Week 9 CA – Preprocessing Unit for the SHA-256

Constructing complex architectures in Verilog

E.9.1 Design the architecture of a modulo-64 adder used for performing the sum of a 64-bit input (***x***) and a Verilog constant 64'd32 (***y***). The result will be retained in a 64-bit output (***z***).

Note: A modulo-64 adder does not propagate a carry beyond 6 bits because it operates within the constraints of the modulo-64 arithmetic.

Solution:

```
module adder_64
(
input [63:0] x,
input [63:0] y,
output [63:0] z // 64-bit ouptut
);
assign z = x + y;
endmodule
```

E.9.2 Construct the architecture of a 64-bit register employed to store the message lenght of an initial data packet. The module has a 64-bit input ***d***, the 1-bit signals ***clk***, ***rst***, ***clr***, and ***ld***, and a 64-bit output denoted by ***q***.

Implement a ***paramerized module*** for the ***rgst*** and use a ***sequential always block*** to describe its behavior.

Solution:

```
module rgst # (
        parameter w = 8,
        parameter iv = {w{1'b0}}
                )(
        input clk,
        input rst_b,
        input [w-1:0] d,
        input ld,
        input clr,
        output reg [w-1:0] q     );
always @ ( posedge clk, negedge rst_b) begin
        if(! rst_b)
                q <= iv;
        else if(clr)
                q <= iv;
        else if(ld)
                q <= d;
end
endmodule
```

E.9.3 Implement a 4-bit counter for indexing the 32-bit packet transfer from the multiplexer to the storage element (register file). The ***cntr*** module has the 1-bit signals ***clk***, ***rst***, ***clr***, and ***c-up***, respectively a 4-bit output denoted by ***q***.

Solution:

```
module cntr (
        input clk,
        input rst_b,
        input c_up,
        input clr,
        output reg [3:0] q      );
always @ ( posedge clk, negedge rst_b) begin
        if(! rst_b)
                q <= 4'b0000;
        else if(clr)
                q <= 4'b0000;
        else if(c_up)
                q <= q + 1;
end
endmodule
```

E.9.4 Design a specialized multiplexer unit called ***pkt_mux*** which receives two main data entries ***pkt*** on 32 bits (for message packets) and ***msg_len*** on 64 bits (for the length of the original message). Output ***o*** is controlled by the following selection lines:

• ***pad_pkt*** - provides a 1 bit followed by 31 bits of 0.

• ***zero_pkt*** - provides a zero package.

• ***hi_msgln*** - provides the most significant half of the message length (received at msg_len entry)

• ***lo_mgln*** - provides the least significant half of the message length

Solution:

```
// Variant 1
module pkt_mux(
        input [63:0]msg_len,
        input [31:0]pkt,
        input pad_pkt, zero_pkt, hi_mgln, lo_mgln,
        output [31:0]o
);

assign o = pad_pkt? {1'b1,31'b0} : ( zero_pkt? {32'b0} : (hi_mgln ? msg_len[63:32] :
(lo_mgln?msg_len[31:0] : pkt )));
endmodule

// Variant 2
module pkt_mux(
        input [63:0]msg_len,
        input [31:0]pkt,
        input pad_pkt, zero_pkt, hi_mgln, lo_mgln,
        output reg [31:0]o );
always @(*) begin
if (pad_pkt) o = {1'b1,31'b0} ;
else if (zero_pkt) o = 32'b0;
else if (hi_mgln) o = msg_len[63:32];
else if (lo_mgln) o = msg_len[31:0];
else o = pkt;
end
endmodule
```

E.9.5. Construct the preprocessing unit for the crytpographic application by instantiating the modules implemented in ***E.9.1*** → ***E.9.4***.

```
module preprocessing_unit( input clk, rst_b,
        input [31:0] pkt,
        input [3:0] s,
        input st_pkt, lo_mgln, hi_mgln, zero_pkt, pad_pkt, inc_mgln, clr, c_up,
        output [3:0] ldx,
        output [511:0] blk );
// Wire Declaration
wire [63:0] adder_out, rgst_out;
wire [31:0] mux_out;
wire [3:0] cntr_out;
// Instantiation
adder_64 add ( .x(rgst_out), .y(64'd32), .o(adder_out) );
rgst # ( .w(64) ) register
( .clk(clk), .rst_b(rst_b), .d(adder_out), .clr(clr), .ld(inc_mgln), .q(rgst_out) );
cntr counter ( .clk(clk), .rst(rst), .clr(clr), .c_up(c_up) );
pkt_mux multiplexer ( .pkt(pkt), .zero_pkt(zero_pkt), .pad_pkt(pad_pkt),
.hi_mgln(hi_mgln), .lo_mgln(lo_mgln), .o(mux_out) );
storage_unit blackbox ( .x(cntr_out), .y(mux_out), .z(st_pkt) );
/* The storage element will receive each data packet during 16 clock rounds. These
iterations are indexed by the 4-bit counter whose output is connected to input x (selection
line) of the decoder unit. The enable line is z. For each value of the selection line one
output is activated allowing the data packet (y) to be written in a 32-bit register.*/
assign ldx = cntr_out;
endmodule
```