CASIO FZ DATA TRANSFERS

Jeff McClintock (jeffm@iconz.co.nz)

This text describes the structure of FZ dump data and dump procedure.
If you find errors, please let me know.

Overview.

A PC can send and receive (dump) data from an FZ sampling synthesizer. The dump contains the raw sample data and the sampler's 'housekeeping' information.

Dumps contain up to 5 types of information:

1.A header, This describes the contents and size of the data to follow. The format of the header depends on the situation. MIDI dumps have a different header than port dumps.
2.Bank data, Contains parameters of one or more banks. e.g. which voices comprise a bank.
3.Voice data, Contains the parameters of one or more voices, e.g. envelope settings.
4.Wave data, Actual 16 bit raw sample data.
5.Effect data, contains any FZ settings not relating directly to a voice or bank. e.g. master volume, foot pedal settings. Effect data starts 960 bytes into a block.

Data is always sent in BLOCKS. Blocks are 1024 bytes in size. Each block can only contain 1 type of data (bank, voice, wave etc.). Extra information (block headers, checksum) may be 'wrapped' around each block, but is discarded as each block is received.

Example

A simple VOICE DUMP, 1 voice. All blocks are the same size except the header. The actual sample data is spread across several WAVE blocks. The last block is padded out to 1024 bytes size. Blocks are sent in this order.

[HEADER]
[VOICE BLOCK]
[WAVE BLOCK]
[WAVE BLOCK]
[WAVE BLOCK]
[WAVE BLOCK] 'total of 4096 bytes sample data (2048 samples)

A full dump

[HEADER]
[BANK BLOCK] ' 1st bank
[BANK BLOCK] ' 2nd bank
[VOICE BLOCK] ' 4 voices
[VOICE BLOCK] ' 4 more voices
[WAVE BLOCK]
[WAVE BLOCK]
[WAVE BLOCK]
[WAVE BLOCK]
[WAVE BLOCK] 'sample data. 8 voices worth. Concatenated

an effect dump
[HEADER]
[EFFECT BLOCK] Effect data starts 960 bytes into the block.

Block Structure

BANK DATA

Each bank uses 656 bytes of data leaving the remainder of the block unused.
This is the data accessed by [EDIT BANK] on the FZ.
Here is the layout of the data in a 'C' language format. Beware, your C compiler may have different size 'short', 'int', or 'long' data types.

```
#define MAXV 64                 // Maximum number of voices

unsigned int   bstep;           // 0-64 Number of voices used in the bank
unsigned short hwid[MAXV];      // Hi-Note settings (MIDI note number)
unsigned short lwid[MAXV];      // Lo-Note settings (MIDI note number)
unsigned short htc[MAXV];       // 1-127 Velocity split hi
unsigned short ltch[MAXV];      // 1-127 Velocity split lo
unsigned short cent[MAXV];      // Original note (MIDI note number)
unsigned short mchn[MAXV];      // 0-15 Recieve MIDI channel
unsigned short gchn[MAXV];      // Each bit represents an 'individual output'
unsigned short bvol[MAXV];      // 1-127 Area volume (Usually 127)
unsigned int   vp[MAXV];        // 0-63 Voice number to use in area
char           name[14];        // Bank name, last two bytes should always be 0
```

What does this all mean?;
short - A binary value stored in 1 byte
int - A binary value stored in 2 bytes
long - A binary value stored in 4 bytes
unsigned- only positive values
char - ASCII codes
[MAXV]- means that there are 64 of each piece of data.
// - preceedes a comment

