Nintendo Gamecube Controller Protocol

Last updated 8th March 2004 (first version was way back on 11th December 2002 :)

This is a reasonably technical document. If you aren't technically inclined, and you just want an easy way to connect your gamecube controllers to a PC, you might be interested to know that a ready made adaptor already exists. It's called the Skillz Cube Connection USB and is only sold by Lik-Sang as far as I'm aware. It only works with the original Nintendo Gamecube controller (it is not compatible with the Wavebird or any 3rd party controllers).

However, if you are interested in homebrew hardware, or just like dismantling things, read on.

Introduction

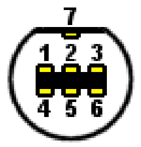
The controller connects to the Gamecube through a proprietary 6-pin connector, with screened cable. The official Nintendo controller only seems to wire 5/6 of these pins, and of those only one seems to be used for data transfer between the console and the controller. This document includes a pin out of the controller port, deduced from an examination of the controller and console with multimeter and oscilloscope, and from some experimentation. Therefore, I make no guarantee that any of this information is accurate, and you use it at your own risk. It's my *best guess* at how this thing works :)

If anyone has useful information to add to this page, please drop me a line (contact details are on index page).

James, 8th March 2004.

Connector Pinout

This is a view of the controller socket on the front of the console, looking into the socket. The numbering scheme is my own:



Pin	Colour	J1	Function
1	Yellow	2	5V power supply (used by rumble motor).
2	Red	3	DATA line: bi-directional data to/from console, pull-up to 3.43V
3	Green	4	Ground.
4	White	5	Ground (Skillz interface has pins 3+4 wired as common ground).
5	-	-	Unknown: not connected by official controller, or Skillz interface.
6	Blue	1	3.43V logic supply.
7	Black	6	Cable shielding / ground. Usually common ground with pin 3.

In the table above, the pin number on the left corresponds to the diagram of the controller socket. The colour code is that of the cable from the official Nintendo Controller (different models might vary), noting that one pin is not used in this case. The third column marked J1 refers to the pinout of the connector inside the controller, which you will only be able to get to if you have the appropriate security screwdriver bit (or improvise your own handmade tool). You can buy a suitable screwdriver from Lik-Sang also. The function column on the right is my best guess at what each pin is for.

Which pins are needed for a homebrew interface?

My prototype interface wires pins 3 and 4 together as common ground, uses a 7805 voltage regulator to provide a 5V supply to pin 1, and uses a variable voltage regulator to provide a 3.43V supply to pin 6. The only other connection that I make is to the data line on pin 2, for which I use a 1K pull-up resistor to the 3.43V rail. I notice that the Skillz adaptor uses a 3.3V regulator, and my initial design also used 3.3V. I suspect that the accuracy of this rail isn't very important. I choose to use 3.43V currently, only because this is what I measured on a PAL Gamecube.

Caution: this should go without saying; but once you have connected a 3.3V and 5V rail and pull-up resistor to your cable, it would likely cause some damage if you were to then connect the cable to a Gamecube. I only mention this because it would be easy to have an accident if you modified a controller extension cable as I did - don't forget to unplug it from your cube first.

Power Supply and Rumble Motor

There are two power rails on the connector, a 3.43V supply that is probably used for the logic, and a 5V supply that appears to be used to power the rumble motor (and perhaps logic also). The ground (3) and shield (7) are connected together.

The 5V power used by the rumble motor is always on, and the motor is controlled by a command sent to the controller. i.e. the controller contains a power transistor to switch the motor on/off, rather than the console doing this. The Yellow 5V power line goes directly to the +ve terminal of the rumble motor, and it looks like the -ve terminal of the motor is attached to a transistor.

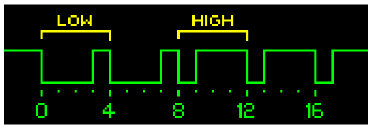
I've not measured the current drawn by the controller yet.

Serial Data Interface

The controller uses one bi-directional data line (Pin 2 - Red) to communicate with the console. This is an active high 3.43V logic signal, using a pull-up resistor to hold the line high, and pulling it low with an open-collector transistor when a low needs to be transmitted. Communication is initiated by the console sending a 24-bit string to the controller, after which the controller responds with 64-bits of button state and joystick data.

Although I first thought that the controller had an internal pull-up resistor (measured 745 ohms), in practice I had to use an external 1K pull-up resistor between the 3.43V rail and the the data line in my prototype interface.

The transfer speed is *rather fast* at around 4us per bit. As with the N64 controller, a low bit is signalled by a 3us low followed by 1us high, and a high bit is signalled by 1us low followed by 3us high. Yes, it's just like the N64 controller!



