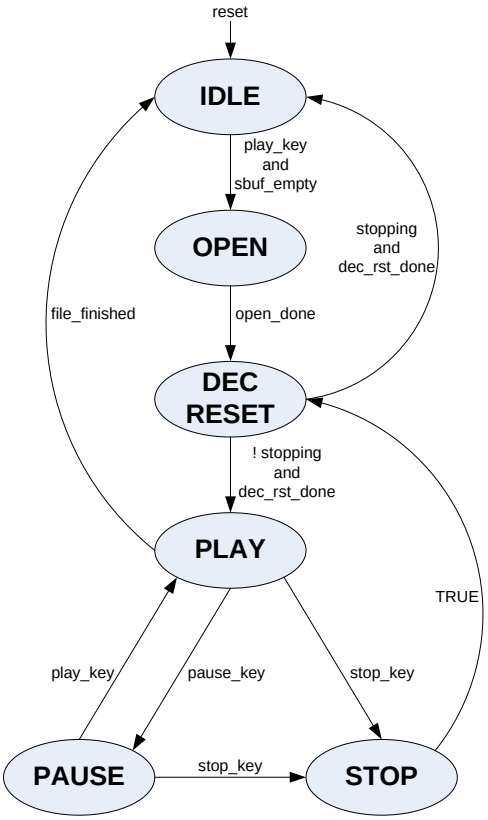
PLAY state machine

PLAY FSM starts MONITOR FSM by the fetch_en signal.

MONITOR FSM stops PLAY FSM by the file_finished signal.



Outputs from the FSM

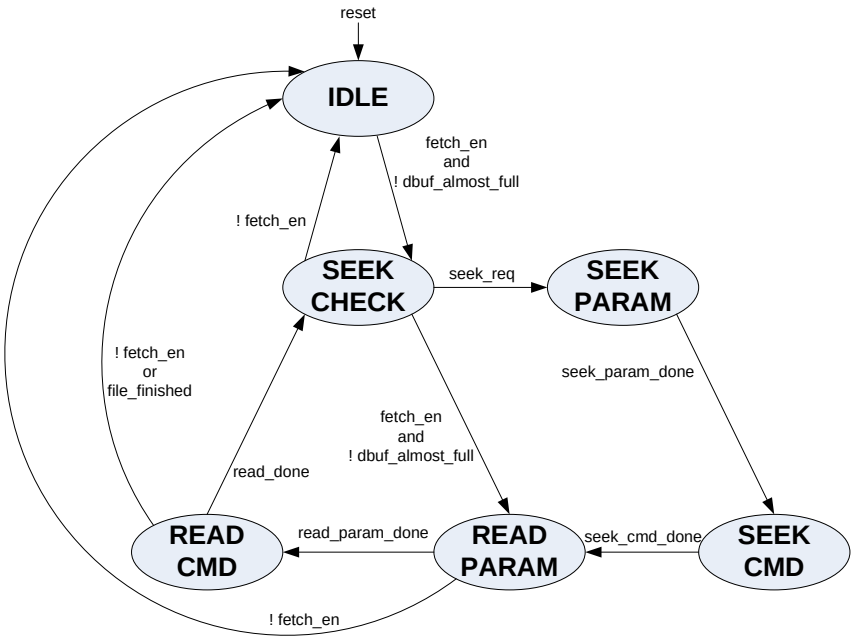
fio_req
fio_busiv
open_done
dec_rst
dbuf_rst
sbuf_rst
fetch_en
hw_din
hw_wr
stopping

Notes.
Outputs are not shown for keeping the drawing simple and understandable.
Transitions that do not change the state, are also not shown.

MONITOR state machine

PLAY FSM starts MONITOR FSM by the fetch_en signal.

MONITOR FSM stops PLAY FSM by the file_finished signal.



Outputs from the FSM

dbuf_wr
fio_req
fio_busiv
fio_busi
fio_ctrl
read_param_done
seek_param_done
seek_cmd_done
read_done
file_finished
this_dword_cnt
total_dword_cnt
seek_req
decrst_onseek