VOICE DATA

Each voice uses 192 bytes of data. Each VOICE block can contain 1 to 4 voices. When a block contains more than 1 voice the voices are aligned on 256 byte boundaries. eg voice 1 data starts at byte 0 and ends at byte 191, voice 2 data starts at byte 256, and ends at 447.
All parameters default to 0 unless otherwise stated.

```
long    wavst;          // Sample data start address
long    waved;          // Sample data end address
```

```
long    genst;              // Sample play start address
long    gened;              // Sample play end address (usually =waved - 2)
int     loop;               // 0000h No waveform,
                            // 01d7h Normal sound,(default)
                            // 101dh Reversed,
                            // 2014h Cueing sound (played with WHEEL),
                            // 0013h Synthesised waveform.
                            // FZ supports up to 8 loops
short   loop_sus;           // 0-8 sustain loop, 8 = No sustain loop
short   loop_end;           // 0-8 last loop, 8 = execute all loops
long    loopst[8];          // Loop starts, upper 8 bits are loop fine settings (set =gened for unused loop)

long    looped[8];          // Loop ends, MSB = Skip or Trace (set =gened for unused loop)

int     loopxf[8];          // 0-1023 Loop cross fade time
unsigned int    looptm[8];  // 1-1022 Loop time 16 ms to 16 s (default is 64), 1024 = continuous loop,
                                //                                                 1023 = loop during
sustain
int     dcp;                // 0-255 loop pitch correction times 1/256 semitone
short   dcf;                // 0-127 Filter cut off frequency
short   dcq;                // 0-127 Filter resonance (Q) lower 3 bits ignored
                            // The FZ has 8 stage envelopes (rate/level type)
                            // A good default is to set all dca_rate[] to 0xc0, and dcf_rate[] to 0x90
                                        except for dca_rate[0]=0x7f,dcf_rate[0]=0x7f,
                                                dcf_stop[0] to 0xff , dca_stop[0] to 0xff
short   dca_sus;            // 0-7 Sustain section # on DCA envelope
short   dca_end;            // 0-7 End section # on DCA envelope (default to 7)
short   dca_rate[8];        // 0-127 Envelope rates MSB = direction up or down
unsigned short  dca_stop[8];        // 0-255 Envelope section end level
short   dcf_sus;                    // 0-7 Sustain section # on DCF envelope
short   dcf_end;                    // 0-7 End section # on DCF envelope (default to 7)
short   dcf_rate[8];                // 0-127 Envelope rates MSB = direction up or down
unsigned short  dcf_stop[8];        // 0-255 Envelope section end level
unsigned int    lfo_delay;          // 0-65535 LFO Delay time 2 ms steps
unsigned short  lfo_name;           // 0-Sine, 1-Saw up, 2-Saw down, 3-triangle,4-rectangle, 5 random.
MSB=phase sync (default MSB =1)
unsigned short  lfo_atck;   // 1-127
short   lfo_rate;           // 0-127 (default to 0x40)
short   lfo_dcp;            // 0-127 LFO effect on Pitch
short   lfo_dca;            // 0-127 LFO effect on amplitute
short   lfo_dcf;            // 0-127 LFO effect on filter
short   lfo_deq;            // 0-127 LFO effect on filter Q
short   vel_deq_kf;         // -127 to 127 ,velocity effect on filter Q
short   dca_kf;             // DCA amplitude Key follow
short   dca_rs;             // DCA rate Key follow
short   dcf_kf;             // DCF amplitude Key follow
short   dcf_rs;             // DCF rate Key follow
short   vel_dca_kf;         // -127 to 127 Key Velocity effect on DCA envelope amplitude (default to 0x30)
short   vel_dca_rs;         // -127 to 127 Key Velocity effect on DCA envelope rate
short   vel_dcf_kf;         // Velocity effect on filter envelope amplitude
short   vel_dcf_rs;         // Velocity effect on filter envelope rate
unsigned short  hwid;       // Hi note (MIDI note number) (default to 0x60)
unsigned short  lwid;       // Low note (MIDI note number)(default to 0x24)
unsigned short  cent;       // Original note (MIDI note number)(default to 0x48)
unsigned short  samp;       // sampling freq 0 - 36khz, 1 - 18khz, 2 - 9khz
```

```
char    name[14];           // ASCII name. last 2 bytes always 0
```

EFFECT DATA

```
struct effectdata
{
short     bend;             // bender depth
short     mvol;             // master volume
short     suss;             // sustain switch ON/OFF
short     mod_lfp;                  // modulation to   lfo pitch
short     mod_lfa;                  //                 lfa amp
short     mod_lff;          //                lfo filter
short     mod_lfq;                  //                 lfo filter q
short     mod_dcf;          //                filter offset
short     mod_dca;          //                amp offset
short     mod_dcq;          //                filter q offset

short     fot_lfp;          // foot volume to..
short     fot_lfa;          //
short     fot_lff;          //
short     fot_lfq;          //
short     fot_dca;          //
short     fot_;dcf          //
short     fot_;dcq          //

short     aft_lpf;          // after touch to..
short     aft_lfa;          //
short     aft_lff;          //
short     aft_lfq;          //
short     aft_dca;          //
short     aft_dcf;          //
short     aft_dcq;          //
};
```

WAVE DATA

This is simply 16 bit sample data, signed.

