# PlayStation Controller Interface

Version 0.1 – Michael McCormack
mike@artpoe.com

## Foreword

The genesis of this project was in support of a FIRST robotics team I was mentoring.  The driver station of a typical FIRST team is a Windows based PC and some collection of USB attached joysticks and game controllers.  Due to the way the FIRST software operates and the habit of Windows to sometimes change the order of attached devices there is a not infrequent need to reconfigure things in the already short moments between matches.  I thought that if I could come up with a way to reduce all the USB peripherals to one device, its order in the joystick list would always be known.

Cypress provided a starting place with a custom application for a dev kit they gave to FIRST teams. This embedded application relies on analog joysticks and unfortunately these things come in only two flavors – real just or real expensive.  While looking at wired console controllers I could press into service I caught a glimpse of Sony's old SCPH-1110 Analog Joystick and I knew I found the solution. Unfortunately, while there was some information on interfacing to the ubiquitous PlayStation dual shock controllers I found nothing useful for the large analog joysticks.

The first two generations of the Sony PlayStation shared the same proprietary controller interface which is not the same as the USB interface adopted in the PS2 on.  At the physical layer, this interface is a 3.3V serial interface which is largely interoperable with commercial microcontroller SPI peripheral blocks.  At the protocol layer, there are many small differences from one type of controller to the next and not everything works seamlessly without some amount of controller specific tweaks.

The logical USB interface for the project is a composite device consisting of a standard HID required for the joystick and a virtual UART.  The UART is as a console interface for debugging and perhaps someday macro support.  The HID profile is a standard USB joystick interface overflowing with enough buttons, analog inputs, etc. to support known PlayStation controllers and to support additional sliders and buttons needed by robot drive teams .

FreeRTOS has been selected as the executive for the firmware.  This RTOS is licensed under the permissive MIT license and for consistency that license is used for the application code as well.  An RTOS is arguably overkill for this project but this firmware will also serve as a starting point for other embedded projects moving forward.  Once it is up and running, having a RTOS seems to make things cleaner and easier.  I also have been working with and expect to work with FreeRTOS in the future so it makes sense personally too.

This project relies on the work of others who explored the PSx game controller interface.  There are at present numerous references on the Internet though many websites that are found on Google no longer exist.  It is safe to assume that as time passes the number of reference websites will continue to diminish.  To the extend allowed by law I will attempt to capture and share pdf copies of reference material to provide persistent backup of the material used in development.  Word of warning though, some references use a voice of authority to provide completely information which with study has proven false.  To validate these documents and do my own investigations, I developed a sniffer for the PlayStation interface.  This is also documented in this project for your use as you see fit.

Enjoy, and should you care to extend this project, good luck

The most up to date version of this project can generally be found at:
    https://github.com/nbxmike/PSOC-Console

## Tools

While there are many sources of knowledge about the PlayStation interface on the web, they are not complete, sometimes contradict each other and in some cases they lie. So to create a USB dongle which would operate with PlayStation controllers, I found needed to investigate the operation of a real PlayStation and its controllers. In particular, I found nothing worked correctly with my SCPH-1110. I have used the following to perform my first person research:

- PlayStation extension cables – these are available for less than three U.S. dollars on a variety of websites. I did not used these as is, they were cut and attached to either my sniffer or logic analyzer.
- Noname Saleae Logic clone – That is the name that the logic analyzer software uses for the cheap Cypress FX2 based USB dongles that I used. This board is not really a clone of the Saleae Logic boards but rather a re-layout of the same chipset with, frankly, some corners cut to reduce cost. The are readily available on the web for less that ten U.S. dollars.
  *Note – should you choose to use this with the sigrok package below, visit https://sigrok.org/wiki/Fx2lafw to get the firmware needed to operate the interface. This was so obvious to everyone else in the world that it took me a while to find it.*
- sigrok protocol analyzer – I use this free bit of software on Linux though it is alleged to run on Windows and Mac. It provides a ton of features though mainly I just use its integrated SPI decoder to help look at stuff. It supports a ton of interface hardware, including the Cypress FX2 based boards mentioned above.
- PSxSniffer – I developed this myself. It is based on the same CY8CKIT-059 hardware that the main project interface is based on. I keep one on hand in addition to the project hardware as the Cypress kit is only ten U.S. dollars and it is just easier to leave it that way. The Cypress project is included in the same workspace as the interface. I also use a python script to capture the data sent by this sniffer, that code is in a directory called “Support” which is part of this project

