RISC0.v

```
`default_nettype none
module top(  // Lattice iCE40 version PDR 7.11.19 / 19.11.19
  // adapted from NW 27.5.09  LL 10.12.09  NW 28.7.2011
  input wire OSCIN, rstBtn, output reg LED,
  input wire MISO, output wire MOSI, SCLK, output wire [1:0] SS,
  inout wire [3:0] gpio);

reg [31:0] cnt1;  // milliseconds
reg [15:0] cnt0;  // clks
reg [3:0] gpo, gpoc;  // gpio out & output ctrl
reg [3:0] spiCtrl;
reg rst, clk50, clk;

wire [31:0] inbus, outbus;
wire [15:0] spiRx;
wire [5:0] ioadr;  // I/O port addr (byte)
wire [3:0] iowadr;  // I/O port addr (word)
wire [3:0] gpi;  // gpio in
wire iord, iowr, spiRdy, sclko, misoi, cnt0Limit;

RISC0 riscx(.clk(clk), .rst(rst), .iord(iord), .iowr(iowr),
   .ioadr(ioadr), .inbus(inbus), .outbus(outbus));
SPI spi(.clk(clk), .rst(rst), .rdy(spiRdy), .wide(spiCtrl[2]),
  .start(iowr & (iowadr ==
4)), .dataTx(outbus[15:0]), .dataRx(spiRx),
  .MISO(misoi), .MOSI(MOSI), .SCLK(sclko));

IOBUFR
misobuf(.clk(clk), .en(1'b0), .out(1'b0), .in(misoi),.pin(MISO));
genvar i;
generate for (i = 0; i < 4; i = i+1)
  IOBUFR gpiobuf(.clk(clk), .en(gpoc[i]),
    .out(gpo[i]), .in(gpi[i]), .pin(gpio[i]));
endgenerate

assign cnt0Limit = (cnt0 == 24999);
assign iowadr = ioadr[5:2];
assign inbus = (iowadr == 0) ? cnt1 :
    (iowadr == 4) ? {16'b0, spiRx} :
    (iowadr == 5) ? {30'b0, misoi, spiRdy} :
    (iowadr == 8) ? {28'b0, gpi} : 0;
assign SCLK = spiCtrl[3] ^ sclko;
assign SS = ~spiCtrl[1:0];  // active-low slave selects

always @(posedge clk) begin
  cnt0 <= cnt0Limit ? 0 : cnt0 + 1;
  cnt1 <= cnt0Limit ? cnt1 + 1 : cnt1;
  rst <= cnt0Limit & (cnt1[3:0] == 0) ? rstBtn : rst;
  LED <= ~rst ? 0 : (iowr & (iowadr == 1)) ? outbus[0] : LED;
  spiCtrl <= ~rst ? 0 : (iowr & (iowadr == 5)) ? outbus[3:0] :
spiCtrl;
  gpo <= ~rst ? 0 : (iowr & (iowadr == 8)) ? outbus[3:0] : gpo;
  gpoc <= ~rst ? 0 : (iowr & (iowadr == 9)) ? outbus[3:0] : gpoc;
end

always @ (posedge OSCIN) clk50 <= ~clk50;
always @ (posedge clk50) clk <= ~clk;
endmodule
//-------------------------------------------------------------------
module IOBUFR(  // 9.11.19 PDR
// iCE40 native registered-input tristate I/O buffer
  input wire clk, input wire en, out,
  output wire in, inout wire pin);

SB_IO #(.PULLUP(1'b1),  // weak pullup enabled
    .PIN_TYPE(6'b1010_00))  // PIN_OUTPUT_TRISTATE,
PIN_INPUT_REGISTERED
  iobuf(.INPUT_CLK(clk), .OUTPUT_CLK(clk), .OUTPUT_ENABLE(en),
    .D_OUT_0(out), .D_IN_0(in), .PACKAGE_PIN(pin));
endmodule
//-------------------------------------------------------------------
module RISC0(  // 8.11.19 PDR
  // adapted from NW 8.10.12  rev. 26.12.2013
  input wire clk, rst, output wire iord, iowr, input wire [31:0]
inbus,
  output wire [5:0] ioadr, output wire [31:0] outbus);

reg [31:0] R [0:15];  // array of 16 registers
