Appendix A
Verilog Simulation Code

The Verilog modules for AMPIRE are listed in alphabetical order, except for *parameter* and *ampire.v*, which appear first.

Module	Description
parameter	Parameter declarations.
ampire.v	Top-level wiring and test setup.
alu.v	Arithmetic logic unit.
arbk.v	Arbiter for K_bus.
arbr.v	Arbiter for R_bus.
bigc.v	Big C-element for rollback synchronization.
ckf.v	Checker for REGDWB outputs to A_bus and B_bus.
cki.v	Checker for instructions executed by the IIU.
ckm.v	Checker for data written to MEMDWB.
ckr.v	Checker for data written to REGDWB.
dmem.v	Data memory.
dq.v	Data queue for memory-to-register transfers.
gates.v	Gate-level components.
iiu.v	Instruction issuing unit.
imem.v	Instruction memory.
iq.v	Instruction queue.
log.v	Instruction log.
memdwb.v	Delayed write buffer for data memory.
pc.v	Program counter.
regdwb.v	Delayed write buffer for register file.
regfile.v	Register file.
restable.v	Register reservation table.

Table A.1: Verilog Modules

parameter

```
  1:    parameter ADDR_WIDTH    = 8;    // width of address bus (bytes)
  2:    parameter ADDR_IGNORE   = 2;    // number of address bits to ignore
  3:    parameter ADDR_INC      = 4;    // amount of address increment (2^IGNORE)
  4:    parameter SEQ_WIDTH     = 4;    // bits for sequence number
  5:    parameter FUNC_WIDTH    = 5;    // ALU function code width
  6:    parameter CHECKERS      = 4;    // number of checkers
  7:    parameter CHKID_WIDTH   = 2;    // number of bits for checker ID
  8:    parameter DWBS          = 2;    // number of DWBs (for validation)
  9:
 10:    parameter OP_WIDTH      = 6;    // bits for op code
 11:    parameter REG_WIDTH     = 5;    // bits for register number
 12:    parameter EXTRA_WIDTH   = 11;   // "extra" field, "func" in R-type
 13:
 14:    parameter REG_SIZE      = 32;   // number of registers (2^REG_WIDTH)
 15:    parameter IMEM_SIZE     = 64;   // number of instruction memory WORDS
 16:    parameter DMEM_SIZE     = 64;   // number of data memory WORDS
 17:
 18:    parameter IMM_WIDTH = REG_WIDTH + EXTRA_WIDTH;          // immediate field
 19:    parameter OFFSET_WIDTH = 2 * REG_WIDTH + IMM_WIDTH;     // offset field
 20:    parameter DATA_WIDTH = OP_WIDTH + OFFSET_WIDTH;         // word size
 21:
 22:    parameter RESET_TIME    = 1;    // 1 for behavioral, 30 for gate-level
 23:    `define BEHAV_IQ                // comment out for gate-level
 24:    `define BEHAV_DQ                // *** CHANGE RESET_TIME ***
 25:    `define BEHAV_ARBR
 26:    `define BEHAV_BIGC
 27:
 28:    // `define WAVE_IQ              // define to add signal waves
 29:    // `define WAVE_REGCMPL
 30:    // `define WAVE_DQ
 31:    // `define WAVE_SEQCMP
 32:    // `define WAVE_ARBR
 33:
 34:    parameter DLY_PC_INC    = 1;    // delay settings
 35:    parameter DLY_IMEM_RD   = 1;
 36:    parameter DLY_IIU_PCINC = 1;
 37:    parameter DLY_IIU_DECOD = 1;
 38:    parameter DLY_IIU_ADD   = 1;
 39:    parameter DLY_RT_RES    = 1;
 40:    parameter DLY_RT_CLR    = 1;
 41:    parameter DLY_RF_RD     = 1;
 42:    parameter DLY_RF_WR     = 1;
 43:    parameter DLY_ALU_ADD   = 1;
 44:    parameter DLY_ALU_SUB   = 1;
 45:    parameter DLY_ALU_AND   = 1;
 46:    parameter DLY_ALU_OR    = 1;
 47:    parameter DLY_ALU_XOR   = 1;
 48:    parameter DLY_ALU_PASS  = 1;
 49:    parameter DLY_ALU_SHIFT = 1;
 50:    parameter DLY_ALU_COMP  = 1;
 51:    parameter DLY_DMEM_RD   = 1;
 52:    parameter DLY_DMEM_WR   = 1;
 53:    parameter DLY_SEQ_COMP  = 1;    // sequence comparison delay for all modules
 54:
 55:    parameter DLY_CKI_CHK   = 1;
 56:    parameter DLY_CKF_CHK   = 1;
 57:    parameter DLY_CKM_CHK   = 1;
 58:    parameter DLY_CKR_CHK   = 1;
 59:
 60:    parameter ID_CKI        = 0;    // checker ID numbers
 61:    parameter ID_CKF        = 1;
 62:    parameter ID_CKM        = 2;
 63:    parameter ID_CKR        = 3;
 64:
 65:    parameter XXX           = 33'bx;    // to overcome default size of 32 bits
 66:    parameter ZZZ           = 33'bz;
 67:
 68:    parameter OP_SPECIAL    = 0;    // reg-reg special ops
 69:    parameter OP_J          = 2;
 70:    parameter OP_JAL        = 3;
 71:    parameter OP_BEQZ       = 4;
```

parameter

```
 72:    parameter OP_BNEZ      = 5;
 73:    parameter OP_ADDUI     = 9;
 74:    parameter OP_SUBUI     = 11;
 75:    parameter OP_ANDI      = 12;
 76:    parameter OP_ORI       = 13;
 77:    parameter OP_XORI      = 14;
 78:    parameter OP_LHI       = 15;   // shift by IMM_WIDTH bits
 79:    parameter OP_TRAP      = 17;   // used as simulator special service
 80:    parameter OP_JR        = 18;
 81:    parameter OP_JALR      = 19;
 82:    parameter OP_SLLI      = 20;
 83:    parameter OP_SRLI      = 22;
 84:    parameter OP_SRAI      = 23;
 85:    parameter OP_SEQI      = 24;
 86:    parameter OP_SNEI      = 25;
 87:    parameter OP_SLTI      = 26;
 88:    parameter OP_SGTI      = 27;
 89:    parameter OP_SLEI      = 28;
 90:    parameter OP_SGEI      = 29;
 91:    parameter OP_LW        = 35;
 92:    parameter OP_SW        = 43;
 93:    parameter OP_ADDUIF    = 49;   // add immediate with fault
 94:    parameter OP_JRF       = 50;   // jump register with fault
 95:    parameter OP_SWF       = 51;   // store word with fault
 96:
 97:    parameter SOP_NOP      = 0;    // reg-reg special opcodes
 98:    parameter SOP_SLL      = 4;
 99:    parameter SOP_SRL      = 6;
100:    parameter SOP_SRA      = 7;
101:    parameter SOP_ADDUF    = 25;   // add with fault
102:    parameter SOP_ADDU     = 33;
103:    parameter SOP_SUBU     = 35;
104:    parameter SOP_AND      = 36;
105:    parameter SOP_OR       = 37;
106:    parameter SOP_XOR      = 38;
107:    parameter SOP_SEQ      = 40;
108:    parameter SOP_SNE      = 41;
109:    parameter SOP_SLT      = 42;
110:    parameter SOP_SGT      = 43;
111:    parameter SOP_SLE      = 44;
112:    parameter SOP_SGE      = 45;
113:
114:    parameter FUNC_ADDU    = 0;    // ALU function codes
115:    parameter FUNC_SUBU    = 1;
116:    parameter FUNC_AND     = 2;
117:    parameter FUNC_OR      = 3;
118:    parameter FUNC_XOR     = 4;
119:    parameter FUNC_LHI     = 5;
120:    parameter FUNC_SLL     = 6;
121:    parameter FUNC_SRL     = 7;
122:    parameter FUNC_SRA     = 8;
123:    parameter FUNC_SLT     = 9;
124:    parameter FUNC_SGT     = 10;
125:    parameter FUNC_SLE     = 11;
126:    parameter FUNC_SGE     = 12;
127:    parameter FUNC_SEQ     = 13;
128:    parameter FUNC_SNE     = 14;
129:    parameter FUNC_PASS    = 15;
130:    parameter FUNC_ADDUF   = 16;   // add with fault
131:
132:    parameter TRAP_BLANK   = 90;   // must be >= REG_SIZE
133:    parameter TRAP_STOP    = 100;
```

ampire.v

```
  1:    // AMPIRE Top-Level Wiring and Test Setup
  2:
  3:    module ampire;
  4:
  5:    `include "parameter"
  6:
  7:    reg     reset;
  8:
  9:    // ======================================================================
 10:    // bus connections
 11:
 12:    trireg [DATA_WIDTH:0]    i_bus,
 13:                             a_bus,
 14:                             b_bus,
 15:                             d_bus;
 16:
 17:    trireg [SEQ_WIDTH-1:0]   r_bus_seq;
 18:    trireg [REG_WIDTH-1:0]   r_bus_dest;
 19:    trireg [DATA_WIDTH:0]    r_bus_data;
 20:    trireg                   r_bus_req;
 21:
 22:    trireg [SEQ_WIDTH-1:0]   k_bus_seq;
 23:    trireg [CHKID_WIDTH-1:0] k_bus_chkid;
 24:    trireg                   k_bus_error,
 25:                             k_bus_req;
 26:
 27:    trireg                   tri_dq_req;
 28:
 29:    pulldown                (r_bus_req);    // do not float handshaking signals
 30:    pulldown                (k_bus_req);
 31:    pulldown                (tri_dq_req);
 32:
 33:    // ======================================================================
 34:    // component outputs
 35:
 36:    wire [ADDR_WIDTH:0]
 37:            pc_out;
 38:    wire    pc_out_retry,
 39:            pc_load_ack,
 40:            pc_out_req;
 41:
 42:    wire [DATA_WIDTH:0]
 43:            imem_data;
 44:    wire    imem_in_ack,
 45:            imem_out_req,
 46:            imem_cancel_ack;
 47:
 48:    wire    iq_in_ack,
 49:            iq_out_req,
 50:            iq_cancel_ack;
 51:
 52:    wire [CHECKERS-1:0]
 53:            iiu_chkbits;
 54:    wire [SEQ_WIDTH-1:0]
 55:            iiu_seq_num;
 56:    wire [REG_WIDTH-1:0]
 57:            iiu_reg_rd,
 58:            iiu_reg_rs,
 59:            iiu_reg_rt;
 60:    wire [FUNC_WIDTH-1:0]
 61:            iiu_alu_func;
 62:    wire    iiu_inst_en,
 63:            iiu_sim_f,
 64:            iiu_mem_rw,
 65:            iiu_retry,
 66:            stop;
 67:    wire    iiu_inst_ack,
 68:            iiu_log_req,
 69:            iiu_cki_req,
 70:            iiu_newpc_req,
 71:            iiu_res_req,
```

```
 72:            iiu_rfile_req,
 73:            iiu_ckf_req,
 74:            iiu_alu_req,
 75:            iiu_mem_req;
 76:
 77:    wire    restable_res_ack,
 78:            restable_clr_ack;
 79:
 80:    wire    alu_comp_ack,
 81:            alu_arb_req;
 82:
 83:    wire [ADDR_WIDTH:0]
 84:            memdwb_out_addr;
 85:    wire [SEQ_WIDTH-1:0]
 86:            memdwb_out_seq;
 87:    wire [REG_WIDTH-1:0]
 88:            memdwb_out_reg;
 89:    wire    memdwb_rw_mode,
 90:            memdwb_out_retry,
 91:            memdwb_in_ack,
 92:            memdwb_chk_req,
 93:            memdwb_val_ack,
 94:            memdwb_out_req;
 95:
 96:    wire    dmem_in_ack;
 97:
 98:    wire    dq_in_ack,
 99:            dq_arb_req;
100:
101:    wire    arbr_ack0,
102:            arbr_ack1;
103:
104:    wire [SEQ_WIDTH-1:0]
105:            regdwb_out_seq;
106:    wire [REG_WIDTH-1:0]
107:            regdwb_out_reg,
108:            regdwb_wrf_reg;
109:    wire [DATA_WIDTH:0]
110:            regdwb_out_data,
111:            regdwb_wrf_data;
112:    wire    regdwb_in_ack,
113:            regdwb_chk_req,
114:            regdwb_clr_req,
115:            regdwb_val_ack,
116:            regdwb_rd_ack,
117:            regdwb_rdf_req,
118:            regdwb_wrf_req;
119:
120:    wire [DATA_WIDTH:0]
121:            rfile_rd_data1,
122:            rfile_rd_data2;
123:    wire    rfile_wr_ack,
124:            rfile_rd_ack;
125:
126:    wire    cki_in_ack,
127:            cki_arb_req;
128:
129:    wire    ckf_in_ack,
130:            ckf_arb_req;
131:
132:    wire    ckm_in_ack,
133:            ckm_arb_req;
134:
135:    wire    ckr_in_ack,
136:            ckr_arb_req;
137:
138:    wire    arbk_ack0,
139:            arbk_ack1,
140:            arbk_ack2,
141:            arbk_ack3;
142:
```

```
143:    wire [SEQ_WIDTH-1:0]
144:            val_seq;
145:    wire    log_ack,
146:            log_chk_ack,
147:            log_valmem_req,
148:            log_valreg_req,
149:            halt_req,
150:            roll_req;
151:
152:    wire    ch_out;
153:
154:    wire    cr_out;
155:
156:    // ===================================================================
157:    // component inputs
158:
159:    wire [ADDR_WIDTH:0]
160:            pc_load = i_bus;
161:    wire    pc_in_retry = iiu_retry,
162:            pc_load_req = iiu_newpc_req,
163:            pc_out_ack = imem_in_ack;
164:
165:    wire [ADDR_WIDTH:0]
166:            imem_addr = pc_out;
167:    wire    imem_retry = pc_out_retry,
168:            imem_in_req = pc_out_req,
169:            imem_out_ack = iq_in_ack,
170:            imem_cancel_req = iiu_newpc_req;
171:
172:    wire [DATA_WIDTH:0]
173:            iq_in_data = imem_data;
174:    wire    iq_in_req = imem_out_req,
175:            iq_out_ack = iiu_inst_ack,
176:            iq_en_out = iiu_inst_en,
177:            iq_cancel_req = iiu_newpc_req;
178:
179:    wire    iiu_inst_req = iq_out_req,
180:            iiu_log_ack = log_ack,
181:            iiu_cki_ack = cki_in_ack,
182:            iiu_pc_ack = pc_load_ack,
183:            iiu_imem_ack = imem_cancel_ack,
184:            iiu_iq_ack = iq_cancel_ack,
185:            iiu_res_ack = restable_res_ack,
186:            iiu_rfile_ack = regdwb_rd_ack,
187:            iiu_ckf_ack = ckf_in_ack,
188:            iiu_alu_ack = alu_comp_ack,
189:            iiu_mem_ack = memdwb_in_ack;
190:
191:    wire [SEQ_WIDTH-1:0]
192:            restable_seq_num = iiu_seq_num;
193:    wire [REG_WIDTH-1:0]
194:            restable_reg_w = iiu_reg_rd,
195:            restable_reg_r1 = iiu_reg_rs,
196:            restable_reg_r2 = iiu_reg_rt,
197:            restable_reg_clr = regdwb_out_reg;
198:    wire    restable_res_req = iiu_res_req,
199:            restable_clr_req = regdwb_clr_req;
200:
201:    wire [SEQ_WIDTH-1:0]
202:            alu_in_seq = iiu_seq_num;
203:    wire [REG_WIDTH-1:0]
204:            alu_in_dest = iiu_reg_rd;
205:    wire [DATA_WIDTH:0]
206:            alu_in_d1 = a_bus,
207:            alu_in_d2 = b_bus;
208:    wire [FUNC_WIDTH-1:0]
209:            alu_in_func = iiu_alu_func;
210:    wire    alu_comp_req = iiu_alu_req,
211:            alu_arb_ack = arbr_ack0,
212:            alu_out_ack = regdwb_in_ack;
213:
```

```
214:    wire [SEQ_WIDTH-1:0]
215:            memdwb_in_seq = iiu_seq_num;
216:    wire [REG_WIDTH-1:0]
217:            memdwb_in_reg = iiu_reg_rd;
218:    wire [DATA_WIDTH:0]
219:            memdwb_in_addr = a_bus,
220:            memdwb_in_data = b_bus;
221:    wire    memdwb_in_rw_mode = iiu_mem_rw,
222:            memdwb_in_retry = iiu_retry,
223:            memdwb_in_req = iiu_mem_req,
224:            memdwb_chk_ack = ckm_in_ack,
225:            memdwb_val_req = log_valmem_req,
226:            memdwb_out_ack = dmem_in_ack,
227:            memdwb_dq_ack = dq_in_ack;
228:
229:    wire [ADDR_WIDTH:0]
230:            dmem_addr = memdwb_out_addr;
231:    wire    dmem_rw_mode = memdwb_rw_mode,
232:            dmem_retry = memdwb_out_retry,
233:            dmem_in_req = memdwb_out_req,
234:            dmem_out_ack = dq_in_ack;
235:
236:    wire [SEQ_WIDTH-1:0]
237:            dq_in_seq = memdwb_out_seq;
238:    wire [REG_WIDTH-1:0]
239:            dq_in_reg = memdwb_out_reg;
240:    wire [DATA_WIDTH:0]
241:            dq_in_data = d_bus;
242:    wire    dq_in_req = tri_dq_req,
243:            dq_arb_ack = arbr_ack1,
244:            dq_out_ack = regdwb_in_ack;
245:
246:    wire    arbr_req0 = alu_arb_req,
247:            arbr_req1 = dq_arb_req;
248:
249:    wire [SEQ_WIDTH-1:0]
250:            regdwb_in_seq = r_bus_seq;
251:    wire [REG_WIDTH-1:0]
252:            regdwb_in_reg = r_bus_dest,
253:            regdwb_rd_reg1 = iiu_reg_rs,
254:            regdwb_rd_reg2 = iiu_reg_rt;
255:    wire [DATA_WIDTH:0]
256:            regdwb_in_data = r_bus_data,
257:            regdwb_rf_data1 = rfile_rd_data1,
258:            regdwb_rf_data2 = rfile_rd_data2;
259:    wire    regdwb_sim_f = iiu_sim_f;
260:    wire    regdwb_in_req = r_bus_req,
261:            regdwb_chk_ack = ckr_in_ack,
262:            regdwb_clr_ack = restable_clr_ack,
263:            regdwb_val_req = log_valreg_req,
264:            regdwb_rd_req = iiu_rfile_req,
265:            regdwb_rdf_ack = rfile_rd_ack,
266:            regdwb_wrf_ack = rfile_wr_ack;
267:
268:    wire [REG_WIDTH-1:0]
269:            rfile_wr_reg = regdwb_wrf_reg,
270:            rfile_rd_reg1 = iiu_reg_rs,
271:            rfile_rd_reg2 = iiu_reg_rt;
272:    wire [DATA_WIDTH:0]
273:            rfile_wr_data = regdwb_wrf_data;
274:    wire    rfile_wr_req = regdwb_wrf_req,
275:            rfile_rd_req = regdwb_rdf_req;
276:
277:    wire [SEQ_WIDTH-1:0]
278:            log_seq = iiu_seq_num,
279:            log_chk_seq = k_bus_seq;
280:    wire [CHECKERS-1:0]
281:            log_chkbits = iiu_chkbits;
282:    wire [CHKID_WIDTH-1:0]
283:            log_chk_chkid = k_bus_chkid;
284:    wire    log_chk_error = k_bus_error;
```

```
285:    wire    log_req = iiu_log_req,
286:            log_chk_req = k_bus_req,
287:            log_valmem_ack = memdwb_val_ack,
288:            log_valreg_ack = regdwb_val_ack,
289:            log_halt_ack = ch_out,
290:            log_roll_ack = cr_out;
291:
292:    wire [SEQ_WIDTH-1:0]
293:            cki_in_seq = iiu_seq_num;
294:    wire [DATA_WIDTH:0]
295:            cki_in_data = b_bus;
296:    wire    cki_in_req = iiu_cki_req,
297:            cki_arb_ack = arbk_ack0,
298:            cki_out_ack = log_chk_ack;
299:
300:    wire [SEQ_WIDTH-1:0]
301:            ckf_in_seq = iiu_seq_num;
302:    wire [DATA_WIDTH:0]
303:            ckf_in_data1 = a_bus,
304:            ckf_in_data2 = b_bus;
305:    wire    ckf_in_req = iiu_ckf_req,
306:            ckf_arb_ack = arbk_ack1,
307:            ckf_out_ack = log_chk_ack;
308:
309:    wire [SEQ_WIDTH-1:0]
310:            ckm_in_seq = iiu_seq_num;
311:    wire [DATA_WIDTH:0]
312:            ckm_in_data1 = a_bus,
313:            ckm_in_data2 = b_bus;
314:    wire    ckm_in_req = memdwb_chk_req,
315:            ckm_arb_ack = arbk_ack2,
316:            ckm_out_ack = log_chk_ack;
317:
318:    wire [SEQ_WIDTH-1:0]
319:            ckr_in_seq = regdwb_out_seq;
320:    wire [DATA_WIDTH:0]
321:            ckr_in_data = regdwb_out_data;
322:    wire    ckr_in_req = regdwb_chk_req,
323:            ckr_arb_ack = arbk_ack3,
324:            ckr_out_ack = log_chk_ack;
325:
326:    wire    arbk_req0 = cki_arb_req,
327:            arbk_req1 = ckf_arb_req,
328:            arbk_req2 = ckm_arb_req,
329:            arbk_req3 = ckr_arb_req;
330:
331:    // ====================================================================
332:    // components
333:
334:    pc pc (pc_load, pc_in_retry, pc_load_req, pc_load_ack, pc_out, pc_out_retry,
335:            pc_out_req, pc_out_ack, reset);
336:
337:    imem imem (imem_addr, imem_retry, imem_in_req, imem_in_ack, imem_data,
338:            imem_out_req, imem_out_ack, imem_cancel_req, imem_cancel_ack, reset);
339:
340:    iq iq (iq_in_data, iq_in_req, iq_in_ack, i_bus, iq_out_req,
341:            iq_out_ack, iq_en_out, iq_cancel_req, iq_cancel_ack, reset);
342:
343:    iiu iiu (i_bus, iiu_inst_req, iiu_inst_ack, iiu_inst_en,
344:            iiu_chkbits, iiu_log_req, iiu_log_ack, iiu_cki_req, iiu_cki_ack,
345:            iiu_newpc_req, iiu_pc_ack, iiu_imem_ack, iiu_iq_ack, iiu_seq_num,
346:            iiu_reg_rd, iiu_reg_rs, iiu_reg_rt, iiu_res_req, iiu_res_ack,
347:            iiu_sim_f, a_bus, b_bus, iiu_rfile_req, iiu_rfile_ack,
348:            iiu_ckf_req, iiu_ckf_ack, iiu_alu_func,
349:            iiu_alu_req, iiu_alu_ack, iiu_mem_rw, iiu_mem_req, iiu_mem_ack,
350:            val_seq, halt_req, iiu_halt_ack, roll_req, iiu_roll_ack,
351:            iiu_retry, stop, reset);
352:
353:    restable restable (restable_seq_num, restable_reg_w, restable_reg_r1,
354:            restable_reg_r2, restable_res_req, restable_res_ack,
355:            restable_reg_clr, restable_clr_req, restable_clr_ack,
```

```
356:            val_seq, halt_req, restable_halt_ack, roll_req, restable_roll_ack,
357:            reset);
358:
359:    alu alu (alu_in_seq, alu_in_dest, alu_in_d1, alu_in_d2, alu_in_func,
360:            alu_comp_req, alu_comp_ack, alu_arb_req, alu_arb_ack, r_bus_seq,
361:            r_bus_dest, r_bus_data, r_bus_req, alu_out_ack,
362:            val_seq, halt_req, alu_halt_ack, roll_req, alu_roll_ack, reset);
363:
364:    memdwb memdwb (memdwb_in_seq, memdwb_in_reg, memdwb_in_addr, memdwb_in_data,
365:            memdwb_in_rw_mode, memdwb_in_retry, memdwb_in_req, memdwb_in_ack,
366:            memdwb_chk_req, memdwb_chk_ack, val_seq, memdwb_val_req,
367:            memdwb_val_ack, memdwb_out_addr, d_bus, memdwb_rw_mode,
368:            memdwb_out_retry, memdwb_out_req, memdwb_out_ack, memdwb_out_seq,
369:            memdwb_out_reg, tri_dq_req, memdwb_dq_ack,
370:            halt_req, memdwb_halt_ack, roll_req, memdwb_roll_ack, reset);
371:
372:    dmem dmem (dmem_addr, d_bus, dmem_rw_mode, dmem_retry, dmem_in_req,
373:            dmem_in_ack, tri_dq_req, dmem_out_ack,
374:            halt_req, dmem_halt_ack, roll_req, dmem_roll_ack, reset);
375:
376:    dq dq (dq_in_seq, dq_in_reg, dq_in_data, dq_in_req, dq_in_ack, dq_arb_req,
377:            dq_arb_ack, r_bus_seq, r_bus_dest, r_bus_data, r_bus_req, dq_out_ack,
378:            val_seq, halt_req, dq_halt_ack, roll_req, dq_roll_ack, reset);
379:
380:    arbr arbr (arbr_req0, arbr_ack0, arbr_req1, arbr_ack1,
381:            halt_req, arbr_halt_ack, roll_req, arbr_roll_ack, reset);
382:
383:    regdwb regdwb (regdwb_in_seq, regdwb_in_reg, regdwb_in_data, regdwb_in_req,
384:            regdwb_in_ack, regdwb_out_seq, regdwb_out_reg, regdwb_out_data,
385:            regdwb_chk_req, regdwb_chk_ack, regdwb_clr_req, regdwb_clr_ack,
386:            val_seq, regdwb_val_req, regdwb_val_ack,
387:            regdwb_rd_reg1, regdwb_rd_reg2, regdwb_sim_f, a_bus, b_bus,
388:            regdwb_rd_req, regdwb_rd_ack, regdwb_rf_data1, regdwb_rf_data2,
389:            regdwb_rdf_req, regdwb_rdf_ack, regdwb_wrf_reg, regdwb_wrf_data,
390:            regdwb_wrf_req, regdwb_wrf_ack,
391:            halt_req, regdwb_halt_ack, roll_req, regdwb_roll_ack, reset);
392:
393:    regfile regfile (rfile_wr_reg, rfile_wr_data, rfile_wr_req, rfile_wr_ack,
394:            rfile_rd_reg1, rfile_rd_reg2, rfile_rd_data1, rfile_rd_data2,
395:            rfile_rd_req, rfile_rd_ack,
396:            halt_req, rfile_halt_ack, roll_req, rfile_roll_ack, reset);
397:
398:    log log (log_seq, log_chkbits, a_bus, log_req, log_ack, log_chk_seq,
399:            log_chk_chkid, log_chk_error, log_chk_req, log_chk_ack,
400:            val_seq, log_valmem_req, log_valmem_ack, log_valreg_req,
401:            log_valreg_ack, halt_req, log_halt_ack, roll_req,
402:            log_roll_ack, reset);
403:
404:    cki cki (cki_in_seq, cki_in_data, cki_in_req, cki_in_ack, cki_arb_req,
405:            cki_arb_ack, k_bus_seq, k_bus_chkid, k_bus_error, k_bus_req,
406:            cki_out_ack, val_seq, halt_req, cki_halt_ack, roll_req, cki_roll_ack,
407:            reset);
408:
409:    ckf ckf (ckf_in_seq, ckf_in_data1, ckf_in_data2, ckf_in_req, ckf_in_ack,
410:            ckf_arb_req, ckf_arb_ack, k_bus_seq, k_bus_chkid, k_bus_error,
411:            k_bus_req, ckf_out_ack, val_seq, halt_req, ckf_halt_ack, roll_req,
412:            ckf_roll_ack, reset);
413:
414:    ckm ckm (ckm_in_seq, ckm_in_data1, ckm_in_data2, ckm_in_req, ckm_in_ack,
415:            ckm_arb_req, ckm_arb_ack, k_bus_seq, k_bus_chkid, k_bus_error,
416:            k_bus_req, ckm_out_ack, val_seq, halt_req, ckm_halt_ack, roll_req,
417:            ckm_roll_ack, reset);
418:
419:    ckr ckr (ckr_in_seq, ckr_in_data, ckr_in_req, ckr_in_ack, ckr_arb_req,
420:            ckr_arb_ack, k_bus_seq, k_bus_chkid, k_bus_error, k_bus_req,
421:            ckr_out_ack, val_seq, halt_req, ckr_halt_ack, roll_req, ckr_roll_ack,
422:            reset);
423:
424:    arbk arbk (arbk_req0, arbk_ack0, arbk_req1, arbk_ack1,
425:            arbk_req2, arbk_ack2, arbk_req3, arbk_ack3,
426:            halt_req, arbk_halt_ack, roll_req, arbk_roll_ack, reset);
```

```
427:
428:    bigc c_halt (iiu_halt_ack, restable_halt_ack, alu_halt_ack, memdwb_halt_ack,
429:            dmem_halt_ack, dq_halt_ack, regdwb_halt_ack, rfile_halt_ack,
430:            cki_halt_ack, ckf_halt_ack, ckm_halt_ack, ckr_halt_ack,
431:            arbr_halt_ack, arbk_halt_ack, ch_out, reset);
432:
433:    bigc c_roll (iiu_roll_ack, restable_roll_ack, alu_roll_ack, memdwb_roll_ack,
434:            dmem_roll_ack, dq_roll_ack, regdwb_roll_ack, rfile_roll_ack,
435:            cki_roll_ack, ckf_roll_ack, ckm_roll_ack, ckr_roll_ack,
436:            arbr_roll_ack, arbk_roll_ack, cr_out, reset);
437:
438:    // ================================================================
439:    // test setup
440:
441:    initial
442:    begin
443:        $freeze_waves;                      // saves simulation time
444:        $gr_waves_memsize (1024 * 1024);
445:        $gr_position ("waves", 50, 0, 1050, 850);
446:
447:        $gr_waves (
448:            "pc_load",      pc_load_req,    // instruction fetch
449:            "pc_out",       pc_out,
450:            "im_req",       pc_out_req,
451:            "im_ack",       pc_out_ack,
452:            "im_inst",      imem_data,
453:            "iq_req",       imem_out_req,
454:            "iq_ack",       imem_out_ack,
455:            "I_bus",        i_bus,
456:            "iiu_req",      iiu_inst_req,
457:            "iiu_ack",      iiu_inst_ack,
458:
459:            "seq_num",      iiu_seq_num,    // instruction preparation
460:            "reg_rs",       iiu_reg_rs,
461:            "reg_rt",       iiu_reg_rt,
462:            "reg_rd",       iiu_reg_rd,
463:            "log_req",      iiu_log_req,
464:            "log_ack",      iiu_log_ack,
465:            "cki_req",      iiu_cki_req,
466:            "res_req",      iiu_res_req,
467:            "rf_req",       iiu_rfile_req,
468:
469:            "A_bus",        a_bus,          // instruction issue & exec
470:            "A_bus_L",      a_bus [ADDR_WIDTH:0],
471:            "B_bus",        b_bus,
472:            "func",         iiu_alu_func,
473:            "alu_req",      iiu_alu_req,
474:            "mem_req",      iiu_mem_req,
475:
476:            "dmem_rq",      memdwb_out_req,
477:            "dq_req",       dq_in_req,
478:
479:            "R_seq",        r_bus_seq,      // register file access
480:            "R_data",       r_bus_data,
481:            "R_dest",       r_bus_dest,
482:            "alu_arb",      alu_arb_req,
483:            "dq_arb",       dq_arb_req,
484:
485:            "clr_req",      regdwb_clr_req,
486:            "wrf_reg",      rfile_wr_reg,
487:            "wrf_req",      rfile_wr_req,
488:
489:            "cki_arb",      cki_arb_req,    // error detection
490:            "ckf_arb",      ckf_arb_req,
491:            "ckm_arb",      ckm_arb_req,
492:            "ckr_arb",      ckr_arb_req,
493:            "chk_ack",      log_chk_ack,
494:            "K_seq",        k_bus_seq,
495:            "K_chkid",      k_bus_chkid,
496:            "K_err",        k_bus_error,
497:
```

```
498:             "val_seq",      val_seq,        // validation & rollback
499:             "val_mem",      log_valmem_req,
500:             "val_reg",      log_valreg_req,
501:             "hlt_req",      halt_req,
502:             "hlt_ack",      log_halt_ack,
503:             "rol_req",      roll_req,
504:             "rol_ack",      log_roll_ack);
505:
506:         $define_group_waves (1, "tst1a",
507:             "pc_load", "pc_out", "im_req", "im_ack", "im_inst",
508:             "iq_req", "iq_ack", "I_bus", "iiu_req", "iiu_ack", "seq_num");
509:
510:         $define_group_waves (2, "tst1b",
511:             "pc_load", "pc_out", "im_req", "im_inst", "iq_req", "I_bus",
512:             "iiu_req", "iiu_ack", "seq_num", "reg_rs", "reg_rt", "reg_rd",
513:             "log_req", "cki_req", "res_req", "rf_req", "func", "alu_req",
514:             "R_seq", "R_data", "R_dest", "alu_arb", "clr_req", "wrf_reg",
515:             "wrf_req", "K_seq", "K_chkid", "val_seq", "val_reg");
516:
517:         $define_group_waves (3, "tst2a",
518:             "I_bus", "iiu_ack", "seq_num", "reg_rs", "reg_rt", "reg_rd",
519:             "log_req", "res_req", "rf_req", "alu_req", "mem_req", "dmem_rq",
520:             "dq_req", "R_seq", "R_data", "R_dest", "alu_arb", "dq_arb",
521:             "clr_req", "wrf_reg", "wrf_req", "val_seq", "val_reg", "val_mem");
522:
523:         $define_group_waves (4, "tst2b",
524:             "seq_num", "log_req", "log_ack", "K_seq", "cki_arb", "ckf_arb",
525:             "ckm_arb", "ckr_arb", "chk_ack");
526:
527:         $define_group_waves (5, "tst3a",
528:             "I_bus", "iiu_ack", "seq_num", "reg_rd", "log_req", "res_req",
529:             "rf_req", "alu_req", "mem_req", "dmem_rq", "dq_req", "R_seq",
530:             "R_data", "R_dest", "alu_arb", "dq_arb", "clr_req", "wrf_reg",
531:             "wrf_req", "val_seq", "val_reg");
532:
533:         $define_group_waves (6, "tst4a",
534:             "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
535:             "reg_rd", "log_req", "res_req", "rf_req", "func", "alu_req",
536:             "mem_req", "dmem_rq", "dq_req", "R_seq", "R_data", "R_dest",
537:             "alu_arb", "dq_arb", "clr_req", "wrf_reg", "wrf_req", "K_seq",
538:             "K_chkid", "K_err", "val_seq", "val_reg", "val_mem", "hlt_req");
539:
540:         $define_group_waves (7, "tst4b",
541:             "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
542:             "reg_rd", "log_req", "res_req", "rf_req", "func", "alu_req",
543:             "R_seq", "R_data", "R_dest", "alu_arb", "clr_req", "wrf_reg",
544:             "wrf_req", "K_seq", "K_chkid", "K_err", "val_seq", "val_reg",
545:             "hlt_req");
546:
547:         $define_group_waves (8, "tst4c",
548:             "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
549:             "reg_rd", "log_req", "res_req", "rf_req", "K_seq", "K_chkid",
550:             "K_err", "val_seq", "hlt_req");
551:
552:         $define_group_waves (9, "tst4d",
553:             "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
554:             "reg_rd", "log_req", "res_req", "rf_req", "func", "alu_req",
555:             "R_seq", "R_data", "R_dest", "alu_arb", "clr_req", "wrf_reg",
556:             "wrf_req", "K_seq", "K_chkid", "K_err", "val_seq", "val_reg",
557:             "hlt_req");
558:
559:         $define_group_waves (10, "tst4e",
560:             "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
561:             "reg_rd", "log_req", "res_req", "rf_req", "mem_req", "dmem_rq",
562:             "K_seq", "K_chkid", "K_err", "val_seq", "val_mem", "hlt_req");
563:
564:         $define_group_waves (11, "tst4f",
565:             "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
566:             "reg_rd", "log_req", "res_req", "rf_req", "mem_req", "dmem_rq",
567:             "dq_req", "R_seq", "R_data", "R_dest", "dq_arb", "clr_req",
568:             "wrf_req", "K_seq", "K_chkid", "K_err", "val_seq", "val_reg",
```

```
569:            "hlt_req");
570:
571:        $define_group_waves (12, "tst4g",
572:            "val_seq", "hlt_req", "hlt_ack", "rol_req", "rol_ack", "seq_num",
573:            "A_bus_L", "B_bus", "log_req", "cki_req", "pc_load", "pc_out");
574:
575:        $define_group_waves (13, "tst5a",
576:            "pc_load", "pc_out", "I_bus", "iiu_req", "iiu_ack", "seq_num",
577:            "log_req", "res_req", "rf_req", "func", "alu_req", "mem_req",
578:            "dmem_rq", "dq_req", "R_seq", "R_dest", "alu_arb", "dq_arb",
579:            "clr_req", "wrf_reg", "wrf_req", "cki_arb", "ckf_arb", "ckm_arb",
580:            "ckr_arb", "K_seq", "K_chkid", "K_err", "val_seq", "val_reg",
581:            "val_mem", "hlt_req");
582:
583:        $define_group_waves (20, "all",
584:            "pc_load", "pc_out", "im_req", "im_ack", "im_inst", "iq_req",
585:            "iq_ack", "I_bus", "iiu_req", "iiu_ack", "seq_num", "reg_rs",
586:            "reg_rt", "reg_rd", "log_req", "log_ack", "cki_req", "res_req",
587:            "rf_req", "A_bus", "A_bus_L", "B_bus", "func", "alu_req", "mem_req",
588:            "dmem_rq", "dq_req", "R_seq", "R_data", "R_dest", "alu_arb",
589:            "dq_arb", "clr_req", "wrf_reg", "wrf_req", "cki_arb", "ckf_arb",
590:            "ckm_arb", "ckr_arb", "chk_ack", "K_seq", "K_chkid", "K_err",
591:            "val_seq", "val_mem", "val_reg", "hlt_req", "hlt_ack", "rol_req",
592:            "rol_ack");
593:
594:        #1;                                     // for #0 $gr_addwaves
595:        $define_group_waves (15, "arbr",
596:            "R_seq", "alu_arb", "dq_arb", "none",
597:            "req0", "v0", "ack0", "req1", "v1", "ack1");
598:
599:        $define_group_waves (16, "iq",
600:            "pc_load", "pc_out", "none", "in_d", "out_d", "in_r", "in_a",
601:            "out_r", "out_a", "can_r", "can_a", "none", "latch", "osc", "q1",
602:            "q2", "same1", "same2", "done", "regclk", "complet");
603:
604:        $define_group_waves (17, "dq",
605:            "in_r", "cout1", "latch", "in_a", "a", "cout2", "out_r", "out_a",
606:            "h_req", "h_ack", "r_req", "clear1", "clear2", "r_ack", "none",
607:            "seq", "error", "req", "_a", "na", "b", "nb", "c", "nc", "d",
608:            "inv", "ack");
609:    end
610:
611:    initial
612:    begin
613:            reset = 1;
614:            #RESET_TIME;
615:            reset = 0;
616:
617:            fork
618:                begin               // normal termination
619:                    wait (stop);
620:                    #400 $stop;
621:                end
622:                begin               // abnormal termination
623:                    #5000;
624:                    $display ("AMPIRE: exceeded maximum simulation time.");
625:                    $stop;
626:                end
627:            join
628:    end
629:
630:    endmodule       // ampire
631:
632:    `include "pc.v"
633:    `include "imem.v"
634:    `include "iq.v"
635:    `include "iiu.v"
636:    `include "restable.v"
637:    `include "alu.v"
638:    `include "memdwb.v"
639:    `include "dmem.v"
```

