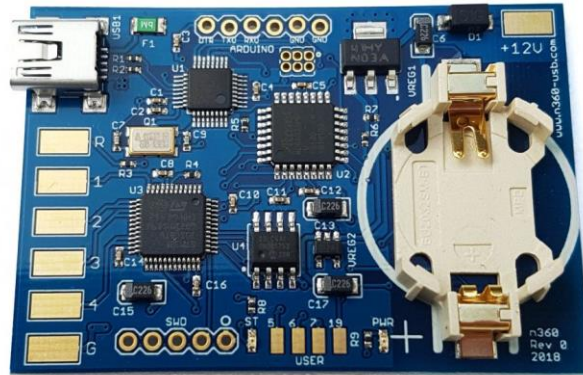


Summary

This document was created by me (Ryze119) during development of my n360 PCB. The n360 is a circuit board which adds Xbox 360 Wireless Controller support to the Nintendo 64 Console.

See <http://n360-usb.com/>



To achieve this goal, it was required to be able to fully emulate a genuine Nintendo 64 controller including the 'Rumble Pak' and 'Controller Pak' peripherals. My efforts with my logic analyser and understanding of the controller protocol is documented below.

With this knowledge, I was able to emulate 4 controllers simultaneously with Rumble/Controller Pak support and full compatibility with every game I tested (which was a lot)

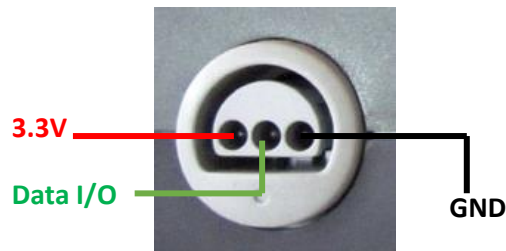
Contents

1	N64 Controller Protocol	2
1.1	Controller Pinout.....	2
1.2	Data Bus Protocol.....	3
1.3	N64 Console Commands	4
1.3.1	Controller Identify Request (0x00).....	4
1.3.2	Controller State Request (0x01).....	5
1.3.3	Read from Peripheral Bus (0x02)	6
1.3.4	Write to Peripheral bus (0x03).....	7
1.3.5	Reset Controller (0xFF).....	8
1.4	Address Format and CRC.....	8
1.5	Data CRC.....	9
1.6	Example Controller Initialisation.....	11

1 N64 Controller Protocol

The N64 console communicates with a N64 Controller with a bidirectional open collector single wire data bus. The N64 console acts as a master and polls each controller with commands. The controller responds to these commands accordingly to facilitate peripheral identification, button/analog stick positions, mem pack read/writes and rumblepak toggling.

1.1 Controller Pinout



View looking into console

3.3V supplies a small amount of power to the N64 Controller.

The data pin is a bidirectional data bus which transmits and receives data from the N64 Controller.

GND is the logic reference and the power return.

1.2 Data Bus Protocol

The data bus has an idle high signal. The data bits ('0's and '1's) begin on the falling edge of the data signal.

A logic level '0' is indicated by the data bus going low for 3us, then going high for 1us.

A logic level '1' is indicated by the data bus going low for 1us, then going high for 3us.

In both instances the complete bit takes 4us. The following bit proceeds instantly. The end of a data stream is indicated by a stop bit. This stop bit varies slightly depending on whether the console or the controller is talking.

The table below illustrates the different bits than can be seen on the data bus. The shaded areas indicate the idle bus state.

	-	1us	2us	3us	4us	-
Logic '0'						
Logic '1'						
Controller Stop Bit						
Console Stop Bit						

The bus is what is known as an 'open-collector' configuration. In short, what this means is that the bus is never 'driven' high by either the controller or the console. Each device can only drive the bus low or 'let go' of the bus. If the device lets go of the bus, it is pulled high by a pull-up resistor connected at each end of the cable. This configuration prevents one device from driving the bus high, whilst the other driving it low causing excessive current in the circuit, thus allowing the single wire to act a send and receive without risk.

1.3 N64 Console Commands

The N64 console initiates all communications between the controllers. Therefore, the controller will sit idle unless information is requested. All the commands I observed through probing a genuine N64 controller with a logic analyser are documented to the best of my knowledge below.

1.3.1 Controller Identify Request (0x00)

The N64 console will send an 8-bit hex 0x00 command to ask if a controller is present and if it is present, does it have something connected to its peripheral port (The port on the back of the controller).

