Chipions’25

Phase 1 mini project

Sequential 8x8 multiplier

Objective:

Build an 8x8 multiplier. The input to the multiplier consists of two 8-bit multiplicands (dataa and datab) and the output from the multiplier is 16-bit product (product8x8_out).

Additional outputs are a done bit (done_flag) and seven signals to drive a 7 segment display (seg_a, seg_b, seg_c, seg_d, seg_e, seg_f & seg_g).

This 8 x 8 multiplier requires four clock cycles to perform the full multiplication. During each cycle, a pair of 4-bit portion of the multiplicands is multiplied by a 4 x 4 multiplier. The multiplication result of these 4-bit slices is then accumulated. At the end of the four cycles (during the 5th cycle), the fully composed 16-bit product can be read at the output. The following equations illustrate the mathematical principles supporting this implementation:

result[15..0] = a[7..0] * b[7..0] =

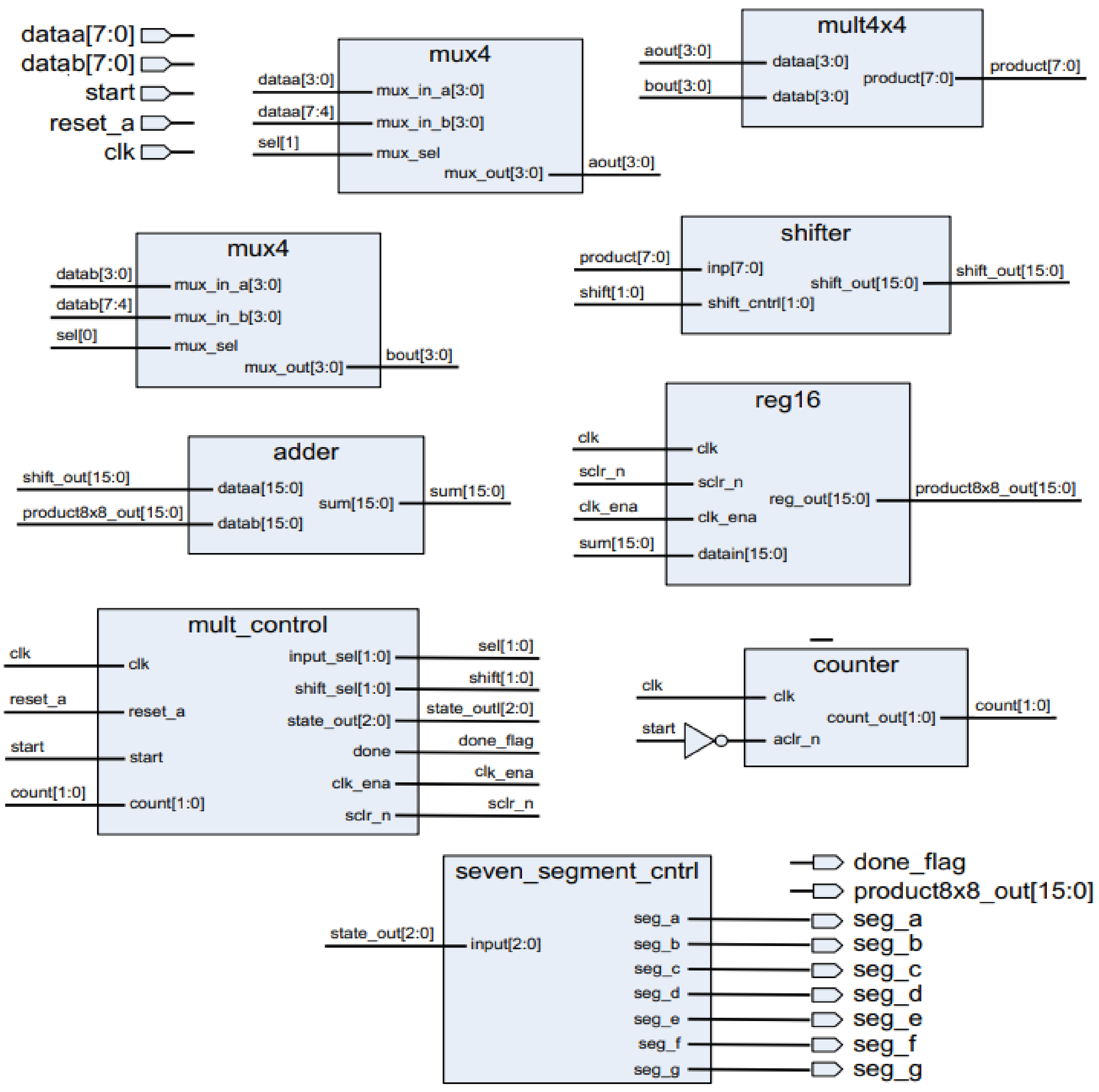
((a[7..4] * 2^4) + a[3..0] * 2^0) * ((b[7..4] * 2^4) + b[3..0] * 2^0) =

