```
--------------------------------------------------------------------------------
Report Summary
--------------------------------------------------------------------------------
Product Info
  Name    : VC Static Master Shell
  Version : T-2022.06 -- May 30, 2022

Report Info
  Created : Oct 14, 2023  14:12:01

  TopLevelModule: router_top

--------------------------------------------------------------------------------
Management Summary
--------------------------------------------------------------------------------
Stage             Family      Fatals    Errors  Warnings     Infos
-----             --------  --------  --------  --------  --------
LANGUAGE_CHECK    CODING           0         0         4         1
STRUCTURAL_CHECK  CODING           0         1         1        12
-----             --------  --------  --------  --------  --------
Total                              0         1         5        13


--------------------------------------------------------------------------------
Tree Summary
--------------------------------------------------------------------------------
Severity  Stage             Tag                  Count
--------  -----             -------------------  -----
error     STRUCTURAL_CHECK  STARC05-2.5.1.2          1
warning   LANGUAGE_CHECK    STARC05-2.11.3.1         3
warning   LANGUAGE_CHECK    STARC05-2.5.1.7          1
warning   STRUCTURAL_CHECK  STARC05-1.3.1.3          1
info      LANGUAGE_CHECK    ReportPortInfo-ML        1
info      STRUCTURAL_CHECK  RegInputOutput-ML       12
--------  -----             -------------------  -----
Total                                              19


--------------------------------------------------------------------------------
STARC05-2.5.1.2  (1 error/0 waived)
```

```
--------------------------------------------------------------------------------
Report Summary
--------------------------------------------------------------------------------
Product Info
  Name    : VC Static Master Shell
  Version : T-2022.06 -- May 30, 2022

Report Info
  Created : Oct 14, 2023  15:51:37

  TopLevelModule: router_top

--------------------------------------------------------------------------------
Management Summary
--------------------------------------------------------------------------------
Stage             Family      Fatals    Errors  Warnings     Infos
-----             --------  --------  --------  --------  --------
LANGUAGE_CHECK    CODING           0         0         0         1
STRUCTURAL_CHECK  CODING           0         1         0        13
-----             --------  --------  --------  --------  --------
Total                              0         1         0        14


--------------------------------------------------------------------------------
Tree Summary
--------------------------------------------------------------------------------
Severity  Stage             Tag                  Count
--------  -----             -------------------  -----
error     STRUCTURAL_CHECK  STARC05-2.5.1.2          1
info      LANGUAGE_CHECK    ReportPortInfo-ML        1
info      STRUCTURAL_CHECK  RegInputOutput-ML       13
--------  -----             -------------------  -----
Total                                              15
```

Modified RTL Code:

Router Top:

```
module router_top(input clock, resetn, pkt_valid, read_enb_0, read_enb_1, read_enb_2,
                                    input [7:0]data_in,
