```
unsigned add(unsigned a, unsigned b);
model pcu${bit_width}{
 port{
   clock clock;
    in load, reset, hold, data_in[$w_1:0];
   out data_out[$w_1:0];
                                                                          }
   register reg[$w_1:0];
 default_control{
  load = '0';
reset = '0';
   hold = '1';
  /** no operation*/
  function nop : idle{
   control{
     in load, reset, hold;
                                                                          }
   protocol{
      [load == 0 && reset == 0 && hold == 1]{
 }
  /** reset */
  function reset : reset{
    assignment{
     reg = 0;
   control{
     in reset;
     [reset == 1]{
       store reg;
                                                                          }
   }
  /** increment */
  function inc{
   assignment{
     reg = add(reg, $inc_step);
  /** write : set program counter value */
 function write{
   input{
     bit_vector data_in;
                                                                      IMAU
   assignment{
     reg = data_in;
   control{
     in bit load;
   protocol{
  [load = '1' && hold data_in]{
       store reg;
  /** read : read program counter value */
  function read{
   output{
     bit_vector data_out;
 priority{ ( reset > ( inc | write ) ), read}
IR
                                                                      DMAU
/** ${bit_width}-bit register */
model reg${bit_width}{
 port{
   clock clock;
                                                                          in
        reset, enb;
   in
                                                                          in
          data in${range};
    in
   out
        data_out${range};
  storage{
   register reg${range};
                                                                          in
 default_control{
   reset = 0;
    enb = 0;
                                                                        }
```

PC

```
/** no operation */
  function nop : idle{
    control{
     in reset, enb;
    protocol{
      [reset == 0 && enb == 0]{
      }
  /** reset */
  function reset: reset{
    assignment{
     reg = 0;
    control{
     in reset;
    }protocol{
     [reset == 1]{
       store reg;
  /** write */
  function write{
    input{
     bit_vector data_in;
    assignment{
     reg = data_in;
    control{
     in enb;
   protocol{
     [enb == 1 && hold data_in]{
        store reg;
      }
  /** read */
  function read{
    input{
    output{
     bit_vector data_out = reg;
 priority{ ( reset > ( nop | write )), read }
/** ${bit_width}-bit instruction memory access unit */
model imau${bit_width}{
  port{
    in addr[${a_range}];
    out addr_bus[${a_range}];
in data_bus[${b_range}];
    out data[${b_range}];
  /** read */
  function read{
    input{
     bit_vector addr;
   output{
     bit_vector data = data_bus;
    protocol{
     valid data;
/** ${bit_width}-bit data memory access unit */
model dmau${bit_width}{
  port{
          reset;
          req, rw, ${acmode_str}${ext_str}addr[${a_range}],
data in[${b range}];
    out ${aerr_str}req_bus, w_mode_bus$wmode_str,
addr_bus[${a_range}];
    inout data_bus[${b_range}];
         ack bus;
    out ack, data_out[${b_range}];
  default_control{
    reset = 0;
    req
            = 0:
```

```
protocol{
  /** no operation */
                                                                                repeat [req == 1 && rw == 0 && ac mode == $str &&
  function nop : idle{
                                                                          ext_mode == 0] until (ack == 1 || reset == 1);
                                                                               if (ack == 1) {
    control{
      in reset, req;
                                                                                  valid data_out;
                                                                                  valid addr_err;
    protocol{
                                                                               }
      [reset ==0 && req == 0]{
                                                                             }
                                                                         FHM_DL_FUNC
  /** reset */
  function reset : reset{
                                                                                  print <<FHM_DL_FUNC
    assignment{
                                                                            /** store ${bit_width} bits */
     reg = 0;
                                                                            function s_${bit_width}{
    control{
                                                                              input{
                                                                               bit_vector addr;
bit_vector data_in;
      in reset;
    protocol{
     [reset == 0]{
        store reg;
                                                                               bit addr_err = addr_err(addr, ac_mode);
      }
                                                                              control{
                                                                                in reset, req, rw, ac_mode;
FHM_DL_FUNC
                                                                              protocol{
                                                                         repeat [req == 1 && rw == 1 && ac_mode == ${b_str}]
until (ack == 1 || reset == 1);
if ($acmode_num != 0) {
                                                                               if (ack == 1) {
         print <<FHM_DL_FUNC
                                                                                  valid addr_err;
  /** load ${bit_width} bits */
  function ld_${bit_width}{
    input{
                                                                             }
      bit_vector addr;
                                                                           }
    output{
                                                                          FHM DL FUNC
      bit_vector data_out = data_bus;
      bit addr_err = addr_err(addr, ac_mode);
                                                                              for (\hat{s}=\$wmode_1,\$w_b=\$bit_width-\$access; \$i>0; \$i--
                                                                          ,$w_b=$w_b-$access){
                                                                                  $str = $a_t_comma . &to_comp($i-1, $acmode_num) .
