Wiimote

From WiiBrew

This may require cleanup (http://en.wikipedia.org/wiki/Wikipedia:Cleanup) to meet WiiBrew's quality standards (http://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style) .

Reason: Make sections collapsable. Add a See Also section

Please improve this article (http://wiibrew.org/w/index.php?title=Wiimote&action=edit) if you can. See also Category:Articles needing cleanup.

[[wiki]]

Please wikify (http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Wikify) this article or section.

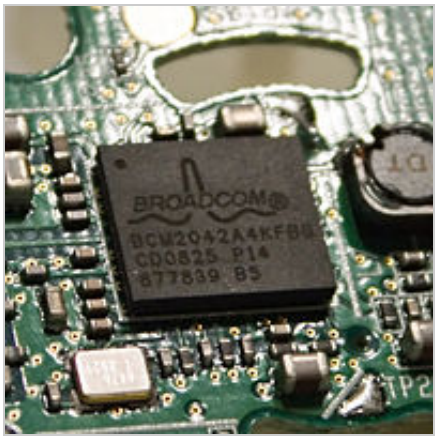
Help improve this article (http://wiibrew.org/w/index.php?title=Wiimote&action=edit) by adding relevant (http://en.wikipedia.org/wiki/Wikipedia:Only_make_links_that_are_relevant_to_the_context) internal links (http://en.wikipedia.org/wiki/Wikipedia:Build_the_web) .

This article is a technical guide to the Wii Remote. For a high-level overview of the Wii Remote, see the Wikipedia entry (http://en.wikipedia.org/wiki/Wii_Remote) .

Wii remote circuit board, top surface

Wii remote circuit board, bottom surface

The **Wii Remote** (informally known as the **Wiimote**) is the Wii's main input device. It is a wireless device, using standard Bluetooth technology to communicate with the Wii. It is built around a Broadcom BCM2042 (http://www.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions/BCM2042) bluetooth System-on-a-chip, and contains multiple peripherals that provide data to it, as well as an expansion port for external add-ons. The Wii Remote uses (and, at times, abuses) the standard Bluetooth HID protocol to communicate with the host, which is directly based upon the USB HID (http://en.wikipedia.org/wiki/USB_human_interface_device_class) standard. As such, it will appear as a standard input device to any Bluetooth host. However, the Wii Remote does not make use of the standard data types and HID descriptor, and only describes its report format length, leaving the actual contents undefined, which makes it useless with standard HID drivers (but some Wiimote Drivers exist). The Wii Remote actually uses a fairly complex set of operations, transmitted through HID Output reports, and returns a number of different data packets through its Input reports, which contain the data from its peripherals.



Broadcom BCM2042 in a Wii remote

Contents

Summary

Reverse engineering and documenting all of the Wii Remote's features is a work in progress. Here are the known features and their status:

Bluetooth Communication		Connecting to the Wii Remote and listening for connections works.
Core Buttons		All working.
Accelerometer		All working.
IR Camera		All working.
Power Button		All working.
Speaker		All working.
Player LEDs		All working.
Status Information		Battery and extension info in Status Report

Extension Controllers		Official extensions are supported, however many 3rd party extensions are not understood. See Extension Controllers page

Legend Perfect or near-perfect | Usable but not complete | Unusable

Bluetooth Communication

The wiimote communicates with the host via standard bluetooth protocol. The wiimote can be placed into discoverable mode for 20s by pressing the ***sync*** button on its back under the battery cover. Holding down the 1 and 2 button continuously will force the wiimote to stay in discoverable mode without turning off. This does not work with the *sync* button, though. When in discoverable mode, a number of the player LEDs based on the battery level will blink. With full battery all four LEDs will blink, the lower the battery the less LEDs will blink. During device inquiry the host will find all discoverable nearby wiimotes. Now the host can establish a bluetooth baseband connection to the wiimote, **no bluetooth pairing is needed**, however, if bluetooth pairing is performed, the wiimote is able to reconnect to the host if disconnected. After a bluetooth baseband connection is established (with or without pairing) the HID channels can be opened and used for reading and writing reports from/to the wiimote.

The newer Wiimote RVL-CNT-01-TR shuts down immediately upon receiving any HID output report if it has been turned on using the 1 + 2 method, although it works using the sync button. It is possible that authentication is now mandatory for the 1 + 2 temporary sync[*check*].

Bluetooth Pairing

The wiimote supports the legacy bluetooth pairing methods. This involves sending a PIN to the wiimote. Bluetooth pairing is not required to use a wiimote and you can proceed by establishing a HID connection without pairing at all. However, if the wiimote is paired, it will actively seek out for its last connected host on disconnection and reestablish the connection. The following section explains the bluetooth device pairing, if no pairing is required, skip this section.

Bluetooth pairing must be initiated by the host by sending a "Require Authentication" HCI command to its bluetooth device. The bluetooth device will ask the host for a link key, which must be rejected so it will ask for a PIN-Code. The PIN-Code is the binary bluetooth address of the wiimote backwards. Following a short piece of C code to calculate the PIN:

```
Lets assume the Wiimote has the bluetooth address "00:1E:35:3B:7E:6D". If you want the PIN for blue

char pin[6];
pin[0] = 0x6D;
pin[1] = 0x7E;
pin[2] = 0x3B;
pin[3] = 0x35;
pin[4] = 0x1E;
pin[5] = 0x00;

Now "pin" contains your bluetooth pin that should be used for pairing your devices.
```

If connecting by holding down the 1+2 buttons, the PIN is the bluetooth address of the wiimote backwards, if connecting by pressing the "sync" button on the back of the wiimote, then the PIN is the bluetooth address of the host backwards.

After sending the PIN to the bluetooth device via HCI commands, the wiimote will return a "Authentication Accepted" command and the pairing is established (both devices are **bonded** now). After pairing you continue with establishing the HID connection the *same way as without pairing*.

If the host successfully bonded with the wiimote **and established an HID connection** the wiimote will save the bluetooth address of the host and enable *single press reconnection*. That means if the wiimote is now disconnected from the host, it will actively seek out for the host if **any** button is pressed and establish a baseband and HID connection. The wiimote will never actively send pairing requests since this is not needed. Also remember that this works with **any** button not only the power-button. However, after establishing the connection, the wiimote sends a button-input-report and this allows the host to see what button was pressed. So the host may reject the new connection if any button except the power-button was pressed.

