Listings

Main

Listing 1: main.vhd

```
1  -- Authors:
   --       Wim Looman, Forrest McKerchar, Henry Jenkins, Joel Koh, Sasha Wang, Tracy
       Jackson
3
   library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;

7  library work;

9  entity main is
     port (
11          clk    : in  std_logic;
            reset  : in  std_logic;
13          tx     : out std_logic;
            rx     : in  std_logic;
15          sw1    : in  std_logic;
            sw2    : in  std_logic
17   );
   end main;
19
   architecture main_arch of main is
21   component cpu IS
       PORT(
23       -- instruction bus
         inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
25       inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
         inst_req  : out std_logic;                     -- Pulled low to request bus
             usage.
27       inst_ack  : in  std_logic;                     -- Pulled high to inform of
             request completion.
         -- data bus
29       data_add  : out    std_logic_vector(15 downto 0); -- Address lines.
         data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
31       data_read : out    std_logic;                     -- High for a read request,
             low for a write request.
         data_req  : out    std_logic;                     -- Pulled low to request bus
             usage.
33       data_ack  : inout std_logic;                     -- Pulled high to inform of
             request completion.
         -- extras
35       clk       : in  std_logic;
         reset     : in  std_logic
37     );
     end component;
39   component mmu_main is
       port (
41       -- instruction bus
         inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
43       inst_data : out std_logic_vector(15 downto 0); -- Data lines.
         inst_req  : in  std_logic;                     -- Pulled low to request bus
             usage.
45       inst_ack  : out std_logic;                     -- Pulled high to inform of
             request completion.
```

```
      -- data bus
47    data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
49    data_read : in    std_logic;                     -- High for a read request,
          low for a write request.
      data_req  : in    std_logic;                     -- Pulled low to request bus
          usage.
51    data_ack  : inout std_logic;                     -- Pulled high to inform of
          request completion.
      -- extras
53    clk          : in  std_logic;
      receive_pin  : in  std_logic;
55    transfer_pin : out std_logic
    );
57  END component;
    component IO is
59       PORT(
            -- data bus --
61          data_add    : IN      std_logic_vector(15 DOWNTO 0); -- address lines --
            data_data   : INOUT   std_logic_vector(7 DOWNTO 0);  -- data lines --
63          data_read   : INOUT   std_logic;                     -- pulled high for
                read, low for write --
            data_req    : INOUT   std_logic;                     -- pulled low to
                request bus usage --
65          data_ack    : INOUT   std_logic;                     -- pulled high to
                inform request completion --
            -- io --
67          clk         : IN      std_logic;
            sw1         : IN      std_logic;
69          sw2         : IN      std_logic);
            --leds      : OUT std_logic_vector(7 DOWNTO 0);
71  END component;
    -- instruction bus
73  signal inst_add  : std_logic_vector(11 downto 0); -- Address lines.
    signal inst_data : std_logic_vector(15 downto 0); -- Data lines.
75  signal inst_req  : std_logic;                     -- Pulled low to request bus
        usage.
    signal inst_ack  : std_logic;                     -- Pulled high to inform of
        request completion.
77  -- data bus
    signal data_add  : std_logic_vector(15 downto 0); -- Address lines.
79  signal data_line : std_logic_vector(7 downto 0);  -- Data lines.
    signal data_read : std_logic;                     -- High for a read request, low
        for a write request.
81  signal data_req  : std_logic;                     -- Pulled low to request bus
        usage.
    signal data_ack  : std_logic;                     -- Pulled high to inform of
        request completion.
83
    begin
85    c : cpu port map(
      -- instruction bus
87    inst_add  => inst_add, -- Instruction address
      inst_data => inst_data,-- Instruction data
89    inst_req  => inst_req, -- Request
      inst_ack  => inst_ack, -- Instruction obtained
91    -- data bus
      data_add  => data_add, -- Data address
93    data_line => data_line,-- Data
      data_read => data_read,-- 1 for read, 0 for write
95    data_req  => data_req, -- Request
      data_ack  => data_ack, -- Data written to/ read from
97    -- extras
      clk       => clk,
99    reset     => reset
      );
101   m : mmu_main port map(
      -- instruction bus
103   inst_add     => inst_add,  -- Address lines.
      inst_data    => inst_data, -- Data lines.
105   inst_req     => inst_req,  -- Pulled low to request bus usage.
      inst_ack     => inst_ack,  -- Pulled high to inform of request completion.
107   -- data bus
```

```
        data_add     => data_add,  -- Address lines.
        data_line    => data_line, -- Data lines.
        data_read    => data_read, -- High for a read request, low for a write request.
        data_req     => data_req,  -- Pulled low to request bus usage.
        data_ack     => data_ack,  -- Pulled high to inform of request completion.
        -- extras
        clk          => clk,
        receive_pin  => rx,
        transfer_pin => tx
    );
    i : io  port map(
            clk          => clk,
            data_add     => data_add,
            data_data    => data_line,
            data_read    => data_read,
            data_req     => data_req,
            data_ack     => data_ack,
            -- io --
            sw1          => sw1,
            sw2          => sw2
        );
end architecture main_arch;
```

IO

Listing 2: IO/debounce.vhd

```
1  ----------------------------------------------------------------------------------
   -- Module Name:    debounce
3  -- Description: Entity to debounce a mechanical switch/button
   -- Authors: Tracy Jackson
5  --          Sasha Wang
   --
7  ----------------------------------------------------------------------------------
   library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_ARITH.ALL;
11 use IEEE.STD_LOGIC_UNSIGNED."+";

13 library work;

15 ENTITY debounce IS
       PORT(clk : IN STD_LOGIC;
17        switch : IN STD_LOGIC;
          switch_state : OUT STD_LOGIC);
19 END debounce;

21 ARCHITECTURE debounced_switch OF debounce IS
    SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);
23 BEGIN
     -- Debounce the switch using a counter
25   PROCESS(clk, switch)
     BEGIN
27        IF switch = '0' THEN
             count <= "000";
29        ELSIF rising_edge(clk) THEN
               IF count /= "111" THEN
31                count <= count + 1;
               END IF;
33        END IF;
          IF count = "111" AND switch = '1' THEN
35           switch_state <= '1';
          ELSE
37            switch_state <= '0';
          END IF;
39   END PROCESS;
   END debounced_switch;
```

