```
//------------------------------------------------------------------------------
// Project       : TcbTree
//------------------------------------------------------------------------------
// File          : rule_table_delete.sv
//------------------------------------------------------------------------------
// Author        : Yao Xin
//------------------------------------------------------------------------------


import tcbtree::*;

module rule_table_delete #(
  parameter RAM_LATENCY = 2,
  parameter A_WIDTH     = RULE_TABLE_ADDR_WIDTH
) (
  input                         clk_i,
  input                         rst_i,

  input  tt_pdata_t             task_i,
  input                         task_valid_i,
  output                        task_ready_o,

  input  rule_ram_data_t        rd_data_i,
  output logic [A_WIDTH-1:0]    rd_addr_o,
  output logic                  rd_en_o,

  output logic [A_WIDTH-1:0]    wr_addr_o,
  output rule_ram_data_t        wr_data_o,
  output logic                  wr_en_o,

  output  [A_WIDTH-1:0]         add_rule_empty_addr_o,
  output                        add_rule_empty_addr_en_o,

  node_table_if.master          node_table_if,

  output tt_result_t            result_o,
  output logic                  result_valid_o,
  input                         result_ready_i
);

enum int unsigned {
  IDLE_S,
  NO_VALID_RULE_ADDR_S,
  READ_HEAD_S,
  GO_ON_CHAIN_S,
  IN_TAIL_WITHOUT_MATCH_S,
  RULE_MATCH_IN_MIDDLE_S,
  RULE_MATCH_IN_TAIL_S,
  CLEAR_RAM_AND_ADDR_S,
  UPD_NODE_TABLE_LEAF_S,
  RESULT_OUTPUT_S
} state, next_state, state_d1;

tt_pdata_t                      task_locked;
logic                           rule_match;
logic                           single_rule;
logic                           got_tail;
logic [A_WIDTH-1:0]             rd_addr;
rule_ram_data_t                 prev_rd_data;
rule_ram_data_t                 prev_prev_rd_data;
logic [A_WIDTH-1:0]             prev_rd_addr;
```

```
logic                           rd_data_val;
logic                           rd_data_val_d1;
logic                           state_first_tick;

logic [A_WIDTH-1:0]             node_table_wr_addr;
node_ram_data_t                 node_table_wr_data;
logic                           node_table_wr_data_val;
logic                           node_table_wr_en;

rd_data_val_counter #(
  .RAM_LATENCY                  ( RAM_LATENCY  )
) rd_data_val_counter (
  .clk_i                        ( clk_i        ),
  .rst_i                        ( rst_i        ),

  .rd_en_i                      ( rd_en_o      ),
  .rd_data_val_o                ( rd_data_val  )

);

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    state <= IDLE_S;
  else
    state <= next_state;

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    state_d1 <= IDLE_S;
  else
    state_d1 <= state;

assign state_first_tick = ( state != state_d1 );


always_comb
  begin
    next_state = state;

    case( state )
      IDLE_S:
        begin
          if( task_valid_i && task_ready_o )
            begin
              if( task_i.no_rule_addr || task_i.inval_rule_node )
                next_state = NO_VALID_RULE_ADDR_S;
              else
                next_state = READ_HEAD_S;
            end
        end

      READ_HEAD_S, GO_ON_CHAIN_S:
        begin
          if( rd_data_val )
            begin
              if( rule_match )
                begin
                  if( state == READ_HEAD_S )
                    next_state = UPD_NODE_TABLE_LEAF_S;
                  else
                    if( got_tail )
```

```
                    next_state = RULE_MATCH_IN_TAIL_S;
                  else
                    next_state = RULE_MATCH_IN_MIDDLE_S;
                end
              else
                if( got_tail )
                  next_state = IN_TAIL_WITHOUT_MATCH_S;
                else
