

Brought to you by **DOULOS**

Languages & Libraries

Tools & Simulators

Icarus Verilog 0.9.7

Compile Options

Run Options

Open EPWave after run

Show output file after run

Download files after run

Examples

Community

Collaborate

Forum

Follow @edaplayground

206

```
design.sv
1 // Code your design here
2 // deep code risc 8bit processor
3 timescale 1ns / 1ps
4 module RISC_SPM (clk, rst);
5     parameter word_size = 8;
6     parameter Sel1_size = 3;
7     parameter Sel2_size = 2;
8     wire [Sel1_size-1:0] Sel_Bus1_Mux;
9     wire [Sel2_size-1:0] Sel_Bus2_Mux;
10
11     input clk, rst;
12
13     // Data Nets
14     wire zero;
15     wire [word_size-1:0] instruction, address, Bus_1, mem_word;
16
17     // Control Nets
18     wire Load_R0, Load_R1, Load_R2, Load_R3, Load_PC, Inc_PC, Load_IR;
19     wire Load_Add_R, Load_Reg_Y, Load_Reg_Z;
20     wire write;
21
22     Processing_Unit M0_Processor (instruction, zero, address, Bus_1, mem_word, Load_R0, Load_R1,
23     Load_R2, Load_R3, Load_PC, Inc_PC, Sel_Bus1_Mux, Load_IR, Load_Add_R, Load_Reg_Y,
24     Load_Reg_Z, Sel_Bus2_Mux, clk, rst);
25
26     Control_Unit M1_Controller (Load_R0, Load_R1, Load_R2, Load_R3, Load_PC, Inc_PC,
27     Sel_Bus1_Mux, Sel_Bus2_Mux, Load_IR, Load_Add_R, Load_Reg_Y, Load_Reg_Z,
28     write, instruction, zero, clk, rst);
29
30     Memory_Unit M2_SRAM (
31     .data_out(mem_word),
32     .data_in(Bus_1),
33     .address(address),
34     .clk(clk),
35     .write(write));
36 endmodule
37
38 module Processing_Unit (instruction, Zflag, address, Bus_1, mem_word, Load_R0, Load_R1, Load_R2,
39 Load_R3, Load_PC, Inc_PC, Sel_Bus1_Mux, Load_IR, Load_Add_R, Load_Reg_Y, Load_Reg_Z,
40 Sel_Bus2_Mux, clk, rst);
41
42     parameter word_size = 8;
43     parameter op_size = 4;
44     parameter Sel1_size = 3;
45     parameter Sel2_size = 2;
46
47     output [word_size-1:0] instruction, address, Bus_1;
48     output Zflag;
49
50     input [word_size-1:0] mem_word;
51     input Load_R0, Load_R1, Load_R2, Load_R3, Load_PC, Inc_PC;
52     input [Sel1_size-1:0] Sel_Bus1_Mux;
53     input [Sel2_size-1:0] Sel_Bus2_Mux;
54     input Load_IR, Load_Add_R, Load_Reg_Y, Load_Reg_Z;
55     input clk, rst;
56
57     wire Load_R0, Load_R1, Load_R2, Load_R3;
58     wire [word_size-1:0] Bus_2;
59     wire [word_size-1:0] R0_out, R1_out, R2_out, R3_out;
60     wire [word_size-1:0] PC_count, Y_value, alu_out;
61     wire alu_zero_flag;
62     wire [op_size-1:0] opcode = instruction[word_size-1:word_size-op_size];
63
64     Register_Unit R0 (R0_out, Bus_2, Load_R0, clk, rst);
65     Register_Unit R1 (R1_out, Bus_2, Load_R1, clk, rst);
66     Register_Unit R2 (R2_out, Bus_2, Load_R2, clk, rst);
67     Register_Unit R3 (R3_out, Bus_2, Load_R3, clk, rst);
68     Register_Unit Reg_Y (Y_value, Bus_2, Load_Reg_Y, clk, rst);
69     D_Flop Reg_Z (Zflag, alu_zero_flag, Load_Reg_Z, clk, rst);
70     Address_Register Addr_R (address, Bus_2, Load_Add_R, clk, rst);
71     Instruction_Register IR (instruction, Bus_2, Load_IR, clk, rst);
72     Program_Counter PC (PC_count, Bus_2, Load_PC, Inc_PC, clk, rst);
73
74     Address_Register Addr_R (address, Bus_2, Load_Add_R, clk, rst);
75     Instruction_Register IR (instruction, Bus_2, Load_IR, clk, rst);
76     Program_Counter PC (PC_count, Bus_2, Load_PC, Inc_PC, clk, rst);
77     Multiplexer_3ch Mux_1 (Bus_1, R0_out, R1_out, R2_out, R3_out, PC_count, Sel_Bus1_Mux);
78     Multiplexer_3ch Mux_2 (Bus_2, alu_out, Bus_1, mem_word, Sel_Bus2_Mux);
79     ALU_RISC ALU (alu_zero_flag, alu_out, Y_value, Bus_1, opcode);
80 endmodule
81
82 module Clock_Unit (clock);
83 output clock;
84 reg clock;
85 parameter delay = 0;
