JAMMA Video Standard (JVS)
Third Edition

Designed by JAMMA
Translated/edited/etc. by Alex Marshall

August 14, 2019

Contents

List of Tables

1 Introduction

1. **Scope of application:** This standard defines the main board used for general video game machines, which satisfies the technical requirements of the Electrical Appliance and Material Control Law, and the Electrical Appliance and Material Safety Law.
2. **Terminology used:** The meaning of the terms used in this standard is as follows.
   - 2.1 **General Video Game Machine:** A coin operated game machine using a CRT monitor.
   - 2.2 **Main PCB:** The central board used to drive the game machine.
   - 2.3 **Video Output:** The outputs from the main PCB to the CRT, consisting of the RGB, and sync signals.
   - 2.4 **Sync Output:** A composite signal with no color data.
   - 2.5 **RGB Output:** Individual color signals, with no sync data.
   - 2.6 **Video Signal Timing Chart:** Shows the change of sync and color signals over time for each vertical and horizontal scan.
   - 2.7 **Raster:** One scanline's worth of video output.
   - 2.8 **DC Power Input:** Various DC power supply inputs supplied to the main PCB.
   - 2.9 **Audio Output:** Analog audio output from the main PCB.
   - 2.10 **I/O Board:** Connects by serial to the main PCB, controls switch inputs, volume, coin counters, lamps, etc.
   - 2.11 **Connectors:** Various connectors used when electrically connecting main PCB and external peripheral device.
   - 2.12 **JAMMA Mark:** Abbreviation logo trademarked by Japan Amusement Machine Industry Association.
3. **Video Output**
   - 3.1 **Video Output:** Video can be output to two monitors.
   - 3.2 **Horizontal Scan Frequency:** The horizontal scan frequence shall be 15.750kHz, 24.830kHz, or 31.500kHz.
   - 3.3 **RGB Output**
     - 3.3.1 **Levels:** RGB output levels shall be 0.7Vp-p±10%.
     - 3.3.2 **Signaling:** RGB output signals are AC coupled, with a 470μF capacitance. Connect to GND with a terminating resistor of 75Ω.
   - 3.4 **Sync Output**
     - 3.4.1 **Levels:** Sync output level shall be TTL levels, with high level voltage being 2.4V through 5.0V.
     - 3.4.2 **Signaling:** Sync signals are DC coupled. Connect to GND with a terminating resistor of 470Ω.
     - 3.4.3 **Types:** Sync output is either Composite only, or Separate. When composite, V-sync is not connected, or TTL low level.
   - 3.5 TODO: Bring back once I add the graphs (see 3.4 in JVST_VER3.pdf if you really care, it's just referring to a chart of video signals).
4. **DC Power Input**
   - 4.1 **Types:** DC power inputs should be +5V0, +3V3, and +12V0-.
   - 4.2 **Margins:** Voltage error, current capacity, ripple, and minimum load current of each DC power input are shown in the below table.

Table 1: DC Power Input Requirements

Power Input	Error	Current Capacity	Ripple	Minimum Load Current
+5V0	±5%	Up to 10A	Up to 80mV p-p	0A
+3V3	±5%	Up to 12A	Up to 80mV p-p	0A
+12V0	±10%	Up to 2A	Up to 120mV p-p	0A

   - 4.3 **Power On Sequence:** The +3V3 rail must come up completely before +5V0. The time between +3V3 reaching 90% and +5V0 reaching 10% must be greater than or equal to 1 millisecond. The time between +3V3 reaching 90% and +5V0 reaching 90% must be between 1 and 1000 milliseconds. All power terminals must be connected, there must be no potential difference between each terminal.
   - 4.4 **Power Off Sequence:** The +5V0 rail must turn off completely before +3V3. The time between +5V0 reaching 90% and +3V3 reaching 90% must be between 1 and 1000 milliseconds.
   - 4.5 TODO: Add the nice diagrams for these

5. **Audio Output**
   - 5.1 **Levels:** Audio output level is 0±6 dB (0V5-2V0 rms, or 1V41-5V66 p-p), with output impedance of 2kΩ or less.
   - 5.2 **Output Channels**
     - 5.2.1 **Stereo output:** Audio output should use two channels.
     - 5.2.2 **Extended output:** Audio expansion output can use up to two channels.
     - 5.2.3 **Mono output:** Audio output should be the same as stereo output, just with the same data.
   - 5.3 **Volume Adjustment:** Volume adjustment is done on the cabinet itself, not on the main PCB.
6. **Standard I/O**
   - 6.1 **Standard I/O:** The I/O refers to the unit which communicates with the main PCB, and controls reading of switches, levers, coins, volume, etc, and controls coin counters, lamp drivers, etc.
     - 6.1.1 **Signal types:** The I/O is connected using a serial connection conforming to EIA-RS485 standard (half duplex).
     - 6.1.2 **Communication Setup**
       - A. Read from the I/O board: the manufacturer, product number, version, etc.
       - B. Allocate addresses for each node in sequence.
       - C. In the future, when version upgrade of the basic part such as communication speed is done, confirm the correspondence by confirm command and switch all nodes to new version by setting command. (TODO: retranslate)
     - 6.1.3 **Coin Count Management**
       - A. The I/O board manages the number of coins, and number of pay-outs. It may save this data in case of power loss.
       - B. The I/O board notifies the main PCB of the number of available coins (or remaining credit for a card system). The main PCB sends information such as number of coins to consume when a game is started.
       - C. The I/O board additionally manages number of pay-outs, such as medal hoppers, and the main PCB only conveys the amount won.
     - 6.1.4 **Command Format:** Command format is specified later.
7. **Connectors**
   - 7.1 **Connector types:** The connector types are unique for each connector's functionality (DC power supply, video output, audio output, standard I/O input, standard I/O output).
   - 7.2 TODO: The rest of this section (mostly not important, requires diagrams)
8. **Main PCB Dimensions**