reg [31:0] H;  // aux register
reg [11:0] PC;
```

RISC0.v

```
reg N, Z, C, OV;  // condition flags
reg stall1;

wire [63:0] product;
wire [32:0] aluRes;
wire [31:0] IR, A, B, C0, C1, regmux, dmin, dmout;
wire [31:0] s1, s2, s3, t1, t2, t3, quotient, remainder;
wire [15:0] imm;
wire [13:0] off, dmadr;
wire [11:0] pcmux, nxpc;
wire [3:0] op, ira, ira0, irb, irc;
wire [2:0] cc;
wire [1:0] sc1, sc0;  // shift counts
wire p, q, u, v, w;  // instruction fields
wire MOV, LSL, ASR, ROR, AND, ANN, IOR, XOR;  // operation
signals
wire ADD, SUB, MUL, DIV, LDR, STR, BR;
wire cond, S, sa, sb, sc, regwr, dmwr, ioenb;
wire stall, stallL, stallM, stallD;

ROM PM (.clk(clk), .adr(pcmux[10:0]), .data(IR));
RAM DM
(.clk(clk), .we(dmwr), .adr(dmadr[12:2]), .din(dmin), .dout(dmout
));
Multiplier mulUnit (.clk(clk), .run(MUL), .stall(stallM),
   .u(~u), .x(B), .y(C1), .z(product));
Divider divUnit (.clk(clk), .run(DIV), .stall(stallD),
   .u(~u), .x(B), .y(C1), .quot(quotient), .rem(remainder));

// decoding
assign p = IR[31], q = IR[30], u = IR[29], v = IR[28];
assign w = IR[16];
assign cc  = IR[26:24];
assign ira = IR[27:24], irb = IR[23:20], irc = IR[3:0];
assign op  = IR[19:16];
assign imm = IR[15:0];
assign off = IR[13:0];  //[19:14] not used
assign MOV = ~p & (op == 0), LSL = ~p & (op == 1), ASR = ~p & (op
== 2),
  ROR = ~p & (op == 3), AND = ~p & (op == 4), ANN = ~p & (op ==
5),
  IOR = ~p & (op == 6), XOR = ~p & (op == 7), ADD = ~p & (op ==
8),
  SUB = ~p & (op == 9), MUL = ~p & (op == 10), DIV = ~p & (op ==
11);
assign LDR = p & ~q & ~u, STR = p & ~q & u;
assign BR = p&q;
assign A = R[ira0], B = R[irb], C0 = R[irc];  // register data
signals
assign ira0 = BR ? 15 : ira;
assign C1 = ~q ? C0 : {{16{v}}, imm};
assign dmadr = B[13:0] + off;
assign dmwr = STR & ~stall;
assign dmin = A;
assign ioenb = (dmadr[13:6] == 8'b11111111);
assign iowr = STR & ioenb, iord = LDR & ioenb;
assign ioadr = dmadr[5:0];
assign outbus = A;

assign sc0 = C1[1:0];  // Arithmetic-logical unit (ALU)
assign sc1 = C1[3:2];  // shifter for ASR and ROR
assign s1 = (sc0 == 3) ? {(w ? B[2:0] : {3{B[31]}}), B[31:3]} :
    (sc0 == 2) ? {(w ? B[1:0] : {2{B[31]}}), B[31:2]} :
    (sc0 == 1) ? {(w ? B[0] : B[31]), B[31:1]} : B;
assign s2 = (sc1 == 3) ? {(w ? s1[11:0] : {12{s1[31]}}),
s1[31:12]} :
    (sc1 == 2) ? {(w ? s1[7:0] : {8{s1[31]}}), s1[31:8]} :
    (sc1 == 1) ? {(w ? s1[3:0] : {4{s1[31]}}), s1[31:4]} : s1;
assign s3 = C1[4] ? {(w ? s2[15:0] : {16{s2[31]}}), s2[31:16]} :
s2;
assign t1 = (sc0 == 3) ? {B[28:0], 3'b0} :  // shifter for LSL
    (sc0 == 2) ? {B[29:0], 2'b0} :
    (sc0 == 1) ? {B[30:0], 1'b0} : B;
assign t2 = (sc1 == 3) ? {t1[19:0], 12'b0} :
    (sc1 == 2) ? {t1[23:0], 8'b0} :
    (sc1 == 1) ? {t1[27:0], 4'b0} : t1;
assign t3 = C1[4] ? {t2[15:0], 16'b0} : t2;
assign aluRes =
  MOV ? (q ?
