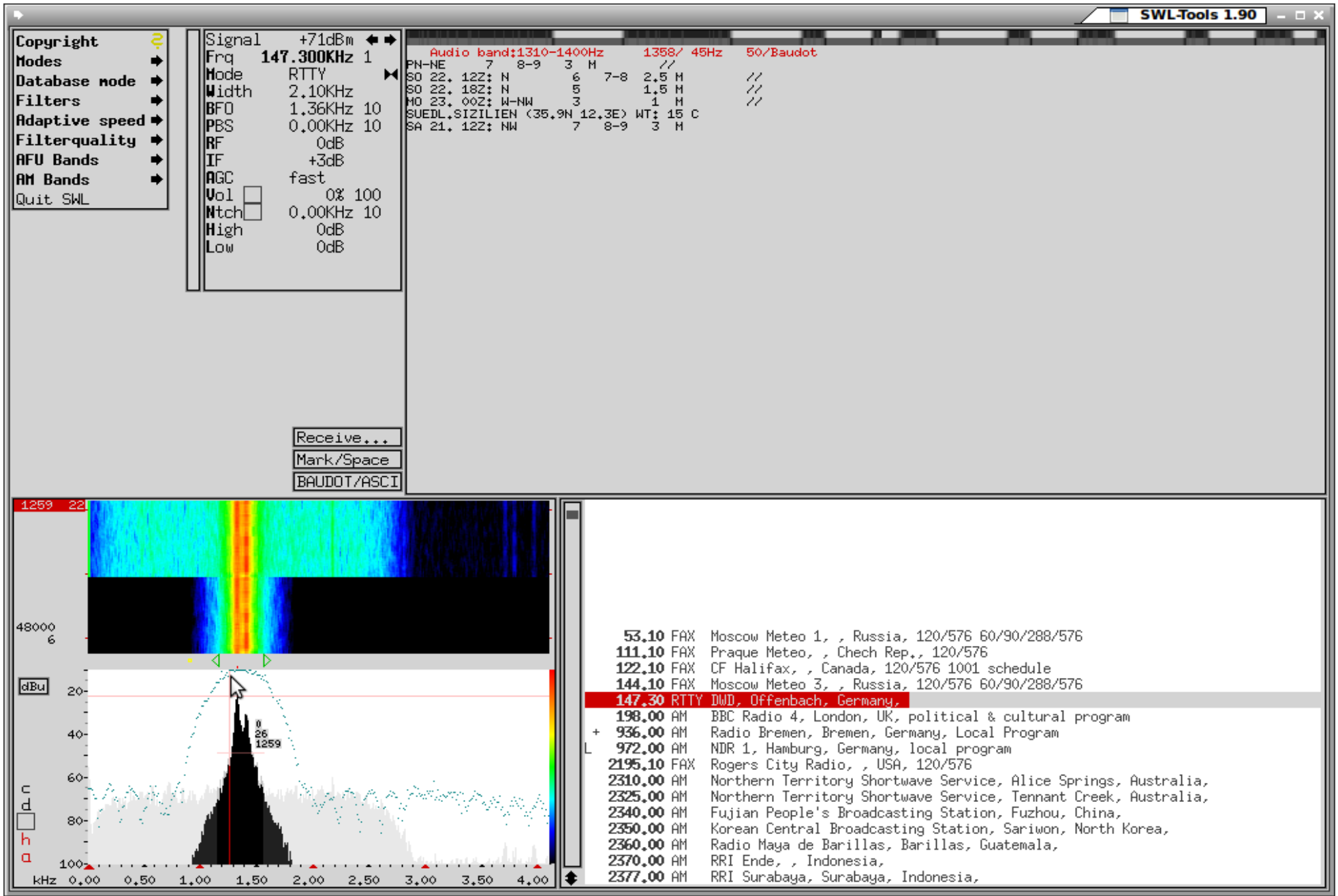
Echtzeit Audio Processing in iForth
Forth Tagung 2009

Hanno Schwalm

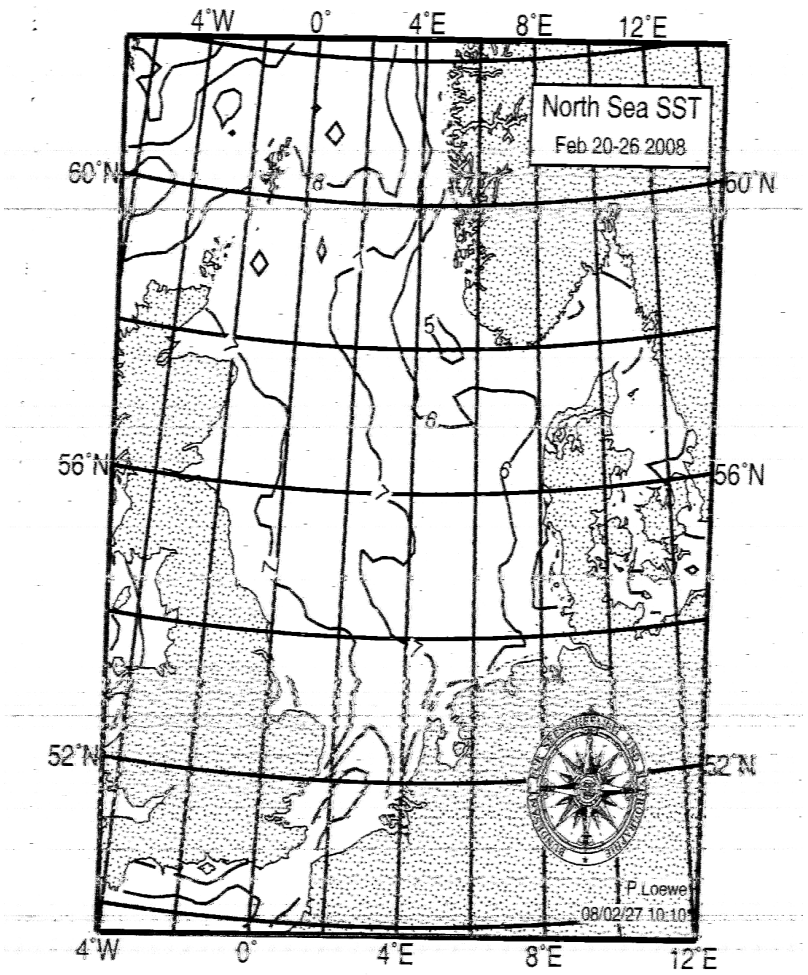
Echtzeit Audio Processing in iForth
Forth Tagung 2009

- Intensivmedizin - Lungenversagen
- Forth seit 1983
  - Z80 FIG ZX81, 68k Forthmacs Atari ST
  - ARM Forthmacs Port (Firmware ->One Laptop Per Child)
  - Linux iForth
    - multithreading, audio, xlib, Linux
  - c.l.f. passiv
- Radio und Audio Processing
- Simulation Lungenfunktion, Trainingssoftware

„Tägliche“ Forth Praxis



„Tägliche“ Forth Praxis



DDH3 DDK6 DDK3

QTUA88_EDZW

BUNDESAMT FÜR SEESCHIFFAHRT UND HYDROGRAPHIE

SEA SURFACE TEMPERATURES: 97874

iJACK (1)
der iForth JACK sound demon client

- Warum „Echtzeit“ Audio System ?
- Warum JACK Audio Connection Kit ?
- Wie ist iJACK implementiert ?
- Was ist in iJACK implementiert ?
- Ziemlich gute Dokumentation

iJACK (2)
der iForth JACK sound demon client

- iForth Erweiterungen & ANS Forth
  - Externe Bibliotheken
  - Callbacks
  - Threads / parallel Extension
  - TO Objects
  - S-Stack
- Demonstrationen
- Portierung auf andere Forth Systeme?
- Diskussion

iJACK (3)
der iForth JACK sound demon client

- iForth ist seit Sommer 2008 für 32/64bit Systeme verfügbar
- Windows/Linux/OSX
- Audio Unterstützung bislang nicht einheitlich gut
  - CPU-lastig; Latenzzeit massiv systemabhängig
- Einfache Entwicklung von Audio Anwendungen
- Schnittstelle zu anderen Audio Anwendungen
  - WAV Daten/Dateien
  - Echtzeit Datenstrom miteinander verbinden
  - MIDI

Was muss iJACK leisten ? (1)