((a[7..4] * b[7..4]) * 2^8) +

((a[7..4] * b[3..0]) * 2^4) +

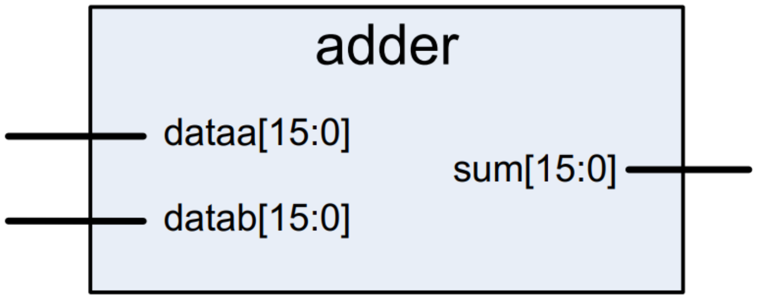
((a[3..0] * b[7..4]) * 2^4)+

((a[3..0] * b[3..0]) * 2^0)



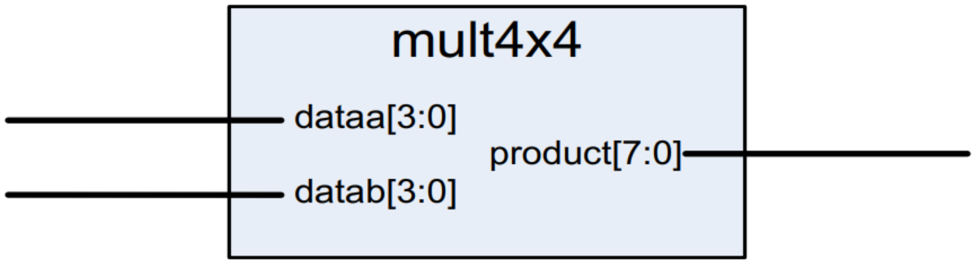
8x8 top level design block diagram

16-bit adder



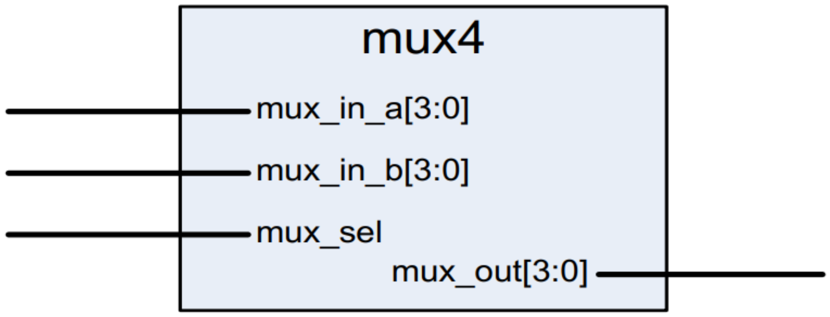
- Write Verilog code to perform addition.
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB adder.v in Quiestasim transcript window).

4x4 multiplier



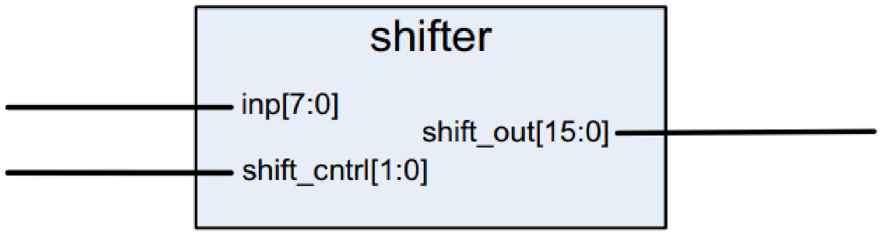
- Write Verilog code to perform multiplication.
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB mult4x4.v in Quiestasim transcript window).