Table 2: Maximum size of main PCB

Dimension	**mm**	**in**
Width	390mm	15.3"
Height	450mm	17.7"
Depth	150mm	5.9"

9. **TODO: items 9 10 and 11 omitted for irrelevance**

TODO: Really nice diagrams

2 Physical Layer

The JVS I/O physical layer uses RS-485 standard for serial transmission. The physical connection uses USB-A and USB-B connectors. The protocol is half-duplex multi-point, so all *DATA* lines are shared. The *Sense* line is point-to-point between adjacent nodes, and pull-down. Each device which supports daisy-chaining will have two JVS I/O ports, with the *outgoing* port being a USB-A connector, and the *incoming* port being a USB-B connector. The master node will thus only have a USB-A connector, and a slave node without daisy chaining will only have a USB-B connector.

Table 3: Physical Layer Pin-out

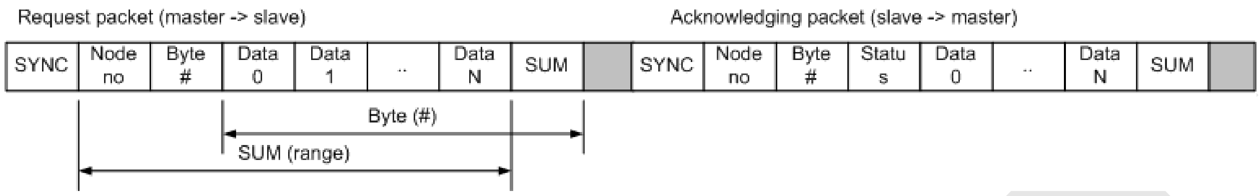
USB Pin	**USB Color**	**JVS Signal**	**RS-485 Signal**
1	Red	Sense	*N/A*
2	White	DATA-	A
3	Green	DATA+	B
4	Black	GND	GND

Table 4: Physical Layer Attributes

Attribute	**Value**
Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	1

3 Link Layer

3.1 Framing



3.1.1 SYNC and MARK

All frames start with a SYNC byte (`0xE0`). When any SYNC byte is encountered, a new packet begins. To allow using the SYNC byte in the payload, there is a MARK byte (`0xD0`) which increments the following byte by one. Thus, a real byte of `0xE0` is encoded as `0xD0 0xDF`, and a real byte of `0xD0` is encoded as `0xD0 0xCF`. The MARK processing is transparent, and as such the rest of the protocol is completely ignorant to its existence. All processing should occur on raw data before MARK addition or after MARK removal.

3.1.2 Node No.

All frames include the destination node number. If the node number is `0xFF`, that indicates that the packet is *Broadcast*, and thus meant for all nodes. Node `0x00` is fixed to be the master node, and nodes `0x01` through `0x1F` are for slave nodes, to be assigned at initialization. Each device should only process packets designated for it.

3.1.3 Byte Count

This field contains the number of bytes left in the packet, including the SUM byte.

3.1.4 SUM

All bytes in the packet aside from SYNC and SUM are added together, modulo 256. If this value and SUM do not match, the packet is corrupt.

3.2 Request/Acknowledge Scheme

3.2.1 Requests

The master is the only node allowed to initiate communication, to prevent cases where multiple devices try to talk at once and blow each other up. To improve performance, multiple commands may be combined into a single packet simply by concatenating them. The acknowledge packet must include responses to all of the commands in the packet.

Listings of the Command codes can be found in Section 4.1.

3.2.2 Acknowledges

Every request (except for most broadcast packets) require an acknowledge packet in response which includes any return values requested. Each acknowledge packet starts with a *Status* code, which provides information on what the slave node thinks about the incoming *packet*. For each command in the request packet, there is one *Report* code which provides information on what the slave node thinks about that specific command. Actual response data follows the Report code.

Listings of the Status and Report codes can be found in Section 4.2.

3.3 Initialization

For initialization, the master should send a *Reset* packet twice to reset all slave nodes to their uninitialized state. This will cause them to set their sense lines high, and to forget their node number.

Following, the master will send *Set Address* packets with increasing address values. This command is the only broadcast command that has a response, which acknowledges that a node received the command.

When a node receives this packet, it should ignore it unless its incoming sense line is low. This makes the farthest node receive address `0x01`, and the closest to receive the last node. When a node has its address set, it should pull sense low.

Master should stop sending packets once its incoming sense goes low, as this means all nodes have addresses.

4 Command Format

Exchange of data between the *main board* and the *I/O board* is performed by a command sent by the master (main board). Since the slaves are unable to send a request to the master, the master must periodically check the slaves for data.

The overhead for transmitting a single packet at a time is relatively large, so typically many packets are sent at once for more efficient data exchange. When a slave receives multiple packets at once, it must process them in order and reply in the order received. If the data takes a long time to process, busy is returned during processing.