- Unterstützung aller iForth Systeme
- Duplexbetrieb
- Mehr als zwei Kanäle z.B 8 oder 16
- Echtzeit mit niedrige Latenz < 20ms
  - Studio / Bühnenanwendungen
- Niedrige CPU Last
  - z.B. 1GHz P3
- Multicore sowie threading Unterstützung
- MIDI Unterstützung

Was muss iJACK leisten ? (2)

- Trennung von Signalverarbeitung und Anwendung
- Replay / Recording
- Resampling
- Umschalten der DSP-Algorithmen zur Laufzeit
- Testbarkeit der DSP-Algorithmen
- Stabil und robust
- Breite Hardwareunterstützung
- Definiertes Audiodaten Format: 32bit-float

Potentielle Lösungen

(1) Alles in Forth? - nicht sinnvoll/machbar

(2) Für jedes OS spezifische Implementation mit einheitlicher Forth-API

(3) Nutzung vorhandener portabler Software und Bibliotheken

- Definitiv weniger aufwendig
- Einfacher zu entwickeln
- Besser zu testen
- Potentiell größere „Entwicklerbasis“

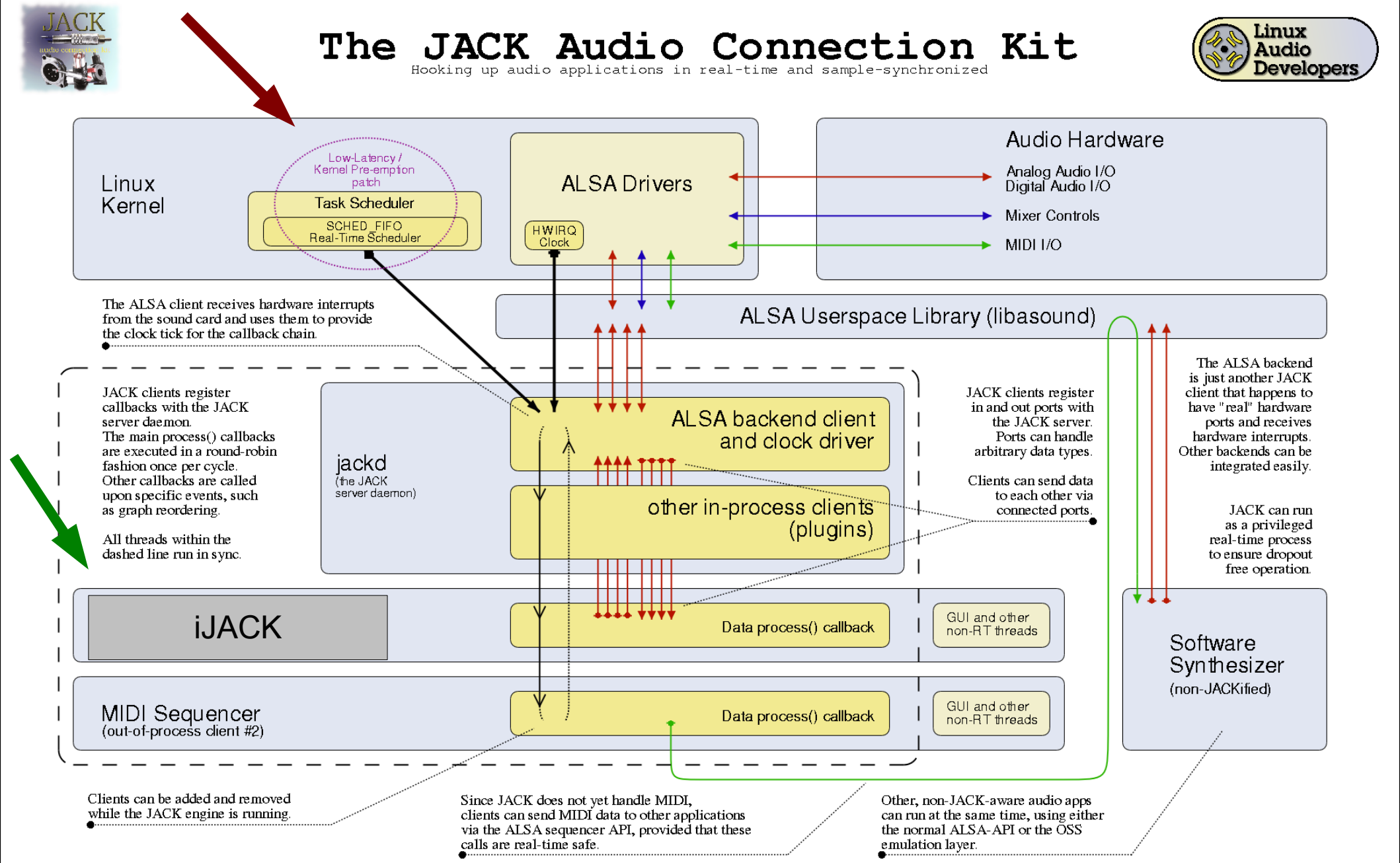
Lösungen für 3

- Portaudio
  - Unter Linux bislang nur OSS
- **Jack Audio Connection Kit**
  - Robust und leistungsfähig
  - Aktive und kompetente Entwicklercommunity
  - gute Dokumentation
  - einfache API
  - breite Softwareunterstützung
  - **KEINE** zusätzliche Latenz

