**PC**

**=========**

**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];**

**}**

**storage{**

**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;**

**}**

**protocol{**

**[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;**

**}**

**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**

**=======**

**/\*\* ${bit\_width}-bit register \*/**

**model reg${bit\_width}{**

**port{**

**clock clock;**

**in reset, enb;**

**in data\_in${range};**

**out data\_out${range};**

**}**

**storage{**

**register reg${range};**

**}**

**default\_control{**

**reset = 0;**

**enb = 0;**

**}**

**/\*\* 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 }**

**}**

**IMAU**

**===========**

**/\*\* ${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;**

**}**

**}**

**}**

**DMAU**

**===========**

**/\*\* ${bit\_width}-bit data memory access unit \*/**

**model dmau${bit\_width}{**

**port{**

**in reset;**

**in 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}];**

**in ack\_bus;**

**out ack, data\_out[${b\_range}];**

**}**

**default\_control{**

**reset = 0;**

**req = 0;**

**}**

**/\*\* no operation \*/**

**function nop : idle{**

**control{**

**in reset, req;**

**}**

**protocol{**

**[reset ==0 && req == 0]{**

**}**

**}**

**}**

**/\*\* reset \*/**

**function reset : reset{**

**assignment{**

**reg = 0;**

**}**

**control{**

**in reset;**

**}**

**protocol{**

**[reset == 0]{**

**store reg;**

**}**

**}**

**}**

**FHM\_DL\_FUNC**

**}**

**if ($acmode\_num != 0){**

**{**

**print <<FHM\_DL\_FUNC**

**/\*\* load ${bit\_width} bits \*/**

**function ld\_${bit\_width}{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**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){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**FHM\_DL\_FUNC**

**}**

**for ($i=$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) . $a\_t\_comma;**

**{**

**print <<FHM\_DL\_FUNC**

**/\*\* load ${w\_b} bits \*/**

**function ld\_${w\_b}{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode, ext\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0 && ac\_mode == $str && ext\_mode == 1] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* load ${w\_b} bits (unsigned) \*/**

**function ldu\_${w\_b}{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode, ext\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0 && ac\_mode == $str && ext\_mode == 0] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**FHM\_DL\_FUNC**

**}**

**}**

**{**

**print <<FHM\_DL\_FUNC**

**/\*\* store ${bit\_width} bits \*/**

**function s\_${bit\_width}{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**output{**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1 && ac\_mode == ${b\_str}] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid addr\_err;**

**}**

**}**

**}**

**FHM\_DL\_FUNC**

**}**

**for ($i=$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) . $a\_t\_comma;**

**{**

**print <<FHM\_DL\_FUNC**

**/\*\* store ${w\_b} bits \*/**

**function s\_${w\_b}{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**output{**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1 && ac\_mode == $str] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid addr\_err;**

**}**

**}**

**}**

**FHM\_DL\_FUNC**

**}**

**}**

**{**

**print <<FHM\_DL\_FUNC**

**/\*\* load : same as ld\_${bit\_width} \*/**

**function load{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**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){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* read : same as ld\_${bit\_width} \*/**

**function read{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**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){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* lh : same as ld\_${b\_h} \*/**

**function lh{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode, ext\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0 && ac\_mode == ${b\_h\_str} && ext\_mode = 1] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* lhu : same as ldu\_${b\_h} \*/**

**function lhu{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode, ext\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0 && ac\_mode == ${b\_h\_str} && ext\_mode = 0] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* lb : same as ld\_${b\_b} \*/**

**function lb{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode, ext\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0 && ac\_mode == ${b\_b\_str} && ext\_mode = 1] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* lbu : same as ldu\_${b\_b} \*/**

**function lbu{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode, ext\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0 && ac\_mode == ${b\_b\_str} && ext\_mode = 0] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* store : same as s\_${bit\_width} \*/**

**function store{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**output{**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1 && ac\_mode == ${b\_str}] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* write : same as s\_${bit\_width} \*/**

**function write{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**output{**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1 && ac\_mode == ${b\_str}] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* sh : same as s\_${b\_h} \*/**

**function sh{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**output{**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1 && ac\_mode == ${b\_h\_str}] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid addr\_err;**

**}**

**}**

**}**

**/\*\* sb : same as s\_${b\_b} \*/**

**function sb{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**output{**

**bit addr\_err = addr\_err(addr, ac\_mode);**

**}**

**control{**

**in reset, req, rw, ac\_mode;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1 && ac\_mode == ${b\_b\_str}] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid addr\_err;**