The new bluetooth pairing method SSP (Secure Simple Pairing) is not supported. Also it is not yet investigated whether a link key has to be created (by sending a PIN) on every connection or whether the link key can be saved and reused on new connections. Though, creating a new link key on every connection works fine.

The wiimote has space for several host addresses (at least 3 are known to work) so it can be paired with more than one host (like PC or Wii) and it will try in reverse order to reconnect to the hosts. That is, the last paired host is tried first and so one. If button 1 and 2 or the sync button on its back are pressed, the wiimote will not actively seek out for its host but instead place itself in discoverable mode and wait for incoming connections so bluetooth pairing does not conflict with normal host-side connections.

It is not known how to remove the hosts addresses from the wiimote, however, with some investigation it should be possible to locate them in the EEPROM and manipulate them. If this is considered a security issue, then don't pair your devices.

SDP information

When queried with the Bluetooth Service Discovery Protocol (SDP (http://www.palowireless.com/infotooth/tutorial/sdp.asp)), the Wii Remote reports back a great deal of information. In particular, it reports:

	Wii Remote/old Wii Remote Plus	new Wii Remote Plus
Name	Nintendo RVL-CNT-01	Nintendo RVL-CNT-01-TR
Vendor ID	0x057e	0x057e
Product ID	0x0306	0x0330
Major Device Class	1280	?
Minor Device Class	4	?
Service Class	0	?
(Summary of all Class Values)	0x002504	0x000508

HID Interface

Establishing a HID connection with the Wii Remote can be done on PSM 0x11 for the control pipe and PSM 0x13 for the data pipe using the Bluetooth L2CAP protocol. On Windows you don't need to deal with L2CAP yourself, and can use high-level windows HID functions.

The HID standard allows devices to be self-describing, using a HID descriptor block. This block includes an enumeration of reports that the device understands. A report can be thought of similar to a network port assigned to a particular service. Reports are unidirectional however, and the HID descriptor lists for each port the direction (Input or Output) and the payload size for each port. Like all Bluetooth HID devices, the Wii Remote reports its HID descriptor block when queried using the SDP protocol. However, no information regarding the actual data units within each report is returned, only the length in bytes.

Note: An "Input" report is sent by the Wii Remote to the host. An "Output" report is sent by the host to the Wii Remote. When using a Wii Remote, all input reports are prepended with 0xa1 and all output reports are prepended with 0xa2; this is the "(a2)" in many example reports below. Output reports are sent over the data pipe, which is also used to read input reports (thus, the control pipe is essentially unused).

The original Wiimotes (RVL-CNT-01) allowed sending commands using SET REPORT (0x52) over the control pipe, instead of using the data pipe; however, this form does not work on the newer RVL-CNT-01-TR, and as such is not recommended.

These are the reports the Wii Remote uses, and their use:

I/O	ID(s)	Size	Function
O	0x10	1	Unknown
O	0x11	1	Player LEDs
O	0x12	2	Data Reporting mode
O	0x13	1	IR Camera Enable
O	0x14	1	Speaker Enable
O	0x15	1	Status Information Request
O	0x16	21	Write Memory and Registers
O	0x17	6	Read Memory and Registers
O	0x18	21	Speaker Data
O	0x19	1	Speaker Mute
O	0x1a	1	IR Camera Enable 2
I	0x20	6	Status Information
I	0x21	21	Read Memory and Registers Data
I	0x22	4	Acknowledge output report, return function result
I	0x30-0x3f	2-21	Data reports

For clarity, the convention in this document is to show packets including the Bluetooth-HID command (in parentheses), report ID, and payload, as described in sections 7.3 and 7.4 of the Bluetooth HID specification (http://www.bluetooth.com/SiteCollectionDocuments/HID_SPEC_V10.pdf) . Each byte is written out in hexadecimal, without the 0x prefix, separated by spaces. For example,

```
(a1) 30 00 00
```

is a DATA input packet (0xa1), on channel 0x30, with the two byte payload 0x00, 0x00. When using higher level HID functions rather than Bluetooth functions, the bytes in parentheses will never be present.

Force Feedback is accessible through the first byte of ALL output reports in the same way. This is not included above to avoid clutter.

Output Report common information

The first byte in many Output reports has a similar meaning. In every single Output Report, bit 0 (0x01) of the first byte controls the Rumble feature. Additionally, bit 2 (0x04) is used in several Output Reports as the ON/OFF flag for the specific feature controlled by it. For example, sending 0x04 to Report 0x19 (Speaker Mute) will mute the speaker:

```
(a2) 19 04
```

Sending 0x00 will unmute it:

```
(a2) 19 00
```

These Output Reports share the above behavior: Data Reporting Mode (0x12), IR Camera Enable (0x13), Speaker Enable (0x14), Speaker Mute (0x19), IR Enable 2 (0x1a).

Input Report common information

The first two bytes of ALL input reports, except 0x3d, contain the Core Buttons (BB BB). This includes all the 0x2~ status reports, not just the 0x3~ data reports. 0x3d is an exception, since it only returns expansion information.

Status Reporting

0x20: Status

To request the status report, send anything to Output Report 0x15. The Status Report will also be automatically sent when an Extension Controller is connected or disconnected.

This will request the status report (and turn off rumble):

```
(a2) 15 00
```

This report is sent either on request (in response to report 0x15), or in response to an expansion being plugged in or unplugged (or synced if wireless). If this report is received when not requested, the application 'MUST' send report 0x12 to change the data reporting mode, otherwise no further data reports will be received.

```
(a1) 20 BB BB LF 00 00 VV
```

BBBB is the core Buttons data. VV is the current battery level, L is the LED state, and F is a bitmask of flags indicating, whether the battery is flat, whether an expansion is currently connected, etc.

The Wii Remote can report its status, which includes the state of a few basic settings, the status of the Extension Controller (connected or disconnected), and the battery level.