## PlayStation Controller models

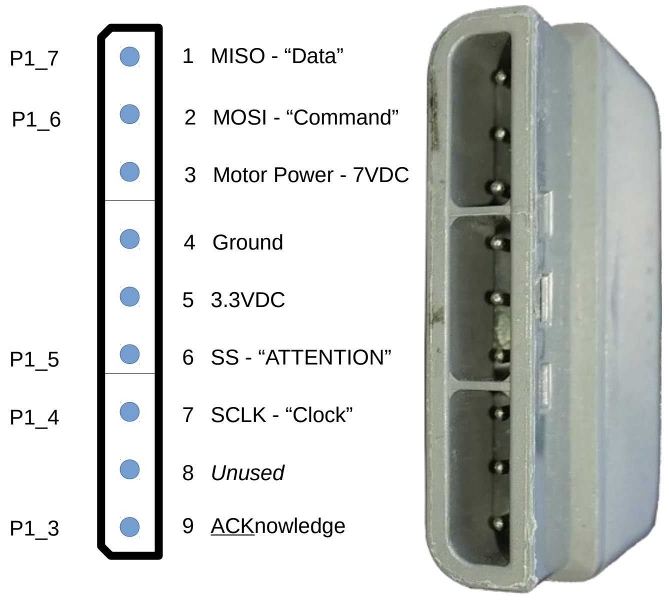
Sony has released a number of different controllers over the years. All these controllers interoperate at the hardware level with all PS1/2 consoles. As game software has progressed; however, later games often have dropped support for earlier controllers. It seems too daunting to catalog all Sony and third party controllers, so for the time being the following list of Sony produced controllers are the focus of this project:

- SCPH-1010 (1994) Original controller without joysticks but rather just the four buttons on each top side.
- SCPH-1030 (1994) Mouse
- SCPH-1080 (1996) Slightly larger controller for the North American / European PlayStation release.
- <u>SCPH-1110</u> (1996) Called the Analog Joystick this was a massive / arcade size controller.
- SCPH-1150 (1997) Dual Analog Controller released in Japan. This controller also included a single rumble motor. This started the LED color named “off” Digital mode (disables joysticks), “red” Analog mode, and “green” Analog Joystick (SCPH-1110) mode monikers
- SCPH-1180 (1997) The Dual Analog Controller for the North American market. There is no rumble on this model (SCPH-1180E for Europe)
- SCPH-1200 (1997) DualShock Analog Controller for all markets which enhanced the rumble by using two rumble motor. This also dropped the “green” Analog Joystick mode.
- SCPH-1090 (1998) Mouse
- <u>SCPH-10010</u> (2000) DualShock 2 Analog Controller added pressure sensitive buttons which can report a byte related to how hard they are pressed for all except the Start, Select, Analog, L3, and R3 buttons.
- SCPH-10150 (2000) DVD handheld Infrared Remote Control
- SCPH-10160 (2000) IR Receiver for use with the SCPH-10150 remote.
- SCPH-110 (2000) DualShock Controller for PSOne. Other than some cosmetic changes such as the PSOne logo and a slightly different connector this is the same as SCPH-10010.
- SCPH-10420 (2003) DVD Remote Control still uses 10160 receiver
- SCPH-10520 (2008?) Analog Controller which appears to be a DualShock 2 controller without rumble motors to reduce cost. It is not well documented on the web so there is limited information at the moment.

Not all these controllers are available for study at the moment, units available to the author are underlined in the list above. Also, frequently Sony released color variants and appended letters to the part numbers above; there will be no attempt to get all color variations. As finances, supply and time coincide more may be studied and this document will be updated accordingly.