The controller response is 24 bits long followed by a controller stop bit.

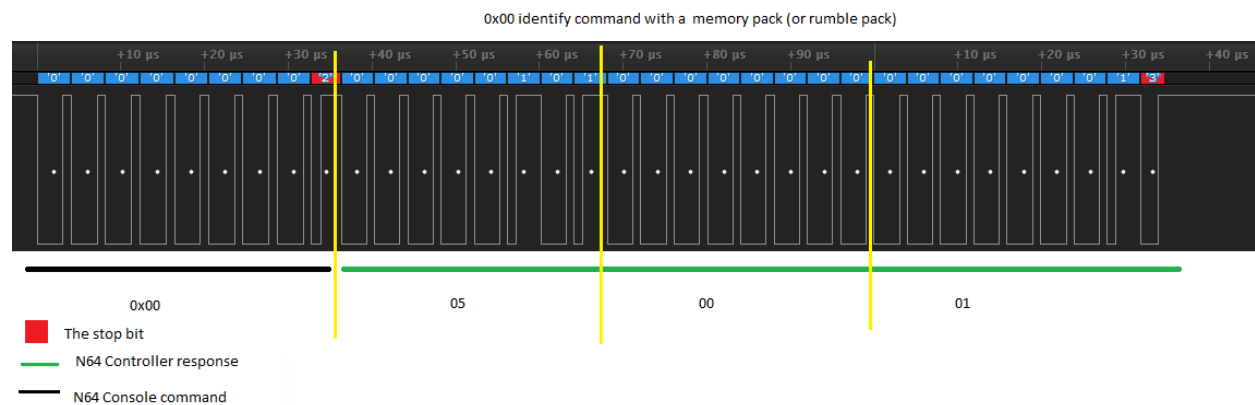
The following responses are possible (Shown in hexadecimal format):

0x050001: The controller will respond with this if it has a rumblepak or mempak installed.

0x050002: The controller will respond with this if it has no peripheral attached

0x050004: The controller will respond with this if the previous controller read/write address showed a CRC error. (See 1.3.3 and 1.3.4)

An example capture from a logic analyser is below. This specific example is with a mempak or rumblepak installed (The response is identical for either peripheral).



If this command is not properly implemented some games will report the controller as disconnected (Super Mario 64, Zelda's etc).

1.3.2 Controller State Request (0x01)

The N64 console will send an 8-bit hex 0x01 command to request the n64 controller for its current button state and axis position. This is the main polling function to each controller.

The controller response is 32 bits followed by a controller stop bit.

An example capture from a logic analyser is below. All the bits corresponding to a button are '0' when the button is not pressed. If a button is pressed, these will be logic '1's.

The position of each button/axis is labelled in the image below. Some extra notes are mentioned below.

Bit marked 1*: Appears to be a joy-stick reset request. This goes high if you press L+R+Start at the same time. In this instance 1*, L and R are reported as '1', but Start is reported as '0'.

This is a request to the N64 console to re-centre the analog stick.

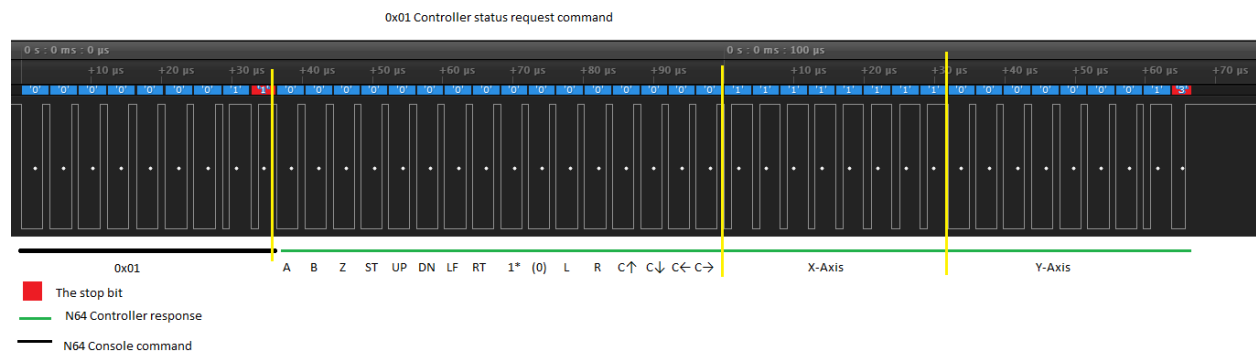
Bit marked (0): Is just always '0'

X-axis: 8-bit signed integer in 2's complement format. The 'zero' analog position reports as 0x00. This can possibly swing to +/- 128, however a genuine brand new analog stick is closer to +/- 81. Anything much greater than this can cause weird issues in some games and be overly sensitive.

