Keywords: USB on-the-go (OTG), Android, 1-Wire

APPLICATION NOTE 5705

INTERFACE 1-WIRE USB ADAPTER WITH ANDROID

By: Wilson Tang, Member of Technical Staff

Abstract: This application note details how to interface the DS9490R, temperature logger iButton devices, and Android over USB On-the-Go (OTG). The communication protocol between each section is reviewed. We review the data sheet information and how to apply it in the end system. A sample Android application illustrating the example code is also available for download.

A similar version of this article appeared December 1, 2013 on *Electronic Products*.

Associated software is available for download.

Introduction

Look around. Smartphones are ubiquitous. Far more than a mere telephone, a smartphone is a powerful computer that fits in your pocket. Since the computing power and sensors available in a smartphone continue to increase, it is now practical to use that phone as the interface to other devices. This article shows how to use the USB port on an Android® smartphone to communicate with slave devices without a computer system. This example here uses the 1-Wire® bus for communication by USB from the Android to a Thermochron® iButton® temperature logger.

System Layout

Crucial to this application is the USB port on the smartphone. When the smartphone uses a USB on-the-go (OTG) transceiver, then USB host functionality is possible with other slave devices, such as mice, flash drives, keyboards, or the Thermochron here. Recent versions of the Android application programming interface (API) support USB host mode at the application level. This functionality lets end users install applications that "talk" with USB peripherals without rooting or installing special drivers on the user's smartphone.

The system block diagram is in **Figure 1**. The Android smartphone must use a USB OTG transceiver. Normally, the phone is connected to a computer and acts as a USB slave, but the USB OTG transceiver allows the phone to be transformed into a USB master. This role reversal requires a special OTG cable to provide the USB type A port and to signal that a USB slave is connected to the smartphone.

This application is a master–slave system with the Android smartphone as the master and the Thermochron data logger as the slave. The system uses a USB to 1-Wire/iButton (DS9490R) adapter to bridge the smartphone with the data logger. A network cable/socket, the DS1402D-DR8, is used as the 1-Wire bus, which only requires one data line. The data logger is the iButton Thermochron (DS1921G).

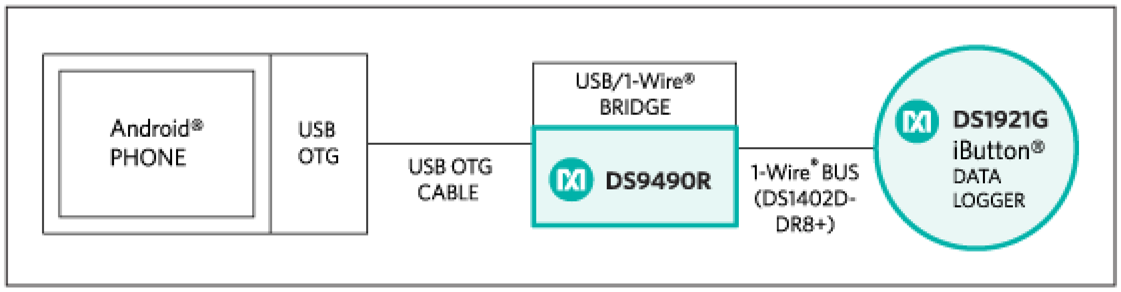


Figure 1. System block diagram uses the DS9490R 1-Wire adapter as the interface between the smartphone and the application device; the DS1402D-DR8 bus cable connects to the application device, here the DS1921G Thermochron iButton.

Important Role of the 1-Wire Bus

The 1-Wire bus is the interface between a single master and multiple slave system. The 1-Wire outputs are open drain, operating with a pullup resistor similar to I^2C. Some 1-Wire slaves can be parasitically powered from the 1-Wire bus, which charges an internal capacitor in the slave device when communications are not occurring. Each 1-Wire slave also has a factory-lasered, unique 64-bit registration number so each slave can be recognized and monitored easily on the bus.

A 1-Wire transaction sequence (**Figure 2**) consists of a reset pulse (t_{rst}) sent to the slave. The reset pulse puts all the slaves into a known state by holding down the 1-Wire bus for a predefined period of time. Next, the slaves acknowledge the master with a presence-detect pulse (t_{pd}) that pulls down the bus after the master releases it.

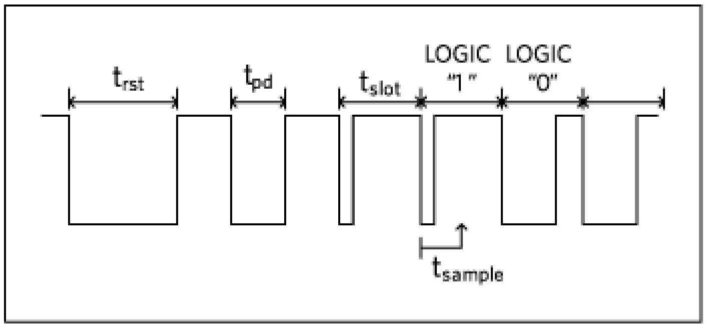


Figure 2. 1-Wire timing diagram.

