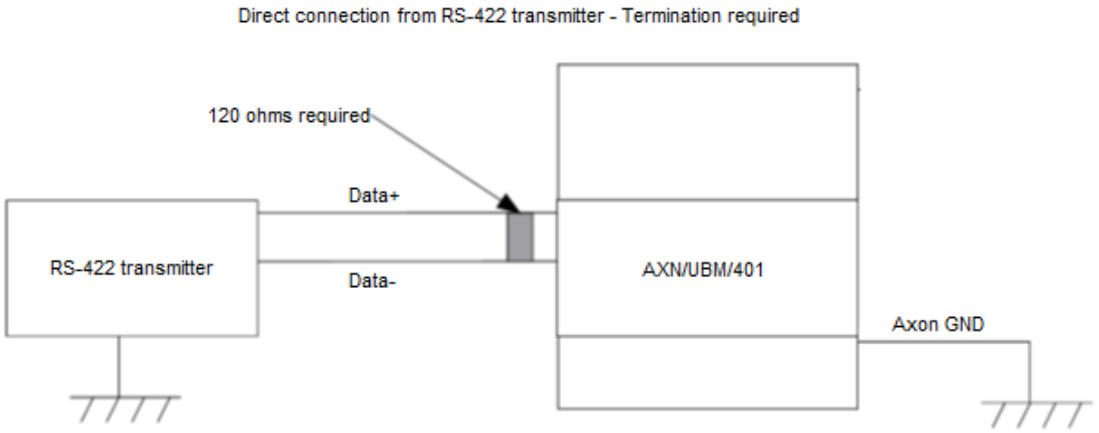


Figure 82-15: Example of termination required on the AXN/UBM/401

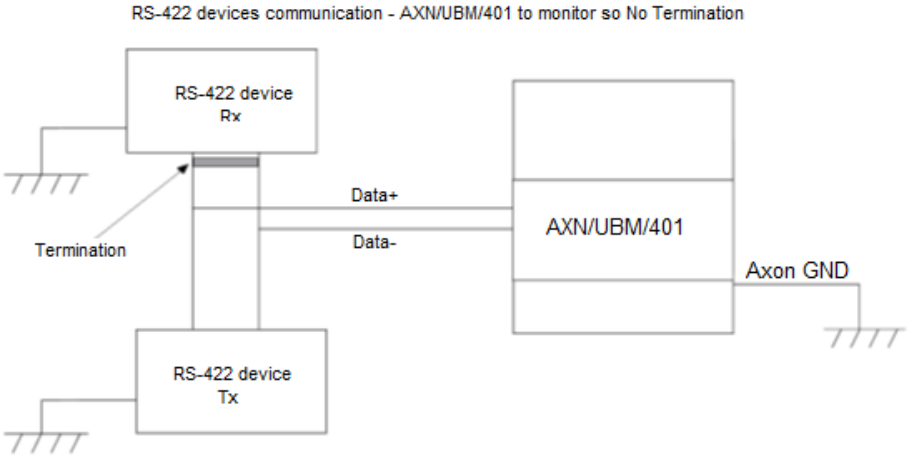


Figure 82-16: Example of no termination required on the AXN/UBM/401

The AXN/UBM/401 can be programmed to have termination. This programmable termination is not active when the power is off, that is, the bus is not terminated if Axon is powered off. If permanent termination is required, then external termination should be used. Refer to the “Getting the most from” section of the *AXN/UBM/401* data sheet.

82.11.2 Grounding

Grounding schemes is a topic of discussion beyond the scope of this technical note. Unlike RS-232, RS-422, and RS-485 transmission is differential. The previous two figures show a system configuration presented without a separate ground wire. The basic rules for electronic circuits still require a clean ground connection to ensure error-free communication between drivers (Tx) and receivers (Rx). For further information refer to *TEC/NOT/063 — Grounding and shielding of the Axon and Acra KAM-500*.

82.11.3 Message gap

The gap between messages can also be used for parsing. This gap needs to be defined if the start sequence appears inside the message or there is no unique start sequence.

The example below shows a chain of characters. The AXN/UBM/401 is programmed with AB as a Start Sequence and with a length of 10 bytes for the Stop Sequence, which means DW0 to DW4 are required. The Parser Data Endianness is set to First byte at low end of word.

If you set the gap at 0—because only the start sequence is used as a parser slot—the module may start at power up parsing when it encounters the start sequence AB inside the message.