4.1 Command Codes

Table 5: Command Listing

Req.	**Name**	**Code**	**Description**	**Request Size**	**Response Size**
GLOBAL COMMANDS (NODE NO.FF)					
○	*Reset*	`F0`	Reset communication status of all slaves	2	0
○	*Set Address*	`F1`	Set the slave address	2	1
	Comm. Method Change	`F2`	Request to change communication speed?	2	0
INITIALIZE COMMANDS					
○	*I/O Identify*	`10`	Get slave ID data	1	MAX 102
○	*Command Revision*	`11`	Command format revision	1	2
○	*JVS Revision*	`12`	JVS revision	1	2
○	*Comm. Version*	`13`	Communications version	1	2
○	*Feature Check*	`14`	Check slave features	1	6+
	Main ID	`15`	Send ID of main board	MAX 102	1
DATA I/O COMMANDS					
	Switch Inputs	`20`	Read controller switches	3	3+
	Coin Inputs	`21`	Read coin counters	2	2 × *slots*
	Analog Inputs	`22`	Read analog inputs	2	2 × *ports*
	Rotary Inputs	`23`	Read rotary inputs	2	2 × *ports*
	Keycode Inputs	`24`	Read keycodes?	1	2
	Screen Position Inputs	`25`	Read touch-panel/gun inputs	2	5
	Misc. Switch Inputs	`26`	Read miscellaneous switch inputs	2	2+
OUTPUT COMMANDS					
	Remaining Payout	`2E`	Read information such as remaining number of medals in hopper?	2	5
○	*Data Retransmit*	`2F`	Checksum error, request a retransmission	1	0
	Coin Counter Decrease	`30`	Decrement the coin counter	4	1
	Payout Counter Increase	`31`	Increment the payout counter	4	1
	Generic Output 1	`32`	Data output to the parallel driver	3+	1

	Analog Output	`33`	Data output to analog output unit	4+	1
	Character Output	`34`	ASCII code output to 7seg LED or LCD	3+	1
	Coin Counter Increase	`35`	Increment the coin counter	4	1
	Payout Counter Decrease	`36`	Decrement the payout counter	4	1
	Generic Output 2	`37`	Byte data output to the parallel driver?	3	1
	Generic Output 3	`38`	Bit data output to the parallel driver?	3	1
MANUFACTURER-SPECIFIC COMMANDS					
		`60`			
		`..`			
		`7F`			
TAITO TYPEX COMMANDS (NODE NO.00)					
	Machine Reset	`00`	Reset the machine	1	1
	DIP Switches	`01`	Get TypeX board DIP switches	2	2
	Unknown 02	`02`	Unknown	?	2
	Unknown 03	`03`	Unknown	?	1
	Watchdog? 04	`04`	Unknown? Watchdog related	1	1
	Watchdog Set	`05`	Set watchdog timer expiration time	3	1
	Unknown 06	`06`	Unknown	?	2
	Unknown 07	`07`	Unknown	?	1
	Watchdog Kick	`08`	Watchdog kick	1	2

4.2 Status and Report Codes

Status is sent with each response to a request packet, to convey the current status to the master. Report is sent with each response to a single request command to tell the results of the command to the master.

When a slave receives a command that it does not support, all subsequent packets are discarded, and returns the *Unknown command* status. Also, if the command that was executed it previously, report the acknowledge packet (if there is a parameter that also) I return.

Table 6: Status Code Listing

Code	Description
`01`	Normal
`02`	Unknown command (Unsupported command)
`03`	Checksum error
`04`	Acknowledge overflow (Acknowledge data is too large)

Table 7: Report Code Listing

Code	Description
`01`	Normal
`02`	Parameter error (parameter is invalid). An incorrect number of parameters were sent.
`03`	Parameter error (parameter is invalid). The data supplied was invalid.
`04`	Busy. The I/O board cannot receive more commands until this command is complete.

4.3 Detailed Command Description

4.3.1 Command: Reset (RESET)

Command name	Request Data		Response Data	
Reset	**F0, D9**		–	
byte0	**F0**	Command Code	–	
byte1	**D9**	Command Code	–	

Resets the communication status of all slaves. Reset is a critical command, so there are two bytes to help ensure that the command is not issued mistakenly. Further, since reliability is paramount, it is recommended that this command be issued twice.

Initialization of the non-communication functions of the I/O board is not defined.

4.3.2 Command: Set Address (SETADDR)

Command name	Request Data		Response Data	
Set Address	**F1, address**		**01 (report)**	
byte0	**F1**	Command Code	**(01)**	Report
byte1	**(01)**	Address	–	

Performs automatic setting of slave addresses. Only one slave will receive this command, and which receives it depends on the state of the sense lines. For slaves, *address* should start at `01` and increase in order. The master is always located at address `00`, and broadcast is always at address `FF`. If an error is detected on the slave's side, a pre-determined error report is returned.

4.3.3 Command: Comm. Method Change (COMMCHG)

Command name	Request Data		Response Data	
Comm. Method Change	**F2, method code**		–	
byte0	**F2**	Command Code	–	
byte1	**(01)**	Method Code	–	

Changes the master and slave communication methods, such as to increase communication speed. This is a broadcast packet, so it is necessary to check that all I/O units are capable of the command sent. Whether this is supported can be determined by reading each I/O unit's *JVS Revision*.

4.3.4 Command: I/O Identify (IOIDENT)

Command name	Request Data		Response Data	
I/O Identify	**10**		**01 (report), "ID CODE...",00**	
byte0	**10**	Command Code	**(01)**	Report
byte1	–		**4E**	N's ASCII code
...	–		**...**	...
byteN	–		**00**	End code

ID code is no more than 100 characters in ASCII format. Contains this information, in order: Maker name, I/O board code, software version number, and details. A zero byte must follow the string to denote the ending. Each field is separated by semicolons. The contents of each field are not particularly specified.

