Keywords: iButton, 1-Wire, OneWire, 1wire, ultra-reliable, communication, techniques, touch, contact, intermittent, tear

APPLICATION NOTE 159

Software Methods to Achieve Robust 1-Wire® Communication in iButton® Applications

Sep 22, 2008

Abstract: 1-Wire devices communicate using a single data line plus ground reference. The 1-Wire protocol includes special provisions to handle highly intermittent ("touch") connections that are common when using 1-Wire devices in the stainless steel iButton package. This application note explains when special attention to data integrity is advised, and discusses techniques to achieve the most reliable communication in iButton applications. The reader should be familiar with the 1-Wire bus protocol and the methods to generate 1-Wire communication using a microcontroller. Although the main focus of this document is on applications with unreliable connections, some of the methods described can improve the reliability in hardwired 1-Wire applications.

Introduction

1-Wire devices communicate using a single data line plus ground reference. The 1-Wire protocol together with additional built-in features make iButtons suitable for applications that need to cope with highly intermittent ("touch") connections. When iButtons are used for applications such as high-security identification or monetary transactions, then highly reliable communication is most critical for successful operation.

General 1-Wire device communication involves several functions: searching (to identify the devices present on the bus), reading device identification number (also known as network address, registration number, 64-bit unique ID, and 64-bit lasered ROM), reading device data or status, and writing memory data or control information. In some cases, it is simple for the software to detect and correct communication failures. When a failure occurs while reading, for example, the software may attempt to read the iButton again, or may rely on the user to remove and reconnect the iButton to start a new transaction. Such a corrective action causes only a minor delay or annoyance to the user.

Writing to an iButton, however, is a much more critical task. If the iButton has departed before a read back to verify, one may not even know that the data was miswritten. If the data was written in error, there may not be an opportunity to rewrite it. The consequences of an unsuccessful write can be quite serious. If an iButton contains monetary data, debits (purchases) usually include writing revised monetary amounts to the iButton each time that a purchase is made. Should one of these updates fail and leave the data in an undefined state, the user may lose the entire monetary balance.

Programs that perform 1-Wire communications over intermittent connections must have a well thought-out plan for handling retries when failures occur. The software developer must consider everything that could have occurred during the failed attempt. Detecting errors sooner rather than later yields a faster system response and reduces the risk of updating data in the wrong (swapped) iButton. This rapid action, in turn, improves the confidence of the users in the technology. Whenever customer satisfaction is critical, the extra step to make 1-Wire communication most robust is strongly advised.

Error Mechanisms

The 1-Wire bus is a wired-OR arrangement. The idle state of the bus is logic high. For communication, the 1-Wire bus is pulled down by low-impedance drivers in the master and slave devices (iButtons), and returns to a high level due to weaker pullup currents provided by the master. Besides communication initiated by the master, the bus can be pulled low from presence pulses generated by arriving iButton devices and electrical short circuits caused when an iButton is improperly seated in the probe.

An external source of interference can turn a logic 1 bit into a logic 0 bit by short-circuiting the bus. A logic 0 bit can be turned into a logic 1 bit, if the interference has sufficient energy to overcome the low-impedance driver generating the 0 bit. For this reason, the probability of a logic 1 being turned into a logic 0 is much higher than the reverse. Loss of contact when the slave returns a 0 bit will result in a 1 bit read by the master. Slave devices can, moreover, mistake a noise pulse of either polarity injected into the 1-Wire bus as a new time slot, which causes all the subsequent bits to be returned out-of-step with the master. The software developer must be aware of bit errors such as inverted bits, dropped bits, extra bits, and shorts. In iButton applications, the bus should always be presumed unreliable; every precaution should be taken to ensure data integrity.

Error Detection/Prevention Methods Provided by iButtons

The cyclic redundancy check (CRC) is one of the error detection methods used by all 1-Wire devices and iButtons. An 8-bit CRC is part of each device's identification number to verify error-free transmission. Memory iButtons often have a built-in 16-bit CRC generator to safeguard data when reading from the device. The 1-Wire file system (see application note 114, "1-Wire File Structure") appends a 16-bit CRC at the end of each data record. To compute this CRC, the CRC generator is preloaded with the page number. Should an error occur in the transmission of the address bits that select the memory page, the CRC verification will fail when the wrong page was accessed.

Although CRCs are a powerful tool to discover the presence of bit errors, they also have a weakness: the CRC of all zeros is zero. This means that a shorted bus (which returns all zero bits) shows a valid CRC, despite the fact that the data is entirely in error. Therefore, additional checks or redundancy mechanisms must be used, such as verifying the validity of the family code contained in the first byte of the identification number. To help alleviate the all-zeros CRC problem, a bit-wise inverted CRC is often used in iButton devices and data structures.