    control{
      in reset, req, rw, ac_mode;
                                                                          $a t comma;
      out ack;
                                                                                       print <<FHM_DL_FUNC
                                                                            /** store ${w_b} bits */
    protocol{
repeat [req == 1 && rw == 0 && ac_mode == ${b_str}]
until (ack == 1 || reset == 1);
    if (ack == 1){
                                                                            function s_${w_b}{
                                                                              input{
                                                                                bit vector addr;
        valid data_out;
                                                                                bit_vector data_in;
         valid addr_err;
     }
                                                                              output{
   }
                                                                               bit addr_err = addr_err(addr, ac_mode);
                                                                              control{
                                                                                in reset, req, rw, ac_mode;
FHM_DL_FUNC
                                                                                out ack;
    for ($i=$wmode_1,$w_b=$bit_width-$access; $i>0; $i--
, $w_b=$w_b-$access) {
                                                                              protocol{
        $str = $a_t_comma . &to_comp($i-1, $acmode_num) .
                                                                               repeat [req == 1 && rw == 1 && ac mode == $str] until
                                                                          (ack == 1 || reset == 1);
$a t comma;
                                                                                if (ack == 1) {
             print <<FHM_DL_FUNC
                                                                                  valid addr_err;
  /** load ${w_b} bits */
                                                                                }
  function ld_${w_b}{
                                                                             }
    input{
      bit_vector addr;
                                                                          FHM_DL_FUNC
    output{
      bit_vector data_out = data_bus;
bit_addr_err = addr_err(addr, ac_mode);
                                                                                  print <<FHM_DL_FUNC
                                                                            /** load : same as ld_${bit_width} */
      in reset, req, rw, ac_mode, ext_mode;
                                                                            function load{
      out ack;
                                                                              input{
                                                                               bit_vector addr;
    protocol{
     repeat [req == 1 && rw == 0 && ac_mode == $str &&
ext_mode == 1] until (ack == 1 || reset == 1);
                                                                                bit_vector data_out = data_bus;
     if (ack == 1) {
                                                                                bit addr_err = addr_err(addr, ac_mode);
         valid data_out;
         valid addr_err;
                                                                              control{
                                                                                in reset, req, rw, ac_mode;
     }
   }
                                                                                out ack;
                                                                              protocol{
                                                                         repeat [req == 1 && rw == 0 && ac_mode == ${b_str}]
until (ack == 1 || reset == 1);
    if (ack == 1){
  /** load ${w_b} bits (unsigned) */
  function ldu_${w_b}{
    input{
                                                                                  valid data_out;
      bit_vector addr;
                                                                                  valid addr_err;
    output{
      bit vector data out = data bus;
      bit addr_err = addr_err(addr, ac_mode);
                                                                            /** read : same as ld_${bit_width} */
      in reset, req, rw, ac_mode, ext_mode;
                                                                            function read{
      out ack;
                                                                              input{
                                                                                bit vector addr;
```

```
out ack;
    output{
      bit_vector data_out = data_bus;
      bit addr_err = addr_err(addr, ac_mode);
                                                                                repeat [req == 1 && rw == 0 && ac_mode == ${b_b_str}
                                                                          && ext_mode = 0] until (ack == 1 || reset == 1);
if (ack == 1) {
    control{
      in reset, req, rw, ac_mode;
out ack;
                                                                                   valid data_out;
                                                                                   valid addr err;
                                                                                }
    protocol{
      repeat [req == 1 && rw == 0 && ac_mode == ${b_str}]
until (ack == 1 || reset == 1);
if (ack == 1) {
                                                                            /** store : same as s_${bit_width} */
         valid data_out;
                                                                            function store{
                                                                              input{
         valid addr_err;
                                                                                bit_vector addr;
    }
                                                                                bit_vector data_in;
                                                                              output{
  /** lh : same as ld ${b h} */
                                                                                bit addr err = addr err(addr, ac mode);
  function lh{
    input{
      bit_vector addr;
                                                                                in reset, req, rw, ac_mode;
                                                                                out ack:
    output{
      bit vector data out = data bus;
                                                                              protocol{