EXAMPLE: `"NAMCO LTD.;I/O PCB-1000;ver1.0;for domestic only,no analog input"`

4.3.5 Command: Command Revision (CMDREV)

Command name	Request Data		Response Data	
Command Revision	**11**		**01 (report), revision**	
byte0	**11**	Command Code	**(01)**	Report
byte1	–		**(13)**	Revision code

Reads the revision of the command format supported by the I/O board. The code is written in BCD, with the low 4 bits being the decimal value.

The current revision is REV1.3, so the value would be `13`.

4.3.6 Command: JVS Revision (JVSREV)

Command name	Request Data		Response Data	
Command Revision	**12**		**01 (report), revision**	
byte0	**12**	Command Code	**(01)**	Report
byte1	–		**(30)**	Revision code

Reads the revision of the JVS supported by the I/O board. The code is written in BCD, with the low 4 bits being the decimal value.

The current revision is REV3.0, so the value would be `30`.

4.3.7 Command: Communications Version (COMMVER)

Command name	Request Data		Response Data	
Comm. Version	**13**		**01 (report), version**	
byte0	**13**	Command Code	**(01)**	Report
byte1	–		**(10)**	Version code

Reads the version of the communication system supported by the I/O board. The code is written in BCD, with the low 4 bits being the decimal value.

The current version is VER1.0, so the value would be `10`.

4.3.8 Command: Feature Check (FEATCHK)

Command name	Request Data		Response Data	
Feature Check	**14**		**01 (report), func code, ..., 00**	
byte0	**14**	Command Code	**(01)**	Report
byte1	–		**(01)**	(Switch input function)
byte2	–		**(02)**	(Switch input function, parameter 1)
byte3	–		**(08)**	(Switch input function, parameter 2)
byte4	–		**(00)**	(Switch input function, parameter 3)
...	–		**...**	... (Next function)
byteN	–		**00**	End code

List the features supported by the I/O board. Each function (except for end code) is 4 bytes, consisting of one function code byte and 3 parameter bytes.

Table 8: Function Code Listing

Function	**Code**	**Prm.1**	**Prm.2**	**Prm.3**	**Parameter description**
End code	00	–	–	–	End function list
INPUT FUNCTIONS					
Switch input	01	*players*	*buttons*	0	Number of players, and how many switches per player. For example, 1L6B + START + SERVICE is 12 switches.
Coin input	02	*slots*	0	0	Number of coin slots
Analog input	03	*channels*	*bits*	0	Number of analog channels, number of effective bits (0 for unknown)
Rotary input	04	*channels*	0	0	Number of rotary encoders
Keycode input	05	0	0	0	?
Screen position input	06	*Xbits*	*Ybits*	*channels*	Number of effective bits for X and Y axis, number of channels
Misc. switch input	07	*SW MSB*	*SW LSB*	0	Number of extra switches (big endian across Prm.1 and Prm.2)
OUTPUT FUNCTIONS					
Card system	10	*slots*	0	0	Number of card slots
Medal hopper	11	*channels*	0	0	Number of medal hoppers
General-purpose output	12	*slots*	0	0	Number of outputs
Analog output	13	*channels*	0	0	Number of analog output channels
Character output	14	*width*	*height*	*type*	Output display resolution in characters, and character code type
Backup	15	0	0	0	Has backup data for data such as number of coins

Table 9: Character Output Type Listing

Code	**Output Type**
00	Unknown
01	ASCII (numeric)
02	ASCII (alphanumeric)
03	ASCII (alphanumeric, half-width katakana)
04	ASCII (kanji support, SHIFT-JIS)

4.3.9 Command: Main ID (MAINID)

Command name	Request Data		Response Data	
Main ID	**15**, "ID CODE...",00		**01 (report)**	
byte0	**15**	Command Code	**(01)**	Report
byte1	**4E**	N's ASCII code	–	
...	**...**	...	–	
byteN	**00**	End code	–	

Sends the Main PCB's ID code to the I/O board. ID code is no more than 100 characters in ASCII format. Contains this information, in order: Maker name, game name, software version number, and details. A zero byte must follow the string to denote the ending. Each field is separated by semicolons. The contents of each field are not particularly specified.

EXAMPLE: `"NAMCO LTD.;TEKKEN2;ver1.6; TEKKEN2 ver B"`

4.3.10 Command: Switch Inputs (SWINP)

Command name	Request Data		Response Data	
Switch Inputs	**20, players, bytes**		**01 (report), data, ...**	
byte0	**20**	Command Code	**(01)**	Report
byte1	**(02)**	Player count	**(00)**	Test and other system data
byte2	**(02)**	Data bytes	**(02)**	Player 1 switch data, first byte
...	–		**...**	...
byteN	–		**(00)**	Last player switch data, last byte

Reads the Switch input data. Player count specifies how many players to read the data from, and data byte count specifies how many data bytes are to be read. The total number of bytes in the request is $(players \times bytes) + 1$. Each bit is set to 1 when ON, 0 when OFF. 0 is returned for a non-existent switch.