PC DATA DUMPS TO AN FZ

Overview

1 - PC sends 'REMOTE CODE' (prepare for transfer), can be LOAD, SAVE, MERGE, VERIFY
2 - PC waits 1 second
3 - PC Sends 'OPEN CODE' This dump header describes the data blocks
3 - PC Sends data blocks

This assumes the FZ is set to [REMOTE MODE], you can skip step 1 and 2 if you use [LOAD VOICE] on the FZ.

When using MIDI, there is a handshaking protocol to ensure all data arrives ok. The FZ acknowledges each packet of data. With the high speed port, there is only low-level hardware handshaking.

header detail

High Speed Port

REMOTE CODE (PORT)
17 bytes of data
[7F][][][][][][eb][ev][sta][mod][][][][][][][sum]

OPEN CODE (PORT)
17 bytes of data
[sta][1L][1H][Bn][Vn][WL][WH][eb][ev][][][][][][][][sum]

[7F] - one byte of data, hexadecimal value 7F (127 decimal)
[] - one byte of data, doesn't matter what (typically 0)
[eb] - The bank number to dump ie 0-7, 0x7f means dump current bank
[ev] - The voice number to dump ie 0-63, 0x7f means dump current voice
[sta] - Type of dump, 0 - FULL, 1 - VOICE, 2 - BANK, 3 - EFFECT
[mod] - 0 - SAVE, 1 - LOAD, 2 - MERGE, 3 - VERIFY
[sum] - Checksum, Add up total of all bytes, logical AND with 0xff (255), subtract from 0x100 (256)
'C' language example: checksum = 256 - (checksum & 255);
[1L][1H]- Total number of blocks to follow, split into high and low bytes
[Bn] - Number of banks
[Vn] - Number of voices
[WL][WH] - Total number of blocks of wave data (samples), split into high and low bytes

When sending an effect dump, Bn, Vn, and WL/WH are all zero (0).

Data is then sent in blocks (1024 bytes) with the addition of a checksum at the end of each block.

MIDI

When transmitting data over MIDI, each block is further divided into packets.Each packet sent to the FZ over MIDI has a SYSEX HEADER, and an end-of-sysex byte.

The Casio FZ MIDI PACKET HEADER
[F0][44][02][00][7n]
7n - the 'n' represents the midi channel. 0=chan 1.

The end of each packet must have [F7]. Each packet of data is acknowledged by a message from the FZ;

REMOTE CODE (MIDI)
11 bytes of data (shown in hexadecimal)
[F0][44][02][00][7n][7F][eb][ev][sta][mod][F7]

OPEN CODE (MIDI)
16 bytes of data
[F0][44][02][00][7n][70][sta][Bn][Vn][W0][W1][W2][W3][eb][ev][F7]

These contains much the same info as the PORT codes. Note that 2-byte values have to be split into 4 nybles (4 bits each).

Each packet of data is then sent, 64 bytes of data at time. Each byte is split into nybles when sent over midi. So 64 bytes of data = 128 bytes over MIDI.

[F0][44][02][00][7n][74][128 bytes of data][checksum][F7]

To calculate checksum. Add up the total of all data bytes, subtract from 256 then logical 'AND' with 127.
Example in 'C'
checksum &= 255;
checksum = (256 - checksum) & 127;

Each time the FZ receives a packet, it replies
[F0][44][02][00][7n][72][F7]
, or if there was an error
[F0][44][02][00][7n][73][F7] (second to last byte changes).

Once the entire dump is complete, the FZ sends a 'Close' command
[F0][44][02][00][7n][71][F7]