After the reset, the slave devices can accept a variety of ROM commands directed to each slave identified by its registration number. A command to Match ROM activates only a single slave device with the correct identifying registration. Search ROM is used to discover the registration numbers of all slaves on the bus. Consequently, although we are only showing one slave here, the application could have multiple 1-Wire-compatible slave devices.

Once a particular ROM command is selected, then device-specific commands can be issued by the master, the Android in our example. Using a temperature logger like the Thermochron as the slave, the master's commands could involve writing or reading to its scratchpad or memory, or converting a temperature.

The 1-Wire interface does not have a clock line, so communications are split into time slots (t_{slot}), each carrying one bit of information. At the beginning of a time slot, the master briefly pulls down the bus to indicate the start of a bit. When transferring a zero, the master or slave continues to hold the bus low. When transferring a one, the master or slave releases the bus. The master or slave reads the bus at a defined time (t_{sample}) after the master indicates the start of the time slot.

USB Communication with the 1-Wire Adapter

The DS9490R is the 1-Wire to USB adapter and it has four USB endpoints: control, interrupt, bulk input (`epIN`), and bulk output (`epOUT`). Generally, the control endpoint is used to send commands to the 1-Wire adapter and to set up the type of transfer. The bulk input/output is used for data transfers, and the interrupt endpoint is to receive time-sensitive information such as status registers and return messages.

Using the Android as a USB Host

There is precedent for the design presented here. The Android API, starting from version 3.1, supports USB host mode. Manuel Di Cerbo[1] connected an Arduino® microcontroller board with an Android phone over USB. Our application modifies DiCerbo's work, extends the basic concepts to the DS9490R USB to 1-Wire adapter, and uses the 1-Wire adapter with the Thermochron instead of the microcontroller.

The DiCerbo example code is the foundation for this project. The code first asks permission from the user to access the USB devices connected to the Android smartphone. The program then searches the vendor and device id and sets up the USB endpoints for communication. The code provides `conn`, the USB device connection used to perform bulk and control transfers; it sets up the USB endpoints `epIN` for bulk input and `epOUT`. This is the same basic setup used in all systems to initialize USB and write low-level USB commands to a slave.

Now we will demonstrate how to use the Android to perform a temperature conversion on the Thermochron and to read the temperature result. Each transaction step (**Table 1**) begins with a `1-Wire Reset`, followed by `Match ROM` to select the slave device, then followed by a final device-specific command.

Table 1. Commands Performed by the 1-Wire Master to the Thermochron Data Logger

1-Wire Reset	Match ROM	Convert Temperature
1-Wire Reset	Match ROM	Read Memory/Register

The 1-Wire reset is performed by a USB control transfer, and the control transfer function prototype from the Android API is shown below.

```
// Performs a control transaction on endpoint zero for this device.
int controlTransfer(int requestType, int request, int value, int index, byte[
] buffer, int length, int timeout)
```

The control transfer is used to initiate `1-Wire Reset`, `Match ROM`, or `Block I/O`. The parameters are described in the data sheet. Later, we will illustrate how this function is used.

A bulk transfer is used for `Match ROM` and read/write memory. Here the endpoint would be either `epIN` or `epOUT`, depending on whether we are reading or writing data. The endpoint buffer either stores data to be sent, or is empty for storing the received data. The length is the number of bytes received or sent. Timeout is the USB timeout setting in milliseconds.

```
// Performs a bulk transaction on the given endpoint.
int bulkTransfer(UsbEndpoint endpoint, byte[] buffer, int length, int timeout
)
```

The code to send the convert temperature command (0x44) to the Thermochron is shown here. First in line 1, the 1-Wire reset is sent by a control transfer, as previously described (Table 1). This is the `1-Wire Reset`, `Match ROM`, and it converts the temperature sequence.

```
// 1-Wire Reset
1       conn.controlTransfer(0x40, 0x01, 0x0C4B, 0x0001, null, 0x0000, 0);
// Match ROM, where romid is the iButton's registration number
2       romid = new byte[] {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
3       conn.bulkTransfer(epOUT, romid, 8, 0);
4       conn.controlTransfer(0x40, 0x01, 0x0065, 0x55, null, 0, 0);
// Convert Temperature for DS1921G
5       data = new byte[]{0x44};
6       conn.bulkTransfer(epOUT, data, data.length, 0);
7       conn.controlTransfer(0x40, 0x01, 0x1075, data.length, null, 0, 0);
```

In line 4 above the match access control transfer sends a `1-Wire Reset`, 0x55, match access ROM command on the 1-Wire bus, followed by the desired slave's ROM registration number. [2]The indexparameter is set to 0x55. The match access command expects the user to preload the registration number to `epOUT`, as shown in lines 2 and 3 of the code. The function parameters are described in the DS2490 data sheet.

