Back

Serial interface for Playstation pad

Introduction

The playstation pad is fantastic to control anything you want because there are 14 buttons and 2 analogic joystics !
I made this interface to controle my webcam and also the robot which participate to the French Robotic Cup (E=m6).
I used the serial port because usual interface I found used the parallele port, but the robot already use it for something else and it had a free serial port. More, with a microcontroller it is easy to have all the psx pad protocole decoded outside the PC. The only negative point is that you have to made a board...

Playstation pad protocol

The Playstation Controller Pinouts

```
LOOKING AT THE PLUG
       -------------------------------
 PIN 1->| o  o  o | o  o  o | o  o  o |
```

```
        \_____________________________/
```

PIN # USAGE

1. DATA
2. COMMAND
3. N/C (9 Volts unused)
4. GND
5. VCC
6. ATT
7. CLOCK
8. N/C
9. ACK

DATA
: Signal from Controller to PSX.
This signal is an 8 bit serial transmission synchronous to the falling edge of clock (That is both the incoming and outgoing signals change on a high to low transition of clock. All the reading of signals is done on the leading edge to allow settling time.)

COMMAND
: Signal from PSX to Controller.
This signal is the counter part of DATA. It is again an 8 bit serial transmission on the falling edge of clock.

VCC
: VCC can vary from 5V down to 3V and the official SONY Controllers will still operate.

ATT
: ATT is used to get the attention of the controller.
This signal will go low for the duration of a transmission. I have also seen this pin called Select, DTR and Command.

CLOCK
: Signal from PSX to Controller.
Used to keep units in sync.

ACK
: Acknowledge signal from Controller to PSX.
This signal should go low for at least one clock period after each 8 bits are sent and ATT is still held low. If the ACK signal does not go low within about 60 us the PSX will then start interogating other devices.

It should also be noted that this is a bus of sorts. This means that the wires are all tied together (except select which is seperate for each device). For the CLK, ATT, and CMD pins this does not matter as the PSX is always the originator. The DATA and ACK pins however can be driven from any one of four devices. To avoid contentions on these lines they are open collectors and can only be driven low.

The PSX Controller Signals

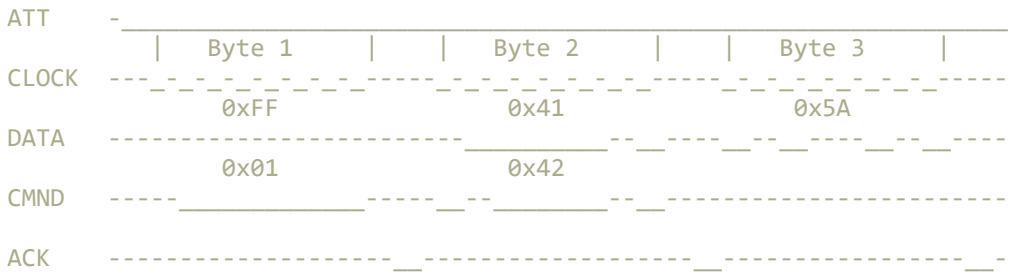
All transmissions are eight bit serial LSB first. All timing in the PSX controller bus is syncronous to the falling edge of the clock. One byte of the transmissions will look kinda like this.

```
            |BIT 0|BIT 1|BIT 2|BIT 3|BIT 4|BIT 5|BIT 6|BIT 7|
CLOCK -----___---___---___---___---___---___---___-----------

DATA  -----0000001111112222223333334444445555556666667777777--------
               *     *     *     *     *     *     *     *
CMND  -----0000001111112222223333334444445555556666667777777--------

ACK   ------------------------------------------------------__-
```

The logic level on the data lines is changed by the transmitting device on the falling edge of clock. This is then read by the receiving device on the leading edge (at the points marked *) allowing time for the signal to settle. After each COMMAND is recieved by a selected controller, that controller needs to pull ACK low for at least one clock tick. If a selected controller does not ACK the PSX will assume that there is no controller present.