**}**

**}**

**}**

**}**

**FHM\_DL\_FUNC**

**}**

**}**

**else{**

**{**

**print <<FHM\_DL\_FUNC**

**/\*\* load : load data \*/**

**function load{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**}**

**control{**

**in reset, req, rw;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**}**

**}**

**}**

**/\*\* read : same as load \*/**

**function read{**

**input{**

**bit\_vector addr;**

**}**

**output{**

**bit\_vector data\_out = data\_bus;**

**}**

**control{**

**in reset, req, rw;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 0] until (ack == 1 || reset == 1);**

**if (ack == 1){**

**valid data\_out;**

**}**

**}**

**}**

**/\*\* store : store data \*/**

**function store{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**control{**

**in reset, req, rw;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1] until (ack == 1 || reset == 1);**

**}**

**}**

**/\*\* write : same as store \*/**

**function write{**

**input{**

**bit\_vector addr;**

**bit\_vector data\_in;**

**}**

**control{**

**in reset, req, rw;**

**out ack;**

**}**

**protocol{**

**repeat [req == 1 && rw == 1] until (ack == 1 || reset == 1);**

**}**

**}**

**}**

**GPR**

**===========**

**/\*\* $bit\_width-bit registerfile with $n\_reg registers, $n\_read read port, $n\_write $n\_write port \*/**

**model regfile${bit\_width}\_${n\_reg}\_${n\_read}\_${n\_write}{**

**port{**

**clock clock;**

**in reset;**

**FHM\_DL\_MODEL**

**}**

**print " in ";**

**for ($i=0; $i<=$n\_write-1; $i++){**

**if ($i == $n\_write - 1){print "w\_enb$i;\n";}**

**else{print "w\_enb$i, ";}**

**}**

**print " in ";**

**for ($i=0; $i<=$n\_write-1; $i++){**

**if ($i == $n\_write - 1){print "w\_sel${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], ";}**

**}**

**print " in ";**

**for ($i=0; $i<=$n\_read-1; $i++){**

**if ($i == $n\_read - 1){print "r\_sel${i}[$n\_sel\_1:0];\n";}**

**else{print "r\_sel${i}[$n\_sel\_1:0], ";}**

**}**

**print " out ";**

**for ($i=0; $i<=$n\_read-1; $i++){**

**if ($i == $n\_read - 1){print "data\_out${i}[$w\_1:0];\n";}**

**else{print "data\_out${i}[$w\_1:0], ";}**

**}**

**print " }\n\n";**

**{**

**print <<FHM\_DL\_NOP1**

**/\*\* no operation \*/**

**function nop : idle{**

**control{**

**in reset;**

**FHM\_DL\_NOP1**

**}**

**for ($i=0; $i<=$n\_write-1; $i++){**

**print " in w\_enb$i;\n";**

**}**

**{**

**print <<FHM\_DL\_NOP2**

**}**

**protocol{**

**FHM\_DL\_NOP2**

**}**

**print " [reset == '0' && ";**

**for ($i=0; $i<=$n\_write-1; $i++){**

**if ($i == $n\_write - 1){print "w\_enb$i == '0']{\n";}**

**else{print "w\_enb$i == '0' && ";}**

**}**

**{**

**print <<FHM\_DL\_NOP\_RESET**

**}**

**}**

**/\*\* reset \*/**

**function reset : reset{**

**control{**

**in reset;**

**}**

**protocol{**

**[reset == '1']{**

**}**

**}**

**}**

**FHM\_DL\_NOP\_RESET**

**}**

**for ($i=0; $i<=$n\_write-1; $i++){**

**{**

**print <<FHM\_DL\_WRITE1**

**/\*\* write$i \*/**

**function write${i}{**

**input{**

**bit\_vector data\_in$i;**

**}**

**assignment{**

**FHM\_DL\_WRITE1**

**}**

**for ($j=0; $j<=$n\_reg-1; $j++){print " reg$j = data\_in$i;\n";}**

**{**

**print <<FHM\_DL\_WRITE2**

**}**

**control{**

**in w\_enb$i;**

**in w\_sel$i;**

**}**

**protocol{**

**FHM\_DL\_WRITE2**

**}**

**for ($j=0; $j<=$n\_reg-1; $j++){**

**$j2 = &to\_comp($j, $n\_sel);**

**print " [w\_enb$i == '1' && w\_sel$i == \"$j2\" && hold data\_in$i]{\n";**

**print " store reg$j;\n }\n";**

**}**

**print " }\n }\n\n";**

**}**

**for ($i=0; $i<=$n\_read-1; $i++){**

**print <<FHM\_DL\_READ**

**/\*\* read$i \*/**

**function read${i}{**

**input{**

**bit\_vector r\_sel$i;**

**}**

**output{**

**bit\_vector data\_out$i;**

**}**

**}**

**FHM\_DL\_READ**

**}**

**print " priority{ ( reset > ( nop | ";**

**for ($i=0; $i<=$n\_write-1; $i++){**

**if ($i == $n\_write - 1){print "write$i ) ), ";}**

**else{print "write$i | ";}**

**}**

**for ($i=0; $i<=$n\_read-1; $i++){**

**if ($i == $n\_read - 1){print "read$i }\n}\n";}**

**else{print "read$i, ";}**

**}**

**ALU0**

**==========**

**unsigned addu(twoscomp a, twoscomp b);**

**unsigned subu(twoscomp a, twoscomp b);**

**unsigned add(twoscomp a, twoscomp b);**

**unsigned sub(twoscomp a, twoscomp b);**

**unsigned and(twoscomp a, twoscomp b);**

**unsigned or(twoscomp a, twoscomp b);**

**unsigned xor(twoscomp a, twoscomp b);**

**unsigned nor(twoscomp a, twoscomp b);**

**unsigned cmpu(twoscomp a, twoscomp b);**

**unsigned cmp(twoscomp a, twoscomp b);**

**unsigned