JACK

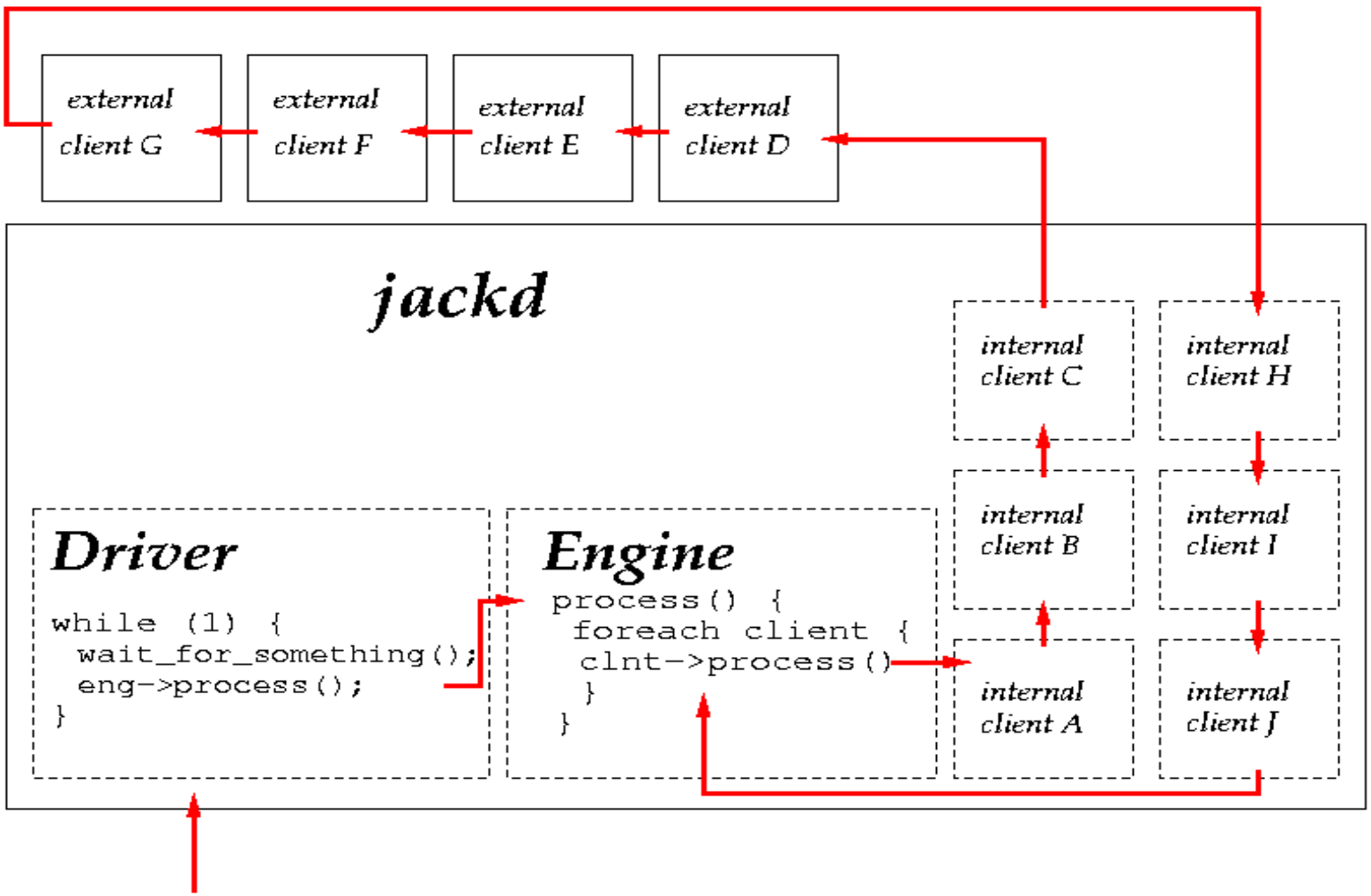
JACK Control

JACK


The JACK Audio Connection Kit
Hooking up audio applications in real-time and sample-synchronized
Linux Audio Developers
Linux Kernel
Low-Latency / Kernel Pre-emption patch
Task Scheduler
SCHED_FIFO Real-Time Scheduler
ALSA Drivers
HWIRQ Clock
Audio Hardware
Analog Audio I/O
Digital Audio I/O
Mixer Controls
MIDI I/O
ALSA Userspace Library (libasound)
The ALSA client receives hardware interrupts from the sound card and uses them to provide the clock tick for the callback chain.
JACK clients register callbacks with the JACK server daemon.
The main process() callbacks are executed in a round-robin fashion once per cycle.
Other callbacks are called upon specific events, such as graph reordering.
All threads within the dashed line run in sync.
jackd
(the JACK server daemon)
ALSA backend client and clock driver
other in-process clients (plugins)
JACK clients register in and out ports with the JACK server.
Ports can handle arbitrary data types.
Clients can send data to each other via connected ports.
The ALSA backend is just another JACK client that happens to have "real" hardware ports and receives hardware interrupts.
Other backends can be integrated easily.
JACK can run as a privileged real-time process to ensure dropout free operation.
iJACK
Data process() callback
GUI and other non-RT threads
MIDI Sequencer
(out-of-process client #2)
Data process() callback
GUI and other non-RT threads
Software Synthesizer
(non-JACKified)
Clients can be added and removed while the JACK engine is running.
Since JACK does not yet handle MIDI, clients can send MIDI data to other applications via the ALSA sequencer API, provided that these calls are real-time safe.
Other, non-JACK-aware audio apps can run at the same time, using either the normal ALSA-API or the OSS emulation layer.

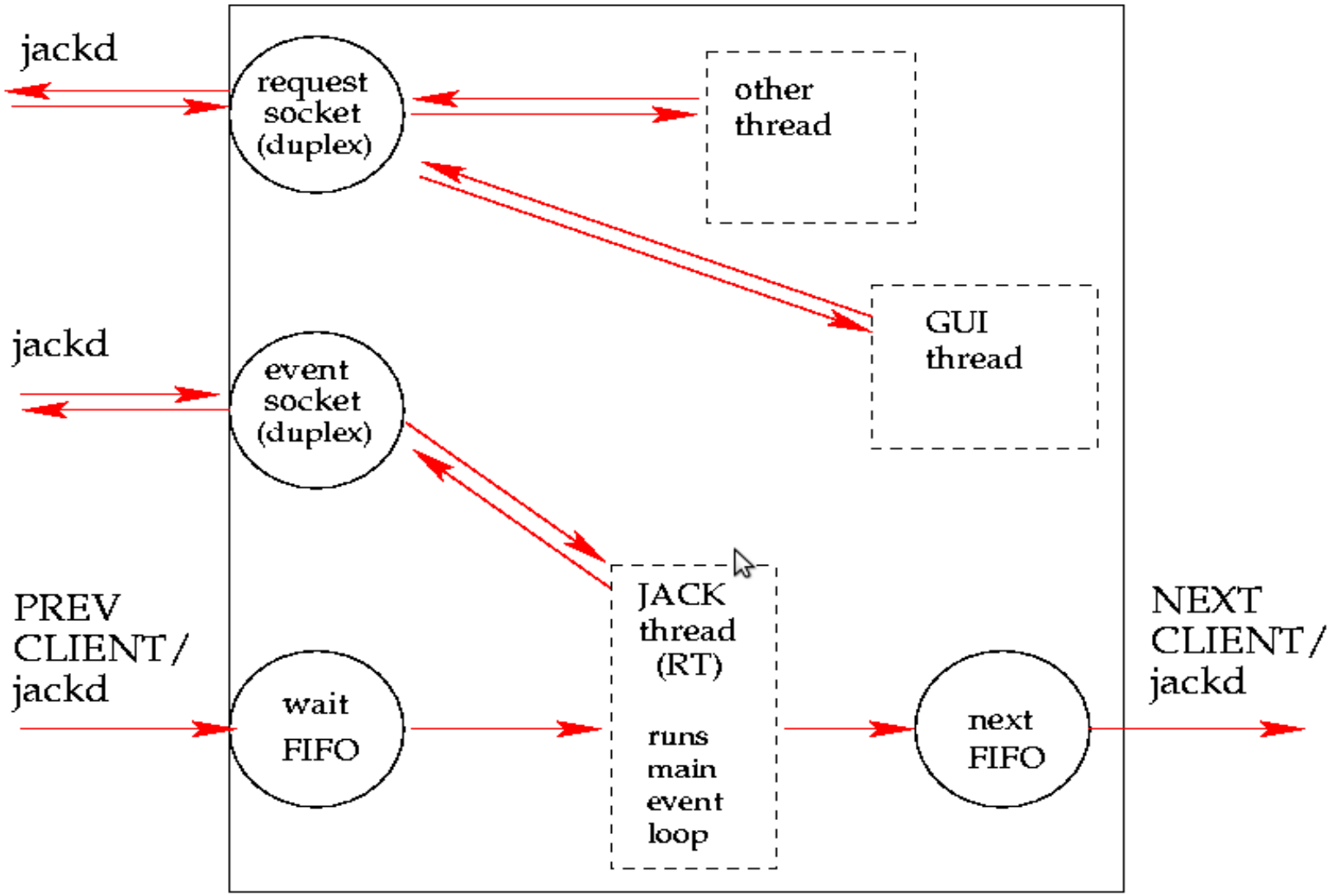
JACK

JACKD: putting it all together



JACK

A JACK EXTERNAL CLIENT



iJACK Statistik

0.96 – 32&64bit Linux, 32bit OSX, Win32

- 132kB Source, davon 60kB iJACK – Rest Erweiterungen
- JACK library Schnittstelle (jacklib) ~9.5kB
- Betriebssystem spezifisch: ~3.5kB
- 64/32bit spezifisch: ~1kB
- Assembler: 0kB

Implementation (1)

- iJACK ist ein externer JACK Client mit
  - Globaler Datenstruktur, die jacklib Parameter und Strukturen in Forth abbildet
    - Implementiert wegen der guten Lesbarkeit als „TO-Objects“

Implementation (1) TO-objects

TO-objects sind

- (Diskussion in c.l.f. Zu FVALUE)
- CREATE … DOES> words
- Folgen im sourcecode den Methoden wie
  - TO +TO 0TO ADDR ...

```
: clt-VAL          ( offset – offset' )
        CREATE   IMMEDIATE COMPILE-ONLY DUP *iJACK + , 1 CELLS +
                 DUP client-size > ABORT" clientsize too small"
        DOES>    @ ILITERAL %VAR @ %VAR OFF
                 CASE      0 OF   POSTPONE @      ENDOF
                           1 OF   POSTPONE !      ENDOF
                          -1 OF   POSTPONE +!     ENDOF
                           2 OF   POSTPONE OFF    ENDOF
                           3 OF                   ENDOF
                          TRUE ABORT" unsupported message for client-VALUE"
                 ENDCASE ; IMMEDIATE PRIVATE
```

Implementation (1b)

- iJACK ist ein externer JACK Client mit
  - Globaler Datenstruktur, die jacklib Parameter und Strukturen in Forth abbildet

```
0  clt-VAL iJACK-id
   clt-VAL #xruns                \ counter for xruns
   clt-VAL #play-current         \ index of current output WAV fifo
   clt-VAL #play-newest          \ index of newest output WAV fifo
   clt-VAL #read-current         \ index of current for input WAV fifo
   clt-VAL #read-newest          \ index of newest input WAV fifo
   clt-VAL *m_event $10 +        \ midievent space

   clt-VAL *m_inproc             \ pointer to function what should be d
   clt-VAL *m_outproc            \ pointer to function what should be d
   clt-VAL *m_inproc-act
   clt-VAL *m_outproc-act

   channel-VAL *io_proc          \ pointers to functions to do the proc
   channel-VAL *ana_proc         \ pointers to functions of analyse por
   channel-VAL *syn_proc         \ pointers to functions of sync ports
   channel-VAL *io_proc-act
```

Implementation (2)

- iJACK ist ein externer JACK Client mit
  - Globaler Datenstruktur, die jacklib Parameter und Strukturen in Forth abbildet
  - Sammlung von Forth Worten (API), die den JACK demon kontrollieren und abfragen

Implementation - Manual

JACKAUDIO **(--)** is a word set holding all words that could be used for implementing sophisticated jack extensions. When writing code using this API it is not necessary to use words defined there. Before using words in this word set, be sure you fully understand their implementation and usage.