Table 10: Switch Input Data Format

	bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
byte0	TEST	TILT1	TILT2	TILT3	*Unused*			
byte1	1P start	1P service	1P up	1P down	1P left	1P right	1P push1	1P push2
byte2	1P push3	1P push4	1P push5	1P push6	1P push7	1P push8	–	–
...	...							
byteN	2P start	2P service	2P up	2P down	2P left	2P right	2P push1	2P push2
byteN+1	2P push3	2P push4	2P push5	2P push6	2P push7	2P push8	–	–

Table 11: Switch Input Data Format (2 Joysticks)

	bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
byte1	1P start	1P service	1P up	1P/L down	1P/L left	1P/L right	1P/R up	1P/R down
byte2	1P/R left	1P/R right	1P push1	1P push2	1P push3	1P push4	–	–

Table 12: Switch Input Data Format (Mahjong Panel)

	bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
byte1	start	service	A	B	C	D	E	F
byte2	G	H	I	J	K	L	M	N
byte3	Kan	Reach	Ron	Bet	Chi	Pon	Big	Small
byte4	Take Score	Last Chance	Flip Flop	–	–	–	–	–

4.3.11 Command: Coin Inputs (COININP)

Command name	Request Data		Response Data	
Coin Inputs	**21, slot**		**01 (report), data, ...**	
byte0	**21**	Command Code	**(01)**	Report
byte1	**(02)**	Slot count	**(00)**	Slot 1 condition, coin count MSB
byte2	–		**(02)**	Slot 1 coin LSB
...	–		**...**	...
byteN	–		**(00)**	Last slot coin LSB

Read the number of coins remaining on the I/O board. Slot count specifies how many coin slots to read the data from. Each slot has two bytes of data. The top two bits contain the slot state, and the remaining 14 bits contain the coin count (up to 16383 coins). Prepaid card systems will also use this command. When using both coins and a card system, the slots are assigned to coins first, then card system.

Table 13: Coin Input Data Format

	bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
byte0	Slot 1 Condition		Slot 1 Coin MSB					
byte1	Slot 1 Coin LSB							
byte2	Slot 2 Condition		Slot 2 Coin MSB					
byte3	Slot 2 Coin LSB							
...	...							

Table 14: Coin Condition Code List

Code	**Description**
00	Normal
01	Coin jam (coin switch is held)
02	Counter disconnected
03	Busy (such as card system processing)

4.3.12 Command: Analog Inputs (ANLINP)

Command name	Request Data		Response Data	
Analog Inputs	**22, channel**		**01 (report), data, ...**	
byte0	**22**	Command Code	**(01)**	Report
byte1	**(02)**	Channel count	**(00)**	Channel 1 MSB
byte2	–		**(02)**	Channel 1 LSB
...	–		**...**	...
byteN	–		**(00)**	Last channel LSB

Reads analog inputs. Analog data is always returned as 16 bits, with the valid bits being moved to the top, and the remainder set to 0. In this way, main board software does not need to know the precision of the analog data.

For an analog joystick, the midpoint should be as close as possible to `0x8000` (I/O board should take care of sanitizing values).

Table 15: 10-bit Analog Input Data Format

	bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
byte0	Channel 1 analog bits 9–2							
byte1	Channel 1 analog bits 1–0		0					
byte2	Channel 2 analog bits 9–2							
byte3	Channel 2 analog bits 1–0		0					
...	...							

4.3.13 Command: Rotary Inputs (ROTINP)

Command name	Request Data		Response Data	
Rotary Inputs	**23, channel**		**01 (report), data, ...**	
byte0	**23**	Command Code	**(01)**	Report
byte1	**(02)**	Channel count	**(00)**	Channel 1 MSB
byte2	–		**(02)**	Channel 1 LSB
...	–		**...**	...
byteN	–		**(00)**	Last channel LSB

Reads rotary encoder inputs, such as mouse data. The initial data is 0, and has 16 bits of data. As the encoder ticks, the accumulator is incremented or decremented. The returned data is the value of this encoder accumulator. Right and up movement is increment, left and down movement is decrement.

4.3.14 Command: Keycode Inputs (KEYINP)

Command name	Request Data		Response Data	
Keycode Inputs	**24, channel**		**01 (report), keycode**	
byte0	**24**	Command Code	**(01)**	Report
byte1	–		**(31)**	Key code

Reads the state of a key array such as a keyboard. Data is always one byte. When code is `00`, there is no data. Bit 7 is assigned to a held key such as shift, and the rest of the byte is used for scancode data.

4.3.15 Command: Screen Position Inputs (SCRPOSINP)

Command name	Request Data		Response Data	
Screen Pos. Inputs	**25, channel**		**01 (report), position(X,MSB), position(X,LSB), position(Y,MSB), position(Y,LSB)**	
byte0	**25**	Command Code	**(01)**	Report
byte1	**(01)**	Channel index	**(00)**	Position(X,MSB)
byte2	–		**(00)**	Position(X,LSB)
byte3	–		**(00)**	Position(Y,MSB)
byte4	–		**(00)**	Position(Y,LSB)

Reads the state of a screen position input, such as a light gun or touch-panel. `(0,0)` is bottom left, and `(0xFFFF,0xFFFF)` is top right. Each channel is a single set of XY, and channel indices start at `01`.

4.3.16 Command: Misc. Switch Inputs (MISCSWINP)

Command name	Request Data		Response Data	
Misc. Switch Inputs	**26, bytes**		**01 (report), data, ...**	
byte0	**26**	Command Code	**(01)**	Report
byte1	**(02)**	Data bytes	**(02)**	Switch data
...	–		**...**	...
byteN	–		**(00)**	Last switch data

Reads switches unrelated to player input. For the first byte, the 7th bit corresponds to SW1, 6th to SW2, etc. Second byte has 7th bit corresponding to SW9, 6th to SW10, etc.