86 parameter half_cycle = 10;
87 initial begin
88     #delay clock = 0;
89     forever #half_cycle clock = ~clock;
90 end
91 endmodule
92
93 module Register_Unit (data_out, data_in, load, clk, rst);
94     parameter word_size = 8;
95     output [word_size-1:0] data_out;
96     input [word_size-1:0] data_in;
97     input load;
98     input clk, rst;
99     reg [word_size-1:0] data_out;
100
101     always @ (posedge clk or negedge rst)
102     if (rst == 0) data_out <= 0; else if (load) data_out <= data_in;
103 endmodule
104
105 module Address_Register (data_out, data_in, load, clk, rst);
106     parameter word_size = 8;
107     output [word_size-1:0] data_out;
108     input [word_size-1:0] data_in;
109     input load, clk, rst;
110     reg [word_size-1:0] data_out;
111     always @ (posedge clk or negedge rst)
112     if (rst == 0) data_out <= 0; else if (load) data_out <= data_in;
113 endmodule
114
115 module Instruction_Register (data_out, data_in, load, clk, rst);
116     parameter word_size = 8;
117     output [word_size-1:0] data_out;
118     input [word_size-1:0] data_in;
119     input load, clk, rst;
120     reg [word_size-1:0] data_out;
121     always @ (posedge clk or negedge rst)
122     if (rst == 0) data_out <= 0; else if (load) data_out <= data_in;
123 endmodule
124
125 module Program_Counter (count, data_in, Load_PC, Inc_PC, clk, rst);
126     parameter word_size = 8;
127     input [word_size-1:0] data_in;
128     input Load_PC, Inc_PC;
129     input clk, rst;
130     reg [word_size-1:0] count;
131     always @ (posedge clk or negedge rst)
132     if (rst == 0) count <= 0; else if (Load_PC) count <= data_in; else if (Inc_PC) count <= count + 1;
133 endmodule
134
135 module Multiplexer_3ch (mux_out, data_a, data_b, data_c, data_d, data_e, sel);
136     parameter word_size = 8;
137     output [word_size-1:0] mux_out;
138     input [word_size-1:0] data_a, data_b, data_c, data_d, data_e;
139     input [2:0] sel;
140
141     assign mux_out = (sel == 0) ? data_a: (sel == 1)
142     ? data_b : (sel == 2)
143     ? data_c: (sel == 3)
144     ? data_d : (sel == 4)
145     ? data_e : 'bx;
146 endmodule
147
148 module Multiplexer_3ch (mux_out, data_a, data_b, data_c, sel);
149     parameter word_size = 8;
150     output [word_size-1:0] mux_out;
151     input [word_size-1:0] data_a, data_b, data_c;
152     input [1:0] sel;
153
154     assign mux_out = (sel == 0) ? data_a: (sel == 1) ? data_b : (sel == 2) ? data_c: 'bx;
155 endmodule
156
157 module ALU_RISC (alu_zero_flag, alu_out, data_1, data_2, sel);
158     parameter word_size = 8;
159     parameter op_size = 4;
160
161     // Opcodes
162     parameter NOP = 4'b0000;
163     parameter ADD = 4'b0001;
164     parameter SUB = 4'b0010;
165     parameter AND = 4'b0011;
166     parameter NOT = 4'b0100;
167     parameter RD = 4'b0101;
168     parameter WR = 4'b0110;
169     parameter BR = 4'b0111;
170     parameter BRZ = 4'b1000;
171
172     output alu_zero_flag;
173     output [word_size-1:0] alu_out;
174     input [word_size-1:0] data_1, data_2;
175     input [op_size-1:0] sel;
176     reg [word_size-1:0] alu_out;
177
178     assign alu_zero_flag = ~(alu_out);
179     always @ (sel or data_1 or data_2)
180     case (sel)
181     NOP: alu_out = 0;
182     ADD: alu_out = data_1 + data_2; // Reg_Y + Bus_1
183     SUB: alu_out = data_1 - data_2;
184     AND: alu_out = data_1 & data_2;
185     NOT: alu_out = ~data_2; // Gets data from Bus_1
186     default: alu_out = 0;
187     endcase
188 endmodule
189
190 module Control_Unit (
191 Load_R0, Load_R1,
192 Load_R2, Load_R3,
193 Load_PC, Inc_PC,
194 Sel_Bus1_Mux, Sel_Bus2_Mux,
195 Load_IR, Load_Add_R, Load_Reg_Y, Load_Reg_Z,