                                    output vld_out_0, vld_out_1, vld_out_2, err, busy,
                                    output [7:0]data_out_0, data_out_1, data_out_2);

wire [2:0]write_enb;
wire[7:0]dout;

router_fifo FIFO_0(.clock(clock), .resetn(resetn), .soft_reset(soft_reset_0),
        .lfd_state(lfd_state), .write_enb(write_enb[0]), .data_in(dout), .read_enb(read_enb_0),
        .full(full_0), .empty(empty_0), .data_out(data_out_0));

router_fifo FIFO_1(.clock(clock), .resetn(resetn), .soft_reset(soft_reset_1),
        .lfd_state(lfd_state), .write_enb(write_enb[1]), .data_in(dout), .read_enb(read_enb_1),
        .full(full_1), .empty(empty_1), .data_out(data_out_1));

router_fifo FIFO_2(.clock(clock), .resetn(resetn), .soft_reset(soft_reset_2),
        .lfd_state(lfd_state), .write_enb(write_enb[2]), .data_in(dout), .read_enb(read_enb_2),
        .full(full_2), .empty(empty_2), .data_out(data_out_2));

router_sync SYNC(.clock(clock), .resetn(resetn), .data_in(data_in[1:0]), .detect_add(detect_add),
        .full_0(full_0), .full_1(full_1), .full_2(full_2), .read_enb_0(read_enb_0),
                        .read_enb_1(read_enb_1), .read_enb_2(read_enb_2),
.write_enb_reg(write_enb_reg),
                        .empty_0(empty_0), .empty_1(empty_1), .empty_2(empty_2),
.vld_out_0(vld_out_0), .vld_out_1(vld_out_1), .vld_out_2(vld_out_2),
                        .soft_reset_0(soft_reset_0), .soft_reset_1(soft_reset_1),
.soft_reset_2(soft_reset_2), .write_enb(write_enb), .fifo_full(fifo_full));

router_fsm FSM(.clock(clock), .resetn(resetn), .pkt_valid(pkt_valid),
                        .data_in(data_in[1:0]), .soft_reset_0(soft_reset_0),
.soft_reset_1(soft_reset_1), .soft_reset_2(soft_reset_2),
                        .fifo_full(fifo_full), .fifo_empty_0(empty_0), .fifo_empty_1(empty_1),
.fifo_empty_2(empty_2),
                        .parity_done(parity_done), .low_pkt_valid(low_pkt_valid), .busy(busy),
.rst_int_reg(rst_int_reg),
                        .full_state(full_state), .lfd_state(lfd_state), .laf_state(laf_state),
.ld_state(ld_state),
                        .detect_add(detect_add), .write_enb_reg(write_enb_reg));

router_reg REG(.clock(clock), .resetn(resetn), .pkt_valid(pkt_valid), .data_in(data_in),
                        .dout(dout), .fifo_full(fifo_full), .detect_add(detect_add),
                        .ld_state(ld_state), .laf_state(laf_state), .full_state(full_state),
                        .lfd_state(lfd_state), .rst_int_reg(rst_int_reg), .err(err),
.parity_done(parity_done), .low_pkt_valid(low_pkt_valid));
endmodule
```

Router FIFO

```
module router_fifo (
  input clock,
  input resetn,
  input write_enb,
  input read_enb,
  input soft_reset,
  input [7:0] data_in,
  input lfd_state,
  output reg [7:0] data_out,
  output empty,
  output full
);
integer i;
reg [8:0] mem [0:15];
reg [4:0] wr_pt;
reg [4:0] rd_pt;
reg [6:0] fifo_counter;
reg lfd_state_s;
always@(posedge clock)
begin
 if(!resetn)
 begin
   fifo_counter <= 0;
 end
 else if(soft_reset)
 begin
   fifo_counter <= 0;
 end
 else if(read_enb & ~empty)
 begin
   if(mem[rd_pt[3:0]][8] == 1'b1)
      fifo_counter <= mem[rd_pt[3:0]][7:2] + 1'b1;
   else if(fifo_counter != 0)
      fifo_counter <= fifo_counter - 1'b1;
 end
end

always@(posedge clock)
begin
if(!resetn)
lfd_state_s <= 0;
else
lfd_state_s <= lfd_state;
end
wire w1=(fifo_counter==0 && data_out !=0)?1'b1:1'b0;
always@(posedge clock)
begin
  if(!resetn)
      data_out <= 8'b00000000;