4.3.17 Command: Remaining Payout (PAYCNT)

Command name	Request Data		Response Data	
Remaining Payout	**2E, channel**		**01 (report), status, remain(hi), remain(mid), remain(lo)**	
byte0	**2E**	Command Code	**(01)**	Report
byte1	**(01)**	Channel index	**(00)**	Hopper status
byte2	–		**(00)**	Remaining (hi)
byte3	–		**(00)**	Remaining (mid)
byte4	–		**(00)**	Remaining (lo)

Reads the remaining number of medals in the specified hopper. The remaining count is 24 bits. Channel indices start at `01`.

Table 16: Coin Condition Code List

Bit	Description
`7`	Busy
`6`	Medal jam
`5`	Medals low
`4`	Medals empty
`3–0`	–

4.3.18 Command: Data Retransmit (RETRANSMIT)

Command name	Request Data		Response Data	
Data Retransmit	**2F**		**Last response**	
byte0	**2F**	Command Code	**...**	

If the master detects a checksum error, it will send this command to request a retransmission of the last acknowledge. The retransmission must include the entire packet, including sync.

4.3.19 Command: Coin Counter Decrease (COINDEC)

Command name	Request Data		Response Data	
Coin Counter Dec.	**30, slot, amount**		**01 (report)**	
byte0	**30**	Command Code	**(01)**	Report
byte1	**(02)**	Slot index	–	
byte2	**(00)**	Amount (MSB)	–	
byte3	**(00)**	Amount (LSB)	–	

Sent to decrement the number of coins stored. The amount to subtract is 16-bit.
For slot designation when using card system and coin slots together, see *coin input command*.

4.3.20 Command: Payout Counter Increase (PAYINC)

Command name	Request Data		Response Data	
Payout Counter Inc.	**31, slot, amount**		**01 (report)**	
byte0	**31**	Command Code	**(01)**	Report
byte1	**(02)**	Slot index	–	
byte2	**(00)**	Amount (MSB)	–	
byte3	**(00)**	Amount (LSB)	–	

Sent to increase the payout counter. The amount to add is 16-bit.

4.3.21 Command: Generic Output 1 (OUTPUT1)

Command name	Request Data		Response Data	
Generic Output 1	**32, bytes, data, ...**		**01 (report)**	
byte0	**32**	Command Code	**(01)**	Report
byte1	**(02)**	Byte count	–	
byte2	**(18)**	First byte	–	
...	**...**	...	–	
byteN	**(4A)**	Last byte	–	

Output data in a set to the generic output drivers. Each bit in each byte corresponds to a single output, with bit 7 of the first byte being the first output. Assuming the output is open collector or drain, when logic is 1, current is drained (output voltage 0V).

4.3.22 Command: Generic Output 2 (OUTPUT2)

Command name	Request Data		Response Data	
Generic Output 2	**37, index, data**		**01 (report)**	
byte0	**37**	Command Code	**(01)**	Report
byte1	**(03)**	Byte index	–	
byte2	**(18)**	Output data	–	

Output one byte of data to the generic output drivers. Each bit in corresponds to a single output, with bit 7 of being the first output. The byte index allows addressing 8 outputs at a time, where `01` is the first set of 8 outputs.

4.3.23 Command: Generic Output 3 (OUTPUT3)

Command name	Request Data		Response Data	
Generic Output 3	**38, index, data**		**01 (report)**	
byte0	**38**	Command Code	**(01)**	Report
byte1	**(03)**	Bit index	–	
byte2	**(00)**	Output data	–	

Outputs one bit to the generic output drivers. If the output data is `00` or `01`, that value is output; if the data is `02`, the output value is inverted.

4.3.24 Command: Analog Output (ANLOUT)

Command name	Request Data		Response Data	
Analog Outputs	**33, channel, data, ...**		**01 (report)**	
byte0	**33**	Command Code	**(01)**	Report
byte1	**(02)**	Channel count	–	
byte2	**(18)**	Channel 1 MSB	–	
byte3	**(01)**	Channel 1 LSB	–	
...	**...**	...	–	
byteN	**(4A)**	Last Channel LSB	–	

Outputs data to analog outputs. Analog data is always specified as 16 bits, with the valid bits being moved to the top, and the remainder set to 0. In this way, main board software does not need to know the precision of the analog data.

Table 17: 10-bit Analog Output Data Format

	bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
byte0	Channel 1 analog bits 9–2							
byte1	Channel 1 analog bits 1–0		0					
byte2	Channel 2 analog bits 9–2							
byte3	Channel 2 analog bits 1–0		0					
...	...							

4.3.25 Command: Character Output (CHAROUT)

Command name	Request Data		Response Data	
Analog Outputs	**34, count, data, ...**		**01 (report)**	
byte0	**34**	Command Code	**(01)**	Report
byte1	**(03)**	Character count	–	
byte2	**(41)**	First character	–	
...	**...**	...	–	
byteN	**(4A)**	Last character	–	

Outputs text to the character output. Text is encoded in ASCII, for kanji use SHIFT-JIS. The following control codes are supported.

