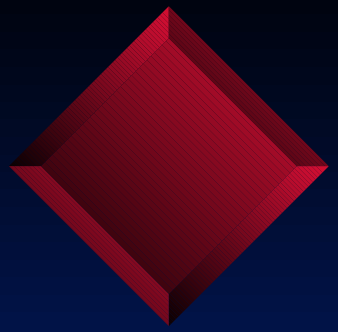


SIO

Link Cable

Memory Card

Peripherals

Dan Burnash March 1996

PlayStation™

Link Cable - LIBCOMB

- Are you connected?
  - Use _comb_control w/o AddComb
  - _comb_control
  - (1, 1, COMB_BIT_DTR|COMB_BIT_RTS);
    - Sets DTR and RTS
    - Other system can check DSR and CTS
    - Good method for determining if systems are ready to link

Initialization

- AddComb
  - Load it once, and leave it installed
- Open and enable events
  - read complete event
  - write complete event
  - error event

Find the Master

- It is important in most games to determine which system is in control
  - Assert and negate lines
  - Pass data
  - Other?

Transferring Data

- Use standard read and write
  - Send 8 bytes at a time
  - Issue the read before writing data
  - Don’t write when waiting to receive data
  - read / write automatically sets lines
  - Limited by interrupt handler, not baud rate

Error Recovery

- Assume errors will occur
  - Cancel the current read
    - Use _comb_control (2, 3, 0)
  - Reset the system
    - Use _comb_control(2, 0, 0)
  - Resend the data or do without it

Terminating the link

- DelComb
  - Do not use it
  - Reloading is not allowed in the current version
  - Reboot required to reenable link after calling DelComb()

Creating a Stable Linked Game

- Make your game deterministic
  - Random number generators may not work
- Passing control pad information
  - Make sure you verify your data
  - If an error occurs, you must resend
- Passing position information
  - Easier to recover when data is lost
  - Guess when you lose data, w/o resending

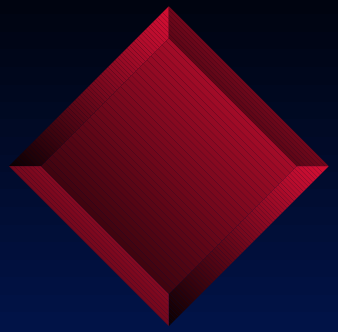
Dan Burnash March 1996

PlayStation™

More to Consider

- Passing critical information
  - Send it twice
- Getting out of sync
  - Machines may run at different speeds
  - Matching frame rates is no guarantee
  - Assume that over time, the machine will get out of sync, so plan for it.
    - Skip a frame to catch up
    - Structure your code so it does not matter

TM



Link Cable
The Ring Buffer system

- The ring buffer concept
  - 1024 byte buffer
  - Data transferred is put into it automatically
  - The program reads the data from the buffer
  - Retrieve data before the buffer overflows

PlayStation™

Initialization

- void comb_open(int baud);
  - Initialize and enable the interrupt
  - Set the baud rate
  - May be called repeatedly to set baud rate

Setting/Checking the Lines

- Get the status
  - int comb_get_stat();
- Get the line status
  - int comb_get_dsr();
  - int comb_get_cts();
- Set the lines
  - void comb_set_dtr(int status);
  - void comb_set_rts(int status);

Writing Data

- Sending data using polled output
  - Routines waits for okay to send, then transmit the data
- comb_write_char(unsigned char data);
- comb_write_int(unsigned long data);

Reading Data

- Just retreives data from the ring buffer
- int comb_read_char();
- Return value
  - 1 byte of data
  - -1 when no more data is present

Error Types

- Parity Error
- Overrun Error
- Framing Error
- Receive buffer full

Error Detection/Correction

- int comb_get_errors();
  - Returns error type, or zero
- int comb_clear_errors();
  - Returns error type, or zero
  - Clears the error
- void comb_flush();
  - Empties the ring buffer
  - Aborts transfers in progress

Memory Card

- Overview
  - 120k usable memory on card
  - 8k Blocksize used on card
  - Permanent storage for multiple games
  - Find a new way to use the card
    - Bonus levels, new characters, special codes, extra features