## Physical Interface

The PlayStation controller interface is largely compatible with the SPI bus used on many microcontrollers. In all public documentation, the major difference between PlayStation and SPI is nomenclature. Since we have not found Sony sourced documentation, this document will use standard SPI nomenclature with the sole reference to vernacular in the following diagram.



***Note**: The first column is reflects port used during program development. It should be ignored for most purposes and will eventually get removed.*

1) MISO – the serial data stream heading to the PlayStation sent by the game controller. Data is latched by the game controller on the rising edge of SCLK. Game Controllers use open drain / collector outputs on this signal, the PlayStation provides a pull-up of apparently 10K Ohms.
2) MOSI – the serial data stream leaving the PlayStation received by the game controller. Data is latched by the PlayStation on the rising edge of SCLK.
3) Motor Power – 7.5 VDC, observed as 7.7 on my PlayStation and 7.9 on my PlayStation 2. Unconfirmed reports that the PlayStation has an 800mA or 1A fuse on the output. This is a constant power source, it is not switched on/off as some consoles do.
4) Ground – the return for power and signals. May or may not be isolated from earth ground.
5) 3.3VDC – regulated power from the PlayStation. Observed to be on the high side of nominal 3.3V at 3.45 on both a SCPH-1001 and PS2. Unconfirmed reports that the PlayStation has a 600ma or 750mA fuse on the output.
6) SS – an active low select line that must remain low during the entire packet transfer. Unlike common SPI implementations, the SS signal must not go high with each byte.
7) SCLK – the clock for the SPI, data is latched on the rising edge. Clock frequency appears to be nominally 500Khz. There are reports that specific games alter the frequency; specifically Dance Dance Revolution is reported to slow the clock to 250Khz.
8) Not populated on many interfaces and no known use (outside of maybe Sony).
9) ACKnowledge – ACK is returned from the game controller to the PlayStation with each byte. Pulled high with a 1K resistor at the PlayStation and pulled low by the controller.

The SPI bus transfers data LSB first. Data is transferred as bit fields or bytes, never larger collections, so byte ordering (hi/lo) is irrelevant

## PlayStation Packets

The PlayStation communicates to its attached game controllers in multi byte transfers that form a packet. Each packet is sent between the two devices with SS asserted continuously. A packet starts when SS is driven low and ends when SS is driven high. If the PlayStation were to drive SS high then low again in the middle of a transfer it would end one packet and start another which would of course cause the two ends to get horribly lost. Some SPI systems allow / require SS to toggle with every byte; the PlayStation keeps SS assert the whole packet long.

Over any SPI bus, data is sent from the master to the slave while simultaneously data is sent from the slave to the master regardless of whether the slave really has data to send – some value will get clocked into the master’s receive circuit. Some slave devices send packets to the master that are responses to the last package sent by the master. In these systems the master is always sending packet N and the slave I always sending response N-1; the Trimanic stepper controllers are one such example. The PlayStation game controllers however respond to the data sent by the PlayStation in the same packet just a little later. While I can’t say for sure, it appears that the controllers and PlayStation exchange a delay byte then the real response data is sent.

A PlayStation packet will always start with the same data fields. The PlayStation starts by sending a command, the game controller starts by sending its operating mode. After these first data elements are received, the two sides send data that is specific to the command from the PlayStation. These commands are documented in later sections.

### Common Header

All messages from a PlayStation to a game controller utilize a three byte header. Widely reported on the internet as constant and confirmed through observation, this header can be considered a constant.

| PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|
| Start of Packet | 0x01 | 0xFF | Start of Packet |
| Command | 0xXX | 0xMN | Mode / Status Bytes |
| Filler | 0x00 | 0x54 | Filler |

The command field is a single byte selected by the host / PlayStation. The available commands are documented later.

The Mode / Status Bytes are two nybbles which are sent by the game controller regardless of the value of the command (since the controller can’t know what that values is yet) which indicate the operating mode of the controller (nybble M) as well as the length (nybble N) in words NOT bytes of the status report as the controller is currently configured. The command send by the host may not require a status report, but the controller has to answer something in these bits and so always tells the host the number of words it would have sent.