Notes.
Outputs are not shown for keeping the drawing simple and understandable.
Transitions that do not change the state, are also not shown.

```
 1 ---------------------------------------------------------------------------------
 2 -- PLAY FSM
 3 ---------------------------------------------------------------------------------
 4   state_register: process (clk, reset)
 5   begin
 6     if (reset = reset_state) then
 7       state <= IDLE;
 8     elsif (clk'event and clk = clk_polarity) then
 9       state <= next_state;
10     end if;
11   end process;
12
13   next_state_comb_logic: process (state, play, pause, stop, open_done,
   dec_rst_done,
14                                   file_finished, music_finished, stopping)
15   begin
16     case state is
17
18       when IDLE =>
19         if (play = '1' and music_finished = '1') then   -- ensure previous file is
   still not playing
20           next_state <= OPEN_ST;
21         else
22           next_state <= IDLE;
23         end if;
24
25       when OPEN_ST =>
26         if (open_done = '1') then
27           next_state <= DEC_RESET;
28         else
29           next_state <= OPEN_ST;
30         end if;
31
32       when DEC_RESET =>
33         if (dec_rst_done = '1' and stopping = '1') then
34           next_state <= IDLE;
35         elsif (dec_rst_done = '1') then
36           next_state <= PLAY_ST;
37         else
38           next_state <= DEC_RESET;
39         end if;
40
41       when PLAY_ST =>
42         if (file_finished = '1') then
43           next_state <= IDLE;
44         elsif (pause = '1') then
45           next_state <= PAUSE_ST;
46         elsif (stop = '1') then
47           next_state <= STOP_ST;
48         else
49           next_state <= PLAY_ST;
50         end if;
51
52       when PAUSE_ST =>
53         if (play = '1') then
54           next_state <= PLAY_ST;
55         elsif (stop = '1') then
56           next_state <= STOP_ST;
```

```
57          else
58            next_state <= PAUSE_ST;
59          end if;
60
61        when STOP_ST =>
62            next_state <= DEC_RESET;
63
64        when others =>
65            next_state <= IDLE;
66      end case;
67    end process;
68
```

```
 1 -------------------------------------------------------------------------------
 2 -- MONITOR FSM
 3 -------------------------------------------------------------------------------
 4   state_register: process (clk, reset)
 5   begin
 6     if (reset = reset_state) then
 7       state <= IDLE;
 8     elsif (clk'event and clk = clk_polarity) then
 9       state <= next_state;
10     end if;
11   end process;
12
13   next_state_comb_logic: process (state, dbuf_wr_en, fetch_en, read_param_done,
   read_done,
14                                   file_finished_s, remain_num_dword,
   total_dword_cnt,
15                                   seek_req, seek_param_done, seek_cmd_done,
   seek_cmd_val)
16   begin
17     case state is
18
19       when IDLE =>
20         if (file_finished_s = '1') then
21           next_state <= IDLE;
22         elsif (dbuf_wr_en = '1') then
23           next_state <= SEEK_CHECK;
24         else
25           next_state <= IDLE;
26         end if;
27
28       when SEEK_CHECK =>
29         if (fetch_en = '0') then   -- if stop command
30           next_state <= IDLE;
31         elsif ( seek_req = '1' and
32                 ( (seek_cmd_val = FIO_FFSEEK and remain_num_dword >
   SEEK_DWORD_MAX) or        -- if enough leg room to seek-fwd
33                   (seek_cmd_val = FIO_BFSEEK and total_dword_cnt > SEEK_DWORD_MAX)
    ) ) then  -- if enough head room to seek-bkw
34           next_state <= SEEK_PARAM;
35         elsif (dbuf_wr_en = '1') then
36           next_state <= READ_PARAM;
37         else
38           next_state <= SEEK_CHECK;
39         end if;
40
41       when READ_PARAM =>
42         if (fetch_en = '0') then   -- if stop command
43           next_state <= IDLE;
44         elsif (read_param_done = '1') then
45           next_state <= READ_CMD;
46         else
47           next_state <= READ_PARAM;
48         end if;
49
50       when READ_CMD =>
51         if (fetch_en = '0') then   -- if stop command
52           next_state <= IDLE;
53         elsif (file_finished_s = '1') then
```

```
54            next_state <= IDLE;
55          elsif (read_done = '1') then
56            next_state <= SEEK_CHECK;
57          else
58            next_state <= READ_CMD;
59          end if;
60
61        when SEEK_PARAM =>
62          if (seek_param_done = '1') then
63            next_state <= SEEK_CMD;
64          else
65            next_state <= SEEK_PARAM;
66          end if;
67
68        when SEEK_CMD =>
69          if (seek_cmd_done = '1') then
70            next_state <= READ_PARAM;
71          else
72            next_state <= SEEK_CMD;
73          end if;
74
75        when others =>
76            next_state <= IDLE;
77      end case;
78    end process;
79
```