JACK-CHANNELS **(-- n)** The default is 4, when you have a powerfull computer and wish to use more audio channels, just set this to 8 or whatever you need. Each channel has 4 ports defined

- An output „synthesizer“ port you may use for creating audio streams like in the demos.
- An input „analyser“ port used for recording or analysing data
- An input and an output port. These ports are used for the audio processing plugins described later. There is also a programmers interface to be used by you to define your own sound processing tools. Also think about using other jack plugins available in the net to be used by your forth application.

START-CLIENT **(--)** and

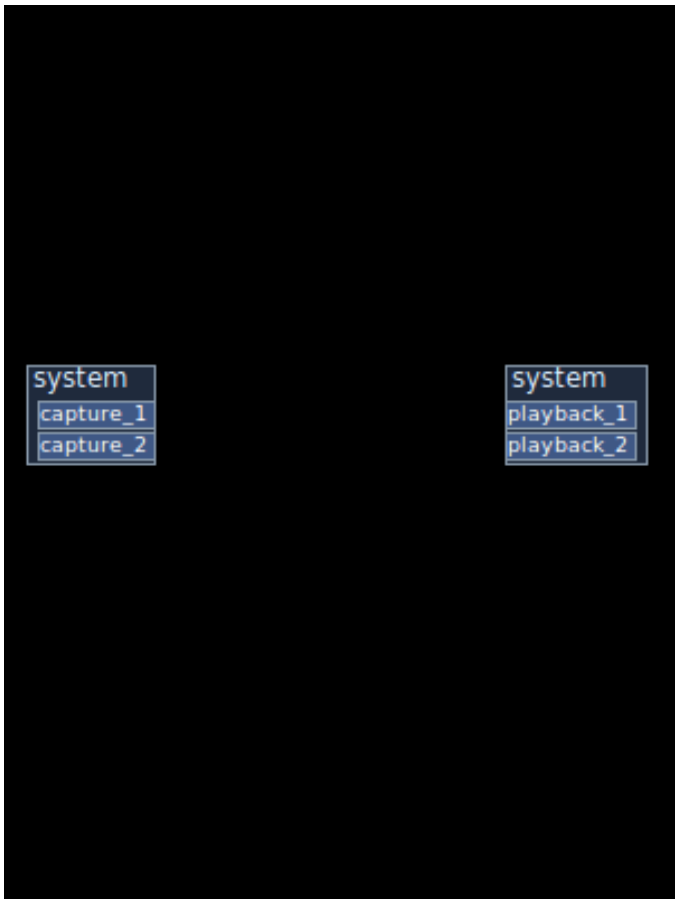
STOP-CLIENT **(--)** start or stop the client working. If the clients name is not unique, the jack demon might choose another name like `client-01`. This is OK as all client handling is done via the client structure **iJACK**. The port connections also take care about this. There is one good reason to do so: Different applications can all use the default client iJACK and don't have to care about other names. So you may start several iFORTH terminals and all can use the same iJACK client and share the connection files – the true client name is just hidden.

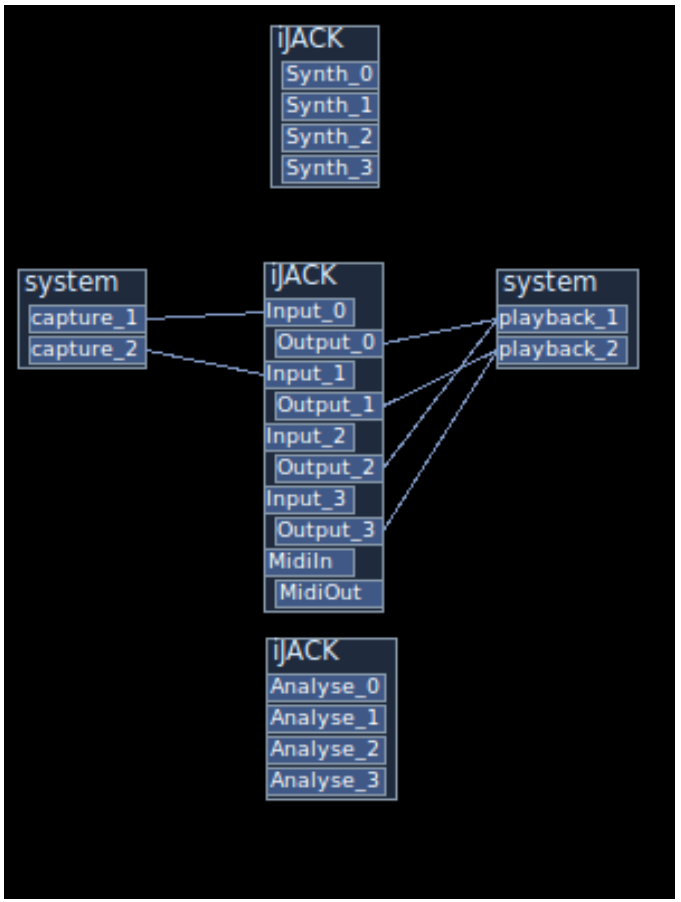
CONNECT-JACKPORTS **(source slen dest dlen --)** connects the ports defined by the two namestrings. You may do any connects desired but remember that the first part of any port name before the colon will be changed to the true clients name.

Implementation (3)

- iJACK ist ein externer JACK Client mit
  - Globaler Datenstruktur, die jacklib Parameter und Strukturen in Forth abbildet
  - Sammlung von Forth Worten (API), die den JACK demon kontrollieren und abfragen
  - 4 Ports pro Kanal – Schnittstelle zu anderen Clients und zur Hardware
    - Synthesizer
    - Analyse
    - Input
    - Output

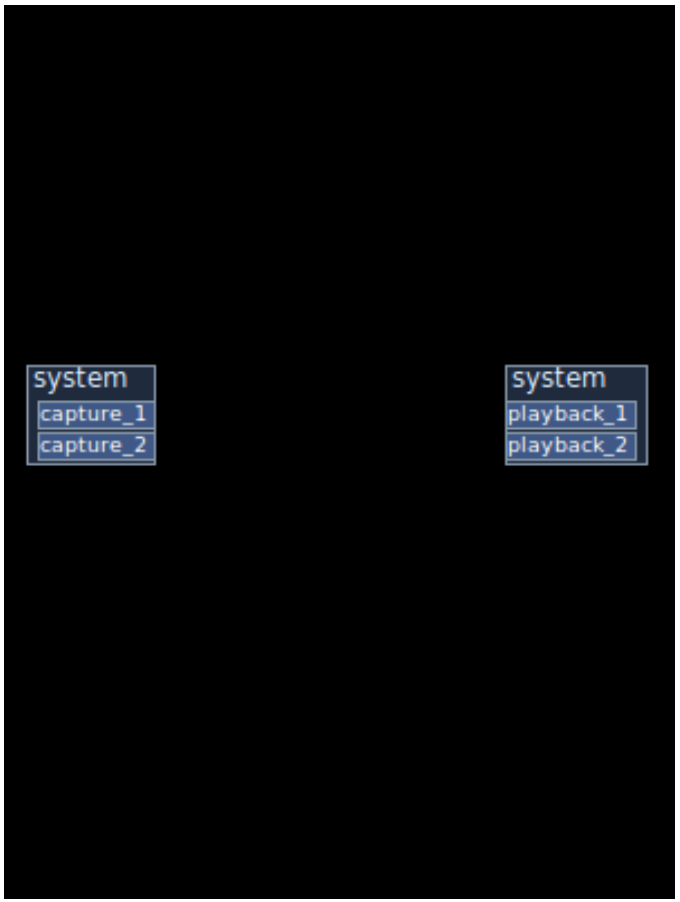
iJACK Ports (1)