cmpzu(twoscomp a);**

**unsigned cmpz(twoscomp a);**

**signed inc(twoscomp a);**

**unsigned incu(twoscomp a);**

**unsigned dec(twoscomp a);**

**unsigned cdec(twoscomp a);**

**unsigned caddu(twoscomp a, twoscomp b);**

**signed cadd(twoscomp a, twoscomp b);**

**unsigned csubu(twoscomp a, twoscomp b);**

**signed csub(twoscomp a, twoscomp b);**

**bit\_vector alu\_flag(bit\_vector mode, bit\_vector a, bit\_vector b, bit cin);**

**/\*\* ${bit\_width}-th alu \*/**

**model alu${bit\_width}{**

**port{**

**in a[${bit\_width\_1}:0], b[${bit\_width\_1}:0], cin, mode[4:0];**

**out result[${bit\_width\_1}:0], flag[3:0];**

**}**

**/\*\* C is '1' when (carry-occurred or not-bollowed) and unsigned-mode else '0'**

**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' \*/**

**/\*\* caddu : unsigned clip add, flag(3):C, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function caddu{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = caddu(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00101" && cin == '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* cadd : signed clip add, flag(3):C, flag(2):Z, flag(1):S, flag(0):V \*/**

**function cadd{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = cadd(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01101" && cin == '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* csubu : unsigned clip subtract, flag(3):C, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function csubu{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = csubu(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00110" && cin == '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* csub : signed clip subtract, flag(3):C, flag(2):Z, flag(1):S, flag(0):V \*/**

**function csub{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = csub(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01110" && cin == '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* addu : unsigned add, flag(3):C, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function addu{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = addu(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00001" && cin == '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* subu : unsigned sub, flag(3):C, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function subu{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = subu(a, b, cin);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00010" && cin == '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* add : signed add, flag(3):C, flag(2):Z, flag(1):S, flag(0):V \*/**

**function add{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = add(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01001" && cin == '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* sub : signed sub, flag(3):C, flag(2):Z, flag(1):S, flag(0):V \*/**

**function sub{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = sub(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01010" && cin == '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* and : and, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function and{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = and(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**}**

**protocol{**

**[mode == "10010"]{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* or : or, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function or{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = or(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**}**

**protocol{**

**[mode == "10000"]{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* xor : xor, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function xor{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = xor(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**}**

**protocol{**

**[mode == "10001"]{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* nor : nor, flag(3)=0, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function nor{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = nor(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**}**

**protocol{**

**[mode == "11000"]{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* cmpu : unsigned comp, flag(3):C, flag(2):Z, flag(1):S, flag(0)=0 \*/**

**function cmpu{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = cmpu(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00010" && cin = '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* cmp : signed comp, flag(3):C, flag(2):Z, flag(1):S, flag(0):V \*/**