ampire.v

```
640:    `include "dq.v"
641:    `include "arbr.v"
642:    `include "regdwb.v"
643:    `include "regfile.v"
644:    `include "cki.v"
645:    `include "ckf.v"
646:    `include "ckm.v"
647:    `include "ckr.v"
648:    `include "arbk.v"
649:    `include "log.v"
650:    `include "bigc.v"
651:    `include "gates.v"
```

alu.v

```
 1:    // ALU (combining ALUQ and ALUF)
 2:
 3:    module alu (in_seq, in_dest, in_d1, in_d2, in_func, comp_req, comp_ack,
 4:            arb_req, arb_ack, tri_seq, tri_dest, tri_data, tri_req, out_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
10:    input [REG_WIDTH-1:0] in_dest;
11:    input [DATA_WIDTH:0] in_d1, in_d2;
12:    input [FUNC_WIDTH-1:0] in_func;
13:    output [SEQ_WIDTH-1:0] tri_seq;
14:    output [REG_WIDTH-1:0] tri_dest;
15:    output [DATA_WIDTH:0] tri_data;
16:    input comp_req, arb_ack, out_ack, halt_req, roll_req, reset;
17:    output comp_ack, arb_req, tri_req, halt_ack, roll_ack;
18:
19:    reg halt_ack, roll_ack;
20:
21:    wire [SEQ_WIDTH-1:0] mid_seq;
22:    wire [REG_WIDTH-1:0] mid_dest;
23:    wire [DATA_WIDTH:0] mid_d1, mid_d2;
24:    wire [FUNC_WIDTH-1:0] mid_func;
25:
26:    aluq aluq (in_seq, in_dest, in_d1, in_d2, in_func, comp_req, comp_ack,
27:            mid_seq, mid_dest, mid_d1, mid_d2, mid_func, mid_req, mid_ack,
28:            val_seq, halt_req, haltq_ack, roll_req, rollq_ack, reset);
29:    aluf aluf (mid_seq, mid_dest, mid_d1, mid_d2, mid_func, mid_req, mid_ack,
30:            arb_req, arb_ack, tri_seq, tri_dest, tri_data, tri_req, out_ack,
31:            val_seq, halt_req, haltf_ack, roll_req, rollf_ack, reset);
32:
33:    always wait (reset)
34:    begin
35:            halt_ack = 0;
36:            roll_ack = 0;
37:            wait (~reset);
38:    end
39:
40:    always @(haltq_ack or haltf_ack)
41:            if (haltq_ack & haltf_ack)
42:                    halt_ack = 1;
43:            else if (~haltq_ack & ~haltf_ack)
44:                    halt_ack = 0;
45:
46:    always @(rollq_ack or rollf_ack)
47:            if (rollq_ack & rollf_ack)
48:                    roll_ack = 1;
49:            else if (~rollq_ack & ~rollf_ack)
50:                    roll_ack = 0;
51:
52:    endmodule       // alu
53:
54:    // ====================================================================
55:    // ALUF (functional part)
56:
57:    module aluf (in_seq, in_dest, in_d1, in_d2, in_func, comp_req, comp_ack,
58:            arb_req, arb_ack, tri_seq, tri_dest, tri_data, tri_req, out_ack,
59:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
60:
61:    `include "parameter"
62:
63:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
64:    input [REG_WIDTH-1:0] in_dest;
65:    input [DATA_WIDTH:0] in_d1, in_d2;
66:    input [FUNC_WIDTH-1:0] in_func;
67:    output [SEQ_WIDTH-1:0] tri_seq;
68:    output [REG_WIDTH-1:0] tri_dest;
69:    output [DATA_WIDTH:0] tri_data;
70:    input comp_req, arb_ack, out_ack, halt_req, roll_req, reset;
71:    output comp_ack, arb_req, tri_req, halt_ack, roll_ack;
```

```
 72:
 73:     reg [SEQ_WIDTH-1:0] out_seq;
 74:     reg [REG_WIDTH-1:0] out_dest;
 75:     reg [DATA_WIDTH:0] out_data;
 76:     reg comp_ack, arb_req, out_req, halt_ack, roll_ack;
 77:
 78:     reg [DATA_WIDTH-1:0] data1, data2;
 79:     reg [FUNC_WIDTH-1:0] func;
 80:     reg [SEQ_WIDTH-1:0] diff;
 81:     reg [5:0] loop;                         // DATA_WIDTH = 32 bits max
 82:     reg valid, parity;
 83:
 84:     assign tri_seq = arb_ack ? out_seq : ZZZ;
 85:     assign tri_dest = arb_ack ? out_dest : ZZZ;
 86:     assign tri_data = arb_ack ? out_data : ZZZ;
 87:     assign tri_req = arb_ack ? out_req : ZZZ;
 88:
 89:     always wait (reset)
 90:     begin
 91:             disable rollback_cycle;
 92:             disable latch_cycle;
 93:             disable output_cycle;
 94:             out_seq = XXX;
 95:             out_dest = XXX;
 96:             out_data = XXX;
 97:             comp_ack = 0;
 98:             arb_req = 0;
 99:             out_req = 0;
100:             halt_ack = 0;
101:             roll_ack = 0;
102:             valid = 0;
103:             wait (~reset);
104:     end
105:
106:     always wait (halt_req & ~reset)
107:     begin :rollback_cycle
108:             #1;
109:             halt_ack = 1;
110:             wait (roll_req);
111:             #1;
112:             disable latch_cycle;
113:             disable output_cycle;
114:             comp_ack = 0;
115:             arb_req = 0;
116:             out_req = 0;
117:
118:             #DLY_SEQ_COMP;
119:             diff = out_seq - val_seq;       // compare sequence numbers
120:             if (~diff [SEQ_WIDTH-1])
121:                 begin
122:                     valid = 0;
123:                     out_seq = XXX;
124:                     out_dest = XXX;
125:                     out_data = XXX;
126:                 end
127:
128:             roll_ack = 1;
129:             fork
130:                 begin
131:                     wait (~roll_req);
132:                     #1;
133:                     roll_ack = 0;
134:                 end
135:                 begin
136:                     wait (~halt_req);
137:                     #1;
138:                     halt_ack = 0;
139:                 end
140:             join
141:     end
142:
```

```
143:    always wait (comp_req & ~valid & ~halt_req & ~reset)
144:    begin :latch_cycle
145:            #1;
146:            wait (~halt_req);
147:            data1 = in_d1 [DATA_WIDTH-1:0];
148:            data2 = in_d2 [DATA_WIDTH-1:0];
149:            func = in_func;
150:            out_seq = in_seq;
151:            out_dest = in_dest;
152:            comp_ack = 1;
153:            valid = 1;
154:            wait (~comp_req);
155:            #1;
156:            wait (~halt_req);
157:            comp_ack = 0;
158:    end
159:
160:    always wait (valid & ~halt_req & ~reset)
161:    begin :output_cycle
162:            #1;
163:            compute;
164:            arb_req = 1;
165:            wait (arb_ack);
166:            #1;
167:            wait (~halt_req);
168:            out_req = 1;
169:            wait (out_ack);
170:            #1;
171:            valid = 0;
172:            arb_req = 0;
173:            wait (~halt_req);
174:            out_req = 0;
175:            wait (~out_ack & ~arb_ack);
176:    end
177:
178:    task compute;
179:    begin
180:        case (func)
181:            FUNC_ADDU, FUNC_ADDUF:
182:                        #DLY_ALU_ADD out_data = data1 + data2;
183:            FUNC_SUBU: #DLY_ALU_SUB out_data = data1 - data2;
184:
185:            FUNC_AND:  #DLY_ALU_AND   out_data = data1 & data2;
186:            FUNC_OR:   #DLY_ALU_OR    out_data = data1 | data2;
187:            FUNC_XOR:  #DLY_ALU_XOR   out_data = data1 ^ data2;
188:            FUNC_PASS: #DLY_ALU_PASS  out_data = data2;
189:            FUNC_LHI:  #DLY_ALU_SHIFT out_data = data2 << IMM_WIDTH;
190:            FUNC_SLL:  #DLY_ALU_SHIFT out_data = data1 << (data2 % DATA_WIDTH);
191:            FUNC_SRL:  #DLY_ALU_SHIFT out_data = data1 >> (data2 % DATA_WIDTH);
192:            FUNC_SRA:  #DLY_ALU_SHIFT
193:                    for (loop=0; loop<(data2 % DATA_WIDTH); loop=loop+1)
194:                        out_data = {data1[DATA_WIDTH-1], data1[DATA_WIDTH-1:1]};
195:
196:            FUNC_SLT: #DLY_ALU_COMP out_data = (data1 < data2);
197:            FUNC_SGT: #DLY_ALU_COMP out_data = (data1 > data2);
198:            FUNC_SLE: #DLY_ALU_COMP out_data = (data1 <= data2);
199:            FUNC_SGE: #DLY_ALU_COMP out_data = (data1 >= data2);
200:            FUNC_SEQ: #DLY_ALU_COMP out_data = (data1 == data2);
201:            FUNC_SNE: #DLY_ALU_COMP out_data = (data1 != data2);
202:
203:            default: out_data = XXX;
204:        endcase
205:
206:        parity = (func == FUNC_ADDUF);              // bad parity
207:        for (loop=0; loop<DATA_WIDTH; loop=loop+1)  // calculate parity
208:            parity = parity ^ out_data [loop];
209:        out_data [DATA_WIDTH] = parity;
210:    end
211:    endtask
212:
213:    endmodule       // aluf
```

```
214:
215:    // ====================================================================
216:    // ALU Input Queue
217:
218:    module aluq (in_seq, in_dest, in_d1, in_d2, in_func, in_req, in_ack,
219:            out_seq, out_dest, out_d1, out_d2, out_func, out_req, out_ack,
220:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
221:
222:    `include "parameter"
223:
224:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
225:    input [REG_WIDTH-1:0] in_dest;
226:    input [DATA_WIDTH:0] in_d1, in_d2;
227:    input [FUNC_WIDTH-1:0] in_func;
228:    output [SEQ_WIDTH-1:0] out_seq;
229:    output [REG_WIDTH-1:0] out_dest;
230:    output [DATA_WIDTH:0] out_d1, out_d2;
231:    output [FUNC_WIDTH-1:0] out_func;
232:    input in_req, out_ack, halt_req, roll_req, reset;
233:    output in_ack, out_req, halt_ack, roll_ack;
234:
235:    reg halt_ack, roll_ack;
236:
237:    wire [SEQ_WIDTH-1:0] seq1;
238:    wire [REG_WIDTH-1:0] dest1;
239:    wire [DATA_WIDTH:0] d1_1, d2_1;
240:    wire [FUNC_WIDTH-1:0] func1;
241:
242:    aluq_buffer buf1 (in_seq, in_dest, in_d1, in_d2, in_func, in_req, in_ack,
243:            seq1, dest1, d1_1, d2_1, func1, req1, ack1,
244:            val_seq, halt_req, halt_a1, roll_req, roll_a1, reset);
245:    aluq_buffer buf2 (seq1, dest1, d1_1, d2_1, func1, req1, ack1, out_seq,
246:            out_dest, out_d1, out_d2, out_func, out_req, out_ack,
247:            val_seq, halt_req, halt_a2, roll_req, roll_a2, reset);
248:
249:    always wait (reset)
250:    begin
251:            halt_ack = 0;
252:            roll_ack = 0;
253:            wait (~reset);
254:    end
255:
256:    always @(halt_a1 or halt_a2)
257:            if (halt_a1 & halt_a2)
258:                    halt_ack = 1;
259:            else if (~halt_a1 & ~halt_a2)
260:                    halt_ack = 0;
261:
262:    always @(roll_a1 or roll_a2)
263:            if (roll_a1 & roll_a2)
264:                    roll_ack = 1;
265:            else if (~roll_a1 & ~roll_a2)
266:                    roll_ack = 0;
267:
268:    endmodule       // aluq
269:
270:    // ====================================================================
271:
272:    module aluq_buffer (in_s, in_ds, in_d1, in_d2, in_f, in_r, in_a,
273:            out_s, out_ds, out_d1, out_d2, out_f, out_r, out_a,
274:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
275:
276:    `include "parameter"
277:
278:    input [SEQ_WIDTH-1:0] in_s, val_seq;
279:    input [REG_WIDTH-1:0] in_ds;
280:    input [DATA_WIDTH:0] in_d1, in_d2;
281:    input [FUNC_WIDTH-1:0] in_f;
282:    output [SEQ_WIDTH-1:0] out_s;
283:    output [REG_WIDTH-1:0] out_ds;
284:    output [DATA_WIDTH:0] out_d1, out_d2;
```

```
285:    output [FUNC_WIDTH-1:0] out_f;
286:    input in_r, out_a, halt_req, roll_req, reset;
287:    output in_a, out_r, halt_ack, roll_ack;
288:
289:    reg [SEQ_WIDTH-1:0] out_s;
290:    reg [REG_WIDTH-1:0] out_ds;
291:    reg [DATA_WIDTH:0] out_d1, out_d2;
292:    reg [FUNC_WIDTH-1:0] out_f;
293:    reg in_a, out_r, halt_ack, roll_ack;
294:
295:    reg [SEQ_WIDTH-1:0] diff;
296:    reg valid;
297:
298:    always wait (reset)
299:    begin
300:            disable rollback_cycle;
301:            disable input_cycle;
302:            disable output_cycle;
303:            out_s = XXX;
304:            out_ds = XXX;
305:            out_d1 = XXX;
306:            out_d2 = XXX;
307:            out_f = XXX;
308:            in_a = 0;
309:            out_r = 0;
310:            halt_ack = 0;
311:            roll_ack = 0;
312:            valid = 0;
313:            wait (~reset);
314:    end
315:
316:    always wait (halt_req & ~reset)
317:    begin :rollback_cycle
318:            #1;
319:            halt_ack = 1;
320:            wait (roll_req);
321:            #1;
322:            disable input_cycle;
323:            disable output_cycle;
324:            in_a = 0;
325:            out_r = 0;
326:
327:            #DLY_SEQ_COMP;
328:            diff = out_s - val_seq;       // compare sequence numbers
329:            if (~diff [SEQ_WIDTH-1])
330:                begin
331:                    valid = 0;
332:                    out_s = XXX;
333:                    out_ds = XXX;
334:                    out_d1 = XXX;
335:                    out_d2 = XXX;
336:                    out_f = XXX;
337:                end
338:
339:            roll_ack = 1;
340:            fork
341:                begin
342:                    wait (~roll_req);
343:                    #1;
344:                    roll_ack = 0;
345:                end
346:                begin
347:                    wait (~halt_req);
348:                    #1;
349:                    halt_ack = 0;
350:                end
351:            join
352:    end
353:
354:    always wait (in_r & ~valid & ~halt_req & ~reset)
355:    begin :input_cycle
```

```
356:            #1;
357:            wait (~halt_req);
358:            out_s = in_s;
359:            out_ds = in_ds;
360:            out_d1 = in_d1;
361:            out_d2 = in_d2;
362:            out_f = in_f;
363:            in_a = 1;
364:            valid = 1;
365:            wait (~in_r);
366:            #1;
367:            wait (~halt_req);
368:            in_a = 0;
369:    end
370:
371:    always wait (valid & ~halt_req & ~reset)
372:    begin :output_cycle
373:            #1;
374:            wait (~halt_req);
375:            out_r = 1;
376:            wait (out_a);
377:            #1;
378:            valid = 0;
379:            wait (~halt_req);
380:            out_r = 0;
381:            wait (~out_a);
382:    end
383:
384:    endmodule       // aluq_buffer
```

arbk.v

```
 1:    // Arbiter for K_bus
 2:
 3:    module arbk (req0, ack0, req1, ack1, req2, ack2, req3, ack3,
 4:            halt_req, halt_ack, roll_req, roll_ack, reset);
 5:
 6:    `include "parameter"
 7:
 8:    input req0, req1, req2, req3, halt_req, roll_req, reset;
 9:    output ack0, ack1, ack2, ack3, halt_ack, roll_ack;
10:
11:    reg ack0, ack1, ack2, ack3, halt_ack, roll_ack;
12:
13:    reg [CHKID_WIDTH-1:0] grant;
14:    reg found;
15:
16:    always wait (reset)
17:    begin
18:            disable rollback_cycle;
19:            disable arb_cycle;
20:            ack0 = 0;
21:            ack1 = 0;
22:            ack2 = 0;
23:            ack3 = 0;
24:            halt_ack = 0;
25:            roll_ack = 0;
26:            grant = 0;
27:            wait (~reset);
28:    end
29:
30:    always wait (halt_req & ~reset)
31:    begin :rollback_cycle
32:            #1;
33:            halt_ack = 1;
34:            wait (roll_req);
35:            #1;
36:            disable arb_cycle;
37:            ack0 = 0;
38:            ack1 = 0;
39:            ack2 = 0;
40:            ack3 = 0;
41:            grant = 0;
42:
43:            roll_ack = 1;
44:            fork
45:                begin
46:                    wait (~roll_req);
47:                    #1;
48:                    roll_ack = 0;
49:                end
50:                begin
51:                    wait (~halt_req);
52:                    #1;
53:                    halt_ack = 0;
54:                end
55:            join
56:    end
57:
58:    always wait ((req0 | req1 | req2 | req3) & ~halt_req & ~reset)
59:    begin :arb_cycle
60:            found = 0;
61:            while (~found)
62:                begin
63:                    grant = grant + 1;              // scan request signals
64:                    if (grant >= CHECKERS)
65:                            grant = 0;
66:                    case (grant)
67:                            0: if (req0)  found = 1;
68:                            1: if (req1)  found = 1;
69:                            2: if (req2)  found = 1;
70:                            3: if (req3)  found = 1;
71:                    endcase
```

```
 72:                 end
 73:
 74:             #1;
 75:             case (grant)
 76:                     0:  begin
 77:                             ack0 = 1;
 78:                             wait (~req0);
 79:                             #1;
 80:                             ack0 = 0;
 81:                         end
 82:                     1:  begin
 83:                             ack1 = 1;
 84:                             wait (~req1);
 85:                             #1;
 86:                             ack1 = 0;
 87:                         end
 88:                     2:  begin
 89:                             ack2 = 1;
 90:                             wait (~req2);
 91:                             #1;
 92:                             ack2 = 0;
 93:                         end
 94:                     3:  begin
 95:                             ack3 = 1;
 96:                             wait (~req3);
 97:                             #1;
 98:                             ack3 = 0;
 99:                         end
100:             endcase
101:    end
102:
103:    endmodule       // arbk
```