196 write, instruction, zero, clk, rst);
197
198     parameter word_size = 8, op_size = 4, state_size = 4;
199     parameter src_size = 2, dest_size = 2, Sel1_size = 3, Sel2_size = 2;
200
201     // State Codes
202     parameter S_idle = 0, S_fet1 = 1, S_fet2 = 2, S_dec = 3;
203     parameter S_ex1 = 4, S_rd1 = 5, S_rd2 = 6;
204     parameter S_wr1 = 7, S_wr2 = 8, S_br1 = 9, S_br2 = 10, S_halt = 11;
205
206     // Opcodes
207     parameter NOP = 0, ADD = 1, SUB = 2, AND = 3, NOT = 4;
208     parameter RD = 5, WR = 6, BR = 7, BRZ = 8;
209
210     // Source and Destination Codes
211     parameter RD = 0, R1 = 1, R2 = 2, R3 = 3;
212
213     output Load_R0, Load_R1, Load_R2, Load_R3;
214     output Load_PC, Inc_PC;
215     output [Sel1_size-1:0] Sel_Bus1_Mux;
216     output Load_IR, Load_Add_R;
217     output Load_Reg_Y, Load_Reg_Z;
218     output [Sel2_size-1:0] Sel_Bus2_Mux;
219
220     input [word_size-1:0] instruction;
221     input zero;
222     input clk, rst;
223
224     reg [state_size-1:0] state, next_state;
225     reg Load_R0, Load_R1, Load_R2, Load_R3, Load_PC, Inc_PC;
226     reg Load_IR, Load_Add_R, Load_Reg_Y;
227     reg Sel_ALU, Sel_Bus1, Sel_Mem;
228     reg Sel_R0, Sel_R1, Sel_R2, Sel_R3, Sel_PC;
229     reg Load_Reg_Z, write;
230     reg err_flag;
231
232     wire [op_size-1:0] opcode = instruction[word_size-1:word_size-op_size];
233     wire [src_size-1:0] src = instruction[src_size+dest_size-1:dest_size];
234     wire [dest_size-1:0] dest = instruction[dest_size-1:0];
235
236     // Mux selectors
237     assign Sel_Bus1_Mux[Sel1_size-1:0] = Sel_R0 ? 0:
238     Sel_R1 ? 1:
239     Sel_R2 ? 2:
240     Sel_R3 ? 3:
241     Sel_PC ? 4: 'bx; // 3-bits, sized number
242
243     assign Sel_Bus2_Mux[Sel2_size-1:0] = Sel_ALU ? 0:
244     Sel_Bus1 ? 1:
245     Sel_Mem ? 2: 'bx;
246
247     always @ (posedge clk or negedge rst) begin: State_transitions
248     if (rst == 0) state <= S_idle; else state <= next_state; end
249
250     /* always @ (state or instruction or zero) begin: Output_and_next_state
251
252     Note: The above event control expression leads to incorrect operation. The state transition causes the activity to be evaluated once, then the
253     opcode had before the state change, which results in Sel_PC = 0 in state 3, which will cause a return to state 1 at the next clock. Finally, opcode
254     is changed, but this does not trigger a re-evaluation because it is not in the event control expression. So, the caution is to be sure to use opcode
255     in the event control expression. That way, the final execution of the behavior uses the value of opcode that results from the state change, and leads
256     to the correct value of Sel_PC.
257     */
258
259     always @ (state or opcode or zero) begin: Output_and_next_state
260     Sel_R0 = 0; Sel_R1 = 0; Sel_R2 = 0; Sel_R3 = 0; Sel_PC = 0;
261     Load_R0 = 0; Load_R1 = 0; Load_R2 = 0; Load_R3 = 0; Load_PC = 0;
262     Inc_PC = 0; Load_Add_R = 0; Load_Reg_Y = 0; Load_Reg_Z = 0;
263     Sel_Bus1 = 0;
264     Sel_Bus2 = 0;
265     Sel_ALU = 0;
266     Sel_Mem = 0;
267     write = 0;
268     err_flag = 0; // Used for de-bug in simulation
269     next_state = state;
270
271     case (state)
272     S_idle:
273         next_state = S_fet1;
274
275     S_fet1: begin
276         next_state = S_fet2;
277         Sel_PC = 1;
278         Sel_Bus1 = 1;
279         Load_Add_R = 1;
280         end
281
282     S_fet2: begin
283         next_state = S_dec;
284         Load_R0 = 1;
285         Inc_PC = 1;
286         end
287
288     S_dec:
289         case (opcode)
290         NOP: next_state = S_fet1;
291         ADD, SUB, AND: begin