To prevent data from corrupting during a memory write access, memory iButtons have a temporary storage area called the scratchpad. Once the data is written to the scratchpad and a read-back confirms that it is error free, the data can be transferred to the target destination in an uninterruptible operation. The data transfer mechanics from the scratchpad to the target location depend on the memory technology. See the sections **Reliably Writing to NVSRAM** and **Reliably Writing to EEPROM** below for details.

Communication Stages Overview

Taking a closer look at the activities of an iButton write transaction, one can distinguish the following stages:

Communication Stage	Challenges to Address
Initial contact of iButton and probe	Contact bounce
Generating time slots to read device identification number(s)	Shorts and intermittent contact
Maintaining a list of devices present on the bus	Contact bounce; multiple devices on the bus; devices coming and going
Selecting (addressing) a particular device	Devices coming and going
Reading from the selected device	The device may no longer be on the bus
Writing to the selected device	Verification of write success; the device may no longer be on the bus

These are the tools and methods that the software developer can deploy to achieve reliable data exchange:

- Read back during write time slots (always)
- CRC of device identification number (always)
- Validity of family code (always)
- Multiple (redundant) reads and read-back verification of data that has been written (always)
- CRC-16 embedded in data records (memory iButtons)
- Device-generated CRC-16 (some memory iButtons)
- Cryptographic validation of data and device identification number in monetary applications

The following section details how to use these tools during the various communication stages.

Making the Communication Stages Reliable

System Interrupts

To eliminate internal errors (i.e., errors inadvertently caused by the system software design), it is important to manage system interrupts properly. A careful review should be made of low-level 1-Wire communication routines to ensure that interrupts are prevented during these time-critical operations. When tested with an iButton firmly affixed at the probe (i.e., all external sources of error removed), the application software should function without any retries. Only after this is proven should the software developer's attention turn to error-handling methods.

Contact Bounce

To measure the contact bounce, a test was conducted with a short-circuited iButton arriving at a 1-Wire reader probe (1kΩ pullup to 5V). **Figure 1** shows an oscilloscope trace of a 100-sample average of repeated contacts. As one can read from the graphic, the electrical contact between the probe and iButton does not stabilize until approximately 12ms after the initial contact. More than 40% of the iButton arrivals in this test had no stable connection even 6ms after the initial contact. The waveform in **Figure 2** shows the contact bounce of a single iButton contact on the probe. Note that contact bounce in this example is severe in the first 4ms to 6ms after contact.