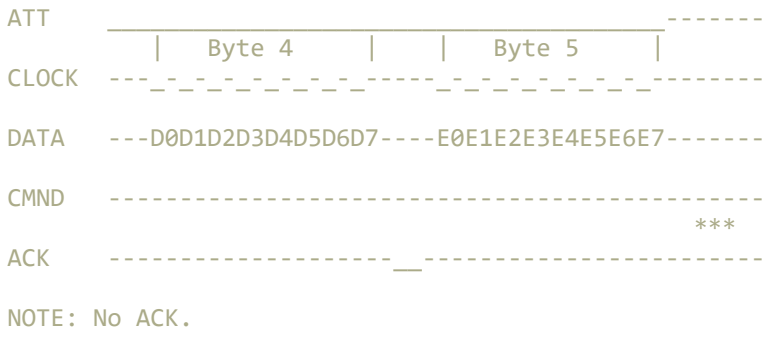
When the PSX wants to read information from a controller it pulls that devices ATT line low and issues a start command (0x01). The Controller Will then reply with its ID (0x41=Digital, 0x23=NegCon, 0x73=Analogue Red LED, 0x53=Analogue Green LED). At the same time as the controller is sending this ID byte the PSX is transmitting 0x42 to request the data. Following this the COMMAND line goes idle and the controller transmits 0x5A to say "here comes the data".

This would look like this for a digital controller



After this command initiation proccess the controller then sends all its data bytes (in the case of a digital controller there is only two). After the last byte is sent ATT will go high and the controller does not need to ACK.

The data transmision for a digital controller would look like this (where A0,A1,A2...B6,B7 are the data bits in the two bytes).



NOTE: No ACK.

The PSX Controller Data

Below are five tables that show the actual bytes sent by the controllers

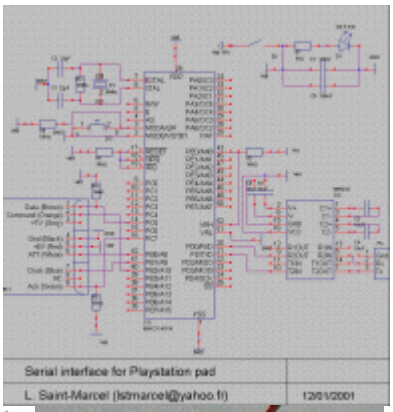
Standard Digital Pad

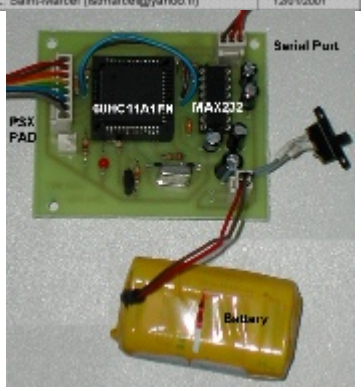
BYTE	CMND	DATA							
01	0x01	idle							
02	0x42	0x41							
03	idle	0x5A	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
04	idle	data	SLCT			STRT	UP	RGHT	DOWN	LEFT
05	idle	data	L2	R2	L1	R1	/\	O	X	[]

```
All Buttons active low.


Analogue Controller in Red Mode

BYTE    CMND    DATA

 01     0x01    idle
 02     0x42    0x73
 03     idle    0x5A    Bit0 Bit1 Bit2 Bit3 Bit4 Bit5 Bit6 Bit7
 04     idle    data    SLCT JOYR JOYL STRT UP   RGHT DOWN LEFT
 05     idle    data    L2   R2   L1   R1   /\   O    X    []
 06     idle    data    Right Joy 0x00 = Left  0xFF = Right
 07     idle    data    Right Joy 0x00 = Up    0xFF = Down
 08     idle    data    Left Joy  0x00 = Left  0xFF = Right
 09     idle    data    Left Joy  0x00 = Up    0xFF = Down

All Buttons active low.
```

Electronic - The board

I used a Motorola 68HC11A1FN (the only microcontroller I had to make a new board) which is almost directly connected to the PAD; there are only two pull up resitances on 'Data' and 'Ack' pins (68hc11 inputs)). And it is interfaced with the PC's serial port using a MAX232.

- Electronic schem : 

- Picture of the board : 

- Wyntypon : psx.wt3
  Gif : psx.gif

Software

Embedded software for 68HC11A1FN (loaded at $B600). Note that the microcontroller E0 pin is set to +5V so that Buffalo is not run when the board is reset but the program directly start at $B600 : psxpad.asc.

To assembly the program and download it, set the 68hc11 pin MODB to GND and you may use the flowwing programs : 68hc11_psx.zip

The embedded proggram continously send requests to the pad and save the results in its 7bytes of its RAM. Then when it received a byte (not $FF) on the serial port, it sends 7 bytes corresponding to the data to the PC. Here is a sample in C++ (for Visual C++) which continuously sends request and display data from the psx pad : cpp_psx.zip

---

Download

Download all necessary files : psx_all.zip

Last update : 12/12/2001

lstmarcel@yahoo.fr