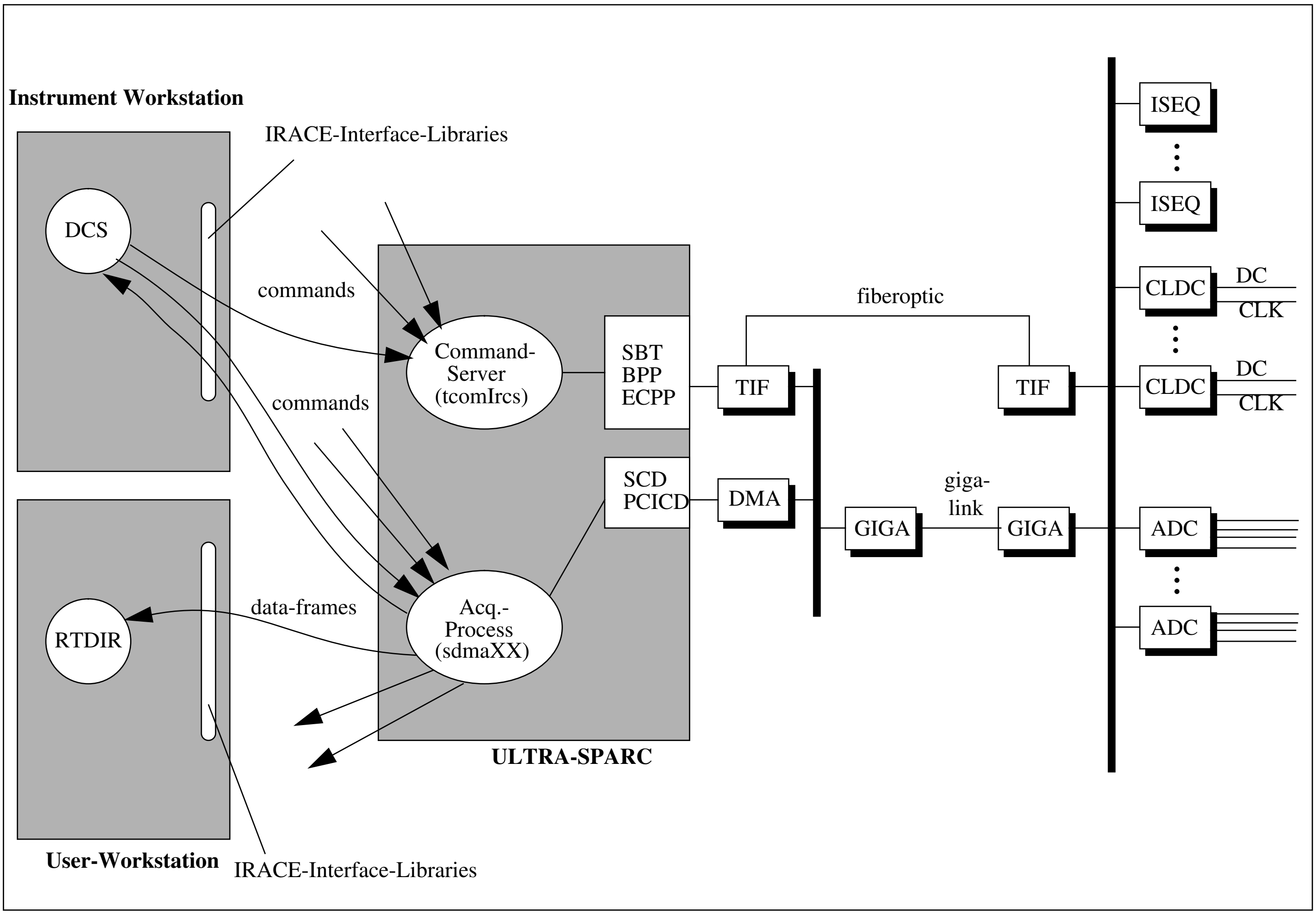
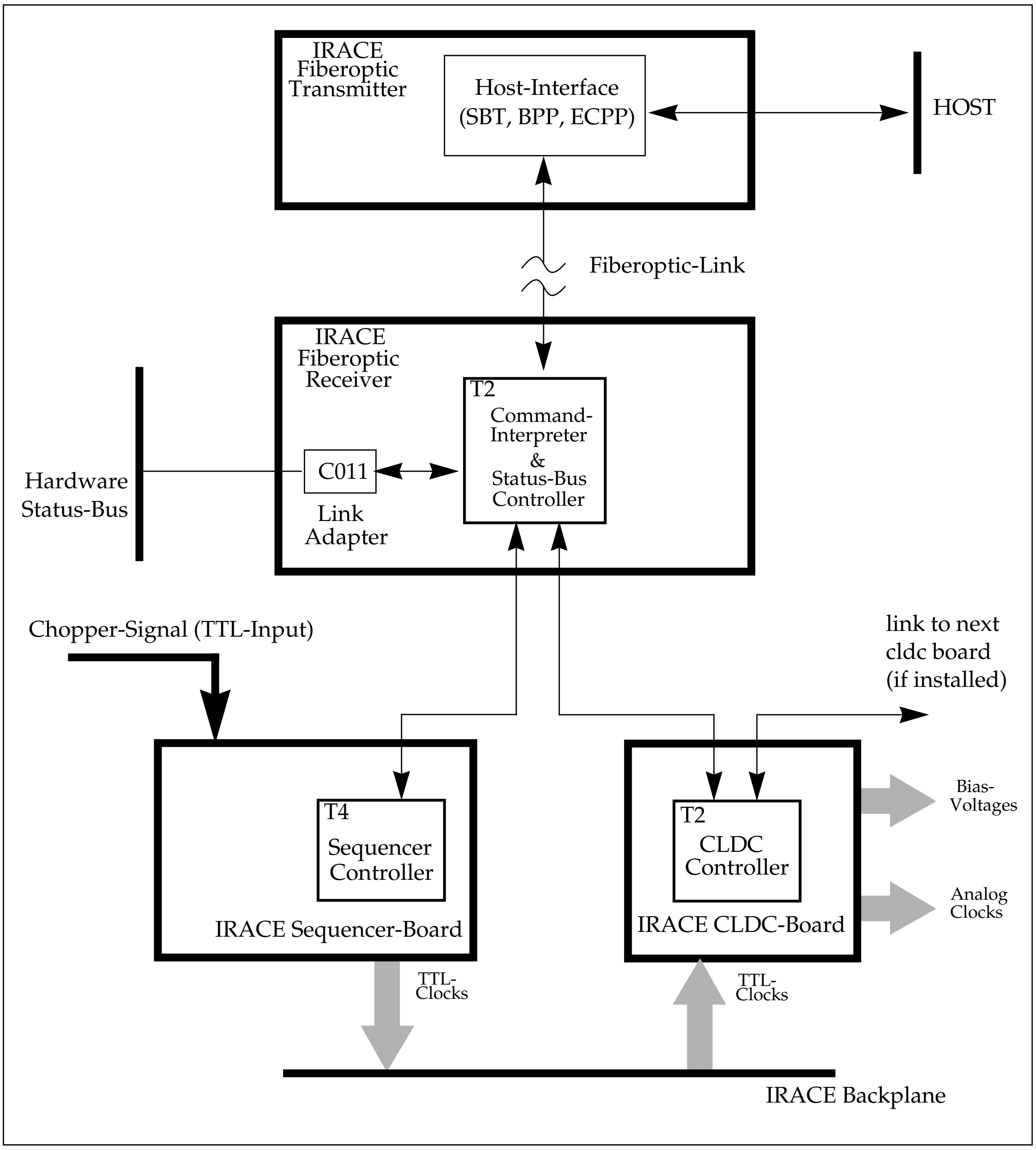
Infrared-Array-Control-Electronics (IRACE)