292             next_state = S_ex1;
293             Sel_Bus1 = 1;
294             Load_Reg_Y = 1;
295             case (src)
296             R0: Sel_R0 = 1;
297             R1: Sel_R1 = 1;
298             R2: Sel_R2 = 1;
299             R3: Sel_R3 = 1;
300             default: err_flag = 1;
301             endcase
302             end // ADD, SUB, AND
303
304         NOT: begin
305             next_state = S_fet1;
306             Load_Reg_Z = 1;
307             Sel_Bus1 = 1;
308             Sel_ALU = 1;
309             case (src)
310             R0: Sel_R0 = 1;
311             R1: Sel_R1 = 1;
312             R2: Sel_R2 = 1;
313             R3: Sel_R3 = 1;
314             default: err_flag = 1;
315             endcase
316             end // NOT
317
318         RD: begin
319             next_state = S_rd1;
320             Sel_PC = 1; Sel_Bus1 = 1; Load_Add_R = 1;
321             end // RD
322
323         WR: begin
324             next_state = S_wr1;
325             Sel_PC = 1; Sel_Bus1 = 1; Load_Add_R = 1;
326             end // WR
327
328         BR: begin
329             next_state = S_br1;
330             Sel_PC = 1; Sel_Bus1 = 1; Load_Add_R = 1;
331             end // BR
332
333         BRZ: if (zero == 1)
334             begin
335                 next_state = S_br1;
336                 Sel_PC = 1; Sel_Bus1 = 1; Load_Add_R = 1;
337             end // BRZ
338         else
339             begin
340                 next_state = S_fet1;
341                 Inc_PC = 1;
342             end
343         end
344     default:
345         next_state = S_idle;
346     endcase // (opcode)
347
348     S_ex1: begin
349         next_state = S_fet1;
350         Load_Reg_Z = 1;
351         Sel_ALU = 1;
352         case (dest)
353         R0: begin Sel_R0 = 1; Load_R0 = 1; end
354         R1: begin Sel_R1 = 1; Load_R1 = 1; end
355         R2: begin Sel_R2 = 1; Load_R2 = 1; end
356         R3: begin Sel_R3 = 1; Load_R3 = 1; end
357         default: err_flag = 1;
358         endcase
359     end
360
361     S_rd1: begin
362         next_state = S_rd2;
363         Sel_Mem = 1;
364         Load_Add_R = 1;
365         Inc_PC = 1;
366         end
367
368     S_wr1: begin
369         next_state = S_wr2;
370         Sel_Mem = 1;
371         Load_Add_R = 1;
372         Inc_PC = 1;
373         end
374
375     S_rd2: begin
376         next_state = S_fet1;
377         Sel_Mem = 1;
378         case (dest)
379         R0: Load_R0 = 1;
380         R1: Load_R1 = 1;
381         R2: Load_R2 = 1;
382         R3: Load_R3 = 1;
383         default: err_flag = 1;
384         endcase
385     end
386
387     S_wr2: begin
388         next_state = S_fet1;
389         write = 1;
390         case (src)
391         R0: Sel_R0 = 1;
392         R1: Sel_R1 = 1;
393         R2: Sel_R2 = 1;
394         R3: Sel_R3 = 1;
395         default: err_flag = 1;
396         endcase
397     end
398
399     S_br1: begin next_state = S_br2; Sel_Mem = 1; Load_Add_R = 1; end
400     S_br2: begin next_state = S_halt; Sel_Mem = 1; Load_PC = 1; end
401     S_halt: begin next_state = S_fet1; end
402     default: next_state = S_idle;
403 end
404 endmodule
405
406 module Memory_Unit (data_out, data_in, address, clk, write);
407     parameter word_size = 8;
408     parameter memory_size = 256;
409
410     input [word_size-1:0] data_in;
411     input [word_size-1:0] address;
412     input clk, write;
413     reg [word_size-1:0] memory [memory_size-1:0];
414
415     assign data_out = memory[address];
416
417     always @ (posedge clk)
418     if (write) memory[address] = data_in;
419 endmodule
420
421 module Memory_Unit (data_out, data_in, address, clk, write);