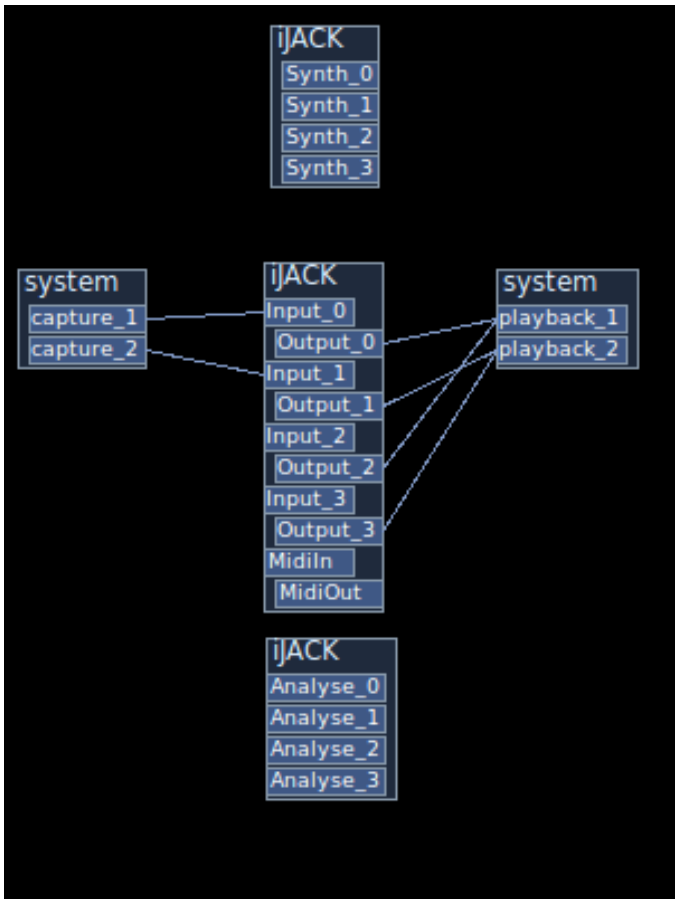




- CLEAR-CONNECTIONS
- (UN)CONNECT-JACKPORTS `( str1 len1 str2 len2 --)`
- (UN)LOAD/SAVE-CONNECTIONS `( str len -- )`

iJACK Ports (2)






```
S" iJACK:Output_0"  S" system:playback_1"    ?CONNECT-JACKPORTS
S" iJACK:Output_1"  S" system:playback_2"    ?CONNECT-JACKPORTS
S" iJACK:Output_2"  S" system:playback_1"    ?CONNECT-JACKPORTS
S" iJACK:Output_3"  S" system:playback_2"    ?CONNECT-JACKPORTS
S" iJACK:Input_0"  S" system:capture_1"  2SWAP  ?CONNECT-JACKPORTS
S" iJACK:Input_1"  S" system:capture_2"  2SWAP  ?CONNECT-JACKPORTS
```

Implementation (4)

- iJACK ist ein externer JACK Client mit
  - Globaler Datenstruktur, die jacklib Parameter und Strukturen in Forth abbildet
  - Sammlung von Forth Worten (API), die den JACK demon kontrollieren und abfragen
  - Ports – Schnittstelle zu anderen clients und zur Hardware
  - Realtime Callbacks in Forth

Callbacks (1)

- Ein callback ist eine temporäre Forth Umgebung. Gemeinsamer Speicher wie ein thread mit iForth. Aber lokale
  - Stacks
  - USER Area
  - „Dictionary“, TIB, PAD …
- Ein callback kann mit Parametern aus externen Programmen aufgerufen werden und liefert Ergebnisse zurück

```
:NONAME ( sr id -- 0 )  @ iJACK-id = IF TO #srate ELSE DROP THEN 0 ;
        CB( _int _int )CB-int NEW-SAMPLERATE PRIVATE
```

Callbacks (2)

- Speicherbereiche werden bei Start von iForth eingerichtet und intern über „bit-semaphores“ verwaltet. System-Speicherverwaltung wird nicht zur Thread/Callback-Laufzeit verwendet.
- Max 32 Threads oder Callbacks gleichzeitig
- Minimaler Aufwand, sehr schnell

Implementation (5)

- iJACK ist ein externer JACK Client mit
  - Globaler Datenstruktur, die jacklib Parameter und Strukturen in Forth abbildet
  - Sammlung von Forth Worten (API), die den JACK demon kontrollieren und abfragen
  - Ports – Schnittstelle zu anderen clients und zur Hardware
  - Realtime Callbacks in Forth
  - jacklib interface

jacklib interface (1)

- Benutzt iForth Modul **`dynlink`**
- Läd dynamische Bibliotheken

```
: (LOAD-JACKLIB)          ( -- n )
        S" /usr/lib/libjack.so.0" LIBRARY-OPEN 0= ?EXIT DROP
        S" /usr/local/lib/libjack.so.0" LIBRARY-OPEN 0= ?EXIT DROP
        64BIT?
        IF      S" /usr/lib64/libjack.so.0" LIBRARY-OPEN 0= ?EXIT DROP
                S" /usr/local/lib64/libjack.so.0" LIBRARY-OPEN 0= ?EXIT DROP
        ELSE    S" /usr/lib32/libjack.so.0" LIBRARY-OPEN 0= ?EXIT DROP
                S" /usr/local/lib32/libjack.so.0" LIBRARY-OPEN 0= ?EXIT DROP
        THEN
        LIBRARY-ERROR CR TYPE FORTH
        TRUE ABORT" Can't find a jack library; libjack.so.0 :: open failed" ;
```

jacklib interface (2)

- Benutzt iForth Modul **`dynlink`**
- Läd dynamische Bibliotheken
- sucht Einsprungaddressen von Funktionen

```
: jack_get_sample_rate                    \ (client_id -- sample_rate ) get the jack s
      JACK-LIB: jack_get_sample_rate C( _int )C-int ;
: jack_cpu_load                           \ (client_id -- FLOAT:cpuload ) get the cpu
      JACK-LIB: jack_cpu_load C( _int )C-float ;
: jack_get_time                           \ ( -- double_time ) get the jacks server ti
      JACK-LIB: jack_get_time C( )C-long ;
```

jacklib interface (3)

- Benutzt iForth Modul **dynlink**
- Läd dynamische Bibliotheken
- sucht Einsprungaddressen von Funktionen
- C(…)C-?? kompiliert Funktionsaufruf

```
: jack_get_sample_rate                  \ (client_id -- sample_rate ) get the jack s
      JACK-LIB: jack_get_sample_rate C( _int )C-int ;
: jack_cpu_load                         \ (client_id -- FLOAT:cpuload ) get the cpu
      JACK-LIB: jack_cpu_load C( _int )C-float ;
: jack_get_time                         \ ( -- double_time ) get the jacks server ti
      JACK-LIB: jack_get_time C( )C-long ;
```

Signal Processing

- Alle Audio Daten sind 32bit Floats
- Jeweils ein buff[#frames] pro Kanal & Port
- DSP komplett mit „Standard“ FORTH Worten
- Der JACK demon startet jeweils einen Callback
  - 48000 kHz Sample-Rate
  - 256 Daten/Kanal/Callback
    - **~190 Callbacks / sec**
    - Latenz ~ 20 msec

PROCESS-FRAMES (1)
Fehlerbehandlung

```
:NONAME   ( frames id -- 0 )
@ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                    \ exit with error with wrong id
iJACK-id jack_transport_query DUP TO #state JackTransportStopped =   \ silence in output when stopped
IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
DELAYED-FRONTENT-JOBS
JACK-CHANNELS 0         \ first get all *safe* buffers
DO      I #synport frames jack_port_get_buffer  I TO *synbuff
        I #anaport frames jack_port_get_buffer  I TO *anabuff
        I #outport frames jack_port_get_buffer  I TO *outbuff
        #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=        \ not playing wav to input buffer ?
        IF      I #inport frames jack_port_get_buffer   I TO *inbuff
        ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff    \ prepare copied/silent data
                I #inport jack_port_connected
                IF      I #inport frames jack_port_get_buffer           \ copy original input
                ELSE    *silence
                THEN    I *inbuff frames SFLOATS MOVE
        THEN
LOOP
WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                      \ playing wav to the input port needs early fill
JACK-CHANNELS 0
DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
        I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
        I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
LOOP
WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                       \ Now it's safe to play wav data to output port
frames BUFF->WAVSTREAM
#m_inport jack_port_connected 0<>  *m_inproc 0<> AND
IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
        *m_inbuff jack_midi_get_event_count
        0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=     \ ( event port_buffer frame_idx --  ?error )
                IF 'OF *m_event  *m_inproc EXECUTE THEN
        LOOP
THEN
#m_outport jack_port_connected 0<> *m_outproc 0<> AND
IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
THEN
0 ;                                                  CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

PROCESS-FRAMES (2)
RT-Callback ohne Semaphore etc.