arbr.v

```
 1:    // Arbiter for R_bus
 2:
 3:    module arbr (req0, ack0, req1, ack1,
 4:            halt_req, halt_ack, roll_req, roll_ack, reset);
 5:
 6:    `include "parameter"
 7:
 8:    input req0, req1, halt_req, roll_req, reset;
 9:    output ack0, ack1, halt_ack, roll_ack;
10:
11:    `ifdef BEHAV_ARBR       // behavioral level
12:
13:    reg ack0, ack1, halt_ack, roll_ack;
14:
15:    reg grant;
16:
17:    always wait (reset)
18:    begin
19:            disable rollback_cycle;
20:            disable arb_cycle;
21:            ack0 = 0;
22:            ack1 = 0;
23:            halt_ack = 0;
24:            roll_ack = 0;
25:            grant = 0;
26:            wait (~reset);
27:    end
28:
29:    always wait (halt_req & ~reset)
30:    begin :rollback_cycle
31:            #1;
32:            halt_ack = 1;
33:            wait (roll_req);
34:            #1;
35:            disable arb_cycle;
36:            ack0 = 0;
37:            ack1 = 0;
38:            grant = 0;
39:
40:            roll_ack = 1;
41:            fork
42:                begin
43:                    wait (~roll_req);
44:                    #1;
45:                    roll_ack = 0;
46:                end
47:                begin
48:                    wait (~halt_req);
49:                    #1;
50:                    halt_ack = 0;
51:                end
52:            join
53:    end
54:
55:    always wait ((req0 | req1) & ~halt_req & ~reset)
56:    begin :arb_cycle
57:            if (req0 & req1)
58:                    grant = ~grant;         // switch to other one
59:            else
60:                    grant = req1;           // 0 if not req1
61:
62:            #1;
63:            if (grant)
64:                begin
65:                    ack1 = 1;
66:                    wait (~req1);
67:                    #1;
68:                    ack1 = 0;
69:                end
70:            else
71:                begin
```

```
 72:                     ack0 = 1;
 73:                     wait (~req0);
 74:                     #1;
 75:                     ack0 = 0;
 76:                 end
 77:     end
 78:
 79:     `else   // gate level
 80:
 81:     and     #1 g1 (v0, req0, ~v1);
 82:     and     #2 g2 (v1, req1, ~v0);
 83:     not     #1 g3 (ack0, nack0);
 84:     not     #1 g4 (ack1, nack1);
 85:     nmos    #1 m1 (nack1, v0, v1);
 86:     nmos    #1 m2 (nack0, v1, v0);
 87:     pullup (nack0), (nack1);
 88:
 89:     buf     #1 g5 (halt_ack, halt_req);
 90:     and     #1 g6 (roll_ack, roll_req, ~ack0, ~ack1);
 91:
 92:     `ifdef WAVE_ARBR
 93:     initial
 94:             #0 $gr_addwaves ("req0", req0, "v0", v0, "ack0", ack0,
 95:                     "req1", req1, "v1", v1, "ack1", ack1);
 96:     `endif
 97:
 98:     `endif  // BEHAV_ARBR
 99:
100:     endmodule       // arbr
```

bigc.v

```
 1:    // Big C-element: used for halt_ack and roll_ack
 2:
 3:    module bigc (in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12,
 4:            in13, in14, out, reset);
 5:
 6:    input in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13,
 7:            in14;
 8:    output out;
 9:    input reset;
10:
11:    `ifdef BEHAV_BIGC       // behavioral level
12:
13:    reg out;
14:
15:    always wait (reset)
16:    begin
17:            out = 0;
18:            wait (~reset);
19:    end
20:
21:    always @(in1 or in2 or in3 or in4 or in5 or in6 or in7 or in8 or in9 or
22:             in10 or in11 or in12 or in13 or in14)
23:            if (in1 & in2 & in3 & in4 & in5 & in6 & in7 & in8 & in9 & in10 &
24:                in11 & in12 & in13 & in14)
25:                    out = 1;
26:            else if (~in1 & ~in2 & ~in3 & ~in4 & ~in5 & ~in6 & ~in7 & ~in8 &
27:                     ~in9 & ~in10 & ~in11 & ~in12 & ~in13 & ~in14)
28:                    out = 0;
29:
30:    `else   // gate level
31:
32:    muller4 x1 (in1, in2, in3, in4, out1);
33:    muller4 x2 (in5, in6, in7, in8, out2);
34:    muller3 x3 (in9, in10, in11, out3);
35:    muller3 x4 (in12, in13, in14, out4);
36:    muller4 x5 (out1, out2, out3, out4, out);
37:
38:    `endif
39:
40:    endmodule       // bigc
```

ckf.v

```
 1:    // CKF (combining CKFQ and CKFF)
 2:
 3:    module ckf (in_seq, in_data1, in_data2, in_req, in_ack, arb_req, arb_ack,
 4:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
10:    input [DATA_WIDTH:0] in_data1, in_data2;
11:    output [SEQ_WIDTH-1:0] tri_seq;
12:    output [CHKID_WIDTH-1:0] tri_chkid;
13:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
14:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
15:
16:    reg halt_ack, roll_ack;
17:
18:    wire [SEQ_WIDTH-1:0] mid_seq;
19:    wire [DATA_WIDTH:0] mid_data1, mid_data2;
20:
21:    ckfq ckfq (in_seq, in_data1, in_data2, in_req, in_ack,
22:            mid_seq, mid_data1, mid_data2, mid_req, mid_ack,
23:            val_seq, halt_req, haltq_ack, roll_req, rollq_ack, reset);
24:    ckff ckff (mid_seq, mid_data1, mid_data2, mid_req, mid_ack, arb_req, arb_ack,
25:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
26:            val_seq, halt_req, haltf_ack, roll_req, rollf_ack, reset);
27:
28:    always wait (reset)
29:    begin
30:            halt_ack = 0;
31:            roll_ack = 0;
32:            wait (~reset);
33:    end
34:
35:    always @(haltq_ack or haltf_ack)
36:            if (haltq_ack & haltf_ack)
37:                    halt_ack = 1;
38:            else if (~haltq_ack & ~haltf_ack)
39:                    halt_ack = 0;
40:
41:    always @(rollq_ack or rollf_ack)
42:            if (rollq_ack & rollf_ack)
43:                    roll_ack = 1;
44:            else if (~rollq_ack & ~rollf_ack)
45:                    roll_ack = 0;
46:
47:    endmodule       // ckf
48:
49:    // ====================================================================
50:    // Checker for Register File (functional part)
51:
52:    module ckff (in_seq, in_data1, in_data2, in_req, in_ack, arb_req, arb_ack,
53:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
54:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
55:
56:    `include "parameter"
57:
58:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
59:    input [DATA_WIDTH:0] in_data1, in_data2;
60:    output [SEQ_WIDTH-1:0] tri_seq;
61:    output [CHKID_WIDTH-1:0] tri_chkid;
62:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
63:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
64:
65:    reg [SEQ_WIDTH-1:0] out_seq;
66:    reg in_ack, arb_req, out_error, out_req, halt_ack, roll_ack;
67:
68:    reg [SEQ_WIDTH-1:0] diff;
69:    reg [DATA_WIDTH:0] data1_buf, data2_buf;
70:    reg [5:0] loop;
71:    reg valid, parity1, parity2;
```

```
 72:
 73:    assign tri_seq = arb_ack ? out_seq : ZZZ;
 74:    assign tri_chkid = arb_ack ? ID_CKF : ZZZ;
 75:    assign tri_error = arb_ack ? out_error : ZZZ;
 76:    assign tri_req = arb_ack ? out_req : ZZZ;
 77:
 78:    always wait (reset)
 79:    begin
 80:            disable rollback_cycle;
 81:            disable latch_cycle;
 82:            disable output_cycle;
 83:            out_seq = XXX;
 84:            in_ack = 0;
 85:            arb_req = 0;
 86:            out_error = XXX;
 87:            out_req = 0;
 88:            halt_ack = 0;
 89:            roll_ack = 0;
 90:            valid = 0;
 91:            wait (~reset);
 92:    end
 93:
 94:    always wait (halt_req & ~reset)
 95:    begin :rollback_cycle
 96:            #1;
 97:            halt_ack = 1;
 98:            wait (roll_req);
 99:            #1;
100:            disable latch_cycle;
101:            disable output_cycle;
102:            in_ack = 0;
103:            arb_req = 0;
104:            out_req = 0;
105:
106:            #DLY_SEQ_COMP;
107:            diff = out_seq - val_seq;       // compare sequence numbers
108:            if (~diff [SEQ_WIDTH-1])
109:                begin
110:                    valid = 0;
111:                    out_seq = XXX;
112:                    out_error = XXX;
113:                end
114:
115:            roll_ack = 1;
116:            fork
117:                begin
118:                    wait (~roll_req);
119:                    #1;
120:                    roll_ack = 0;
121:                end
122:                begin
123:                    wait (~halt_req);
124:                    #1;
125:                    halt_ack = 0;
126:                end
127:            join
128:    end
129:
130:    always wait (in_req & ~valid & ~halt_req & ~reset)
131:    begin :latch_cycle
132:            #1;
133:            wait (~halt_req);
134:            out_seq = in_seq;
135:            data1_buf = in_data1;
136:            data2_buf = in_data2;
137:            in_ack = 1;
138:            valid = 1;
139:            wait (~in_req);
140:            #1;
141:            wait (~halt_req);
142:            in_ack = 0;
```

```
143:    end
144:
145:    always wait (valid & ~halt_req & ~reset)
146:    begin :output_cycle
147:            #1;
148:            parity1 = 0;
149:            parity2 = 0;
150:            for (loop=0; loop<1+DATA_WIDTH; loop=loop+1)  // calculate parity
151:                begin
152:                    parity1 = parity1 ^ data1_buf [loop];
153:                    parity2 = parity2 ^ data2_buf [loop];
154:                end
155:            if (parity1 === 1'bx)  parity1 = 1;
156:            if (parity2 === 1'bx)  parity2 = 1;
157:
158:            #DLY_CKF_CHK;
159:            out_error = parity1 | parity2;
160:            arb_req = 1;
161:            wait (arb_ack);
162:            #1;
163:            wait (~halt_req);
164:            out_req = 1;
165:            wait (out_ack);
166:            #1;
167:            valid = 0;
168:            arb_req = 0;
169:            wait (~halt_req);
170:            out_req = 0;
171:            wait (~out_ack & ~arb_ack);
172:    end
173:
174:    endmodule       // ckff
175:
176:    // ============================================================================
177:    // Queue for Register File Checker
178:
179:    module ckfq (in_seq, in_data1, in_data2, in_req, in_ack,
180:            out_seq, out_data1, out_data2, out_req, out_ack,
181:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
182:
183:    `include "parameter"
184:
185:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
186:    input [DATA_WIDTH:0] in_data1, in_data2;
187:    output [SEQ_WIDTH-1:0] out_seq;
188:    output [DATA_WIDTH:0] out_data1, out_data2;
189:    input in_req, out_ack, halt_req, roll_req, reset;
190:    output in_ack, out_req, halt_ack, roll_ack;
191:
192:    reg halt_ack, roll_ack;
193:
194:    wire [SEQ_WIDTH-1:0] seq1;
195:    wire [DATA_WIDTH:0] data11, data21;
196:
197:    ckf_buf buf1 (in_seq, in_data1, in_data2, in_req, in_ack,
198:            seq1, data11, data21, req1, ack1,
199:            val_seq, halt_req, halt_a1, roll_req, roll_a1, reset);
200:    ckf_buf buf2 (seq1, data11, data21, req1, ack1,
201:            out_seq, out_data1, out_data2, out_req, out_ack,
202:            val_seq, halt_req, halt_a2, roll_req, roll_a2, reset);
203:
204:    always wait (reset)
205:    begin
206:            halt_ack = 0;
207:            roll_ack = 0;
208:            wait (~reset);
209:    end
210:
211:    always @(halt_a1 or halt_a2)
212:            if (halt_a1 & halt_a2)
213:                    halt_ack = 1;
```

```
214:            else if (~halt_a1 & ~halt_a2)
215:                    halt_ack = 0;
216:
217:    always @(roll_a1 or roll_a2)
218:            if (roll_a1 & roll_a2)
219:                    roll_ack = 1;
220:            else if (~roll_a1 & ~roll_a2)
221:                    roll_ack = 0;
222:
223:    endmodule       // ckfq
224:
225:    // ===========================================================================
226:
227:    module ckf_buf (in_s, in_d1, in_d2, in_r, in_a,
228:            out_s, out_d1, out_d2, out_r, out_a,
229:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
230:
231:    `include "parameter"
232:
233:    input [SEQ_WIDTH-1:0] in_s, val_seq;
234:    input [DATA_WIDTH:0] in_d1, in_d2;
235:    output [SEQ_WIDTH-1:0] out_s;
236:    output [DATA_WIDTH:0] out_d1, out_d2;
237:    input in_r, out_a, halt_req, roll_req, reset;
238:    output in_a, out_r, halt_ack, roll_ack;
239:
240:    reg [SEQ_WIDTH-1:0] out_s;
241:    reg [DATA_WIDTH:0] out_d1, out_d2;
242:    reg in_a, out_r, halt_ack, roll_ack;
243:
244:    reg [SEQ_WIDTH-1:0] diff;
245:    reg valid;
246:
247:    always wait (reset)
248:    begin
249:            disable rollback_cycle;
250:            disable input_cycle;
251:            disable output_cycle;
252:            out_s = XXX;
253:            out_d1 = XXX;
254:            out_d2 = XXX;
255:            in_a = 0;
256:            out_r = 0;
257:            halt_ack = 0;
258:            roll_ack = 0;
259:            valid = 0;
260:            wait (~reset);
261:    end
262:
263:    always wait (halt_req & ~reset)
264:    begin :rollback_cycle
265:            #1;
266:            halt_ack = 1;
267:            wait (roll_req);
268:            #1;
269:            disable input_cycle;
270:            disable output_cycle;
271:            in_a = 0;
272:            out_r = 0;
273:
274:            #DLY_SEQ_COMP;
275:            diff = out_s - val_seq;       // compare sequence numbers
276:            if (~diff [SEQ_WIDTH-1])
277:                begin
278:                    valid = 0;
279:                    out_s = XXX;
280:                    out_d1 = XXX;
281:                    out_d2 = XXX;
282:                end
283:
284:            roll_ack = 1;
```

```
285:            fork
286:                begin
287:                    wait (~roll_req);
288:                    #1;
289:                    roll_ack = 0;
290:                end
291:                begin
292:                    wait (~halt_req);
293:                    #1;
294:                    halt_ack = 0;
295:                end
296:            join
297:    end
298:
299:    always wait (in_r & ~valid & ~halt_req & ~reset)
300:    begin :input_cycle
301:            #1;
302:            wait (~halt_req);
303:            out_s = in_s;
304:            out_d1 = in_d1;
305:            out_d2 = in_d2;
306:            in_a = 1;
307:            valid = 1;
308:            wait (~in_r);
309:            #1;
310:            wait (~halt_req);
311:            in_a = 0;
312:    end
313:
314:    always wait (valid & ~halt_req & ~reset)
315:    begin :output_cycle
316:            #1;
317:            wait (~halt_req);
318:            out_r = 1;
319:            wait (out_a);
320:            #1;
321:            valid = 0;
322:            wait (~halt_req);
323:            out_r = 0;
324:            wait (~out_a);
325:    end
326:
327:    endmodule       // ckf_buf
```

cki.v

```
 1:    // CKI (combining CKIQ and CKIF)
 2:
 3:    module cki (in_seq, in_data, in_req, in_ack, arb_req, arb_ack,
 4:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
10:    input [DATA_WIDTH:0] in_data;
11:    output [SEQ_WIDTH-1:0] tri_seq;
12:    output [CHKID_WIDTH-1:0] tri_chkid;
13:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
14:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
15:
16:    reg halt_ack, roll_ack;
17:
18:    wire [SEQ_WIDTH-1:0] mid_seq;
19:    wire [DATA_WIDTH:0] mid_data;
20:
21:    ckiq ckiq (in_seq, in_data, in_req, in_ack,
22:            mid_seq, mid_data, mid_req, mid_ack,
23:            val_seq, halt_req, haltq_ack, roll_req, rollq_ack, reset);
24:    ckif ckif (mid_seq, mid_data, mid_req, mid_ack, arb_req, arb_ack,
25:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
26:            val_seq, halt_req, haltf_ack, roll_req, rollf_ack, reset);
27:
28:    always wait (reset)
29:    begin
30:            halt_ack = 0;
31:            roll_ack = 0;
32:            wait (~reset);
33:    end
34:
35:    always @(haltq_ack or haltf_ack)
36:            if (haltq_ack & haltf_ack)
37:                    halt_ack = 1;
38:            else if (~haltq_ack & ~haltf_ack)
39:                    halt_ack = 0;
40:
41:    always @(rollq_ack or rollf_ack)
42:            if (rollq_ack & rollf_ack)
43:                    roll_ack = 1;
44:            else if (~rollq_ack & ~rollf_ack)
45:                    roll_ack = 0;
46:
47:    endmodule       // cki
48:
49:    // ====================================================================
50:    // Checker for I_bus (functional part)
51:
52:    module ckif (in_seq, in_data, in_req, in_ack, arb_req, arb_ack,
53:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
54:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
55:
56:    `include "parameter"
57:
58:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
59:    input [DATA_WIDTH:0] in_data;
60:    output [SEQ_WIDTH-1:0] tri_seq;
61:    output [CHKID_WIDTH-1:0] tri_chkid;
62:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
63:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
64:
65:    reg [SEQ_WIDTH-1:0] out_seq;
66:    reg in_ack, arb_req, out_error, out_req, halt_ack, roll_ack;
67:
68:    reg [SEQ_WIDTH-1:0] diff;
69:    reg [DATA_WIDTH:0] data_buf;
70:    reg [5:0] loop;
71:    reg valid, parity;
```

```
 72:
 73:    assign tri_seq = arb_ack ? out_seq : ZZZ;
 74:    assign tri_chkid = arb_ack ? ID_CKI : ZZZ;
 75:    assign tri_error = arb_ack ? out_error : ZZZ;
 76:    assign tri_req = arb_ack ? out_req : ZZZ;
 77:
 78:    always wait (reset)
 79:    begin
 80:            disable rollback_cycle;
 81:            disable latch_cycle;
 82:            disable output_cycle;
 83:            out_seq = XXX;
 84:            in_ack = 0;
 85:            arb_req = 0;
 86:            out_error = XXX;
 87:            out_req = 0;
 88:            halt_ack = 0;
 89:            roll_ack = 0;
 90:            valid = 0;
 91:            wait (~reset);
 92:    end
 93:
 94:    always wait (halt_req & ~reset)
 95:    begin :rollback_cycle
 96:            #1;
 97:            halt_ack = 1;
 98:            wait (roll_req);
 99:            #1;
100:            disable latch_cycle;
101:            disable output_cycle;
102:            in_ack = 0;
103:            arb_req = 0;
104:            out_req = 0;
105:
106:            #DLY_SEQ_COMP;
107:            diff = out_seq - val_seq;       // compare sequence numbers
108:            if (~diff [SEQ_WIDTH-1])
109:                begin
110:                    valid = 0;
111:                    out_seq = XXX;
112:                    out_error = XXX;
113:                end
114:
115:            roll_ack = 1;
116:            fork
117:                begin
118:                    wait (~roll_req);
119:                    #1;
120:                    roll_ack = 0;
121:                end
122:                begin
123:                    wait (~halt_req);
124:                    #1;
125:                    halt_ack = 0;
126:                end
127:            join
128:    end
129:
130:    always wait (in_req & ~valid & ~halt_req & ~reset)
131:    begin :latch_cycle
132:            #1;
133:            wait (~halt_req);
134:            out_seq = in_seq;
135:            data_buf = in_data;
136:            in_ack = 1;
137:            valid = 1;
138:            wait (~in_req);
139:            #1;
140:            wait (~halt_req);
141:            in_ack = 0;
142:    end
```

```
143:
144:    always wait (valid & ~halt_req & ~reset)
145:    begin :output_cycle
146:            #1;
147:            parity = 0;
148:            for (loop=0; loop<1+DATA_WIDTH; loop=loop+1)  // calculate parity
149:                    parity = parity ^ data_buf [loop];
150:            if (parity === 1'bx)  parity = 1;
151:
152:            #DLY_CKI_CHK;
153:            out_error = parity;
154:            arb_req = 1;
155:            wait (arb_ack);
156:            #1;
157:            wait (~halt_req);
158:            out_req = 1;
159:            wait (out_ack);
160:            #1;
161:            valid = 0;
162:            arb_req = 0;
163:            wait (~halt_req);
164:            out_req = 0;
165:            wait (~out_ack & ~arb_ack);
166:    end
167:
168:    endmodule       // ckif
169:
170:    // ====================================================================
171:    // Queue for I_bus Checker
172:
173:    module ckiq (in_seq, in_data, in_req, in_ack,
174:            out_seq, out_data, out_req, out_ack,
175:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
176:
177:    `include "parameter"
178:
179:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
180:    input [DATA_WIDTH:0] in_data;
181:    output [SEQ_WIDTH-1:0] out_seq;
182:    output [DATA_WIDTH:0] out_data;
183:    input in_req, out_ack, halt_req, roll_req, reset;
184:    output in_ack, out_req, halt_ack, roll_ack;
185:
186:    reg halt_ack, roll_ack;
187:
188:    wire [SEQ_WIDTH-1:0] seq1;
189:    wire [DATA_WIDTH:0] data1;
190:
191:    cki_buf buf1 (in_seq, in_data, in_req, in_ack, seq1, data1, req1, ack1,
192:                    val_seq, halt_req, halt_a1, roll_req, roll_a1, reset);
193:    cki_buf buf2 (seq1, data1, req1, ack1, out_seq, out_data, out_req, out_ack,
194:                    val_seq, halt_req, halt_a2, roll_req, roll_a2, reset);
195:
196:    always wait (reset)
197:    begin
198:            halt_ack = 0;
199:            roll_ack = 0;
200:            wait (~reset);
201:    end
202:
203:    always @(halt_a1 or halt_a2)
204:            if (halt_a1 & halt_a2)
205:                    halt_ack = 1;
206:            else if (~halt_a1 & ~halt_a2)
207:                    halt_ack = 0;
208:
209:    always @(roll_a1 or roll_a2)
210:            if (roll_a1 & roll_a2)
211:                    roll_ack = 1;
212:            else if (~roll_a1 & ~roll_a2)
213:                    roll_ack = 0;
```

```
214:
215:    endmodule       // ckiq
216:
217:    // ===========================================================================
218:
219:    module cki_buf (in_s, in_d, in_r, in_a, out_s, out_d, out_r, out_a,
220:                    val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
221:
222:    `include "parameter"
223:
224:    input [SEQ_WIDTH-1:0] in_s, val_seq;
225:    input [DATA_WIDTH:0] in_d;
226:    output [SEQ_WIDTH-1:0] out_s;
227:    output [DATA_WIDTH:0] out_d;
228:    input in_r, out_a, halt_req, roll_req, reset;
229:    output in_a, out_r, halt_ack, roll_ack;
230:
231:    reg [SEQ_WIDTH-1:0] out_s;
232:    reg [DATA_WIDTH:0] out_d;
233:    reg in_a, out_r, halt_ack, roll_ack;
234:
235:    reg [SEQ_WIDTH-1:0] diff;
236:    reg valid;
237:
238:    always wait (reset)
239:    begin
240:            disable rollback_cycle;
241:            disable input_cycle;
242:            disable output_cycle;
243:            out_s = XXX;
244:            out_d = XXX;
245:            in_a = 0;
246:            out_r = 0;
247:            halt_ack = 0;
248:            roll_ack = 0;
249:            valid = 0;
250:            wait (~reset);
251:    end
252:
253:    always wait (halt_req & ~reset)
254:    begin :rollback_cycle
255:            #1;
256:            halt_ack = 1;
257:            wait (roll_req);
258:            #1;
259:            disable input_cycle;
260:            disable output_cycle;
261:            in_a = 0;
262:            out_r = 0;
263:
264:            #DLY_SEQ_COMP;
265:            diff = out_s - val_seq;       // compare sequence numbers
266:            if (~diff [SEQ_WIDTH-1])
267:                begin
268:                    valid = 0;
269:                    out_s = XXX;
270:                    out_d = XXX;
271:                end
272:
273:            roll_ack = 1;
274:            fork
275:                begin
276:                    wait (~roll_req);
277:                    #1;
278:                    roll_ack = 0;
279:                end
280:                begin
281:                    wait (~halt_req);
282:                    #1;
283:                    halt_ack = 0;
284:                end
```

```
285:            join
286:    end
287:
288:    always wait (in_r & ~valid & ~halt_req & ~reset)
289:    begin :input_cycle
290:            #1;
291:            wait (~halt_req);
292:            out_s = in_s;
293:            out_d = in_d;
294:            in_a = 1;
295:            valid = 1;
296:            wait (~in_r);
297:            #1;
298:            wait (~halt_req);
299:            in_a = 0;
300:    end
301:
302:    always wait (valid & ~halt_req & ~reset)
303:    begin :output_cycle
304:            #1;
305:            wait (~halt_req);
306:            out_r = 1;
307:            wait (out_a);
308:            #1;
309:            valid = 0;
310:            wait (~halt_req);
311:            out_r = 0;
312:            wait (~out_a);
313:    end
314:
315:    endmodule       // cki_buf
```