VV is the current battery level, and LF is a bitmask of flags:

Bit	Mask	Meaning
0	0x01	Battery is nearly empty
1	0x02	An Extension Controller is connected
2	0x04	Speaker enabled
3	0x08	IR camera enabled
4	0x10	LED 1
5	0x20	LED 2
6	0x40	LED 3
7	0x80	LED 4

0x21: Read Memory Data

This report is sent when a read memory request is made. It returns 1 to 16 bytes of data at a time.

```
(a1) 21 BB BB SE AA AA DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
```

BBBB is the core Buttons data.

S (high nybble of SE) is the size in bytes, minus one, for the current data packet. This is 0xf (16 bytes) for all but the last packet, where it might be less if the requested number of bytes is not a multiple of 16.

E (low nybble of SE) is the error flag. Known error values are 0 for no error, 7 when attempting to read from a write-only register or an expansion that is not connected, and 8 when attempting to read from nonexistant memory addresses.

AA AA are the 2 least significant bytes of the absolute memory address of the first byte of data returned (the high byte of the offset is not returned, and neither is whether it is a register or memory that is being used. Thus, this must be known from the read request).

The DD bytes are the data, padded with zeroes to 16 bytes. If more than 16 bytes are requested, multiple packets will be received, with AA AA addresses increasing by 16 each time.

0x22: Acknowledge output report, return function result

This input report is sent to the host to report an error related to an output report, or the function result from that output report. It is sent when bit 1 of the first byte of any output report is set.

```
(a1) 22 BB BB RR EE
```

BBBB is the core Buttons data.

RR is the output report number that the Wii remote is acknowledging it received.

EE is the error code or function result. 00 = success. 03 = error. 04 = unknown (possibly returned by report 16H, 17H or 18H) 05 = unknown (possibly returned by report 12H). 08 = unknown (possibly returned bt report 16H).

Data Reporting

The Wii Remote has a number of different data reporting modes. Each of these modes combines certain Core data features with data from external peripherals, and sends it to the host through one of the report IDs, determined by the mode. The data format from the peripherals is determined by the peripherals themselves, all the Wii Remote controller does is pull bytes from them and send them out to the host. Due to this, certain feature combinations are not available, as there are not enough bytes for them in any of the output modes.

The Data Reporting Mode is set by sending a two-byte command to Report 0x12:

```
(a2) 12 TT MM
```

Bit 2 of TT specifies whether continuous reporting is desired. If bit 2 (0x04) is set, the Wii Remote will send reports whether there has been any change to the data or not. Otherwise, the Wii Remote will only send an output report when the data has changed.

MM specifies the Reporting Mode. Each Mode is specified by the Output Report ID that the data will be sent to. For example, this will set mode to 0x33:

```
(a2) 12 00 33
```

Data will then arrive through Input Report 0x33.

Upon powerup, the Data Reporting Mode defaults to 0x30. Following a connection or disconnection event on the Extension Port, data reporting is disabled and the Data Reporting Mode must be reset before new data can arrive.

Modes which include Accelerometer data also embed part of it in the unused Buttons bits. In all modes except for 0x3e/0x3f, the Buttons data includes the LSBs of the Accelerometer data. In mode 0x3e/0x3f, the interleaved Buttons data includes the Z-axis Accelerometer data.

0x30: Core Buttons

This mode returns data from the buttons in the Wii Remote:

```
(a1) 30 BB BB
```

BBBB is the core Buttons data.

0x31: Core Buttons and Accelerometer

This mode returns data from the buttons and the accelerometer in the Wii Remote:

```
(a1) 31 BB BB AA AA AA
```

BBBB is the core Buttons data. AA AA AA is the Accelerometer data.

0x32: Core Buttons with 8 Extension bytes

This mode returns data from the buttons in the Wii Remote, and data from an extension controller connected to it:

```
(a1) 32 BB BB EE EE EE EE EE EE EE EE
```

BBBB is the core Buttons data. The 8 EE bytes are from the Extension Controller currently connected to the Wii Remote.

0x33: Core Buttons and Accelerometer with 12 IR bytes

This mode returns data from the buttons, accelerometer, and IR Camera in the Wii Remote:

```
(a1) 33 BB BB AA AA AA II II II II II II II II II II II II
```

BBBB is the core Buttons data. AA AA AA is the Accelerometer data. The 12 II bytes are from the built-in IR Camera.

0x34: Core Buttons with 19 Extension bytes

This mode returns data from the buttons in the Wii Remote, and data from an extension controller connected to it:

```
(a1) 34 BB BB EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE
```

BBBB is the core Buttons data. The 19 EE bytes are from the Extension Controller currently connected to the Wii Remote.

0x35: Core Buttons and Accelerometer with 16 Extension Bytes

This mode returns data from the buttons and accelerometer in the Wii Remote, and data from an extension controller connected to it:

```
(a1) 35 BB BB AA AA AA EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE
```

BBBB is the core Buttons data. AA AA AA is the Accelerometer data. The 16 EE bytes are from the Extension Controller currently connected to the Wii Remote.

0x36: Core Buttons with 10 IR bytes and 9 Extension Bytes

This mode returns data from the buttons and IR camera in the Wii Remote, and data from an extension controller connected to it:

```
(a1) 36 BB BB II II II II II II II II II EE EE EE EE EE EE EE EE EE
```

BBBB is the core Buttons data. The 10 II bytes are from the built-in IR Camera, and the 9 EE bytes are from the Extension Controller currently connected to the Wii Remote.

0x37: Core Buttons and Accelerometer with 10 IR bytes and 6 Extension Bytes

This mode returns data from the buttons, accelerometer, and IR camera in the Wii Remote, and data from an extension controller connected to it:

```
(a1) 37 BB BB AA AA AA II II II II II II II II II II EE EE EE EE EE EE
```

BBBB is the core Buttons data. AA AA AA is the Accelerometer data. The 10 II bytes are from the built-in IR Camera, and the 6 EE bytes are from the Extension Controller currently connected to the Wii Remote.

0x3d: 21 Extension Bytes

This mode returns data from an extension controller connected to the Wii Remote. It is the only input report that does not include core buttons.

```
```

```
(a1) 3d EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE EE
```

The 21 EE bytes are from the Extension Controller currently connected to the Wii Remote.