422     parameter word_size = 8;
423     parameter memory_size = 256;
424
425     input [word_size-1:0] data_in;
426     input [word_size-1:0] address;
427     input clk, write;
428     reg [word_size-1:0] memory [memory_size-1:0];
429
430     assign data_out = memory[address];
431
432     always @ (posedge clk)
433     if (write) memory[address] = data_in;
434 endmodule
435
436 module Memory_Unit (data_out, data_in, address, clk, write);
437     parameter word_size = 8;
438     parameter memory_size = 256;
439
440     input [word_size-1:0] data_in;
441     input [word_size-1:0] address;
442     input clk, write;
443     reg [word_size-1:0] memory [memory_size-1:0];
444
445     assign data_out = memory[address];
446
447     always @ (posedge clk)
448     if (write) memory[address] = data_in;
449 endmodule
450
451 module Memory_Unit (data_out, data_in, address, clk, write);
452     parameter word_size = 8;
453     parameter memory_size = 256;
454
455     input [word_size-1:0] data_in;
456     input [word_size-1:0] address;
457     input clk, write;
458     reg [word_size-1:0] memory [memory_size-1:0];
459
460     assign data_out = memory[address];
461
462     always @ (posedge clk)
463     if (write) memory[address] = data_in;
464 endmodule
465
466 module Memory_Unit (data_out, data_in, address, clk, write);
467     parameter word_size = 8;
468     parameter memory_size = 256;
469
470     input [word_size-1:0] data_in;
471     input [word_size-1:0] address;
472     input clk, write;
473     reg [word_size-1:0] memory [memory_size-1:0];
474
475     assign data_out = memory[address];
476
477     always @ (posedge clk)
478     if (write) memory[address] = data_in;
479 endmodule
480
481 module Memory_Unit (data_out, data_in, address, clk, write);
482     parameter word_size = 8;
483     parameter memory_size = 256;
484
485     input [word_size-1:0] data_in;
486     input [word_size-1:0] address;
487     input clk, write;
488     reg [word_size-1:0] memory [memory_size-1:0];
489
490     assign data_out = memory[address];
491
492     always @ (posedge clk)
493     if (write) memory[address] = data_in;
494 endmodule
495
496 module Memory_Unit (data_out, data_in, address, clk, write);
497     parameter word_size = 8;
498     parameter memory_size = 256;
499
500     input [word_size-1:0] data_in;
501     input [word_size-1:0] address;
502     input clk, write;
503     reg [word_size-1:0] memory [memory_size-1:0];
504
505     assign data_out = memory[address];
506
507     always @ (posedge clk)
508     if (write) memory[address] = data_in;
509 endmodule
510
511 module Memory_Unit (data_out, data_in, address, clk, write);
512     parameter word_size = 8;
513     parameter memory_size = 256;
514
515     input [word_size-1:0] data_in;
516     input [word_size-1:0] address;
517     input clk, write;
518     reg [word_size-1:0] memory [memory_size-1:0];
519
520     assign data_out = memory[address];
521
522     always @ (posedge clk)
523     if (write) memory[address] = data_in;
524 endmodule
525
526 module Memory_Unit (data_out, data_in, address, clk, write);
527     parameter word_size = 8;
528     parameter memory_size = 256;
529
530     input [word_size-1:0] data_in;
531     input [word_size-1:0] address;
532     input clk, write;
533     reg [word_size-1:0] memory [memory_size-1:0];
534
535     assign data_out = memory[address];
536
537     always @ (posedge clk)
538     if (write) memory[address] = data_in;
539 endmodule
540
541 module Memory_Unit (data_out, data_in, address, clk, write);
542     parameter word_size = 8;
543     parameter memory_size = 256;
544
545     input [word_size-1:0] data_in;
546     input [word_size-1:0] address;