ckm.v

```
 1:    // CKM (combining CKMQ and CKMF)
 2:
 3:    module ckm (in_seq, in_addr, in_data, in_req, in_ack, arb_req, arb_ack,
 4:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
10:    input [DATA_WIDTH:0] in_addr, in_data;
11:    output [SEQ_WIDTH-1:0] tri_seq;
12:    output [CHKID_WIDTH-1:0] tri_chkid;
13:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
14:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
15:
16:    reg halt_ack, roll_ack;
17:
18:    wire [ADDR_WIDTH:0] adpt_addr = in_addr;        // addr "adaptor"
19:    wire [SEQ_WIDTH-1:0] mid_seq;
20:    wire [ADDR_WIDTH:0] mid_addr;
21:    wire [DATA_WIDTH:0] mid_data;
22:
23:    ckmq ckmq (in_seq, adpt_addr, in_data, in_req, in_ack,
24:            mid_seq, mid_addr, mid_data, mid_req, mid_ack,
25:            val_seq, halt_req, haltq_ack, roll_req, rollq_ack, reset);
26:    ckmf ckmf (mid_seq, mid_addr, mid_data, mid_req, mid_ack, arb_req, arb_ack,
27:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
28:            val_seq, halt_req, haltf_ack, roll_req, rollf_ack, reset);
29:
30:    always wait (reset)
31:    begin
32:            halt_ack = 0;
33:            roll_ack = 0;
34:            wait (~reset);
35:    end
36:
37:    always @(haltq_ack or haltf_ack)
38:            if (haltq_ack & haltf_ack)
39:                    halt_ack = 1;
40:            else if (~haltq_ack & ~haltf_ack)
41:                    halt_ack = 0;
42:
43:    always @(rollq_ack or rollf_ack)
44:            if (rollq_ack & rollf_ack)
45:                    roll_ack = 1;
46:            else if (~rollq_ack & ~rollf_ack)
47:                    roll_ack = 0;
48:
49:    endmodule       // ckm
50:
51:    // ========================================================================
52:    // Checker for Memory Operations (functional part)
53:
54:    module ckmf (in_seq, in_addr, in_data, in_req, in_ack, arb_req, arb_ack,
55:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
56:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
57:
58:    `include "parameter"
59:
60:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
61:    input [ADDR_WIDTH:0] in_addr;
62:    input [DATA_WIDTH:0] in_data;
63:    output [SEQ_WIDTH-1:0] tri_seq;
64:    output [CHKID_WIDTH-1:0] tri_chkid;
65:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
66:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
67:
68:    reg [SEQ_WIDTH-1:0] out_seq;
69:    reg in_ack, arb_req, out_error, out_req, halt_ack, roll_ack;
70:
71:    reg [SEQ_WIDTH-1:0] diff;
```

```
 72:    reg [ADDR_WIDTH:0] addr_buf;
 73:    reg [DATA_WIDTH:0] data_buf;
 74:    reg [5:0] loop;
 75:    reg valid, parity1, parity2;
 76:
 77:    assign tri_seq = arb_ack ? out_seq : ZZZ;
 78:    assign tri_chkid = arb_ack ? ID_CKM : ZZZ;
 79:    assign tri_error = arb_ack ? out_error : ZZZ;
 80:    assign tri_req = arb_ack ? out_req : ZZZ;
 81:
 82:    always wait (reset)
 83:    begin
 84:            disable rollback_cycle;
 85:            disable latch_cycle;
 86:            disable output_cycle;
 87:            out_seq = XXX;
 88:            in_ack = 0;
 89:            arb_req = 0;
 90:            out_error = XXX;
 91:            out_req = 0;
 92:            halt_ack = 0;
 93:            roll_ack = 0;
 94:            valid = 0;
 95:            wait (~reset);
 96:    end
 97:
 98:    always wait (halt_req & ~reset)
 99:    begin :rollback_cycle
100:            #1;
101:            halt_ack = 1;
102:            wait (roll_req);
103:            #1;
104:            disable latch_cycle;
105:            disable output_cycle;
106:            in_ack = 0;
107:            arb_req = 0;
108:            out_req = 0;
109:
110:            #DLY_SEQ_COMP;
111:            diff = out_seq - val_seq;       // compare sequence numbers
112:            if (~diff [SEQ_WIDTH-1])
113:                begin
114:                    valid = 0;
115:                    out_seq = XXX;
116:                    out_error = XXX;
117:                end
118:
119:            roll_ack = 1;
120:            fork
121:                begin
122:                    wait (~roll_req);
123:                    #1;
124:                    roll_ack = 0;
125:                end
126:                begin
127:                    wait (~halt_req);
128:                    #1;
129:                    halt_ack = 0;
130:                end
131:            join
132:    end
133:
134:    always wait (in_req & ~valid & ~halt_req & ~reset)
135:    begin :latch_cycle
136:            #1;
137:            wait (~halt_req);
138:            out_seq = in_seq;
139:            addr_buf = in_addr;
140:            data_buf = in_data;
141:            in_ack = 1;
142:            valid = 1;
```

```
143:            wait (~in_req);
144:            #1;
145:            wait (~halt_req);
146:            in_ack = 0;
147:    end
148:
149:    always wait (valid & ~halt_req & ~reset)
150:    begin :output_cycle
151:            #1;
152:            parity1 = 0;
153:            for (loop=0; loop<1+ADDR_WIDTH; loop=loop+1)  // calculate parity
154:                    parity1 = parity1 ^ addr_buf [loop];
155:            if (parity1 === 1'bx)  parity1 = 1;
156:            parity2 = 0;
157:            for (loop=0; loop<1+DATA_WIDTH; loop=loop+1)  // calculate parity
158:                    parity2 = parity2 ^ data_buf [loop];
159:            if (parity2 === 1'bx)  parity2 = 1;
160:
161:            #DLY_CKM_CHK;
162:            out_error = parity1 | parity2;
163:            arb_req = 1;
164:            wait (arb_ack);
165:            #1;
166:            wait (~halt_req);
167:            out_req = 1;
168:            wait (out_ack);
169:            #1;
170:            valid = 0;
171:            arb_req = 0;
172:            wait (~halt_req);
173:            out_req = 0;
174:            wait (~out_ack & ~arb_ack);
175:    end
176:
177:    endmodule       // ckmf
178:
179:    // ========================================================================
180:    // Queue for Memory Operations Checker
181:
182:    module ckmq (in_seq, in_addr, in_data, in_req, in_ack,
183:            out_seq, out_addr, out_data, out_req, out_ack,
184:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
185:
186:    `include "parameter"
187:
188:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
189:    input [ADDR_WIDTH:0] in_addr;
190:    input [DATA_WIDTH:0] in_data;
191:    output [SEQ_WIDTH-1:0] out_seq;
192:    output [ADDR_WIDTH:0] out_addr;
193:    output [DATA_WIDTH:0] out_data;
194:    input in_req, out_ack, halt_req, roll_req, reset;
195:    output in_ack, out_req, halt_ack, roll_ack;
196:
197:    reg halt_ack, roll_ack;
198:
199:    wire [SEQ_WIDTH-1:0] seq1;
200:    wire [ADDR_WIDTH:0] addr1;
201:    wire [DATA_WIDTH:0] data1;
202:
203:    ckm_buf buf1 (in_seq, in_addr, in_data, in_req, in_ack,
204:            seq1, addr1, data1, req1, ack1,
205:            val_seq, halt_req, halt_a1, roll_req, roll_a1, reset);
206:    ckm_buf buf2 (seq1, addr1, data1, req1, ack1,
207:            out_seq, out_addr, out_data, out_req, out_ack,
208:            val_seq, halt_req, halt_a2, roll_req, roll_a2, reset);
209:
210:    always wait (reset)
211:    begin
212:            halt_ack = 0;
213:            roll_ack = 0;
```

```
214:            wait (~reset);
215:    end
216:
217:    always @(halt_a1 or halt_a2)
218:            if (halt_a1 & halt_a2)
219:                    halt_ack = 1;
220:            else if (~halt_a1 & ~halt_a2)
221:                    halt_ack = 0;
222:
223:    always @(roll_a1 or roll_a2)
224:            if (roll_a1 & roll_a2)
225:                    roll_ack = 1;
226:            else if (~roll_a1 & ~roll_a2)
227:                    roll_ack = 0;
228:
229:    endmodule       // ckmq
230:
231:    // ====================================================================
232:
233:    module ckm_buf (in_s, in_d1, in_d2, in_r, in_a,
234:            out_s, out_d1, out_d2, out_r, out_a,
235:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
236:
237:    `include "parameter"
238:
239:    input [SEQ_WIDTH-1:0] in_s, val_seq;
240:    input [ADDR_WIDTH:0] in_d1;
241:    input [DATA_WIDTH:0] in_d2;
242:    output [SEQ_WIDTH-1:0] out_s;
243:    output [ADDR_WIDTH:0] out_d1;
244:    output [DATA_WIDTH:0] out_d2;
245:    input in_r, out_a, halt_req, roll_req, reset;
246:    output in_a, out_r, halt_ack, roll_ack;
247:
248:    reg [SEQ_WIDTH-1:0] out_s;
249:    reg [ADDR_WIDTH:0] out_d1;
250:    reg [DATA_WIDTH:0] out_d2;
251:    reg in_a, out_r, halt_ack, roll_ack;
252:
253:    reg [SEQ_WIDTH-1:0] diff;
254:    reg valid;
255:
256:    always wait (reset)
257:    begin
258:            disable rollback_cycle;
259:            disable input_cycle;
260:            disable output_cycle;
261:            out_s = XXX;
262:            out_d1 = XXX;
263:            out_d2 = XXX;
264:            in_a = 0;
265:            out_r = 0;
266:            halt_ack = 0;
267:            roll_ack = 0;
268:            valid = 0;
269:            wait (~reset);
270:    end
271:
272:    always wait (halt_req & ~reset)
273:    begin :rollback_cycle
274:            #1;
275:            halt_ack = 1;
276:            wait (roll_req);
277:            #1;
278:            disable input_cycle;
279:            disable output_cycle;
280:            in_a = 0;
281:            out_r = 0;
282:
283:            #DLY_SEQ_COMP;
284:            diff = out_s - val_seq;       // compare sequence numbers
```

```
285:            if (~diff [SEQ_WIDTH-1])
286:                begin
287:                    valid = 0;
288:                    out_s = XXX;
289:                    out_d1 = XXX;
290:                    out_d2 = XXX;
291:                end
292:
293:            roll_ack = 1;
294:            fork
295:                begin
296:                    wait (~roll_req);
297:                    #1;
298:                    roll_ack = 0;
299:                end
300:                begin
301:                    wait (~halt_req);
302:                    #1;
303:                    halt_ack = 0;
304:                end
305:            join
306:    end
307:
308:    always wait (in_r & ~valid & ~halt_req & ~reset)
309:    begin :input_cycle
310:            #1;
311:            wait (~halt_req);
312:            out_s = in_s;
313:            out_d1 = in_d1;
314:            out_d2 = in_d2;
315:            in_a = 1;
316:            valid = 1;
317:            wait (~in_r);
318:            #1;
319:            wait (~halt_req);
320:            in_a = 0;
321:    end
322:
323:    always wait (valid & ~halt_req & ~reset)
324:    begin :output_cycle
325:            #1;
326:            wait (~halt_req);
327:            out_r = 1;
328:            wait (out_a);
329:            #1;
330:            valid = 0;
331:            wait (~halt_req);
332:            out_r = 0;
333:            wait (~out_a);
334:    end
335:
336:    endmodule       // ckm_buf
```

ckr.v

```
 1:    // CKR (combining CKRQ and CKRF)
 2:
 3:    module ckr (in_seq, in_data, in_req, in_ack, arb_req, arb_ack,
 4:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
10:    input [DATA_WIDTH:0] in_data;
11:    output [SEQ_WIDTH-1:0] tri_seq;
12:    output [CHKID_WIDTH-1:0] tri_chkid;
13:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
14:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
15:
16:    reg halt_ack, roll_ack;
17:
18:    wire [SEQ_WIDTH-1:0] mid_seq;
19:    wire [DATA_WIDTH:0] mid_data;
20:
21:    ckrq ckrq (in_seq, in_data, in_req, in_ack,
22:            mid_seq, mid_data, mid_req, mid_ack,
23:            val_seq, halt_req, haltq_ack, roll_req, rollq_ack, reset);
24:    ckrf ckrf (mid_seq, mid_data, mid_req, mid_ack, arb_req, arb_ack,
25:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
26:            val_seq, halt_req, haltf_ack, roll_req, rollf_ack, reset);
27:
28:    always wait (reset)
29:    begin
30:            halt_ack = 0;
31:            roll_ack = 0;
32:            wait (~reset);
33:    end
34:
35:    always @(haltq_ack or haltf_ack)
36:            if (haltq_ack & haltf_ack)
37:                    halt_ack = 1;
38:            else if (~haltq_ack & ~haltf_ack)
39:                    halt_ack = 0;
40:
41:    always @(rollq_ack or rollf_ack)
42:            if (rollq_ack & rollf_ack)
43:                    roll_ack = 1;
44:            else if (~rollq_ack & ~rollf_ack)
45:                    roll_ack = 0;
46:
47:    endmodule       // ckr
48:
49:    // ====================================================================
50:    // Checker for R_bus (functional part)
51:
52:    module ckrf (in_seq, in_data, in_req, in_ack, arb_req, arb_ack,
53:            tri_seq, tri_chkid, tri_error, tri_req, out_ack,
54:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
55:
56:    `include "parameter"
57:
58:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
59:    input [DATA_WIDTH:0] in_data;
60:    output [SEQ_WIDTH-1:0] tri_seq;
61:    output [CHKID_WIDTH-1:0] tri_chkid;
62:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
63:    output in_ack, arb_req, tri_error, tri_req, halt_ack, roll_ack;
64:
65:    reg [SEQ_WIDTH-1:0] out_seq;
66:    reg in_ack, arb_req, out_error, out_req, halt_ack, roll_ack;
67:
68:    reg [SEQ_WIDTH-1:0] diff;
69:    reg [DATA_WIDTH:0] data_buf;
70:    reg [5:0] loop;
71:    reg valid, parity;
```

```
 72:
 73:    assign tri_seq = arb_ack ? out_seq : ZZZ;
 74:    assign tri_chkid = arb_ack ? ID_CKR : ZZZ;
 75:    assign tri_error = arb_ack ? out_error : ZZZ;
 76:    assign tri_req = arb_ack ? out_req : ZZZ;
 77:
 78:    always wait (reset)
 79:    begin
 80:            disable rollback_cycle;
 81:            disable latch_cycle;
 82:            disable output_cycle;
 83:            out_seq = XXX;
 84:            in_ack = 0;
 85:            arb_req = 0;
 86:            out_error = XXX;
 87:            out_req = 0;
 88:            halt_ack = 0;
 89:            roll_ack = 0;
 90:            valid = 0;
 91:            wait (~reset);
 92:    end
 93:
 94:    always wait (halt_req & ~reset)
 95:    begin :rollback_cycle
 96:            #1;
 97:            halt_ack = 1;
 98:            wait (roll_req);
 99:            #1;
100:            disable latch_cycle;
101:            disable output_cycle;
102:            in_ack = 0;
103:            arb_req = 0;
104:            out_req = 0;
105:
106:            #DLY_SEQ_COMP;
107:            diff = out_seq - val_seq;       // compare sequence numbers
108:            if (~diff [SEQ_WIDTH-1])
109:                begin
110:                    valid = 0;
111:                    out_seq = XXX;
112:                    out_error = XXX;
113:                end
114:
115:            roll_ack = 1;
116:            fork
117:                begin
118:                    wait (~roll_req);
119:                    #1;
120:                    roll_ack = 0;
121:                end
122:                begin
123:                    wait (~halt_req);
124:                    #1;
125:                    halt_ack = 0;
126:                end
127:            join
128:    end
129:
130:    always wait (in_req & ~valid & ~halt_req & ~reset)
131:    begin :latch_cycle
132:            #1;
133:            wait (~halt_req);
134:            out_seq = in_seq;
135:            data_buf = in_data;
136:            in_ack = 1;
137:            valid = 1;
138:            wait (~in_req);
139:            #1;
140:            wait (~halt_req);
141:            in_ack = 0;
142:    end
```

```
143:
144:    always wait (valid & ~halt_req & ~reset)
145:    begin :output_cycle
146:            #1;
147:            parity = 0;
148:            for (loop=0; loop<1+DATA_WIDTH; loop=loop+1)  // calculate parity
149:                    parity = parity ^ data_buf [loop];
150:            if (parity === 1'bx)  parity = 1;
151:
152:            #DLY_CKR_CHK;
153:            out_error = parity;
154:            arb_req = 1;
155:            wait (arb_ack);
156:            #1;
157:            wait (~halt_req);
158:            out_req = 1;
159:            wait (out_ack);
160:            #1;
161:            valid = 0;
162:            arb_req = 0;
163:            wait (~halt_req);
164:            out_req = 0;
165:            wait (~out_ack & ~arb_ack);
166:    end
167:
168:    endmodule       // ckrf
169:
170:    // ====================================================================
171:    // Queue for R_bus Checker
172:
173:    module ckrq (in_seq, in_data, in_req, in_ack,
174:            out_seq, out_data, out_req, out_ack,
175:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
176:
177:    `include "parameter"
178:
179:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
180:    input [DATA_WIDTH:0] in_data;
181:    output [SEQ_WIDTH-1:0] out_seq;
182:    output [DATA_WIDTH:0] out_data;
183:    input in_req, out_ack, halt_req, roll_req, reset;
184:    output in_ack, out_req, halt_ack, roll_ack;
185:
186:    reg halt_ack, roll_ack;
187:
188:    wire [SEQ_WIDTH-1:0] seq1;
189:    wire [DATA_WIDTH:0] data1;
190:
191:    ckr_buf buf1 (in_seq, in_data, in_req, in_ack, seq1, data1, req1, ack1,
192:                    val_seq, halt_req, halt_a1, roll_req, roll_a1, reset);
193:    ckr_buf buf2 (seq1, data1, req1, ack1, out_seq, out_data, out_req, out_ack,
194:                    val_seq, halt_req, halt_a2, roll_req, roll_a2, reset);
195:
196:    always wait (reset)
197:    begin
198:            halt_ack = 0;
199:            roll_ack = 0;
200:            wait (~reset);
201:    end
202:
203:    always @(halt_a1 or halt_a2)
204:            if (halt_a1 & halt_a2)
205:                    halt_ack = 1;
206:            else if (~halt_a1 & ~halt_a2)
207:                    halt_ack = 0;
208:
209:    always @(roll_a1 or roll_a2)
210:            if (roll_a1 & roll_a2)
211:                    roll_ack = 1;
212:            else if (~roll_a1 & ~roll_a2)
213:                    roll_ack = 0;
```

```
214:
215:    endmodule       // ckrq
216:
217:    // ====================================================================
218:
219:    module ckr_buf (in_s, in_d, in_r, in_a, out_s, out_d, out_r, out_a,
220:                    val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
221:
222:    `include "parameter"
223:
224:    input [SEQ_WIDTH-1:0] in_s, val_seq;
225:    input [DATA_WIDTH:0] in_d;
226:    output [SEQ_WIDTH-1:0] out_s;
227:    output [DATA_WIDTH:0] out_d;
228:    input in_r, out_a, halt_req, roll_req, reset;
229:    output in_a, out_r, halt_ack, roll_ack;
230:
231:    reg [SEQ_WIDTH-1:0] out_s;
232:    reg [DATA_WIDTH:0] out_d;
233:    reg in_a, out_r, halt_ack, roll_ack;
234:
235:    reg [SEQ_WIDTH-1:0] diff;
236:    reg valid;
237:
238:    always wait (reset)
239:    begin
240:            disable rollback_cycle;
241:            disable input_cycle;
242:            disable output_cycle;
243:            out_s = XXX;
244:            out_d = XXX;
245:            in_a = 0;
246:            out_r = 0;
247:            halt_ack = 0;
248:            roll_ack = 0;
249:            valid = 0;
250:            wait (~reset);
251:    end
252:
253:    always wait (halt_req & ~reset)
254:    begin :rollback_cycle
255:            #1;
256:            halt_ack = 1;
257:            wait (roll_req);
258:            #1;
259:            disable input_cycle;
260:            disable output_cycle;
261:            in_a = 0;
262:            out_r = 0;
263:
264:            #DLY_SEQ_COMP;
265:            diff = out_s - val_seq;       // compare sequence numbers
266:            if (~diff [SEQ_WIDTH-1])
267:                begin
268:                    valid = 0;
269:                    out_s = XXX;
270:                    out_d = XXX;
271:                end
272:
273:            roll_ack = 1;
274:            fork
275:                begin
276:                    wait (~roll_req);
277:                    #1;
278:                    roll_ack = 0;
279:                end
280:                begin
281:                    wait (~halt_req);
282:                    #1;
283:                    halt_ack = 0;
284:                end
```

```
285:            join
286:    end
287:
288:    always wait (in_r & ~valid & ~halt_req & ~reset)
289:    begin :input_cycle
290:            #1;
291:            wait (~halt_req);
292:            out_s = in_s;
293:            out_d = in_d;
294:            in_a = 1;
295:            valid = 1;
296:            wait (~in_r);
297:            #1;
298:            wait (~halt_req);
299:            in_a = 0;
300:    end
301:
302:    always wait (valid & ~halt_req & ~reset)
303:    begin :output_cycle
304:            #1;
305:            wait (~halt_req);
306:            out_r = 1;
307:            wait (out_a);
308:            #1;
309:            valid = 0;
310:            wait (~halt_req);
311:            out_r = 0;
312:            wait (~out_a);
313:    end
314:
315:    endmodule       // ckr_buf
```

dmem.v

```
 1:    // Data Memory
 2:
 3:    module dmem (addr, data, rw_mode, retry, in_req, in_ack,
 4:            tri_out_req, out_ack,
 5:            halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [ADDR_WIDTH:0] addr;
10:    inout [DATA_WIDTH:0] data;
11:    input rw_mode, retry, in_req, out_ack, halt_req, roll_req, reset;
12:    output in_ack, tri_out_req, halt_ack, roll_ack;
13:
14:    reg in_ack, tri_out_req, halt_ack, roll_ack;
15:
16:    reg [DATA_WIDTH:0] data_out, dmemory [0:DMEM_SIZE-1];
17:    reg [ADDR_WIDTH:0] loop;        // one extra bit for loop termination
18:    reg [5:0] loop1;                // DATA_WIDTH = 32 bits max
19:    reg parity;
20:
21:    assign data = (rw_mode & in_req) ? data_out : ZZZ;
22:
23:    always wait (reset)
24:    begin
25:            disable rollback_cycle;
26:            disable memory_cycle;
27:            data_out = XXX;
28:            in_ack = 0;
29:            tri_out_req = ZZZ;
30:            halt_ack = 0;
31:            roll_ack = 0;
32:            for (loop = 0; loop < DMEM_SIZE; loop = loop + 1)
33:                    dmemory [loop] = XXX;
34:            $readmemh ("inst.hex", dmemory, 0);
35:            wait (~reset);
36:    end
37:
38:    always wait (halt_req & ~reset)
39:    begin :rollback_cycle
40:            #1;
41:            halt_ack = 1;
42:            wait (roll_req);
43:            #1;
44:            disable memory_cycle;
45:            data_out = XXX;
46:            in_ack = 0;
47:            tri_out_req = ZZZ;
48:
49:            roll_ack = 1;
50:            fork
51:                begin
52:                    wait (~roll_req);
53:                    #1;
54:                    roll_ack = 0;
55:                end
56:                begin
57:                    wait (~halt_req);
58:                    #1;
59:                    halt_ack = 0;
60:                end
61:            join
62:    end
63:
64:    // Input is not acknowledged until output is acknowledged so that
65:    // register and sequence number can be routed from DWB to DQ without
66:    // going through DMEM.
67:
68:    always wait (in_req & ~halt_req & ~reset)
69:    begin :memory_cycle
70:            if (rw_mode)    // 1 for read
71:                begin
```

```
 72:                     #DLY_DMEM_RD;
 73:                     data_out = dmemory [addr[ADDR_WIDTH-1:ADDR_IGNORE]];
 74:
 75:                     if (retry)              // correct parity error
 76:                         begin
 77:                             parity = 0;
 78:                             for (loop1=0; loop1<DATA_WIDTH; loop1=loop1+1)
 79:                                     parity = parity ^ data_out [loop1];
 80:                             data_out [DATA_WIDTH] = parity;
 81:                             dmemory [addr[ADDR_WIDTH-1:ADDR_IGNORE]] = data_out;
 82:                         end
 83:
 84:                     #1;
 85:                     wait (~halt_req);
 86:                     tri_out_req = 1;
 87:                     wait (out_ack);
 88:                     #1;
 89:                     in_ack = 1;
 90:                     wait (~halt_req);
 91:                     tri_out_req = ZZZ;              // tri-state it
 92:                     wait (~in_req);
 93:                     #1;
 94:                     in_ack = 0;
 95:                     wait (~out_ack);
 96:                 end
 97:             else            // 0 for write
 98:                 begin
 99:                     #DLY_DMEM_WR;
100:                     wait (~halt_req);
101:                     dmemory [addr[ADDR_WIDTH-1:ADDR_IGNORE]] = data;
102:                     in_ack = 1;
103:                     wait (~in_req);
104:                     #1;
105:                     wait (~halt_req);
106:                     in_ack = 0;
107:                 end
108:    end
109:
110:    endmodule       // dmem
```