0x3e / 0x3f: Interleaved Core Buttons and Accelerometer with 36 IR bytes

Both 0x3e and 0x3f are equivalent, and return data alternately through report IDs 0x3e and 0x3f. The data is interleaved, and is returned at half the speed of other modes (as two reports are needed for a single data unit). This mode returns data from the buttons, accelerometer, and IR camera in the Wii Remote:

```
(a1) 3e BB BB AA II II II II II II II II II II II II II II II II II
(a1) 3f BB BB AA II II II II II II II II II II II II II II II II II
```

BBBB is the core button data, as specified in the Buttons section. AA AA is the Accelerometer data, in a format specific to this mode described in the Interleaved Accelerometer Reporting section. The 36 II bytes are from the built-in IR Camera.

Memory and Registers

The Wii Remote includes a built-in EEPROM memory, part of which is accessible to the user to store that. This user part is used to store calibration constants, as well as the Mii Data. Additionally, many peripherals on the Wii Remote have registers which are accessible through a portion of the address space.

Both built-in memory and peripheral registers are accessed using the same reports, where a flag is used to select between the two.

Reading and Writing

To read data, commands are sent to Output Report 0x17:

```
(a2) 17 MM FF FF FF SS SS
```

FF FF FF is the offset, and SS SS is the size to read in bytes (both in big-endian format). Bit 2 (0x04) of MM selects the address space. Clearing this bit results in reading from EEPROM Memory, while setting it results in reading from the control registers. Setting bit 3 (0x08) also works to access registers, but setting both results in errors. As with all other reports, it also includes the Rumble flag, which must be set to the current rumble state to avoid affecting it.

Data read is returned through Input Report 0x21:

```
(a1) 21 BB BB SE FF FF DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
```

BB BB is the state of the buttons on the Wii Remote. During data reads, regular input reporting is temporarily suspended. Button data is available through the data input reports, but no other input data can be collected while the transfer lasts. FF FF is the offset expressed in absolute memory address of the Wii remote memory for the first byte of data returned (the high byte of the offset is not returned, and neither is which data memory is being used. Thus, this must be known from the read request). E (low nybble of SE) is the error flag. Known error values are 0 for no error, 7 when attempting to read from a write-only register, and 8 when attempting to read from nonexistant memory. S (high nybble of SE) is the size in bytes, minus one, for the current data packet. This is 0xf (16 bytes) for all but the last packet, where it might be less if the requested number of bytes is not a multiple of 16. The DD bytes are the data, padded with zeroes to 16 bytes. If more than 16 bytes are requested, multiple packets will be received, with FF FF offsets increasing by 16 each time.

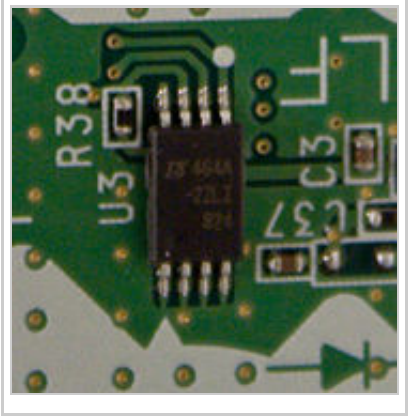
To write data, commands are sent to Output Report 0x16:

```
(a2) 16 MM FF FF FF SS DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
```

The meaning of the bytes is the same as during reads, except that size can be a maximum of 16 bytes (as there is only space for that much data), and the actual data to write follows (the DD bytes), padded out to 16 bytes.

Some kind of acknowledgement is received on Input Report 0x22. This has not been investigated yet.

EEPROM Memory



There is a 128kbit (= 16kB) EEPROM chip (Data Sheet (http://www.st.com/stonline/products/literature/ds/4578/m24128-bw.pdf) / Full EEPROM dump from a sample Wii Remote (http://www.sparkfun.com/tutorial/WiiRemote/NintendoWii-I2C-Data.zip)) in the Wii Remote. Part of its contents include code for the built-in microcontroller, and a generic section which can be freely read and written by the host. This section is 0x1700 bytes long, and part of this memory is used to store the Mii Data. It can be accessed by reading from/writing to addresses 0x0000-0x16FF in the Wii Remote's virtual memory space; in the actual EEPROM chip, the data is located at 0x0070-0x176F.

The firmware stored in the Wiimote has been disassembled.

The BCM2042 (http://www.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions/BCM2042) microcontroller built into the Wii Remote includes a large 108kb on-chip ROM section for storing firmware. If the EEPROM chip really contains code for the BCM2042 then this was probably done to make firmware updates possible, so there might be a way of accessing the other parts of the EEPROM via Bluetooth as well. From the BCM2042 Product Brief (http://www.broadcom.com/collateral/pb/2042-PB03-R.pdf) : "ROM-based design eliminates external flash memories; Flash option offered to support feature development".

On a virgin Wii Remote, acquired separately (not bundled with a Wii), that has never communicated with any device (except the PC used to dump the memory contents), most of the memory is blank (0x00). However, the first few bytes contain some information:

```
0000:  A1 AA 8B 99 AE 9E 78 30 A7 74 D3 A1 AA 8B 99 AE
0010:  9E 78 30 A7 74 D3 82 82 82 15 9C 9C 9E 38 40 3E
0020:  82 82 82 15 9C 9C 9E 38 40 3E 00 00 00 00 00 00
0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

This can be better visualized as two sequences, each one repeated twice:

```
0000:  A1 AA 8B 99 AE 9E 78 30 A7 74 D3
000B:  A1 AA 8B 99 AE 9E 78 30 A7 74 D3
0016:  82 82 82 15 9C 9C 9E 38 40 3E
0020:  82 82 82 15 9C 9C 9E 38 40 3E
```

It is not yet clear why these sentences are repeated; but since at least the second one is known to be calibration data, maybe one version contains the calibration data that is actually being used, while the other version is meant for backup purposes (for example a "Return to factory settings" option) in case there will be a way of recalibrating the Wii Remote with future Wii firmware updates.

The four bytes starting at 0x0016 and 0x0020 store the calibrated zero offsets for the accelerometer (high 8 bits of X,Y,Z in the first three bytes, low 2 bits packed in the fourth byte as --XXYYZZ). Apparently, the four bytes at 0x001A and 0x24 store the force of gravity on those axes. The function of other data bytes is not known, and most of them differ between Wii Remotes. Some or all of these bytes might not be used by the Wii. However, there has been a case of a Wii Remote where Extension functionality was lost following a battery change, and restoring these bytes (which had been previously overwritten) fixed the problem. The Extension controllers did not work with a PC either (which did not explicitly use these bytes), suggesting some of these might be used by the Wii Remote itself. This is unconfirmed, but it is advised that these never be overwritten, and recommended that they be backed up, just in case.

At 0x16D0, there is some more unknown data:

```
16D0:  00 00 00 FF 11 EE 00 00 33 CC 44 BB 00 00 66 99
16E0:  77 88 00 00 2B 01 E8 13 00 00 00 00 00 00 00 00
16F0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

In contrast to the data at 0x0000, this data seems to differ in only a few bytes between different Wii Remotes.

Known memory ranges are listed below. Note that the "user data" area is 0x0FA0 = 4000 bytes long, which seems to confirm the 4kB figure that has been mentioned (meaning 4000 bytes, that is, using the SI prefix meaning instead of the binary meaning).

Start	End	Length	Initial Value	Use
0x0000	0x0029	0x002A	See above	Calibration values / pre-set data
0x002A	0x0FC9	0x0FA0	Zeroed	User data / Unknown uses
0x0FCA	0x12B9	0x02f0	Zeroed	Mii Data block 1
0x12BA	0x15A9	0x02f0	Zeroed	Mii Data block 2
0x15AA	0x16CF	0x0126	Zeroed	Unknown / Unused

0x16D0	0x16FF	0x0030	See above	Unknown data

The top byte of the address is unused, which means memory is mirrored every 0x10000 bytes. Reading from unused addresses where the low 16 bits are >= 0x1700 will result in error returns.

Control Registers

The Wii Remote has several memory mapped register spaces corresponding to different peripherals in it. These include the Speaker, Extension Controllers, and the IR Camera.

> **Reminder**
> Remember to set bit 2 (0x04) on the first byte of the Output Report, otherwise you'll overwrite EEPROM memory!

The peripheral to access is selected by the first byte of the address, and the lower 16 bits specify the register to access within that peripheral. The lowest bit of the high byte is ignored, which means every peripheral is mirrored at its address + 0x10000. Known peripherals are listed below:

Start	End	Use
0xA20000	0xA20009	Speaker settings
0xA40000	0xA400FF	Extension Controller settings and data
0xA60000	0xA600FF	Wii Motion Plus settings and data
0xB00000	0xB00033	IR Camera settings

Most of these are also mirrored across the high bits of the individual peripheral. For example, the second byte of the address is ignored in the Extension controller address, which means any address of the form 0xA4xx00 will work (as will 0xA5xx00).

Input Features

The Wii Remote has two input features that are controlled directly by the Broadcom chip: a Three-Axis Accelerometer and 11 Buttons. Additionally, it has an IR Camera with an object tracking processor, and an expansion port that allows for external input features such as those contained in the Nunchuk and the Classic Controller (see Extension Controllers).

Buttons

The Wii Remote has 11 buttons on its front face, and one trigger-style button on the back. Of these, the Power button is special and is treated differently by the Wii Remote. All the other buttons are independantly accessible through a two-byte bitmask which is transmitted first in most Input Reports. A button will report a 1-bit if pressed, or a 0-bit otherwise. By default, these are sent only when the state of any button changes, in Data Reporting Mode 0x30. However, the Wii Remote may be configured to report the state of the buttons continuously; see Data Reporting.

Core Buttons

The Wii Remote has 11 buttons that are used as regular input devices: A, B (trigger), a 4-directional D-Pad, +, -, Home, 1, and 2. These are reported as bits in a two-byte bitmask. These are the assignments, in big-endian order:

Bit	Mask	First Byte	Second Byte
0	0x01	D-Pad Left	Two
1	0x02	D-Pad Right	One
2	0x04	D-Pad Down	B
3	0x08	D-Pad Up	A
4	0x10	Plus	Minus
5	0x20	Other uses	Other uses
6	0x40	Other uses	Other uses
7	0x80	Unknown	Home

Power Button

When the Wii Remote is turned off, pressing the Power button will attempt to wake up the Wii that is synchronized to it. The mechanism for this is unknown, and it is handled entirely within the Wii's bluetooth module. When the Wii Remote is turned on and connected to a host, pressing and holding the Power button for a few seconds will turn the Wii Remote off and request disconnection from the host. The disconnection reason included with the Baseband (ACL) disconnection request indicates that the power button was pressed: REMOTE DEVICE TERMINATED CONNECTION DUE TO POWER OFF (0x15). Another possible value is REMOTE DEVICE TERMINATED CONNECTION DUE TO LOW RESOURCES (0x14), which indicates that the Wii Remote performed a controlled shut down due to a low battery condition.

Sync Button

The sync button is hidden under the battery cover. When the Sync button is pressed, the Wii remote will disconnect from whatever it is currently connected to, make itself discoverable, and accept pairing or connection requests for exactly 20 seconds (regardless of how long the button is held down for).

The "syncing" of a Wii Remote involves standard Bluetooth pairing. When the Sync button is pressed on the remote, it will accept pairing requests. The required PIN is the hosts's Bluetooth address, backwards (last byte first), in binary (6 bytes). Most current Bluetooth implementations don't deal with this correctly, as they usually consider the PIN to be a regular null-terminated ASCII string (no 00 bytes, etc) and most Bluetooth addresses will contain null bytes. Any further steps that need to be taken after the Wii Remote is paired have not been reverse engineered yet.

Once the Wii Remote is synced, when a button is pressed, it will actively seek out its paired host and try to connect to it, instead of the other way around. Establishing a connection can be done on PSM 0x11 for writing and PSM 0x13 for reading using the Bluetooth L2CAP protocol.