      bit addr_err = addr_err(addr, ac_mode);
                                                                                repeat [req == 1 && rw == 1 && ac_mode == ${b_str}]
                                                                          until (ack == 1 || reset == 1);
    control {
                                                                               if (ack == 1) {
      in reset, req, rw, ac_mode, ext_mode;
                                                                                   valid addr_err;
      out ack;
                                                                                }
                                                                              }
      repeat [req == 1 && rw == 0 && ac_mode == ${b_h_str}
&& ext_mode = 1] until (ack == 1 || reset == 1);
if (ack == 1) {
                                                                            /** write : same as s_${bit_width} */
                                                                            function write(
         valid data out;
                                                                              input{
         valid addr_err;
                                                                                bit vector addr;
                                                                                bit_vector data_in;
    }
                                                                              output{
                                                                               bit addr_err = addr_err(addr, ac_mode);
  /** lhu : same as ldu_${b_h} */
  function lhu{
    input{
                                                                                in reset, req, rw, ac_mode;
      bit_vector addr;
                                                                                out ack;
                                                                              protocol{
    output{
                                                                                repeat [req == 1 && rw == 1 && ac mode == ${b str}]
      bit vector data out = data bus;
      bit addr_err = addr_err(addr, ac_mode);
                                                                          until (ack == 1 || reset == 1);
                                                                                if (ack == 1) {
    control (
                                                                                  valid addr_err;
                                                                                }
      in reset, req, rw, ac_mode, ext_mode;
      out ack;
                                                                              }
                                                                            /** sh : same as s_${b_h} */
      repeat [req == 1 && rw == 0 && ac_mode == ${b_h_str}
&& ext_mode = 0] until (ack == 1 || reset == 1);
if (ack == 1) {
                                                                            function sh{
                                                                              input{
         valid data out;
                                                                                bit vector addr:
         valid addr_err;
                                                                                bit_vector data_in;
      }
   }
                                                                              output{
                                                                               bit addr_err = addr_err(addr, ac_mode);
  /** lb : same as ld_${b_b} */
                                                                              control{
  function 1b{
                                                                                in reset, req, rw, ac_mode;
    input{
                                                                                out ack;
      bit_vector addr;
                                                                              protocol(
                                                                                repeat [req == 1 && rw == 1 && ac mode == ${b h str}]
    output{
                                                                          until (ack == 1 || reset == 1);
      bit_vector data_out = data_bus;
      bit addr_err = addr_err(addr, ac_mode);
                                                                               if (ack == 1) {
                                                                                   valid addr_err;
    control{
                                                                                }
      in reset, req, rw, ac_mode, ext_mode;
                                                                              }
      out ack;
                                                                            /** sb : same as s_${b_b} */
repeat [req == 1 && rw == 0 && ac_mode == ${b_b_str} && ext_mode = 1] until (ack == 1 || reset == 1);
    if (ack == 1) {
       valid data_out;
                                                                            function sb{
                                                                              input{
                                                                                bit_vector addr;
                                                                                bit_vector data_in;
         valid addr_err;
      }
    }
                                                                                bit addr err = addr err(addr, ac mode);
                                                                              control{
  /** lbu : same as ldu ${b b} */
                                                                                in reset, req, rw, ac mode;
  function lbu{
    input{
      bit_vector addr;
                                                                              protocol{
                                                                          repeat [req == 1 && rw == 1 && ac_mode == ${b_b_str}]
until (ack == 1 || reset == 1);
if (ack == 1)(
    output{
      bit_vector data_out = data_bus;
      bit addr_err = addr_err(addr, ac_mode);
                                                                                   valid addr_err;
    control(
                                                                            1
      in reset, req, rw, ac mode, ext mode;
```

```
for ($i=0; $i<=$n_write-1; $i++){
                                                                                   if ($i == $n_write - 1) {print
FHM_DL_FUNC
                                                                               "w_sel${i}[$n_sel_1:0];\n";}
    }
                                                                                   else{print "w_sel${i}[$n_sel_1:0], ";}
