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

import kicktree::*;

module rule_table_insert #(
  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,

  input [NODE_ADDR_WIDTH-1:0] node_empty_addr_i,
  input                         node_empty_addr_val_i,
  output logic                  node_empty_addr_rd_ack_o,

  input  [A_WIDTH-1:0]          rule_empty_addr_i,
  input                         rule_empty_addr_val_i,
  output logic                  rule_empty_addr_rd_ack_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,
  READ_HEAD_S,
  GO_ON_CHAIN_S,
  NO_EMPTY_ADDR_S,
  WR_RULE_S,
  UPD_RULE_NEX_ADDR_S,
  UPD_NODE_TABLE_CHILD_S,
  UPD_NODE_TABLE_LEAF_S,
  LEAF_NODE_FULL_S,
  RESULT_OUTPUT_S
} state, next_state, state_d1;
```

```
logic                   no_node_empty_addr;
logic                   no_rule_empty_addr;

tt_pdata_t              task_locked;
logic                   id_match;
logic                   got_tail;
logic [A_WIDTH-1:0]     rd_addr;
logic [1:0]             rd_cnt;

logic                   rd_data_val;
logic                   rd_data_val_d1;
rule_ram_data_t         rd_data_locked1, rd_data_locked2;
logic [A_WIDTH-1:0]     rd_addr_locked1, rd_addr_locked2;
logic                   state_first_tick;

logic                   no_rule_addr;
logic                   inval_rule_node;
logic                   rule_on_head;

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;

rule_ram_data_t         rd_data_locked;

logic [NODE_ADDR_WIDTH-1:0] node_empty_addr_locked;
logic [A_WIDTH-1:0]         rule_empty_addr_locked;

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.leaf_node_full )
              next_state = LEAF_NODE_FULL_S;
            else if( task_i.no_rule_addr || task_i.inval_rule_node)
              next_state = ( no_node_empty_addr || no_rule_empty_addr ) ?
                           NO_EMPTY_ADDR_S : WR_RULE_S;
            else
              next_state = READ_HEAD_S;
          end
      end

    READ_HEAD_S, GO_ON_CHAIN_S:
      begin
        if( rd_data_val )
          begin
            if( id_match || got_tail )
              next_state = ( no_rule_empty_addr ) ? NO_EMPTY_ADDR_S : WR_RULE_S;
            else
              next_state = GO_ON_CHAIN_S;
          end
      end

    WR_RULE_S:
      begin
          if ( rule_on_head )
            next_state = UPD_NODE_TABLE_LEAF_S;
          else
            next_state = UPD_RULE_NEX_ADDR_S;
      end

    UPD_RULE_NEX_ADDR_S:
      begin
          next_state = RESULT_OUTPUT_S;
      end

    UPD_NODE_TABLE_LEAF_S:
      begin
        if ( node_table_if.wr_en && node_table_if.wr_ready )
          begin
            if ( no_rule_addr )
              next_state = UPD_NODE_TABLE_CHILD_S;
            else
              next_state = RESULT_OUTPUT_S;
          end
      end

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

    NO_EMPTY_ADDR_S, LEAF_NODE_FULL_S, RESULT_OUTPUT_S:
      begin
        if( result_valid_o && result_ready_i )
            next_state = IDLE_S;
      end

    default: next_state = IDLE_S;
  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 )
    rd_addr <= '0;
  else
    if( ( state == IDLE_S ) && ( next_state == READ_HEAD_S ) )
      rd_addr <= task_i.found_first_rule_addr;
    else
      if( rd_data_val && ( next_state == GO_ON_CHAIN_S ) )
        rd_addr <= rd_data_i.next_addr;

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 )
  begin
    if( rd_data_val )
      rd_data_locked1 <= rd_data_i;
  end

always_ff@( posedge clk_i )
  begin
    if( rd_data_val )
      rd_data_locked2 <= rd_data_locked1;
  end

always_ff @( posedge clk_i or posedge rst_i )
  if( rst_i )
    rd_cnt <= 1'b0;
  else
    if( state == IDLE_S )
      rd_cnt <= 1'b0;
    else
      if( rd_en_o )
        rd_cnt <= rd_cnt + 1;

always_ff@( posedge clk_i )
  begin
    if( rd_en_o )
      rd_addr_locked1 <= rd_addr_o;
  end

always_ff@( posedge clk_i )
  begin
    if( rd_en_o )
      rd_addr_locked2 <= rd_addr_locked1;
  end