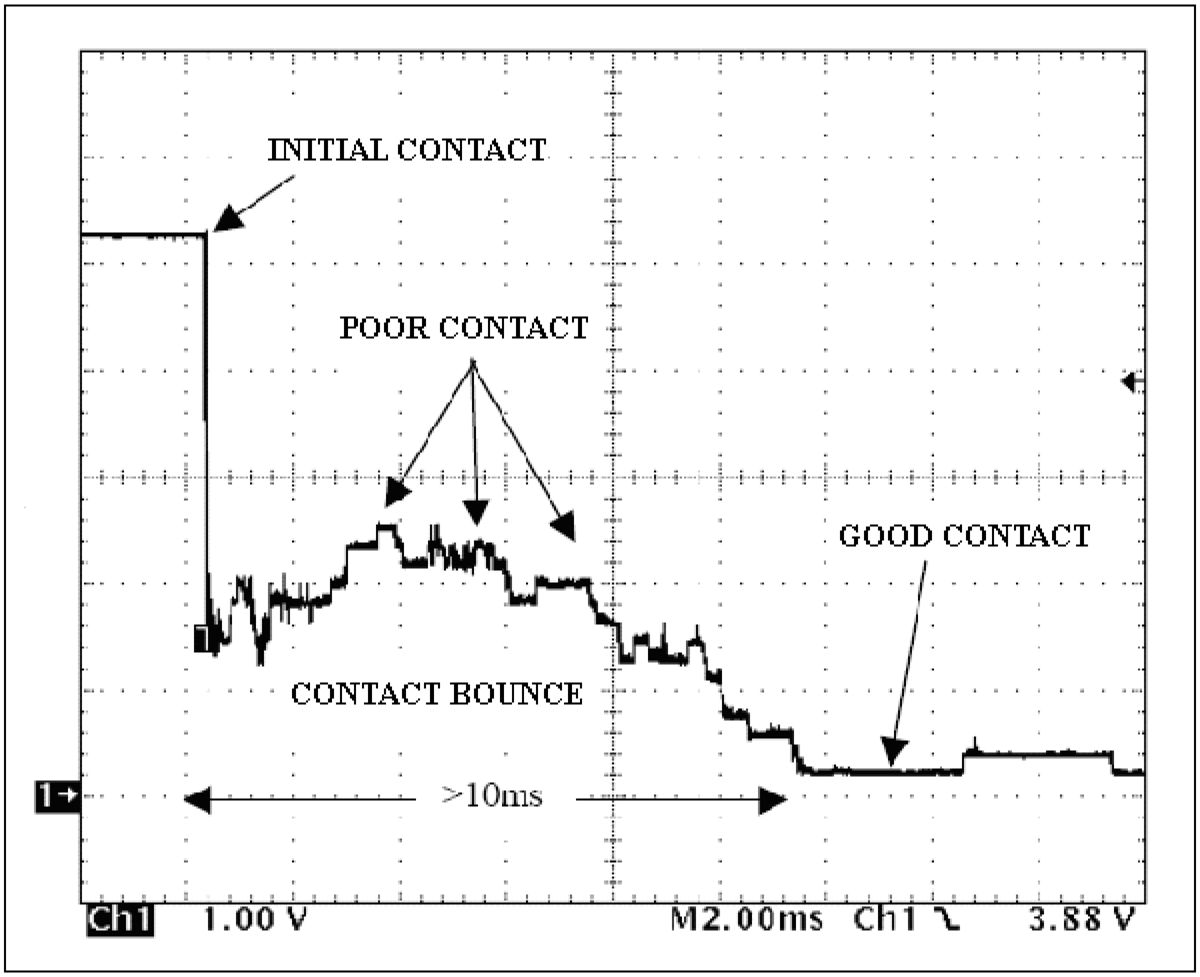

Figure 1. Contact bounce averaged from 100 contacts using a shorted iButton.

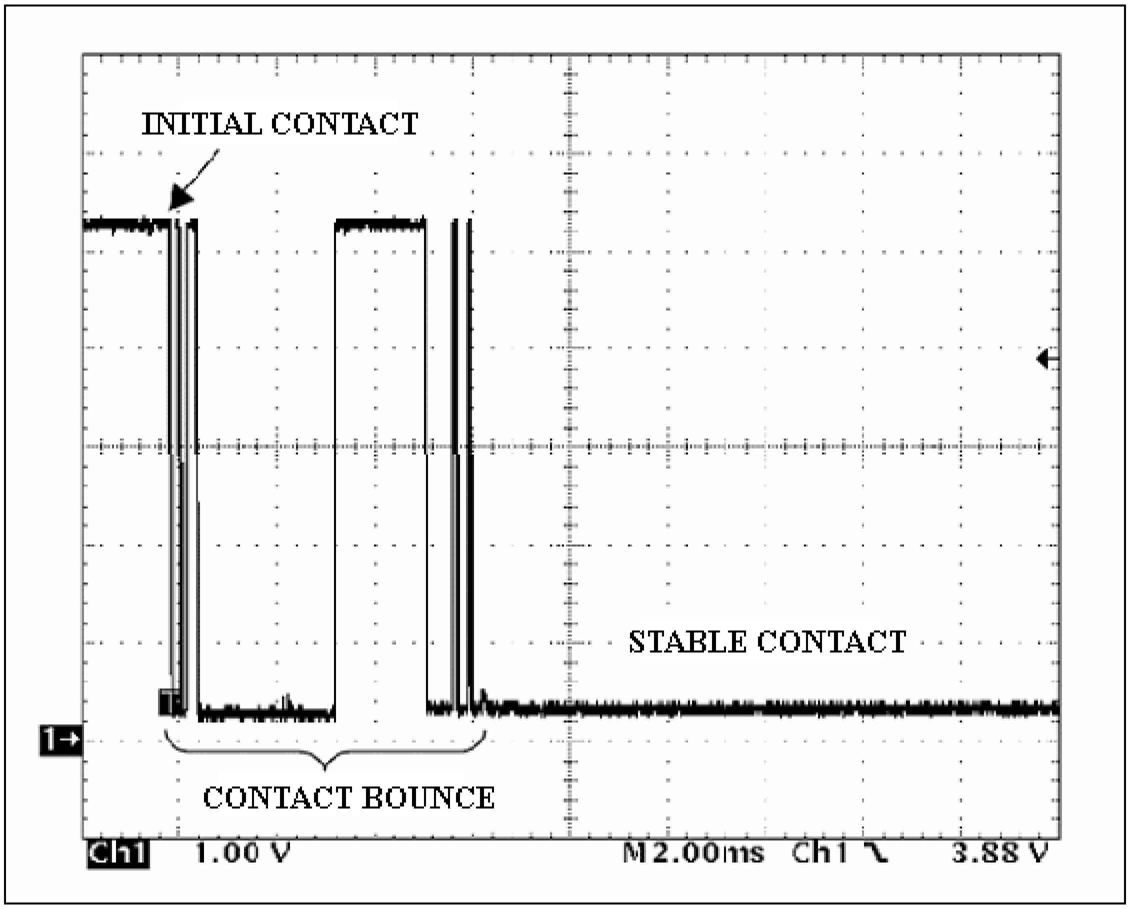

Figure 2. Contact bounce of a single iButton arriving at the probe.