                                                                              print "
                                                                                         in ";
                                                                              for ($i=0; $i<=$n_write-1; $i++){
   if ($i == $n_write - 1){print
"data_in${i}[$w_1:0];\n";}
   else{print "data_in${i}[$w_1:0], ";}</pre>
         print <<FHM_DL_FUNC
  /** load : load data */
  function load{
    input{
      bit_vector addr;
                                                                              print "
                                                                                          in ":
                                                                              for ($i=0; $i<=$n_read-1; $i++){
    if ($i == $n_read - 1) {print
    "r_sel${i} [$n_sel_1:0]; \n";}
    output{
      bit_vector data_out = data_bus;
                                                                                   else{print "r_sel${i}[$n_sel_1:0], ";}
      in reset, req, rw;
                                                                              print "
      out ack;
                                                                                           out ":
                                                                              for ($i=0; $i<=$n_read-1; $i++) {
    if ($i == $n_read - 1) {print
    "data_out${i}[$w_1:0]; \n";}
    protocol{
      repeat [req == 1 && rw == 0] until (ack == 1 \mid \mid reset
                                                                                   else{print "data_out${i}[$w_1:0], ";}
      if (ack == 1) {
         valid data_out;
                                                                              print " }\n\n";
      }
    }
                                                                                print <<FHM DL NOP1
                                                                                 /** no operation */
                                                                                 function nop : idle{
  /** read : same as load */
                                                                                   control{
  function read{
                                                                                     in reset;
                                                                              FHM DL NOP1
    input{
      bit_vector addr;
                                                                               for ($i=0; $i<=$n_write-1; $i++){
                                                                                  print "
                                                                                                in w_enb$i;\n";
      bit_vector data_out = data_bus;
                                                                                   print <<FHM_DL_NOP2
    control{
      in reset, req, rw;
                                                                                   protocol{
                                                                              FHM_DL_NOP2
    protocol{
                                                                              print "
      repeat [req == 1 && rw == 0] until (ack == 1 || reset
                                                                                             [reset == '0' && ";
                                                                               for ($i=0; $i<=$n_write-1; $i++){
== 1);
                                                                                   if ($i == $n write - 1) {print "w_enb$i == '0'] {\n";} else{print "w_enb$i == '0' && ";}
      if (ack == 1) {
         valid data_out;
    }
                                                                                   print <<FHM DL NOP RESET
  /** store : store data */
  function store{
                                                                                 /** reset */
    input{
      bit_vector addr;
                                                                                 function reset : reset{
      bit_vector data_in;
                                                                                   control{
                                                                                     in reset;
      in reset, req, rw;
                                                                                  protocol{
                                                                                     [reset == '1']{
      out ack;
                                                                                     }
    protocol{
                                                                                  }
      repeat [req == 1 && rw == 1] until (ack == 1 || reset
                                                                              FHM_DL_NOP_RESET
                                                                               for ($i=0; $i<=$n_write-1; $i++){
  /** write : same as store */
  function write{
                                                                                       print <<FHM_DL_WRITE1
                                                                                 /** write$i */
    input{
      bit vector addr;
                                                                                 function write${i}{
      bit_vector data_in;
                                                                                   input{
                                                                                     bit_vector data_in$i;
    control{
      in reset, req, rw;
                                                                                   assignment{
      out ack;
                                                                              FHM_DL_WRITE1
                                                                                   for ($j=0; $j<=$n_reg-1; $j++){print "
    protocol{
                                                                                                                                    reg$j =
      repeat [req == 1 && rw == 1] until (ack == 1 || reset
                                                                               data_in$i;\n";}
                                                                                       print <<FHM_DL_WRITE2
                                                                                   control{
                                                                                     in w_enb$i;
                                                                                     in w sel$i;
                                                                                   protocol{
/** $bit_width-bit registerfile with $n_reg registers,