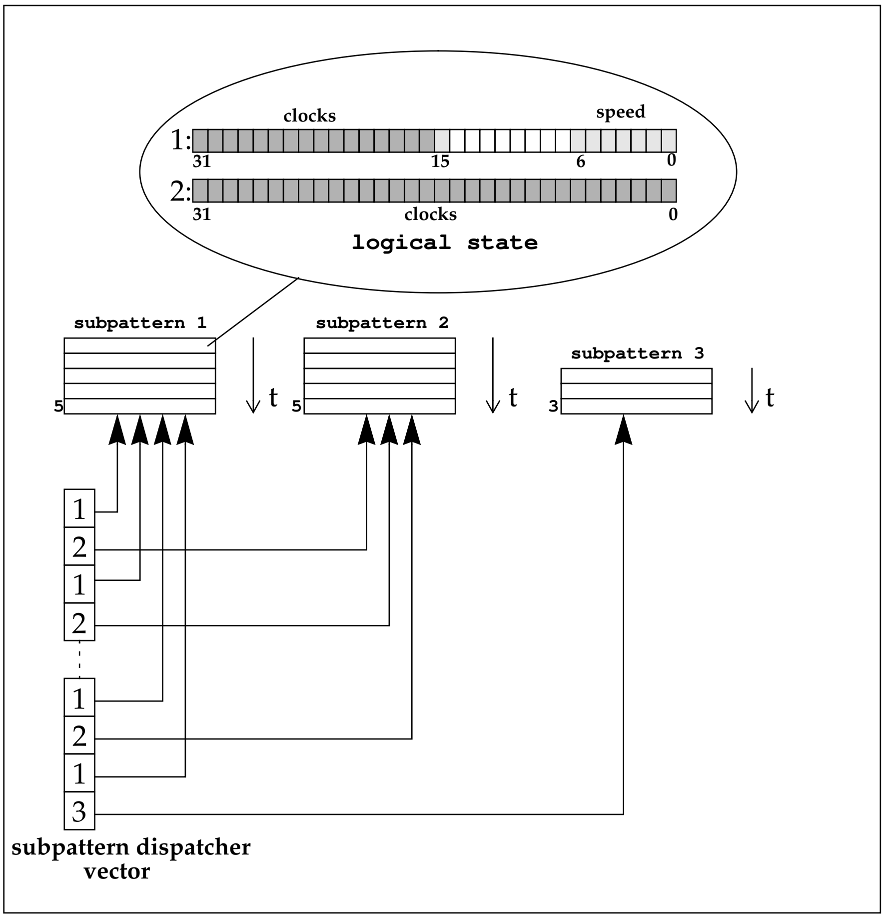
IRACE OVERVIEW



IRACE FRONT-END



SEQUENCER-PATTERN-ORGANIZATION



CLOCK-PATTERN FILE

```
##########################################################################
# E.S.O. - VLT project
#
# xxxx.clk
#
# who       when      what
# --------  --------  ----------------------------------------------
#
##########################################################################
#
# DESCRIPTION
# Clock-pattern file.
#
##########################################################################

# Header parameters
DET.CHIP.NO                 1;  # Detector Type
DET.SUBPAT.NO              15;  # No of Sub-Patterns
DET.CLKP.NO                11;  # No of Clocks

# Placement of software clocks onto hardware clocks
# Maximum 5 clocks per line. Can continue over several lines if terminated
# with ",".
DET.CLKP.SWHWCLK1  "1,2,3,4,5,";
DET.CLKP.SWHWCLK2  "6,7,8,9,33,";
DET.CLKP.SWHWCLK3  "11";

DET.SUBPAT1.NAME        "Delay";         # Subpattern name
DET.SUBPAT1.STATES            4;         # No of states
DET.SUBPAT1.RFAC              1;         # repetition factor
DET.SUBPAT1.STATEV1      "0000";         # SYNC Fast
DET.SUBPAT1.STATEV2      "0000";         # CK1 Fast
DET.SUBPAT1.STATEV3      "0000";         # CK2 Fast
DET.SUBPAT1.STATEV4      "0000";         # SYNC Slow
DET.SUBPAT1.STATEV5      "0000";         # CK1 Slow
DET.SUBPAT1.STATEV6      "0000";         # CK2 Slow
DET.SUBPAT1.STATEV7      "0000";         # CK RESET
DET.SUBPAT1.STATEV8      "0000";         # VBout CK
DET.SUBPAT1.STATEV9      "0000";         # Trigger
DET.SUBPAT1.STATEV10     "0000";         # Convert
DET.SUBPAT1.STATEV11     "0000";         # Spare
DET.SUBPAT1.RSPEEDV      "10,10,10,10";  # readspeed
DET.SUBPAT1.RSPEEDP      "0,0,0,0";      # permission

### up to 31 subpatterns like this
# DET.SUBPAT<nr>.NAME               "xxxxxx";      #subpattern name
# DET.SUBPAT<nr>.STATES             x;             #no of states
# DET.SUBPAT<nr>.RFAC               x;             #repetition factor
# DET.SUBPAT<nr>.STATEV<statNr>     "xxxx";        #DET.SUBPAT<nr>.STATES states
# DET.SUBPAT<nr>.RSPEEDV            "x,x,x,x";     #read speed for each state
# DET.SUBPAT<nr>.RSPEEDP            "x,x,x,x";     #permission for each state
#
# nr     = [1 ... DET.SUBPAT.NO]
# statNr = [1 ... DET.CLKP.NO]
###
```