When an iButton arrives at a reader (probe), it generates a presence pulse of 60µs to 240µs. A reader that tests the state of the 1-Wire bus at a rate of 17000 samples per second (~60µs) or higher will most likely detect the activity caused by the presence pulse. Once the reader becomes aware that an iButton has made contact at the probe, it usually attempts to read the identification number of the device. A read ROM sequence at standard speed takes about 6ms to complete. A program that tests for the arrival of an iButton and then immediately attempts to read the identification number will probably fail. For success, a 10ms waiting time followed by multiple reset/presence detect cycles and repeated read ROM sequences is necessary to reliably read the identification number.

Shorts and Open Circuits

Besides intentional sabotage with a coin or piece of metal, shorts typically occur when an iButton is placed on an iButton probe and the data contact of the iButton bridges the probe's data and GND contacts. The duration of such a short can vary from a few milliseconds up to a second or even more. Although a short does not damage 1-Wire devices on the bus, it causes a communication reset which, in turn, resets the 1-Wire data rate to standard speed. If the master were using overdrive speed and was unaware of the short, subsequent communication attempts at overdrive speed would fail. From an iButton's perspective, loss of contact is equivalent to a short; in either case there is no voltage between data and GND. Consequently, the reset to standard speed also occurs if the connection between the iButton and probe is lost.

The appropriate places to implement short-circuit checks are bus-reset and time-slot subroutines. Testing for a short should always be done just prior to driving the bus low at the start of a bus-reset or time slot, i.e., when the bus has had ample time to recover from any previous low-level pulse. If the bus is tested for a short too soon after it has been released from a low, the voltage may not have risen to a valid high level because of the capacitance of the bus from cable, probe, and iButton(s). In the extreme, a rise time of several microseconds is permissible. Special attention is needed with bus-reset subroutines. To distinguish between short and presence pulse, the logic level on the bus must be tested multiple times during the presence detect window. After a valid presence pulse of 60µs to 240µs, the bus returns to logic-high level; if there is a short, the bus remains at logic 0 for an extended time.

Generating Time Slots

1-Wire communication occurs in time slots. There are write time slots and read time slots. Each time slot transfers a single bit. For the most part, a read time slot is just a write-one time slot where the slave device turns the 1 bit into a 0 bit when responding with a 0. Typically, a subroutine that generates time slots performs the functions of reading and writing simultaneously. Such a time-slot subroutine (see **Figure 3**) consists of nine steps, as explained below. The timing values refer to standard speed.

1. Take the bus LOW to start the time slot.
2. Wait 6µs.
3. Output the bit to be written to the bus. If a 1, release the bus and allow it to return high. If a 0, continue to hold the bus low.
4. Wait 9µs.
5. Sample the level on the bus at 15µs after start of slot (low level = 0, high level = 1).
6. Wait 45µs.
7. Allow the bus to float high at 60µs after start of slot. This ends a write-zero in progress.
8. Wait 10µs for bus recovery time.
9. Return the sampled bit.

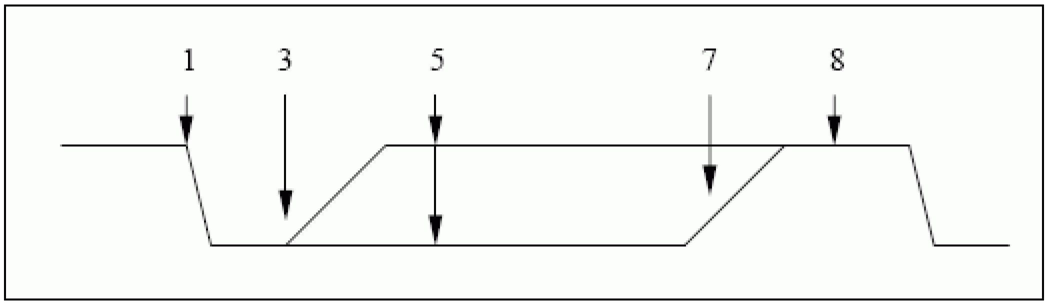

Figure 3. Creating a 1-Wire time slot, the numbers correspond to steps above.

Each time a bit is written to the bus, the bus state is also read back and returned to the calling routine. Reading is done by writing a 1 and allowing the slave to modify the 1 to 0, as necessary. Therefore, a single subroutine is good for reading and writing. A byte-level subroutine simply calls the bit-level routine eight times and returns the result as a byte.

The read back during write has another advantage: it uncovers error conditions such as short circuit on the bus, an unexpected presence pulse from a new arrival, a noise pulse, or a 0 bit from a slave device that lost synchronization with the master. Therefore, for robust communication it is important that the software compare the data written to the bus with the data returned. This comparison only applies when writing to 1-Wire slaves (iButtons). When reading, the data read will naturally be different from all 1s.