                  next_state = GO_ON_CHAIN_S;
            end
        end

      UPD_NODE_TABLE_LEAF_S:
        begin
          if ( node_table_if.wr_ready && node_table_if.wr_ready )
                next_state = RESULT_OUTPUT_S;
        end

      RULE_MATCH_IN_MIDDLE_S, RULE_MATCH_IN_TAIL_S:
        begin
          next_state = RESULT_OUTPUT_S;
        end

      CLEAR_RAM_AND_ADDR_S, RESULT_OUTPUT_S, NO_VALID_RULE_ADDR_S,
      IN_TAIL_WITHOUT_MATCH_S:
        begin
          if( result_valid_o && result_ready_i )
            next_state = IDLE_S;
        end

      default:
        begin
          next_state = IDLE_S;
        end
    endcase
  end

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    task_locked <= '0;
  else
    if( task_ready_o && task_valid_i )
      task_locked <= task_i;

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    begin
      rd_addr      <= '0;
      prev_rd_addr <= '0;
    end
  else
    if( ( state == IDLE_S ) && ( next_state == READ_HEAD_S ) )
      begin
        rd_addr      <= task_i.found_first_rule_addr;
        prev_rd_addr <= rd_addr;
      end
    else
      if( rd_data_val && ( next_state == GO_ON_CHAIN_S ) )
        begin
          rd_addr      <= rd_data_i.next_addr;
          prev_rd_addr <= rd_addr;
        end
```

```
always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    rd_data_val_d1 <= 1'b0;
  else
    rd_data_val_d1 <= rd_data_val;

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    begin
      prev_rd_data <= '0;
      prev_prev_rd_data <= '0;
    end
  else
    if( rd_data_val )
      begin
        prev_rd_data      <= rd_data_i;
        prev_prev_rd_data <= prev_rd_data;
      end

always_latch
  if( rd_data_val )
    got_tail = ( rd_data_i.next_addr_val == 1'b0  );

always_latch
  if( rd_data_val && ( state == READ_HEAD_S ) )
    single_rule = ( rd_data_i.next_addr_val == 1'b0 );

assign rule_match   = ( task_locked.cmd.range.sa_point[LowDim]  ==
                        rd_data_i.rule_range.sa_point[LowDim] ) &&
                      ( task_locked.cmd.range.sa_point[HighDim] ==
                        rd_data_i.rule_range.sa_point[HighDim] ) &&
                      ( task_locked.cmd.range.da_point[LowDim]  ==
                        rd_data_i.rule_range.da_point[LowDim] ) &&
                      ( task_locked.cmd.range.da_point[HighDim] ==
                        rd_data_i.rule_range.da_point[HighDim] ) &&
                      ( task_locked.cmd.range.sp_point[LowDim]  ==
                        rd_data_i.rule_range.sp_point[LowDim] ) &&
                      ( task_locked.cmd.range.sp_point[HighDim] ==
                        rd_data_i.rule_range.sp_point[HighDim] ) &&
                      ( task_locked.cmd.range.dp_point[LowDim]  ==
                        rd_data_i.rule_range.dp_point[LowDim] ) &&
                      ( task_locked.cmd.range.dp_point[HighDim] ==
                        rd_data_i.rule_range.dp_point[HighDim] ) &&
                      ( task_locked.cmd.range.pro_point[LowDim] ==
                        rd_data_i.rule_range.pro_point[LowDim] ) &&
                      ( task_locked.cmd.range.pro_point[HighDim]==
                        rd_data_i.rule_range.pro_point[HighDim] );

assign task_ready_o = ( state == IDLE_S );
assign rd_en_o      = ( state_first_tick || rd_data_val_d1 ) &&
                      (( state == READ_HEAD_S ) || ( state == GO_ON_CHAIN_S ));
assign rd_addr_o    = rd_addr;
assign wr_en_o      = state_first_tick && ( ( state == RULE_MATCH_IN_MIDDLE_S  ) ||
                                           ( state == RULE_MATCH_IN_TAIL_S    ) ||
                                           ( state == CLEAR_RAM_AND_ADDR_S    ) );

rule_ram_data_t rd_data_locked;

always_ff @( posedge clk_i )
  if( rd_data_val )
    rd_data_locked <= rd_data_i;
