Arduino Sensor Network Documentation – Overview

Brent Seidel
Phoenix, AZ

2018-11-20

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.

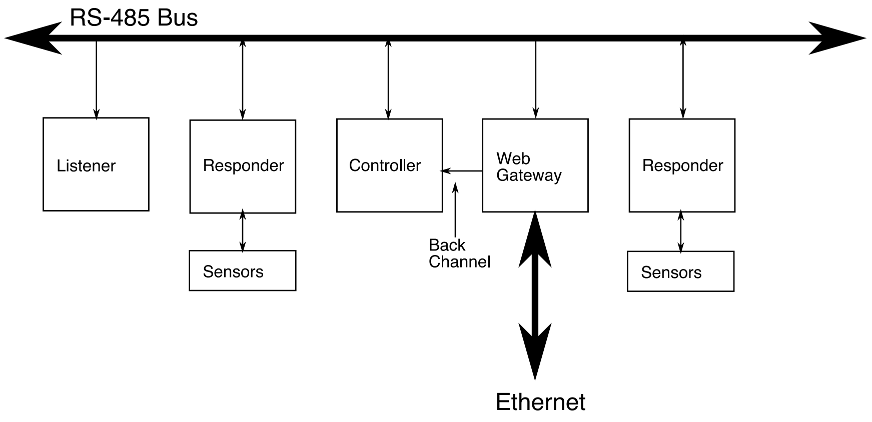

Figure 1: Example Sensor Network

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.

2.4 Web Gateway

The web gateway is a specialized listener. It collects data from the bus and presents it as web pages and XML formatted data. It is uses the `Ada-Web-Server` repository as the web server. The RS-485 listener interface is implemented as an Ada Task that receives and processes data from the bus. The back channel is another Ada Task that is used to transmit commands to the controller. The use of a task is intended to prevent multiple commands being transmitted simultaneously and getting garbled.

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.

Value	Symbol	Meaning
0	`CMD_READ`	Read data from the specified address.
1	`CMD_RESET`	Perform a software reset.
2	`CMD_WRITE`	Perform an application specific write.

Table 1: Defined Command Codes

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. The address is optionally followed by a command code and value, each preceded by an apmersand '`&`'. This is followed by a percent sign, '`%`' and a currently unused checksum in hex. The message is terminated by a CR-LF.

If a command code is not specified, the default command code of 0 for `CMD_READ` is used. The currently defined command codes are in Table 1.

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.

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 2: Defined Message Types

Value	Symbol	Meaning
0	`DISCRETE_UNKNOWN`	Unknown discretes.
1	`DISCRETE_CMD`	Command discretes.
2	`DISCRETE_MIXED`	Mixed discrete types.
3	`DISCRETE_SWITCH`	Discretes from switches.

Table 3: Defined Discrete Types

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 2. Some of the message types have a subtype. In particular, discretes have the defined types as shown in Table 3, with more likely to be added. The analog values have the defined types as shown in Table 4, with more likely to be added.

Value	Symbol	Meaning
0	`ANALOG_UNKNOWN`	Unknown analog types.
32	`ANALOG_MIXED`	Mixed analog types.
64	`ANALOG_POT`	Analog values from potentiometers.

Table 4: Defined Analog Types

3.4 Back-Channel Commands

While the commands are human usable, they are designed for ease of parsing, not ease of use. It is expected that higher level software will provide a wrapper for them. At this point, they should be considered to be experimental and subject to change. There are two groups of commands:

- General commands tell the controller to do something that applies to all nodes on the network. The specific general commands currently defined are:
  - `!A` Tell all nodes to turn their indicator LED off.
  - `!B` Tell all nodes to turn their indicator LED on.
  - `!C` Tell all nodes to toggle their indicator LED.
  - `!D` Tell all nodes to identify themselves.
- Directed commands are sent to a specific node telling it to do something. The currently defined directed commands are:
  - `!R<node>,` Request node it node id of `<node>` to perform a software reset.
  - `!S<cmd>,<node>,<arg>,` Send an arbitrary command, `<cmd>`, to node id `<node>`, with command arguments, `<arg>`.

  Where all numbers are in hexidecimal. Note that the trailing comma is required.