Listing 3: IO/IO.vhd

```
   ----------------------------------------------------------------------------------
2  -- Module Name:    IO
   -- Description: Entity to hangle IO
4  -- Authors: Tracy Jackson
   --          Sasha Wang
6  --
   ----------------------------------------------------------------------------------
8  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_ARITH.ALL;
   --use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
   library work;
14 use work.debounce;
   use work.switch_reg;
16 use work.led_io;

18

20 ---- Uncomment the following library declaration if instantiating
   ---- any Xilinx primitives in this code.
22 --library UNISIM;
   --use UNISIM.VComponents.all;
```

```
24
entity IO is
26        PORT(
                -- data bus --
28              data_add        : IN            std_logic_vector(15 DOWNTO 0);
                    -- address lines --
                data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data
                    lines --
30              data_read   : INOUT         std_logic;
                    -- pulled high for read, low for write --
                data_req        : INOUT     std_logic;
                    -- pulled low to request bus usage --
32              data_ack        : INOUT     std_logic;
                    -- pulled high to inform request completion --
                -- io --
34                  clk         : IN            std_logic;
                    sw1         : IN            std_logic;
36                  sw2         : IN            std_logic);
            --leds      : OUT std_logic_vector(7 DOWNTO 0);
38 end IO;

40 architecture io of IO is

42
COMPONENT led_io
44      PORT(
            data_add        : IN            std_logic_vector(15 DOWNTO 0);      --
                address lines --
46          data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data lines
                --
            data_read   : INOUT         std_logic;                              --
                pulled high for read, low for write --
48          data_req        : INOUT     std_logic;                              --
                pulled low to request bus usage --
            data_ack        : INOUT     std_logic;                              --
                pulled high to inform request completion --
50
            clock       : IN            std_logic
52          );
END COMPONENT;
54
COMPONENT switch_io IS
56      PORT ( data_add         : IN            std_logic_vector(15 DOWNTO 0);
            data_data       : INOUT         std_logic_vector(7 DOWNTO 0);
58          data_read       : INOUT         std_logic;
            data_req        : INOUT         std_logic;
60          data_ack        : INOUT         std_logic;
            clk             : IN            std_logic;
62          sw1             : IN            std_logic;
            sw2             : IN            std_logic
64          );
END COMPONENT;
66


68
BEGIN
70
led: led_io PORT MAP(data_add, data_data, data_read, data_req, data_ack, clk);
72 switch: switch_io PORT MAP(data_add, data_data, data_read, data_req, data_ack,
    clk,sw1,sw2);

74 ----------------------------------------------------------------------------------

76 END io;
```

Listing 4: IO/leds.vhd

```
  ----------------------------------------------------------------------------
2 -- Module Name:    led_io
  -- Description: Entity control output LEDs
4 -- Authors: Tracy Jackson
  --          Sasha Wang
```

```
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

library work;

ENTITY led_io IS
        PORT(
                data_add        : IN            std_logic_vector(15 DOWNTO 0);
                    -- address lines --
                data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data
                    lines --
                data_read   : INOUT         std_logic;
                    -- pulled high for read, low for write --
                data_req        : INOUT     std_logic;
                    -- pulled low to request bus usage --
                data_ack        : INOUT     std_logic;
                    -- pulled high to inform request completion --
                --
                clock       : IN            std_logic;
                );
END led_io;

ARCHITECTURE led_arch OF led_io IS
        Signal led_enable       : std_logic;
        Signal led_state        : std_logic_vector(7 DOWNTO 0);
BEGIN

   -- Determine if it is the LEDs being accessed
   PROCESS(clock, data_req, data_add, data_read)
   BEGIN
            IF data_req = '0' AND data_add = "0000000000001110" AND data_read = '0'
                THEN
                    led_enable <= '1';
            ELSE
                    led_enable <= '0';
            END IF;
   END PROCESS;

     -- process of data from the CPU and output to LEDS
    PROCESS(clock, led_enable)
    BEGIN
            IF rising_edge(clock) THEN
                IF led_enable = '1' THEN
                        led_state <= data_data;
                        data_ack <= '0';
                END IF;
            END IF;
    END PROCESS;


END led_arch;
```

Listing 5: IO/switch_register.vhd

```
----------------------------------------------------------------------------------
-- Module Name:    switch_reg
-- Description: Entity to store switch state (can be extended to more than one)
-- Authors: Tracy Jackson
--          Sasha Wang
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
```

```
ENTITY switch_reg IS
PORT( D                 : IN STD_LOGIC;
      clk,enable    : IN STD_LOGIC;
      Q                 : OUT STD_LOGIC);
END switch_reg;

ARCHITECTURE reg_arch OF switch_reg IS
BEGIN
     PROCESS(D, enable, clk)
      BEGIN
            IF rising_edge(clk) THEN --Need else there???
                    IF enable = '1' THEN
                        Q <= D;
                    END IF;
            END IF;
     END PROCESS;
END reg_arch;
```