Mode Nybble M

| Value | Mode |
|---|---|
| 0x12 | PS2 Mouse |

| Value | Mode |
|---|---|
| 0x2X | Namco NegCon |
| 0x4X | Digital, reports word report size 1 at startup, most game reconfigure to 3 |
| 0x5X | Analog Joystick (Green) |
| 0x7X | Analog Controller / Dual Shock (Red) |
| 0xFX | Setup (Electronique claims this is “DS2 native mode”) |

## Bit Fields

Several commands share the same bit fields so they are documented here as a convenience.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Buttons 0 | SEL | L3 | R3 | Start | ↑ | → | ↓ | ← |
| Buttons 1 | L2 | R2 | L1 | R1 | Δ | ○ | X | □ |

***Note**: Should the last four symbols of Buttons 1 get mangled by your system fonts, they are Triangle, Circle, X and Square.*

## Startup Anomalous Behavior

At startup, the PlayStation will perform a command sequence of 0x45, 0x46:00, 0x46:1, 0x47, 0x4C:0, 0x4C:1. No one seems to know what the data returned is used for. One source reports that 0x45 is a model number but data they observed has not yet been confirmed.

## Configuration / Button – 0x40

*Unconfirmed* - Enables pressure reporting for a single button.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x40 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x54 | Filler |
| 3 | Button # | 0xNN | 0x00 | Constant |
| 4 | Constant | 0x02 | 0x00 | Constant |
| 5 | Constant | 0x00 | 0x02 | Constant |
| 6 | Constant | 0x00 | 0x00 | Constant |
| 7 | Constant | 0x00 | 0x00 | Constant |
| 8 | Constant | 0x00 | 0x54 | Filler |

## Configuration / Report Capabilities – 0x41

*Unconfirmed* – The controller returns a bit field of its ability to provide different polling results. It is dependent on the controller's mode, that is it reports capabilities only in the present context (e.g. it can't return analog data in digital mode so it does not set those bits).

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x41 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x5A | Filler |
| 3 | Filler | 0x5A | 0xSS | Enabled Reports 0 |
| 4 | Filler | 0x5A | 0xTT | Enabled Reports 1 |
| 5 | Filler | 0x5A | 0xUU | Enabled Reports 2 |
| 6 | Filler | 0x5A | 0x00 | Filler |
| 7 | Filler | 0x5A | 0x00 | Filler |
| 8 | Filler | 0x5A | 0x5A | Filler |

When this command is issued, the PlayStation the full number status bytes are read, though they are not typically valid (seemingly always 00).

## Poll & Motors – 0x42

*Unconfirmed* - The PlayStation sends this message to get the current status of the user input and, when appropriate, set the feedback motors. The motor fields in byte 3 and 4 are dependent on the mapping used in command 0x4D:

- The small motor, normally motor 0, will only turn on for value 0xFF though it is customarily turned off with value 0x00.
- The large motor, normally motor 1, can operate at variable speeds though values less than 0x40 may not be sufficient to turn on the motor.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x42 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x5A | Filler |

Data fields depend on the value of the Status Bytes field and previous configuration commands. If the motors were mapped with a 0x4D Map Motor command, they are used. Analog data is returned only if $N \geq 3$; Pressure data only if $N \geq 9$.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 3 | Motor 0 | 0xWW | 0xXX | Buttons 0 |
| 4 | Motor 1 | 0xYY | 0xXX | Buttons 1 |