2-1 multiplexer:



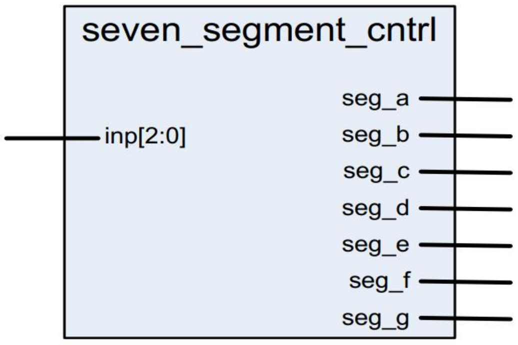
- Write Verilog code to perform 2-1 mux.
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB mux4.v in Quiestasim transcript window).

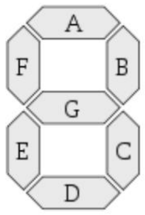
Shifter:



- Write Verilog code to perform 8-bit to 16-bit left shifter (hint: see equation in page 1)
- If shift_cntrl = 0 or 3, then no shift
- If shift_cntrl =1, then 4-bit shift left
- If shift_cntrl = 2, then 8-bit shift left
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB shifter.v in Quiestasim transcript window).

7-segment display encoder:



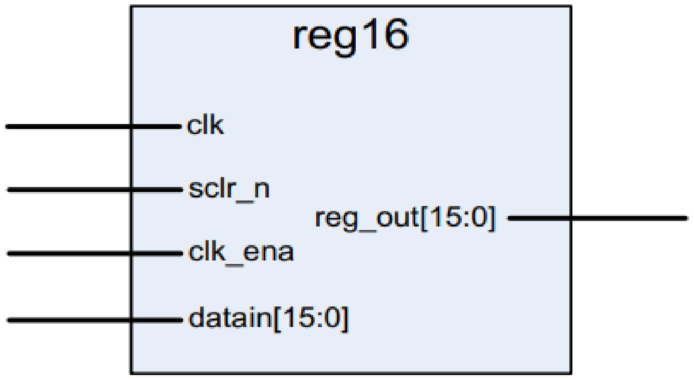


- Write Verilog code to perform encoder as following table:

Inputs	Outputs							LED Display
inp [2:0]	**seg_a**	**seg_b**	**seg_c**	**seg_d**	**seg_e**	**seg_f**	**seg_g**	
000	1	1	1	1	1	1	0	0
001	0	1	1	0	0	0	0	1
010	1	1	0	1	1	0	1	2
011	1	1	1	1	0	0	1	3
All other values	1	0	0	1	1	1	1	E

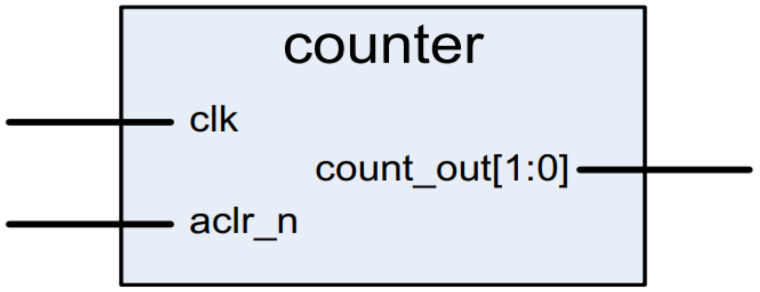
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB seven_segment_cntrl.v in Quiestasim transcript window).

Synchronous 16-bit register:



- Write Verilog code to perform synchronous 16-bit register where sclr_n is a reset signal and clk_ena is a clock enable signal.
- If clk_ena is high and sclr_n is low then the output of register is cleared.
- If clk_ena is high and sclr_n is high then output of register is set equal to its input.
- If clk_ena is low then do nothing.
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB reg16.v in Quiestasim transcript window).