Listing 6: IO/switches.vhd

```
--------------------------------------------------------------------------------
-- Module Name:    switch_io
-- Description: Entity to control input from switches
-- Authors: Tracy Jackson
--          Sasha Wang
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

library work;
use work.debounce;
use work.switch_reg;

entity switch_io is
       PORT(
                -- data bus --
                data_add        : IN            std_logic_vector(15 DOWNTO 0);
                    -- address lines --
                data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data
                    lines --
                data_read   : INOUT         std_logic;
                    -- pulled high for read, low for write --
                data_req        : INOUT     std_logic;
                    -- pulled low to request bus usage --
                data_ack        : INOUT     std_logic;
                    -- pulled high to inform request completion --
                -- io --
            clk         : IN            std_logic;
            sw1         : IN            std_logic;
            sw2         : IN            std_logic);
end switch_io;

architecture Behavioral of switch_io is


signal enable1                  : std_logic;
signal switch1_connection   : std_logic;
signal switch1_output       : std_logic;

signal enable2                  : std_logic;
signal switch2_connection   : std_logic;
signal switch2_output       : std_logic;


COMPONENT debounce
         PORT(clk, switch : IN STD_LOGIC;
                switch_state: OUT STD_LOGIC);
END COMPONENT;

COMPONENT switch_reg
```

```
        PORT( D            : IN STD_LOGIC;
              clk, enable  : IN STD_LOGIC;
              Q              : OUT STD_LOGIC);
END COMPONENT;


BEGIN

sw1_debouncer: debounce PORT MAP(clk, sw1, switch1_connection);
sw1_status: switch_reg PORT MAP(switch1_connection,clk, enable1, switch1_output);

sw2_debouncer: debounce PORT MAP(clk, sw2,switch2_connection);
sw2_status: switch_reg PORT MAP(switch2_connection,clk, enable2, switch2_output);

PROCESS(clk,switch1_output,switch2_output, data_ack)
BEGIN
IF rising_edge(clk) THEN
    IF switch1_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
        1, disable switch_reg from getting any more info
        enable1 <= '0';

    --ELSIF data_ack = '0' AND data_add = "0000000000001110" THEN -- when the data is
         sent to the CPU, enable the switch_reg again
    --  enable1 <= '1';
    ELSE
        enable1 <= '1';
    END IF;

    IF switch2_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
        1, disable switch_reg from getting any more info
        enable2 <= '0';

    --ELSIF data_ack = '0' AND data_add = "0000000000001100" THEN -- when the data is
         sent to the CPU, enable the switch_reg again
    --  enable2 <= '1';
    ELSE
        enable2 <= '1';
    END IF;


END IF;
END PROCESS;

PROCESS(clk, data_add, data_read)
BEGIN
    IF rising_edge(clk) THEN
        IF data_req = '0' AND data_read = '1' THEN
            IF data_add = "0000000000001110" THEN -- switch1 address
                IF switch1_output = '1' THEN
                    data_data <= "00000001";
                ELSE
                    data_data <= "00000000";
                END IF;
                data_ack <= '0';
            END IF;
            IF data_add = "0000000000001100" THEN -- switch2 address
                IF switch2_output = '1' THEN
                    data_data <= "00000001";
                ELSE
                    data_data <= "00000000";
                END IF;
                data_ack <= '0';
            END IF;
        ELSIF data_req = '1' AND data_ack = '0' THEN
            data_ack <= 'Z';
        END IF;
    END IF;
 END PROCESS;
```

```
117
    ------------------------------------------------------------------------------------------
119
    END Behavioral;
```

CPU

Listing 7: processor/alu.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  alu - alu_arch
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;


library work;
use work.fulladder8;
--use work.cpu.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alu is
    Port (f   : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
          rx  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
          ry  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
          ro  : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
          Cin : in   STD_LOGIC;                      -- Carry in
          sr  : out  STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
              C(1), N(2)
end alu;


architecture alu_arch of alu is
    component fulladder8 IS
    Port (A    : in   STD_LOGIC_VECTOR( 7 downto 0);
          B    : in   STD_LOGIC_VECTOR( 7 downto 0);
          Cin  : in   STD_LOGIC;
          Sum  : out  STD_LOGIC_VECTOR( 7 downto 0);
          Cout : out  STD_LOGIC
          );
    end component;
    signal   A         : std_logic_vector(7 downto 0);
    signal   B         : std_logic_vector(7 downto 0);
    signal   AdderCin  : std_logic;
    signal   Sum       : std_logic_vector(7 downto 0);
    signal   AdderCout : std_logic;
    signal   Z,C,N     : std_logic; -- Make the code easier to read
    signal   output    : std_logic_vector(7 downto 0); -- used to allow reading of ro
BEGIN
    Adder: fulladder8 port map(A, B, AdderCin, Sum, AdderCout);
```

```
68    process(f, rx, ry, Cin, Sum, AdderCout)
        --signal Z,C,N  : std_logic; -- Make the code easier to read
70    BEGIN
        -- use case statement to achieve
72      -- different operations of ALU

74        AdderCin <= '0';
          A <= (others => '0');
76        B <= (others => '0');
          output <= (others => '0');
78        C <= '0';
          N <= '0';
80        IF f = "0001" THEN -- Do AND operation
             output <= ry and rx;
82        ELSIF f = "0011" THEN -- Do OR  operation
            output <= ry or rx;
84        ELSIF f = "0101" THEN
            output <= not rx;
86        ELSIF f = "0111" THEN -- Do XOR operation
            output <= ry xor rx;
88        ELSIF f = "1001" THEN -- Do ADD operation
            AdderCin <= '0';
90          A <= ry;
            B <= rx;
92          output <= Sum;
          ELSIF f = "1011" THEN -- Do ADC operation
94          AdderCin <= Cin;
            A <= ry;
96          B <= rx;
            output <= Sum;
98        ELSIF f = "1101" THEN -- Do SUB operation
            AdderCin <= '1';
100         A <= ry;
            B <= (not rx);
102         output <= Sum;
          ELSIF f = "1111" THEN -- Do SBB operation
104         AdderCin <= (not Cin);
            A <= ry;
106         B <= (not rx);
            output <= Sum;
108       ELSIF f = "0100" THEN -- Do NEG operation ( two's complement )
            AdderCin <= '1';
110         A <= (others => '0');
            B <= (not rx);
112         output <= Sum;
            C <= AdderCout;
114         N <= output(7);
          ELSIF f = "0110" THEN -- Do CMP operation
116         AdderCin <= '1';
            A <= rx;
118         B <= (not ry);
            output <= Sum;
120         C <= AdderCout;
            N <= output(7);
122       ELSE
            AdderCin <= '0';
124         A <= (others => '0');
            B <= (others => '0');
126         output <= (others => '0');
            C <= '0';
128         N <= '0';
          END IF;
130 --    if (output = "00000000") then -- Set the Zero in status register
    --      sr(0) <= '1';
132 --    ELSE
    --      sr(0) <= '0';
134 --    end if;

136       C <= AdderCout; -- Carry is always 0
          N <= output(7); -- This might need to be changed to '0'
138       ro <= output;
      end process;
140     Z <= not (output(0) AND output(1) AND output(2) AND output(3) AND output(4)
```

```
                AND output(5) AND output(6) AND output(7));
142     sr(0) <= Z; --Z(0)
        sr(1) <= C; --C(1)
144     sr(2) <= N; --N(2)
        sr(15 downto 3) <= (others => '0');
146
    end alu_arch;
```

Listing 8: processor/ar.vhd

```
1  ----------------------------------------------------------------------------------
   -- Company:
3  -- Engineer:
   --
5  -- Create Date:  18:59:20 09/18/2010
   -- Design Name:
7  -- Module Name:  ar - Behavioral
   -- Project Name:
9  -- Target Devices:
   -- Tool versions:
11 -- Description:
   --
13 -- Dependencies:
   --
15 -- Revision:
   -- Revision 0.01 - File Created
17 -- Additional Comments:
   --
19 ----------------------------------------------------------------------------------
   library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;