Pushing the analog stick fully left will output ~-81, pushing the analog stick fully right will output ~+81 on the X-axis.

Y-axis: 8-bit signed integer in 2's complement format. The 'zero' analog position reports as 0x00. This can possibly swing to +/- 128, however a genuine brand new analog stick is closer to +/- 81. Anything much greater than this can cause weird issues in some games and be overly sensitive.

Pushing the analog stick fully up with output ~+81, pushing the analog stick fully down with output ~-81 on the Y-axis.



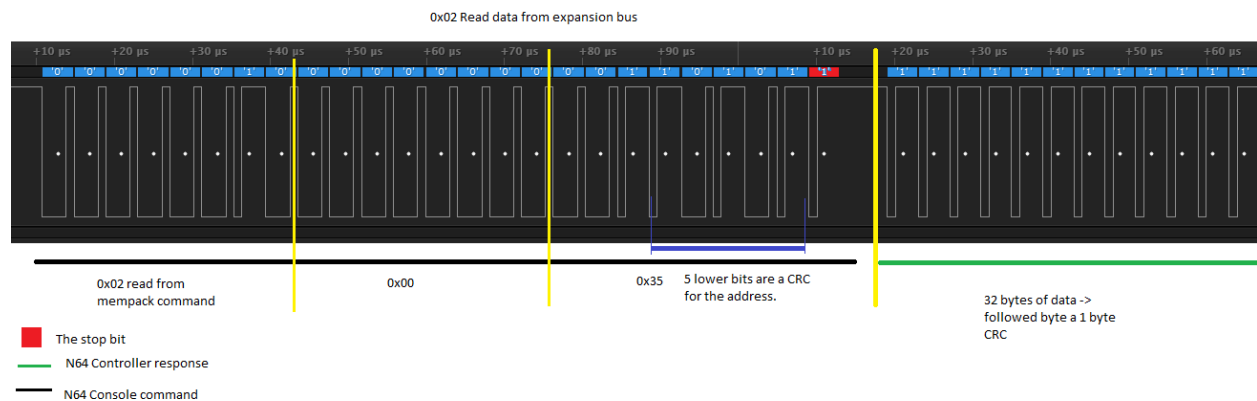
1.3.3 Read from Peripheral Bus (0x02)

The N64 console will send an 8-bit hex 0x02 command to request a read access of the controller peripheral bus. This can be a read from a rumblepak or mempak. Although the rumblepak is not specifically a memory device it is controlled by similar commands. Unfortunately, I don't believe it is possible to create a device that is simultaneously a mempak and rumblepak that would work reliably in all games.

The 0x02 command from the console is followed immediately by a 16-bit address. This address indicates the location that the read from the mempak should occur. The controller will then read from the mempak at the location specified and respond with an exactly 32-byte long data stream from that address followed immediately by an 8-bit cyclic redundancy check (CRC) of the data stream then a controller stop bit. The data CRC is discussed in Section 1.5.

The 16-bit address is managed by the console and the controller protocol does not need to manage mempak address usages. Refer to Section 1.4 for the address format and CRC. In this capture the address write happens to be at address 0x0020 (decimal 32) with an address CRC of binary 10101.

An example capture from a logic analyser can be seen below for a read command:



1.3.3.1 Exceptions:

If a read request is sent when a rumblepak is installed it must always respond with 32 bytes of 0x00 followed immediately by a data CRC unless the address is 0x8000 (see below). Some games will not recognise the rumblepak if this is not implemented (For example Tony Hawks PS2).

If a read request at address 0x8000 is attempted when a mempak is installed the controller will always respond with 32 bytes of 0x00+data CRC.

If a read request at address 0x8000 is attempted when a rumblepak is installed it will respond with 32 bytes of 0x80 if 0x80's (or any non-zero?) has previously been written to address 0x8000 by separate command, otherwise it will respond with 32 bytes of 0x00. A write command to 0x8000 consisting of 32 bytes of 0xFE will reset the state of address 0x8000, so reads after this will return 0x00s again. These must all be followed by a data CRC.