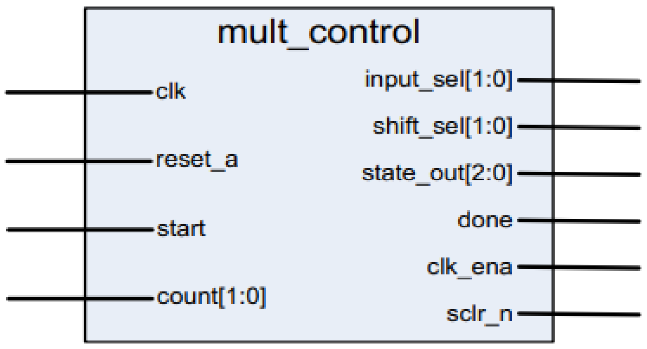
2-bit Counter with asynchronous control:



- Write Verilog code to perform 2-bit counter where aclr_n is a reset signal
- If aclr_n is low, then counter goes to 00 immediately
- If aclr_n is high, output of counter increments by 1 on every rising edge clock.
- Write its testbench.
- Put screenshots of testbench results and circuit schematic

(hint: to show schematic in Quiestasim, type vsim -debugDB counter.v in Quiestasim transcript window).

Multiplier controller



This state machine will manage all the operations that occur within the 8x8 multiplier using 6 defined states: **idle, lsb, mid, msb, calc_done, and err.**

The state machine in the **LSB** state multiplies the lowest 4 bits of the two 8-bit multiplicands ((a[3..0] * b[3..0]) * 2^0). This intermediate result is saved in an accumulator.

The state machine in the **MID** state performs cross multiplication ((a[3..0] * b[7..4]) * 2^4) and ((a[7..4] * b[3..0]) * 2^4). This is done in successive clock cycles.

The products of both multiply operations are added to the content of the accumulator as they are completed and clocked back into the accumulator.

The state machine in the **MSB** state multiplies the highest 4 bits of the two 8-bit multiplicands ((a[7..4] * b[7..4]) * 2^8).

This product is added with the content of the accumulator and clocked back into the accumulator. This result is the final product:

result[15..0] = a[7..0] * b[7..0] = ((a[7..4] * b[7..4]) * 2^8) +

((a[7..4] * b[3..0]) * 2^4) +

((a[3..0] * b[7..4]) * 2^4) +

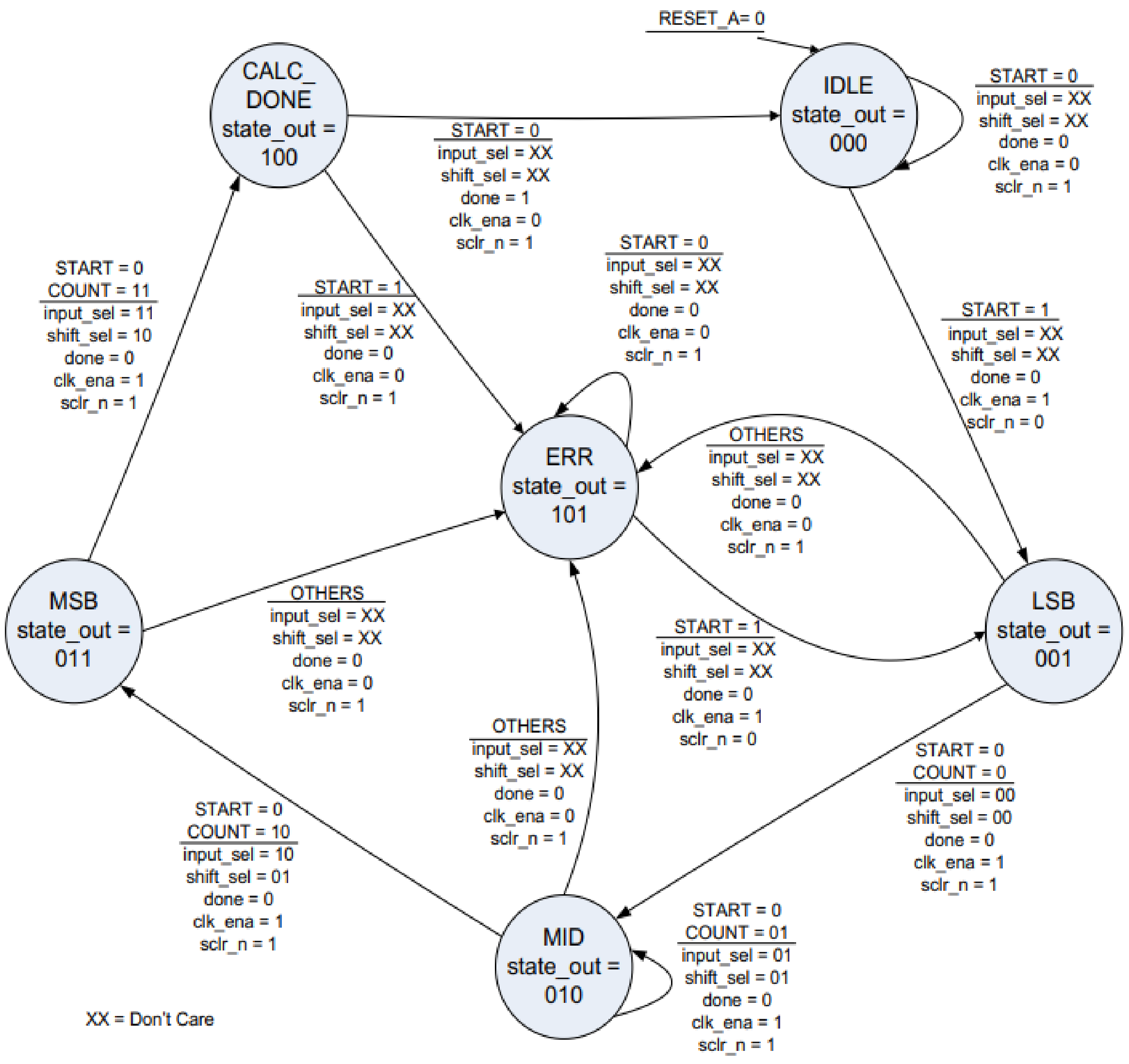
((a[3..0] * b[3..0]) * 2^0)