547     input clk, write;
548     reg [word_size-1:0] memory [memory_size-1:0];
549
550     assign data_out = memory[address];
551
552     always @ (posedge clk)
553     if (write) memory[address] = data_in;
554 endmodule
555
556 module Memory_Unit (data_out, data_in, address, clk, write);
557     parameter word_size = 8;
558     parameter memory_size = 256;
559
560     input [word_size-1:0] data_in;
561     input [word_size-1:0] address;
562     input clk, write;
563     reg [word_size-1:0] memory [memory_size-1:0];
564
565     assign data_out = memory[address];
566
567     always @ (posedge clk)
568     if (write) memory[address] = data_in;
569 endmodule
570
571 module Memory_Unit (data_out, data_in, address, clk, write);
572     parameter word_size = 8;
573     parameter memory_size = 256;
574
575     input [word_size-1:0] data_in;
576     input [word_size-1:0] address;
577     input clk, write;
578     reg [word_size-1:0] memory [memory_size-1:0];
579
580     assign data_out = memory[address];
581
582     always @ (posedge clk)
583     if (write) memory[address] = data_in;
584 endmodule
585
586 module Memory_Unit (data_out, data_in, address, clk, write);
587     parameter word_size = 8;
588     parameter memory_size = 256;
589
590     input [word_size-1:0] data_in;
591     input [word_size-1:0] address;
592     input clk, write;
593     reg [word_size-1:0] memory [memory_size-1:0];
594
595     assign data_out = memory[address];
596
597     always @ (posedge clk)
598     if (write) memory[address] = data_in;
599 endmodule
600
601 module Memory_Unit (data_out, data_in, address, clk, write);
602     parameter word_size = 8;
603     parameter memory_size = 256;
604
605     input [word_size-1:0] data_in;
606     input [word_size-1:0] address;
607     input clk, write;
608     reg [word_size-1:0] memory [memory_size-1:0];
609
610     assign data_out = memory[address];
611
612     always @ (posedge clk)
613     if (write) memory[address] = data_in;
614 endmodule
615
616 module Memory_Unit (data_out, data_in, address, clk, write);
617     parameter word_size = 8;
618     parameter memory_size = 256;
619
620     input [word_size-1:0] data_in;
621     input [word_size-1:0] address;
622     input clk, write;
623     reg [word_size-1:0] memory [memory_size-1:0];
624
625     assign data_out = memory[address];
626
627     always @ (posedge clk)
628     if (write) memory[address] = data_in;
629 endmodule
630
631 module Memory_Unit (data_out, data_in, address, clk, write);
632     parameter word_size = 8;
633     parameter memory_size = 256;
634
635     input [word_size-1:0] data_in;
636     input [word_size-1:0] address;
637     input clk, write;
638     reg [word_size-1:0] memory [memory_size-1:0];
639
640     assign data_out = memory[address];
641
642     always @ (posedge clk)
643     if (write) memory[address] = data_in;
644 endmodule
645
646 module Memory_Unit (data_out, data_in, address, clk, write);
647     parameter word_size = 8;
648     parameter memory_size = 256;
649
650     input [word_size-1:0] data_in;
651     input [word_size-1:0] address;
652     input clk, write;
653     reg [word_size-1:0] memory [memory_size-1:0];
654
655     assign data_out = memory[address];
656
657     always @ (posedge clk)
658     if (write) memory[address] = data_in;
659 endmodule
660
661 module Memory_Unit (data_out, data_in, address, clk, write);
662     parameter word_size = 8;
663     parameter memory_size = 256;
664
665     input [word_size-1:0] data_in;
666     input [word_size-1:0] address;
667     input clk, write;
668     reg [word_size-1:0] memory [memory_size-1:0];
669
670     assign data_out = memory[address];