23 library work;
   use work.reg16;
25
   entity ar is
27   Port (clk         : in   STD_LOGIC;
         enable      : in   STD_LOGIC;
29       Sel8Bit     : in   STD_LOGIC;
         SelHighByte : in   STD_LOGIC;
31       ByteInput   : in   STD_LOGIC_VECTOR (7 downto 0);
         SelRi       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
             register
33       SelRo       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
             register
         Ri          : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input
35       Ro          : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
   end ar;
37
   architecture Behavioral of ar is
39   component reg16 IS
       port(I      : in  std_logic_vector(15 downto 0);
41          clock  : in  std_logic;
            enable : in  std_logic;
43          reset  : in  std_logic;
            Q      : out std_logic_vector(15 downto 0)
45         );
     end component;
47
     signal  R0E   : std_logic;  -- Enable signals
49   signal  R1E   : std_logic;
     signal  R2E   : std_logic;
51   signal  input : std_logic_VECTOR (15 downto 0);
     signal  Q0    : std_logic_VECTOR (15 downto 0);
53   signal  Q1    : std_logic_VECTOR (15 downto 0);
     signal  Q2    : std_logic_VECTOR (15 downto 0);
55 BEGIN
       reg_0 : reg16 port map(input, clk, R0E, '0', Q0);
57     reg_1 : reg16 port map(input, clk, R1E, '0', Q1);
       reg_2 : reg16 port map(input, clk, R2E, '0', Q2);
59
     SetInput: process(clk, enable, SelRi, Ri)
```

```
61    BEGIN
        R0E <= '0';
63      R1E <= '0';
        R2E <= '0';
65      IF enable = '1' THEN
          case SelRi IS
67          WHEN "00" =>
              R0E <= '1';
69          WHEN "01" =>
              R1E <= '1';
71          WHEN "10" =>
              R2E <= '1';
73          WHEN others =>
              NULL; -- None of them are enabled
75        END CASE;
        END IF;
77    end process;

79    -- Select if 1 or 2 Bytes is to be written and if
      SetNumBytes: process(clk, Ri, SelRi, ByteInput, Sel8Bit, SelHighByte, Q0, Q1, Q2)
81    BEGIN
        IF Sel8Bit = '0' THEN
83        input <= Ri;
        ELSE
85        if SelHighByte = '1' THEN
            input(15 downto 8) <= ByteInput;
87          case SelRi IS
              WHEN "00" =>
89              input(7 downto 0) <= Q0(7 downto 0);
              WHEN "01" =>
91              input(7 downto 0) <= Q1(7 downto 0);
              WHEN others =>
93              input(7 downto 0) <= Q2(7 downto 0);
            END CASE;
95        else
            input(7 downto 0) <= ByteInput;
97          case SelRi IS
              WHEN "00" =>
99              input(15 downto 8) <= Q0(15 downto 8);
              WHEN "01" =>
101             input(15 downto 8) <= Q1(15 downto 8);
              WHEN others =>
103             input(15 downto 8) <= Q2(15 downto 8);
            END CASE;
105       END IF;
        END IF;
107   end process;

109   -- Set the output Ro
      WITH SelRo SELECT
111   Ro <= Q0 WHEN "00",
            Q1 WHEN "01",
113         Q2 WHEN others;
    end Behavioral;
```

Listing 9: processor/cpu.vhd

```
1  ----------------------------------------------------------------------------------
   -- Company:
3  -- Engineer:
   --
5  -- Create Date:    16:09:46 09/15/2010
   -- Design Name:
7  -- Module Name:    cpu - cpu_arch
   -- Project Name:
9  -- Target Devices:
   -- Tool versions:
11 -- Description:
   --
13 -- Dependencies:
   --
15 -- Revision:
```

```
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
use work.alu;
use work.cu;
use work.ar;
use work.gpr;
use work.sr;
use work.pc;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity cpu is

  PORT(
      -- instruction bus
      inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
      inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : out std_logic;                     -- Pulled low to request bus
          usage.
      inst_ack  : in  std_logic;                     -- Pulled high to inform of
          request completion.
      -- data bus
      data_add  : out    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read : out    std_logic;                     -- High for a read request,
          low for a write request.
      data_req  : out    std_logic;                     -- Pulled low to request bus
          usage.
      data_ack  : inout std_logic;                     -- Pulled high to inform of
          request completion.
      -- extras
      clk       : in  std_logic;
      reset     : in  std_logic
    );

end cpu;

architecture cpu_arch of cpu is
  component alu IS
    Port (f    : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
          rx   : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
          ry   : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
          ro   : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
          Cin  : in   STD_LOGIC;                      -- Carry in
          sr   : out  STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
              C(1), N(2)
  END component;
  component ar is
    Port (clk         : in   STD_LOGIC;
          enable      : in   STD_LOGIC;
          Sel8Bit     : in   STD_LOGIC;
          SelHighByte : in   STD_LOGIC;
          ByteInput   : in   STD_LOGIC_VECTOR (7 downto 0);
          SelRi       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
              register
          SelRo       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
              register
          Ri          : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input
          Ro          : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
  END component;
  component cu IS
```

```
81  Port (reset       : in STD_LOGIC;                          -- '0' for reset
          clock       : in STD_LOGIC;                          -- clock
83
          alu_f       : out STD_LOGIC_VECTOR (3 downto 0);     -- Function
85        alu_Cin     : out STD_LOGIC;                         -- Carry in to ALU

87        -- General Purpose Registers
          gpr_InSel   : out STD_LOGIC;                         -- select the input path (0
              - cu, 1 - ALU)
89        gpr_en      : out STD_LOGIC;                         -- enable write to GPR
          gpr_SelRx   : out STD_LOGIC_VECTOR (2 downto 0);     -- select GPR output x
91        gpr_SelRy   : out STD_LOGIC_VECTOR (2 downto 0);     -- select GPR output y
          gpr_SelRi   : out STD_LOGIC_VECTOR (2 downto 0);     -- select GPR input
93        gpr_Ri      : out STD_LOGIC_VECTOR (7 downto 0);     -- input to GPR
          gpr_Rx      : in STD_LOGIC_VECTOR (7 downto 0);      -- output Rx from GPR
95        --gpr_Ry      : in STD_LOGIC_VECTOR (7 downto 0);    -- output Ry from GPR ,
              not used