```

```
  else if(soft_reset)
     data_out <= 8'bzzzzzzzz;
          else
          begin
            if(w1)
            data_out <= 8'bz;
            else if(read_enb && ~empty)
            data_out <= mem[rd_pt[3:0]];
          end
end

always@(posedge clock)
begin
 if(!resetn)
   begin
            for(i = 0;i<16;i=i+1) begin
              mem[i] <= 0;
             end
          end
          else if(soft_reset)
          begin
           for(i = 0;i<16;i=i+1) begin
             mem[i] <= 0;
           end
           end
           else
           begin
            if(write_enb && !full)
              {mem[wr_pt[3:0]]}<= {lfd_state_s,data_in};
                    end
                    end

always@(posedge clock)
begin
 if(!resetn) begin
    rd_pt <= 5'b00000;
            wr_pt <= 5'b00000;
            end
 else if(soft_reset) begin
    rd_pt <= 5'b00000;
            wr_pt <= 5'b00000;
            end
          else begin
            if(!full && write_enb)
            wr_pt <= wr_pt + 1;
            else
            wr_pt <= wr_pt;
            if(!empty && read_enb)
            rd_pt <= rd_pt + 1;
```

```
        else
        rd_pt <= rd_pt;
        end
        end

assign full= (wr_pt=={~rd_pt[4],rd_pt[3:0]})?1'b1:1'b0;
assign empty=(wr_pt== rd_pt)?1'b1:1'b0;


endmodule
```

Router FSM

```
module router_fsm(input clock,resetn,pkt_valid,
                                    input [1:0] data_in,
                                    input
fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_reset_1,soft_reset_2,parity_don
e, low_pkt_valid,
                                    output
write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy);


parameter DECODE_ADDRESS = 3'b000, LOAD_FIRST_DATA = 3'b001, LOAD_DATA = 3'b010,
FIFO_FULL_STATE = 3'b011, LOAD_AFTER_FULL = 3'b100, LOAD_PARITY = 3'b101,
CHECK_PARITY_ERROR = 3'b110, WAIT_TILL_EMPTY = 3'b111;
reg [2:0] present_state, next_state;
reg [2:0] addr;
//present_state logic
always@(posedge clock)
begin
if(~resetn)
present_state<= DECODE_ADDRESS;
else
if((soft_reset_0 && data_in==2'b00) || (soft_reset_1 && data_in==2'b01) || (soft_reset_2 &&
data_in==2'b10))
present_state <= DECODE_ADDRESS;
else
present_state <= next_state;
end


//internal variable addr logic
always@(posedge clock)
begin
if(~resetn)
addr<=0;
else if((soft_reset_0 && data_in==2'b00) || (soft_reset_1 && data_in==2'b01) || (soft_reset_2 &&
data_in==2'b10))
addr <= 0;
else if(detect_add)
addr <= data_in;
```

```
end

//next_state logic
always@(*)
begin
next_state = present_state;
begin
case(present_state)
DECODE_ADDRESS : if((pkt_valid && (data_in==0) && fifo_empty_0)||(pkt_valid && (data_in==1)
&& fifo_empty_1) ||(pkt_valid && (data_in==2) && fifo_empty_2))
next_state=LOAD_FIRST_DATA;
else if((pkt_valid && (data_in==0) && ~fifo_empty_0) ||(pkt_valid && (data_in==1) &&
~fifo_empty_1) ||(pkt_valid && (data_in==2) && ~fifo_empty_2))
next_state=WAIT_TILL_EMPTY;
else next_state = DECODE_ADDRESS;
LOAD_FIRST_DATA : next_state = LOAD_DATA;
LOAD_DATA : if(fifo_full)
next_state = FIFO_FULL_STATE;
else if(!fifo_full && !pkt_valid)
next_state = LOAD_PARITY;
else next_state = LOAD_DATA;
default : next_state = DECODE_ADDRESS;
LOAD_PARITY : next_state = CHECK_PARITY_ERROR;
CHECK_PARITY_ERROR : if(!fifo_full)
next_state = DECODE_ADDRESS;
else
next_state = FIFO_FULL_STATE;
FIFO_FULL_STATE : if(fifo_full)
next_state = FIFO_FULL_STATE;
else
next_state = LOAD_AFTER_FULL;
LOAD_AFTER_FULL : if(!parity_done && !low_pkt_valid)