671
672     always @ (posedge clk)
673     if (write) memory[address] = data_in;
674 endmodule
675
676 module Memory_Unit (data_out, data_in, address, clk, write);
677     parameter word_size = 8;
678     parameter memory_size = 256;
679
680     input [word_size-1:0] data_in;
681     input [word_size-1:0] address;
682     input clk, write;
683     reg [word_size-1:0] memory [memory_size-1:0];
684
685     assign data_out = memory[address];
686
687     always @ (posedge clk)
688     if (write) memory[address] = data_in;
689 endmodule
690
691 module Memory_Unit (data_out, data_in, address, clk, write);
692     parameter word_size = 8;
693     parameter memory_size = 256;
694
695     input [word_size-1:0] data_in;
696     input [word_size-1:0] address;
697     input clk, write;
698     reg [word_size-1:0] memory [memory_size-1:0];
699
700     assign data_out = memory[address];
701
702     always @ (posedge clk)
703     if (write) memory[address] = data_in;
704 endmodule
705
706 module Memory_Unit (data_out, data_in, address, clk, write);
707     parameter word_size = 8;
708     parameter memory_size = 256;
709
710     input [word_size-1:0] data_in;
711     input [word_size-1:0] address;
712     input clk, write;
713     reg [word_size-1:0] memory [memory_size-1:0];
714
715     assign data_out = memory[address];
716
717     always @ (posedge clk)
718     if (write) memory[address] = data_in;
719 endmodule
720
721 module Memory_Unit (data_out, data_in, address, clk, write);
722     parameter word_size = 8;
723     parameter memory_size = 256;
724
725     input [word_size-1:0] data_in;
726     input [word_size-1:0] address;
727     input clk, write;
728     reg [word_size-1:0] memory [memory_size-1:0];
729
730     assign data_out = memory[address];
731
732     always @ (posedge clk)
733     if (write) memory[address] = data_in;
734 endmodule
735
736 module Memory_Unit (data_out, data_in, address, clk, write);
737     parameter word_size = 8;
738     parameter memory_size = 256;
739
740     input [word_size-1:0] data_in;
741     input [word_size-1:0] address;
742     input clk, write;
743     reg [word_size-1:0] memory [memory_size-1:0];
744
745     assign data_out = memory[address];
746
747     always @ (posedge clk)
748     if (write) memory[address] = data_in;
749 endmodule
750
751 module Memory_Unit (data_out, data_in, address, clk, write);
752     parameter word_size = 8;
753     parameter memory_size = 256;
754
755     input [word_size-1:0] data_in;
756     input [word_size-1:0] address;
757     input clk, write;
758     reg [word_size-1:0] memory [memory_size-1:0];
759
760     assign data_out = memory[address];
761
762     always @ (posedge clk)
763     if (write) memory[address] = data_in;
764 endmodule
765
766 module Memory_Unit (data_out, data_in, address, clk, write);
767     parameter word_size = 8;
768     parameter memory_size = 256;
769
770     input [word_size-1:0] data_in;
771     input [word_size-1:0] address;
772     input clk, write;
773     reg [word_size-1:0] memory [memory_size-1:0];
774
775     assign data_out = memory[address];
776
777     always @ (posedge clk)
778     if (write) memory[address] = data_in;
779 endmodule
780
781 module Memory_Unit (data_out, data_in, address, clk, write);
782     parameter word_size = 8;
783     parameter memory_size = 256;
784
785     input [word_size-1:0] data_in;
786     input [word_size-1:0] address;
787     input clk, write;
788     reg [word_size-1:0] memory [memory_size-1:0];
789
790     assign data_out = memory[address];
791
792     always @ (posedge clk)
793     if (write) memory[address] = data_in;
794 endmodule
795