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 5 | Filler | 0x00 | 0xXX | Right Analog X |
| 6 | Filler | 0x00 | 0xXX | Right Analog Y |
| 7 | Filler | 0x00 | 0xXX | Left Analog X |
| 8 | Filler | 0x00 | 0xXX | Left Analog Y |
| 9 | Filler | 0x00 | 0xXX | → Pressure |
| 10 | Filler | 0x00 | 0xXX | ← Pressure |
| 11 | Filler | 0x00 | 0xXX | ↑ Pressure |
| 12 | Filler | 0x00 | 0xXX | ↓ Pressure |
| 13 | Filler | 0x00 | 0xXX | Δ Pressure |
| 14 | Filler | 0x00 | 0xXX | ○ Pressure |
| 15 | Filler | 0x00 | 0xXX | X Pressure |
| 16 | Filler | 0x00 | 0xXX | □ Pressure |
| 17 | Filler | 0x00 | 0xXX | L1 Pressure |
| 18 | Filler | 0x00 | 0xXX | R1 Pressure |
| 19 | Filler | 0x00 | 0xXX | L2 Pressure |
| 20 | Filler | 0x00 | 0xXX | R2 Pressure |

## Configuration & Poll – 0x43

*Unconfirmed* - The PlayStation seems to always read all of the data in the report when it sends this command though web sources indicate that it can be truncated.  As with 0x42 Poll, the number of data fields is specified in the game controller response byte 1.  Motor control is not possible with this command.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x5A | Filler |
| 3 | Exit Configuration<br>Enter Configuration | 0x00<br>0x01 | 0xXX | Buttons 0 |
| 4 | Filler | 0x5A | 0xXX | Buttons 1 |
| 5 | Filler | 0x5A | 0xXX | Right Analog X |
| 6 | Filler | 0x5A | 0xXX | Right Analog Y |
| 7 | Filler | 0x5A | 0xXX | Left Analog X |
| 8 | Filler | 0x5A | 0xXX | Left Analog Y |
| 9 | Filler | 0x5A | 0xXX | → Pressure |

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 10 | Filler | 0x5A | 0xXX | ← Pressure |
| 11 | Filler | 0x5A | 0xXX | ↑ Pressure |
| 12 | Filler | 0x5A | 0xXX | ↓ Pressure |
| 13 | Filler | 0x5A | 0xXX | ∆ Pressure |
| 14 | Filler | 0x5A | 0xXX | ○ Pressure |
| 15 | Filler | 0x5A | 0xXX | X Pressure |
| 16 | Filler | 0x5A | 0xXX | □ Pressure |
| 17 | Filler | 0x5A | 0xXX | L1 Pressure |
| 18 | Filler | 0x5A | 0xXX | R1 Pressure |
| 19 | Filler | 0x5A | 0xXX | L2 Pressure |
| 20 | Filler | 0x5A | 0xXX | R2 Pressure |

## Configure Mode – 0x44

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode. The command will set the controller and can also optionally lock the game controller into this mode (i.e. the mode button does not work when the used presses it.)

Sending this command will cause the pressure button sense configuration to revert to the reset state of off (i.e. If you put the game controller into analog mode, enable pressure reporting for buttons then issue a command to put it in analog mode pressure reporting is disabled again.)

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x44 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x5A | Filler |
| 3 | Enter Digital mode<br>Enter Analog Mode | 0x00<br>0x01 | 0x00 | Filler |
| 4 | Lock if 0x03 ONLY | 0x0X | 0x00 | Filler |
| 5 | Filler | 0x00 | 0x00 | Filler |
| 6 | Filler | 0x00 | 0x00 | Filler |
| 7 | Filler | 0x00 | 0x00 | Filler |
| 8 | Filler | 0x00 | 0x00 | Filler |

When this command is issued by a real PlayStation the full number status bytes are read, though they are not typically valid (seemingly always 00).

## Query – 0x45

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode.  Appears to report some form of model number, or according to some a constant value.  Data is reported all differently on the web, so more investigation is needed.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x5A | Filler |
| 3 | Filler | 0x5A | 0xXX | |
| 4 | Filler | 0x5A | 0xXX | |
| 5 | Filler | 0x5A | 0xXX | |
| 6 | Filler | 0x5A | 0xXX | |
| 7 | Filler | 0x5A | 0xXX | |
| 8 | Filler | 0x5A | 0xXX | |

Byte 3 – In Curious Inventor - 01 for Guitar Hero, 03 for Dual Shock
Byte 5 – In Curious Inventor – LED status of 00 for off, 01 for on.

