Arduino Sensor Network Documentation – Overview

Brent Seidel
Phoenix, AZ

2018-09-09

1 Synopsis

This document will contain descriptions of the various components of the Arduino Sensor Network. First is a brief overview of the various components and then a discussion of the communication protocol used between them. An example network is shown in Figure 1.

2 Components

The network consists of two required components plus optional components. The required components are a single controller and one or more responders. Listeners may optionally be added as they simply monitor the bus traffic and do not transmit. A BeagleBone Black listener is used to present the data from the network on web pages. Note that the component boundaries are a little fuzzy in that some components can perform functions from multiple categories.

2.1 Controller

The controller is responsible for the operation of the bus. It sends a request command to each node on the bus and waits for a response. It may also send data messages. Typically requests are done in a round-robin fashion – request node 1, request node 2, etc. If a node does not respond within a timeout period, it is marked as missing and the next node is polled.

It would probably be possible to combine the controller functions and the web gateway listener function into a BeagleBone Black or other similar embedded computer. However, keeping them separate allows the controller to be simpler and keep the network functioning even if the web gateway has problems. So, if presenting data collected on a web page is the primary function of the network, go ahead and combine them. If, however the primary function of the network is monitoring and controlling something, it would be better to keep the web gateway separate.

2.2 Responder

One or more responders may be attached to the bus. Each responder can produce one or more addresses of data. The first address (address 0) is used for node identification. It contains the name and the number of addresses produced by the node. Additional addresses are produced depending on the sensors attached to the node and the node configuration.

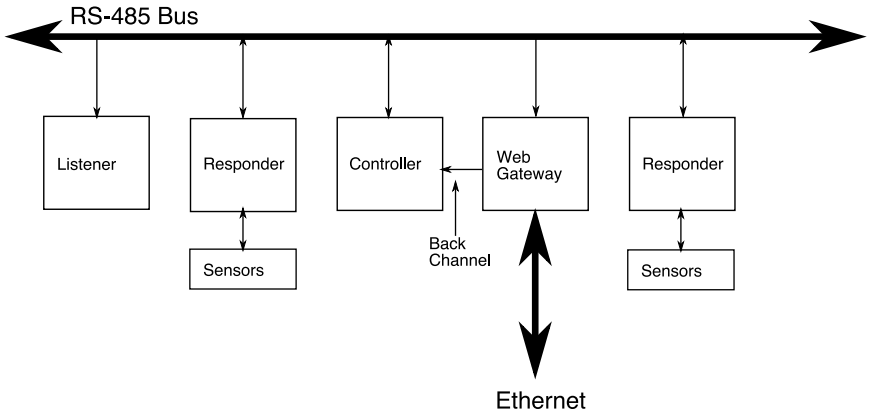

Figure 1: Example Sensor Network

2.3 Listener

Any number of listeners may be added to the network. These simply monitor the network traffic and usually present it in a different format. A listener can also be used to perform actions based on certain data on the network, though it would probably be better to use a responder that can report back what it is doing.

The BeagleBone Black is used as one example of a listener that presents the data on a web page. Another example listener is an Arduino that simply listens to the bus and copies the characters to the host computer for display.

3 Protocol

3.1 Physical Layer

The physical layer is RS-485 twisted pair. CAT-5 cable is used because I happen to have a bunch of CAT-5 available and it works. One pair is used leaving the other pairs available for other uses such as power distribution. The baud rate is set to 115.2kbps. This can be changed depending on the application. The only constraint is that all nodes agree on the baud rate.

3.2 Node Organization

Each node is given a unique node ID. The controller node is given ID 0. The message format allows 32 bits for the node ID though typically small numbers would be used. Each node can produce a number of addresses of data. Address 0 is used to provide node information. The information includes the number of addresses that the node can produce and the node's name.

3.3 Messages

Two types of messages are defined: Requests which only come from the controller and request a specific address of data from a specific node, and Responses which are the data produced by a

responder in response to a request.

To aid in debugging, the format of the messages is in ASCII text. The type of message is indicated by the first character, '' for requests and '#' for responses. The first two elements in both types of messages is the device ID and the address separated by a forward slash, '/'. The end of the message is a percent sign, '%' followed by a (currently unimplemented) checksum and a CR-LF.

All of the numbers are 32 bit unsigned integers expressed in hexadecimal, with leading zeros optional.

3.3.1 Requests

Requests should only be sent by the controller node. The first character in a request message is an ampersand, ''. This is followed by the device ID in hex, followed by a forward slash, '/'. Then the address in hex, followed by a percent sign, '%'. This is followed by a currently unused checksum in hex. The message is terminated by a CR-LF.

A typical request looks something like, '@00000001/00000000%FF'. This requests address 0 from device 1. Since leading zeros are optional, the same message can also be expressed as, '@1/0%FF'.

3.3.2 Responses

Responses are sent in response to a request from the controller node. The first character in a response message is an octathorpe, '#'. This is followed by the device ID in hex, the address in hex, and the message type in hex, separated by a forward slash, '/', and ended by an ampersand, '&'. This is followed by the data fields. These are up to 32 32-bit hex numbers separated by apmersands, '&' and terminated by a percent sign, '%'. This is followed by a currently unused checksum in hex. The message is terminated by a CR-LF.

The format of the responses flexible and the number of data fields sent depends on the type of message. It may even vary between messages of the same type. A typical response looks something like #2/1/4&0&0&1E690&175C0F1&C775%FF. This is a response from device ID 2, address 1 with message type 4 (MSG_TYPE_BME280)

If the controller has data to send, it can send a response message without sending a request, as the request can be considered to be implied.

3.3.3 Message Types

To make the response messages easier to decode, each message has a message type field. This indicates what to do with the rest of the data in the message. Not all messages are currently implemented and more will likely be added. The various message types are in Table 1. Some of the message types have a subtype. In particular, discretes have the defined types as shown in Table 2, with more likely to be added. When analogs get defined, there will be subtypes to indicate what the analog values represent.

Value	Symbol	Meaning
0	MSG_TYPE_UNKNOWN	Undefined or not present.
1	MSG_TYPE_EMPTY	Everything is OK, but no data to send
2	MSG_TYPE_NAK	Address not supported
3	MSG_TYPE_INFO	Address 0 information message
4	MSG_TYPE_BME280	BME280 sensor values
5	MSG_TYPE_DISCRETE	Discretes
6	MSG_TYPE_ANALOG	Analog values (not yet implemented)
7	MSG_TYPE_VERSION	Version/Software ID (not yet implemented)
8	MSG_TYPE_CCS811	CCS811 sensor values
9	MSG_TYPE_TSL2561	TSL2651 sensor values

Table 1: Defined Message Types

Value	Symbol	Meaning
0	DISCRETE_UNKNOWN	Unknown discretes.
1	DISCRETE_CMD	Command discretes

Table 2: Defined Discrete Types