When the gamecube or the controller sends a string of bits, it terminates it with a single (high) stop bit. Therefore, in order to send the string 00000000, the gamecube would in fact send 000000001.

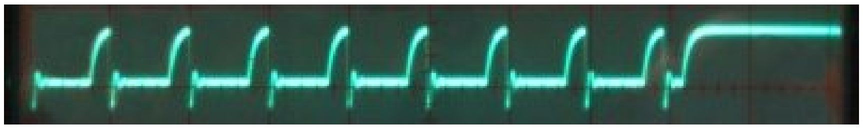
Timing Measurements

Initially (in my Dec.2003 document), I had thought that the timing was around 5us per bit, but I now believe that was wrong, and that the timebase on the 'scope was inaccurate. Philipp Kastner sent me a plot from a storage 'scope that showed 4us per bit, and I then went back and tried to measure the timings again, using the parallel port.

Using the parallel port, I timed the interval between the first high-to-low transition at the start of a command, and the final low-to-high transition at the end of the reply from the pad. The sample rate of the parallel port was around 1us per bit, leading to a possible timing error of around plus or minus 2us. An average of 10 successive measurements gave around 348us total time. Assuming a total of 24+64 = 88 bits, that equates to 3.95us per bit. This assumes no significant delay between the command and response from the pad. These timings were made using QueryPerformanceCounter, under Windows 2000, on a P4 2.8GHz, i875P chipset.

Probing for the Controller

With no controller attached, the gamecube probes for a controller by sending the sequence 000000001 about every 12ms. The oscilloscope trace below shows a typical probe sequence, with the 'scope triggered on the negative edge. When you connect a controller it will respond to this sequence, so you know that it is attached. More work is needed to examine the initial conversation between the Gamecube and controller to see if there is any useful information (e.g. about what type of controller is attached?)



Polling the Controller for Joystick/Button Data

With an official controller attached, there is a typical interval of about 6ms between successive updates. In fact, I believe that the update rate is controlled by the game, perhaps divided from the video frame rate. Each update lasts around 348us. The sequence starts with a 24-bit command from the console:

    0100 0000 0000 0011 0000 0010

After the 24-bit command word, the controller responds with a string of bits that contain the state of all the buttons along with joystick position data. The sequence of the returned data is as follows. Note that the buttons are listed in transmission order, from left to right (i.e. the left most bit is transmitted first).

Byte 0	0	0	0	Start	Y	X	B	A
Byte 1	1	L	R	Z	D-Up	D-Down	D-Right	D-Left
Byte 2	Joystick X Value (8 bit)							

Byte 3	Joystick Y Value (8 bit)
Byte 4	C-Stick X Value (8 bit)
Byte 5	C-Stick Y Value (8 bit)
Byte 6	Left Button Value (8 bit) - may be 4-bit mode also?
Byte 7	Right Button Value (8 bit) - may be 4-bit mode also?

As listed above, the L/R buttons are the end-stops on the L/R shoulder buttons. Note that between the A and L buttons, there is a bit that always appears to be high. Also, the three leading bits do not seem to be affected by the buttons (so far I have seen the sequence 000 and 001 appear here).

Making it rumble

The last bit of the command is the 'rumble' control. Setting this bit to one enables the rumble motor, and clearing it disables the motor. No initialisation sequence seems to be needed. As soon as you connect the controller, you can send the 24-bit command sequence and the pad will respond with data, and can be made to rumble.

Other observations

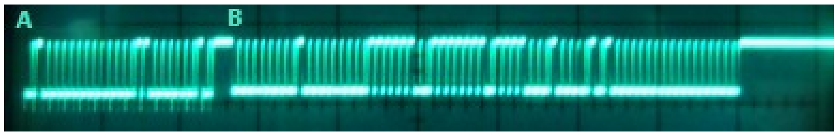
Given that there are 24-bits in the command word, it seems likely that there will be a series of different commands to reset the controller, or to perhaps query what kind of hardware is attached to the console. Further experimentation is needed to identify other commands.

Previously, it seemed that there was a delay of about 15us before the pad responded to a command from the console. However, in recent experiments (on a different controller - perhaps it varies between controller versions) this delay seems to be gone. I suspect that this might vary between controllers, or might be related to how frequently the controller is polled. It was clearly there in my first experiments though, as can be seen from the 'scope photos later in this document.

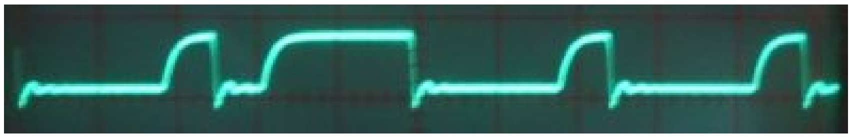
When examining the output of the Skillz Cube Connection, it looked as though only 4-bit analogue data is returned for the Left/Right shoulder buttons. I need to go back and verify this, but it seems that the pad might support different modes with 4-bit and 8-bit resolution. Having said that, I'm not sure it matters, who wants 4-bit data when you can have 8-bit?