```

```
always_comb
  begin
    wr_data_o = prev_prev_rd_data;
    wr_addr_o = 'x;

    case( state )
      CLEAR_RAM_AND_ADDR_S:
        begin
          wr_data_o = '0;
          wr_addr_o = rd_addr;
        end
      RULE_MATCH_IN_MIDDLE_S:
        begin
          wr_data_o.next_addr     = rd_data_locked.next_addr;
          wr_data_o.next_addr_val = rd_data_locked.next_addr_val;
          wr_addr_o               = prev_rd_addr;
        end
      RULE_MATCH_IN_TAIL_S:
        begin
          wr_data_o.next_addr     = '0;
          wr_data_o.next_addr_val = 1'b0;
          wr_addr_o               = prev_rd_addr;
        end
      default:
        begin
          wr_data_o = prev_prev_rd_data;
          wr_addr_o = 'x;
        end
    endcase
  end

// ******* node table update *******

always_ff@( posedge clk_i )
  begin
    if ( state == UPD_NODE_TABLE_LEAF_S )
      begin
        if ( node_table_if.wr_ready && node_table_if.wr_data_val )
          begin
            node_table_wr_en                <= 1'b0;
            node_table_wr_data_val          <= 1'b0;
          end
        else begin
          node_table_wr_en                  <= 1'b1;
          node_table_wr_data_val            <= 1'b1;
        end
      end
    else begin
      node_table_wr_en                      <= 1'b0;
      node_table_wr_data_val                <= 1'b0;
    end
  end

always_ff@( posedge clk_i )
  begin
    if ( state == UPD_NODE_TABLE_LEAF_S )
      begin
        node_table_wr_addr                  <= task_locked.found_leaf_node_addr;
        node_table_wr_data.isleaf           <= 1'b1;
        node_table_wr_data.node_val         <= rd_data_locked.next_addr_val;
        node_table_wr_data.first_rule_addr  <= rd_data_locked.next_addr;
```

```
        node_table_wr_data.nrules           <= task_locked.leaf_rule_num - 1;
      end
    else
      begin
        node_table_wr_addr                  <= 'x;
        node_table_wr_data                  <= '0;
      end
  end

assign node_table_if.wr_addr      = node_table_wr_addr;
assign node_table_if.wr_data      = node_table_wr_data;
assign node_table_if.wr_data_val  = node_table_wr_data_val;
assign node_table_if.wr_en        = node_table_wr_en;



assign add_rule_empty_addr_o     = rd_addr;
assign add_rule_empty_addr_en_o  = state_first_tick && ( state == RESULT_OUTPUT_S );


assign result_o.cmd                   = task_locked.cmd;
assign result_o.rule_id               = task_locked.cmd.rule_id;
assign result_o.found_leaf_node_addr  = task_locked.found_leaf_node_addr;
assign result_o.found_tree_node_addr  = task_locked.found_tree_node_addr;
assign result_o.found_first_rule_addr = task_locked.found_first_rule_addr;
assign result_o.rescode               = (( state == NO_VALID_RULE_ADDR_S )||
                                        ( state == IN_TAIL_WITHOUT_MATCH_S )) ?
                                        DELETE_NOT_SUCCESS_NO_ENTRY : DELETE_SUCCESS;

tt_chain_state_t chain_state;

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    chain_state <= NO_CHAIN;
  else
    if( state_first_tick )
      begin
        case( state )
          NO_VALID_RULE_ADDR_S    : chain_state <= NO_CHAIN;
          IN_TAIL_WITHOUT_MATCH_S : chain_state <= IN_TAIL_NO_MATCH;
          UPD_NODE_TABLE_LEAF_S   : chain_state <= IN_HEAD;
          RULE_MATCH_IN_MIDDLE_S  : chain_state <= IN_MIDDLE;
          RULE_MATCH_IN_TAIL_S    : chain_state <= IN_TAIL;
          default                 : chain_state <= NO_CHAIN;
        endcase
      end

assign result_o.chain_state = chain_state;

assign result_valid_o = ( state == RESULT_OUTPUT_S          ) ||
                        ( state == CLEAR_RAM_AND_ADDR_S     ) ||
                        ( state == NO_VALID_RULE_ADDR_S     ) ||
                        ( state == IN_TAIL_WITHOUT_MATCH_S  );



endmodule
```