## Query – 0x46

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode.  Appears to report some form of model number, or according to some a constant value.  Data is reported all differently on the web, so more investigation is needed.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x54 | Filler |
| 3 | | | 0xXX | |
| 4 | | | 0xXX | |
| 5 | | | 0xXX | |
| 6 | | | 0xXX | |
| 7 | | | 0xXX | |
| 8 | | | 0xXX | |

## Query – 0x47

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode. Appears to report some form of model number, or according to some a constant value. Data is reported all differently on the web, so more investigation is needed.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x54 | Filler |
| 3 | | | 0xXX | |
| 4 | | | 0xXX | |
| 5 | | | 0xXX | |
| 6 | | | 0xXX | |
| 7 | | | 0xXX | |
| 8 | | | 0xXX | |

## Query – 0x4C

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode. Appears to report some form of model number, or according to some a constant value. Data is reported all differently on the web, so more investigation is needed.

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x54 | Filler |
| 3 | | | 0xXX | |
| 4 | | | 0xXX | |
| 5 | | | 0xXX | |
| 6 | | | 0xXX | |
| 7 | | | 0xXX | |
| 8 | | | 0xXX | |

## Configure Motor Map – 0x4D

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode. The Map Motors command determines the function of the command bytes 3 and

4 of the 0x42 Poll command. At power up, all motors are disabled as 0xFF is reported for all previous values. The following are the motor codes:

0x00 – Small motor

0x01 – Large motor

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x54 | Filler |
| 3 | Motor 0 | 0xNN | 0xPP | Previous Value |
| 4 | Motor 1 | 0xMM | 0xTT | Previous Value |
| 5 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |
| 6 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |
| 7 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |
| 8 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |

0xNN is always either 0x00 or 0xFF on a real PlayStation
0xMM is always either 0x01 or 0xFF on a real PlayStation

## Configure report data – 0x4F

*Unconfirmed* – This command is only valid when the Mode nybble is F; that is the game controller is in configuration mode. The Map Motors command determines the function of the command bytes 3 and 4 of the 0x42 Poll command. At power up, all motors are disabled as 0xFF is reported for all previous values. The following are the motor codes:

0x00 – Small motor

0x01 – Large motor

| Byte | PlayStation | MOSI | MISO | Game Controller |
|---|---|---|---|---|
| 0 | Start of Packet | 0x01 | 0xFF | Start of Packet |
| 1 | Command | 0x43 | 0xMN | Mode / Status Bytes |
| 2 | Filler | 0x00 | 0x54 | Filler |
| 3 | Motor 0 | 0xNN | 0xPP | Previous Value |
| 4 | Motor 1 | 0xMM | 0xTT | Previous Value |
| 5 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |
| 6 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |
| 7 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |
| 8 | Filler | 0xFF | 0xXX | *Previous Value (assumed)* |