```
:NONAME   ( frames id -- 0 )
@ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                          \ exit with error with wrong id
iJACK-id jack_transport_query DUP TO #state JackTransportStopped =         \ silence in output when stopped
IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
DELAYED-FRONTENT-JOBS
JACK-CHANNELS 0          \ first get all *safe* buffers
DO      I #synport frames jack_port_get_buffer  I TO *synbuff
        I #anaport frames jack_port_get_buffer  I TO *anabuff
        I #outport frames jack_port_get_buffer  I TO *outbuff
        #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=          \ not playing wav to input buffer ?
        IF      I #inport frames jack_port_get_buffer   I TO *inbuff
        ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff   \ prepare copied/silent data
                I #inport jack_port_connected
                IF      I #inport frames jack_port_get_buffer          \ copy original input
                ELSE    *silence
                THEN    I *inbuff frames SFLOATS MOVE
        THEN
LOOP
WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                        \ playing wav to the input port needs early fill
JACK-CHANNELS 0
DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
        I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
        I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
LOOP
WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                         \ Now it's safe to play wav data to output port
frames BUFF->WAVSTREAM
#m_inport jack_port_connected 0<>  *m_inproc 0<> AND
IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
        *m_inbuff jack_midi_get_event_count
        0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=       \ ( event port_buffer frame_idx --  ?error )
                IF 'OF *m_event  *m_inproc EXECUTE THEN
        LOOP
THEN
#m_outport jack_port_connected 0<> *m_outproc 0<> AND
IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
THEN
0 ;                                                    CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

PROCESS-FRAMES (3)
Daten im Buffer

```
:NONAME   ( frames id -- 0 )
@ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                          \ exit with error with wrong id
iJACK-id jack_transport_query DUP TO #state JackTransportStopped =         \ silence in output when stopped
IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
DELAYED-FRONTENT-JOBS
JACK-CHANNELS 0          \ first get all *safe* buffers
DO      I #synport frames jack_port_get_buffer  I TO *synbuff
        I #anaport frames jack_port_get_buffer  I TO *anabuff
        I #outport frames jack_port_get_buffer  I TO *outbuff
        #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=          \ not playing wav to input buffer ?
        IF      I #inport frames jack_port_get_buffer   I TO *inbuff
        ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff   \ prepare copied/silent data
                I #inport jack_port_connected
                IF      I #inport frames jack_port_get_buffer          \ copy original input
                ELSE    *silence
                THEN    I *inbuff frames SFLOATS MOVE
        THEN
LOOP
WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                        \ playing wav to the input port needs early fill
JACK-CHANNELS 0
DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
        I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
        I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
LOOP
WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                         \ Now it's safe to play wav data to output port
frames BUFF->WAVSTREAM
#m_inport jack_port_connected 0<>  *m_inproc 0<> AND
IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
        *m_inbuff jack_midi_get_event_count
        0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=       \ ( event port_buffer frame_idx --  ?error )
                IF 'OF *m_event  *m_inproc EXECUTE THEN
        LOOP
THEN
#m_outport jack_port_connected 0<> *m_outproc 0<> AND
IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
THEN
0 ;                                                    CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

PROCESS-FRAMES (4)
„playing“ WAV Data

```
:NONAME   ( frames id -- 0 )
@ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                          \ exit with error with wrong id
iJACK-id jack_transport_query DUP TO #state JackTransportStopped =         \ silence in output when stopped
IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
DELAYED-FRONTENT-JOBS
JACK-CHANNELS 0         \ first get all *safe* buffers
DO      I #synport frames jack_port_get_buffer  I TO *synbuff
        I #anaport frames jack_port_get_buffer  I TO *anabuff
        I #outport frames jack_port_get_buffer  I TO *outbuff
        #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=          \ not playing wav to input buffer ?
        IF      I #inport frames jack_port_get_buffer   I TO *inbuff
        ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff   \ prepare copied/silent data
                I #inport jack_port_connected
                IF      I #inport frames jack_port_get_buffer          \ copy original input
                ELSE    *silence
                THEN    I *inbuff frames SFLOATS MOVE
        THEN
LOOP
WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                        \ playing wav to the input port needs early fill
JACK-CHANNELS 0
DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
        I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
        I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
LOOP
WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                         \ Now it's safe to play wav data to output port
frames BUFF->WAVSTREAM
#m_inport jack_port_connected 0<>  *m_inproc 0<> AND
IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
        *m_inbuff jack_midi_get_event_count
        0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=       \ ( event port_buffer frame_idx --  ?error )
                IF 'OF *m_event  *m_inproc EXECUTE THEN
        LOOP
THEN
#m_outport jack_port_connected 0<> *m_outproc 0<> AND
IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
THEN
0 ;                                                    CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

PROCESS-FRAMES (4b)
„recording“ WAV Data

```
:NONAME   ( frames id -- 0 )
 @ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                          \ exit with error with wrong id
 iJACK-id jack_transport_query DUP TO #state JackTransportStopped =         \ silence in output when stopped
 IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
 DELAYED-FRONTENT-JOBS
 JACK-CHANNELS 0         \ first get all *safe* buffers
 DO      I #synport frames jack_port_get_buffer  I TO *synbuff
         I #anaport frames jack_port_get_buffer  I TO *anabuff
         I #outport frames jack_port_get_buffer  I TO *outbuff
         #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=          \ not playing wav to input buffer ?
         IF      I #inport frames jack_port_get_buffer   I TO *inbuff
         ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff   \ prepare copied/silent data
                 I #inport jack_port_connected
                 IF      I #inport frames jack_port_get_buffer          \ copy original input
                 ELSE    *silence
                 THEN    I *inbuff frames SFLOATS MOVE
         THEN
 LOOP
 WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                         \ playing wav to the input port needs early fill
 JACK-CHANNELS 0
 DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
         I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
         I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
 LOOP
 WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                          \ Now it's safe to play wav data to output port
 frames BUFF->WAVSTREAM
 #m_inport jack_port_connected 0<>  *m_inproc 0<> AND
 IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
         *m_inbuff jack_midi_get_event_count
         0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=       \ ( event port_buffer frame_idx --  ?error )
                 IF 'OF *m_event  *m_inproc EXECUTE THEN
         LOOP
 THEN
 #m_outport jack_port_connected 0<> *m_outproc 0<> AND
 IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
 THEN
 0 ;                                                     CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

PROCESS-FRAMES (5)
.. Signal Processing ..