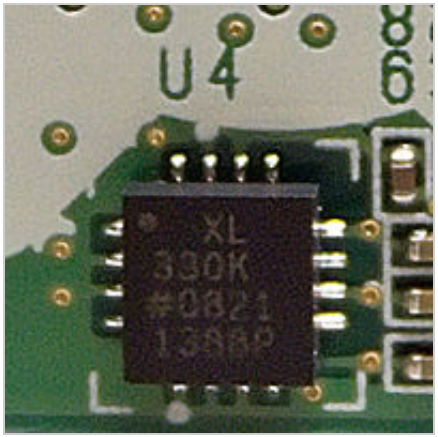
Button Hardware

The physical hardware of the buttons varies: there are membrane switches and microswitch click buttons. There has been some success soldering wires to the membrane switch contacts and actuating the switch through an external switch. The following table describes the physical hardware for each input.

Function	Switch type	Circuit board surface
A	membrane (http://en.wikipedia.org/wiki/Membrane_switch)	top, SW9
B	membrane	bottom, SW8
-	microswitch (http://en.wikipedia.org/wiki/Microswitch)	top, SW10
Home	microswitch	top, SW11
+	microswitch	top, SW5
1	membrane	top, SW7
2	membrane	top, SW6
Up	membrane	top, SW4
Down	membrane	top, SW3
Left	membrane	top, SW1
Right	membrane	top, SW2
Sync		bottom, SW12
Power		top, SW13

Accelerometer

The Wii Remote includes a three-axis linear accelerometer located on the top suface of the circuit board, slightly left of the large A button. The integrated circuit is the ADXL330 (http://www.seekic.com/icdata/%20ADXL330.html) (data sheet (http://www.analog.com/UploadedFiles/Data_Sheets/ADXL330.pdf)), manufactured by Analog Devices. This device is physically rated to measure accelerations over a range of at least +/- 3g with 10% sensitivity.



ADXL330 in a Wii remote

Since the accelerometer actually measures the force exerted by a set of small proof masses inside of it with respect to its enclosure, the accelerometer measures linear acceleration in a free fall frame of reference. If the Wii Remote is in free fall, it will report zero acceleration. At rest, it will report an upward acceleration (+Z, when horizontal) equal to the acceleration due to gravity, g (approximately 9.8 m/s²) but in the opposite direction. This fact can be used to derive tilt from the acceleration outputs when the Wii Remote is reasonably still.

Normal Accelerometer Reporting

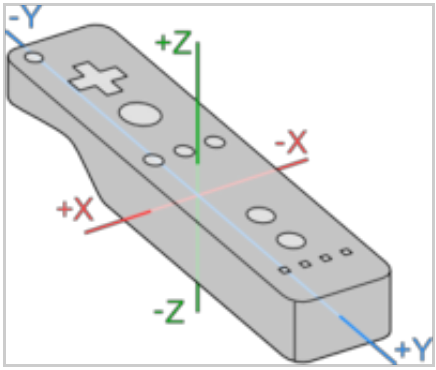
In all Data Reporting Modes which include Accelerometer data except for mode 0x3e/0x3f, the accelerometer data is reported as three consecutive bytes:

```
(a1) RR BB BB XX YY ZZ [...]
```

XX, YY, and ZZ are unsigned bytes representing the acceleration in each of the three axis, where zero acceleration is approximately 0x80. The coordinate system is shown in the diagram above (note that this is different from the coordinate system used by GlovePIE). Additionally, the BB BB Buttons bytes also include the LSBs of the acceleration values in the unused bits, according to the following table:

		Bit						
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0		**X**<1:0>						
1		**Z**<1>	**Y**<1>					

Note that X has 10 bits of precision, while Y and Z only have 9. For consistency, they are assumed all to have a 10-bit range and the LSB is always set to zero for Y and Z.



Coordinate system used by Wii Remote

Interleaved Accelerometer Reporting

In Data Reporting Mode 0x3e/0x3f, the accelerometer data is spread over two reports:

```
(a1) 3e BB BB XX [...]
(a1) 3f BB BB YY [...]
```

In this mode, the LSBs are not available. Instead, X and Y acceleration is reported as a single byte, and the Z value is encoded in the unused bits of the BB BB Buttons data as follows:

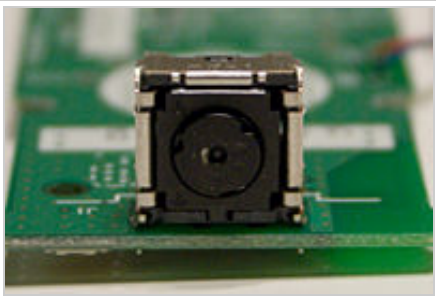
		Bit							
Report ID	**Byte**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x3e	0		**Z**<5:4>						
0x3e	1		**Z**<7:6>						
0x3f	0		**Z**<1:0>						
0x3f	1		**Z**<3:2>						

IR Camera

The Wii Remote includes a 128x96 monochrome camera with built-in image processing. The camera looks through an infrared pass filter in the remote's plastic casing. The camera's built-in image processing is capable of tracking up to 4 moving objects, and these data are the only data available to the host. Raw pixel data is not available to the host, so the camera cannot be used to take a conventional picture. The built-in processor uses 8x subpixel analysis to provide 1024x768 resolution for the tracked points. The Sensor Bar that comes with the Wii includes two IR LED clusters at each end, which are tracked by the Wii Remote to provide pointing information. The distance

between the centers of the LED clusters is 20 cm (as measured on one unit).

The IR Camera is enabled by setting bit 2 on output reports 0x13 and 0x1a:



Wii remote camera

```
(a2) 13 04
(a2) 1a 04
```

The first enables a 24MHz pixel clock on pin 7 of the camera. The second pulls pin 4 low - probably an active-low enable.

Mechanical Characteristics

The camera component is mounted on the bottom surface of the circuit board. The camera module itself is mounted in a socket perpendicular to the circuit board; to remove just the camera module, no desoldering is required. The process is as follows:

First, orient the camera so that you are looking into the lens with the PCB horizontal and below the lens.. There are four metal clips, two on each of the vertical sides of the socket. Use something tiny to slide between each metal clip and the camera module: maybe wire wrap wire? Then look at the back of the camera module, opposite the lens. There is a small rectangular hole in the middle of each vertical side of the socket. Use a pin or something to pry/press the camera module out.