                                                                              FHM DL WRITE2
$n_read read port, $n_write $n_write port */
model regfile${bit_width}_${n_reg}_${n_read}_${n_write}{
                                                                                   for ($j=0; $j<=$n_reg-1; $j++){
                                                                                        port{
    clock clock;
    in reset;
                                                                               && hold data_in$i]{\n";
                                                                                      print "
FHM_DL_MODEL
                                                                                                        store reg$j;\n
                                                                                                                                }\n";
print "
          in ";
                                                                                   print " }\n }\n\n";
for ($i=0; $i<=$n_write-1; $i++){
    if ($i == $n write - 1) {print "w_enb$i;\n";}
else{print "w_enb$i, ";}
                                                                              for ($i=0; $i<=$n_read-1; $i++){
    print <<FHM_DL_READ
    /** read$i */</pre>
print " in ";
```

```
function read${i}{
                                                                                   valid result:
                                                                                   valid flag;
    input{
      bit_vector r_sel$i;
                                                                              }
    output{
      bit_vector data_out$i;
                                                                            /** csubu : unsigned clip subtract, flag(3):C, flag(2):Z,
    }
                                                                          flag(1):S, flag(0)=0 */
                                                                            function csubu{
FHM_DL_READ
                                                                              input{
                                                                                unsigned a, b;
print " priority{ ( reset > ( nop | ";
for ($i=0; $i<=$n_write-1; $i++) {
    if ($i == $n_write - 1) {print "write$i ) ), ";}</pre>
                                                                              output{
                                                                                unsigned
                                                                                           result = csubu(a, b);
    else{print "write$i | ";}
                                                                                bit_vector flag = alu_flag(mode, a, b);
                                                                              control(
for ($i=0; $i<=$n_read-1; $i++){
    if ($i == $n_read - 1) {print "read$i }\n}\n";}
else{print "read$i, ";}
                                                                                in mode:
                                                                                in cin;
}
                                                                              protocol{
                                                                                [mode == "00110" && cin == '1']{
ALU0
                                                                                   valid result;
                                                                                   valid flag;
unsigned
           addu(twoscomp a, twoscomp b);
                                                                                }
            subu(twoscomp a, twoscomp b);
                                                                              }
unsigned
            add(twoscomp a, twoscomp b);
unsigned
unsigned
            sub(twoscomp a, twoscomp b);
                                                                            /** csub : signed clip subtract, flag(3):C, flag(2):Z,
unsigned
            and(twoscomp a, twoscomp b);
           or(twoscomp a, twoscomp b);
xor(twoscomp a, twoscomp b);
unsigned
                                                                          flag(1):S, flag(0):V */
                                                                            function csub{
unsigned
unsigned
            nor(twoscomp a, twoscomp b);
                                                                              input{
            cmpu(twoscomp a, twoscomp b);
                                                                                unsigned a, b;
unsigned
unsigned
            cmp(twoscomp a, twoscomp b);
unsigned
            cmpzu(twoscomp a);
                                                                              output{
unsigned
            cmpz(twoscomp a);
                                                                                unsigned
                                                                                           result = csub(a, b);
                                                                                bit vector flag = alu flag(mode, a, b);
            inc(twoscomp a);
signed
unsigned
            incu(twoscomp a);
unsigned
            dec(twoscomp a);
                                                                              control {
unsigned
            cdec(twoscomp a);
                                                                                in mode:
unsigned
            caddu(twoscomp a, twoscomp b);
                                                                                in cin:
            cadd(twoscomp a, twoscomp b);
signed
unsigned
            csubu(twoscomp a, twoscomp b);
                                                                              protocol{
signed
            csub(twoscomp a, twoscomp b);
                                                                                 [mode == "01110" && cin == '1']{
bit_vector alu_flag(bit_vector mode, bit_vector a,
                                                                                   valid result;
bit_vector b, bit cin);
                                                                                   valid flag;
/** ${bit width}-th alu */
                                                                              }
model alu${bit width}{
       a[${bit_width_1}:0], b[${bit_width_1}:0], cin,
                                                                            /** addu : unsigned add, flag(3):C, flag(2):Z, flag(1):S,
                                                                          flag(0)=0 */
mode[4:01:
    out result[${bit width 1}:0], flag[3:0];
                                                                            function addu{