```
:NONAME   ( frames id -- 0 )
@ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                          \ exit with error with wrong id
iJACK-id jack_transport_query DUP TO #state JackTransportStopped =         \ silence in output when stopped
IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
DELAYED-FRONTENT-JOBS
JACK-CHANNELS 0         \ first get all *safe* buffers
DO      I #synport frames jack_port_get_buffer  I TO *synbuff
        I #anaport frames jack_port_get_buffer  I TO *anabuff
        I #outport frames jack_port_get_buffer  I TO *outbuff
        #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=          \ not playing wav to input buffer ?
        IF      I #inport frames jack_port_get_buffer   I TO *inbuff
        ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff    \ prepare copied/silent data
                I #inport jack_port_connected
                IF      I #inport frames jack_port_get_buffer          \ copy original input
                ELSE    *silence
                THEN    I *inbuff frames SFLOATS MOVE
        THEN
LOOP
WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                            \ playing wav to the input port needs early fill
JACK-CHANNELS 0
DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
        I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
        I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
LOOP
WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                             \ Now it's safe to play wav data to output port
frames BUFF->WAVSTREAM
#m_inport jack_port_connected 0<>  *m_inproc 0<> AND
IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
        *m_inbuff jack_midi_get_event_count
        0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=       \ ( event port_buffer frame_idx --  ?error )
                IF 'OF *m_event  *m_inproc EXECUTE THEN
        LOOP
THEN
#m_outport jack_port_connected 0<> *m_outproc 0<> AND
IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
THEN
0 ;                                                    CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

PROCESS-FRAMES (6)
MIDI – (bislang keine Anwendung)

```
:NONAME   ( frames id -- 0 )
@ iJACK-id <> IF DROP TRUE EXIT THEN LOCAL frames                          \ exit with error with wrong id
iJACK-id jack_transport_query DUP TO #state JackTransportStopped =         \ silence in output when stopped
IF JACK-CHANNELS 0 DO *silence I #outport frames jack_port_get_buffer frames SFLOATS MOVE LOOP 0 EXIT THEN
DELAYED-FRONTENT-JOBS
JACK-CHANNELS 0         \ first get all *safe* buffers
DO      I #synport frames jack_port_get_buffer  I TO *synbuff
        I #anaport frames jack_port_get_buffer  I TO *anabuff
        I #outport frames jack_port_get_buffer  I TO *outbuff
        #play-current #play-newest <> WAV-PLAY-MODE 0= AND 0=          \ not playing wav to input buffer ?
        IF      I #inport frames jack_port_get_buffer   I TO *inbuff
        ELSE    *myinbuff I $1000 SFLOATS * +           I TO *inbuff   \ prepare copied/silent data
                I #inport jack_port_connected
                IF      I #inport frames jack_port_get_buffer          \ copy original input
                ELSE    *silence
                THEN    I *inbuff frames SFLOATS MOVE
        THEN
LOOP
WAV-PLAY-MODE 0= IF frames WAVSTREAM->BUFF THEN                            \ playing wav to the input port needs early fill
JACK-CHANNELS 0
DO      I *inbuff I *outbuff frames I DUP *io_proc ?DUP         IF EXECUTE ELSE DROP SFLOATS MOVE THEN
        I #synport jack_port_connected 0<> I *syn_proc 0<> AND  IF I *synbuff frames I DUP *syn_proc EXECUTE THEN
        I #anaport jack_port_connected 0<>  I *ana_proc 0<> AND IF I *anabuff frames I DUP *ana_proc EXECUTE THEN
LOOP
WAV-PLAY-MODE   IF frames WAVSTREAM->BUFF THEN                             \ Now it's safe to play wav data to output port
frames BUFF->WAVSTREAM
#m_inport jack_port_connected 0<>  *m_inproc 0<> AND
IF      #m_inport frames jack_port_get_buffer TO *m_inbuff
        *m_inbuff jack_midi_get_event_count
        0 ?DO   'OF *m_event  *m_inbuff I jack_midi_event_get 0=       \ ( event port_buffer frame_idx --  ?error )
                IF 'OF *m_event  *m_inproc EXECUTE THEN
        LOOP
THEN
#m_outport jack_port_connected 0<> *m_outproc 0<> AND
IF      #m_outport frames jack_port_get_buffer  TO *m_outbuff
THEN
0 ;                                                  CB( _int _int )CB-int PROCESS-FRAMES PRIVATE
```

Was kann iJACK ?

- Verwaltung des Client
- Port Verwaltung und Verbindungen
- Kontrolle und Abfrage des JACK demon
- Mehrere iForth Programme mit jeweils einem Client möglich
  - Daten können auch zwischen Forth Anwendungen „weitergeleitet“ werden
- Unterstützung von WAV Daten

WAV Daten (1)

- **„Blockweises“ Play und Record über zwei FIFOs gesteuert**
  - Trennung von Datengenerierung und Callback
- Formate
  - 8bit, 16bit, 32bit, 32bit-float
- Freie Anzahl von Kanälen
- Resampling per „linear approximation“
- Komplette WAV files können im Hintergrund (thread) abgespielt oder aufgezeichnet werden

WAV Daten (2)
z.B. **PLAY-WAV**

- (addr size rate channels mode - - id error?)
- Beliebiges size möglich
- Negatives size bedeutet „sofortiges“ play
  - Umschalten des FIFOs im nächsten PROCESS-FRAMES
- Kanalsortierung; Lautstärke
- Sample-exakte Ausgabe
- Mischung mit anderen Port daten vor oder nach IO-Processing

WAV Daten (3)
z.B. **PLAY-WAV**

- (addr size rate channels mode - - id error?)
- **error?** Diverse Fehlercodes oder FALSE

WAV Daten (4)
z.B. **PLAY-WAV**

- (addr size rate channels mode - - id error?)
- error? Diverse Fehlercodes oder FALSE
- **id** kann benutzt werden mit
- WAV-PLAYING? (id - - 0 | time)
  - time: msec bis Daten abgespielt sind
    - Definiertes Warten
    - Dynamische Speicherverwaltung
- STOP-WAV-PLAYING (ms id)

Input-Output Data Processing (1)

- Als default werden Audiodaten kopiert
- **SET-JACK-IO-PROCESS** (xt channel)
  definiert die DPS Funktion für einen Kanal
- **'xt'** (data-in data-out #frames channel)

Input-Output Data Processing (2)

- Modul zur Verwaltung diverser DSP Module ist vorhanden **jack/io_procs**
- Pro Kanal 32 Plugin Objekte möglich
- Für schnelle Entwicklung eigener Module liegt Prototyp vor
- Alle Plugin Objekte können gesteuert und abgefragt werden

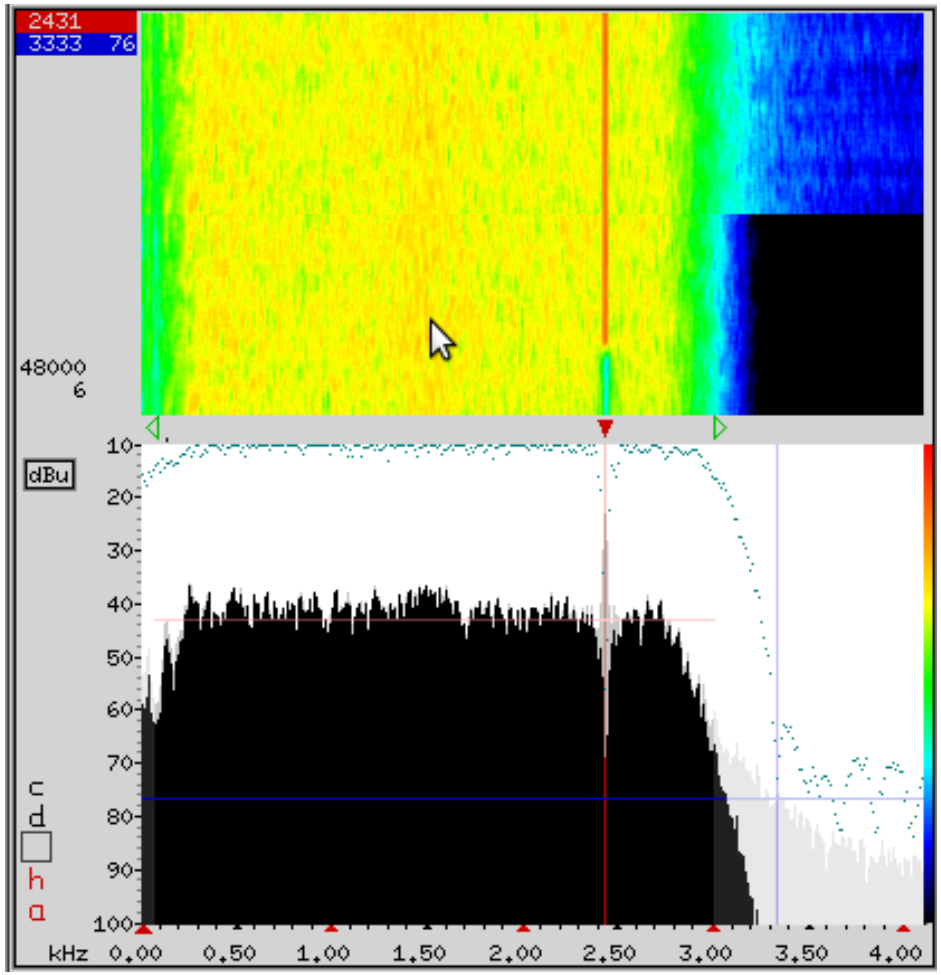
IO-Plugins API

- SET-JACK-PLUGIN
  - (filterobject slot channel --)
- REMOVE-JACK-PLUGIN
  - (filterobject --)
- GET-PLUGIN-OBJECT
  - (slot channel -- 0 | filterobject)
- LIST-PLUGINS
  - (--)