```

```
    (~u ? {{16{v}}, imm} : {imm, 16'b0}) :
    (~u ? C0 : (~v ? H : {N, Z, C, OV, 20'b0, 8'b10100000}))) :
  LSL ? t3 : (ASR|ROR) ? s3 :
  AND ? B & C1 : ANN ? B & ~C1 : IOR ?B|C1: XOR ? B ^ C1 :
  ADD ? B + C1 + (u & C) : SUB ? B - C1 - (u & C) :
  MUL ? product[31:0] : DIV ? quotient : 0;
assign regwr = ~p & ~stall | (LDR & stall1)| (BR & cond & v) ;
assign regmux =
  (LDR & ~ioenb) ? dmout :
  (LDR & ioenb) ? inbus :
  (BR & v) ? {18'b0, nxpc, 2'b0} : aluRes;
assign S = N ^ OV;
assign nxpc = PC + 1;
assign cond = IR[27] ^
  ((cc == 0) & N | // MI, PL
   (cc == 1) & Z | // EQ, NE
   (cc == 2) & C | // CS, CC
   (cc == 3) & OV | // VS, VC
   (cc == 4) & (C|Z) | // LS, HI
   (cc == 5) & S | // LT, GE
   (cc == 6) & (S|Z) | // LE, GT
   (cc == 7)); // T, F
assign pcmux = (~rst) ? 0 : stall ? PC :
  (BR & cond & u) ? off[11:0] + nxpc :
  (BR & cond & ~u) ? C0[13:2] : nxpc;
assign sa = aluRes[31];
assign sb = B[31];
assign sc = C1[31] ^ SUB;
assign stall = stallL | stallM | stallD;
assign stallL = LDR & ~stall1;

always @ (posedge clk) begin
  PC <= pcmux;
  stall1 <= stallL;
  R[ira0] <= regwr ? regmux : A;
  N <= regwr ? regmux[31] : N;
  Z <= regwr ? (regmux[31:0] == 0) : Z;
  C <= (ADD|SUB) ? aluRes[32] : C;
  OV <= (ADD|SUB) ? (sa & ~sb & ~sc | ~sa & sb & sc) : OV;
  H <= MUL ? product[63:32] : DIV ? remainder : H;
end

endmodule
//-------------------------------------------------------------------
module Multiplier(  // NW 14.9.2015
  input wire clk, run, u, input wire [31:0] x, y,
  output wire stall, output wire [63:0] z);

reg [63:0] P;   // product
reg [5:0] S;    // state
wire [32:0] w1;
wire [31:0] w0;
assign stall = run & ~(S == 33);
assign w0 = P[0] ? y : 0;
assign w1 = (S == 32) & u ? {P[63], P[63:32]} - {w0[31], w0} :
       {P[63], P[63:32]} + {w0[31], w0};
assign z = P;
always @ (posedge clk) begin
  P <= (S == 0) ? {32'b0, x} : {w1[32:0], P[31:1]};
  S <= run ? S+1 : 0;
end
endmodule
//-------------------------------------------------------------
module Divider(  // NW 20.9.2015 // PR 9.11.19
  input wire clk, run, u, input wire [31:0] x, y,  // y>0
  output wire stall, output reg [31:0] quot, rem);
reg [63:0] RQ;
reg [5:0] S;  // state
wire [63:0] RQ0;
wire [31:0] x0, w0, w1;
wire sign;
assign stall = run & ~(S == 33);
assign sign = x[31] & u;
assign x0 = sign ? -x : x;
assign w0 = RQ[62: 31];
assign w1 = w0 - y;
assign RQ0 = (S == 0) ? {32'b0, x0}
  : {(w1[31] ? w0 : w1), RQ[30:0], ~w1[31]};

always @ (posedge clk) begin
