afero

afPro UART Protocol

This section describes using afPro (Afero Serial Protocol) over a UART interface at 9600 baud 8-N-1 with no flow control. The examples used assume a standalone MCU as the master, and ASR as the slave. Note that the MCU must always be the master and ASR must always be the slave.

We use timing diagrams from a logic analyzer to show the traffic.

The Approach

To minimize differences between afPro used with SPI vs. UART, all transactions are driven by the master, the MCU. To use afPro in a master/slave scheme, two capabilities must be provided:

1. Non-hardware flow control to allow the slave to prepare for transactions as well as handling collisions (i.e., the master and the slave both have something to send).
2. Ability for the slave to request the master to read from it.

To accommodate 1, we use a sync message to communicate the number of bytes to be sent from the master or the slave. This message can be sent at any time to make sure the master stays in sync with slave. It is also used to recognize collisions; that is, when both master and slave want to send at the same time.

To accommodate 2, we define a single-byte message to indicate the slave is ready to receive data. The slave can send this byte when it wants the master to communicate with it. The master can then send a sync message to begin the transaction.

Resetting ASR When the MCU Reboots

Whenever the MCU reboots, it must force ASR to reboot as well, thus canceling any unresolved transactions between master and slave. The MCU does this by sending a Reset signal to ASR. But first the MCU must be ready to receive a request from ASR:

1. MCU prepares the UART interface for communication with ASR.
2. MCU asserts the Reset signal for a minimum of 250 ms.

This sequence forces ASR to start from a known (initial) state. ASR will initiate the Zero Sync right after it reboots, as described below in Example 1.

Attribute Value Change Rules

afLib for Arduino API

MCU Coding Tips

Hardware Reference >

Technical Notes

Training Labs >

Release Notes >

Sync Request/Response and Acknowledge Messages

A sync message is the first thing sent for every afPro transaction. There are two types of sync messages: Sync Request/Response and Sync Acknowledge.

UINT8_T	UINT16_T	UINT16_T	UINT8_T
Message Type	Master Tx byte count	Slave Tx byte count	Checksum

Both messages have the same fixed format. The Message Type is 0x30 for a Sync Request/Response, and 0x31 for a Sync Acknowledge.

The Message Type byte is followed by two bytes that indicate how many bytes the MCU (master) wants to send, then two bytes indicating how many bytes ASR (slave) wants to send. When the MCU makes the initial Sync Request, the ASR bytes will contain zeroes since the MCU is not expecting any data from ASR. On the other hand, the Sync Response from ASR will include the MCU byte values in addition to its own byte values, since ASR is acknowledging the number of bytes it knows it will receive from the MCU, per the Sync Request.

Finally, the checksum is simply the unsigned sum of all the other bytes in the message.

Note that:

- Sync Requests always come from the master.
- Sync Responses always come from the slave.
- Sync Acknowledges always come from the master.
- Ready bytes always come from the slave.

The simplest form of sync is as follows: the master sends a Sync Request with both byte counts set to zero. If the slave also has no bytes to send, it responds by sending a Sync Response with both byte counts set to zero. The master then sends a Sync Acknowledge with both byte counts set to zero. The transaction is complete.

For all the timing diagrams in the examples below, the byte counts are little-endian and the signal order is:

- Bytes from MCU (Tx) to Afero (Rx)
- Bytes from Afero (Tx) to MCU (Rx)

Example Transactions

In these examples, note that:

- It is *always* ASR that sends the Ready byte (0x32); the MCU controls the data flow by sending the Sync Request message (0x30).

- ASR responds to an MCU Sync Request message (0x30) by sending a Sync Response message (also 0x30).
- The ASR Ready byte is used to drive the MCU state machine to move between message types.
- ASR sends the Ready byte between all message types *except* between Sync Requests from the MCU and Sync Responses from ASR.

We'll look at these example transactions:

- Example 1: Zero Sync
- Example 2: Set Attribute
- Example 3: Update Attribute
- Example 4: Set Attribute with Collision

Example 1: Zero Sync

Zero sync is required for the first transaction after a reset, but is not required after that. However, zero sync is allowed any time after reset to establish synchronization between master and slave. Generally, after the first sync, the MCU and ASR will fill in their respective bytes to send.

The zero sync transaction starts with ASR sending the Ready byte (0x32). This lets the MCU know it can now send data. The full sequence is:

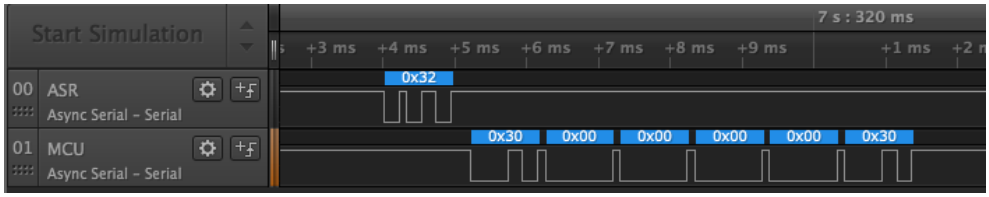
The events occur in this order:

1. ASR sends Ready byte.
2. MCU sends Sync Request.
3. ASR sends Sync Response.
4. ASR sends Ready byte.
5. MCU sends Sync Acknowledge.
6. ASR sends Ready byte.