**function cmp{**

**input{**

**unsigned a, b;**

**}**

**output{**

**unsigned result = cmp(a, b);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01010" && cin = '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* cmpzu : unsigned comp with zero, flag(3):C, flag(2):Z, flag(1):S, flag(0):0 \*/**

**function cmpzu{**

**input{**

**unsigned a;**

**}**

**output{**

**unsigned result = cmpzu(a);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00000" && cin = '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* cmpz : signed comp with zero, flag(3):C, flag(2):Z, flag(1):S, flag(0):0 \*/**

**function cmpz{**

**input{**

**unsigned a;**

**}**

**output{**

**unsigned result = cmpz(a);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01000" && cin = '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* inc : inc, flag(3):C, flag(2):Z, flag(1):S, flag(0):V \*/**

**function inc{**

**input{**

**unsigned a;**

**}**

**output{**

**unsigned result = inc(a);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "01000" && cin = '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* incu : unsigned inc, flag(3):C, flag(2):Z, flag(1):S, flag(0):0 \*/**

**function incu{**

**input{**

**unsigned a;**

**}**

**output{**

**unsigned result = incu(a);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00000" && cin = '1']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* dec : unsigned dec, flag(3):C, flag(2):Z, flag(1):S, flag(0):0 \*/**

**function dec{**

**input{**

**unsigned a;**

**}**

**output{**

**unsigned result = dec(a);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00011" && cin = '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**/\*\* cdec : unsigned dec(clip), flag(3):C, flag(2):Z, flag(1):S, flag(0):0 \*/**

**function cdec{**

**input{**

**unsigned a;**

**}**

**output{**

**unsigned result = cdec(a);**

**bit\_vector flag = alu\_flag(mode, a, b, cin);**

**}**

**control{**

**in mode;**

**in cin;**

**}**

**protocol{**

**[mode == "00111" && cin = '0']{**

**valid result;**

**valid flag;**

**}**

**}**

**}**

**}**

**EXT0**

**======**

**unsigned extz(unsigned data\_in);**

**unsigned exts(unsigned data\_in);**

**/\*\* ${bit\_width}-bit extender : ${bit\_width}-bit to ${bit\_width\_out}-bit \*/**

**model extender${bit\_width}\_${bit\_width\_out}{**

**port{**

**in data\_in[$w:0], mode;**

**out data\_out[$w2:0];**

**}**

**/\*\* zero : zero extention \*/**

**function zero{**

**input{**

**unsigned data\_in;**

**}**

**output{**

**unsigned data\_out = extz(a);**

**}**

**control{**

**in mode;**

**}**

**protocol{**

**[mode == '0']{**

**valid data\_out;**

**}**

**}**

**}**

**/\*\* sign : sign extention \*/**

**function sign{**

**input{**

**unsigned data\_in;**

**}**

**output{**

**unsigned data\_out = exts(a);**

**}**

**control{**

**in mode;**

**}**

**protocol{**

**[mode == '1']{**

**valid data\_out;**

**}**

**}**

**}**

**}**

**MUL0**

**=======**

**DIV0**

**=======**

**SFT0**

**=======**

**unsigned shift(unsigned data\_in, unsigned mode);\n\n";**

**/\*\* ${bit\_width}-bit shifter : ${info} \*/**

**model shifter\_var{**

**port{**

**in data\_in[${bit\_width\_1}:0], mode, ctrl[${ctrl\_width\_1}:0];**

**out data\_out[${bit\_width\_1}:0];**

**}**

**/\*\* shift left logical \*/**

**function sll{**

**input{**

**unsigned data\_in;**

**unsigned ctrl;**

**}**

**output{**

**unsigned data\_out = ${func}(${func\_par});**

**}**

**control{**

**unsigned mode;**

**}**

**protocol{**

**[mode == "00"]{**

**valid data\_out;**

**}**

**}**

**}**

**/\*\* shift left arithmetic \*/**

**function sla{**

**input{**

**unsigned data\_in;**

**unsigned ctrl;**

**}**

**output{**

**unsigned data\_out = ${func}(${func\_par});**

**}**

**control{**

**unsigned mode;**

**}**

**protocol{**

**[mode == "01"]{**

**valid data\_out;**

**}**

**}**

**}**

**/\*\* shift right logical \*/**

**function srl{**

**input{**

**unsigned data\_in;**

**unsigned ctrl;**

**}**

**output{**

**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**

**=======**