If a read request is attempted when there is no peripheral installed it will always respond with 32 bytes of 0x00 followed by an *inverted* data CRC (i.e the data CRC for all 0x00 bytes is normally 0x00, when inverted this becomes 0xFF).

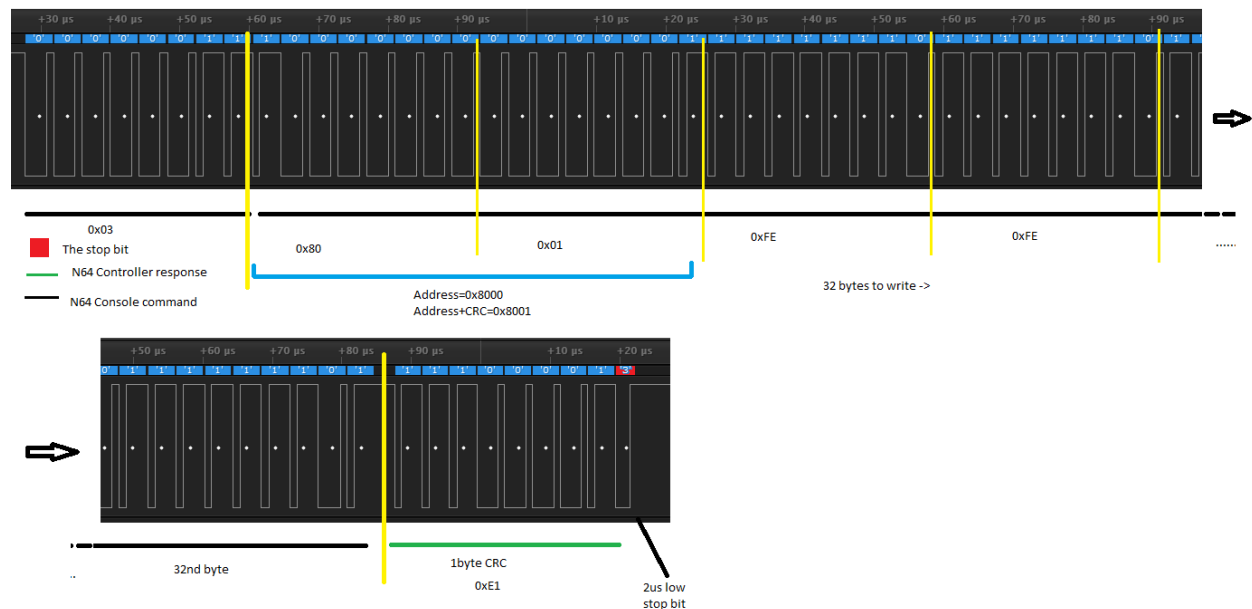
1.3.4 Write to Peripheral bus (0x03)

The N64 console will send an 8-bit hex 0x03 command to request a write access to the controller peripheral bus. This can be a write to a rumblepak or mempak. Although the rumblepak is not specifically a memory device it is controlled by similar commands.

The 0x03 command from the console is followed immediately by a 16-bit address. This address indicates the location that the write to the mempak should occur (Refer to section 1.4 for the address format) The address is then followed immediately by a 32-byte long data stream indicating the data that should be written.

The controller will read in the 32-byte data stream and respond with the calculated data CRC. The data CRC is discussed in Section 1.5. This is used by the console to verify the integrity of the data received by the controller. The data is then written to the memory device.

An example capture from a logic analyser can be seen below for a write command:



For this specific example, the write address is 0x8000 (See Section 1.4 for address format and address CRC). The mempak memory map has the range 0x0000 to 0x7FFF, this corresponds to a mempak capacity of 32kbytes. Address 0x8000 is used as an initialisation address.

1.3.4.1 Rumblepak:

The rumblepak is latched on or off by a write command to address 0xC000. (0xC01B with address CRC).

To turn on the rumblepak the N64 console sends 0x03 C01B followed by 32 bytes of 0x01. The N64 controller responds with a data CRC and turns on the rumble motor.

To turn off the rumblepak the N64 console sends 0x03 C01B followed by 32 bytes of 0x00. The N64 controller responds with a data CRC and turns off the rumble motor.

1.3.4.2 Exceptions:

If a write request is attempted at address 0x8000 when there is a rumblepak or mempak, the controller will respond with a data CRC only. No writing to the memory peripheral takes place.