Storing FZ data on PC disk

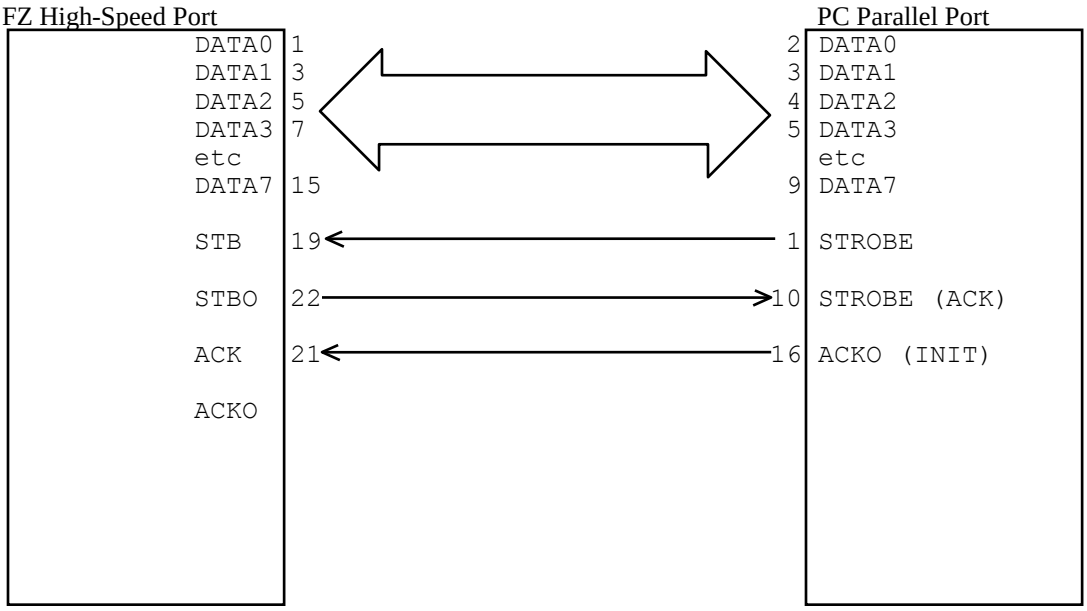
FZDUMP stores data on disk much the same as it is received. The data is stored in blocks of 1024 bytes. The header describing the data is stored starting at 1000 bytes into the file (taking advantage of an unused section of the first block). The file has the extension .FZF (Full dump), .FZB (Bank dump), or .FZV (Voice Dump). The file extension is redundant, as the file header also indicates a full dump (bn > 1), bank dump (bn = 1) or voice dump (bn = 0).

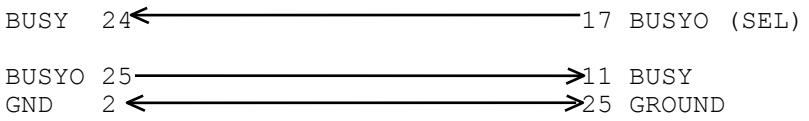
```
struct fz_file_header {
   long          indicator;          /* FZCom file indicator     (always =  476549224)    00 */
   short         version;            /* FZCom file format version no.  (always = 1) 04 */
   unsigned char   status;         /* Type of file (Full:0, Voice:1 Bank:2, ,Effects:3) 05 */
   unsigned char   bn;             /* Number of banks in this file   06 */
   unsigned char   vn;             /* Number of voices in this file  07 */
   unsigned char   dum2;           /* unused? (set to 0)*/
   short         block_count;        /* Number of blocks in the file  (1 block = 1024 bytes)  08 */
   short         wn_block_count;  /* Number of PCM data blocks      0A */
   short      unused;                /* Reserved         (set to 0)           0C */

};
```