The read during write function is also supported by integrated 1-Wire masters such as the DS2480B, DS2490, and DS2482. Applications on PCs may be subject to port-driver constraints and may be unable to respond to errors until the entire command transaction is completed.

Obtaining the Device Identification Number

Since the iButton touch contact should be considered unreliable, it is important to watch for communication errors and to detect them as early as possible. This allows the software to start a retry sooner, saving time and improving the performance of the application.

When arriving at a probe and following a bus reset, iButton devices generate a presence pulse. The 1-Wire master (microcontroller) may use the presence pulse or any activity such as a short on the bus to cause an interrupt, thus invoking program code to process the arrival of the iButton. In battery-powered applications, this presence pulse is used first to wake-up a sleeping microcontroller and then to invoke the arrival routine. Alternatively, a reader (master) may simply poll, or issue repeated reset pulses, and watch for the appearance of a presence pulse as evidence of a device on the bus. The polling method, however, is applicable only in systems that, by design, cannot have more than one iButton on the bus. If this condition is not met, a more elaborate method is required. See the section **Maintaining a List of Devices Present on the Bus** for details.

When the master has detected the presence pulse from an iButton at the probe, there are two ways to learn the device identification number: the Read ROM method and the Search ROM method.

The **Read ROM method** uses the Read ROM command. This method yields a valid result only if a single 1-Wire device is on the bus. The key to verifying a correct reading is the CRC at the end of the identification number. In case of bit errors caused by temporary loss of contact or short, the CRC read will not match the CRC computed by the master, thus indicating that the reading was not successful. Executing the Read ROM command without an iButton at the probe results in an identification number of eight FFh bytes. The CRC check unveils the error. Reading the ROM while the probe is shorted yields an identification number of eight 00h bytes. In this case, the CRC check fails. However, with the knowledge that a 00h family code is not assigned, the data is identified as invalid. Since there is a 1 in 256 chance of a valid CRC despite a bit error, it is advisable to read the identification number again and to compare the two results for a match. This action will ensure that the reading was error free.

The **Search ROM method** uses the Search ROM command, which is a more interactive process to read the identification number. Search ROM uses a binary search technique in which the iButton participates in a two-way communication over 192 time slots. In this way, bit errors such as disconnect, short, and losing synchronization are easily detected. A single pass through Search ROM takes slightly more time than two passes of Read ROM. Successful completion of Search ROM, even without checking the CRC, is a strong indication that a device is present and ready for subsequent communications. For more information on the use of the Search ROM technique see application note 187, "1-Wire Search Algorithm."

Maintaining a List of Devices Present on the Bus

Many applications require the reliable detection of iButton arrivals at and departure from the bus. To detect arrivals and departures, one needs a table of the devices known to be present on the bus plus some management information. One component of the management information is called "age." This age information is used for debouncing and to decide whether a device has departed from the bus. Debouncing reduces the number of false arrivals/departures reported due to read errors. One scheme to detect arrivals and departures is described by the following algorithm:

1. The bus is continually scanned using Search ROM to discover each identification number in turn. Without any errors, one scan cycle requires as many Search ROM passes as there are devices on the bus.
2. After a device has been identified, the device table is referenced to determine whether the device has been found in a previous cycle. If the device is not listed in the table, then it just arrived. It is added to the table and reported as a new arrival.

3. When a device is added to the table and when a device just identified is already in the table, the age data for that device is set to "n" (n represents a numeric value, such as 5).
4. Before a new scan cycle begins, the age information for all devices in the table is decremented. If an age value reaches zero, then the device has not been observed in n scan cycles. This device is removed from the table and reported as a departure.

To implement this scheme, one needs a table with a 9-byte entry for each device. The table must be large enough to store as many entries as there can be devices on the bus.

To increase the reliability of the arrivals/departure algorithm, it is necessary to test the validity of the family code, to check for a valid CRC, and to make new arrivals tentative. The expanded algorithm looks like this:

1. The bus is continually scanned using Search ROM to discover each identification number in turn. Without any errors, one scan cycle requires as many Search ROM passes as there are devices on the bus.
2. After a device has been identified, the identification number is checked for nonzero family code and for a valid CRC.
3. An identification number with valid family code and CRC is looked up in the device table to determine whether the device was found in a previous cycle.
4. If the device is not listed in the table, then it just arrived. It is added to the table. The age data is set to 3, and a flag is set to mark the arrival as tentative.
5. If the device is already listed in the table and flagged as tentative, then the age value is incremented by two. If the age value is greater than 3, then: the age byte is set to n; the tentative flag is cleared; and an arrival event is reported.
6. If the device is already listed in the table and not flagged as tentative, the age value is incremented by 1 up to, but not beyond, n.
7. Before a new scan cycle begins and the bus is not shorted, the age information for all devices in the table is decremented by 1. If an age value reaches zero, then the device has not been observed in n scan cycles. This device is removed from the table, and, if the device was not flagged as tentative, reported as a departure.