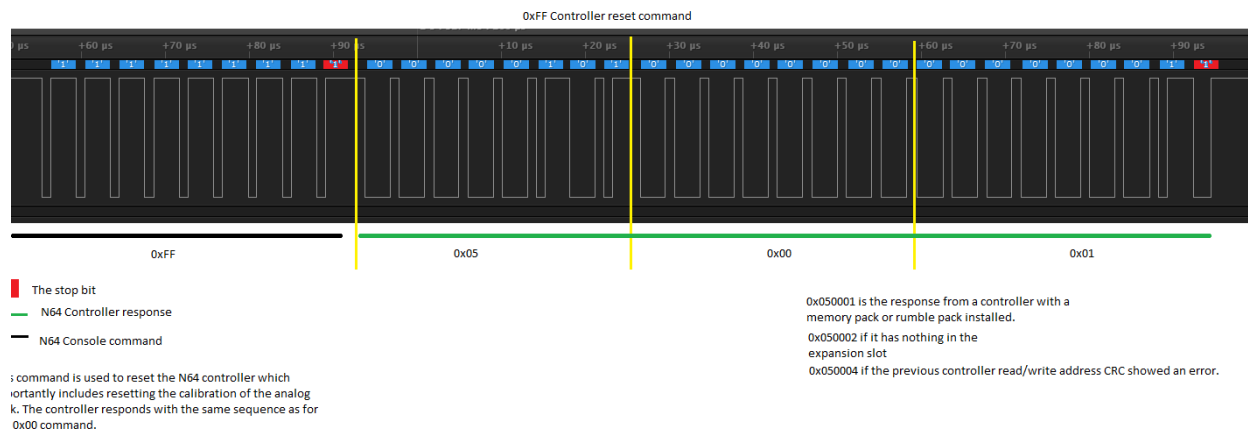
If a write request is attempted at any address when there is a rumblepak installed it will just respond with a data CRC. No writing to the memory peripheral takes place.

If a write request is attempted when there is no peripheral installed it will respond with an *inverted* data CRC. No writing to the memory peripheral takes place.

1.3.5 Reset Controller (0xFF)

The N64 console will send an 8-bit hex 0xFF command to which I believe is to force a reset of the analog stick position. The controller will just respond with the same response detailed in the 0x00 command. This response is required for some games to detect the controller (i.e Hexen).

An example capture from a logic analyser can be seen below for a 0xFF command:



1.4 Address Format and CRC

The 16-bit address is always aligned on 32-byte boundaries, meaning that the lower 5-bits of the address are not actually needed and instead are used as a 5-bit CRC of the address itself. This is used by the controller to verify the integrity of the address it has received.

The CRC is in the lowest 5-bits of the 16-bit address. The following example demonstrates the address format.

If you consider the binary value of the 16-bit address in the image from Section 1.3.3 we have.

00000000 001**10101** (0x0035)

The values highlighted is the n64 console calculated CRC of the address. The actual address is 00000000 001**00000** (the lower 5 bits are ignored when considering the actual address only).

In this image in Section 1.3.3, the address to read from is 0x0020 and when 'OR'd with the CRC of 10101 the hex final is read as 0x0035 on a data analyser. The CRC of the address is presumably re-calculated

within the N64 controller and compared to the lower 5 bits, and if they match the read proceeds. The rest of this document will consider the address only when referring to the 16-bit address.

An example C-code implementation to calculate the address CRC is below:

```
unsigned short n64_addr_encode(unsigned short addr){
    unsigned char table[11]={0x15, 0x1F, 0x0B, 0x16, 0x19, 0x07, 0x0E,
                             0x1C, 0x0D, 0x1A, 0x01};

    unsigned char i=0;

    //Mask off the lower 5 bits. These should be zero.
    addr&=0xFFE0;

    for(i=0;i<11;i++){
        if (addr & (1<<(i+5))){
            addr ^= table[i];
        }
    }
    return addr; //Returns the original address with the crc in the lower 5 bits
}
```

You can perform a quick test yourself with the follow Linux based commands using g++ compiler:

```
wget --output-document addr_encode.cpp https://pastebin.com/raw/6Hh08tj7
g++ addr_encode.cpp -o addr_encode
./addr_encode>output.txt
```

1.5 Data CRC

The data CRC is an 8-bit CRC calculated over the 32-byte long data stream.