Table 18: Character Output Control Codes

Name	**Mnemonic**	**Code**	**Description**
Carriage Return	`CR`	`0D`	Return display position to beginning of line
Line Feed	`LF`	`0A`	One row line feed
Form Feed	`FF`	`0C`	Page break (full erase)
Backspace	`BS`	`08`	Backspace one character

4.3.26 Command: Coin Counter Increase (COININC)

Command name	Request Data		Response Data	
Coin Counter Inc.	**35, slot, amount**		**01 (report)**	
byte0	**35**	Command Code	**(01)**	Report
byte1	**(02)**	Slot index	–	
byte2	**(00)**	Amount (MSB)	–	
byte3	**(00)**	Amount (LSB)	–	

Sent to increase the number of coins stored. The amount to add is 16-bit.
For slot designation when using card system and coin slots together, see *coin input command*.

4.3.27 Command: Payout Counter Decrease (PAYDEC)

Command name	Request Data		Response Data	
Payout Counter Dec.	**36, slot, amount**		**01 (report)**	
byte0	**36**	Command Code	**(01)**	Report
byte1	**(02)**	Slot index	–	
byte2	**(00)**	Amount (MSB)	–	
byte3	**(00)**	Amount (LSB)	–	

Sent to decrease the payout counter. The amount to subtract is 16-bit.

4.3.28 Command: Taito TypeX Machine Reset (TTXRESET)

Command name	Request Data		Response Data	
Machine Reset	**00**		**01 (report)**	
byte0	**00**	Command Code	**(01)**	Report

Resets the Taito TypeX machine.

4.3.29 Command: Taito TypeX DIP Switches (TTXDIP)

Command name	Request Data		Response Data	
DIP Switches	**01, ??**		**01 (report), ??**	
byte0	**01**	Command Code	**(01)**	Report
byte1	**(01)**	?? Always 1?	**(01)**	DIP Switches

Gets the DIP switch state from the TTX I/O board.

4.3.30 Command: Taito TypeX Unk02 (TTXUNK2)

Command name	Request Data		Response Data	
Unknown 02	**02, ?**		**01 (report), ??**	
byte0	**02**	Command Code	**(01)**	Report
byte1	**??**	??	**(52)**	Always 52?
byte2	**??**	??	–	

Unknown.

4.3.31 Command: Taito TypeX Unk03 (TTXUNK3)

Command name	Request Data		Response Data	
Unknown 03	**03, ?**		**01 (report)**	
byte0	**03**	Command Code	**(01)**	Report
byte1	**??**	??	–	
byte2	**??**	??	–	

Unknown.

4.3.32 Command: Taito TypeX Watchdog? 04 (TTXWDT4)

Command name	Request Data		Response Data	
Watchdog? 04	**04**		**01 (report)**	
byte0	**04**	Command Code	**(01)**	Report

Unknown.

4.3.33 Command: Taito TypeX Watchdog Set (TTXWDTSET)

Command name	Request Data		Response Data	
Watchdog Set	**05, time**		**01 (report)**	
byte0	**05**	Command Code	**(01)**	Report
byte1	**(0B)**	Period (MSB)	–	
byte2	**(B8)**	Period (LSB)	–	

Sets the watchdog timer period. Period is in 100ths of a second.

4.3.34 Command: Taito TypeX Unk06 (TTXUNK6)

Command name	Request Data		Response Data	
Unknown 06	**06, ?**		**01 (report), ??**	
byte0	**06**	Command Code	**(01)**	Report
byte1	**??**	??	**??**	??
byte2	**??**	??	–	

Unknown.

4.3.35 Command: Taito TypeX Unk07 (TTXUNK7)

Command name	Request Data		Response Data	
Unknown 07	**07, ?**		**01 (report)**	
byte0	**07**	Command Code	**(01)**	Report
byte1	**??**	??	–	
byte2	**??**	??	–	

Unknown.

4.3.36 Command: Taito TypeX Watchdog Kick (TTXWDTKICK)

Command name	Request Data		Response Data	
Watchdog Kick	**08**		**01 (report)**	
byte0	**08**	Command Code	**(01)**	Report
byte1	–		**(??)**	?

TypeX Watchdog kick

4.3.37 Command: NAMCO Extended NOP (NAMCOEXTNOP)

Command name	Request Data		Response Data	
NAMCOEXTNOP	**70, cmd**		**01 (report)**	
byte0	**70**	Command Code	**(01)**	Report
byte1	**(00)**	Command Code	–	

No-operation. Most unimplemented NAMCO extended command codes act as this.

4.3.38 Command: NAMCO Extended Read (NAMCOEXTREAD)

Command name	Request Data		Response Data	
NAMCOEXTREAD	**70, 01, address**		**01 (report), data**	
byte0	**70**	Command Code	**(01)**	Report
byte1	**01**	Command Code	**(00)**	Data byte 1
byte2	**(C0)**	Address MSB	**(00)**	Data byte 2
byte3	**(20)**	Address LSB	**(00)**	Data byte 3
...	–		**...**	...
byte8	–		**(00)**	Data byte 8

Reads 8 bytes out of the I/O unit's memory.

4.3.39 Command: NAMCO Extended ID? (NAMCOEXTID)

Command name	Request Data		Response Data	
NAMCOEXTID	**70, 02**		**01 (report), data**	
byte0	**70**	Command Code	**(01)**	Report
byte1	**02**	Command Code	**(19)**	Data byte 1
byte2	–		**(97)**	Data byte 2
byte3	–		**(03)**	Data byte 3
byte4	–		**(05)**	Data byte 4
byte5	–		**(03)**	Data byte 5
byte6	–		**(19)**	Data byte 6
byte7	–		**(35)**	Data byte 7
byte8	–		**(29)**	Data byte 8

Reads 8 bytes of fixed memory out of the I/O unit. Data read depends on the I/O unit?