PlayStation™

Initializing the memcard system

- open & enable the memory card events
- InitCARD(); / / init card bios
- StartCARD(); / / starts bios
- _bu_init(); / / inits file system

Testing for the existence of a memory card

- _card_info(channel)
  - check events for information
  - card_event returns the event that occured when using polling event technique
  - possible events are:
    - card exists
    - new card
    - error
    - timeout - no card in slot

Testing the format

- _card_read(channel,0,&buf[0]);
- if(buf[0]=='M' && buf[1]=='C')
  - return(FORMATTED);
- else
  - return(UNFORMATTED);

Formatting a card

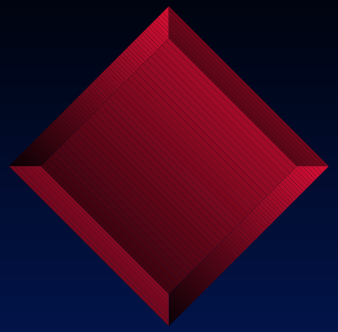
- Prompt the user before a format
  - avoid the possibility of overwriting data
- Use format(device) / / device is:
    - bu00: / / port 1
    - bu10: / / port 2
    - bu00: thru bu03: / / port 1 multi tap cards
    - bu10: thru bu13: / / port 2 multi tap cards
  - bu = backup unit

Finding and Reading Files

- firstfile() and nextfile()
  - Find each filename
- Retreive data from file
  - open(bu##:filename, ..) and read(fd, ..)

Creating a file

- File size
  - Specify when creating file
  - Cannot be changed, except by deleting the file and recreating
- Steps
  - Open the file, specifying size
  - Close file
  - Reopen file for write access



File Header Structure

- typedef struct {
  - char Magic[2]; // always “SC”
  - char Type; // specifies # of icons
  - char BlockEntry; // #of blocks used
  - char Title[64]; // must be in shift jis
  - char reserve[28];
  - char Clut[32]; // 4 bit clut for icons
  - char Icon[3][128]; // 3 icons
- } _CARD;

Title

- The Title
  - Must be in Shift JIS(SJIS) 2 byte char form
    - ROM card reader requires it
    - Don’t submit without it
  - The how to of SJIS
    - Use the converter we provide, or
    - Get the fish book
      - Understanding Japanese Information Processing
      - isbn 1-56592-043-0

Icons

- The ROM card reader displays icons
- The header clut and icon fields
- Put in 3 icons for a rotating image
- The type field determines the number of icons that will be used

PlayStation™

The Filename

- All filenames start with:
  - BA - for U.S. market
  - BI - for Japanese market
  - BE - for European market
- The 10 digit product code comes next
- Add to it for multiple unique filenames
- Up to 21 ascii characters in length

PlayStation™

Checksum your data

- Verify your data
  - Use a checksum to assure it is correct
  - Try your own method, just do it

Writing data

- write()
  - Writes data in 128 byte, or a multiple of 128 byte, blocks
  - If it is not a multiple of 128, the write will fail

The MultiTap

- Make a great 4, 8, or 16 player game
- Link with Libtap
- Identifying the “tap”
- Retrieving data from each controller
- Limitations of the interface

Libtap

- Libtap has new pad functions
  - InitTAP(...) / / replaces InitPAD
  - StartTAP() / / replaces StartPAD
- These two functions take the same args as their counterparts

Reading the pad data

- char not_present_flag
  - 1 if port empty
- char id
  - id of a multi tap is 0x80
- data
  - depends on controller type

Data for a multi tap

- The multi tap has data for 4 controllers
- Each controller has:
  - char not_present_flag
  - char id
  - controller data
- For a standard controller, it would be 2 bytes of digital data

Structure for std controller and Multi tap

- typedef struct {
  - char_not_present_flag;
  - char id;
  - u_short pad_data;
- } std_con_type;
- typedef struct {
  - char not_present_flag; / / 0
  - char id; / / 0x80
  - std_con_type std_data[4];
- } controller data;

Analog Joystick

- char not_present_flag
- char id / / 0x53 for analog joystick
- u_short data; / / digital button data
- char analog_1;
- char analog_2;
- char analog_3;
- char analog_4;

Peripherals

- Each Controller has a unique id and unique data
  - Analog Joystick
  - NegCon
  - Light Gun - coming soon

PlayStation™