97        -- Status Register
          sr_en       : out STD_LOGIC;                         -- enable write to SR
99        sr_reset    : out STD_LOGIC;                         -- reset SR
          sr_Ro       : in STD_LOGIC_VECTOR (15 downto 0);     -- output from SR
101       -- control unit doesnt write to SR, the ALU does

103       -- Program Counter
          pc_en       : out STD_LOGIC;                         -- enable write to PC
105       pc_reset    : out STD_LOGIC;                         -- reset PC
          pc_Ri       : out STD_LOGIC_VECTOR (15 downto 0);    -- input to PC
107       pc_Ro       : in STD_LOGIC_VECTOR (15 downto 0);     -- output from PC

109       -- Address Registers
          ar_en       : out STD_LOGIC;                         -- enable write to AR
111       ar_SelRi    : out STD_LOGIC_VECTOR (1 downto 0);     -- select AR in
          ar_SelRo    : out STD_LOGIC_VECTOR (1 downto 0);     -- select AR out
113       ar_Ri       : out STD_LOGIC_VECTOR (15 downto 0);    -- input to AR
          ar_Ro       : in STD_LOGIC_VECTOR (15 downto 0);     -- output from AR
115       ar_sel8Bit  : out STD_LOGIC;                         -- only write half the AR
          ar_selHByte : out STD_LOGIC;                         -- high or low half of the
              AR to write
117       ar_ByteIn   : out STD_LOGIC_VECTOR (7 downto 0);     -- 8 bit input to write
              half of AR

119       -- Instruction memory
          inst_add    : out STD_LOGIC_VECTOR (11 downto 0);    -- Instruction address
121       inst_data   : in STD_LOGIC_VECTOR (15 downto 0);     -- Instruction data
          inst_req    : out STD_LOGIC;                         -- Request
123       inst_ack    : in STD_LOGIC;                          -- Instruction obtained

125       data_add    : out STD_LOGIC_VECTOR (15 downto 0);    -- Data address
          data_data   : inout STD_LOGIC_VECTOR (7 downto 0);-- Data
127       data_read   : out STD_LOGIC;                         -- 1 for read, 0 for write
          data_req    : out STD_LOGIC;                         -- Request
129       data_ack    : in STD_LOGIC                           -- Data written to/ read
              from

131       );
    END component;
133 component gpr is
      Port (clk       : in    STD_LOGIC;
135         enable    : in    STD_LOGIC;
            SelRx     : in    STD_LOGIC_VECTOR (2 downto 0);   -- The Rx output selection
                value
137         SelRy     : in    STD_LOGIC_VECTOR (2 downto 0);   -- The Ry output selection
                value
            SelRi     : in    STD_LOGIC_VECTOR (2 downto 0);   -- The Ri input selection
                value
139         SelIn     : in    STD_LOGIC;  -- Select where the input should be from the CU
                or CDB
            RiCU      : in    STD_LOGIC_VECTOR (7 downto 0);   -- Input from the Control
                Unit
141         RiCDB     : in    STD_LOGIC_VECTOR (7 downto 0);   -- Input from the Common
                Data Bus
            Rx        : out   STD_LOGIC_VECTOR (7 downto 0);   -- The Rx output
```

```
143           Ry       : out  STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
    END component;
145   component sr is
        Port (clk      : in  STD_LOGIC;
147           enable   : in  STD_LOGIC;
              reset    : in  STD_LOGIC;
149           Ri       : in  STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
              Ro       : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
151   END component;
    component pc is
153   Port (clk      : in  STD_LOGIC;
          enable   : in  STD_LOGIC;
155       reset    : in  STD_LOGIC;
          Ri       : in  STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
157       Ro       : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
    END component;
159   signal alu_Cin    : std_logic;
    signal alu_f      : std_logic_vector(3 downto 0);
161   signal alu_rx     : std_logic_vector(7 downto 0);
    signal alu_ry     : std_logic_vector(7 downto 0);
163
    signal sr_reset   : std_logic;
165   signal sr_enable  : std_logic;
    signal sr_Ro      : std_logic_vector(15 downto 0);
167   signal sr_input   : std_logic_vector(15 downto 0);

169   signal ar_enable   : STD_LOGIC;                       -- enable write to AR
    signal ar_SelRi    : STD_LOGIC_VECTOR (1 downto 0);  -- select AR in
171   signal ar_SelRo    : STD_LOGIC_VECTOR (1 downto 0);  -- select AR out
    signal ar_Ri       : STD_LOGIC_VECTOR (15 downto 0); -- input to AR
173   signal ar_Ro       : STD_LOGIC_VECTOR (15 downto 0);  -- output from AR
    signal ar_sel8Bit  : STD_LOGIC;                       -- only write half the AR
175   signal ar_selHByte : STD_LOGIC;                       -- high or low half of the AR
        to write
    signal ar_ByteIn   : STD_LOGIC_VECTOR (7 downto 0);  -- 8 bit input to write half
        of AR
177
    signal pc_reset   : std_logic;
179   signal pc_enable  : std_logic;
    signal pc_Ri      : std_logic_vector(15 downto 0);
181   signal pc_Ro      : std_logic_vector(15 downto 0);

183   signal gpr_InSel  : std_logic;
    signal gpr_enable : std_logic;
185   signal gpr_SelRx  : std_logic_vector(2 downto 0);
    signal gpr_SelRy  : std_logic_vector(2 downto 0);
187   signal gpr_SelRi  : std_logic_vector(2 downto 0);
    signal gpr_RiCU   : std_logic_vector(7 downto 0);
189   signal gpr_RiCDB  : std_logic_vector(7 downto 0);
  begin
191
    a: alu port map(
193           f    => alu_f,
              rx   => alu_rx,
195           ry   => alu_ry,
              ro   => gpr_RiCDB,
197           Cin  => alu_Cin,
              sr   => sr_input
199          );
    c: cu port map(
201
              reset     => reset,-- '0' for reset
203           clock     => clk,-- clock

205           alu_f     => alu_f,-- Function
              alu_Cin   => alu_Cin, -- Carry into the ALU
207
              -- General Purpose Registers
209           gpr_InSel => gpr_InSel,-- select the input path (0 - cu, 1 - ALU)
              gpr_en    => gpr_enable,-- enable write to GPR
211           gpr_SelRx => gpr_SelRx,-- select GPR output x
              gpr_SelRy => gpr_SelRy,-- select GPR output y
213           gpr_SelRi => gpr_SelRi,-- select GPR input
```