Once the camera module is free of its socket, it may be further disassembled by gently prising up the tiny PCB with gold contacts; this is gently glued to the module's structure, but will come loose without damage. At this point you have three pieces: the camera socket, still attached to the Wiimote PCB, the camera module housing, complete with lens and dichroic filter (of unknown optical properties), and a tiny PCB with the camera chip and eight gold contacts on the bottom.

Optical Characteristics

The IR camera has an effective field of view is about 33 degrees horizontally and 23 degrees vertically (as measured on one unit). With the IR-pass filter intact, 940nm sources are detected with approximately twice the intensity of equivalent 850nm sources, but are not resolved as well at close distances. If the filter is removed, it can

track any bright object. However, the IR filter referred to here is not only the dark plastic window of the wiimote but also a teensy slab of dichroic-coated glass inside the camera module. One may operate the wiimote having installed neither, one or the other, or both filters.

Initialization

> **Reminder**
> Remember to set bit 2 (0x04) on the first byte of the Output Reports to write to registers!

The following procedure should be followed to turn on the IR Camera:

1. Enable IR Camera (Send 0x04 to Output Report 0x13)
2. Enable IR Camera 2 (Send 0x04 to Output Report 0x1a)
3. Write 0x08 to register 0xb00030
4. Write Sensitivity Block 1 to registers at 0xb00000
5. Write Sensitivity Block 2 to registers at 0xb0001a
6. Write Mode Number to register 0xb00033
7. Write 0x08 to register 0xb00030 (again)

After these steps, the Wii Remote will be in one of 3 states: IR camera on but not taking data, IR camera on and taking data and half sensitivity, IR camera on and taking data at full sensitivity. Which state you end up in appears to be pretty much random. Repeat the steps until you're in the desired state. To avoid the random state put a delay of at least 50ms between every single byte transmission.

The Wii preforms these steps slightly different, differences in bold:

1. Enable IR Pixel Clock (send **0x06** to Output Report 0x13)
2. Enable IR Logic (send **0x06** to Output Report 0x1A)
3. Write **0x01** to register 0xb00030
4. Write Sensitivity Block 1 to registers at 0xb00000
5. Write Sensitivity Block 2 to registers at 0xb0001a
6. Write Mode Number to register 0xb00033
7. Write 0x08 to register 0xb00030 (again)

Adding bit 0x02 to reports 0x13 and 0x1a is a request for acknowledgement (if set, wiimote will respond with report 0x22).

Sensitivity Settings

Sensitivity is controlled by two configuration blocks, 9 bytes and 2 bytes long. The following settings are known to work:

Block 1	Block 2	Notes
`00 00 00 00 00 00 90 00 C0`	`40 00`	Suggested by Marcan
`00 00 00 00 00 00 FF 00 0C`	`00 00`	Suggested by Kestrel (max sensitivity)

00 00 00 00 00 00 90 00 41	40 00	Suggested by inio (high sensitivity)
02 00 00 71 01 00 64 00 fe	fd 05	Wii level 1
02 00 00 71 01 00 96 00 b4	b3 04	Wii level 2
02 00 00 71 01 00 aa 00 64	63 03	Wii level 3 (Suggested by Cliff)
02 00 00 71 01 00 c8 00 36	35 03	Wii level 4
07 00 00 71 01 00 72 00 20	1f 03	Wii level 5

The last byte of Block 1 determines the intensity sensitivity, with increasing values reducing the sensitivity. Both bytes of Block 2 must be zero for the full sensitivity range to be available. Setting the sensitivity as high as possible, without unwanted light being tracked, is recommended to achieve the highest subpixel resolution. As the sensitivity is reduced, the subpixel resolution also reduces, approaching the true sensor resolution of 128x96.

Data Formats

The IR Camera can return different sets of data describing the objects it is tracking. When the IR camera identifies an object, it assigns it to the first available object slot. If an object moves out of view, its slot is marked as empty (returns 0xFF data), but other objects retain their slots. For example, if the camera is tracking two objects and the first moves out of view, the data returned will be [empty, second object, empty, empty]. With more than four objects visible, the camera is prone to rapidly switching between some of them. This could allow perception of more than four objects, at a reduced response speed and reliability.

Mode	Mode Number
Basic	1
Extended	3
Full	5

The data format MUST match the number of bytes available in the Reporting Mode selected. Even choosing a mode with space for more bytes than necessary will not work, it has to be an exact match.

Basic Mode

In Basic Mode, the IR Camera returns 10 bytes of data corresponding to the X and Y locations of each of the four dots. Each location is encoded in 10 bits and has a range of 0-1023 for the X dimension, and 0-767 for the Y dimension. Each pair of dots is packed into 5 bytes, and two of these are transmitted for a total of 4 dots and 10 bytes.

This is the data format for a pair of objects:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**X1**<7:0>							
1	**Y1**<7:0>							

2	**Y1**<9:8>	**X1**<9:8>	**Y2**<9:8>	**X2**<9:8>
3	**X2**<7:0>			
4	**Y2**<7:0>			

Extended Mode

In Extended Mode, the IR Camera returns the same data as it does in Basic Mode, plus a rough size value for each object. The data is returned as 12 bytes, three bytes per object. Size has a range of 0-15.

This is the data format for each object:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**X**<7:0>							
1	**Y**<7:0>							
2	**Y**<9:8>		**X**<9:8>		**S**<3:0>			

Full Mode

In Full Mode, the IR Camera returns even more data, 9 bytes per object for a total of 36 bytes for all four. The data is split up between two input reports of 18 bytes each (see Data Reporting Mode 0x3e/0x3f). The first three bytes of each object are the same as the extended mode, and are followed by the bounding box of the pixels included in the blob along with a deeper intensity value. The data format of each object is:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**X**<7:0>							
1	**Y**<7:0>							
2	**Y**<9:8>		**X**<9:8>		**S**<3:0>			
3	0	**X min**<6:0>						
4	0	**Y min**<6:0>						
5	0	**X max**<6:0>						
6	0	**Y max**<6:0>						
7	0							
8	**Intensity**<7:0>							

Feedback Features

The Wii Remote sports three feedback features: Player LEDs, Rumble, and the Speaker.