next_state = LOAD_DATA;
else if(!parity_done && low_pkt_valid)
next_state=LOAD_PARITY;
else next_state = DECODE_ADDRESS;
WAIT_TILL_EMPTY : if((fifo_empty_0 && (addr == 0)) || (fifo_empty_1 && (addr ==1)) ||
(fifo_empty_2 && (addr == 2)))
next_state = LOAD_FIRST_DATA;
else next_state = WAIT_TILL_EMPTY;
endcase
end
end
//output signals
assign detect_add = (present_state == DECODE_ADDRESS) ? 1'b1 : 1'b0;
assign lfd_state =(present_state==LOAD_FIRST_DATA) ? 1'b1 : 1'b0;
assign ld_state=(present_state==LOAD_DATA) ? 1'b1 : 1'b0;
assign full_state=(present_state==FIFO_FULL_STATE) ? 1'b1 : 1'b0;
assign laf_state=(present_state==LOAD_AFTER_FULL) ? 1'b1 : 1'b0;
assign rst_int_reg=(present_state==CHECK_PARITY_ERROR) ? 1'b1 : 1'b0;
```

```
assign
write_enb_reg=((present_state==LOAD_DATA)||(present_state==LOAD_AFTER_FULL)||(present_stat
e==LOAD_PARITY)) ? 1'b1 : 1'b0;
assign
busy=((present_state==LOAD_FIRST_DATA)||(present_state==FIFO_FULL_STATE)||(present_state==L
OAD_AFTER_FULL)||(present_state==LOAD_PARITY)||(present_state==CHECK_PARITY_ERROR)||(pr
esent_state==WAIT_TILL_EMPTY)) ? 1'b1 : 1'b0;

endmodule
```

Router Synchronizer

```
module router_sync( input
clock,resetn,detect_add,write_enb_reg,read_enb_0,read_enb_1,read_enb_2,empty_0,empty_1,em
pty_2,full_0,full_1,full_2,
                                        input [1:0]data_in,
                                        output wire vld_out_0,vld_out_1,vld_out_2,
                                        output reg [2:0]write_enb,
                                        output reg fifo_full, soft_reset_0,soft_reset_1,soft_reset_2);

reg [4:0] timer_0, timer_1, timer_2;
 reg [1:0] int_addr_reg;


wire w1=(timer_0==5'd29)?1'b1:1'b0;
wire w2=(timer_1==5'd29)?1'b1:1'b0;
wire w3=(timer_2==5'd29)?1'b1:1'b0;


//timer_0 and soft_reset_0 logic
always@(posedge clock)
 begin
  if(~resetn)
  begin
    timer_0<=0;
   soft_reset_0<=0;
   end
  else if(vld_out_0)
   begin
          if(!read_enb_0)
            begin
                  if(w1==5'd29)
                    begin
                          soft_reset_0<=1'b1;
                          timer_0<=0;
                         end
                  else
                    begin
               begin
                          soft_reset_0<=0;
                          timer_0<=timer_0+1'b1;
```

```
                        end
                    end
            end
end
end



//timer_1 and soft_reset_1 logic
always@(posedge clock)
 begin
  if(~resetn)
  begin
     timer_1<=0;
   soft_reset_1<=0;
   end
  else if(vld_out_1)
   begin
            if(!read_enb_1)
              begin
                    if(w2==5'd29)
                      begin
                              soft_reset_1<=1'b1;
                              timer_1<=0;
                             end
                    else
                      begin
                begin
                              soft_reset_1<=0;
                              timer_1<=timer_1+1'b1;
                             end
                    end
            end
end
end

//timer_2 and soft_reset_2 logic
always@(posedge clock)
 begin
  if(~resetn)
    begin
     timer_2<=0;
     soft_reset_2<=0;
    end
  else if(vld_out_2)
    begin
            if(!read_enb_2)
              begin
                    if(w3==5'd29)
                      begin
                              soft_reset_2<=1'b1;
                              timer_2<=0;
                             end