What does it look like on an oscilloscope?

For those that don't have access to an oscilloscope or logic analyser, here are some blurry photographs badly taken from a tired old 'scope. In this figure, point A is the start of the 24-bit command word sent by the console, and point B marks the start of the 64-bit response from the controller. The quality of the image is quite poor, but it's actually possible to see the individual data bits. Note that a delay seems to be evident in this 'scope image between the command word and the start of the reply. In more recent experiments, I haven't seen this delay.



Finally, here is a close-up view of the individual data bits when transmitting binary 0100:



Homebrew Interfacing

Recently, I built a simple homebrew interface to allow me to experiment further with the controller. Using this interface, it was possible to reliably read all the button, joystick, c-stick and left/right shoulder button values from the original (official Nintendo) wired controller (DOL-003 it says on the bottom of my controller). When I tried it with a third part controller (MadCatz MicroCon) it didn't work, but I think it's probably just a bit of tweaking of the electronics and timing (which are, to be frank, rather poorly implemented at the moment).

Anyway, it's a start; it can talk to the official controller, and can even make it rumble. And you know what the best part is? Yes, you can download the source code here.

Prototype Software

First things first, this was developed on Windows 2000, and has currently only been tested on a P4 2.8GHz with i875P chipset. Hopefully, that isn't the minimum specification, but I wouldn't be surprised if the timing messes up on a slower computer. If it does, let me know, and we will see if it can be fixed.

I'm releasing this so that people can experiment with it, assuming some basic knowledge of electronics and software. This stuff isn't yet ready for any practical use; i.e. there are no proper drivers yet.

You will need a few things before you can use this software:

1. Some home made hardware (circuit diagram to follow shortly, but there is a description of pin connections in the source code for the impatient or the hardcore, which should be just enough to be able to build it).
2. The giveio device driver (download it).
3. Nerves of steel / willingness to potentially destroy your PC and controller :)

The program works by using direct port I/O on the parallel port. That isn't normally allowed from user mode on Windows NT/2000/XP, so I downloaded and installed a driver called **giveio** which you can easily find with the help of google (and I will add a link here soon hopefully). When this is installed (you will need administrator rights to do this), it basically breaks the protection mechanism so that your program can do direct port I/O.

Once you have installed the **giveio** service, the program will start and enable the giveio service automatically as required. This means that you don't need to set the giveio service to automatically start with windows (and I don't recommend doing this either, from a safety point of view).

I've not tested the program on Windows 95/98 (does anyone still use it?), and in fact I would be very surprised if it worked at all on that OS. If you know different, let me know.

Finally, here is the source code:

gcpad1.cpp

It compiles with MS Visual Studio .NET as a console project, and probably will work with Visual C++ 6.

It uses some inline assembler, and the syntax might be MS specific, but should be easily alterable to work with other compilers I think.

What next?

The first thing to do is to refine the hardware design and then to test it on more PCs. I think a reasonable minimum specification to aim for would be P3 1GHz. The software needs improving, with a kernel mode device driver to talk to the hardware, and a DirectInput driver to allow the controller to be used as a normal joystick device. Support for more than one controller would be nice. With the current shift register, there are two inputs, so it should be fairly easy to support a second controller.

I've been discussing with Philipp Kastner the possibility of building an interface around a PIC microcontroller. This would allow a serial or even USB interface to be developed. One obstacle is that the PICs with USB support are only available in UV erasable versions (no flashable version) which makes them a pain to develop for. Unfortunately, not everyone has access to a PIC programmer of course. However, with a PIC, it should be possible to support several controllers with a single interface. What would be really, really nice is a front panel drive bay with four Gamecube controller ports..

Now, if only there were some decent games on the PC, apart from boring old 3rd person shooters.

Credits

Huge thanks to Philipp Kastner for inspiring me to work on this again, I'd abandoned it really (I blame work, and of course my friends Zelda, Link and Mario, for occupying too many hours).

The shift register hardware design was inspired by the N64 Controller Interface project first described by Stephan Hans, Simon Nield, F.P.Earle et.al. nice work!

The GC Linux project is definitely worth a look, some detail about the controller commands is emerging in their documentation (search for YAGCD).

All the people who wrote to me over the last year or so, I can't remember all the names, but I'll credit you all when I find the old e-mails!

Thanks to Sara, for putting up with the wires, flashing lights and tools strewn everywhere :)