The CRC polynomial has the form $X_7 + X_2 + X_0$ or hex 0x85. The CRC initial value is 0x00 and the output is inverted if no peripheral is connected. The CRC output is non-inverted if a rumble or mempak is installed.

A significant challenge I found with implementing my own memory pack protocol is the speed at which the CRC calculation is performed. A genuine controller after a write request will receive all the data, calculate the CRC and respond in approximately 6us. This speed is extremely challenging even with a modern powerful microcontroller. A microcontroller with a configurable hardware CRC unit is highly recommended. In any case an example C-code implementation to calculate the data CRC is below:

```

unsigned char n64_get_crc(unsigned char *data){
    unsigned char crc = 0;
    for(int i = 0; i <= 32; i++){
        for(int j = 7; j >= 0; j--){
            unsigned char tmp = 0;
            if(crc & 0x80) tmp = 0x85;
            crc <<= 1;
            if(i < 32){
                if(data[i] & (0x01 << j)){
                    crc |= 0x1;
                }
            }
            crc ^= tmp;
        }
    }
    return crc; //Returns the non-inverted CRC of a 32-byte datastring
}

```

The CRC calculation can be sped up by using a look-up table at the downside of using more space. This example uses a previously generated 256-byte lookup table:

```

unsigned char n64_get_crc_quick(unsigned char *data){
    static const unsigned char crctable[256] =
    {143,133,128,64,32,16,8,4,2,1,194,97,242,121,254,127,253,188,94,47,213,
    168,84,42,21,200,100,50,25,206,103,241,186,93,236,118,59,223,173,148,74,
    37,208,104,52,26,13,196,98,49,218,109,244,122,61,220,110,55,217,174,87,
    233,182,91,239,181,152,76,38,19,203,167,145,138,69,224,112,56,28,14,7,
    193,162,81,234,117,248,124,62,31,205,164,82,41,214,107,247,185,158,79,
    229,176,88,44,22,11,199,161,146,73,230,115,251,191,157,140,70,35,211,
    171,151,137,134,67,227,179,155,143,133,128,64,32,16,8,4,2,1,194,97,242,
    121,254,127,253,188,94,47,213,168,84,42,21,200,100,50,25,206,103,241,
    186,93,236,118,59,223,173,148,74,37,208,104,52,26,13,196,98,49,218,109,
    244,122,61,220,110,55,217,174,87,233,182,91,239,181,152,76,38,19,203,167,
    145,138,69,224,112,56,28,14,7,193,162,81,234,117,248,124,62,31,205,164,82,
    41,214,107,247,185,158,79,229,176,88,44,22,11,199,161,146,73,230,115,251,
    191,157,140,70,35,211,171,151,137,134,67,227,179,155,143,133};
    unsigned char crc = 0;

    for(int byte=0; byte<=32; byte++){
        for (int bit=0; bit<8; bit++){
            if(data[byte] & 1<<(7-bit)){
                crc ^= crctable[byte*8 + bit];
            }
        }
    }
    return crc; //Returns the non-inverted CRC of a 32-byte datastring
}

```

You can perform a quick test yourself with the follow Linux based commands using g++ compiler (edit n64_crc.cpp to change the data packet):

```

wget --output-document n64_crc.cpp https://pastebin.com/raw/nwf7aQqL
g++ n64_crc.cpp -o n64_crc
./n64_crc

```

1.6 Example Controller Initialisation

The controller initialisation sequence varies slightly from game to game; however, a typical set of commands will determine if a controller is connected to a port then determine if a peripheral is plugged into the back of the controller and finally what peripheral it is. The sequence might be:

0x00 command. The controller responds with 0x050001 if it has a mempak or rumblepak installed or 0x050002 with no pack.

The 0x00 command may repeat a couple times, the response is the same.

0x03 write command to address 0x8000 with 32 bytes of 0xFE. The controller responds with an 8-bit data CRC. The data CRC should be 0xE1. I believe this resets the initialisation state of the controller.

0x03 write command to address 0x8000 with 32 bytes of 0x00. The controller responds with an 8-bit data CRC. The data CRC should be 0x00.

0x02 read command from address 0x8000. The controller responds with 32 bytes of 0x80 + data CRC if a rumblepak is installed or 32 bytes of 0x00 + data CRC if a mempak is installed.

0x01 controller poll. The controller responds with a current button and axis state.