So the AXN/UBM/401 could output DW0 = AB, DW1 = CD, DW2 = EF then all the other DW will be random data or data previously stored in RAM, the next instance of the data word, will be DW0 = AB, DW1 = G\n then DW2 will be from the next instance, that is, AB, DW3= DD…

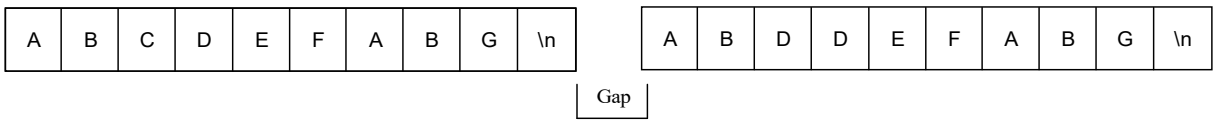


Figure 82-17: Example of gap between messages

To parse the whole message correctly, the AXN/UBM/401 gap between messages needs to be programmed with at least 1 byte. The module then outputs DW0 = AB, DW1 = CD, DW2 = EF, DW3= AB, DW4= G\n.

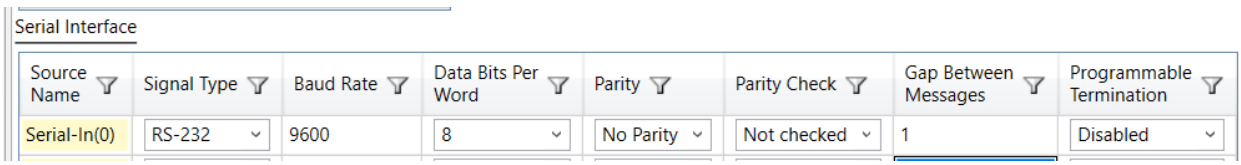


Figure 82-18: Example of gap between messages being set to 1 character in DAS Studio 3

As explained in “82.8.2 Parsing Mode” on page 7, parsing modes Length Only and Stop Sequence Only can use Message gap when the incoming message has no unique start sequence or the start sequence is present within the message. It is recommended to use these advanced parsing options with prior knowledge of the incoming message. Care must be taken in order to ensure the programmed message gap is not greater than the time of the next incoming message. Also power up conditions should be considered as incorrect parsing could occur when the module is powered up in the middle of the reception of a message.

82.11.4 Wildcard in Start Sequence

Serial Builder supports wildcard symbols '*' in the Start Sequence. When using Hex format for Start Sequence, each wildcard represents 4 bits (nibble), therefore two wild cards symbols “**” are required to represent one character boundary (1 byte). When using Binary format for Start Sequence, each bit can be wildcarded on a bit-to-bit basis.

Instrument	Channel	Package	Parsing Mode	Package Length (Bytes)	Start Sequence Format	Start Sequence	Stop Sequence Format	Stop Sequence	Payload Count
MyKAD_UBM_103	Serial-In(0)	***MySerialPackage***	Start Sequence + Length	12	Hex	11**22	*n/a*	*n/a*	0

Figure 82-19: Serial Builder - Example of wildcard used on a 3-character start sequence using Hex format

82.11.5 Bad Stop Bit

The module detects a Bad Stop Bit if it’s not as expected.

For example:

• No parity is configured and a message is received that was transmitted with a Parity bit. In this case the parity bit is decoded as a bad stop bit.

• Parity is configured and a message is received that was transmitted with no Parity bit. In this case since no Parity bit was sent, the module decodes the received Stop bit as an invalid Parity bit and then expects a Stop bit but never receives it and thus reports it as a Bad Stop Bit.

82.11.6 Packetizer packet format (parser aligned)

For more information regarding the structure of parser-aligned iNET-X packets and various packetizer scenarios, refer to the “iNET-X Packetizer format” section of the *AXN/UBM/401*data sheet.

82.11.7 Recommended reading

To better understand this paper, read the following documents.

Table 82-1: Data sheets

Document	Description
AXN/UBM/401	RS-232, RS-422 or RS-485 serial bus parser/packetizer - 16ch

Table 82-2: Technical notes

Document	Description
TEC/NOT/063	Grounding and shielding of the Axon and Acra KAM-500
TEC/NOT/067	IENA and iNET-X packet payload formats

Table 82-3: User manual

Document	Description
DOC/MAN/030	DAS Studio 3 User Manual

This page is intentionally blank