                                                                              input{
                                                                                unsigned a, b;
   /** C is '1' when (carry-occurred or not-bollowed) and
unsigned-mode else '0'
                                                                              output{
                                                                                unsigned result = addu(a, b);
bit_vector flag = alu_flag(mode, a, b);
       V is '1' when overflowed and signed-mode else '0' S is equal to MSB of result
       Z is '1' when result = 0 else '0' */
                                                                              control{
  /** caddu : unsigned clip add, flag(3):C, flag(2):Z,
flag(1):S, flag(0)=0 */
                                                                                in cin:
  function caddu{
    input{
                                                                              protocol{
      unsigned a, b;
                                                                                [mode == "00001" && cin == '0']{
                                                                                   valid result;
                                                                                   valid flag;
      unsigned
                 result = caddu(a, b);
                                                                                }
                                                                              }
      bit_vector flag = alu_flag(mode, a, b);
    control{
                                                                             /** subu : unsigned sub, flag(3):C, flag(2):Z, flag(1):S,
      in mode;
      in cin;
                                                                          flag(0)=0 */
                                                                            function subu{
    protocol{
                                                                              input{
      [mode == "00101" && cin == '0']{
                                                                                unsigned a, b;
         valid result;
         valid flag;
                                                                              output{
      }
                                                                                unsigned
                                                                                           result = subu(a, b, cin);
                                                                                bit_vector flag = alu_flag(mode, a, b, cin);
    }
                                                                              control{
  /** cadd : signed clip add, flag(3):C, flag(2):Z,
                                                                                in mode;
flag(1):S, flag(0):V */
                                                                                in cin;
  function cadd{
    input{
                                                                              protocol{
                                                                                 [mode == "00010" && cin == '1']{
      unsigned a, b;
                                                                                   valid result;
                                                                                   valid flag;
      unsigned result = cadd(a, b);
                                                                                }
      bit_vector flag = alu_flag(mode, a, b);
                                                                              }
    control{
                                                                            /** add : signed add, flag(3):C, flag(2):Z, flag(1):S,
      in mode;
                                                                          flag(0):V */
      in cin;
                                                                            function add{
    protocol{
                                                                              input{
      [mode == "01101" && cin == '0']{
                                                                                unsigned a, b;
```

```
}
    output{
      unsigned
                result = add(a, b);
      bit_vector flag = alu_flag(mode, a, b, cin);
                                                                         /** nor : nor, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0
    control{
                                                                         function nor{
      in mode:
                                                                           input{
      in cin;
                                                                            unsigned a, b;
   protocol{
                                                                          output{
      [mode == "01001" && cin == '0']{
                                                                            unsigned
                                                                                       result = nor(a, b);
        valid result;
                                                                            bit_vector flag = alu_flag(mode, a, b, cin);
        valid flag;
     }
                                                                          control{
   }
                                                                            in mode;
                                                                          protocol{
                                                                            [mode == "11000"] {
  /** sub : signed sub, flag(3):C, flag(2):Z, flag(1):S,
flag(0):V */
                                                                               valid result;
  function sub{
                                                                               valid flag;
    input{
     unsigned a, b;
   output{
                                                                         /** cmpu : unsigned comp, flag(3):C, flag(2):Z,
     unsigned
                result = sub(a, b);
      bit_vector flag = alu_flag(mode, a, b, cin);
                                                                      flag(1):S, flag(0)=0 */
                                                                         function cmpu{
    control(
                                                                          input{
      in mode:
                                                                            unsigned a, b;
      in cin:
                                                                          output{
   protocol{
                                                                            unsigned
                                                                                       result = cmpu(a, b);