Casio FZ to PC High Speed Port - Low Level Details

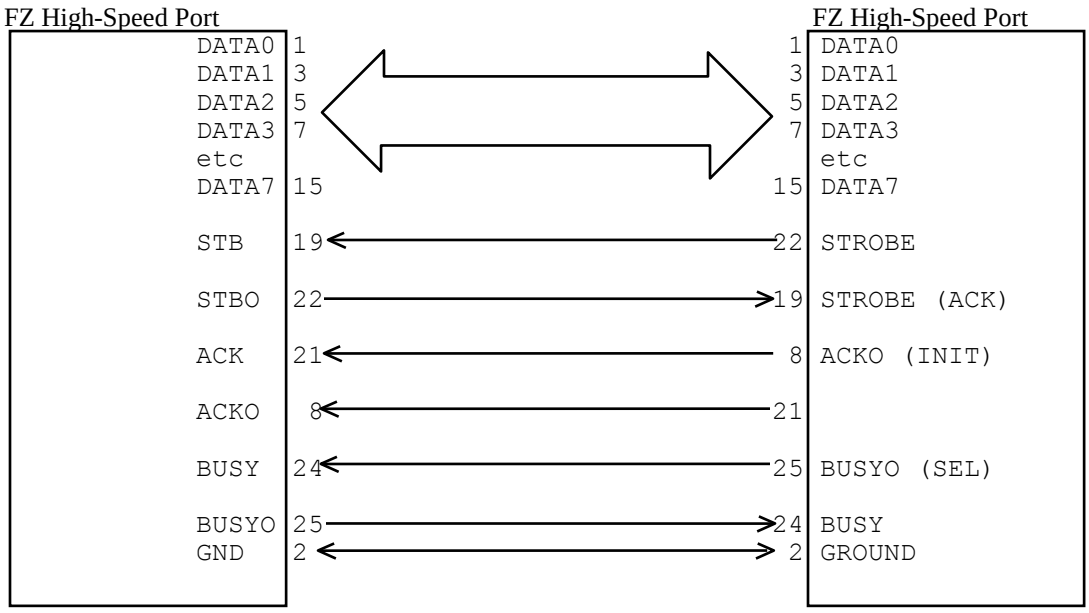
PC and FZ pin-outs

Note: I am using some PC pins for different purposes than intended (standard use in brackets)




BUSY 24
17 BUSYO (SEL)
BUSYO 25
11 BUSY
GND 2
25 GROUND

FZ to FZ cable pin-outs. DO NOT EVER CONNECT THIS TO A PC



For general information on programming the PC parallel port see 'Interfacing the IBM PC Parallel Printer Port - Zhahai Stewart' (Available on the WWW)

Signals.

Signal line	PC Pin	Register bit	Notes
TRISTATE	NA	C4	Sets PC Port to input mode, all data pins enter a high-impedance state. This only works on bi-directional ports. You must not connect an FZ to the PC unless the port is tri-stated. If you do not understand this point, DON'T try it.
BUSYO	17	C3 (inverted)	Indicates PC is not ready to receive data
ACKO	16	C2 (inverted)	Set FZ's direction of data transfer (ACK low sets FZ to tri-state mode)
STROBEO	1	C0	Pulsing this tells FZ to accept 1 data byte
BUSY	11	S7	Tells PC, FZ has accepted a byte
STROBE	10	S6	Tells PC to accept 1 byte (Via an interrupt).

Low level data transfer details

Here is a brief overview of how to transfer data to an FZ. The timing is very critical, you must use an interrupt routine to receive data. You must disable all interrupts when sending data.

Because the FZ was not designed to talk to a PC, some of it's signals are inverted from what you would expect (eg the strobe line). So the PC interrupt is triggered on the trailing edge of the strobe signal, not the leading edge. This means you don't get the first byte of any data transfer to the PC (you get a 0 byte instead). Each block of data from the FZ has a checksum. You can use this to recover the 'lost' first byte.

If both the FZ and the PC try to send data at the same time, you will destroy your PC or FZ port (or both). A better way would be to make some type of buffer circuit to prevent this (don't ask me how). I have tried various resistors in the data lines, but they make the transfer unreliable. I now use no resistors (I MUST unplug the cable when not running FZ-DUMP).

For an example of how to do this see FZ-DUMP for DOS source code at:

www.iconz.co.nz/~jeffm/fzdump.htm

The relevant files are 'port_io.cpp' for the low level stuff, and 'fz_port.cpp' for the high level stuff.

Preparing PC to receive data

prepare PC port for input, and FZ port for output:

TRISTATE data lines
raise ACK

clear FZ's output buffer

lower BUSY
small time delay (approx. 2 milli second)
raise BUSY

Sequence of events receiving 1 byte FZ to PC

PC lowers BUSY
FZ Places DATA on outputs
FZ pulses STROBEOUT (0.25 micro sec)
PC raises BUSY
PC reads DATA off inputs

Preparing PC to receive data

lower ACK
lower BUSY
lower STROBE

Sequence of events sending 1 byte PC to FZ

PC Places DATA on outputs
PC raises BUSYO
PC raises STROBO
PC waits for FZ to raise BUSY
FZ reads DATA off inputs
PC lowers STROBEO
PC lowers BUSY
PC waits for FZ to lower BUSY

Written by Jeff McClintock
Last Updated : 16/5/1999

Disclaimer:
This may have errors. If you connect your FZ to your PC wrongly, you might destroy it.