4.3.40 Command: NAMCO Extended Cmd 03h (NAMCOEXT03)

Command name	Request Data		Response Data	
NAMCOEXT03	**70, 03**		**01 (report), data**	
byte0	**70**	Command Code	**(01)**	Report
byte1	**03**	Command Code	**(19)**	Data byte

Reads byte from I/O unit memory 0xC00F. (what does this do?)

4.3.41 Command: NAMCO Extended Cmd 04h (NAMCOEXT04)

Command name	Request Data		Response Data	
NAMCOEXT04	**70, 04**		**01 (report), data**	
byte0	**70**	Command Code	**(01)**	Report
byte1	**04**	Command Code	**(FF)**	Data byte 1
byte2	–		**(FF)**	Data byte 2

Some devices implement this, some return FF FF.

4.3.42 Command: NAMCO Extended Cmd 22h (NAMCOEXT22)

Command name	Request Data		Response Data	
NAMCOEXT22	**70, 22**		**01 (report)**	
byte0	**70**	Command Code	**(01)**	Report
byte1	**22**	Command Code	–	
byte2	**??**	??	–	
byte3	**??**	??	–	
byte4	**??**	??	–	
byte5	**??**	??	–	
byte6	**??**	??	–	

CyberLead does not even read the inputs, just skips the bytes.

4.3.43 Command: CyberLead LED NOP (CYBERLEDNOP)

Command name	Request Data		Response Data	
CYBERLEDNOP	**71, cmd**		**01 (report)**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**(00)**	Command Code	–	

No-operation. Unimplemented commands do this.

4.3.44 Command: CyberLead LED Set Scene (CYBERLEDSCENE)

Command name	Request Data		Response Data	
CYBERLED01	**71, 01, data**		**01 (report)**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**01**	Command Code	–	
byte2	**(01)**	Data byte 1	–	
...	**...**	...	–	
byte9	**(01)**	Data byte 8	–	

Seems to write 'scene' information.
Writes 8 bytes to 0xC3A0-0xC3A8 in the I/O memory. This will get sent to the LED I/O unit soon after.

4.3.45 Command: CyberLead LED Data Check (CYBERLEDCHECK)

Command name	Request Data		Response Data	
CYBERLED02	**71, 02, data**		**01 (report), info**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**02**	Command Code	**(00)**	Info
byte2	**(01)**	Data cookie	–	
byte3	**(54)**	Data byte 1	–	
...	**...**	...	–	
byte10	**(20)**	Data byte 8	–	

Cookie appears to always be 1. The data bytes are an identifier for the LED data. If this is the same as the currently uploaded LED data, 1 is returned through info. If they do not match, 2 is returned (always?). If 0 is returned, try the command again.

Info seems to be the currently-running cookie if one was already running? Cookie is ignored by LED unit. LED compares data bytes to cached values, and sets led02needsrefresh to 1 if they're different.

4.3.46 Command: CyberLead LED Cmd 04h (CYBERLED04)

Command name	Request Data		Response Data	
CYBERLED04	**71, 04**		**01 (report)**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**04**	Command Code	–	

LED unit sets led02needsrefresh to 2.

4.3.47 Command: CyberLead LED Data Upload (CYBERLEDUPLOAD)

Command name	Request Data		Response Data	
CYBERLED05	**71, 05, cookie, data**		**01 (report), info**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**05**	Command Code	**(00)**	Info
byte2	**(02)**	Data cookie?	–	
byte3	**(5E)**	LED data plane	–	
byte4	**(5E)**	LED data bank	–	
byte5	**(5E)**	LED data page	–	
byte6	**(5E)**	LED data byte 1	–	
...	**...**	...	–	
byte69	**(5E)**	LED data byte 64	–	
byte70	**(5E)**	Data hash MSB	–	
byte71	**(5E)**	Data hash LSB	–	

Table 19: Data Bank Restrictions

Bank Number	First Usable Page
00	3E
01	28
02	18
03	00
04	*none*
05	*none*
06	00
...	00
0F	00

Bank must be `0F` or less, and page must be `3F` or less.

Hash is computed from all the data bytes except the data hash and command codes. Hash is computed with this algorithm:

hash = (cookie shl 8) or cookie for byte in data, hash = (hash + (byte shl 8)) xor byte

Info's bit6 is set on the first upload command (always?). If info's bit0 is set, command succeeded, otherwise retry this command.

4.3.48 Command: CyberLead LED Cmd 06h (CYBERLED06)

Command name	Request Data		Response Data	
CYBERLED06	**71, 06, cookie**		**01 (report), info**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**06**	Command Code	**(00)**	Info
byte2	**(02)**	Data cookie?	–	

Sent after each entire LED plane is uploaded. If info returns 1, command succeeded. If info returns 0, retry this command.

Seems very related to Cmd 02h, no idea what it does. Sends to LED unit, then sends 07h to LED unit.

Cmd06 sets a variable to 67h, Cmd07 returns 00 if that variable is non-zero. Variable is checked for 67h and cleared periodically???

4.3.49 Command: CyberLead LED Upload End (CYBERLEDFINISH)

Command name	Request Data		Response Data	
CYBERLED08	**71, 08**		**01 (report)**	
byte0	**71**	Command Code	**(01)**	Report
byte1	**08**	Command Code	–	

Finishes uploading LED data.

LED unit sets led02needsrefresh to 1.