dq.v

```
 1:    // Data Queue (combining DQQ and DQBUS)
 2:
 3:    module dq (in_seq, in_reg, in_data, in_req, in_ack, arb_req, arb_ack,
 4:            tri_seq, tri_reg, tri_data, tri_req, out_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
10:    input [REG_WIDTH-1:0] in_reg;
11:    input [DATA_WIDTH:0] in_data;
12:    output [SEQ_WIDTH-1:0] tri_seq;
13:    output [REG_WIDTH-1:0] tri_reg;
14:    output [DATA_WIDTH:0] tri_data;
15:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
16:    output in_ack, arb_req, tri_req, halt_ack, roll_ack;
17:
18:    reg halt_ack, roll_ack;
19:
20:    wire [SEQ_WIDTH-1:0] mid_seq;
21:    wire [REG_WIDTH-1:0] mid_reg;
22:    wire [DATA_WIDTH:0] mid_data;
23:
24:    dqq dqq (in_seq, in_reg, in_data, in_req, in_ack,
25:            mid_seq, mid_reg, mid_data, mid_req, mid_ack,
26:            val_seq, halt_req, haltq_ack, roll_req, rollq_ack, reset);
27:    dqbus dqbus (mid_seq, mid_reg, mid_data, mid_req, mid_ack, arb_req, arb_ack,
28:            tri_seq, tri_reg, tri_data, tri_req, out_ack,
29:            val_seq, halt_req, haltb_ack, roll_req, rollb_ack, reset);
30:
31:    always wait (reset)
32:    begin
33:            halt_ack = 0;
34:            roll_ack = 0;
35:            wait (~reset);
36:    end
37:
38:    always @(haltq_ack or haltb_ack)
39:            if (haltq_ack & haltb_ack)
40:                    halt_ack = 1;
41:            else if (~haltq_ack & ~haltb_ack)
42:                    halt_ack = 0;
43:
44:    always @(rollq_ack or rollb_ack)
45:            if (rollq_ack & rollb_ack)
46:                    roll_ack = 1;
47:            else if (~rollq_ack & ~rollb_ack)
48:                    roll_ack = 0;
49:
50:    endmodule       // dq
51:
52:    // ====================================================================
53:    // Data Queue Bus Handler
54:
55:    module dqbus (in_seq, in_reg, in_data, in_req, in_ack, arb_req, arb_ack,
56:            tri_seq, tri_reg, tri_data, tri_req, out_ack,
57:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
58:
59:    `include "parameter"
60:
61:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
62:    input [REG_WIDTH-1:0] in_reg;
63:    input [DATA_WIDTH:0] in_data;
64:    output [SEQ_WIDTH-1:0] tri_seq;
65:    output [REG_WIDTH-1:0] tri_reg;
66:    output [DATA_WIDTH:0] tri_data;
67:    input in_req, arb_ack, out_ack, halt_req, roll_req, reset;
68:    output in_ack, arb_req, tri_req, halt_ack, roll_ack;
69:
70:    reg [SEQ_WIDTH-1:0] out_seq;
71:    reg [REG_WIDTH-1:0] out_reg;
```

```
 72:    reg [DATA_WIDTH:0] out_data;
 73:    reg in_ack, arb_req, out_req, halt_ack, roll_ack;
 74:
 75:    reg [SEQ_WIDTH-1:0] diff;
 76:    reg valid;
 77:
 78:    assign tri_seq = arb_ack ? out_seq : ZZZ;
 79:    assign tri_reg = arb_ack ? out_reg : ZZZ;
 80:    assign tri_data = arb_ack ? out_data : ZZZ;
 81:    assign tri_req = arb_ack ? out_req : ZZZ;
 82:
 83:    always wait (reset)
 84:    begin
 85:            disable rollback_cycle;
 86:            disable latch_cycle;
 87:            disable output_cycle;
 88:            out_seq = XXX;
 89:            out_reg = XXX;
 90:            out_data = XXX;
 91:            in_ack = 0;
 92:            arb_req = 0;
 93:            out_req = 0;
 94:            halt_ack = 0;
 95:            roll_ack = 0;
 96:            valid = 0;
 97:            wait (~reset);
 98:    end
 99:
100:    always wait (halt_req & ~reset)
101:    begin :rollback_cycle
102:            #1;
103:            halt_ack = 1;
104:            wait (roll_req);
105:            #1;
106:            disable latch_cycle;
107:            disable output_cycle;
108:            in_ack = 0;
109:            arb_req = 0;
110:            out_req = 0;
111:
112:            #DLY_SEQ_COMP;
113:            diff = out_seq - val_seq;       // compare sequence numbers
114:            if (~diff [SEQ_WIDTH-1])
115:                begin
116:                    valid = 0;
117:                    out_seq = XXX;
118:                    out_reg = XXX;
119:                    out_data = XXX;
120:                end
121:
122:            roll_ack = 1;
123:            fork
124:                begin
125:                    wait (~roll_req);
126:                    #1;
127:                    roll_ack = 0;
128:                end
129:                begin
130:                    wait (~halt_req);
131:                    #1;
132:                    halt_ack = 0;
133:                end
134:            join
135:    end
136:
137:    always wait (in_req & ~valid & ~halt_req & ~reset)
138:    begin :latch_cycle
139:            #1;
140:            wait (~halt_req);
141:            out_seq = in_seq;
142:            out_reg = in_reg;
```

```
143:            out_data = in_data;
144:            in_ack = 1;
145:            valid = 1;
146:            wait (~in_req);
147:            #1;
148:            wait (~halt_req);
149:            in_ack = 0;
150:    end
151:
152:    always wait (valid & ~halt_req & ~reset)
153:    begin :output_cycle
154:            #1;
155:            arb_req = 1;
156:            wait (arb_ack);
157:            #1;
158:            wait (~halt_req);
159:            out_req = 1;
160:            wait (out_ack);
161:            #1;
162:            valid = 0;
163:            arb_req = 0;
164:            wait (~halt_req);
165:            out_req = 0;
166:            wait (~out_ack & ~arb_ack);
167:    end
168:
169:    endmodule       // dqbus
170:
171:    // ========================================================================
172:    // Data Queue (real queue)
173:
174:    module dqq (in_seq, in_reg, in_data, in_req, in_ack,
175:            out_seq, out_reg, out_data, out_req, out_ack,
176:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
177:
178:    `include "parameter"
179:
180:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
181:    input [REG_WIDTH-1:0] in_reg;
182:    input [DATA_WIDTH:0] in_data;
183:    output [SEQ_WIDTH-1:0] out_seq;
184:    output [REG_WIDTH-1:0] out_reg;
185:    output [DATA_WIDTH:0] out_data;
186:    input in_req, out_ack, halt_req, roll_req, reset;
187:    output in_ack, out_req, halt_ack, roll_ack;
188:
189:    reg halt_ack, roll_ack;
190:
191:    wire [SEQ_WIDTH-1:0] seq1;
192:    wire [REG_WIDTH-1:0] reg1;
193:    wire [DATA_WIDTH:0] data1;
194:
195:    dq_buffer buf1 (in_seq, in_reg, in_data, in_req, in_ack,
196:            seq1, reg1, data1, req1, ack1,
197:            val_seq, halt_req, halt_a1, roll_req, roll_a1, reset);
198:    dq_buffer buf2 (seq1, reg1, data1, req1, ack1,
199:            out_seq, out_reg, out_data, out_req, out_ack,
200:            val_seq, halt_req, halt_a2, roll_req, roll_a2, reset);
201:
202:    always wait (reset)
203:    begin
204:            halt_ack = 0;
205:            roll_ack = 0;
206:            wait (~reset);
207:    end
208:
209:    always @(halt_a1 or halt_a2)
210:            if (halt_a1 & halt_a2)
211:                    halt_ack = 1;
212:            else if (~halt_a1 & ~halt_a2)
213:                    halt_ack = 0;
```

```
214:
215:    always @(roll_a1 or roll_a2)
216:            if (roll_a1 & roll_a2)
217:                    roll_ack = 1;
218:            else if (~roll_a1 & ~roll_a2)
219:                    roll_ack = 0;
220:
221:    endmodule       // ckiq
222:
223:    // ===========================================================================
224:
225:    module dq_buffer (in_s, in_g, in_d, in_r, in_a,
226:            out_s, out_g, out_d, out_r, out_a,
227:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
228:
229:    `include "parameter"
230:
231:    input [SEQ_WIDTH-1:0] in_s, val_seq;
232:    input [REG_WIDTH-1:0] in_g;
233:    input [DATA_WIDTH:0] in_d;
234:    output [SEQ_WIDTH-1:0] out_s;
235:    output [REG_WIDTH-1:0] out_g;
236:    output [DATA_WIDTH:0] out_d;
237:    input in_r, out_a, halt_req, roll_req, reset;
238:    output in_a, out_r, halt_ack, roll_ack;
239:
240:    `ifdef BEHAV_DQ
241:
242:    // ===========================================================================
243:    // behavioral level
244:
245:    reg [SEQ_WIDTH-1:0] out_s;
246:    reg [REG_WIDTH-1:0] out_g;
247:    reg [DATA_WIDTH:0] out_d;
248:    reg in_a, out_r, halt_ack, roll_ack;
249:
250:    reg [SEQ_WIDTH-1:0] diff;
251:    reg valid;
252:
253:    always wait (reset)
254:    begin
255:            disable rollback_cycle;
256:            disable input_cycle;
257:            disable output_cycle;
258:            out_s = XXX;
259:            out_g = XXX;
260:            out_d = XXX;
261:            in_a = 0;
262:            out_r = 0;
263:            halt_ack = 0;
264:            roll_ack = 0;
265:            valid = 0;
266:            wait (~reset);
267:    end
268:
269:    always wait (halt_req & ~reset)
270:    begin :rollback_cycle
271:            #1;
272:            halt_ack = 1;
273:            wait (roll_req);
274:            #1;
275:            disable input_cycle;
276:            disable output_cycle;
277:            in_a = 0;
278:            out_r = 0;
279:
280:            #DLY_SEQ_COMP;
281:            diff = out_s - val_seq;       // compare sequence numbers
282:            if (~diff [SEQ_WIDTH-1])
283:                begin
284:                    valid = 0;
```

```
285:                    out_s = XXX;
286:                    out_g = XXX;
287:                    out_d = XXX;
288:                end
289:
290:            roll_ack = 1;
291:            fork
292:                begin
293:                    wait (~roll_req);
294:                    #1;
295:                    roll_ack = 0;
296:                end
297:                begin
298:                    wait (~halt_req);
299:                    #1;
300:                    halt_ack = 0;
301:                end
302:            join
303:    end
304:
305:    always wait (in_r & ~valid & ~halt_req & ~reset)
306:    begin :input_cycle
307:            #1;
308:            wait (~halt_req);
309:            out_s = in_s;
310:            out_g = in_g;
311:            out_d = in_d;
312:            in_a = 1;
313:            valid = 1;
314:            wait (~in_r);
315:            #1;
316:            wait (~halt_req);
317:            in_a = 0;
318:    end
319:
320:    always wait (valid & ~halt_req & ~reset)
321:    begin :output_cycle
322:            #1;
323:            wait (~halt_req);
324:            out_r = 1;
325:            wait (out_a);
326:            #1;
327:            valid = 0;
328:            wait (~halt_req);
329:            out_r = 0;
330:            wait (~out_a);
331:    end
332:
333:    endmodule       // dq_buffer_b (behavioral level)
334:
335:    `else
336:
337:    // ====================================================================
338:    // gate level
339:
340:    muller2c   x1 (in_r, ~cout2, clear1, cout1);
341:    dlr        x2 (cout1, ~halt_req, clear1, latch);
342:    and     #1 g1 (a, complete, ~roll_req);
343:    muller2c   x3 (a, ~out_a, clear2, cout2);
344:    dlr        x4 (cout2, ~halt_req, clear1, out_r);
345:    regcmpl    x5 (latch, reset, regclk, complete);
346:    wire in_a = complete;
347:
348:    or      #1 g2 (clear1, roll_req, reset);
349:    and     #1 g3 (b, inval, roll_req);
350:    or      #1 g4 (clear2, b, reset);
351:
352:    buf     #18 g5  (dhalt, halt_req);              // 4 (x2) + 14 (x5)
353:    and     #1  g6  (halt_ack, dhalt, halt_req);
354:    and     #1  g7  (c, roll_req, ~cout1, ~in_a, ~out_r);
355:    and     #1  g8  (d, clear2, ~cout2);
```

```
356:    and     #1  g9  (e, ~inval, cmp_done);
357:    or      #1  g10 (f, d, e);
358:    muller3c    x6  (c, f, cmp_done, reset, roll_ack);
359:    seqcmp      x7  (out_s, val_seq, roll_req, inval, cmp_done);
360:
361:    `ifdef WAVE_DQ
362:    initial
363:    #0 $gr_addwaves ("in_r", in_r, "cout1", cout1, "latch", latch, "in_a", in_a,
364:            "a", a, "cout2", cout2, "out_r", out_r, "out_a", out_a,
365:            "h_req", halt_req, "h_ack", halt_ack, "r_req", roll_req,
366:            "clear1", clear1, "clear2", clear2, "r_ack", roll_ack);
367:    `endif
368:
369:    dlr     b40 (in_s[0], regclk, reset, out_s[0]),
370:            b41 (in_s[1], regclk, reset, out_s[1]),
371:            b42 (in_s[2], regclk, reset, out_s[2]),
372:            b43 (in_s[3], regclk, reset, out_s[3]);
373:
374:    dl      b50 (in_g[0], regclk, out_g[0]),
375:            b51 (in_g[1], regclk, out_g[1]),
376:            b52 (in_g[2], regclk, out_g[2]),
377:            b53 (in_g[3], regclk, out_g[3]),
378:            b54 (in_g[4], regclk, out_g[4]);
379:
380:    dl      b0  (in_d[0], regclk, out_d[0]),
381:            b1  (in_d[1], regclk, out_d[1]),
382:            b2  (in_d[2], regclk, out_d[2]),
383:            b3  (in_d[3], regclk, out_d[3]),
384:            b4  (in_d[4], regclk, out_d[4]),
385:            b5  (in_d[5], regclk, out_d[5]),
386:            b6  (in_d[6], regclk, out_d[6]),
387:            b7  (in_d[7], regclk, out_d[7]),
388:            b8  (in_d[8], regclk, out_d[8]),
389:            b9  (in_d[9], regclk, out_d[9]),
390:            b10 (in_d[10], regclk, out_d[10]),
391:            b11 (in_d[11], regclk, out_d[11]),
392:            b12 (in_d[12], regclk, out_d[12]),
393:            b13 (in_d[13], regclk, out_d[13]),
394:            b14 (in_d[14], regclk, out_d[14]),
395:            b15 (in_d[15], regclk, out_d[15]),
396:            b16 (in_d[16], regclk, out_d[16]),
397:            b17 (in_d[17], regclk, out_d[17]),
398:            b18 (in_d[18], regclk, out_d[18]),
399:            b19 (in_d[19], regclk, out_d[19]),
400:            b20 (in_d[20], regclk, out_d[20]),
401:            b21 (in_d[21], regclk, out_d[21]),
402:            b22 (in_d[22], regclk, out_d[22]),
403:            b23 (in_d[23], regclk, out_d[23]),
404:            b24 (in_d[24], regclk, out_d[24]),
405:            b25 (in_d[25], regclk, out_d[25]),
406:            b26 (in_d[26], regclk, out_d[26]),
407:            b27 (in_d[27], regclk, out_d[27]),
408:            b28 (in_d[28], regclk, out_d[28]),
409:            b29 (in_d[29], regclk, out_d[29]),
410:            b30 (in_d[30], regclk, out_d[30]),
411:            b31 (in_d[31], regclk, out_d[31]),
412:            b32 (in_d[32], regclk, out_d[32]);      // parity bit
413:
414:    endmodule       // dq_buffer_g (gate level)
415:
416:    // ========================================================================
417:
418:    module seqcmp (seq, error, req, inv, ack);
419:    input [3:0] seq, error;
420:    input req;
421:    output inv, ack;
422:
423:    brwend  x1 (seq[0], error[0], req, a, na);
424:    brw     x2 (seq[1], error[1], a, na, req, b, nb);
425:    brw     x3 (seq[2], error[2], b, nb, req, c, nc);
426:    sub     x4 (seq[3], error[3], c, nc, req, d, inv, ack);
```

```
427:
428:    `ifdef WAVE_SEQCMP
429:    initial
430:    #0 $gr_addwaves ("seq", seq, "error", error, "req", req, "ack", ack,
431:            "_a", a, "na", na, "b", b, "nb", nb, "c", c, "nc", nc, "d", d,
432:            "inv", inv);
433:    `endif
434:
435:    endmodule       // seqcmp
436:
437:    `endif
```

gates.v

```
 1:    // Gate-Level Components
 2:
 3:    module muller2 (in1, in2, out);         // 2-input C-element
 4:    input in1, in2;
 5:    output out;
 6:    supply1 vcc;
 7:    supply0 gnd;
 8:
 9:    nmos    #2 m1 (out, gnd, b);
10:    nmos    #1 m2 (b, a, out);
11:    nmos    #1 m3 (a, gnd, in1);
12:    nmos    #1 m4 (a, gnd, in2);
13:    nmos    #1 m6 (b, c, in1);
14:    nmos    #1 m7 (c, gnd, in2);
15:
16:    pullup  (b);
17:    pullup  (out);
18:
19:    endmodule       // muller2
20:
21:    // ========================================================================
22:
23:    module muller3 (in1, in2, in3, out);    // 3-input C-element
24:    input in1, in2, in3;
25:    output out;
26:    supply1 vcc;
27:    supply0 gnd;
28:
29:    nmos    #2 m1 (out, gnd, b);
30:    nmos    #1 m2 (b, a, out);
31:    nmos    #1 m3 (a, gnd, in1);
32:    nmos    #1 m4 (a, gnd, in2);
33:    nmos    #1 m5 (a, gnd, in3);
34:    nmos    #1 m6 (b, c, in1);
35:    nmos    #1 m7 (c, d, in2);
36:    nmos    #1 m8 (d, gnd, in3);
37:
38:    pullup  (b);
39:    pullup  (out);
40:
41:    endmodule       // muller3
42:
43:    // ========================================================================
44:
45:    module muller4 (in1, in2, in3, in4, out);       // 4-input C-element
46:    input in1, in2, in3, in4;
47:    output out;
48:
49:    muller2 x1 (in1, in2, out1);
50:    muller2 x2 (in3, in4, out2);
51:    muller2 x3 (out1, out2, out);
52:
53:    endmodule       // muller4
54:
55:    // ========================================================================
56:
57:    module muller2c (in1, in2, clear, out);         // 2-input with clear
58:    input in1, in2, clear;
59:    output out;
60:
61:    and     #1 g1 (out1, in1, ~clear);
62:    and     #1 g2 (out2, in2, ~clear);
63:    muller2    m1 (out1, out2, out);
64:
65:    endmodule       // muller2c
66:
67:    // ========================================================================
68:
69:    module muller3c (in1, in2, in3, clear, out);    // 3-input with clear
70:    input in1, in2, in3, clear;
71:    output out;
```

```
 72:
 73:    and     #1 g1 (out1, in1, ~clear);
 74:    and     #1 g2 (out2, in2, ~clear);
 75:    and     #1 g3 (out3, in3, ~clear);
 76:    muller3    m1 (out1, out2, out3, out);
 77:
 78:    endmodule       // muller3c
 79:
 80:    // ===========================================================================
 81:
 82:    module dl (D, G, Q);            // D-Latch
 83:    input D, G;
 84:    output Q;
 85:    trireg a;
 86:
 87:    nmos    #1 m1 (a, D, G);
 88:    pmos    #1 m2 (a, Q, G);
 89:    not     #1 g1 (b, a);
 90:    not     #1 g2 (Q, b);
 91:
 92:    endmodule       // dl
 93:
 94:    // ===========================================================================
 95:
 96:    module dls (D, G, S, Q);        // D-Latch with Set
 97:    input D, G, S;
 98:    output Q;
 99:    supply1 vcc;
100:    trireg a;
101:
102:    nmos    #1 m1 (a, D, e);
103:    pmos    #1 m2 (a, Q, f);
104:    nmos    #1 m3 (a, vcc, S);
105:    not     #1 g1 (b, a);
106:    not     #1 g2 (Q, b);
107:    and     #1 g3 (e, G, ~S);
108:    or      #1 g4 (f, G, S);
109:
110:    endmodule       // dls
111:
112:    // ===========================================================================
113:
114:    module dlr (D, G, R, Q);        // D-Latch with Reset
115:    input D, G, R;
116:    output Q;
117:    supply0 gnd;
118:    trireg a;
119:
120:    nmos    #1 m1 (a, D, e);
121:    pmos    #1 m2 (a, Q, f);
122:    nmos    #1 m3 (a, gnd, R);
123:    not     #1 g1 (b, a);
124:    not     #1 g2 (Q, b);
125:    and     #1 g3 (e, G, ~R);
126:    or      #1 g4 (f, G, R);
127:
128:    endmodule       // dlr
129:
130:    // ===========================================================================
131:
132:    module dffr (D, CLK, R, Q);     // D-Flip Flop with Reset
133:    input D, CLK, R;
134:    output Q;
135:
136:    dlr     master (D, CLK, R, a);
137:    dlr     slave (a, ~CLK, R, Q);
138:
139:    endmodule       // dffr
140:
141:    // ===========================================================================
142:    // done->complete delay: (1 for g4)+(1 for dl disable)+(1 for safety)
```

```
143:    // reset requires 5 time units, critical path in muller3c
144:
145:    module regcmpl (latch, reset, regclk, complete);
146:    input latch, reset;
147:    output regclk, complete;
148:
149:    dffr       x1 (nosc, latch, reset, osc);
150:    dls        x2 (osc, latch, reset, q1);
151:    dlr        x3 (nosc, latch, reset, q2);
152:    muller3c   x4 (same1, same2, latch, reset, done);
153:    not     #1 g1 (nosc, osc);
154:    xnor    #1 g2 (same1, osc, q1);
155:    xnor    #1 g3 (same2, nosc, q2);
156:    and     #1 g4 (regclk, ~done, latch);
157:    and     #3 g5 (complete, done, ~reset);
158:
159:    `ifdef WAVE_REGCMPL
160:    initial
161:    #0 $gr_addwaves ("latch", latch, "osc", osc, "q1", q1, "q2", q2,
162:            "same1", same1, "same2", same2, "done", done, "regclk", regclk,
163:            "complet", complete);
164:    `endif
165:
166:    endmodule       // regcmpl
167:
168:    // ========================================================================
169:    // A minus B, X=borrow in, Z=borrow out, R=request
170:
171:    module brw (A, B, X, NX, R, Z, NZ);     // full borrow circuit
172:    input A, B, X, NX, R;
173:    output Z, NZ;
174:    supply1 vcc;
175:    supply0 gnd;
176:    trireg Y, NY;
177:
178:    not     #1 g1  (NA, A);
179:    not     #1 g2  (NB, B);
180:    buf     #1 g3  (DR, R);
181:    not     #1 g4  (Z, NY);
182:    not     #1 g5  (NZ, Y);
183:    pmos    #1 m1  (NY, vcc, DR);
184:    pmos    #1 m2  (Y, vcc, DR);
185:    nmos    #1 m3  (c, gnd, DR);
186:
187:    nmos    #1 m4  (NY, d, X);
188:    nmos    #1 m5  (d, c, NA);
189:    nmos    #1 m6  (d, c, B);
190:    nmos    #1 m7  (NY, e, NA);
191:    nmos    #1 m8  (e, c, B);
192:    nmos    #1 m9  (Y, f, A);
193:    nmos    #1 m10 (f, g, NB);
194:    nmos    #1 m11 (g, c, A);
195:    nmos    #1 m12 (Y, g, NX);
196:    nmos    #1 m13 (g, c, NB);
197:
198:    endmodule       // brw
199:
200:    // ========================================================================
201:    // A minus B, Z=borrow out, R=request
202:
203:    module brwend (A, B, R, Z, NZ);         // no borrow in
204:    input A, B, R;
205:    output Z, NZ;
206:    supply1 vcc;
207:    supply0 gnd;
208:    trireg Y, NY;
209:
210:    not     #1 g1 (NA, A);
211:    not     #1 g2 (NB, B);
212:    buf     #1 g3 (DR, R);
213:    not     #1 g4  (Z, NY);
```

```
214:    not     #1 g5  (NZ, Y);
215:    pmos    #1 m1 (NY, vcc, DR);
216:    pmos    #1 m2 (Y, vcc, DR);
217:    nmos    #1 m3 (c, gnd, DR);
218:
219:    nmos    #1 m4 (NY, d, NA);
220:    nmos    #1 m5 (d, c, B);
221:    nmos    #1 m6 (Y, c, A);
222:    nmos    #1 m7 (Y, c, NB);
223:
224:    endmodule       // brwend
225:
226:    // ========================================================================
227:    // use A/NA for slower signal
228:
229:    module stxor (A, NA, B, NB, R, Z, NZ);  // self-timed XOR
230:    input A, NA, B, NB, R;
231:    output Z, NZ;
232:    supply1 vcc;
233:    supply0 gnd;
234:    trireg Y, NY;
235:
236:    not     #1 g1  (Z, NY);
237:    not     #1 g2  (NZ, Y);
238:    pmos    #1 m1  (NY, vcc, R);
239:    pmos    #1 m2  (Y, vcc, R);
240:    nmos    #1 m3  (c, gnd, R);
241:
242:    nmos    #1 m4  (NY, d, NA);
243:    nmos    #1 m5  (d, c, B);
244:    nmos    #1 m6  (NY, e, A);
245:    nmos    #1 m7  (e, c, NB);
246:    nmos    #1 m8  (Y, f, A);
247:    nmos    #1 m9  (f, c, B);
248:    nmos    #1 m10 (Y, f, NB);
249:    nmos    #1 m11 (f, c, NA);
250:
251:    endmodule       // stxor
252:
253:    // ========================================================================
254:    // A minus B, X=borrow in, D=difference, R=request
255:
256:    module sub (A, B, X, NX, REQ, D, ND, ACK);      // subtraction bitslice
257:    input A, B, X, NX, REQ;
258:    output D, ND, ACK;
259:    supply1 vcc;
260:    supply0 gnd;
261:    trireg D, ND;
262:
263:    not     #1 g1 (NA, A);
264:    not     #1 g2 (NB, B);
265:    buf     #1 g3 (DR, REQ);
266:    or      #1 g4 (ACK, D, ND);
267:
268:    stxor   x1 (A, NA, B, NB, DR, c, nc);
269:    stxor   x2 (c, nc, X, NX, DR, D, ND);
270:
271:    endmodule       // sub
```

iiu.v

```
 1:    // Instruction Issuing Unit
 2:
 3:    module iiu (inst_bus, inst_req, inst_ack, inst_en,
 4:            chkbits, log_req, log_ack, cki_req, cki_ack,
 5:            newpc_req, pc_ack, imem_ack, iq_ack,
 6:            seq_num, reg_rd, reg_rs, reg_rt, res_req, res_ack,
 7:            sim_f, a_bus, b_bus, rfile_req, rfile_ack, ckf_req, ckf_ack,
 8:            alu_func, alu_req, alu_ack,
 9:            mem_rw, mem_req, mem_ack,
10:            val_seq, halt_req, halt_ack, roll_req, roll_ack,
11:            retry, stop, reset);
12:
13:    ‘include "parameter"
14:
15:    inout [DATA_WIDTH:0] inst_bus, a_bus, b_bus;
16:    output [CHECKERS-1:0] chkbits;
17:    output [SEQ_WIDTH-1:0] seq_num;
18:    output [REG_WIDTH-1:0] reg_rd, reg_rs, reg_rt;
19:    output [FUNC_WIDTH-1:0] alu_func;
20:    input [SEQ_WIDTH-1:0] val_seq;
21:    input halt_req, roll_req;
22:    output inst_en, sim_f, mem_rw, retry, stop;
23:    input inst_req, log_ack, cki_ack, pc_ack, imem_ack, iq_ack, res_ack,
24:            rfile_ack, ckf_ack, alu_ack, mem_ack, reset;
25:    output inst_ack, log_req, cki_req, newpc_req, res_req, rfile_req, ckf_req,
26:            alu_req, mem_req, halt_ack, roll_ack;
27:
28:    reg [CHECKERS-1:0] chkbits;
29:    reg [SEQ_WIDTH-1:0] seq_num;
30:    reg [REG_WIDTH-1:0] reg_rd, reg_rs, reg_rt;
31:    reg [FUNC_WIDTH-1:0] alu_func;
32:    reg inst_en, sim_f, mem_rw, retry, stop;
33:    reg inst_ack, log_req, cki_req, newpc_req, res_req, rfile_req, ckf_req,
34:            alu_req, mem_req, halt_ack, roll_ack;
35:
36:    reg [DATA_WIDTH:0] inst_buf, abus_in, abus_out, bbus_in, bbus_out;
37:    reg [ADDR_WIDTH:0] int_pc;
38:    reg [5:0] loop;                         // data/addr 32 bits max
39:    reg go_decode, parity;
40:
41:    wire [OP_WIDTH-1:0] opcode = inst_buf [DATA_WIDTH-1:OFFSET_WIDTH];
42:    wire [REG_WIDTH-1:0] rs = inst_buf [OFFSET_WIDTH-1:OFFSET_WIDTH-REG_WIDTH];
43:    wire [REG_WIDTH-1:0] rt = inst_buf [OFFSET_WIDTH-REG_WIDTH-1:IMM_WIDTH];
44:    wire [REG_WIDTH-1:0] rd = inst_buf [IMM_WIDTH-1:IMM_WIDTH-REG_WIDTH];
45:    wire [OFFSET_WIDTH-1:0] offset = inst_buf [OFFSET_WIDTH-1:0];
46:    wire [IMM_WIDTH-1:0] imm = inst_buf [IMM_WIDTH-1:0];
47:    wire [EXTRA_WIDTH-1:0] extra = inst_buf [EXTRA_WIDTH-1:0];
48:
49:
50:    assign inst_bus = inst_en ? ZZZ : int_pc;
51:    assign a_bus = (rfile_req | halt_ack) ? ZZZ : abus_out;
52:    assign b_bus = rfile_req ? ZZZ : bbus_out;
53:
54:
55:    always wait (reset)
56:    begin
57:            disable rollback_cycle;
58:            disable fetch_cycle;
59:            disable decode_cycle;
60:            chkbits = XXX;
61:            seq_num = 0;            // initial sequence number
62:            reg_rs = XXX;
63:            reg_rt = XXX;
64:            reg_rd = XXX;
65:            alu_func = XXX;
66:            inst_en = 1;            // enable instructions
67:            sim_f = 0;
68:            mem_rw = XXX;
69:            retry = 0;
70:            stop = 0;
71:            inst_ack = 0;           // handshaking signals
```

```
 72:             log_req = 0;
 73:             cki_req = 0;
 74:             newpc_req = 0;
 75:             res_req = 0;
 76:             rfile_req = 0;
 77:             ckf_req = 0;
 78:             alu_req = 0;
 79:             mem_req = 0;
 80:             halt_ack = 0;
 81:             roll_ack = 0;
 82:
 83:             int_pc = 0;             // initial internal PC
 84:             abus_out = XXX;
 85:             bbus_out = XXX;
 86:             go_decode = 0;
 87:             wait (~reset);
 88:     end
 89:
 90:
 91:     always wait (halt_req & ~reset)
 92:     begin :rollback_cycle
 93:             #1;
 94:             wait (~newpc_req & ~pc_ack & ~imem_ack & ~iq_ack);
 95:             halt_ack = 1;
 96:             wait (roll_req);
 97:             #1;
 98:             disable fetch_cycle;
 99:             disable decode_cycle;
100:             chkbits = XXX;
101:             reg_rs = XXX;
102:             reg_rt = XXX;
103:             reg_rd = XXX;
104:             alu_func = XXX;
105:             inst_en = 0;
106:             sim_f = 0;
107:             mem_rw = XXX;
108:             inst_ack = 0;
109:             log_req = 0;
110:             cki_req = 0;
111:             newpc_req = 0;
112:             res_req = 0;
113:             rfile_req = 0;
114:             ckf_req = 0;
115:             alu_req = 0;
116:             mem_req = 0;
117:
118:             abus_out = XXX;
119:             bbus_out = XXX;
120:             go_decode = 0;
121:             retry = 1;                      // rollback flag
122:
123:             roll_ack = 1;
124:             wait (~roll_req);
125:             #1;
126:             seq_num = val_seq;              // sequence number with error
127:             int_pc = a_bus;                 // address with error
128:             fork
129:                 begin
130:                     roll_ack = 0;
131:                     wait (~halt_req);
132:                     #1;
133:                     halt_ack = 0;
134:                 end
135:                 pc_handler;
136:             join
137:     end
138:
139:
140:     always wait (inst_req & ~go_decode & inst_en & ~halt_req & ~reset)
141:     begin :fetch_cycle
142:             #1;
```

```
143:            inst_buf = inst_bus;
144:            bbus_out = inst_bus;    // for CKI
145:            abus_out = int_pc;      // for LOG
146:            #DLY_IIU_PCINC;
147:            int_pc = int_pc + ADDR_INC;
148:            parity = 0;
149:            for (loop=0; loop<ADDR_WIDTH; loop=loop+1)
150:                    parity = parity ^ int_pc [loop];
151:            int_pc [ADDR_WIDTH] = parity;
152:            inst_ack = 1;
153:            go_decode = 1;
154:            wait (~inst_req);
155:            #1;
156:            inst_ack = 0;
157:    end
158:
159:
160:    always wait (go_decode & ~halt_req & ~reset)
161:    begin :decode_cycle
162:            #DLY_IIU_DECOD;
163:            case (opcode)                   // determine check bits
164:                    OP_SPECIAL:
165:                            if (extra == SOP_NOP)
166:                                    chkbits = 4'b 0001;
167:                            else
168:                                    chkbits = 4'b 1011;
169:                    OP_J, OP_TRAP:
170:                            chkbits = 4'b 0001;
171:                    OP_BEQZ, OP_BNEZ, OP_JR, OP_JRF:
172:                            chkbits = 4'b 0011;
173:                    OP_SW, OP_SWF:
174:                            chkbits = 4'b 0111;
175:                    OP_LW:
176:                            chkbits = 4'b 1111;
177:                    OP_JAL:
178:                            chkbits = 4'b 1001;
179:                    default:                        // JALR, ALU immediate
180:                            chkbits = 4'b 1011;
181:            endcase
182:            #1;
183:
184:            wait (~halt_req);
185:            log_req = 1;                    // log instruction
186:            wait (log_ack);
187:            #1;
188:            fork
189:                begin
190:                    wait (~halt_req);
191:                    log_req = 0;
192:                    wait (~log_ack);
193:                end
194:                begin
195:                    wait (~halt_req);
196:                    cki_req = 1;            // check instruction
197:                    wait (cki_ack);
198:                    #1;
199:                    wait (~halt_req);
200:                    cki_req = 0;
201:                    wait (~cki_ack);
202:                end
203:            join
204:
205:            case (opcode)
206:                    OP_SPECIAL: special_handler;
207:                    OP_BEQZ, OP_BNEZ: branch_handler;
208:                    OP_J, OP_JAL, OP_JR, OP_JALR, OP_JRF: jump_handler;
209:                    OP_LW, OP_SW, OP_SWF: memory_handler;
210:                    OP_TRAP: trap_handler;
211:
212:                    OP_ADDUI:  begin  alu_func=FUNC_ADDU;   alu_control;  end
213:                    OP_SUBUI:  begin  alu_func=FUNC_SUBU;   alu_control;  end
```

```
214:                    OP_ANDI:   begin  alu_func=FUNC_AND;    alu_control;  end
215:                    OP_ORI:    begin  alu_func=FUNC_OR;     alu_control;  end
216:                    OP_XORI:   begin  alu_func=FUNC_XOR;    alu_control;  end
217:                    OP_LHI:    begin  alu_func=FUNC_LHI;    alu_control;  end
218:                    OP_SLLI:   begin  alu_func=FUNC_SLL;    alu_control;  end
219:                    OP_SRLI:   begin  alu_func=FUNC_SRL;    alu_control;  end
220:                    OP_SRAI:   begin  alu_func=FUNC_SRA;    alu_control;  end
221:                    OP_SEQI:   begin  alu_func=FUNC_SEQ;    alu_control;  end
222:                    OP_SNEI:   begin  alu_func=FUNC_SNE;    alu_control;  end
223:                    OP_SLTI:   begin  alu_func=FUNC_SLT;    alu_control;  end
224:                    OP_SGTI:   begin  alu_func=FUNC_SGT;    alu_control;  end
225:                    OP_SLEI:   begin  alu_func=FUNC_SLE;    alu_control;  end
226:                    OP_SGEI:   begin  alu_func=FUNC_SGE;    alu_control;  end
227:                    OP_ADDUIF: begin  alu_func=FUNC_ADDUF;  alu_control;  end
228:
229:                    default:
230:                        begin
231:                            $display ("Invalid Opcode %h", inst_buf);
232:                            stop = 1;
233:                            wait (reset);
234:                        end
235:            endcase
236:            retry = 0;                      // reset rollback flag
237:            go_decode = 0;
238:            seq_num = seq_num + 1;
239:    end
240:
241:
242:    task special_handler;
243:    begin
244:            case (extra)
245:                    SOP_NOP:  ;
246:                    SOP_ADDU:  begin  alu_func=FUNC_ADDU;   alu_control;  end
247:                    SOP_SUBU:  begin  alu_func=FUNC_SUBU;   alu_control;  end
248:                    SOP_AND:   begin  alu_func=FUNC_AND;    alu_control;  end
249:                    SOP_OR:    begin  alu_func=FUNC_OR;     alu_control;  end
250:                    SOP_XOR:   begin  alu_func=FUNC_XOR;    alu_control;  end
251:                    SOP_SLL:   begin  alu_func=FUNC_SLL;    alu_control;  end
252:                    SOP_SRL:   begin  alu_func=FUNC_SRL;    alu_control;  end
253:                    SOP_SRA:   begin  alu_func=FUNC_SRA;    alu_control;  end
254:                    SOP_SEQ:   begin  alu_func=FUNC_SEQ;    alu_control;  end
255:                    SOP_SNE:   begin  alu_func=FUNC_SNE;    alu_control;  end
256:                    SOP_SLT:   begin  alu_func=FUNC_SLT;    alu_control;  end
257:                    SOP_SGT:   begin  alu_func=FUNC_SGT;    alu_control;  end
258:                    SOP_SLE:   begin  alu_func=FUNC_SLE;    alu_control;  end
259:                    SOP_SGE:   begin  alu_func=FUNC_SGE;    alu_control;  end
260:                    SOP_ADDUF: begin  alu_func=FUNC_ADDUF;  alu_control;  end
261:
262:                    default:
263:                        begin
264:                            $display ("Invalid Special Opcode %h", inst_buf);
265:                            stop = 1;
266:                            wait (reset);
267:                        end
268:            endcase
269:    end
270:    endtask
271:
272:
273:    task alu_control;
274:    begin
275:            if ((alu_func == FUNC_ADDUF) & retry)   // no error if rollback
276:                    alu_func = FUNC_ADDU;
277:            reg_rs = rs;
278:            reg_rt = (opcode == OP_SPECIAL) ? rt : 0;
279:            reg_rd = (opcode == OP_SPECIAL) ? rd : rt;
280:            reserve_handler;
281:            regfile_handler;
282:            abus_out = abus_in;
283:            bbus_out = (opcode == OP_SPECIAL) ? bbus_in :
284:                    {{(DATA_WIDTH-IMM_WIDTH){imm[IMM_WIDTH-1]}}, imm};
```

```
285:            alu_handler;
286:    end
287:    endtask
288:
289:
290:    task memory_handler;
291:    begin
292:            reg_rs = rs;
293:            reg_rt = (opcode == OP_LW) ? 0 : rt;
294:            reg_rd = (opcode == OP_LW) ? rt : 0;
295:            reserve_handler;
296:            regfile_handler;
297:            mem_rw = (opcode == OP_LW);
298:            #DLY_IIU_ADD;
299:            abus_out = abus_in +
300:                    {{(DATA_WIDTH-IMM_WIDTH){imm[IMM_WIDTH-1]}}, imm};
301:            parity = 0;
302:            for (loop=0; loop<ADDR_WIDTH; loop=loop+1)
303:                    parity = parity ^ abus_out [loop];
304:            abus_out [ADDR_WIDTH] = parity;
305:            bbus_out = bbus_in;
306:            if ((opcode == OP_SWF) & ~retry)        // simulate fault
307:                    bbus_out [DATA_WIDTH] = ~bbus_out [DATA_WIDTH];
308:            #1;
309:            wait (~halt_req);
310:            mem_req = 1;
311:            wait (mem_ack);
312:            #1;
313:            wait (~halt_req);
314:            mem_req = 0;
315:            wait (~mem_ack);
316:    end
317:    endtask
318:
319:
320:    task branch_handler;
321:    begin
322:            reg_rs = rs;
323:            reg_rt = 0;
324:            reg_rd = 0;
325:            reserve_handler;
326:            regfile_handler;
327:            if (((opcode==OP_BEQZ) && (abus_in==0)) ||
328:                ((opcode==OP_BNEZ) && (abus_in!=0)))
329:                begin
330:                    #DLY_IIU_ADD;
331:                    int_pc = int_pc[ADDR_WIDTH-1:0] + {{(DATA_WIDTH-IMM_WIDTH)
332:                            {imm[IMM_WIDTH-1]}}, imm};      // sign ext.
333:                    pc_handler;
334:                end
335:    end
336:    endtask
337:
338:
339:    task jump_handler;
340:    begin
341:            bbus_out = int_pc[ADDR_WIDTH-1:0];      // for JAL and JALR
342:
343:            if ((opcode==OP_J) || (opcode==OP_JAL))
344:                begin
345:                    #DLY_IIU_ADD;
346:                    int_pc = int_pc[ADDR_WIDTH-1:0] + {{(DATA_WIDTH-OFFSET_WIDTH)
347:                            {offset[OFFSET_WIDTH-1]}}, offset};
348:                    pc_handler;
349:                end
350:            else
351:                begin
352:                    sim_f = (opcode == OP_JRF) & ~retry;    // simulate fault
353:                    reg_rs = rs;
354:                    reg_rt = 0;
355:                    reg_rd = 0;
```

```
356:                    reserve_handler;
357:                    regfile_handler;
358:                    int_pc = abus_in;
359:                    pc_handler;
360:                end
361:
362:            if ((opcode==OP_JAL) || (opcode==OP_JALR))
363:                begin
364:                    alu_func = FUNC_PASS;
365:                    reg_rs = 0;
366:                    reg_rt = 0;
367:                    reg_rd = REG_SIZE - 1;  // save to highest register
368:                    reserve_handler;
369:                    alu_handler;
370:                end
371:    end
372:    endtask
373:
374:
375:    task trap_handler;
376:    begin
377:        case (offset)
378:            TRAP_BLANK: $display();
379:
380:            TRAP_STOP:
381:                    begin
382:                            $display ("Trap Stop %h", inst_buf);
383:                            stop = 1;
384:                            wait (reset);
385:                    end
386:
387:            default:
388:                    if (offset < REG_SIZE)
389:                        begin
390:                            reg_rs = offset;
391:                            reg_rt = 0;
392:                            reg_rd = 0;
393:                            reserve_handler;
394:                            regfile_handler;
395:                            $display ("Reg %d:  Dec=%d  Hex=%h  Bin=%b",
396:                                    reg_rs, abus_in [DATA_WIDTH-1:0],
397:                                    abus_in , abus_in);
398:                        end
399:                    else
400:                        begin
401:                            $display ("Invalid Trap Service %h", inst_buf);
402:                            stop = 1;
403:                            wait (reset);
404:                        end
405:        endcase
406:    end
407:    endtask
408:
409:
410:    task reserve_handler;
411:    begin
412:            #1;
413:            wait (~halt_req);
414:            res_req = 1;
415:            wait (res_ack);
416:            #1;
417:            wait (~halt_req);
418:            res_req = 0;
419:            wait (~res_ack);
420:            #1;
421:    end
422:    endtask
423:
424:
425:    task regfile_handler;
426:    begin
```

```
427:            rfile_req = 1;
428:            wait (rfile_ack);
429:            #1;
430:            wait (~halt_req);
431:            abus_in = a_bus;
432:            bbus_in = b_bus;
433:            ckf_req = 1;            // notify checker
434:            wait (ckf_ack);
435:            #1;
436:            rfile_req = 0;
437:            wait (~halt_req);
438:            ckf_req = 0;
439:            wait (~ckf_ack & ~rfile_ack);
440:    end
441:    endtask
442:
443:
444:    task alu_handler;
445:    begin
446:            parity = 0;
447:            for (loop=0; loop<DATA_WIDTH; loop=loop+1)
448:                    parity = parity ^ abus_out [loop];
449:            abus_out [DATA_WIDTH] = parity;
450:            parity = 0;
451:            for (loop=0; loop<DATA_WIDTH; loop=loop+1)
452:                    parity = parity ^ bbus_out [loop];
453:            bbus_out [DATA_WIDTH] = parity;
454:            #1;
455:            wait (~halt_req);
456:            alu_req = 1;
457:            wait (alu_ack);
458:            #1;
459:            wait (~halt_req);
460:            alu_req = 0;
461:            wait (~alu_ack);
462:    end
463:    endtask
464:
465:
466:    task pc_handler;                // not interrupted by rollback
467:    begin
468:            parity = 0;
469:            for (loop=0; loop<ADDR_WIDTH; loop=loop+1)      // calculate parity
470:                    parity = parity ^ int_pc [loop];
471:            int_pc [ADDR_WIDTH] = parity;
472:
473:            wait (~inst_ack);               // finish inst fetch cycle
474:            inst_en = 0;
475:            #1;
476:            newpc_req = 1;
477:            wait (pc_ack & imem_ack & iq_ack);
478:            #1;
479:            inst_en = 1;
480:            newpc_req = 0;
481:            wait (~pc_ack & ~imem_ack & ~iq_ack);
482:    end
483:    endtask
484:
485:    endmodule       // iiu
```