4 Logging

Data can be logged to files for later analysis. Provisions exist for logging to be started and stopped and started on a global basis or by record type. Starting global logging creates the files for the data and stopping global logging closes the files. To log data, logging has to be started globally and logging of the desired data has to be enabled. The data records generally supported for logging are:

- Node information records
- BME280 Temperature, pressure, and humidity sensor
- CCS811 eCO_2 and TVOC (total volatile organic compounds) air quality sensor
- TLS2561 light sensor

The discretes and analogs are customized for the application and thus any logging of them would also have to be customized. The existing logging code can be used as an example. Note that logging can generate an enormous amount of data, so be prepared.

5 API

A number of the defined URLs return XML information. This can be used to create an API that can be used by programs beyond just a web browser.

5.1 Graphics

Two of the URLs return an image in SVG format. Typically, it is used for display on a web page, but could be used for other purposes.

5.1.1 Dial

The URL is `/Dial`. The parameters are:

- `min` - Specify the minimum value shown on the dial.
- `max` - Specify the maximum value shown on the dial.
- `value` - Specify the value indicated by the dial's pointer.

5.1.2 Thermometer

The URL is `/Thermometer`. The parameters are:

- `min` - Specify the minimum value shown on the thermometer.
- `max` - Specify the maximum value shown on the thermometer.
- `value` - Specify the value indicated by the thermometer.

5.2 Data and Control

The rest of the URLs either return data or are used to control the system.

5.2.1 Send a Command to a Node

This function is used to send a command to the controller via the back channel. The URL is `/xml/Command`. There is one parameter:

- `command` - The command to send as text.

The returned data is returned in a `<xml>` block with the following contents:

- `command` - Echoes the sent command, if the `command` parameter was present, or
- `error` - Provides an error message if the `command|` parameter was not present.

5.2.2 Get System Counters

The URL is `/xml/Counter`. There are no parameters. The returned data is in a `<xml>` block with the following contents:

- `<counter>` - XML container for number of web requests handled counter.
- `<tasks>` - XML contained for number of currently active web handler tasks.
- `<rs485>` - XML container for RS-485 state machine activity counter.

5.2.3 Get Maximum Device Number on the Network

The URL is `/xml/Devices`. There are no parameters. The returned data is in a `<xml>` block with the following contents:

- `<length>` - XML container for the maximum device number found on the network. Used to set the length of data structures.

5.2.4 Get Data from a Device on the Network

While this call is simple, this returns the most complex set of data. The URL is `/xml/DevData`. There is one parameter:

- `device` - Contains the node ID of the device to get data from.

The returned data is in an `<xml>` block. The following components may, or may not be present depending on the configuration of the node:

- `<info>` - This should always be present.
  - `<validity>` - Validity status for info data.
  - `<aging>` - Number of seconds since info message received.
  - `<addresses>` - Number of addresses supported by device.
  - `<name>` - Name of the node.
- `<bme280>` - Present if node is configured with a BME280 sensor.
  - `<validity>` - Validity status for BME280 data.
  - `<aging>` - Number of seconds since BME280 message received.
  - `<bme280_status>` - Status of the BME280 sensor reported by node.
  - `<bme280_age>` - Number of frames since BME280 sensor data read.
  - `<bme280_temp_c>` - Temperature in degrees Celcius.
  - `<bme280_pressure_pa>` - Atmospheric pressure in Pascals.
  - `<bme280_humidity>` - Relative humidity in percent.
- `<discretes>` - Present if node is configured to transmit discretes.
  - `<validity>` - Validity status for discretes data.
  - `<aging>` - Number of seconds since discretes message received.
  - `<disc_type>` - Code indicating the type of these discretes.
  - `<disc_value>` - Value of the discretes as a 32 bit unsigned integer. One bit for each discrete.
- `<analogs>` - Present if node is configured to transmit analog values.
  - `<validity>` - Validity status for analog data.
  - `<aging>` - Number of seconds since analog message received.

  - `<analog_type>` - Code indicating the type of these analog values.
  - `<analog_count>` - The number of analog value provided in this message.
  - `<value>` - Value of analog reading. This entry is repeated `analog_count` times. Once for each analog value.