Next, we'll look at the individual events.

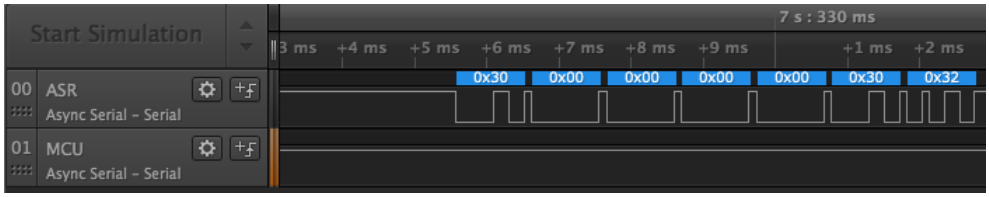
Sync Request (Events 1-2)

After the MCU receives a Ready byte from ASR, the MCU sends a Sync Request message:



Sync Response (Events 3-4)

ASR responds with a Sync Response message indicating it has no bytes to send to the MCU, then follows with a Ready byte:



Sync Acknowledge (Events 5-6)

Since neither the MCU nor ASR has any bytes to send and the MCU has received the Ready byte from ASR, the MCU can respond with a Sync Acknowledge. The Sync Acknowledge looks just like the Sync Request, except the message type is 0x31. Finally, the MCU waits for a final Ready byte from ASR to know the transaction is complete:

At this point the master and slave are in sync. Neither one has anything to send.

Example 2: Set Attribute

In this example, the Afero Cloud requests the MCU change the value of one of its attributes using a Set Attribute transaction. The MCU makes the change, resulting in an Update Attribute transaction (detailed in Example 3: Update Attribute).

The transaction starts with ASR sending the Ready byte to let the MCU know it has something to send. The transaction ends with ASR sending the Ready byte to signal the operation has completed:

There are eight distinct events:

1. ASR sends Ready byte.
2. MCU sends Sync Request.
3. ASR sends Sync Response.
4. ASR sends Ready byte.

5 MCU sends Sync Acknowledge.

6 ASR sends Ready byte.

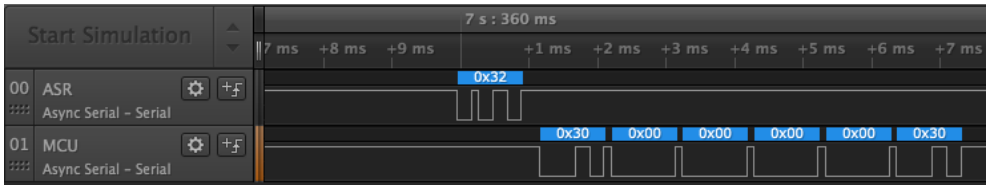
7 ASR sends data and MCU reads data.

8 ASR sends Ready byte.

Let's look at each event in more detail.

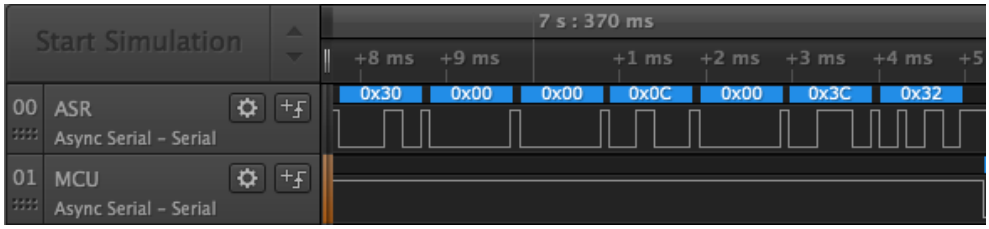
Sync Request (Events 1-2)

The transaction begins with ASR sending the Ready byte to signal the MCU it has something to send. The MCU responds by sending a Sync Request message with MCU bytes to send set to zero since it has nothing to send, and ASR bytes to send set to zero since it does not yet know how many bytes ASR wants to send:



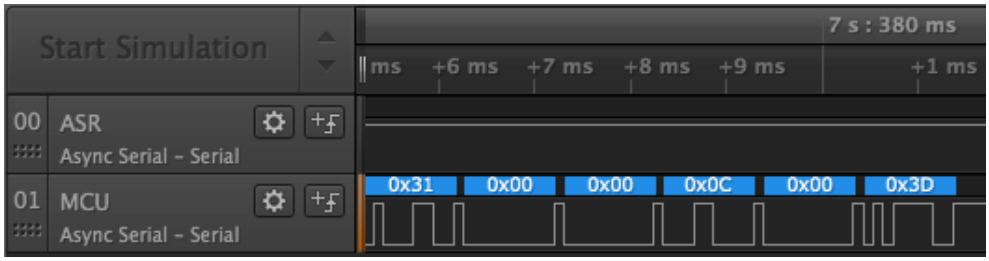
Sync Response (Events 3-4)