The data sheet for the Thermochron identifies 0x44 as the code for beginning temperature conversion. (**Table 2**). The convert temperature command is performed by writing 0x44 using a block I/O operation. For a block I/O operation, write the output data to `epOUT` as shown in line 6 above. The control transfer in line 7 then executes the block I/O command.

Table 2. Thermochron Memory and Control Commands (using bulk I/O)

Memory/Control Commands	Command Code	Description
Read Memory	0xF0	Read the data from the internal register. Follow the command by the register address, with LSb first. Continue with 0xff dummy data for each byte read.
Convert Temperature	0x44	Begins the temperature conversion.

The code below shows the sequence for reading the temperature register data with a USB bulk I/O transfer. The Thermochron's read memory command code is 0xF0 (see Table 2). This is followed by the target register address (TA) of 0x0211, which has read-only access and is split into two bytes (line 8). Data written and sent on the 1-Wire bus is all looped back to the master, since the bus consists of only a single wire. The master then needs to write dummy data (0xff) to the bus. The slave responds and overrides the 0xff data because, as noted much earlier, the 1-Wire is an open-drain bus. The net effect is the ANDing of the data with 0xff.

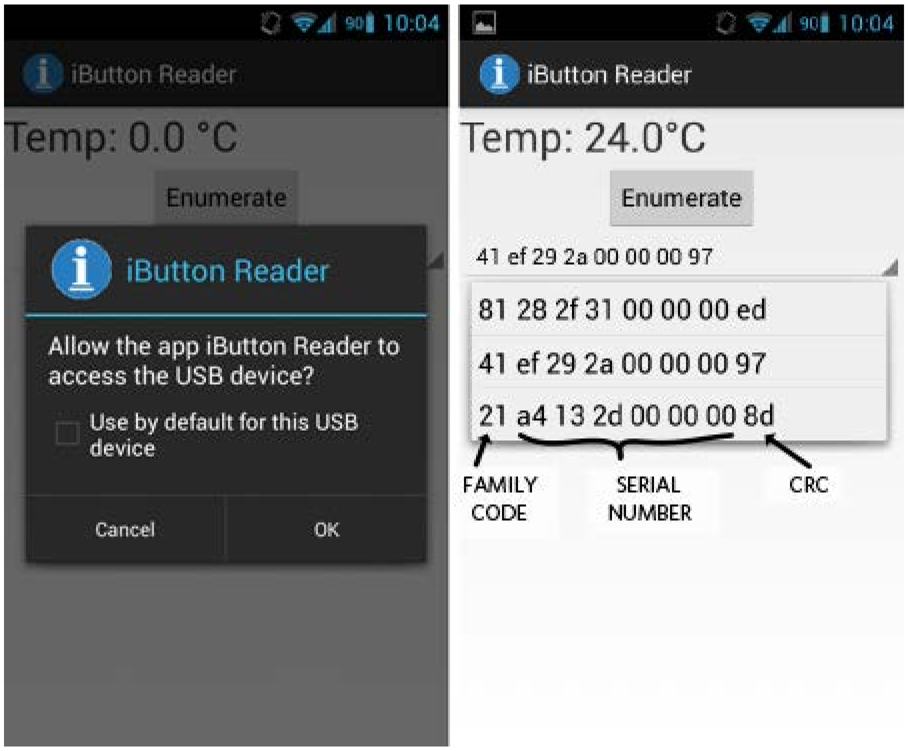
These series of commands are put on to `epOUT` and the control transfer executes the command by sending it to the 1-Wire bus (lines 9 and 10). The read back data is in the USB endpoint,`epIN` . This is copied to `tempdata` with the bulk transfer command on line 12. The resulting raw temperature code is then converted into the corresponding temperature value on line 13.

```
// 1-Wire Reset and Match ROM
// (omitted) ...
// Read Temperature Register/Memory Command
//          Read Memory, TA2, TA1, dummy data
8       command = new byte[] {(byte)0xf0, 0x11, 0x02, (byte)0xff, (byte)0xff}
;
9       conn.bulkTransfer(epOUT, command, command.length, 0);
10      conn.controlTransfer(0x40, 0x01, 0x1075, command.length, null, 0, 0);
// Return Data from input endpoint
11      byte[] tempdata = new byte[5];
12      conn.bulkTransfer(epIN, tempdata, 5, 0);
// Temperature calculation
13      temperature = (int)(tempdata[4] & 0xff)/2.0 – 40;
```

Our Sample Android Application

The sample Android application is shown in **Figure 3**. When the user runs the program and presses the Enumerate button, a screen is displayed asking for permission to access the USB device. Once the user taps OK, the application performs a search and lists all the slaves' 64-bit registration numbers in the dropdown menu. When the user selects a specific registration number, then the application performs the routine detailed above and displays the real-time temperature from the Thermochron. The enumerate button and selection of registration number are connected to spawn an`AsyncTask` . These tasks are threads that complete the requested action and update the user interface (UI) when done. The tasks are not performed in the UI thread to prevent it from freezing while waiting for the iButton to respond.