- `<ccs811>` - Present if node is configured with a CCS811 sensor.
  - `<validity>` - Validity status for CCS811 data.
  - `<aging>` - Number of seconds since CCS811 message received.
  - `<ccs811_status>` - Status of the CCS811 sensor reported by node.
  - `<ccs811_age>` - Number of frames since CCS811 sensor data read.
  - `<ccs811_eco2>` - Measurement of CO_2 concentration.
  - `<ccs811_tvoc>` - Measurement of total volatile organic compounds (TVOC) concentration.
- `<tsl2561>` - Present if node is configured with a TSL2561 sensor.
  - `<validity>` - Validity status for TSL2561 data.
  - `<aging>` - Number of seconds since TSL2561 message received.
  - `<tsl2561_status>` - Status of the TSL2561 sensor reported by node.
  - `<tsl2561_age>` - Number of frames since TSL2561 sensor data read.
  - `<tsl2561_data0>` - Measurement of one band of light.
  - `<tsl2561_data1>` - Measurement of one band of light.
  - `<tsl2561_lux>` - Calculated illumination in Lux.
- `<pca9685>` - Present if node is configured with a PCA9685 PWM controller.
  - `<validity>` - Validity status for PCA9685 data.
  - `<aging>` - Number of seconds since PCA9685 message received.
  - `<channel>` - One of these entries for each of the 16 PWM channels. Each entry contains three comma separated values. The values are:
    - A boolean indicating if this channel has been set.
    - An unsigned integer indicating the PWM “On” count.
    - An unsigned integer indicating the PWM “Off” count.
- `<error>` - Present if an error has occurred.

5.2.5 Control Debugging Messages

Various debugging messages can be displayed on the console. This entry can be used to turn messages on or off and to report the status of debugging messages. The URL is `/xml/Debug`. There are several parameters:

- `rs485.char` - Set to ‘T’ to print characters that are read from the RS-485 bus.

- `rs485.msg` - Set to 'T' to print the type of message that is read from the RS-485 bus.
- `http.head` - Set to 'T' to print the headers from each HTTP request.
- `http.msg` - Set to 'T' to print the HTTP requests.
- `web.dbg` - Set to 'T' to print debugging messages from the web server.

The returned data is in a `<xml>` block with the following contents:

- `<rs485.char>` - `TRUE` if RS-485 character debugging is enabled.
- `<rs485.msg>` - `TRUE` if RS-485 message debugging is enabled.
- `<http.head>` - `TRUE` if HTTP header debugging is enabled.
- `<http.msg>` - `TRUE` if HTTP request debugging is enabled.
- `<web.dbg>` - `TRUE` if web debugging is enabled.

5.2.6 Control Data Logging

Starting logging is a two step process. First the specific messages to log need to be enabled. Then logging needs to be turned on. The URL is `/xml/Log`. There are several parameters:

- `log.info` - Set to 'T' to enable logging of info messages.
- `log.BME280` - Set to 'T' to enable logging of BME280 messages.
- `log.CCS811` - Set to 'T' to enable logging of CCS811 messages.
- `log.TSL2561` - Set to 'T' to enable logging of TSL2561 messages.
- `logging` - Set to 'T' to turn logging on.

The returned data is in a `<xml>` block with the following contents:

- `<log.info>` - `TRUE` if info message logging is enabled.
- `<log.BME280>` - `TRUE` if BME280 message logging is enabled.
- `<log.CCS811>` - `TRUE` if CCS811 message logging is enabled.
- `<log.TSL2561>` - `TRUE` if TSL2561 message logging is enabled.

Note that there is currently no item that gives the overall state of logging. This will need to be added at some point.

5.2.7 Get Device Name

The URL is `/xml/Name` name internal. There is one parameter:

- `device` - Specify the device number to get information from.

The returned data is in a `<xml>` block with the following contents:

- `<info>` - Block containing device information, if present. The information consists of the following:
  - `<validity>` - Validity code for the data.
  - `<aging>` - The time in seconds since the info record was received from the node.
  - `<presence>` - The time in seconds since any records were received from the node.
  - `<addresses>` - The number of addresses supported by the node.
  - `<name>` - The node's name as a string.
- `<error>` - Block containing an error message if device information cannot be provided.