Vorhandene Plugins

- Filter
  - Finite Impulse Response
  - 10 Pol Infinite Impulse Response
  - Adaptive (Least Mean Square)
  - Notch
  - Echo&Delay
- FFT
  - Analyse (Hamming, Hann, ..), Maxima, S/N
  - Anzeige (bislang wegen GUI Modul nur Linux)
- Decoder
  - z.Z. RTTY, WEFAX, CW

FFT Display




```
Used iJACK Slots (channel,slot)
(0, 3) xt: $01299780,    0Hz, FFT plugin
(0,11) xt: $01299000, 3000Hz, Lowpass FIR-Filter
(0,12) xt: $01298B80,   80Hz, Highpass FIR-Filter
(0,15) xt: $012728C0, 2431Hz, Bandstop IIR Filter
(0,28) xt: $01299880,    0Hz, FFT plugin
```

Plugin Entwicklung (1)

```
HERE ," Prototype" CONSTANT #proto-id PRIVATE
#plug   plug_VALUE proto_cnt
        plug_VALUE proto_par
        plug_VALUE proto_buff
        CONSTANT #proto_struct PRIVATE

$FF CONSTANT #proto_mask PRIVATE
: calc-proto-val  \ ( object -- ) (F val -- val' )
        <PLUG  FDUP proto_cnt #proto_mask AND SFLOATS
        proto_buff + SF!
        1 +TO proto_cnt  PLUG> ; PRIVATE
: CHK-PROTO
        #proto-id plug_type <>
        ABORT" Invalid PROTOTYPE object" ; PRIVATE
: MAKE-PROTOTYPE          ( -- object )
        #proto_struct ALLOCATE THROW <PLUG INIT-PLUGIN
        -123 ( magic ) TO plug_filtype  0TO proto_cnt
        #proto-id TO plug_type
        #proto_mask 1+ SFLOATS ALLOCATE THROW TO proto_buff
        @PLUG >R
        :NONAME R@ ILITERAL POSTPONE calc-proto-val POSTPONE ;
        TO plug_xt  PLUG> R> ;
: SET-PROTO-PAR ( n object -- )
        <PLUG CHK-PROTO TO proto_par PLUG> ;
```

Plugin Entwicklung (2)

```
HERE ," Prototype" CONSTANT #proto-id PRIVATE
#plug   plug_VALUE proto_cnt
        plug_VALUE proto_par
        plug_VALUE proto_buff
        CONSTANT #proto_struct PRIVATE

$FF CONSTANT #proto_mask PRIVATE
: calc-proto-val  \ ( object -- ) (F val -- val' )
        <PLUG  FDUP proto_cnt #proto_mask AND SFLOATS
        proto_buff + SF!
        1 +TO proto_cnt  PLUG> ; PRIVATE
: CHK-PROTO
        #proto-id plug_type <>
        ABORT" Invalid PROTOTYPE object" ; PRIVATE
: MAKE-PROTOTYPE          ( -- object )
        #proto_struct ALLOCATE THROW <PLUG INIT-PLUGIN
        -123 ( magic ) TO plug_filtype  0TO proto_cnt
        #proto-id TO plug_type
        #proto_mask 1+ SFLOATS ALLOCATE THROW TO proto_buff
        @PLUG >R
        :NONAME R@ ILITERAL POSTPONE calc-proto-val POSTPONE ;
        TO plug_xt  PLUG> R> ;
: SET-PROTO-PAR ( n object -- )
        <PLUG CHK-PROTO TO proto_par PLUG> ;
```

```
plug_VALUE plug_channel
plug_VALUE plug_xt
plug_VALUE plug_type
plug_VALUE plug_filtype
plug_VALUE plug_frequency
plug_VALUE plug_used?
```

Plugin Entwicklung (3)

```
HERE ," Prototype" CONSTANT #proto-id PRIVATE
#plug   plug_VALUE proto_cnt
        plug_VALUE proto_par
        plug_VALUE proto_buff
        CONSTANT #proto_struct PRIVATE

$FF CONSTANT #proto_mask PRIVATE
: calc-proto-val  \ ( object -- ) (F val -- val' )
        <PLUG  FDUP proto_cnt #proto_mask AND SFLOATS
        proto_buff + SF!
        1 +TO proto_cnt  PLUG> ; PRIVATE
: CHK-PROTO
        #proto-id plug_type <>
        ABORT" Invalid PROTOTYPE object" ; PRIVATE
: MAKE-PROTOTYPE          ( -- object )
        #proto_struct ALLOCATE THROW <PLUG INIT-PLUGIN
        -123 ( magic ) TO plug_filtype  0TO proto_cnt
        #proto-id TO plug_type
        #proto_mask 1+ SFLOATS ALLOCATE THROW TO proto_buff
        @PLUG >R
        :NONAME R@ ILITERAL POSTPONE calc-proto-val POSTPONE ;
        TO plug_xt  PLUG> R> ;
: SET-PROTO-PAR ( n object -- )
        <PLUG CHK-PROTO TO proto_par PLUG> ;
```

Plugin Entwicklung (4)

```
HERE ," Prototype" CONSTANT #proto-id PRIVATE
#plug   plug_VALUE proto_cnt
        plug_VALUE proto_par
        plug_VALUE proto_buff
        CONSTANT #proto_struct PRIVATE

$FF CONSTANT #proto_mask PRIVATE
: calc-proto-val  \ ( object -- ) (F val -- val' )
        <PLUG  FDUP proto_cnt #proto_mask AND SFLOATS
        proto_buff + SF!
        1 +TO proto_cnt  PLUG> ; PRIVATE
: CHK-PROTO
        #proto-id plug_type <>
        ABORT" Invalid PROTOTYPE object" ; PRIVATE
: MAKE-PROTOTYPE          ( -- object )
        #proto_struct ALLOCATE THROW <PLUG INIT-PLUGIN
        -123 ( magic ) TO plug_filtype  0TO proto_cnt
        #proto-id TO plug_type
        #proto_mask 1+ SFLOATS ALLOCATE THROW TO proto_buff
        @PLUG >R
        :NONAME R@ ILITERAL POSTPONE calc-proto-val POSTPONE ;
        TO plug_xt  PLUG> R> ;
: SET-PROTO-PAR ( n object -- )
        <PLUG CHK-PROTO TO proto_par PLUG> ;
```

```
plug_VALUE plug_channel
plug_VALUE plug_xt
plug_VALUE plug_type
plug_VALUE plug_filtype
plug_VALUE plug_frequency
plug_VALUE plug_used?
```

S - Stack

- Thread-safe „Extra“-Stack ermöglicht oft einfachere Handhabung von Daten
- >S S> S
- Diverse in iJACK benutzte Objekte werden über den Top-of-S-Stack definiert
- speziell die Plugin-Objekte freuen sich über eine gute Implementation
  - z.B. Performance-Sprung bei 32 → 64 bit wegen eigenem Register in iForth

Portierungsprobleme

- Systemebene
  - Callbacks
  - Externe Libraries
  - Threads
- S – Stack
  - Gleich schnelle Implementation nicht möglich
  - In Plugins eventuell LOCAL
  - In WAV FIFOs
- TO – Objects
  - Konflikte mit ANS „TO“

Probleme – Still missing :-((

- „Free running JACK“ ermöglicht Anwendungen unabhängig von System sample rate
  - Bislang nicht getestet
- MIDI
  - Erste Testsitzung mit Marcel 29.03.2009 :-)
- „Repositioning“
  - JACK kann im Prinzip vor- und zurückspulen, wie dies in iForth zu implementieren ist → keine Ahnung. Kenne aber auch keine andere Echtzeitanwendung, die dies kann.

Demonstrationen ...

- RTTY Signal mit High-/Lowpass Filtern
- SSB Signal mit Störcarrier → Adaptive Filter
- Gerne auch Ausprobieren am Gerät

Wunschdenken ...

- Andere Forth-Leute sind auch an Audio Anwendungen interessiert
- Kommentare zu iJACK und der API
- Portierung (Änderung) auf andere Forth Systeme
- Entwicklung weiterer Plugins

hanno@schwalm-bremen.de
mhx@iae.nl