      [mode == "01010" && cin == '1']{
                                                                            bit_vector flag = alu_flag(mode, a, b, cin);
        valid result;
        valid flag;
                                                                           control{
     }
                                                                            in mode;
   }
                                                                            in cin;
                                                                          protocol{
                                                                             [mode == "00010" && cin = '1']{
  /** and : and, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0
                                                                               valid result;
 function and{
                                                                               valid flag;
                                                                            }
    input{
      unsigned a, b;
    output{
                                                                      /** cmp : signed comp, flag(3):C, flag(2):Z, flag(1):S, flag(0):V */
     unsigned result = and(a, b);
     bit_vector flag = alu_flag(mode, a, b, cin);
                                                                         function cmp{
                                                                           input{
                                                                            unsigned a, b;
      in mode;
   protocol{
                                                                          output{
      [mode == "10010"]{
                                                                            unsigned result = cmp(a, b);
        valid result;
                                                                            bit_vector flag = alu_flag(mode, a, b, cin);
        valid flag;
     }
                                                                           control{
   }
                                                                            in mode;
                                                                            in cin;
  /** or : or, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0
                                                                          protocol{
                                                                             [mode == "01010" && cin = '1']{
  function or{
                                                                               valid result;
   input{
                                                                               valid flag;
     unsigned a, b;
                                                                            }
                                                                          }
    output{
      unsigned
                result = or(a, b);
     bit_vector flag = alu_flag(mode, a, b, cin);
                                                                         /** cmpzu : unsigned comp with zero, flag(3):C,
                                                                      flag(2):Z, flag(1):S, flag(0):0 */
function cmpzu{
   control{
     in mode;
                                                                          input{
                                                                            unsigned a;
   protocol{
      [mode == "10000"]{
    valid result;
                                                                           output{
                                                                                       result = cmpzu(a);
                                                                            unsigned
                                                                            bit_vector flag = alu_flag(mode, a, b, cin);
        valid flag;
                                                                           control{
                                                                            in mode;
                                                                            in cin;
  /** xor : xor, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0
                                                                          protocol{
                                                                             [mode == "00000" && cin = '0']{
  function xor{
                                                                               valid result;
    input{
      unsigned a, b;
                                                                               valid flag;
                                                                            }
                                                                          }
   output{
      unsigned
                result = xor(a, b);
     bit_vector flag = alu_flag(mode, a, b, cin);
                                                                         /** cmpz : signed comp with zero, flag(3):C, flag(2):Z,
    control{
                                                                       flag(1):S, flag(0):0 */
     in mode;
                                                                         function cmpz{
                                                                          input{
   protocol{
                                                                            unsigned a;
      [mode == "10001"] {
        valid result;
                                                                           output{
        valid flag;
                                                                            unsigned result = cmpz(a);
                                                                            bit_vector flag = alu_flag(mode, a, b, cin);
     }
```

```
}
    control{
      in mode;
      in cin;
                                                                      EXT0
   protocol{
                                                                      unsigned extz(unsigned data_in);
      [mode == "01000" && cin = '0']{
                                                                      unsigned exts(unsigned data in);
        valid result;
        valid flag;
                                                                      /** ${bit width}-bit extender : ${bit width}-bit to
                                                                      ${bit_width_out}-bit */
   }
                                                                      model extender${bit_width}_${bit_width_out}{
 1
                                                                        port{
                                                                         in data_in[$w:0], mode;
                                                                          out data_out[$w2:0];
  /** inc : inc, flag(3):C, flag(2):Z, flag(1):S, flag(0):V