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

```
always_latch
  if( state == WR_RULE_S )
    rule_empty_addr_locked = rule_empty_addr_i;

always_latch
  if( state == UPD_NODE_TABLE_LEAF_S )
    node_empty_addr_locked = node_empty_addr_i;

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

assign id_match           = rd_data_val &&
                            ( task_locked.cmd.rule_id <= rd_data_i.rule_id );
assign no_node_empty_addr = !node_empty_addr_val_i;
assign no_rule_empty_addr = !rule_empty_addr_val_i;
assign rule_on_head       = ( rd_cnt <= 1 ) ;
assign no_rule_addr       = ( task_locked.no_rule_addr == 1);
assign inval_rule_node    = ( task_locked.inval_rule_node == 1);
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 rule_empty_addr_rd_ack_o = ( state == WR_RULE_S );
assign node_empty_addr_rd_ack_o = ( state == UPD_NODE_TABLE_LEAF_S ) &&
                                    no_rule_addr &&
                                  ( node_table_if.wr_en && node_table_if.wr_ready ) ;


always_ff@( posedge clk_i )
  begin
    if ( ( state == WR_RULE_S ) || ( state == UPD_RULE_NEX_ADDR_S ) )
      wr_en_o <= 1;
    else
      wr_en_o <= 0;
  end

always_ff@( posedge clk_i )
  begin
    if ( state == WR_RULE_S )
      begin
        wr_data_o.next_addr     <= {{RULE_ADDR_WIDTH{1'b0}},
                                    rd_addr_locked1[A_WIDTH-1:0]};
        wr_data_o.next_addr_val <= rule_on_head ? ((got_tail || no_rule_addr ||
                                    inval_rule_node) ? 1'b0 : 1'b1 ) :
                                    (got_tail ? 1'b0 : 1'b1);
        wr_data_o.rule_id       <= task_locked.cmd.rule_id;
        wr_data_o.rule_range    <= task_locked.cmd.range;
        wr_addr_o               <= rule_empty_addr_i;
      end

    else if ( state == UPD_RULE_NEX_ADDR_S)
      begin
        wr_data_o.next_addr     <= {{RULE_ADDR_WIDTH{1'b0}},
                                    wr_addr_o[A_WIDTH-1:0]};
        wr_data_o.next_addr_val <= 1'b1;
        wr_data_o.rule_id       <= got_tail ? rd_data_locked1.rule_id :
                                              rd_data_locked2.rule_id;
        wr_data_o.rule_range    <= got_tail ? rd_data_locked1.rule_range :
                                              rd_data_locked2.rule_range;
        wr_addr_o               <= got_tail ? rd_addr_locked1 :
```

```
                                                rd_addr_locked2;
      end

    else
      begin
        wr_data_o = '0;;
        wr_addr_o = 'x;
      end
  end


always_ff@( posedge clk_i )
  begin
    if (( state == UPD_NODE_TABLE_LEAF_S ) || ( state == UPD_NODE_TABLE_CHILD_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                  <= no_rule_addr ? node_empty_addr_i :
                                               task_locked.found_leaf_node_addr;
        node_table_wr_data.isleaf           <= 1'b1;
        node_table_wr_data.node_val         <= 1'b1;
        node_table_wr_data.first_rule_addr  <= rule_empty_addr_locked;
        node_table_wr_data.nrules           <= task_locked.leaf_rule_num + 1;

      end

    else if ( state == UPD_NODE_TABLE_CHILD_S )
      begin
        node_table_wr_addr                  <= task_locked.found_tree_node_addr;
        node_table_wr_data                  <= task_locked.found_tree_node_data;

        case  ( task_locked.child_node_del_ins )
          'd0: node_table_wr_data.child_addr[0] <= node_empty_addr_locked;
          'd1: node_table_wr_data.child_addr[1] <= node_empty_addr_locked;
          'd2: node_table_wr_data.child_addr[2] <= node_empty_addr_locked;
          'd3: node_table_wr_data.child_addr[3] <= node_empty_addr_locked;
        endcase
    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;


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_EMPTY_ADDR_S           : chain_state <= NO_CHAIN;
          LEAF_NODE_FULL_S          : chain_state <= NO_CHAIN;
          UPD_RULE_NEX_ADDR_S       : chain_state <= NOT_IN_HEAD;
          UPD_NODE_TABLE_LEAF_S     : chain_state <= IN_HEAD;
          UPD_NODE_TABLE_CHILD_S    : chain_state <= IN_HEAD;
          default                   : chain_state <= NO_CHAIN;
        endcase
      end


always_comb
  begin
    result_o.cmd                    = task_locked.cmd;
    result_o.rule_id                = task_locked.cmd.rule_id;
    result_o.found_leaf_node_addr   = task_locked.found_leaf_node_addr;
    result_o.found_tree_node_addr   = task_locked.found_tree_node_addr;
    result_o.found_first_rule_addr  = task_locked.found_first_rule_addr;
    result_o.chain_state            = chain_state;

    case( state )
      WR_RULE_S:        result_o.rescode = INSERT_SUCCESS;
      NO_EMPTY_ADDR_S:  result_o.rescode = INSERT_NOT_SUCCESS_TABLE_IS_FULL;
      LEAF_NODE_FULL_S: result_o.rescode = INSERT_NOT_SUCCESS_LEAF_NODE_FULL;
      default:          result_o.rescode = INSERT_SUCCESS;
    endcase
  end

assign result_valid_o = ( state == NO_EMPTY_ADDR_S ) ||
                        ( state == RESULT_OUTPUT_S ) ||
                        ( state == LEAF_NODE_FULL_S );


endmodule
```