SUBPATTERN DISPATCHER FILE

```
###########################################################################
# E.S.O. - VLT project
#
# xxxx.ssd
#
# who       when      what
# --------  --------  ----------------------------------------------
#
###########################################################################
#
# DESCRIPTION
# Subpattern dispatcher file.
#
###########################################################################

#
# subpattern declaration
#
FRAMESTART         = 1
READ               = 2
FRAMEEND           = 3

#
# frame start
#
EXEC FRAMESTART

#
# readout loop
#
LOOP (DET.NY/2)
    EXEC READ (DET.NX/2)
END

#
# frame end
#
EXEC FRAMEEND
```

VOLTAGE FILE

```
########################################################################
# E.S.O. - VLT project
#
# xxxx.config
#
# who       when      what
# --------  --------  ---------------------------------------------
#
########################################################################
#
# DESCRIPTION
# Voltage file.
#
########################################################################

########################################################################
# Clock Voltages:
########################################################################

DET.VOLT.CLKHINM1  "ck1Hi.SyncFast";  # Name High Clock Voltage 1
DET.VOLT.CLKHI1                -7.35;  # Level High Clock Voltage 1
DET.VOLT.CLKHIRA1    "[-8.0, -6.5]";  # Range High Clock Voltage 1
DET.VOLT.CLKHIDAC1                 1;  # DAC Ch High Clock Voltage 1
DET.VOLT.CLKHITEL1                 1;  # Tel Ch High Clock Voltage 1

DET.VOLT.CLKLONM1  "ck1Lo.SyncFast";  # Name Low Clock Voltage 1
DET.VOLT.CLKLO1                -3.03;  # Level Low Clock Voltage 1
DET.VOLT.CLKLORA1    "[-3.5, -2.0]";  # Range Low Clock Voltage 1
DET.VOLT.CLKLODAC1                 2;  # DAC Ch Low Clock Voltage 1
DET.VOLT.CLKLOTEL1                 2;  # Tel Ch Low Clock Voltage 1

### 16 clocks in this syntax

########################################################################
# DC Voltages:
########################################################################

DET.VOLT.DCNM1       "DC1-VDDUC";  # Name DC Voltage 1
DET.VOLT.DC1                 -3.8;  # Level DC Voltage 1
DET.VOLT.DCRA1   "[-4.5, -3.5]";  # Range DC Voltage 1
DET.VOLT.DCDAC1                33;  # DAC Ch DC Voltage 1
DET.VOLT.DCTEL1                33;  # Tel Ch DC Voltage 1

### 15 dc voltages in this syntax
```

IRACE SEQUENCER MODES

- Normal running mode
- External trigger mode

In normal running mode the sequencer starts immediately after receiving the `start'-command. It terminates, when the sequencer loop terminates or when a `stop'-command is received.