```
            gpr_Ri    => gpr_RiCU,-- input to GPR
            gpr_Rx    => alu_rx,-- Rx from GPR
            --gpr_Ry    => alu_ry,-- Ry from GPR

            -- Status Register
            sr_en     => sr_enable,-- enable write to SR
            sr_reset  => sr_reset,-- reset SR
            sr_Ro     => sr_Ro,-- output from SR
            -- control unit doesnt write to SR, the ALU does

            -- Program Counter
            pc_en     => pc_enable,-- enable write to PC
            pc_reset  => pc_reset,-- reset PC
            pc_Ri     => pc_Ri,-- input to PC
            pc_Ro     => pc_Ro,-- output from PC

            -- Address Registers
            ar_en       => ar_enable,   -- enable write to AR
            ar_SelRi    => ar_SelRi,    -- select AR in
            ar_SelRo    => ar_SelRo,    -- select AR out
            ar_sel8Bit  => ar_sel8Bit,
            ar_selHByte => ar_selHByte,
            ar_ByteIn   => ar_ByteIn,
            ar_Ri       => ar_Ri,       -- input to AR
            ar_Ro       => ar_Ro,       -- output from AR

            -- Instruction memory
            inst_add  => inst_add ,-- Instruction address
            inst_data => inst_data,-- Instruction data
            inst_req  => inst_req ,-- Request
            inst_ack  => inst_ack ,-- Instruction obtained

            data_add  => data_add ,-- Data address
            data_data => data_line,-- Data
            data_read => data_read,-- 1 for read, 0 for write
            data_req  => data_req ,-- Request
            data_ack  => data_ack  -- Data written to/ read from
          );
    address : ar port map(
            clk         => clk,
            enable      => ar_enable,
            Sel8Bit     => ar_Sel8Bit,
            SelHighByte => ar_selHByte,
            ByteInput   => ar_ByteIn,
            SelRi       => ar_SelRi,
            SelRo       => ar_SelRo,
            Ri          => ar_Ri,
            Ro          => ar_Ro
          );
g : gpr port map(
            clk    => clk,
            enable => gpr_enable,
            SelRx  => gpr_SelRx,
            SelRy  => gpr_SelRy,
            SelRi  => gpr_SelRi,
            SelIn  => gpr_InSel,
            RiCU   => gpr_RiCU,
            RiCDB  => gpr_RiCDB,
            Rx     => alu_rx,
            Ry     => alu_ry
          );
s : sr port map(
            clk    => clk,
            enable => sr_enable,
            reset  => sr_reset,
            Ri     => sr_input,
            Ro     => sr_Ro
            );
  programcounter: pc port map(
            clk     => clk,
            enable  => pc_enable,
            reset   => pc_reset,
            Ri      => pc_Ri,
```

```
287             Ro      => pc_Ro
            );
289 end cpu_arch;
```

Listing 10: processor/cu.vhd

```
 1 ----------------------------------------------------------------------------------
   -- Company:
 3 -- Engineer:
   --
 5 -- Create Date:  18:59:20 09/18/2010
   -- Design Name:
 7 -- Module Name:  cu - Behavioral
   -- Project Name:
 9 -- Target Devices:
   -- Tool versions:
11 -- Description: The control unit
   --
13 -- Dependencies:
   --
15 -- Revision:
   -- Revision 0.01 - File Created
17 -- Additional Comments:
   --
19 ----------------------------------------------------------------------------------
   library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.NUMERIC_STD.ALL;
23 use ieee.std_logic_arith.all;
   --use ieee.std_logic_unsigned.all;
25

27 library work;
   --use work.fulladder;
29 --use work.cpu.ALL;

31 -- Uncomment the following library declaration if using
   -- arithmetic functions with Signed or Unsigned values
33 --use IEEE.NUMERIC_STD.ALL;

35 -- Uncomment the following library declaration if instantiating
   -- any Xilinx primitives in this code.
37 --library UNISIM;
   --use UNISIM.VComponents.all;
39
   entity cu is
41   Port (reset       : in STD_LOGIC;                       -- '0' for reset
         clock       : in STD_LOGIC;                       -- clock
43
         alu_f       : out STD_LOGIC_VECTOR (3 downto 0);  -- Function
45       alu_Cin     : out STD_LOGIC;                      -- Carry in to ALU

47       -- General Purpose Registers
         gpr_InSel   : out STD_LOGIC;                      -- select the input path (0
              - cu, 1 - ALU)
49       gpr_en      : out STD_LOGIC;                      -- enable write to GPR
         gpr_SelRx   : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR output x
51       gpr_SelRy   : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR output y
         gpr_SelRi   : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR input
53       gpr_Ri      : out STD_LOGIC_VECTOR (7 downto 0);  -- input to GPR
         gpr_Rx      : in STD_LOGIC_VECTOR (7 downto 0);   -- output Rx from GPR
55       --gpr_Ry      : in STD_LOGIC_VECTOR (7 downto 0);   -- output Ry from GPR ,
             not used

57       -- Status Register
         sr_en       : out STD_LOGIC;                      -- enable write to SR
59       sr_reset    : out STD_LOGIC;                      -- reset SR
         sr_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from SR
61       -- control unit doesnt write to SR, the ALU does

63       -- Program Counter
         pc_en       : out STD_LOGIC;                      -- enable write to PC
```

```
65          pc_reset    : out STD_LOGIC;                        -- reset PC
            pc_Ri       : out STD_LOGIC_VECTOR (15 downto 0); -- input to PC
67          pc_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from PC

69          -- Address Registers
            ar_en       : out STD_LOGIC;                        -- enable write to AR
71          ar_SelRi    : out STD_LOGIC_VECTOR (1 downto 0);  -- select AR in
            ar_SelRo    : out STD_LOGIC_VECTOR (1 downto 0);  -- select AR out
73          ar_Ri       : out STD_LOGIC_VECTOR (15 downto 0); -- input to AR
            ar_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from AR
75          ar_sel8Bit  : out STD_LOGIC;                        -- only write half the AR
            ar_selHByte : out STD_LOGIC;                        -- high or low half of the
                AR to write
77          ar_ByteIn   : out STD_LOGIC_VECTOR (7 downto 0);  -- 8 bit input to write
                half of AR