The ASR Sync Response message is next, which says it has 12 bytes to send, followed by a Ready byte:



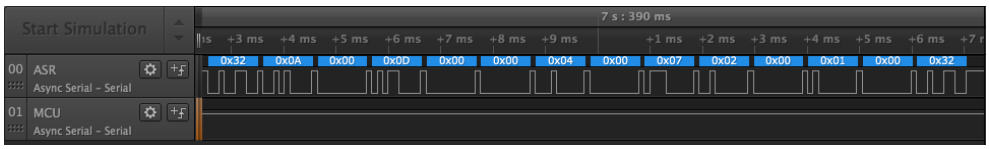
Sync Acknowledge (Event 5)

After receiving the Ready byte, the MCU sends a Sync Acknowledge message that includes the byte count from the ASR Sync Response. The MCU is telling ASR it is ready to receive 12 bytes:



Receive Data (Events 6-8)

Next, ASR sends a Ready byte followed by the 12 bytes of data that make up the actual Set Attribute command. The MCU reads the data:



After sending the data, ASR completes the transaction by sending a Ready byte.

Example 3: Update Attribute

In this example, the MCU changes the attribute value locally and then instigates an Update Attribute transaction:

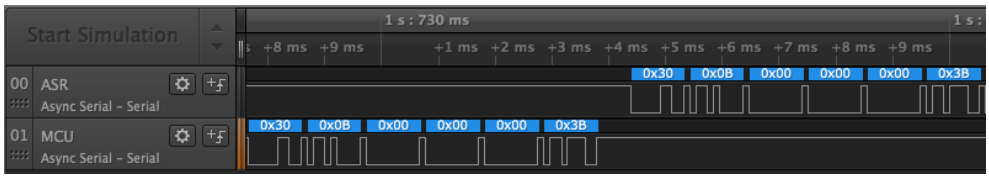
The entire transaction consists of the following events:

1. MCU sends Sync Request.
2. ASR sends Sync Response.
3. ASR sends Ready byte.
4. MCU sends Sync Acknowledge.
5. ASR sends Ready byte.
6. MCU sends data and ASR reads data.
7. ASR sends Ready byte.

Now we'll look at each event with the timing diagrams.

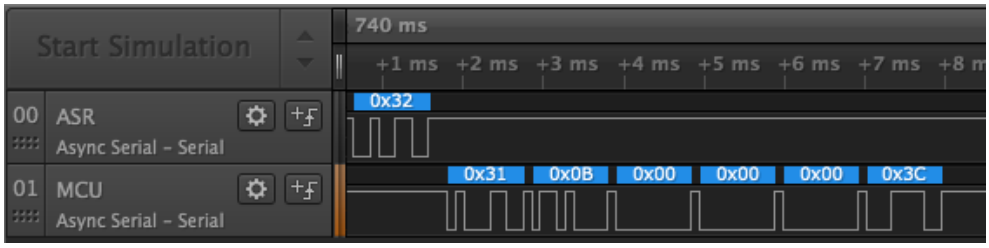
Sync Request/Response (Events 1-2)

The communication begins with the MCU sending a Sync Request message. The MCU wants to send Update Attribute data consisting of 11 bytes, so it sends 0x0B in the Sync Request message. ASR then sends a Sync Response:



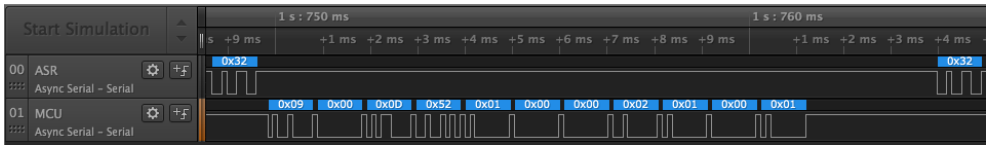
Sync Acknowledge (Events 3-4)

ASR sends a Ready byte, and the MCU sends the Sync Acknowledge, confirming it's sending 11 bytes:



Receive Data from MCU (Events 5-7)

Only after the Sync Acknowledge and the next ASR Ready byte from ASR does the MCU send the actual data:



ASR completes the transaction with a final Ready byte.

Example 4: Set Attribute with Collision

Lastly, there is the rare situation in which both the master and the slave have data to send at the same time. In this case, the Sync Request message from the master and the Sync Response message from the slave will both contain non-zero values. We call this case a "collision".

The protocol dictates that whenever there is a collision, the MCU wins. Both the MCU and ASR notice the collision. ASR queues its Sync Response and prepares to handle the MCU Sync Request. The MCU just waits for the next Ready byte from ASR and then resends the Sync Request. At this point the transaction looks just like Example 3: Update Attribute, above.

Next: *Device Attribute Message Protocol*

Last modified: December 5, 2017