```

RISC0.v

```
  RQ <= RQ0;
  S <= run ? S+1 : 0;
  quot <= ~sign ? RQ0[31:0] :
   (RQ0[63:32] == 0) ? -RQ0[31:0] : -RQ0[31:0] - 1;
  rem <= ~sign ? RQ0[63:32] : (RQ0[63:32] == 0) ? 0 : y -
RQ0[63:32];
end
endmodule
//-----------------------------------------------------------------
module SPI(  // PDR 7.11.19 / 12.11.19
  input wire clk, rst, start, wide, output reg rdy,
  input wire [15:0] dataTx, output wire [15:0] dataRx,
  input MISO, output MOSI, output SCLK);

reg [15:0] shreg;
reg [7:0] tick;
reg [3:0] bitcnt;
wire endbit, endtick;

assign endtick = (tick == 249);  // 25MHz clk / 250 = 100Kbps
assign endbit = (bitcnt[2:0] == 7) & (~wide | bitcnt[3]);
assign dataRx = {wide ? shreg[15:8] : 8'b0, shreg[7:0]};
assign MOSI = (~rst | rdy) ? 1 : wide ? shreg[15] : shreg[7];
assign SCLK = (~rst | rdy) ? 0 : tick[7];

always @ (posedge clk) begin
  tick <= (~rst | rdy | endtick) ? 0 : tick + 1;
  rdy <= (~rst | endtick & endbit) ? 1 : start ? 0 : rdy;
  bitcnt <= (~rst | start) ? 0 : (endtick & ~endbit) ? bitcnt + 1
: bitcnt;
  shreg <= ~rst ? -1 : start ? dataTx : endtick ? {shreg[14:0],
MISO} : shreg;
end
endmodule
//-----------------------------------------------------------------
module RAM(input wire clk, we,  // PDR 7.11.19
  input wire [10:0] adr, input wire [31:0] din, output reg [31:0]
dout);

reg [31:0] mem [0:2047];  // 2K words

always @(posedge clk) begin
  if (we) mem[adr] <= din;
  dout <= mem[adr];
end
endmodule
//-----------------------------------------------------------------
`ifdef TEST
module ROM(input wire clk,  // PDR 7.11.19 / 12.11.19
  input wire [10:0] adr, output reg [31:0] data);

  always @(posedge clk)  // basic assurance test
    data <= (adr == 0) ? 32'h42130014
      : (adr == 1) ? 32'hA20FFFC4
      : (adr == 2) ? 32'h820FFFD4
      : (adr == 3) ? 32'h42230001
      : (adr == 4) ? 32'hE8000001
      : (adr == 5) ? 32'hA10FFFD0
      : (adr == 6) ? 32'h41180001
      : (adr == 7) ? 32'hE7FFFFF8 : 0;
endmodule
`endif
//-----------------------------------------------------------------
// $ yosys -DTEST -p 'synth_ice40 -blif risc0.blif' RISC0.v
// $ arachne-pnr -d 8k -P tq144:4k -o risc0.asc -p RISC0.pcf \
       risc0.blif
// $ icetime -d hx8k -P tq144:4k -p RISC0.pcf -t risc0.asc
// $ cat 64xFF.bin risc0.bin 8xFF.bin > risc0.dfu \
       && dfu-suffix -a risc0.dfu
// $ dfu-util -D risc0.dfu
// $ dd if=/dev/zero ibs=64 count=1 | tr "\000" "\377" \
       > 64xFF.bin
// $ dd if=64xFF.bin ibs=8 count=1 > 8xFF.bin
```