79          -- Instruction memory
            inst_add    : out STD_LOGIC_VECTOR (11 downto 0); -- Instruction address
81          inst_data   : in STD_LOGIC_VECTOR (15 downto 0);  -- Instruction data
            inst_req    : out STD_LOGIC;                        -- Request
83          inst_ack    : in STD_LOGIC;                         -- Instruction obtained

85          data_add    : out STD_LOGIC_VECTOR (15 downto 0); -- Data address
            data_data   : inout STD_LOGIC_VECTOR (7 downto 0);-- Data
87          data_read   : out STD_LOGIC;                        -- 1 for read, 0 for write
            data_req    : out STD_LOGIC;                        -- Request
89          data_ack    : in STD_LOGIC                          -- Data written to/ read
                from

91          );
    end cu;
93

95  architecture Behavioral of cu is
      component fulladder16 IS
97    Port (A     : in    STD_LOGIC_VECTOR(15 downto 0);
            B     : in    STD_LOGIC_VECTOR(15 downto 0);
99          Cin   : in    STD_LOGIC;
            Sum   : out   STD_LOGIC_VECTOR(15 downto 0);
101         Cout  : out   STD_LOGIC
            );
103   end component;

105   type states is (reset_state, fetch, decode, execute);
      signal state      : states := reset_state;
107   signal next_state : states := reset_state;

109   signal opcode     : std_logic_vector(15 downto 0);  -- unprocessed instruction

111   -- Decoded data
      signal rx : std_logic_vector(2 downto 0);
113   signal ry : std_logic_vector(2 downto 0);
      signal ay : std_logic_vector(1 downto 0);
115
      -- Indicates what needs to be executed
117   signal write_gpr    : std_logic;
      signal write_sr     : std_logic;
119   signal write_pc     : std_logic;
      signal write_ar     : std_logic;
121   signal write_memory : std_logic;

123   -- full adders
      signal   A16         : std_logic_vector(15 downto 0);
125   signal   B16         : std_logic_vector(15 downto 0);
      signal   AdderCin16  : std_logic;
127   signal   Sum16       : std_logic_vector(15 downto 0);
      signal   AdderCout16 : std_logic;
129
      signal v    : STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit immediate
131


133 BEGIN
      Adder16: fulladder16 port map(A16, B16, AdderCin16, Sum16, AdderCout16);
```

```
135
    -- Process instruction
137 -- Assumes all instructions are valid
    process(clock, state, opcode, gpr_Rx, sr_Ro, pc_Ro, ar_Ro, inst_data, inst_ack,
        data_data, data_ack,
139         rx, ry, ay, v, write_gpr, write_sr, write_pc, write_ar, write_memory)
    BEGIN
141   if rising_edge(clock) then
        case state is
143       when reset_state =>
            sr_reset <= '0';
145         pc_reset <= '0';
            next_state <= fetch;
147
          when fetch =>
149         sr_reset <= '1';
            pc_reset <= '1';
151
            gpr_en <= '0';
153         sr_en <= '0';
            pc_en <= '0';
155         ar_en <= '0';

157         write_gpr <= '0';
            write_sr <= '0';
159         write_pc <= '0';
            write_ar <= '0';
161         write_memory <= '0';

163         inst_add <= pc_Ro(11 downto 0);
            if inst_ack = '0' then
165           inst_req <= '1';
            else
167           opcode <= inst_data;
              inst_req <= '0';
169
              -- increment program counter
171           AdderCin16 <= '1';
              A16 <= PC_Ro;
173           B16 <= "0000000000000000";
              pc_Ri <= Sum16;
175           pc_en <= '1';

177           next_state <= decode;
            end if;
179       when decode =>
            pc_en <= '0';
181
            -- ALU
183         if opcode(15) = '0' and opcode(10) = '0' and not opcode(14 downto 11) =
                "0010" then

185           ry <= opcode(7 downto 5);
              rx <= opcode(2 downto 0);
187
              gpr_SelRy <= ry;
189           gpr_SelRx <= rx;
              gpr_SelRi <= ry;
191           gpr_InSel <= '1';
              alu_f <= opcode(14 downto 11);
193           alu_Cin <= sr_Ro(1);  -- Carry

195           if not opcode(14 downto 11) = "0110" then -- CMP doesnt write to gpr, all
                  others do
              write_gpr <= '1';
197           end if;

199           write_sr <= '1';
              next_state <= execute;
201
            -- Branching
203         elsif opcode(11 downto 10) = "11" then
```

```
205             v <= "00000000";  -- initialise v
207             if opcode(15) = '1' then
                  case opcode(14 downto 12) is
209                 when "000" => -- BEQ
                      if sr_Ro(0) = '1' then -- Z=1
211                     v <= opcode(9 downto 2);
                      end if;
213                 when "001" => -- BNE
                      if sr_Ro(0) = '0' then -- Z=0
215                     v <= opcode(9 downto 2);
                      end if;
217                 when "010" => -- BLT
                      if sr_Ro(0) = '0' and sr_Ro(2) = '1' then -- Z=0 and N=1
219                     v <= opcode(9 downto 2);
                      end if;
221                 when "011" => -- BGT
                      if sr_Ro(0) = '0' and sr_Ro(2) = '0' then -- Z=0 and N=0
223                     v <= opcode(9 downto 2);
                      end if;
225                 when "100" => -- BC
                      if sr_Ro(1) = '1' then -- C=1
227                     v <= opcode(9 downto 2);
                      end if;
229                 when "101" => -- BNC
                      if sr_Ro(1) = '0' then -- C=0
231                     v <= opcode(9 downto 2);
                      end if;
233                 when "110" => -- RJMP
                      v <= opcode(9 downto 2);
235
                    when others =>
237                   v <= "00000000";
                  end case;
239
                  -- PC <- PC + v
241               AdderCin16 <= '0';
                  A16 <= PC_Ro;
243               B16 <= "00000000" & v;
                  pc_Ri <= Sum16;
245
                elsif opcode(15 downto 12) = "0111" then -- JMP
247               ay <= opcode(6 downto 5);

249               -- PC <- ay
                  ar_SelRo <= ay;
251               pc_Ri <= ar_Ro;
                else
253               -- should not reach here
                  pc_Ri <= pc_Ro; -- no change
255             end if;

257             write_pc <= '1';
                next_state <= execute;
259

261           -- Addressing
              else
263             gpr_Insel <= '0';