In the final application the code is abstracted into general 1-Wire operations such as `oneWireReset()`, `oneWireMatchRom()`, `oneWireSearch`, `oneWireWrite()`, and `oneWireRead()`. Also, these commands together can be further abstracted into iButton functions such as `convertTemperature()` and `readMemory()`. This allows `AsyncTask` to call the correct function.



Allow the app iButton Reader to access the USB device?

Use by default for this USB device

```
41 ef 29 2a 00 00 00 97
81 28 2f 31 00 00 00 ed
41 ef 29 2a 00 00 00 97
21 a4 13 2d 00 00 00 8d
```

Figure 3. The Android application, USB permissions (left). Temperature measurement for a specific Thermochron iButton is identified by its registration number (right).

The sample application contains the UI code for linking the user interactions, such as pressing a button, into executing a command to the 1-Wire slave. In addition to the DS1921G Thermochron, the application supports the iButton temperature logger (DS1922L/DS1922T) and iButton Hygrochron temperature and humidity logger (DS1923). As **Figure 3** shows, the registration number for each device is separated into three fields: family code, serial number, and cyclic redundancy check (CRC). The family code can be read from each registration number to determine the exact device model.

Conclusion

The code for this sample application is programmed using Eclipse and is available for download. After accessing the whole project file, it is easy to explore and modify the source code. The project code uses the abstracted functions to indirectly use the low-level USB commands. The **General References** at the end of this article are useful for understanding the Android USB API and 1-Wire commands. These resources were used extensively for the development of this sample application. Possible modifications could change the application to work with other 5V 1-Wire slaves such as memories, or to enable additional iButton features. The customizable options are numerous and the end result depends on a designer's system requirements.

Appendix

Table 3 shows the parameters needed to perform a 1-Wire reset, match access, and block I/O. The request type is defined by the USB specification. The other fields are defined in the DS2490 data sheet. The request ID is the command type, such as control, 1-Wire communications, or configuring the mode. The value parameter is a bit field, which defines the specific command and its settings. The index field provides additional command parameters, if needed, such as the number of read/write bytes in a block I/O command. An example command parameter is to enable the internal strong pullup after an I/O command. The remaining fields do not carry any information useful for the discussion.

Table 3. Control Command Parameters for Reset, Match Access, and Block I/O from the DS2490 Data Sheet

Name	1-Wire Reset	Match Access	Block I/O	Description
Request Type	0x40	0x40	0x40	Defined in the USB specification.
Request	0x01	0x01	0x01	0x01 for communication command.
Value	0x0C4B	0x0065	0x1075	Defines the operation to perform with parameters' bits.
Index	0x0001	0x0055	Block size	Additional command parameters.
Buffer	Null	Null	Null	Not used for this example.
Length	0	0	0	Length of buffer.
Timeout	0	0	0	USB timeout.

Endnotes

1. Di Cerbo, Manuel "Android USB Host + Arduino: How to communicate without rooting your Android Tablet or Phone," **Using Android in Industrial Automation**, February 9, 2012.
2. In this article we assume that the user already knows the 1-Wire registration number engraved on topof the Thermochron. The unique id number for the Thermochron or other iButton device can also be found electronically through a search algorithm. The search ROM operation performs a binary tree search operation and is detailed in Maxim Integrated application note 187, "1-Wire Search Algorithm." The Android iButton example project does include this algorithm.

General References
Android API, Package Index

Maxim Integrated iButton: 1-Wire Public Domain Kit

Related Parts	
DS1402D-DR8	1-Wire Network Cables
DS1921G	Thermochron iButton Device
DS1922L	iButton Temperature Loggers with 8KB Data-Log Memory
DS1922T	iButton Temperature Loggers with 8KB Data-Log Memory
DS1923	iButton Hygrochron Temperature/Humidity Logger with 8KB Data-Log Memory
DS9490R	USB to 1-Wire/iButton Adapter

More Information
For Technical Support: http://www.maximintegrated.com/en/support
For Samples: http://www.maximintegrated.com/en/samples
Other Questions and Comments: http://www.maximintegrated.com/en/contact