To be reported as an arrival, a device must be discovered in two consecutive scan cycles. To be reported as a departure, a device must not be found in n consecutive scans cycles. If the bus is performing well, then none of the entries is flagged as tentative, and the age values are equal to n. Devices that are occasionally missed in a scan cycle have age values lower than n. This algorithm has proven most reliable in reporting arrivals and departures from the 1-Wire bus. If a short is detected on the 1-Wire bus, then the arrivals/departures algorithm may be suspended until the short is removed. Without this exception all devices could be marked as departures in a short amount of time.

Reliably Addressing a Device

Once the identification number of a device that has newly arrived at the probe has been verified as correct, this number is used to select the device for subsequent communications. In systems where only one device may exist on the bus, one may be tempted to use Skip ROM instead of the Match ROM. This is not advisable, because the device that was identified before could have departed and a different device could have arrived. Therefore, it is necessary to begin each transaction at least using a Match ROM sequence.

Besides Match ROM, there is another method called "Strong Access" to address a 1-Wire device. Strong Access uses the Search ROM function with a predetermined direction for every bit in the sequence. In other words, the device is discovered in a search that already knows the device that it expects to find.

The end result is the same as with Match ROM, but Search ROM requires 128 correct bits of feedback from the device, thereby assuring that the device to be addressed is still present. Since a missing device cannot return the proper responses, the Search ROM method discovers that a device is missing within a few bit times.

To decide whether Match ROM or Search ROM is the right choice for a given application, it is necessary to know the system response for the two methods. Match ROM at standard speed takes approximately 5.6ms to complete (0.7ms at overdrive speed). A Search ROM sequence requires approximately 14ms to complete (1.7ms at overdrive speed). For an application that requires 10 access cycles to complete a transaction—for example, the Strong Access method adds 84ms at standard speed—this may not be accepted by the users. Newer 1-Wire devices support a ROM function command called "Resume." This command takes the same time as Skip ROM, but ensures that maximum one device can respond to the subsequent memory or control function command. This eliminates the risk of accidentally deleting data if the target device has departed from the bus. In general, if a short system response time is critical for user acceptance, one should consider using overdrive speed. See the section on **1-Wire Speed Considerations** for additional information.

Reliably Reading from a Device

Once a device is reliably addressed, the master sends: the Read Memory command; the memory target address; and as many read time slots as needed to read the memory data of interest. If the Read Memory command was received error free, errors could still corrupt the transmission of the memory target address and affect the memory data on its way to the master. If the Read Memory command was not received error free, the iButton would, with high probability, ignore the command.

The memory of iButtons is organized in pages of 32 or 64 bytes. With the exception of the DS1920, DS1971, DS1990, and DS1991, iButtons are compatible to the 1-Wire File Structure described in application note 114 mentioned above. This file system puts a length byte at the beginning of each page and appends a continuation pointer and inverted CRC16 at the end of the data field. This system provides a high level of error control when reading iButton memory data. Since the CRC generator is preloaded with the memory page number, the 1-Wire File Structure not only safeguards against bit errors in the data stream, but also uncovers target address errors. When using the DS1971 or DS1991, data packets should be formatted similar to that described in application note 114.

Besides their compatibility with the 1-Wire File Structure, the following iButtons have a built-in CRC generator to safeguard memory read access: DS1921, DS1922, DS1923, DS196x series, DS1977, and DS198x series. Except for the DS1982, the device-generated CRC is a 16-bit type. The built-in CRC generator, which usually requires an extended read command for activation, inserts a CRC after the last byte of a memory page is transmitted. When relying on this CRC generator, the page data does not need to be formatted (saving 4 bytes per page vs. the format in application note 114). If the page data is formatted, the device-generated CRC provides another layer to verify the data integrity.

If a read error is uncovered, the standard remedy is to read the same data (or page) again. Multiple reading is also advised if memory data is unformatted and the iButton does not generate a CRC.

Some applications may require that the data be encrypted or that a Message Authentication Code (e.g., a SHA-1 MAC) be embedded in the page. One could argue that a failed decryption or invalid MAC is an indication of a read error. Although this is true, it is imprudent to set up a system this way, since one could not distinguish between read error and intentional tampering to commit fraud. Therefore, the data should first be checked for physical integrity using the means explained above, and then, in a second step, the cryptographic validation should be applied. Only then can the appropriate action be taken and valid statistics be collected.

Reliably Writing to NVSRAM

NVSRAM-based memory iButtons have a built-in battery that provides backup of the scratchpad data and energy to complete the transfer to the target memory, regardless of the activities on the 1-Wire bus during the transfer. To ensure that the communication was successful, it is necessary: 1) to verify the data in the scratchpad, 2) to read back the memory page that was updated, and 3) to compare with the intended data.