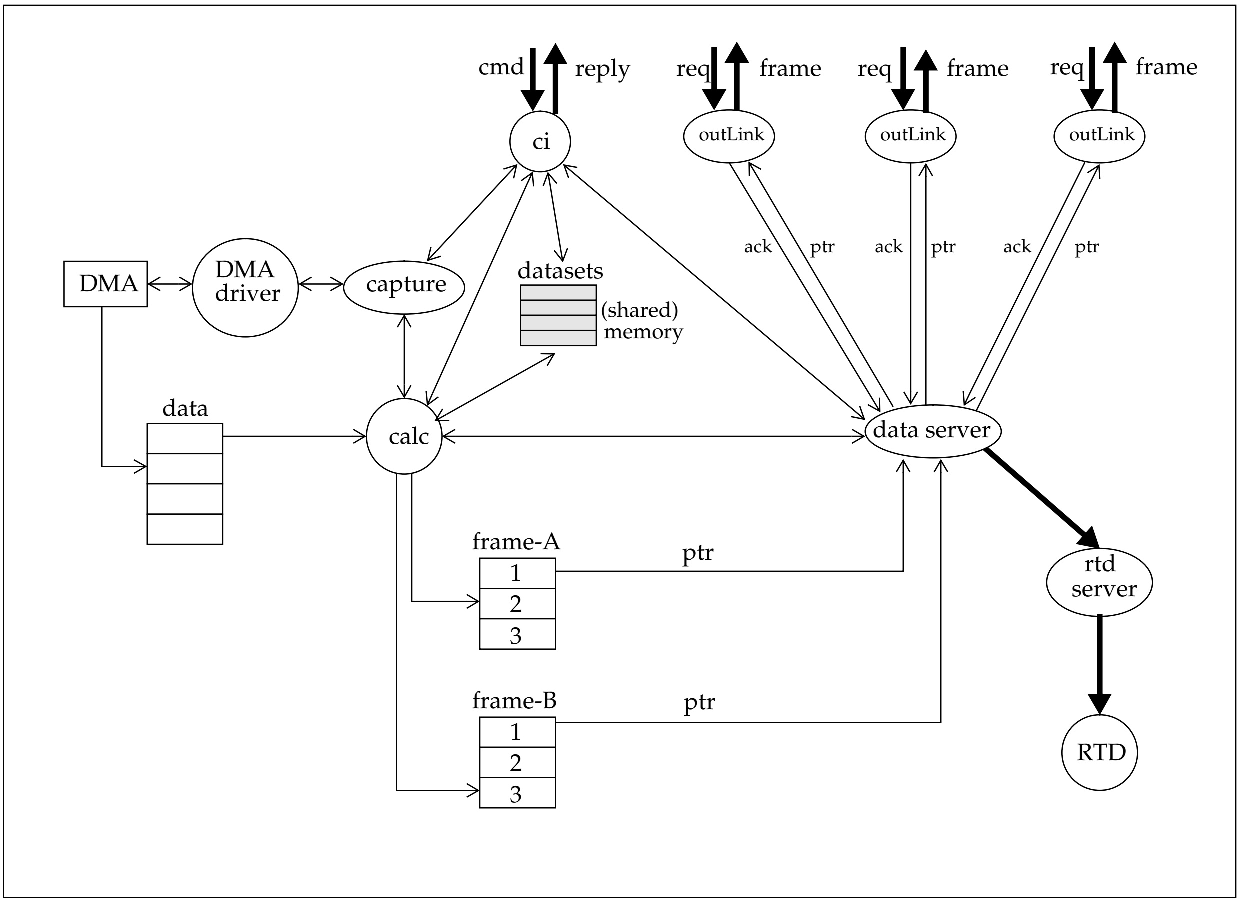
In external trigger mode after receiving the `start'-command the sequencer waits for a rising edge on the chopper TTL-input and then starts executing the sequencer program loop. When the program loop reaches a synchronization point, the execution is suspended, until the next edge is reached. The sequencer-hardware can be configured to resume execution on either the rising-edge or on both edges or to run continuously after the first rising edge.