```

```
                    else
                      begin
                begin
                              soft_reset_2<=0;
                              timer_2<=timer_2+1'b1;
                             end
                  end
          end
end
end


//int_addr_reg logic
always@(posedge clock)
begin
 if (~resetn)
   int_addr_reg<=0;
 else if(detect_add)
   int_addr_reg<=data_in;
end

//write enb logic
always@(*)
 begin
 write_enb=3'b000;
 if(write_enb_reg)
 begin
 case(int_addr_reg)
 2'b00 : write_enb = 3'b001;
2'b01 : write_enb = 3'b010;
2'b10 : write_enb = 3'b100;
default:write_enb = 3'b000;
endcase
end
end

//fifo_full logic
always@(*)
  begin
   case(int_addr_reg)
2'b00 : fifo_full = full_0;
2'b01 : fifo_full = full_1;
2'b10 : fifo_full = full_2;
default:fifo_full = 1'b0;
endcase
end

assign vld_out_0 = ~empty_0;
assign vld_out_1 = ~empty_1;
assign vld_out_2 = ~empty_2;
endmodule
```

Router Register

```
module router_reg(
input clock,
   input resetn,
   input pkt_valid,
   input [7:0] data_in,
   input fifo_full,
   input detect_add,
   input ld_state,
   input laf_state,
   input full_state,
   input lfd_state,
   input rst_int_reg,
   output reg err,
   output reg parity_done,
   output reg low_pkt_valid,
   output reg [7:0] dout
);

   reg [7:0] Header_byte;
   reg [7:0] fifo_full_state_byte;
   reg [7:0] Packet_parity;
   reg [7:0] Internal_parity;

   // dout logic
   always @(posedge clock) begin
      if (~resetn) begin
         dout <= 8'b0;
      end else if (lfd_state) begin
         dout <= Header_byte;
      end else if (ld_state && ~fifo_full) begin
         dout <= data_in;
      end else if (laf_state) begin
         dout <= fifo_full_state_byte;
      end
   end

   // Header_byte and fifo_full_state_byte
   always @(posedge clock) begin
      if (~resetn) begin
         Header_byte <= 8'b0;
         fifo_full_state_byte <= 8'b0;
      end else begin
         if (pkt_valid && detect_add) begin
            Header_byte <= data_in;
         end else if (ld_state && fifo_full) begin
            fifo_full_state_byte <= data_in;
         end
      end
   end
```

```
  // parity_done logic
  always @(posedge clock) begin
    if (~resetn) begin
      parity_done <= 1'b0;
    end else begin
      if (ld_state && ~pkt_valid && ~fifo_full) begin
        parity_done <= 1'b1;
      end else if (laf_state && ~parity_done && low_pkt_valid) begin
        parity_done <= 1'b1;
      end else begin
        if (detect_add) begin
          parity_done <= 1'b0;
        end
      end
    end
  end

  // low_pkt_valid logic
  always @(posedge clock) begin
    if (~resetn) begin
      low_pkt_valid <= 1'b0;
    end else begin
      if (rst_int_reg) begin
        low_pkt_valid <= 1'b0;
      end else if (~pkt_valid && ld_state) begin
        low_pkt_valid <= 1'b1;
      end
    end
  end

  // Packet_parity logic
  always @(posedge clock) begin
    if (~resetn) begin
      Packet_parity <= 8'b0;
    end else if ((ld_state && ~pkt_valid && ~fifo_full) || (laf_state && low_pkt_valid &&
~parity_done)) begin
      Packet_parity <= data_in;
    end else if (~pkt_valid && rst_int_reg) begin
      Packet_parity <= 8'b0;
    end else begin
      if (detect_add) begin
        Packet_parity <= 8'b0;
      end
    end
  end

  // internal_parity
  always @(posedge clock) begin
    if (~resetn) begin
      Internal_parity <= 8'b0;
    end else if (detect_add) begin
```

```
        Internal_parity <=