imem.v

```
 1:    // Instruction Memory
 2:
 3:    module imem (addr, retry, in_req, in_ack, data, out_req, out_ack,
 4:                    cancel_req, cancel_ack, reset);
 5:
 6:    `include "parameter"
 7:
 8:    input [ADDR_WIDTH:0] addr;
 9:    output [DATA_WIDTH:0] data;
10:    input retry, in_req, out_ack, cancel_req, reset;
11:    output in_ack, out_req, cancel_ack;
12:
13:    reg [DATA_WIDTH:0] data, imemory [0:IMEM_SIZE-1];
14:    reg in_ack, out_req, cancel_ack;
15:
16:    reg [5:0] loop;                // DATA_WIDTH = 32 bits max
17:    reg parity;
18:
19:    initial
20:            $readmemh ("inst.hex", imemory, 0);
21:
22:    always wait (reset)
23:    begin
24:            disable memory_cycle;
25:            disable cancel_cycle;
26:            data = XXX;
27:            in_ack = 0;
28:            out_req = 0;
29:            cancel_ack = 0;
30:            wait (~reset);
31:    end
32:
33:    always wait (cancel_req & ~reset)
34:    begin :cancel_cycle
35:            #1;
36:            disable memory_cycle;
37:            data = XXX;
38:            in_ack = 0;
39:            out_req = 0;
40:            cancel_ack = 1;
41:            wait (~cancel_req);
42:            #1;
43:            cancel_ack = 0;
44:    end
45:
46:    always wait (in_req & ~cancel_req & ~reset)
47:    begin :memory_cycle
48:            #DLY_IMEM_RD;
49:            data = imemory [addr[ADDR_WIDTH-1:ADDR_IGNORE]];
50:
51:            if (retry)                      // correct parity error
52:                begin
53:                    parity = 0;
54:                    for (loop=0; loop<DATA_WIDTH; loop=loop+1)
55:                            parity = parity ^ data [loop];
56:                    data [DATA_WIDTH] = parity;
57:                    imemory [addr[ADDR_WIDTH-1:ADDR_IGNORE]] = data;
58:                end
59:
60:            fork
61:                begin
62:                    in_ack = 1;
63:                    wait (~in_req);
64:                    #1;
65:                    in_ack = 0;
66:                end
67:                begin
68:                    #1;
69:                    out_req = 1;
70:                    wait (out_ack);
71:                    #1;
```

```
72:                     out_req = 0;
73:                     wait (~out_ack);
74:                 end
75:             join
76:     end
77:
78:     endmodule       // imem
```

iq.v

```
 1:    // Instruction Queue
 2:
 3:    module iq (in_data, in_req, in_ack, tri_data, out_req, out_ack, en_out,
 4:            cancel_req, cancel_ack, reset);
 5:
 6:    `include "parameter"
 7:
 8:    input [DATA_WIDTH:0] in_data;
 9:    output [DATA_WIDTH:0] tri_data;
10:    input in_req, out_ack, en_out, cancel_req, reset;
11:    output in_ack, out_req, cancel_ack;
12:
13:    `ifdef BEHAV_IQ
14:
15:    // ====================================================================
16:    // behavioral level
17:
18:    wire [DATA_WIDTH:0] data1, out_data;
19:
20:    iq_buffer_b buf1 (in_data, in_req, in_ack, data1, req1, ack1, cancel_req,
21:                    cancel_a1, reset);
22:    iq_buffer_b buf2 (data1, req1, ack1, out_data, out_req, out_ack, cancel_req,
23:                    cancel_a2, reset);
24:
25:    assign tri_data = en_out ? out_data : ZZZ;
26:
27:    reg cancel_ack;
28:
29:    always wait (reset)
30:    begin
31:            cancel_ack = 0;
32:            wait (~reset);
33:    end
34:
35:    always @(cancel_a1 or cancel_a2)
36:            if (cancel_a1 & cancel_a2)
37:                    cancel_ack = 1;
38:            else if (~cancel_a1 & ~cancel_a2)
39:                    cancel_ack = 0;
40:
41:    endmodule       // iq (behavioral level)
42:
43:    // ====================================================================
44:    // behavioral level
45:
46:    module iq_buffer_b (in_d, in_r, in_a, out_d, out_r, out_a, cancel_r,
47:                    cancel_a, reset);
48:
49:    `include "parameter"
50:
51:    input [DATA_WIDTH:0] in_d;
52:    output [DATA_WIDTH:0] out_d;
53:    input in_r, out_a, cancel_r, reset;
54:    output in_a, out_r, cancel_a;
55:
56:    reg [DATA_WIDTH:0] out_d;
57:    reg valid, in_a, out_r, cancel_a;
58:
59:    always wait (reset)
60:    begin
61:            disable input_cycle;
62:            disable output_cycle;
63:            disable cancel_cycle;
64:            out_d = XXX;
65:            valid = 0;
66:            in_a = 0;
67:            out_r = 0;
68:            cancel_a = 0;
69:            wait (~reset);
70:    end
71:
```

```
 72:    always wait (cancel_r & ~reset)
 73:    begin :cancel_cycle
 74:            #1;
 75:            disable input_cycle;
 76:            disable output_cycle;
 77:            out_d = XXX;
 78:            valid = 0;
 79:            in_a = 0;
 80:            out_r = 0;
 81:            cancel_a = 1;
 82:            wait (~cancel_r);
 83:            #1;
 84:            cancel_a = 0;
 85:    end
 86:
 87:    always wait (in_r & ~valid & ~cancel_r & ~reset)
 88:    begin :input_cycle
 89:            #1;
 90:            out_d = in_d;
 91:            in_a = 1;
 92:            valid = 1;
 93:            wait (~in_r);
 94:            #1;
 95:            in_a = 0;
 96:    end
 97:
 98:    always wait (valid & ~cancel_r & ~reset)
 99:    begin :output_cycle
100:            #1;
101:            out_r = 1;
102:            wait (out_a);
103:            #1;
104:            valid = 0;
105:            out_r = 0;
106:            wait (~out_a);
107:    end
108:
109:    endmodule       // iq_buffer_b (behavioral level)
110:
111:    `else
112:
113:    // ============================================================================
114:    // gate level
115:
116:    wire [DATA_WIDTH:0] data1, out_data;
117:
118:    iq_buffer_g buf1 (in_data, in_req, in_ack, data1, req1, ack1, cancel_req,
119:                      cancel_a1, reset);
120:    iq_buffer_g buf2 (data1, req1, ack1, out_data, out_req, out_ack, cancel_req,
121:                      cancel_a2, reset);
122:
123:    muller2 m1 (cancel_a1, cancel_a2, cancel_ack);
124:
125:    bufif1 #0
126:            (tri_data[0], out_data[0], en_out),
127:            (tri_data[1], out_data[1], en_out),
128:            (tri_data[2], out_data[2], en_out),
129:            (tri_data[3], out_data[3], en_out),
130:            (tri_data[4], out_data[4], en_out),
131:            (tri_data[5], out_data[5], en_out),
132:            (tri_data[6], out_data[6], en_out),
133:            (tri_data[7], out_data[7], en_out),
134:            (tri_data[8], out_data[8], en_out),
135:            (tri_data[9], out_data[9], en_out),
136:            (tri_data[10], out_data[10], en_out),
137:            (tri_data[11], out_data[11], en_out),
138:            (tri_data[12], out_data[12], en_out),
139:            (tri_data[13], out_data[13], en_out),
140:            (tri_data[14], out_data[14], en_out),
141:            (tri_data[15], out_data[15], en_out),
142:            (tri_data[16], out_data[16], en_out),
```

```
143:            (tri_data[17], out_data[17], en_out),
144:            (tri_data[18], out_data[18], en_out),
145:            (tri_data[19], out_data[19], en_out),
146:            (tri_data[20], out_data[20], en_out),
147:            (tri_data[21], out_data[21], en_out),
148:            (tri_data[22], out_data[22], en_out),
149:            (tri_data[23], out_data[23], en_out),
150:            (tri_data[24], out_data[24], en_out),
151:            (tri_data[25], out_data[25], en_out),
152:            (tri_data[26], out_data[26], en_out),
153:            (tri_data[27], out_data[27], en_out),
154:            (tri_data[28], out_data[28], en_out),
155:            (tri_data[29], out_data[29], en_out),
156:            (tri_data[30], out_data[30], en_out),
157:            (tri_data[31], out_data[31], en_out),
158:            (tri_data[32], out_data[32], en_out);   // parity bit
159:
160:    endmodule       // iq (gate level)
161:
162:    // ========================================================================
163:    // gate level
164:
165:    module iq_buffer_g (in_d, in_r, in_a, out_d, out_r, out_a, cancel_r,
166:                        cancel_a, reset);
167:
168:    `include "parameter"
169:
170:    input [DATA_WIDTH:0] in_d;
171:    output [DATA_WIDTH:0] out_d;
172:    input in_r, out_a, cancel_r, reset;
173:    output in_a, out_r, cancel_a;
174:
175:    muller2c   x1 (in_r, ~out_r, clear, latch);
176:    muller2c   x2 (complete, ~out_a, clear, out_r);
177:    regcmpl    x3 (latch, clear, regclk, complete);
178:    or      #1 g1 (clear, cancel_r, reset);
179:    buf     #7 g2 (cancel_a, cancel_r);
180:    wire in_a = complete;
181:
182:    `ifdef WAVE_IQ
183:    initial
184:    #0 $gr_addwaves ("in_d", in_d, "out_d", out_d, "in_r", in_r, "in_a", in_a,
185:            "out_r", out_r, "out_a", out_a, "can_r", cancel_r,
186:            "can_a", cancel_a);
187:    `endif
188:
189:    dl      b0  (in_d[0], regclk, out_d[0]),
190:            b1  (in_d[1], regclk, out_d[1]),
191:            b2  (in_d[2], regclk, out_d[2]),
192:            b3  (in_d[3], regclk, out_d[3]),
193:            b4  (in_d[4], regclk, out_d[4]),
194:            b5  (in_d[5], regclk, out_d[5]),
195:            b6  (in_d[6], regclk, out_d[6]),
196:            b7  (in_d[7], regclk, out_d[7]),
197:            b8  (in_d[8], regclk, out_d[8]),
198:            b9  (in_d[9], regclk, out_d[9]),
199:            b10 (in_d[10], regclk, out_d[10]),
200:            b11 (in_d[11], regclk, out_d[11]),
201:            b12 (in_d[12], regclk, out_d[12]),
202:            b13 (in_d[13], regclk, out_d[13]),
203:            b14 (in_d[14], regclk, out_d[14]),
204:            b15 (in_d[15], regclk, out_d[15]),
205:            b16 (in_d[16], regclk, out_d[16]),
206:            b17 (in_d[17], regclk, out_d[17]),
207:            b18 (in_d[18], regclk, out_d[18]),
208:            b19 (in_d[19], regclk, out_d[19]),
209:            b20 (in_d[20], regclk, out_d[20]),
210:            b21 (in_d[21], regclk, out_d[21]),
211:            b22 (in_d[22], regclk, out_d[22]),
212:            b23 (in_d[23], regclk, out_d[23]),
213:            b24 (in_d[24], regclk, out_d[24]),
```

```
214:            b25 (in_d[25], regclk, out_d[25]),
215:            b26 (in_d[26], regclk, out_d[26]),
216:            b27 (in_d[27], regclk, out_d[27]),
217:            b28 (in_d[28], regclk, out_d[28]),
218:            b29 (in_d[29], regclk, out_d[29]),
219:            b30 (in_d[30], regclk, out_d[30]),
220:            b31 (in_d[31], regclk, out_d[31]),
221:            b32 (in_d[32], regclk, out_d[32]);      // parity bit
222:
223:    endmodule       // iq_buffer_g (gate level)
224:
225:    `endif
```

log.v

```
  1:    // Instruction Log
  2:
  3:    module log (log_seq, log_chkbits, a_bus, log_req, log_ack,
  4:            chk_seq, chk_chkid, chk_error, chk_req, chk_ack,
  5:            val_seq, valmem_req, valmem_ack, valreg_req, valreg_ack,
  6:            halt_req, halt_ack, roll_req, roll_ack, reset);
  7:
  8:    ‘include "parameter"
  9:
 10:    input [SEQ_WIDTH-1:0] log_seq, chk_seq;
 11:    input [CHECKERS-1:0] log_chkbits;
 12:    inout [DATA_WIDTH:0] a_bus;
 13:    input [CHKID_WIDTH-1:0] chk_chkid;
 14:    output [SEQ_WIDTH-1:0] val_seq;
 15:    input chk_error, reset;
 16:    input log_req, chk_req, valmem_ack, valreg_ack, halt_ack, roll_ack;
 17:    output log_ack, chk_ack, valmem_req, valreg_req, halt_req, roll_req;
 18:
 19:    reg [SEQ_WIDTH-1:0] val_seq;
 20:    reg chk_ack, valmem_req, valreg_req, halt_req, roll_req;
 21:
 22:    reg [SEQ_WIDTH-1:0] clr_seq;
 23:    reg [CHKID_WIDTH-1:0] clr_id;
 24:    reg [DWBS-1:0] buf_dwb;
 25:    reg del_ack, clr_req, arbdel_req, arbdel_ack, arbchk_req, arbchk_ack;
 26:
 27:    wire [DWBS-1:0] log_dwb = log_chkbits [CHECKERS-1:CHECKERS-DWBS];
 28:    wire [SEQ_WIDTH-1:0] del_seq;
 29:    wire [CHECKERS-1:0] del_chkbits;
 30:    wire [DWBS-1:0] del_dwb;
 31:    wire del_req, clr_ack;
 32:
 33:    logq logq (log_seq, log_chkbits, log_dwb, a_bus, log_req, log_ack,
 34:            del_seq, del_chkbits, del_dwb, del_req, del_ack,
 35:            clr_seq, clr_id, clr_req, clr_ack,
 36:            halt_req, haltq_ack, roll_req, rollq_ack, reset);
 37:
 38:    always wait (reset)
 39:    begin
 40:            disable delete_cycle;
 41:            disable checker_cycle;
 42:            disable arb_cycle;
 43:            val_seq = XXX;
 44:            chk_ack = 0;
 45:            valmem_req = 0;
 46:            valreg_req = 0;
 47:            halt_req = 0;
 48:            roll_req = 0;
 49:            del_ack = 0;
 50:            clr_req = 0;
 51:            arbdel_req = 0;
 52:            arbdel_ack = 0;
 53:            arbchk_req = 0;
 54:            arbchk_ack = 0;
 55:            wait (~reset);
 56:    end
 57:
 58:    always wait (chk_req & ~reset)
 59:    begin :checker_cycle
 60:            #1;
 61:            clr_seq = chk_seq;
 62:            clr_id = chk_chkid;
 63:            if (chk_error)
 64:                begin                       // rollback process
 65:                    arbchk_req = 1;
 66:                    wait (arbchk_ack);
 67:                    #1;
 68:                    val_seq = chk_seq;      // sequence number with error
 69:                    halt_req = 1;
 70:                    wait (halt_ack & haltq_ack);
 71:                    #1;
```

```
 72:                     roll_req = 1;
 73:                     disable delete_cycle;   // take care of local business
 74:                     arbdel_req = 0;
 75:                     wait (roll_ack & rollq_ack);
 76:                     #1;
 77:                     roll_req = 0;
 78:                     wait (~roll_ack & ~rollq_ack);
 79:                     #1;
 80:                     halt_req = 0;
 81:                     arbchk_req = 0;
 82:                     wait (~halt_ack & ~haltq_ack & ~arbchk_ack);
 83:                 end
 84:             else
 85:                 fork
 86:                     begin                           // clear the check bit
 87:                             #1;
 88:                             clr_req = 1;
 89:                             wait (clr_ack);
 90:                             #1;
 91:                             clr_req = 0;
 92:                             wait (~clr_ack);
 93:                     end
 94:                     begin
 95:                             chk_ack = 1;            // finish K-bus transaction
 96:                             wait (~chk_req);
 97:                             #1;
 98:                             chk_ack = 0;
 99:                     end
100:                 join
101:     end
102:
103:     always wait (del_req & ~halt_req & ~reset)
104:     begin :delete_cycle
105:             wait (del_chkbits == 0);        // until all bits are cleared
106:             #1;
107:             arbdel_req = 1;
108:             wait (arbdel_ack);
109:             #1;
110:             val_seq = del_seq;
111:             buf_dwb = del_dwb;
112:             del_ack = 1;                    // delete from log
113:             fork
114:                 begin
115:                     wait (~del_req);
116:                     #1;
117:                     del_ack = 0;
118:                 end
119:                 case (buf_dwb)              // validate appropriate DWB
120:                     2'b01:
121:                         begin
122:                             #1;
123:                             valmem_req = 1;
124:                             wait (valmem_ack);
125:                             #1;
126:                             valmem_req = 0;
127:                             arbdel_req = 0;
128:                             wait (~valmem_ack & ~arbdel_ack);
129:                         end
130:                     2'b10, 2'b11:
131:                         begin
132:                             #1;
133:                             valreg_req = 1;
134:                             wait (valreg_ack);
135:                             #1;
136:                             valreg_req = 0;
137:                             arbdel_req = 0;
138:                             wait (~valreg_ack & ~arbdel_ack);
139:                         end
140:                     default:
141:                         begin
142:                             arbdel_req = 0;
```

```
143:                            wait (~arbdel_ack);
144:                        end
145:                endcase
146:            join
147:    end
148:
149:    always wait ((arbchk_req | arbdel_req) & ~reset)
150:    begin :arb_cycle
151:            #1;
152:            if (arbchk_req)         // invalidate has priority over validate
153:                begin
154:                    arbchk_ack = 1;
155:                    wait (~arbchk_req);
156:                    #1;
157:                    arbchk_ack = 0;
158:                end
159:            else
160:                begin
161:                    arbdel_ack = 1;
162:                    wait (~arbdel_req);
163:                    #1;
164:                    arbdel_ack = 0;
165:                end
166:    end
167:
168:    endmodule       // log
169:
170:    // ================================================================
171:    // Log Queue: Note that address is not used in the output.
172:
173:    module logq (in_seq, in_chkbits, in_dwb, a_bus, in_req, in_ack,
174:            out_seq, out_chkbits, out_dwb, out_req, out_ack,
175:            clr_seq, clr_id, clr_req, clr_ack,
176:            halt_req, halt_ack, roll_req, roll_ack, reset);
177:
178:    `include "parameter"
179:
180:    input [SEQ_WIDTH-1:0] in_seq, clr_seq;
181:    input [CHECKERS-1:0] in_chkbits;
182:    input [DWBS-1:0] in_dwb;
183:    inout [DATA_WIDTH:0] a_bus;
184:    output [SEQ_WIDTH-1:0] out_seq;
185:    output [CHECKERS-1:0] out_chkbits;
186:    output [DWBS-1:0] out_dwb;
187:    input [CHKID_WIDTH-1:0] clr_id;
188:    input in_req, out_ack, clr_req, halt_req, roll_req, reset;
189:    output in_ack, out_req, clr_ack, halt_ack, roll_ack;
190:
191:    reg clr_ack, halt_ack, roll_ack;
192:
193:    wire [SEQ_WIDTH-1:0] seq1, seq2, seq3, seq4, seq5, seq6, seq7;
194:    wire [CHECKERS-1:0] chkbits1, chkbits2, chkbits3, chkbits4, chkbits5,
195:            chkbits6, chkbits7;
196:    wire [DWBS-1:0] dwb1, dwb2, dwb3, dwb4, dwb5, dwb6, dwb7;
197:    wire [ADDR_WIDTH:0] addr1, addr2, addr3, addr4, addr5, addr6, addr7,
198:            not_used, tri_addr;
199:    wire [ADDR_WIDTH:0] adpt_addr = a_bus;          // "addr" adaptor
200:
201:    assign a_bus = {ZZZ, tri_addr};                 // compensate for narrow bus
202:
203:
204:    log_buf buf1 (in_seq, in_chkbits, in_dwb, adpt_addr, in_req, in_ack,
205:            seq1, chkbits1, dwb1, addr1, req1, ack1,
206:            clr_seq, clr_id, clr_req, clr_a1,
207:            halt_req, halt_a1, roll_req, roll_a1, tri_addr, reset);
208:
209:    log_buf buf2 (seq1, chkbits1, dwb1, addr1, req1, ack1,
210:            seq2, chkbits2, dwb2, addr2, req2, ack2,
211:            clr_seq, clr_id, clr_req, clr_a2,
212:            halt_req, halt_a2, roll_req, roll_a2, tri_addr, reset);
213:
```

```
214:    log_buf buf3 (seq2, chkbits2, dwb2, addr2, req2, ack2,
215:            seq3, chkbits3, dwb3, addr3, req3, ack3,
216:            clr_seq, clr_id, clr_req, clr_a3,
217:            halt_req, halt_a3, roll_req, roll_a3, tri_addr, reset);
218:
219:    log_buf buf4 (seq3, chkbits3, dwb3, addr3, req3, ack3,
220:            seq4, chkbits4, dwb4, addr4, req4, ack4,
221:            clr_seq, clr_id, clr_req, clr_a4,
222:            halt_req, halt_a4, roll_req, roll_a4, tri_addr, reset);
223:
224:    log_buf buf5 (seq4, chkbits4, dwb4, addr4, req4, ack4,
225:            seq5, chkbits5, dwb5, addr5, req5, ack5,
226:            clr_seq, clr_id, clr_req, clr_a5,
227:            halt_req, halt_a5, roll_req, roll_a5, tri_addr, reset);
228:
229:    log_buf buf6 (seq5, chkbits5, dwb5, addr5, req5, ack5,
230:            seq6, chkbits6, dwb6, addr6, req6, ack6,
231:            clr_seq, clr_id, clr_req, clr_a6,
232:            halt_req, halt_a6, roll_req, roll_a6, tri_addr, reset);
233:
234:    log_buf buf7 (seq6, chkbits6, dwb6, addr6, req6, ack6,
235:            seq7, chkbits7, dwb7, addr7, req7, ack7,
236:            clr_seq, clr_id, clr_req, clr_a7,
237:            halt_req, halt_a7, roll_req, roll_a7, tri_addr, reset);
238:
239:    log_buf buf8 (seq7, chkbits7, dwb7, addr7, req7, ack7,
240:            out_seq, out_chkbits, out_dwb, not_used, out_req, out_ack,
241:            clr_seq, clr_id, clr_req, clr_a8,
242:            halt_req, halt_a8, roll_req, roll_a8, tri_addr, reset);
243:
244:
245:    always wait (reset)
246:    begin
247:            clr_ack = 0;
248:            halt_ack = 0;
249:            roll_ack = 0;
250:            wait (~reset);
251:    end
252:
253:    always @(clr_a1 or clr_a2 or clr_a3 or clr_a4 or clr_a5 or clr_a6 or
254:             clr_a7 or clr_a8)
255:            if (clr_a1 & clr_a2 & clr_a3 & clr_a4 & clr_a5 & clr_a6 & clr_a7 &
256:                clr_a8)
257:                            clr_ack = 1;
258:            else if (~clr_a1 & ~clr_a2 & ~clr_a3 & ~clr_a4 & ~clr_a5 & ~clr_a6 &
259:                     ~clr_a7 & ~clr_a8)
260:                            clr_ack = 0;
261:
262:    always @(halt_a1 or halt_a2 or halt_a3 or halt_a4 or halt_a5 or halt_a6 or
263:             halt_a7 or halt_a8)
264:            if (halt_a1 & halt_a2 & halt_a3 & halt_a4 & halt_a5 & halt_a6 &
265:                halt_a7 & halt_a8)
266:                            halt_ack = 1;
267:            else if (~halt_a1 & ~halt_a2 & ~halt_a3 & ~halt_a4 & ~halt_a5 &
268:                     ~halt_a6 & ~halt_a7 & ~halt_a8)
269:                            halt_ack = 0;
270:
271:    always @(roll_a1 or roll_a2 or roll_a3 or roll_a4 or roll_a5 or roll_a6 or
272:             roll_a7 or roll_a8)
273:            if (roll_a1 & roll_a2 & roll_a3 & roll_a4 & roll_a5 & roll_a6 &
274:                roll_a7 & roll_a8)
275:                            roll_ack = 1;
276:            else if (~roll_a1 & ~roll_a2 & ~roll_a3 & ~roll_a4 & ~roll_a5 &
277:                     ~roll_a6 & ~roll_a7 & ~roll_a8)
278:                            roll_ack = 0;
279:
280:    endmodule       // logq
281:
282:    // ====================================================================
283:
284:    module log_buf (in_s, in_k, in_w, in_d, in_r, in_a,
```

```
285:            out_s, out_k, out_w, out_d, out_r, out_a,
286:            clr_s, clr_i, clr_r, clr_a,
287:            halt_r, halt_a, roll_r, roll_a, tri_addr, reset);
288:
289:    `include "parameter"
290:
291:    input [SEQ_WIDTH-1:0] in_s, clr_s;
292:    input [CHECKERS-1:0] in_k;
293:    input [DWBS-1:0] in_w;
294:    input [ADDR_WIDTH:0] in_d;
295:    output [SEQ_WIDTH-1:0] out_s;
296:    output [CHECKERS-1:0] out_k;
297:    output [DWBS-1:0] out_w;
298:    output [ADDR_WIDTH:0] out_d, tri_addr;
299:    input [CHKID_WIDTH-1:0] clr_i;
300:    input in_r, out_a, clr_r, halt_r, roll_r, reset;
301:    output in_a, out_r, clr_a, halt_a, roll_a;
302:
303:    reg [SEQ_WIDTH-1:0] out_s;
304:    reg [CHECKERS-1:0] out_k;
305:    reg [DWBS-1:0] out_w;
306:    reg [ADDR_WIDTH:0] out_d, tri_addr;
307:    reg in_a, out_r, clr_a, halt_a, roll_a;
308:
309:    reg [SEQ_WIDTH-1:0] diff;
310:    reg valid;
311:
312:    always wait (reset)
313:    begin
314:            disable clear_cycle;
315:            disable rollback_cycle;
316:            disable input_cycle;
317:            disable output_cycle;
318:            out_s = XXX;
319:            out_k = XXX;
320:            out_w = XXX;
321:            out_d = XXX;
322:            tri_addr = ZZZ;
323:            in_a = 0;
324:            out_r = 0;
325:            clr_a = 0;
326:            halt_a = 0;
327:            roll_a = 0;
328:            valid = 0;
329:            wait (~reset);
330:    end
331:
332:    always wait (clr_r & ~reset)            // clear check bit
333:    begin :clear_cycle
334:            #1;
335:            clr_a = 1;
336:            wait (~clr_r);
337:            #1;
338:            if (out_s == clr_s)
339:                    out_k [clr_i] = 0;
340:            #1;
341:            clr_a = 0;
342:    end
343:
344:    always wait (halt_r & ~reset)
345:    begin :rollback_cycle
346:            #1;
347:            halt_a = 1;
348:            wait (roll_r);
349:            #1;
350:            disable input_cycle;
351:            disable output_cycle;
352:            in_a = 0;
353:            out_r = 0;
354:            if (valid & (out_s == clr_s))
355:                    tri_addr = out_d;       // address to be rolled back
```

```
356:
357:            #DLY_SEQ_COMP;
358:            diff = out_s - clr_s;           // compare sequence numbers
359:            if (~diff [SEQ_WIDTH-1])
360:                begin
361:                    valid = 0;              // invalidate entry
362:                    out_s = XXX;
363:                    out_k = XXX;
364:                    out_w = XXX;
365:                    out_d = XXX;
366:                end
367:
368:            roll_a = 1;
369:            fork
370:                begin
371:                    wait (~roll_r);
372:                    #1;
373:                    roll_a = 0;
374:                end
375:                begin
376:                    wait (~halt_r);
377:                    #1;
378:                    tri_addr = ZZZ;         // put back tri-state
379:                    halt_a = 0;
380:                end
381:            join
382:    end
383:
384:    always wait (in_r & ~valid & ~clr_r & ~halt_r & ~reset)
385:    begin :input_cycle
386:            #1;
387:            wait (~clr_r & ~clr_a & ~halt_r);
388:            out_s = in_s;
389:            out_k = in_k;
390:            out_w = in_w;
391:            out_d = in_d;
392:            in_a = 1;
393:            valid = 1;
394:            wait (~in_r);
395:            #1;
396:            wait (~clr_r & ~clr_a & ~halt_r);
397:            in_a = 0;
398:    end
399:
400:    always wait (valid & ~clr_r & ~halt_r & ~reset)
401:    begin :output_cycle
402:            #1;
403:            wait (~clr_r & ~clr_a & ~halt_r);
404:            out_r = 1;
405:            wait (out_a);
406:            #1;
407:            valid = 0;
408:            wait (~clr_r & ~clr_a & ~halt_r);
409:            out_r = 0;
410:            wait (~out_a);
411:    end
412:
413:    endmodule       // log_buf
```