796 module Memory_Unit (data_out, data_in, address, clk, write);
797     parameter word_size = 8;
798     parameter memory_size = 256;
799
800     input [word_size-1:0] data_in;
801     input [word_size-1:0] address;
802     input clk, write;
803     reg [word_size-1:0] memory [memory_size-1:0];
804
805     assign data_out = memory[address];
806
807     always @ (posedge clk)
808     if (write) memory[address] = data_in;
809 endmodule
810
811 module Memory_Unit (data_out, data_in, address, clk, write);
812     parameter word_size = 8;
813     parameter memory_size = 256;
814
815     input [word_size-1:0] data_in;
816     input [word_size-1:0] address;
817     input clk, write;
818     reg [word_size-1:0] memory [memory_size-1:0];
819
820     assign data_out = memory[address];
821
822     always @ (posedge clk)
823     if (write) memory[address] = data_in;
824 endmodule
825
826 module Memory_Unit (data_out, data_in, address, clk, write);
827     parameter word_size = 8;
828     parameter memory_size = 256;
829
830     input [word_size-1:0] data_in;
831     input [word_size-1:0] address;
832     input clk, write;
833     reg [word_size-1:0] memory [memory_size-1:0];
834
835     assign data_out = memory[address];
836
837     always @ (posedge clk)
838     if (write) memory[address] = data_in;
839 endmodule
840
841 module Memory_Unit (data_out, data_in, address, clk, write);
842     parameter word_size = 8;
843     parameter memory_size = 256;
844
845     input [word_size-1:0] data_in;
846     input [word_size-1:0] address;
847     input clk, write;
848     reg [word_size-1:0] memory [memory_size-1:0];
849
850     assign data_out = memory[address];
851
852     always @ (posedge clk)
853     if (write) memory[address] = data_in;
854 endmodule
855
856 module Memory_Unit (data_out, data_in, address, clk, write);
857     parameter word_size = 8;
858     parameter memory_size = 256;
859
860     input [word_size-1:0] data_in;
861     input [word_size-1:0] address;
862     input clk, write;
863     reg [word_size-1:0] memory [memory_size-1:0];
864
865     assign data_out = memory[address];
866
867     always @ (posedge clk)
868     if (write) memory[address] = data_in;
869 endmodule
870
871 module Memory_Unit (data_out, data_in, address, clk, write);
872     parameter word_size = 8;
873     parameter memory_size = 256;
874
875     input [word_size-1:0] data_in;
876     input [word_size-1:0] address;
877     input clk, write;
878     reg [word_size-1:0] memory [memory_size-1:0];
879
880     assign data_out = memory[address];
881
882     always @ (posedge clk)
883     if (write) memory[address] = data_in;
884 endmodule
885
886 module Memory_Unit (data_out, data_in, address, clk, write);
887     parameter word_size = 8;
888     parameter memory_size = 256;
889
890     input [word_size-1:0] data_in;
891     input [word_size-1:0] address;
892     input clk, write;
893     reg [word_size-1:0] memory [memory_size-1:0];
894
895     assign data_out = memory[address];
896
897     always @ (posedge clk)
898     if (write) memory[address] = data_in;
899 endmodule
900
901 module Memory_Unit (data_out, data_in, address, clk, write);
902     parameter word_size = 8;
903     parameter memory_size = 256;
904
905     input [word_size-1:0] data_in;
906     input [word_size-1:0] address;
907     input clk, write;
908     reg [word_size-1:0] memory [memory_size-1:0];
909
910     assign data_out = memory[address];
911
912     always @ (posedge clk)
913     if (write) memory[address] = data_in;
914 endmodule
915
916 module Memory_Unit (data_out, data_in, address, clk, write);
917     parameter word_size = 8;
918     parameter memory
```