      8'b0; end else if
      (lfd_state) begin
        Internal_parity <= Header_byte;
      end else if (ld_state && pkt_valid && ~full_state) begin
        Internal_parity <= Internal_parity ^ data_in;
      end else if (~pkt_valid && rst_int_reg) begin
        Internal_parity <= 8'b0;
      e
    n
    d
    e
    n
    d

    // error logic
    always @(posedge clock)
      beginif (~resetn) begin
        err <=
      1'b0; end
      else begin
        if (parity_done == 1'b1 && (Internal_parity != Packet_parity))
          beginerr <= 1'b1;
        end else
          begin
          err <=
          1'b0;
        e
      n
      d
      e
      n
      d
    end

endmodule
```

```
--------------------------------------------------------------------------------
Management Summary
--------------------------------------------------------------------------------
Stage             Family      Fatals    Errors  Warnings     Infos
-----             --------  --------  --------  --------  --------
LANGUAGE_CHECK    CODING           0         8       606         3
STRUCTURAL_CHECK  CODING           0         3         1        47
-----             --------  --------  --------  --------  --------
Total                              0        11       607        50


--------------------------------------------------------------------------------
Tree Summary
--------------------------------------------------------------------------------
Severity  Stage             Tag                  Count
--------  -----             -------------------  -----
error     LANGUAGE_CHECK    STARC05-2.10.3.2a        7
error     LANGUAGE_CHECK    W71                      1
error     STRUCTURAL_CHECK  InferLatch               3
warning   LANGUAGE_CHECK    NoAssignX-ML            17
warning   LANGUAGE_CHECK    STARC05-2.1.4.5         21
warning   LANGUAGE_CHECK    STARC05-2.1.5.3          9
warning   LANGUAGE_CHECK    STARC05-2.10.2.3        13
warning   LANGUAGE_CHECK    STARC05-2.11.3.1         5
warning   LANGUAGE_CHECK    STARC05-2.2.3.3        189 *
warning   LANGUAGE_CHECK    W116                     1
warning   LANGUAGE_CHECK    W224                     9
warning   LANGUAGE_CHECK    W240                     3
warning   LANGUAGE_CHECK    W287a                    4
warning   LANGUAGE_CHECK    W287b                   31
warning   LANGUAGE_CHECK    W415a                  258 *
warning   LANGUAGE_CHECK    W486                     1
warning   LANGUAGE_CHECK    W528                    45
warning   STRUCTURAL_CHECK  STARC05-1.4.3.4          1

pport MobaXterm by subscribing to the professional edition here:  https://mobaxterm.mobatek.net
```

Management Summary

Stage	Family	Fatals	Errors	Warnings	Infos
LANGUAGE_CHECK	CODING	0	0	582	3
STRUCTURAL_CHECK	CODING	0	0	0	47
Total		0	0	582	50

Tree Summary

Severity	Stage	Tag	Count	
warning	LANGUAGE_CHECK	STARC05-2.1.4.5	14	
warning	LANGUAGE_CHECK	STARC05-2.1.5.3	9	
warning	LANGUAGE_CHECK	STARC05-2.10.2.3	13	
warning	LANGUAGE_CHECK	STARC05-2.11.3.1	5	
warning	LANGUAGE_CHECK	STARC05-2.2.3.3	189	*
warning	LANGUAGE_CHECK	W116	1	
warning	LANGUAGE_CHECK	W224	9	
warning	LANGUAGE_CHECK	W240	3	
warning	LANGUAGE_CHECK	W287a	4	
warning	LANGUAGE_CHECK	W287b	31	
warning	LANGUAGE_CHECK	W415a	258	*
warning	LANGUAGE_CHECK	W486	1	
warning	LANGUAGE_CHECK	W528	45	
info	LANGUAGE_CHECK	ReportPortInfo-ML	1	
info	LANGUAGE_CHECK	W240	1	
info	LANGUAGE_CHECK	W528	1	
info	STRUCTURAL_CHECK	RegInputOutput-ML	47	
Total			632	