The entire process is best explained by an example, such as a purchase at a vending machine. In this case the data is encrypted for security. The steps below use functions explained on preceding pages. For more details refer to the eCash Evaluation Kit (part number: DSECASH).

1. Detect the iButton arrival.
2. Obtain the iButton's identification number and check for integrity (i.e., family code, CRC).
3. Reliably address the iButton. If addressing fails, **END** (failure).
4. Reliably read the page that stores the monetary balance. This includes verification of data integrity. (Using the 1-Wire data structure, this step requires reading multiple pages, sequentially or randomly.) If there is no success after multiple attempts, wait for iButton departure. **END** (failure).
5. Cryptographic validation of page data, check for sufficient funds. If steps fails, display error message, wait for iButton departure. **END** (failure).
6. Reliably address the iButton. If addressing fails, **END** (failure).
7. Write new encrypted balance to scratchpad. Depending on iButton model, byte offset, and number of bytes written, the Write Scratchpad command may return a CRC for transfer integrity check. If so, use this CRC accordingly.
8. Reliably address the iButton. If addressing fails, **END** (failure).
9. Read scratchpad; compare address and data for match with data written to the scratchpad. If no match, go to 6).
10. Reliably address the iButton. If addressing fails, **END** (failure).
11. Issue Copy Scratchpad command, wait 32µs.
12. Reliably address the iButton. If addressing fails, **END** (failure).
13. Read updated page, compare to intended data. If data matches, dispense product; **END** (completed). If no match, go to 8).

This example assumes an iButton without special security features. The DS1963S has several write cycle counters that increment with every write access to an associated memory page. The counter value can be included in the encryption to create unique instances of the monetary value, e.g., to prevent replay attacks. As a side effect, the recovery from errors such as writing the same data twice is more complicated. Attempts to recover from a failed write must go back several steps so that the new counter value can be properly included in the encryption of the data.

Reliably Writing to EEPROM

EEPROM iButtons also use a scratchpad for temporary storage and verification of new data. However, since they do not have a battery as backup, their scratchpad is volatile. If the contact breaks before the 10ms memory write cycle of a copy scratchpad command is completed, the target memory location may be erased. If EEPROM iButtons are used in an application that requires writing to the memory, two write accesses are necessary to maintain the old data in case the update cycle cannot be completed. It is absolutely necessary to verify the data in the scratchpad and to read back the updated memory page for comparison with the intended data. For EEPROM iButtons, it is highly recommended to implement the 1-Wire File Structure as described in application note 114, since that method provides the only way to check memory data for integrity. The device-generated CRC, if available, safeguards only against communication errors. A complicating factor with EEPROM iButtons is the scratchpad size, which fits only ¼ memory page with the DS1961S and DS1972. The example below applies to a device with a scratchpad size equal to a memory page size.

Example Starting Conditions

The device (e.g., DS1973 or DS1977) is formatted according to application note 114, i.e., page 0 stores the device directory. The single-page data file alternates between pages 1 and 2. The file's directory entry tells whether page 1 or page 2 is valid. One byte within the data file is reserved for the application software to use as a counter, which is incremented whenever the file is updated.

Task: Safely update the data file. Using functions explained on preceding pages, these are the steps to follow:

1. Detect the iButton's arrival.
2. Obtain the iButton's identification number and check for integrity (i.e., family code, CRC).
3. Reliably address the iButton. If addressing fails, **END** (failure).
4. Reliably read the directory page. This includes verification of data integrity and storing the directory data in a buffer. If there is no success after multiple attempts, wait for iButton departure. **END** (failure).
5. Reliably address the iButton. If addressing fails, **END** (failure).
6. Reliably read the file data. This includes verification of data integrity. If there is no success after multiple attempts, wait for iButton departure. **END** (failure).
7. Reliably address the iButton. If addressing fails, **END** (failure).
8. Write new file data to scratchpad using the **alternate memory page**. Depending on iButton model, byte offset, and number of bytes written, the Write Scratchpad command may return a CRC for transfer integrity check. If so, use this CRC accordingly.
9. Reliably address the iButton. If addressing fails, **END** (failure).
10. Read scratchpad; compare address and data for match with data written to the scratchpad. If there is no match, go to 7.
11. Reliably address the iButton. If addressing fails, **END** (failure).
12. Issue Copy Scratchpad command, wait 10ms (t_{PROG}).
13. Reliably address the iButton. If addressing fails, **END** (failure).
14. Read updated page, compare to intended data. If there is no match, go to 7.
15. Update directory information in the buffer for the **alternate memory page**; this includes updating the bitmap of used pages and a new CRC16 at the end of the directory data.
16. Reliably address the iButton. If addressing fails, **END** (failure).
17. Write the updated directory information from the buffer to scratchpad (for page 0). If Write Scratchpad returns a CRC for transfer integrity check, use this CRC accordingly.
18. Reliably address the iButton. If addressing fails, **END** (failure).
19. Read scratchpad; compare address and data for match with data in buffer. If there is no match, go to 16.
20. Reliably address the iButton. If addressing fails, **END** (failure).
21. Issue Copy Scratchpad command, wait 10ms (t_{PROG}).
22. Reliably address the iButton. If addressing fails, **END** (failure).
23. Read directory page, compare to data in buffer. If there is no match, go to 16.
24. If data matches, **END** (success).