IRACE SEQUENCER PROGRAM-LOOP

Synchronization points are marked with `**<---**':

```
ASSIGN DELAY MyDelay.ssd
ASSIGN RESET MyReset.ssd
ASSIGN FRAME MyFrame.ssd

# Define time for the minimum DIT
MINDIT = TIME(FRAME)

# Calculate exposure time
EXPO_TIME = (...)

# Calculate Chopping Frequency
CHOP_FREQ = (1 / (2 * (((DET.NDIT+DET.NDITSKIP) * DET.DIT) + DET.CHOP.TRANSTIME)))

<---
# Sequencer main loop
LOOP INFINITE

    # Chop half cycle
    LOOP (DET.NDIT + DET.NDITSKIP)

        # Integration
        RESET 1
        FRAME 1
        DELAY ((DET.DIT - TIME(FRAME)) / TIME(DELAY))
        FRAME 1

    END

    # Synchronize with chopper
    SYNC <---

END
<---
```

IRACE PRE-PROCESSOR
SOLARIS DMA DATA-ACQUISITION MODULE



DATA RECEPTION

The *sdmaOpenDataLine()* routine distinguishes between video- and science-data transfer. In science mode the oldest available, but not yet transferred frame, which matches the frame-type in the request structure is transferred. In video-mode for each frame-type matching the request, the latest not yet transferred frame is selected. From this selection the oldest frame is transferred. In video-mode additionally the retransmit flag in the request-structure is checked, to allow frames of the specified type to be transmitted again. This can be used, to display different windows of the same frame during long integrations. The sequence for the function calls should be the following:

- *sdmaPingDataLine()* (optional);
- *sdmaOpenDataLine();*
- *sdmaSendDataRequest();*
- s*dmaWaitDataReply();*
- check frame header;
- if frame type is sdmaFRAME_NO_FRAME, then send a new data request. Otherwise do either transfer the frame with *sdmaAcceptFrame()* or skip it with *sdmaSkipFrame();*
- continue with *sdmaSendDataRequest()*;

The linkId can be used for a select() call before sdmaWaitDataReply. To cancel a request one has to call *sdmaCancelReq()* before *sdmaWaitDataReply()* and then send a new request with *sdmaSendDataRequest()*. If any of these functions fails, one should call *sdmaCloseDataLine()* and then try to reopen with s*dmaOpenDataLine()* to recover the data-channel from failure. The calling program should not exit with an open request. When the process is waiting for the data reply, *sdmaAbortReq()* or *sdmaCancelReq()* have to be called before exiting. It is recommended to add s*dmaAbortReq()* or *sdmaCancelReq()* to a signal handler.

SIMULATION MODE

IRACE SEQUENCER AND CLOCK MODULE SIMULATOR (tcomScls):

- Started instead of the command server process;
- Simulates all system services functions;
- Simulates sequencer, cldc - and status-bus functions;
- Supports both Sun-Solaris and HP-UX;

ACQUISITION PROCESS:

- The capture task has a built-in simulator. The input-ringbuffer is pre-filled with data. The input sequence is simulated via microsleep functions (resolution: 10ms per input-frame).

- Calculation-task, command interpreter and output task remain untouched.

- For simulation on HP-UX the sdma-library contains a thread-wrapper using posix-threads (no real-time performance). The MAKE_POSIX flag has to be set, when compiling on HP_UX.

IRACE TASKS RUNNING ON THE ULTRA-SPARC

The tasks are started as remote-processes from the instrument workstation. The remote exec- (and remote kill-) functions are part of the interface library supplied together with the irace module.

IRACE COMMAND SERVER:

- converts commands to transputer link protocol
- provides all system service functions:
    - open link
    - close link
    - reset
    - boot
    - setting of link timeout
    - abort
- multiplexed command input (for debug-line)

SOLARIS DMA ACQUISITION PROCESS:

- Multithreaded environment (Solaris Realtime Class)
- Command Interpreter Task (highest priority) for start-/stop-/parameter- and status-commands
- Capture-Task (interface to DMA-driver; also does buffer-overrun checks))
- Calculation-Task (process data in parallel and create data-frames)
- Output-Task(s) (transfer data-frames in parallel to any requesting client on any host)
- The calculation task is designed as a main-process. The other task are started by the main-process via functions, which are part of the sdma-library. The sdma-library contains all input-, output-, parameter- and synchronization functions.

INTERACTIVE MODE