Player LEDs

There are four blue LEDs on the front face of the Wii Remote. During discovery and before initialization, these LEDs blink at a fixed rate. The number of blinking LEDs is proportional to the battery voltage, indicating battery charge (all four are lit for newly charged batteries, and only the first is lit if the batteries are low and should be replaced).

Wii remote player LEDs

During gameplay with the Wii, one LED is lit to indicate the player number assigned to the Wii Remote. However, the LEDs are independently controllable by the host, and can be set to display any pattern. They can also be modulated at a moderately high speed, enabling some brightness control at the cost of a lot of Bluetooth bandwidth. Sigma-delta modulation works reasonably well for this.

The LEDs can be controlled by sending a report with ID 0x11:

```
(a2) 11 LL
```

The high nybble of LL controls the four LEDs. Bit 4 of LL controls the first LED, and bit 7 controls the last:

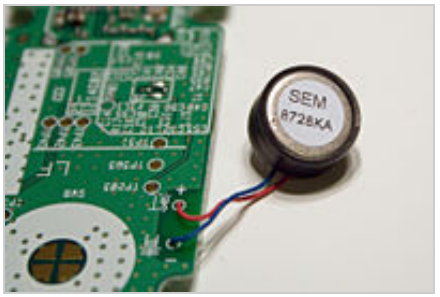
Bit	Mask	LEDs
4	0x10	
5	0x20	
6	0x40	
7	0x80	

Turning off all LEDs for an extended period of time is discouraged, as it might lead the user to believe the Wii Remote is turned off and disconnected, when in fact it is active.

The LEDs are surface mount parts, driven at 2.66 VDC.

Rumble

The Wii Remote includes a rumble feature, which is implemented as a small motor attached to an off-center weight. It will cause the controller to vibrate when activated.



Wii remote rumble motor

The rumble motor can be turned on or off through any of the Output Reports. Setting the LSB (bit 0) of the first byte of any output report will activate the rumble motor, and unsetting it will deactivate it. For example, the following report will turn the rumble motor on:

```
(a2) 11 01
```

However, this will also have the side-effect of turning off all LEDs. Since there is no output report that only affects the rumble motor, and all of them do affect it, an implementation might need to store both the rumble and LED values locally (for example), and use the same Output Report for both. Another possibility would be using the status request report (0x15). The rumble bit needs to be set properly with every single report sent, to avoid inadvertently turning the rumble motor off.

Different photos of the rumble motor hardware show different markings. One example is *SEM 8728DA*. The Wii Remote drives it at 3.3 VDC and it draws 35 mA. It would be reasonable to think that the rumble motor could be removed and the motor replaced with another device with equal voltage and equal or less current draw.

Speaker

The Wii Remote has a small low-quality 21mm piezo-electric speaker, used for short sound effects during gameplay. The sound is streamed directly from the host, and the speaker has some adjustable parameters.

The speaker is controlled by using three output reports, together with a section of the register address space of the Wii Remote.

Report 0x14 is used to enable or disable the speaker. Setting bit 2 will enable the speaker, and clearing it will disable it. For example, to enable the speaker, send:

```
(a2) 14 04
```

Report 0x19 is used to mute or unmute the speaker, and works identically to report 0x14. 0x04 will mute the speaker, and 0x00 will unmute it.

Report 0x18 is used to send speaker data. 1-20 bytes may be sent at once:

```
(a2) 18 LL DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
```

LL specifies the data length, shifted left by three bits. The DD bytes are the speaker data. To fullfill the report length requirements, the data must be padded if it is less than 20 bytes long. Sound data must be sent at the proper rate.

Initialization Sequence

> **Reminder**
> Remember to set bit 2 (0x04) on the first byte of the Output Reports to write to registers!

The following sequence will initialize the speaker:

1. Enable speaker (Send 0x04 to Output Report 0x14)
2. Mute speaker (Send 0x04 to Output Report 0x19)
3. Write 0x01 to register 0xa20009

4. Write 0x08 to register 0xa20001
5. Write 7-byte configuration to registers 0xa20001-0xa20008
6. Write 0x01 to register 0xa20008
7. Unmute speaker (Send 0x00 to Output Report 0x19)

Speaker Configuration

7 bytes control the speaker settings, including volume. The full purpose of these bytes is not known, but the following values seem to produce some sound:

```
00 FF RR RR VV 00 00
```

RR RR specify the sample rate (little-endian format), using the following formulae:

pcm_sample_rate = 12000000 / rate_value adpcm_sample_rate = 6000000 / rate_value

The standard value is 0x7d0, for 3000Hz 4-bit PCM

FF configures the data format. Setting it to 0x40 configures the speaker to use signed 8-bit PCM, while setting it to 0x00 configures it to use 4-bit Yamaha ADPCM. VV specifies the volume, which has a range of 0x00-0xFF for 8-bit mode, and 0x00-0x40 for 4-bit mode.

This configuration can be used to play 4-bit ADPCM sound at 3000Hz:

```
00 00 D0 07 40 00 00
```

This configuration can be used to play 8-bit PCM sound at 1500Hz sample rate:

```
00 40 40 1f 40 00 00
```

It looks like the best compromise between sample rate and slow bluetooth chipsets / drivers is playing 8-bit PCM at 2000Hz, so a new 20-byte chunk of audio data is sent every 10 milliseconds.

```
00 40 70 17 60 00 00
```

Sound Data Format

The Wii Remote can use multiple sound formats at multiple sampling rates. PC drivers currently seem unable to keep up with the higher rates.

The 4-bit ADPCM is Yamaha ADPCM (for example, as implemented in ffmpeg).

8-bit signed PCM mode works, but when in 8-bit mode the sampling frequency must be made so low that the audio quality is pretty bad.

Extension Controllers

Wiimote/Extension Controllers

Notes

See Also

- Disassembled Firmware shows raw firmware dumps from several wiimotes.
- Extension Controllers explains the protocol used by the wiimote extensions.
- Motion analysis gives hints how to implement accelerometer parsers.
- Pointing shows how to create a pointing device with IR data.
- Library lists several wiimote library implementations.
- Mii Data shows raw wiimote data dumps.

Acknowledgements

Some of the information here is based on the documentation at Wiili