Should the first update (file data) fail, no information is lost and the update can be tried later. If the directory update fails, then both data sets exist in the iButton but the directory is not current. Knowing the two possible locations of the data file (system knowledge) and reading both versions, the counter value in the data file tells which version is current. This way the directory data can be corrected at the next occasion when the iButton arrives at the probe.

For devices with a scratchpad of ¼ memory page size, the scheme needs to be modified. In this case, the alternating locations are the second and third quarter of the file's data page. Administrative data (i.e., continuation pointer, CRC16) leave only 5 bytes for the data and update counter. The first quarter of the memory page merely holds the length byte, which indicates whether the second or third quarter of the page is valid. In the first update cycle, the new data is written to the alternate location; in the second cycle, the length byte is updated. An example of this "A-B scheme" in a vending application using the DS1961S is found in application note 1820, "White Paper 1: SHA Devices Used in Small Cash Systems," Table 3 and related text (pages 6 to 7) and Step BF3 (pages 58 to 60).

1-Wire Speed Considerations

In many iButton applications, cable length and loading may preclude the use of the overdrive (high-speed) mode. In those applications where the embedded electronics are in close proximity to the iButton probe, however, this high-speed communication mode can greatly reduce transaction times.

There are two ways to get an overdrive-capable iButton into overdrive mode. One way uses the Overdrive Skip command, which puts all devices on the bus into overdrive. The other way uses the Overdrive Match command, which only puts the device with the matching identification number into overdrive. The device remains in overdrive mode until it gets a 1-Wire reset at standard speed, or it is disconnected from the bus. A device that arrives at the probe always starts at standard speed.

The following example shows the time savings for a simple Read ROM function.

Step	Explanation	Time at STD speed	Time at OD speed
1-Wire Reset/Presence Detect Cycle at Standard Speed		960µs	960µs
Command Overdrive Skip ROM	8 time slots	N/A	8 × 65µs
1-Wire Reset/Presence Detect Cycle at Overdrive Speed		N/A	96µs
Command Read ROM	8 time slots	8 × 65µs	8 × 8µs
Read Device Identification Number	64 time slots	64 × 65µs	64 × 8µs
	Total time	5640µs	2152µs

Despite of the extra command and reset cycle of getting to overdrive speed, using overdrive obtains the device identification number in 2152µs compared to 5640µs at standard speed. The more communication that occurs at overdrive, the more substantial are the time savings. The downside to using overdrive is the additional caution required after a short or loss of contact was encountered. See section on **Shorts and Open Circuits** for details.

In the above calculation the time-slot duration for standard speed is 65µs and 8µs for overdrive, which matches newer device data sheets. If the time slots are created in real time by a microcontroller, the timing will deviate from data sheet specifications. The section on **Generating Time Slots**, for example, shows time slots of 70µs, which is permissible as long as the sampling for the read back takes place early enough (#5 in Figure 3) and there is sufficient recovery time (#8 in Figure 3).

Summary

Applications that depend on highly reliable communication with iButtons through intermittent (touch) connections require careful design, special algorithms, and thorough testing. The program flow must be designed to provide early detection of errors, effective retry, and error recovery methods without neglecting the system response time. If the proper precautions are taken, however, highly reliable iButton communication can be achieved despite the intermittent contact.

Related Parts		
DS1904	RTC iButton	Free Samples
DS1920	Temperature iButton®	
DS1921G	Thermochron iButton	
DS1922E	High-Temperature Logger iButton® with 8KB Data-Log Memory	
DS1922L	Temperature Logger iButton with 8KB Datalog Memory	
DS1923	Hygrochron Temperature/Humidity Logger iButton with 8KB Data-Log Memory	
DS1961S	1Kb Protected EEPROM iButton with SHA-1 Engine	
DS1963S	SHA iButton	
DS1971	256-Bit EEPROM iButton®	
DS1972	1024-Bit EEPROM iButton	
DS1973	4Kb EEPROM iButton®	Free Samples
DS1977	Password-Protected 32KB EEPROM iButton	Free Samples
DS1982	1Kb Add-Only iButton®	Free Samples
DS1985	16Kb Add-Only iButton®	Free Samples
DS1990A	Serial Number iButton	Free Samples
DS1992	1Kb/4Kb Memory iButton®	Free Samples
DS1993	1Kb/4Kb Memory iButton®	Free Samples
DS1995	16Kb Memory iButton®	Free Samples
DS1996	64Kb Memory iButton®	Free Samples