0xNN is always either 0x00 or 0xFF on a real PlayStation

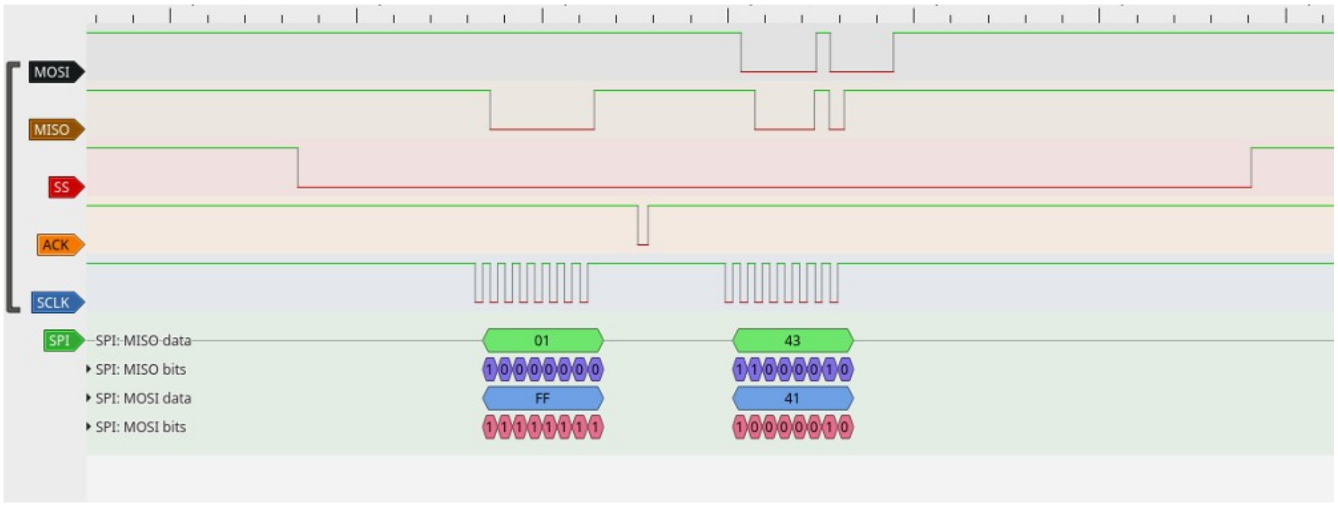
0xMM is always either 0x01 or 0xFF on a real PlayStation
A

## SCPH-1110 & ACKs

The PlayStation Analog Joystick was one of the first controllers released by Sony. It can operate in “Analog” mode where it reports the position of the joystick as *digital* values between -128 and 127 or is can operate in “Digital” mode where it reports the joysticks as arrow buttons. The game controller has a large switch that is used to change between the two modes.

Unlike later Dual Shock, and maybe the Dual Analog, controllers, the Analog Joystick will not enter config mode. This means the controller cannot be switched between modes by the PlayStation or host. It also means that this controller cannot report it’s model or other information typically exchanged while being configured. However, a PlayStation will *try* to put the controller into config mode, which is futile.

This inability to change modes or enter config resulted in an interesting discovery, the ACK line is absolutely necessary to emulating the PlayStation with an adapter. When the SCPH-1110 starts up, it returns a 0x41 in digital mode or 0x53 if it is in analog mode when the PlayStation first polls it. However, when the PlayStation sends a type 0x43 Poll-Config message, the SCPH-1110 does not ACKnowledge the Command Byte as shown in the logic analyzer trace:



*NOTE MOSI and MISO are swapped on my logic analyzer so that the PlayStation is on top and the controller is below. If I learn how to change the order I will change the names*

The PlayStation repeats the attempt to send a 0x43 message twelve times before it apparently decides to give up. When the switch is changed in the middle of a game, this is repeated at the same ~15 millisecond rate used to normal polling.

Also, the controller has two sets of buttons, large buttons in the middle and another set distributed over the joystick handles. Each button returns the same code when pressed. In other words, there is no way to know if, for example whether the R1 button that was pressed is the one in the center cluster or the one on the left stalk at the thumb.

## References

### USB

Microsoft explanation of USB-IADs – LINK
Intel explanation of USB-IADs – LINK
Cypress explanation of USB-IADs – LINK1, LINK2

### PlayStation 1 / 2 Interface

Sony Playstation Controller Port – LINK
PS2 Command Set by Nathan Scherdin – LINK
PS1 Pad Interface – LINK
PlayStation 2 Controller Arduino Library v1.0 – LINK
PSX Controllers – LINK
Playstation.txt by Joshua Walker – LINK
The analyze infomation of PAD-controler and Memery I/F in PlayStation – LINK
Sony Playstation (PSX) joystick controller port pinout – LINK
How to interface Arduino with a Playstation game-port controller – LINK
Interfacing PS2 controller with AVR Bit Bang – LINK
How PlayStation Works  – LINK
PSP-Nx-v4 User Guide – LINK
Arduino PS2 Controller Interface – LINK
Decoding PS2 Wired and Wireless Controller for Interfacing with Pic Micro Controller – LINK