memdwb.v

```
 1:    // Memory Delayed Write Buffer: Note initial wait bit = 1
 2:
 3:    module memdwb (in_seq, in_reg, in_addr, in_data, in_rw_mode, in_retry,
 4:            in_req, in_ack,
 5:            chk_req, chk_ack, val_seq, val_req, val_ack,
 6:            out_addr, tri_data, rw_mode, out_retry, out_req, out_ack,
 7:            out_seq, out_reg, tri_dq_req, dq_ack,
 8:            halt_req, halt_ack, roll_req, roll_ack, reset);
 9:
10:    `include "parameter"
11:
12:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
13:    input [REG_WIDTH-1:0] in_reg;
14:    input [DATA_WIDTH:0] in_addr, in_data;
15:    output [ADDR_WIDTH:0] out_addr;
16:    output [DATA_WIDTH:0] tri_data;
17:    output [SEQ_WIDTH-1:0] out_seq;
18:    output [REG_WIDTH-1:0] out_reg;
19:    input in_rw_mode, in_retry, reset;
20:    output rw_mode, out_retry;
21:    input in_req, chk_ack, val_req, out_ack, dq_ack, halt_req, roll_req;
22:    output in_ack, chk_req, val_ack, out_req, tri_dq_req, halt_ack, roll_ack;
23:
24:    reg [ADDR_WIDTH:0] out_addr;
25:    reg [SEQ_WIDTH-1:0] out_seq;
26:    reg [REG_WIDTH-1:0] out_reg;
27:    reg rw_mode, out_retry;
28:    reg in_ack, chk_req, out_req, tri_dq_req, halt_ack, roll_ack;
29:
30:    reg [ADDR_WIDTH:0] rd_addr;
31:    reg [DATA_WIDTH:0] out_data;
32:    reg [SEQ_WIDTH-1:0] diff;
33:    reg go_read, arbrd_req, arbrd_ack, arbwr_req, arbwr_ack, wrq_req,
34:            com_ack, rdq_req, haltc_ack, rollc_ack;
35:
36:    wire [ADDR_WIDTH:0] adpt_addr = in_addr;        // "addr" adaptor
37:    wire [ADDR_WIDTH:0] com_addr;
38:    wire [DATA_WIDTH:0] com_data, tri_qout;
39:
40:    memdwbq memdwbq (in_seq, 1'b1, adpt_addr, in_data, wrq_req, wrq_ack,
41:            com_wait, com_addr, com_data, com_req, com_ack,
42:            val_seq, val_req, val_ack,
43:            rd_addr, rdq_req, rdq_ack, match, tri_qout,
44:            halt_req, haltq_ack, roll_req, rollq_ack, reset);
45:
46:    assign tri_data = rw_mode ? ZZZ : out_data;
47:
48:    always wait (reset)
49:    begin
50:            disable rollback_cycle;
51:            disable input_cycle;
52:            disable read_cycle;
53:            disable commit_cycle;
54:            disable arb_cycle;
55:            out_addr = XXX;
56:            out_seq = XXX;
57:            out_reg = XXX;
58:            rw_mode = XXX;
59:            out_retry = 0;
60:            in_ack = 0;
61:            chk_req = 0;
62:            out_req = 0;
63:            tri_dq_req = ZZZ;
64:            halt_ack = 0;
65:            roll_ack = 0;
66:            out_data = XXX;
67:            go_read = 0;
68:            arbrd_req = 0;
69:            arbrd_ack = 0;
70:            arbwr_req = 0;
71:            arbwr_ack = 0;
```

```
 72:             wrq_req = 0;
 73:             com_ack = 0;
 74:             rdq_req = 0;
 75:             haltc_ack = 0;
 76:             rollc_ack = 0;
 77:             wait (~reset);
 78:     end
 79:
 80:     always @(haltq_ack or haltc_ack)
 81:             if (haltq_ack & haltc_ack)
 82:                     halt_ack = 1;
 83:             else if (~haltq_ack & ~haltc_ack)
 84:                     halt_ack = 0;
 85:
 86:     always @(rollq_ack or rollc_ack)
 87:             if (rollq_ack & rollc_ack)
 88:                     roll_ack = 1;
 89:             else if (~rollq_ack & ~rollc_ack)
 90:                     roll_ack = 0;
 91:
 92:     always wait (halt_req & ~reset)
 93:     begin :rollback_cycle
 94:             #1;
 95:             haltc_ack = 1;
 96:             wait (roll_req);
 97:             #1;
 98:             disable input_cycle;
 99:             disable read_cycle;
100:             disable commit_cycle;
101:             disable arb_cycle;
102:             out_addr = XXX;
103:             rw_mode = XXX;
104:             out_retry = 0;
105:             in_ack = 0;
106:             chk_req = 0;
107:             out_req = 0;
108:             tri_dq_req = ZZZ;
109:             out_data = XXX;
110:             arbrd_req = 0;
111:             arbrd_ack = 0;
112:             arbwr_req = 0;
113:             arbwr_ack = 0;
114:             wrq_req = 0;
115:             com_ack = 0;
116:             rdq_req = 0;
117:
118:             #DLY_SEQ_COMP;
119:             diff = out_seq - val_seq;       // compare sequence numbers
120:             if (~diff [SEQ_WIDTH-1])
121:                 begin
122:                     go_read = 0;
123:                     out_seq = XXX;
124:                     out_reg = XXX;
125:                 end
126:
127:             rollc_ack = 1;
128:             fork
129:                 begin
130:                     wait (~roll_req);
131:                     #1;
132:                     rollc_ack = 0;
133:                 end
134:                 begin
135:                     wait (~halt_req);
136:                     #1;
137:                     haltc_ack = 0;
138:                 end
139:             join
140:     end
141:
142:     always wait (in_req & ~halt_req & ~reset)
```

```
143:    begin :input_cycle
144:            #1;
145:            if (in_rw_mode)
146:                begin
147:                    wait (~go_read & ~halt_req);
148:                    out_seq = in_seq;
149:                    out_reg = in_reg;
150:                    rd_addr = in_addr;
151:                    out_retry = in_retry;
152:                    #1;
153:                    wait (~halt_req);
154:                    go_read = 1;            // start read cycle
155:                    chk_req = 1;            // notify checker
156:                    wait (chk_ack);
157:                    #1;
158:                    fork                    // finish up transactions
159:                        begin
160:                            wait (~halt_req);
161:                            chk_req = 0;
162:                            wait (~chk_ack);
163:                        end
164:                        begin
165:                            wait (~halt_req);
166:                            in_ack = 1;
167:                            wait (~in_req);
168:                            #1;
169:                            wait (~halt_req);
170:                            in_ack = 0;
171:                        end
172:                    join
173:                end
174:            else
175:                begin
176:                    wrq_req = 1;            // write to queue
177:                    wait (wrq_ack);         // wait until actually accepted
178:                    #1;
179:                    chk_req = 1;            // notify checker
180:                    wait (chk_ack);
181:                    #1;
182:                    fork                    // finish up transactions
183:                        begin
184:                            wrq_req = 0;
185:                            chk_req = 0;
186:                            wait (~wrq_ack & ~chk_ack);
187:                        end
188:                        begin
189:                            in_ack = 1;
190:                            wait (~in_req);
191:                            #1;
192:                            in_ack = 0;
193:                        end
194:                    join
195:                end
196:    end
197:
198:    always wait (go_read & ~halt_req & ~reset)
199:    begin :read_cycle
200:            #1;
201:            rdq_req = 1;                            // search queue first
202:            wait (rdq_ack);
203:            #1;
204:            arbrd_req = 1;                          // request memory cycle
205:            wait (arbrd_ack);
206:            #1;
207:            if (match)
208:                begin                               // data still in queue
209:                    out_data = tri_qout;
210:                    rdq_req = 0;                    // release queue
211:                    rw_mode = 0;                    // write mode to access bus
212:                    #1;
213:                    wait (~halt_req);
```

```
214:                    tri_dq_req = 1;                 // write DQ directly
215:                    wait (dq_ack);
216:                    #1;
217:                    go_read = 0;
218:                    wait (~halt_req);
219:                    tri_dq_req = ZZZ;               // tri-state it
220:                    arbrd_req = 0;
221:                    wait (~dq_ack & ~arbrd_ack & ~rdq_ack);
222:                end
223:            else
224:                begin
225:                    rdq_req = 0;                    // release queue
226:                    out_addr = rd_addr;
227:                    rw_mode = 1;                    // read mode
228:                    #1;
229:                    wait (~halt_req);
230:                    out_req = 1;
231:                    wait (out_ack);
232:                    #1;
233:                    go_read = 0;
234:                    wait (~halt_req);
235:                    out_req = 0;
236:                    arbrd_req = 0;
237:                    wait (~out_ack & ~arbrd_ack & ~rdq_ack);
238:                end
239:    end
240:
241:    always wait (com_req & ~halt_req & ~reset)
242:    begin :commit_cycle
243:            wait (~com_wait);               // wait until validated
244:            #1;
245:            arbwr_req = 1;                  // request memory cycle
246:            wait (arbwr_ack);
247:            #1;
248:            out_addr = com_addr;
249:            out_data = com_data;
250:            rw_mode = 0;                    // write mode
251:            #1;
252:            wait (~halt_req);
253:            out_req = 1;                    // write to data memory
254:            wait (out_ack);
255:            #1;
256:            wait (~halt_req);
257:            com_ack = 1;                    // delete from queue
258:            out_req = 0;
259:            arbwr_req = 0;
260:            wait (~com_req);
261:            #1;
262:            wait (~halt_req);
263:            com_ack = 0;
264:            wait (~out_ack & ~arbwr_ack);
265:    end
266:
267:    always wait ((arbrd_req | arbwr_req) & ~halt_req & ~reset)
268:    begin :arb_cycle
269:            #1;
270:            if (arbrd_req)                  // read has priority over write
271:                begin
272:                    arbrd_ack = 1;
273:                    wait (~arbrd_req);
274:                    #1;
275:                    arbrd_ack = 0;
276:                end
277:            else
278:                begin
279:                    arbwr_ack = 1;
280:                    wait (~arbwr_req);
281:                    #1;
282:                    arbwr_ack = 0;
283:                end
284:    end
```

```
285:
286:    endmodule       // memdwb
287:
288:    // ==============================================================
289:    // MEM_DWB Queue: Note that sequence number is not used in the output.
290:
291:    module memdwbq (in_seq, in_wait, in_addr, in_data, in_req, in_ack,
292:            out_wait, out_addr, out_data, out_req, out_ack,
293:            val_seq, val_req, val_ack,
294:            read_addr, read_req, read_ack, match, tri_out,
295:            halt_req, halt_ack, roll_req, roll_ack, reset);
296:
297:    `include "parameter"
298:
299:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
300:    input [ADDR_WIDTH:0] in_addr, read_addr;
301:    input [DATA_WIDTH:0] in_data;
302:    output [ADDR_WIDTH:0] out_addr;
303:    output [DATA_WIDTH:0] out_data, tri_out;
304:    input in_wait, reset;
305:    output out_wait, match;
306:    input in_req, out_ack, val_req, read_req, halt_req, roll_req;
307:    output in_ack, out_req, val_ack, read_ack, halt_ack, roll_ack;
308:
309:    reg val_ack, read_ack, halt_ack, roll_ack;
310:
311:    reg read_ready, read_en1, read_en2, read_en3, read_en4, haltr_ack, rollr_ack;
312:
313:    wire [SEQ_WIDTH-1:0] seq1, seq2, seq3, not_used;
314:    wire [ADDR_WIDTH:0] addr1, addr2, addr3;
315:    wire [DATA_WIDTH:0] data1, data2, data3;
316:    wire match1, match2, match3, match4;
317:
318:    assign match = match1 | match2 | match3 | match4;
319:
320:
321:    memdwb_buf buf1 (in_seq, in_wait, in_addr, in_data, in_req, in_ack,
322:            seq1, wait1, addr1, data1, req1, ack1,
323:            val_seq, val_req, val_ack1,
324:            read_addr, read_req, read_ack1, match1, read_en1, tri_out,
325:            halt_req, halt_a1, roll_req, roll_a1, reset);
326:
327:    memdwb_buf buf2 (seq1, wait1, addr1, data1, req1, ack1,
328:            seq2, wait2, addr2, data2, req2, ack2,
329:            val_seq, val_req, val_ack2,
330:            read_addr, read_req, read_ack2, match2, read_en2, tri_out,
331:            halt_req, halt_a2, roll_req, roll_a2, reset);
332:
333:    memdwb_buf buf3 (seq2, wait2, addr2, data2, req2, ack2,
334:            seq3, wait3, addr3, data3, req3, ack3,
335:            val_seq, val_req, val_ack3,
336:            read_addr, read_req, read_ack3, match3, read_en3, tri_out,
337:            halt_req, halt_a3, roll_req, roll_a3, reset);
338:
339:    memdwb_buf buf4 (seq3, wait3, addr3, data3, req3, ack3,
340:            not_used, out_wait, out_addr, out_data, out_req, out_ack,
341:            val_seq, val_req, val_ack4,
342:            read_addr, read_req, read_ack4, match4, read_en4, tri_out,
343:            halt_req, halt_a4, roll_req, roll_a4, reset);
344:
345:
346:    always wait (reset)
347:    begin
348:            disable rollback_cycle;
349:            disable read_cycle;
350:            val_ack = 0;
351:            read_ack = 0;
352:            halt_ack = 0;
353:            roll_ack = 0;
354:            read_ready = 0;
355:            read_en1 = 0;
```

```
356:            read_en2 = 0;
357:            read_en3 = 0;
358:            read_en4 = 0;
359:            haltr_ack = 0;
360:            rollr_ack = 0;
361:            wait (~reset);
362:    end
363:
364:    always @(val_ack1 or val_ack2 or val_ack3 or val_ack4)
365:            if (val_ack1 & val_ack2 & val_ack3 & val_ack4)
366:                    val_ack = 1;
367:            else if (~val_ack1 & ~val_ack2 & ~val_ack3 & ~val_ack4)
368:                    val_ack = 0;
369:
370:    always @(read_ack1 or read_ack2 or read_ack3 or read_ack4)
371:            if (read_ack1 & read_ack2 & read_ack3 & read_ack4)
372:                    read_ready = 1;
373:            else if (~read_ack1 & ~read_ack2 & ~read_ack3 & ~read_ack4)
374:                    read_ready = 0;
375:
376:    always @(halt_a1 or halt_a2 or halt_a3 or halt_a4 or haltr_ack)
377:            if (halt_a1 & halt_a2 & halt_a3 & halt_a4 & haltr_ack)
378:                    halt_ack = 1;
379:            else if (~halt_a1 & ~halt_a2 & ~halt_a3 & ~halt_a4 & ~haltr_ack)
380:                    halt_ack = 0;
381:
382:    always @(roll_a1 or roll_a2 or roll_a3 or roll_a4 or rollr_ack)
383:            if (roll_a1 & roll_a2 & roll_a3 & roll_a4 & rollr_ack)
384:                    roll_ack = 1;
385:            else if (~roll_a1 & ~roll_a2 & ~roll_a3 & ~roll_a4 & ~rollr_ack)
386:                    roll_ack = 0;
387:
388:    always wait (halt_req & ~reset)
389:    begin :rollback_cycle
390:            #1;
391:            haltr_ack = 1;
392:            wait (roll_req);
393:            #1;
394:            disable read_cycle;
395:            read_ready = 0;
396:            read_ack = 0;
397:            read_en1 = 0;
398:            read_en2 = 0;
399:            read_en3 = 0;
400:            read_en4 = 0;
401:
402:            rollr_ack = 1;
403:            fork
404:                begin
405:                    wait (~roll_req);
406:                    #1;
407:                    rollr_ack = 0;
408:                end
409:                begin
410:                    wait (~halt_req);
411:                    #1;
412:                    haltr_ack = 0;
413:                end
414:            join
415:    end
416:
417:    always wait (read_ready & ~halt_req & ~reset)
418:    begin :read_cycle
419:            #1;
420:            casex ({match1, match2, match3, match4})        // priority decoder
421:                    4'b1???: read_en1 = 1;
422:                    4'b01??: read_en2 = 1;
423:                    4'b001?: read_en3 = 1;
424:                    4'b0001: read_en4 = 1;
425:            endcase
426:            #1;
```

```
427:            read_ack = 1;
428:            wait (~read_ready);
429:            #1;
430:            read_en1 = 0;
431:            read_en2 = 0;
432:            read_en3 = 0;
433:            read_en4 = 0;
434:            read_ack = 0;
435:    end
436:
437:    endmodule       // memdwbq
438:
439:    // ========================================================================
440:
441:    module memdwb_buf (in_seq, in_wait, in_addr, in_data, in_req, in_ack,
442:            out_seq, out_wait, out_addr, out_data, out_req, out_ack,
443:            val_seq, val_req, val_ack,
444:            read_addr, read_req, read_ack, match, read_en, tri_out,
445:            halt_req, halt_ack, roll_req, roll_ack, reset);
446:
447:    `include "parameter"
448:
449:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
450:    input [ADDR_WIDTH:0] in_addr, read_addr;
451:    input [DATA_WIDTH:0] in_data;
452:    output [SEQ_WIDTH-1:0] out_seq;
453:    output [ADDR_WIDTH:0] out_addr;
454:    output [DATA_WIDTH:0] out_data, tri_out;
455:    input in_wait, read_en, reset;
456:    output out_wait, match;
457:    input in_req, out_ack, val_req, read_req, halt_req, roll_req;
458:    output in_ack, out_req, val_ack, read_ack, halt_ack, roll_ack;
459:
460:    reg [SEQ_WIDTH-1:0] out_seq;
461:    reg [ADDR_WIDTH:0] out_addr;
462:    reg [DATA_WIDTH:0] out_data;
463:    reg out_wait, match;
464:    reg in_ack, out_req, val_ack, read_ack, halt_ack, roll_ack;
465:
466:    reg [SEQ_WIDTH-1:0] diff;
467:    reg valid;
468:
469:    assign tri_out = read_en ? out_data : ZZZ;
470:
471:    always wait (reset)
472:    begin
473:            disable validate_cycle;
474:            disable rollback_cycle;
475:            disable find_cycle;
476:            disable input_cycle;
477:            disable output_cycle;
478:            out_seq = XXX;
479:            out_addr = XXX;
480:            out_data = XXX;
481:            out_wait = XXX;
482:            match = 0;
483:            in_ack = 0;
484:            out_req = 0;
485:            val_ack = 0;
486:            read_ack = 0;
487:            halt_ack = 0;
488:            roll_ack = 0;
489:            valid = 0;
490:            wait (~reset);
491:    end
492:
493:    always wait (val_req & ~reset)          // not same time as rollback
494:    begin :validate_cycle                   // clear "wait" bit
495:            #1;
496:            val_ack = 1;
497:            wait (~val_req);
```

```
498:            #1;
499:            if (out_seq == val_seq)
500:                    out_wait = 0;
501:            #1;
502:            val_ack = 0;
503:    end
504:
505:    always wait (halt_req & ~reset)
506:    begin :rollback_cycle
507:            #1;
508:            halt_ack = 1;
509:            wait (roll_req);
510:            #1;
511:            disable find_cycle;
512:            disable input_cycle;
513:            disable output_cycle;
514:            match = 0;
515:            in_ack = 0;
516:            out_req = 0;
517:            read_ack = 0;
518:
519:            #DLY_SEQ_COMP;
520:            diff = out_seq - val_seq;       // compare sequence numbers
521:            if (~diff [SEQ_WIDTH-1])
522:                begin
523:                    valid = 0;
524:                    out_seq = XXX;
525:                    out_addr = XXX;
526:                    out_data = XXX;
527:                    out_wait = XXX;
528:                end
529:
530:            roll_ack = 1;
531:            fork
532:                begin
533:                    wait (~roll_req);
534:                    #1;
535:                    roll_ack = 0;
536:                end
537:                begin
538:                    wait (~halt_req);
539:                    #1;
540:                    halt_ack = 0;
541:                end
542:            join
543:    end
544:
545:    always wait (read_req & ~halt_req & ~reset)     // find uncommitted data
546:    begin :find_cycle
547:            #1;
548:            match = valid & (out_addr == read_addr);
549:            #1;
550:            read_ack = 1;
551:            wait (~read_req);
552:            #1;
553:            match = 0;
554:            read_ack = 0;
555:    end
556:
557:    always wait (in_req & ~valid & ~val_req & ~read_req & ~halt_req & ~reset)
558:    begin :input_cycle
559:            #1;
560:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
561:            out_seq = in_seq;
562:            out_wait = in_wait;
563:            out_addr = in_addr;
564:            out_data = in_data;
565:            in_ack = 1;
566:            valid = 1;
567:            wait (~in_req);
568:            #1;
```

```
569:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
570:            in_ack = 0;
571:    end
572:
573:    always wait (valid & ~val_req & ~read_req & ~halt_req & ~reset)
574:    begin :output_cycle
575:            #1;
576:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
577:            out_req = 1;
578:            wait (out_ack);
579:            #1;
580:            valid = 0;
581:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
582:            out_req = 0;
583:            wait (~out_ack);
584:    end
585:
586:    endmodule       // memdwb_buf
```