  function inc{
                                                                        /** zero : zero extention */
    input{
     unsigned a;
                                                                        function zero{
                                                                          input{
    output{
                                                                            unsigned data in;
                result = inc(a);
      unsigned
      bit_vector flag = alu_flag(mode, a, b, cin);
                                                                          output{
                                                                           unsigned data_out = extz(a);
    control{
     in mode;
                                                                          control{
      in cin;
                                                                           in mode;
   protocol{
                                                                          protocol{
      [mode == "01000" && cin = '1']{
                                                                           [mode == '0']{
        valid result;
                                                                              valid data_out;
        valid flag;
                                                                            }
   }
                                                                        /** sign : sign extention */
  /** incu : unsigned inc, flag(3):C, flag(2):Z, flag(1):S,
                                                                        function sign{
flag(0):0 */
                                                                          input{
  function incu{
                                                                            unsigned data in;
    input{
      unsigned a;
                                                                          output{
                                                                            unsigned data_out = exts(a);
   output{
     unsigned
                result = incu(a);
                                                                          control{
     bit_vector flag = alu_flag(mode, a, b, cin);
                                                                           in mode;
    control{
                                                                          protocol{
                                                                            [mode == '1']{
      in mode:
                                                                              valid data_out;
      in cin;
      [mode == "00000" && cin = '1']{
        valid result;
        valid flag;
                                                                      MUL0
     }
   }
                                                                      DIV0
  /** dec : unsigned dec, flag(3):C, flag(2):Z, flag(1):S,
flag(0):0 */
 function dec{
                                                                      SFT0
    input{
      unsigned a;
                                                                      unsigned shift(unsigned data_in, unsigned mode);\n\n";
                                                                      /** ${bit_width}-bit shifter : ${info} */
   output{
                                                                      model shifter_var{
      unsigned
                result = dec(a);
                                                                        port{
      bit_vector flag = alu_flag(mode, a, b, cin);
                                                                          in data_in[${bit_width_1}:0], mode,
                                                                      ctrl[${ctrl_width_1}:0];
    control{
                                                                          out data_out[${bit_width_1}:0];
      in mode;
      in cin:
                                                                        /** shift left logical */
   protocol{
                                                                        function sll{
      [mode == "00011" && cin = '0']{
                                                                          input{
        valid result;
                                                                            unsigned data_in;
        valid flag;
                                                                            unsigned ctrl;
     }
   }
                                                                          output{
                                                                           unsigned data_out = ${func}(${func_par});
  /** cdec : unsigned dec(clip), flag(3):C, flag(2):Z,
                                                                          control{
flag(1):S, flag(0):0 */
                                                                           unsigned mode;
  function cdec{
   input{
                                                                          protocol{
                                                                            [mode == "00"]{
      unsigned a;
                                                                              valid data_out;
    output{
                                                                            }
      unsigned
                result = cdec(a);
                                                                         }
      bit_vector flag = alu_flag(mode, a, b, cin);
                                                                        /** shift left arithmetic */
      in mode;
                                                                        function sla{
      in cin;
                                                                          input{
                                                                            unsigned data in;
                                                                            unsigned ctrl;
   protocol{
      [mode == "00111" && cin = '0']{
        valid result;
        valid flag;
                                                                           unsigned data_out = ${func}(${func_par});
    1
                                                                          control{
```

```
unsigned mode;
    protocol{
       [mode == "01"]{
         valid data_out;
   }
  /** shift right logical */
  function srl{
    input{
      unsigned data_in;
unsigned ctrl;
      unsigned data_out = ${func}(${func_par});
    control{
      unsigned mode;
    protocol{
  [mode == "10"]{
    valid data_out;
}
   }
  /** shift right arithmetic */
  function sra{
    input{
      unsigned data_in;
unsigned ctrl;
    output{
      unsigned data_out = ${func}(${func_par});
    control{
     unsigned mode;
    protocol{
  [mode == "11"]{
         valid data_out;
 }
EXT1
```