The state machine in the CALC_DONE state asserts the **done_flag** output to indicate the final product has been calculated and is ready for reading by downstream logic.

The state machine in the ERR state indicates incorrect inputs have been received.

There are two inputs to the state machine: start and count. The start signal is asserted for one clock cycle to begin an 8x8 multiply operation on the next clock cycle. The start signal must only be asserted for one clock cycle. The count signal is used by the state machine to track the multiplication cycles.

The outputs of **mult_control** state machine control the other blocks in the design.



RESET_A= 0
IDLE state_out = 000
START = 0 input_sel = XX shift_sel = XX done = 0 clk_ena = 0 sclr_n = 1
CALC_ DONE state_out = 100
START = 0 input_sel = XX shift_sel = XX done = 1 clk_ena = 0 sclr_n = 1
START = 0 input_sel = XX shift_sel = XX done = 0 clk_ena = 0 sclr_n = 1
START = 1 input_sel = XX shift_sel = XX done = 0 clk_ena = 1 sclr_n = 0
START = 0 COUNT = 11 input_sel = 11 shift_sel = 10 done = 0 clk_ena = 1 sclr_n = 1
START = 1 input_sel = XX shift_sel = XX done = 0 clk_ena = 0 sclr_n = 1
ERR state_out = 101
OTHERS input_sel = XX shift_sel = XX done = 0 clk_ena = 0 sclr_n = 1
MSB state_out = 011
OTHERS input_sel = XX shift_sel = XX done = 0 clk_ena = 0 sclr_n = 1
LSB state_out = 001
START = 1 input_sel = XX shift_sel = XX done = 0 clk_ena = 1 sclr_n = 0
OTHERS input_sel = XX shift_sel = XX done = 0 clk_ena = 0 sclr_n = 1
START = 0 COUNT = 0 input_sel = 00 shift_sel = 00 done = 0 clk_ena = 1 sclr_n = 1
START = 0 COUNT = 10 input_sel = 10 shift_sel = 01 done = 0 clk_ena = 1 sclr_n = 1
MID state_out = 010
START = 0 COUNT = 01 input_sel = 01 shift_sel = 01 done = 0 clk_ena = 1 sclr_n = 1
XX = Don't Care

- Write_Verilog code to perform this state machine.
- Write its testbench.
- Put screenshots of testbench results and circuit schematic
  (hint: to show schematic in Quiestasim, type vsim -debugDB mult_control.v in Quiestasim transcript window).

Finally, put all together a top-level design as shown in page 2 using instantiations required.

GOOD LUCK!