pc.v

```
 1:    // Program Counter
 2:
 3:    module pc (pc_load, in_retry, load_req, load_ack,
 4:            pc_out, out_retry, out_req, out_ack, reset);
 5:
 6:    `include "parameter"
 7:
 8:    input [ADDR_WIDTH:0] pc_load;
 9:    output [ADDR_WIDTH:0] pc_out;
10:    input in_retry, load_req, out_ack, reset;
11:    output out_retry, load_ack, out_req;
12:
13:    reg [ADDR_WIDTH:0] pc_out;
14:    reg out_retry, load_ack, out_req;
15:
16:    reg [5:0] loop;                 // ADDR_WIDTH = 32 bits max
17:    reg parity;
18:
19:    always wait (reset)
20:    begin
21:            disable increment_cycle;
22:            disable load_cycle;
23:            out_retry = 0;
24:            load_ack = 0;
25:            out_req = 0;
26:            pc_out = 0;
27:            wait (~reset);
28:            out_req = 1;
29:    end
30:
31:    always wait (load_req & ~reset)
32:    begin :load_cycle
33:            #1;
34:            disable increment_cycle;
35:            out_req = 0;
36:            pc_out = {pc_load[ADDR_WIDTH:ADDR_IGNORE], {ADDR_IGNORE{1'b0}}};
37:            out_retry = in_retry;           // rollback flag
38:            load_ack = 1;
39:            wait (~load_req);
40:            #1;
41:            load_ack = 0;
42:            out_req = 1;
43:    end
44:
45:    always wait (out_ack & ~load_req & ~reset)
46:    begin :increment_cycle
47:            #1;
48:            out_req = 0;
49:            #DLY_PC_INC;
50:            pc_out = pc_out + ADDR_INC;
51:            out_retry = 0;                  // reset rollback flag
52:
53:            parity = 0;
54:            for (loop=0; loop<ADDR_WIDTH; loop=loop+1)      // calculate parity
55:                    parity = parity ^ pc_out [loop];
56:            pc_out [ADDR_WIDTH] = parity;
57:
58:            wait (~out_ack);
59:            #1;
60:            out_req = 1;
61:    end
62:
63:    endmodule       // pc
```

regdwb.v

```
 1:    // Delayed Write Buffer for Register File: Note initial wait bit = 1
 2:
 3:    module regdwb (in_seq, in_reg, in_data, in_req, in_ack,
 4:            out_seq, out_reg, out_data, chk_req, chk_ack, clr_req, clr_ack,
 5:            val_seq, val_req, val_ack,
 6:            rd_reg1, rd_reg2, sim_f, tri_data1, tri_data2, rd_req, rd_ack,
 7:            rf_data1, rf_data2, rdf_req, rdf_ack,
 8:            wrf_reg, wrf_data, wrf_req, wrf_ack,
 9:            halt_req, halt_ack, roll_req, roll_ack, reset);
10:
11:    `include "parameter"
12:
13:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
14:    input [REG_WIDTH-1:0] in_reg, rd_reg1, rd_reg2;
15:    input [DATA_WIDTH:0] in_data, rf_data1, rf_data2;
16:    output [SEQ_WIDTH-1:0] out_seq;
17:    output [REG_WIDTH-1:0] out_reg, wrf_reg;
18:    output [DATA_WIDTH:0] out_data, tri_data1, tri_data2, wrf_data;
19:    input sim_f, reset;
20:    input in_req, chk_ack, clr_ack, val_req, rd_req, rdf_ack, wrf_ack,
21:            halt_req, roll_req;
22:    output in_ack, chk_req, clr_req, val_ack, rd_ack, rdf_req, wrf_req,
23:            halt_ack, roll_ack;
24:
25:    reg [SEQ_WIDTH-1:0] out_seq;
26:    reg [REG_WIDTH-1:0] out_reg, wrf_reg;
27:    reg [DATA_WIDTH:0] out_data, wrf_data;
28:    reg in_ack, chk_req, clr_req, rd_ack, wrf_req, halt_ack, roll_ack;
29:
30:    reg [DATA_WIDTH:0] out_data1, out_data2;
31:    reg [SEQ_WIDTH-1:0] diff;
32:    reg go_check, go_clear, inq_req, com_ack, haltc_ack, rollc_ack;
33:
34:    tri [DATA_WIDTH:0] tri_out1, tri_out2;
35:    wire [REG_WIDTH-1:0] com_reg;
36:    wire [DATA_WIDTH:0] com_data;
37:    wire rdf_req = rd_req;                  // route read request
38:
39:    regdwbq regdwbq (in_seq, 1'b1, in_reg, in_data, inq_req, inq_ack,
40:            com_wait, com_reg, com_data, com_req, com_ack,
41:            val_seq, val_req, val_ack,
42:            rd_reg1, rd_reg2, rd_req, rdq_ack,
43:            match1, match2, tri_out1, tri_out2,
44:            halt_req, haltq_ack, roll_req, rollq_ack, reset);
45:
46:    assign tri_data1 = rd_req ? out_data1 : ZZZ;
47:    assign tri_data2 = rd_req ? out_data2 : ZZZ;
48:
49:    always wait (reset)
50:    begin
51:            disable rollback_cycle;
52:            disable input_cycle;
53:            disable check_cycle;
54:            disable clear_cycle;
55:            disable commit_cycle;
56:            disable read_cycle;
57:            out_seq = XXX;
58:            out_reg = XXX;
59:            wrf_reg = XXX;
60:            out_data = XXX;
61:            wrf_data = XXX;
62:            in_ack = 0;
63:            chk_req = 0;
64:            clr_req = 0;
65:            rd_ack = 0;
66:            wrf_req = 0;
67:            halt_ack = 0;
68:            roll_ack = 0;
69:            out_data1 = XXX;
70:            out_data2 = XXX;
71:            go_check = 0;
```

```
 72:            go_clear = 0;
 73:            inq_req = 0;
 74:            com_ack = 0;
 75:            haltc_ack = 0;
 76:            rollc_ack = 0;
 77:            wait (~reset);
 78:    end
 79:
 80:    always @(haltq_ack or haltc_ack)
 81:            if (haltq_ack & haltc_ack)
 82:                    halt_ack = 1;
 83:            else if (~haltq_ack & ~haltc_ack)
 84:                    halt_ack = 0;
 85:
 86:    always @(rollq_ack or rollc_ack)
 87:            if (rollq_ack & rollc_ack)
 88:                    roll_ack = 1;
 89:            else if (~rollq_ack & ~rollc_ack)
 90:                    roll_ack = 0;
 91:
 92:    always wait (halt_req & ~reset)
 93:    begin :rollback_cycle
 94:            #1;
 95:            haltc_ack = 1;
 96:            wait (roll_req);
 97:            #1;
 98:            disable input_cycle;
 99:            disable check_cycle;
100:            disable clear_cycle;
101:            disable commit_cycle;
102:            disable read_cycle;
103:            wrf_reg = XXX;
104:            wrf_data = XXX;
105:            in_ack = 0;
106:            chk_req = 0;
107:            clr_req = 0;
108:            rd_ack = 0;
109:            wrf_req = 0;
110:            out_data1 = XXX;
111:            out_data2 = XXX;
112:            inq_req = 0;
113:            com_ack = 0;
114:
115:            #DLY_SEQ_COMP;
116:            diff = out_seq - val_seq;       // compare sequence numbers
117:            if (~diff [SEQ_WIDTH-1])
118:                begin
119:                    go_check = 0;
120:                    go_clear = 0;
121:                    out_seq = XXX;
122:                    out_reg = XXX;
123:                    out_data = XXX;
124:                end
125:
126:            rollc_ack = 1;
127:            fork
128:                begin
129:                    wait (~roll_req);
130:                    #1;
131:                    rollc_ack = 0;
132:                end
133:                begin
134:                    wait (~halt_req);
135:                    #1;
136:                    haltc_ack = 0;
137:                end
138:            join
139:    end
140:
141:    always wait (in_req & ~go_check & ~go_clear & ~halt_req & ~reset)
142:    begin :input_cycle
```

```
143:            #1;
144:            wait (~halt_req);
145:            out_seq = in_seq;
146:            out_reg = in_reg;
147:            out_data = in_data;
148:            inq_req = 1;
149:            wait (inq_ack);                 // wait until actually accepted
150:            #1;
151:            in_ack = 1;
152:            go_check = 1;
153:            go_clear = 1;
154:            wait (~in_req);
155:            #1;
156:            wait (~halt_req);
157:            inq_req = 0;
158:            wait (~inq_ack);
159:            #1;
160:            in_ack = 0;
161:    end
162:
163:    always wait (go_check & ~halt_req & ~reset)
164:    begin :check_cycle
165:            #1;
166:            wait (~halt_req);
167:            chk_req = 1;
168:            wait (chk_ack);
169:            #1;
170:            go_check = 0;
171:            wait (~halt_req);
172:            chk_req = 0;
173:            wait (~chk_ack);
174:    end
175:
176:    always wait (go_clear & ~halt_req & ~reset)
177:    begin :clear_cycle
178:            #1;
179:            wait (~halt_req);
180:            clr_req = 1;
181:            wait (clr_ack);
182:            #1;
183:            go_clear = 0;
184:            wait (~halt_req);
185:            clr_req = 0;
186:            wait (~clr_ack);
187:    end
188:
189:    always wait (com_req & ~halt_req & ~reset)
190:    begin :commit_cycle
191:            wait (~com_wait);               // wait until validated
192:            #1;
193:            wrf_reg = com_reg;
194:            wrf_data = com_data;
195:            #1;
196:            wait (~halt_req);
197:            wrf_req = 1;                    // write to register file
198:            wait (wrf_ack);
199:            #1;
200:            wait (~rd_req & ~halt_req);
201:            com_ack = 1;                    // delete from queue
202:            wrf_req = 0;
203:            wait (~com_req);
204:            #1;
205:            wait (~rd_req & ~halt_req);
206:            com_ack = 0;
207:            wait (~wrf_ack);
208:    end
209:
210:    always wait (rdq_ack & rdf_ack & ~halt_req & ~reset)
211:    begin :read_cycle
212:            #1;
213:            out_data1 = match1 ? tri_out1 : rf_data1;
```

```
214:            out_data2 = match2 ? tri_out2 : rf_data2;
215:            if (sim_f)                      // simulate fault
216:                begin
217:                    out_data1 [DATA_WIDTH] = ~out_data1 [DATA_WIDTH];
218:                    out_data2 [DATA_WIDTH] = ~out_data2 [DATA_WIDTH];
219:                end
220:            #1;
221:            rd_ack = 1;
222:            wait (~rdq_ack & ~rdf_ack);
223:            #1;
224:            rd_ack = 0;
225:    end
226:
227:    endmodule       // regdwb
228:
229:    // ========================================================================
230:    // REG_DWB Queue: Note that sequence number is not used in the output.
231:
232:    module regdwbq (in_seq, in_wait, in_reg, in_data, in_req, in_ack,
233:            out_wait, out_reg, out_data, out_req, out_ack,
234:            val_seq, val_req, val_ack,
235:            read_reg1, read_reg2, read_req, read_ack,
236:            match1, match2, tri_out1, tri_out2,
237:            halt_req, halt_ack, roll_req, roll_ack, reset);
238:
239:    `include "parameter"
240:
241:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
242:    input [REG_WIDTH-1:0] in_reg, read_reg1, read_reg2;
243:    input [DATA_WIDTH:0] in_data;
244:    output [REG_WIDTH-1:0] out_reg;
245:    output [DATA_WIDTH:0] out_data, tri_out1, tri_out2;
246:    input in_wait, reset;
247:    output out_wait, match1, match2;
248:    input in_req, out_ack, val_req, read_req, halt_req, roll_req;
249:    output in_ack, out_req, val_ack, read_ack, halt_ack, roll_ack;
250:
251:    reg val_ack, read_ack, halt_ack, roll_ack;
252:
253:    reg read_ready, read_en11, read_en21, read_en31, read_en41, read_en12,
254:            read_en22, read_en32, read_en42, haltr_ack, rollr_ack;
255:
256:    wire [SEQ_WIDTH-1:0] seq1, seq2, seq3, not_used;
257:    wire [REG_WIDTH-1:0] reg1, reg2, reg3;
258:    wire [DATA_WIDTH:0] data1, data2, data3;
259:    wire match11, match21, match31, match41, match12, match22, match32, match42;
260:
261:    assign match1 = match11 | match21 | match31 | match41;
262:    assign match2 = match12 | match22 | match32 | match42;
263:
264:
265:    regdwb_buf buf1 (in_seq, in_wait, in_reg, in_data, in_req, in_ack,
266:            seq1, wait1, reg1, data1, req1, ack1,
267:            val_seq, val_req, val_ack1,
268:            read_reg1, read_reg2, read_req, read_ack1,
269:            match11, match12, read_en11, read_en12, tri_out1, tri_out2,
270:            halt_req, halt_a1, roll_req, roll_a1, reset);
271:
272:    regdwb_buf buf2 (seq1, wait1, reg1, data1, req1, ack1,
273:            seq2, wait2, reg2, data2, req2, ack2,
274:            val_seq, val_req, val_ack2,
275:            read_reg1, read_reg2, read_req, read_ack2,
276:            match21, match22, read_en21, read_en22, tri_out1, tri_out2,
277:            halt_req, halt_a2, roll_req, roll_a2, reset);
278:
279:    regdwb_buf buf3 (seq2, wait2, reg2, data2, req2, ack2,
280:            seq3, wait3, reg3, data3, req3, ack3,
281:            val_seq, val_req, val_ack3,
282:            read_reg1, read_reg2, read_req, read_ack3,
283:            match31, match32, read_en31, read_en32, tri_out1, tri_out2,
284:            halt_req, halt_a3, roll_req, roll_a3, reset);
```

```
285:
286:    regdwb_buf buf4 (seq3, wait3, reg3, data3, req3, ack3,
287:            not_used, out_wait, out_reg, out_data, out_req, out_ack,
288:            val_seq, val_req, val_ack4,
289:            read_reg1, read_reg2, read_req, read_ack4,
290:            match41, match42, read_en41, read_en42, tri_out1, tri_out2,
291:            halt_req, halt_a4, roll_req, roll_a4, reset);
292:
293:
294:    always wait (reset)
295:    begin
296:            disable read_cycle;
297:            val_ack = 0;
298:            read_ack = 0;
299:            read_ready = 0;
300:            halt_ack = 0;
301:            roll_ack = 0;
302:            read_en11 = 0;
303:            read_en21 = 0;
304:            read_en31 = 0;
305:            read_en41 = 0;
306:            read_en12 = 0;
307:            read_en22 = 0;
308:            read_en32 = 0;
309:            read_en42 = 0;
310:            haltr_ack = 0;
311:            rollr_ack = 0;
312:            wait (~reset);
313:    end
314:
315:    always @(val_ack1 or val_ack2 or val_ack3 or val_ack4)
316:            if (val_ack1 & val_ack2 & val_ack3 & val_ack4)
317:                    val_ack = 1;
318:            else if (~val_ack1 & ~val_ack2 & ~val_ack3 & ~val_ack4)
319:                    val_ack = 0;
320:
321:    always @(read_ack1 or read_ack2 or read_ack3 or read_ack4)
322:            if (read_ack1 & read_ack2 & read_ack3 & read_ack4)
323:                    read_ready = 1;
324:            else if (~read_ack1 & ~read_ack2 & ~read_ack3 & ~read_ack4)
325:                    read_ready = 0;
326:
327:    always @(halt_a1 or halt_a2 or halt_a3 or halt_a4 or haltr_ack)
328:            if (halt_a1 & halt_a2 & halt_a3 & halt_a4 & haltr_ack)
329:                    halt_ack = 1;
330:            else if (~halt_a1 & ~halt_a2 & ~halt_a3 & ~halt_a4 & ~haltr_ack)
331:                    halt_ack = 0;
332:
333:    always @(roll_a1 or roll_a2 or roll_a3 or roll_a4 or rollr_ack)
334:            if (roll_a1 & roll_a2 & roll_a3 & roll_a4 & rollr_ack)
335:                    roll_ack = 1;
336:            else if (~roll_a1 & ~roll_a2 & ~roll_a3 & ~roll_a4 & ~rollr_ack)
337:                    roll_ack = 0;
338:
339:    always wait (halt_req & ~reset)
340:    begin :rollback_cycle
341:            #1;
342:            haltr_ack = 1;
343:            wait (roll_req);
344:            #1;
345:            disable read_cycle;
346:            read_ready = 0;
347:            read_ack = 0;
348:            read_en11 = 0;
349:            read_en21 = 0;
350:            read_en31 = 0;
351:            read_en41 = 0;
352:            read_en12 = 0;
353:            read_en22 = 0;
354:            read_en32 = 0;
355:            read_en42 = 0;
```

```
356:
357:            rollr_ack = 1;
358:            fork
359:                begin
360:                    wait (~roll_req);
361:                    #1;
362:                    rollr_ack = 0;
363:                end
364:                begin
365:                    wait (~halt_req);
366:                    #1;
367:                    haltr_ack = 0;
368:                end
369:            join
370:    end
371:
372:    always wait (read_ready & ~halt_req & ~reset)
373:    begin :read_cycle
374:            #1;
375:            casex ({match11, match21, match31, match41})    // priority decoder
376:                    4'b1???: read_en11 = 1;
377:                    4'b01??: read_en21 = 1;
378:                    4'b001?: read_en31 = 1;
379:                    4'b0001: read_en41 = 1;
380:            endcase
381:            casex ({match12, match22, match32, match42})
382:                    4'b1???: read_en12 = 1;
383:                    4'b01??: read_en22 = 1;
384:                    4'b001?: read_en32 = 1;
385:                    4'b0001: read_en42 = 1;
386:            endcase
387:            #1;
388:            read_ack = 1;
389:            wait (~read_ready);
390:            #1;
391:            read_en11 = 0;
392:            read_en21 = 0;
393:            read_en31 = 0;
394:            read_en41 = 0;
395:            read_en12 = 0;
396:            read_en22 = 0;
397:            read_en32 = 0;
398:            read_en42 = 0;
399:            read_ack = 0;
400:    end
401:
402:    endmodule       // regdwbq
403:
404:    // ====================================================================
405:
406:    module regdwb_buf (in_seq, in_wait, in_reg, in_data, in_req, in_ack,
407:            out_seq, out_wait, out_reg, out_data, out_req, out_ack,
408:            val_seq, val_req, val_ack,
409:            read_reg1, read_reg2, read_req, read_ack,
410:            match1, match2, read_en1, read_en2, tri_out1, tri_out2,
411:            halt_req, halt_ack, roll_req, roll_ack, reset);
412:
413:    `include "parameter"
414:
415:    input [SEQ_WIDTH-1:0] in_seq, val_seq;
416:    input [REG_WIDTH-1:0] in_reg, read_reg1, read_reg2;
417:    input [DATA_WIDTH:0] in_data;
418:    output [SEQ_WIDTH-1:0] out_seq;
419:    output [REG_WIDTH-1:0] out_reg;
420:    output [DATA_WIDTH:0] out_data, tri_out1, tri_out2;
421:    input in_wait, read_en1, read_en2, reset;
422:    output out_wait, match1, match2;
423:    input in_req, out_ack, val_req, read_req, halt_req, roll_req;
424:    output in_ack, out_req, val_ack, read_ack, halt_ack, roll_ack;
425:
426:    reg [SEQ_WIDTH-1:0] out_seq;
```

```
427:    reg [REG_WIDTH-1:0] out_reg;
428:    reg [DATA_WIDTH:0] out_data;
429:    reg out_wait, match1, match2;
430:    reg in_ack, out_req, val_ack, read_ack, halt_ack, roll_ack;
431:
432:    reg [SEQ_WIDTH-1:0] diff;
433:    reg valid;
434:
435:    assign tri_out1 = read_en1 ? out_data : ZZZ;
436:    assign tri_out2 = read_en2 ? out_data : ZZZ;
437:
438:    always wait (reset)
439:    begin
440:            disable validate_cycle;
441:            disable rollback_cycle;
442:            disable find_cycle;
443:            disable input_cycle;
444:            disable output_cycle;
445:            out_seq = XXX;
446:            out_reg = XXX;
447:            out_data = XXX;
448:            out_wait = XXX;
449:            match1 = 0;
450:            match2 = 0;
451:            in_ack = 0;
452:            out_req = 0;
453:            val_ack = 0;
454:            read_ack = 0;
455:            halt_ack = 0;
456:            roll_ack = 0;
457:            valid = 0;
458:            wait (~reset);
459:    end
460:
461:    always wait (val_req & ~reset)          // not same time as rollback
462:    begin :validate_cycle                   // clear "wait" bit
463:            #1;
464:            val_ack = 1;
465:            wait (~val_req);
466:            #1;
467:            if (out_seq == val_seq)
468:                    out_wait = 0;
469:            #1;
470:            val_ack = 0;
471:    end
472:
473:    always wait (halt_req & ~reset)
474:    begin :rollback_cycle
475:            #1;
476:            halt_ack = 1;
477:            wait (roll_req);
478:            #1;
479:            disable find_cycle;
480:            disable input_cycle;
481:            disable output_cycle;
482:            match1 = 0;
483:            match2 = 0;
484:            in_ack = 0;
485:            out_req = 0;
486:            read_ack = 0;
487:
488:            #DLY_SEQ_COMP;
489:            diff = out_seq - val_seq;       // compare sequence numbers
490:            if (~diff [SEQ_WIDTH-1])
491:                begin
492:                    valid = 0;
493:                    out_seq = XXX;
494:                    out_reg = XXX;
495:                    out_data = XXX;
496:                    out_wait = XXX;
497:                end
```

```
498:
499:            roll_ack = 1;
500:            fork
501:                begin
502:                    wait (~roll_req);
503:                    #1;
504:                    roll_ack = 0;
505:                end
506:                begin
507:                    wait (~halt_req);
508:                    #1;
509:                    halt_ack = 0;
510:                end
511:            join
512:    end
513:
514:    always wait (read_req & ~halt_req & ~reset)     // find uncommitted data
515:    begin :find_cycle
516:            #1;                                 // do NOT match R0 !!!
517:            match1 = valid & (out_reg == read_reg1) & (read_reg1 != 0);
518:            match2 = valid & (out_reg == read_reg2) & (read_reg2 != 0);
519:            #1;
520:            read_ack = 1;
521:            wait (~read_req);
522:            #1;
523:            match1 = 0;
524:            match2 = 0;
525:            read_ack = 0;
526:    end
527:
528:    always wait (in_req & ~valid & ~val_req & ~read_req & ~halt_req & ~reset)
529:    begin :input_cycle
530:            #1;
531:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
532:            out_seq = in_seq;
533:            out_wait = in_wait;
534:            out_reg = in_reg;
535:            out_data = in_data;
536:            in_ack = 1;
537:            valid = 1;
538:            wait (~in_req);
539:            #1;
540:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
541:            in_ack = 0;
542:    end
543:
544:    always wait (valid & ~val_req & ~read_req & ~halt_req & ~reset)
545:    begin :output_cycle
546:            #1;
547:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
548:            out_req = 1;
549:            wait (out_ack);
550:            #1;
551:            valid = 0;
552:            wait (~val_req & ~val_ack & ~read_req & ~halt_req);
553:            out_req = 0;
554:            wait (~out_ack);
555:    end
556:
557:    endmodule       // regdwb_buf
```

regfile.v

```
 1:    // Register File (3-port: 1 write, 2 read)
 2:
 3:    module regfile (wr_reg, wr_data, wr_req, wr_ack,
 4:            rd_reg1, rd_reg2, rd_data1, rd_data2, rd_req, rd_ack,
 5:            halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [REG_WIDTH-1:0] wr_reg, rd_reg1, rd_reg2;
10:    input [DATA_WIDTH:0] wr_data;
11:    output [DATA_WIDTH:0] rd_data1, rd_data2;
12:    input wr_req, rd_req, halt_req, roll_req, reset;
13:    output wr_ack, rd_ack, halt_ack, roll_ack;
14:
15:    reg [DATA_WIDTH:0] rd_data1, rd_data2, rfile [0:REG_SIZE-1];
16:    reg wr_ack, rd_ack, halt_ack, roll_ack;
17:    reg [REG_WIDTH:0] loop;         // one extra bit for loop termination
18:
19:    always wait (reset)
20:    begin
21:            disable rollback_cycle;
22:            disable write_cycle;
23:            disable read_cycle;
24:            rd_data1 = XXX;
25:            rd_data2 = XXX;
26:            wr_ack = 0;
27:            rd_ack = 0;
28:            halt_ack = 0;
29:            roll_ack = 0;
30:            for (loop = 0; loop < REG_SIZE; loop = loop + 1)
31:                    rfile [loop] = XXX;
32:            wait (~reset);
33:    end
34:
35:    always wait (halt_req & ~reset)
36:    begin :rollback_cycle
37:            #1;
38:            halt_ack = 1;
39:            wait (roll_req);
40:            #1;
41:            disable write_cycle;
42:            disable read_cycle;
43:            rd_data1 = XXX;
44:            rd_data2 = XXX;
45:            wr_ack = 0;
46:            rd_ack = 0;
47:
48:            roll_ack = 1;
49:            fork
50:                begin
51:                    wait (~roll_req);
52:                    #1;
53:                    roll_ack = 0;
54:                end
55:                begin
56:                    wait (~halt_req);
57:                    #1;
58:                    halt_ack = 0;
59:                end
60:            join
61:    end
62:
63:    always wait (wr_req & ~halt_req & ~reset)
64:    begin :write_cycle
65:            #DLY_RF_WR;
66:            wait (~halt_req);
67:            rfile [wr_reg] = wr_data;
68:            wr_ack = 1;
69:            wait (~wr_req);
70:            #1;
71:            wait (~halt_req);
```

```
72:             wr_ack = 0;
73:     end
74:
75:     always wait (rd_req & ~halt_req & ~reset)
76:     begin :read_cycle
77:             #DLY_RF_RD;
78:             rd_data1 = rd_reg1 ? rfile [rd_reg1] : 0;
79:             rd_data2 = rd_reg2 ? rfile [rd_reg2] : 0;
80:             #1;
81:             wait (~halt_req);
82:             rd_ack = 1;
83:             wait (~rd_req);
84:             #1;
85:             wait (~halt_req);
86:             rd_ack = 0;
87:     end
88:
89:     endmodule       // regfile
```

restable.v

```
 1:    // Reservation Table (for registers)
 2:
 3:    module restable (seq_num, reg_w, reg_r1, reg_r2, res_req, res_ack,
 4:            reg_clr, clr_req, clr_ack,
 5:            val_seq, halt_req, halt_ack, roll_req, roll_ack, reset);
 6:
 7:    `include "parameter"
 8:
 9:    input [SEQ_WIDTH-1:0] seq_num, val_seq;
10:    input [REG_WIDTH-1:0] reg_w, reg_r1, reg_r2, reg_clr;
11:    input res_req, clr_req, halt_req, roll_req, reset;
12:    output res_ack, clr_ack, halt_ack, roll_ack;
13:
14:    reg [SEQ_WIDTH-1:0] seq_table [0:REG_SIZE-1];
15:    reg res_table [0:REG_SIZE-1];
16:    reg res_ack, clr_ack, halt_ack, roll_ack;
17:
18:    reg [SEQ_WIDTH-1:0] diff;
19:    reg [REG_WIDTH:0] loop;         // one extra bit for loop termination
20:
21:    always wait (reset)
22:    begin
23:            disable rollback_cycle;
24:            disable reserve_cycle;
25:            disable clear_cycle;
26:            res_ack = 0;
27:            clr_ack = 0;
28:            halt_ack = 0;
29:            roll_ack = 0;
30:            for (loop = 0; loop < REG_SIZE; loop = loop + 1)
31:                    res_table [loop] = 0;
32:            wait (~reset);
33:    end
34:
35:    always wait (halt_req & ~reset)
36:    begin :rollback_cycle
37:            #1;
38:            halt_ack = 1;
39:            wait (roll_req);
40:            #1;
41:            disable reserve_cycle;
42:            disable clear_cycle;
43:            res_ack = 0;
44:            clr_ack = 0;
45:
46:            #DLY_SEQ_COMP;
47:            for (loop = 1; loop < REG_SIZE; loop = loop + 1)
48:                begin
49:                    diff = seq_table [loop] - val_seq;
50:                    if (~diff[SEQ_WIDTH-1])
51:                            res_table [loop] = 0;
52:                end
53:
54:            roll_ack = 1;
55:            fork
56:                begin
57:                    wait (~roll_req);
58:                    #1;
59:                    roll_ack = 0;
60:                end
61:                begin
62:                    wait (~halt_req);
63:                    #1;
64:                    halt_ack = 0;
65:                end
66:            join
67:    end
68:
69:    always wait (res_req & ~halt_req & ~reset)
70:    begin :reserve_cycle
71:            #DLY_RT_RES;
```

```
72:             wait (~res_table [reg_w] & ~res_table [reg_r1] &
73:                     ~res_table [reg_r2]);
74:             #1;
75:             wait (~halt_req);
76:             res_table [reg_w] = 1 & (reg_w != 0);   // R0 always available
77:             seq_table [reg_w] = seq_num;
78:             res_ack = 1;
79:             wait (~res_req);
80:             #1;
81:             wait (~halt_req);
82:             res_ack = 0;
83:     end
84:
85:     always wait (clr_req & ~halt_req & ~reset)
86:     begin :clear_cycle
87:             #DLY_RT_CLR;
88:             wait (~halt_req);
89:             res_table [reg_clr] = 0;
90:             clr_ack = 1;
91:             wait (~clr_req);
92:             #1;
93:             wait (~halt_req);
94:             clr_ack = 0;
95:     end
96:
97:     endmodule       // restable
```