265             case opcode(12 downto 10) is

267               when "001" => -- Load
                    if opcode(15) = '1' then  -- immediate
269                   rx <= '0' & opcode(1 downto 0);
                      v <= opcode(9 downto 2);
271
                      -- rx <- v
273                   gpr_SelRi <= rx;
                      gpr_Ri <= v;
275                   write_gpr <= '1';
                      next_state <= execute;
277                 else                        -- direct
```

```
                    rx <= opcode(2 downto 0);
279                 ay <= opcode(6 downto 5);

281                 -- rx <- [ay]
                    gpr_SelRi <= rx;
283                 ar_selRo <= ay;
                    data_add <= ar_Ro;
285                 data_read <= '1';
                    if data_ack = '0' then  -- request data
287                   data_req <= '1';
                    else                    -- data obtained
289                   gpr_Ri <= data_data;
                      data_req <= '0';
291                   write_gpr <= '1';

293                   case opcode(14 downto 13) is
                        when "01" =>            -- auto increment
295                       AdderCin16 <= '1';
                          A16 <= ar_Ro;
297                       B16 <= "0000000000000000";
                          ar_selRi <= ay;
299                       ar_sel8bit <= '0';
                          ar_Ri <= Sum16;
301                       write_ar <= '1';
                        when "10" =>            -- auto decrement
303                       AdderCin16 <= '0';
                          A16 <= ar_Ro;
305                       B16 <= "1111111111111111";
                          ar_selRi <= ay;
307                       ar_sel8bit <= '0';
                          ar_Ri <= Sum16;
309                       write_ar <= '1';
                        when others =>
311                       -- do nothing
                      end case;
313                   next_state <= execute;
                    end if;
315               end if;

317             when "101" => -- Store
                  if opcode(15) = '1' then  -- immediate
319                 ay <= opcode(1 downto 0);
                    v <= opcode(9 downto 2);
321
                    -- [ay] <- v
323                 ar_selRo <= ay;
                    data_add <= ar_Ro;
325                 data_read <= '0';
                    data_data <= v;
327                 write_memory <= '1';
                    next_state <= execute;
329                 else                    -- direct
                    rx <= opcode(2 downto 0);
331                 ay <= opcode(6 downto 5);

333                 -- [ay] <- rx
                    gpr_selRx <= rx;
335                 ar_selRo <= ay;
                    data_add <= ar_Ro;
337                 data_read <= '0';
                    data_data <= gpr_Rx;
339                 write_memory <= '1';

341                 case opcode(14 downto 13) is
                      when "01" =>          -- auto increment
343                     AdderCin16 <= '1';
                        A16 <= ar_Ro;
345                     B16 <= "0000000000000000";
                        ar_selRi <= ay;
347                     ar_sel8bit <= '0';
                        ar_Ri <= Sum16;
349                     write_ar <= '1';
                      when "10" =>          -- auto decrement
```

```
351                     AdderCin16 <= '0';
                        A16 <= ar_Ro;
353                     B16 <= "1111111111111111";
                        ar_selRi <= ay;
355                     ar_sel8bit <= '0';
                        ar_Ri <= Sum16;
357                     write_ar <= '1';
                      when others =>
359                     -- do nothing
                    end case;
361                 next_state <= execute;
                  end if;
363
                when "100" => -- Move
365               if opcode(9) = '1' then
                    rx <= opcode(2 downto 0);
367                 ay <= opcode(6 downto 5);

369                 -- ayn <- rx
                    gpr_selRx <= rx;
371                 ar_selRi <= ay;
                    ar_sel8bit <= '1';
373                 ar_ByteIn <= gpr_Rx;

375                 if opcode(8) = '1' then     -- high
                      ar_selHByte <= '1';
377                 else                        -- low
                      ar_selHByte <= '0';
379                 end if;

381                 write_ar <= '1';
                    next_state <= execute;
383
                  elsif opcode(4) = '1' then
385                 rx <= opcode(7 downto 5);
                    ay <= opcode(1 downto 0);
387
                    -- rx <- ayn
389                 gpr_selRi <= rx;
                    ar_selRo <= ay;
391
                    if opcode(3) = '1' then     -- high
393                   gpr_Ri <= ar_Ro(15 downto 8);
                    else                        -- low
395                   gpr_Ri <= ar_Ro(7 downto 0);
                    end if;
397
                    write_gpr <= '1';
399                 next_state <= execute;

401               else
                    rx <= opcode(2 downto 0);
403                 ry <= opcode(7 downto 5);

405                 -- ry <- rx
                    gpr_SelRx <= rx;
407                 gpr_SelRi <= ry;
                    gpr_Ri <= gpr_Rx;
409
                    write_gpr <= '1';
411                 next_state <= execute;

413               end if;

415
                when others =>
417               -- should not reach here

419           end case;

421         end if;

423       when execute =>
```

```
            if write_memory = '1' then
              if data_ack = '0' then  -- request write
                data_req <= '1';
              else                    -- data written
                data_req <= '0';
          gpr_en <= write_gpr;
          sr_en <= write_sr;
          pc_en <= write_pc;
          ar_en <= write_ar;
                next_state <= fetch;
              end if;
            else
        gpr_en <= write_gpr;
        sr_en <= write_sr;
        pc_en <= write_pc;
        ar_en <= write_ar;
              next_state <= fetch;
            end if;

          when others =>
            -- shouldnt reach here
            next_state <= reset_state;
        end case;
      end if;
    end process;

    process(clock, reset, next_state)
    BEGIN
      if reset = '0' then
        state <= reset_state;
      elsif rising_edge(clock) then
        state <= next_state;
      end if;
    end process;

end Behavioral;
```

Listing 11: processor/fulladder.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  fulladder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder is
  Port (Ax    : in   STD_LOGIC;
        Bx    : in   STD_LOGIC;
        Ci    : in   STD_LOGIC;
        Sx    : out  STD_LOGIC;
        Co    : out  STD_LOGIC
        );
end fulladder;


architecture arch_fulladder of fulladder is
```

```
34 BEGIN
     process(Ax, Bx, Ci)
36   BEGIN
       Sx <= (Ax XOR Bx) XOR Ci;
38     Co <= (Ax and Bx) or (Ax and Ci) OR (Bx AND Ci);
     end process;
40 end arch_fulladder;

42 ----------------------------------------------------------------------------

44 library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
46
   entity fulladder8 is
48   Port (A    : in   STD_LOGIC_VECTOR( 7 downto 0);
           B    : in   STD_LOGIC_VECTOR( 7 downto 0);
50         Cin  : in   STD_LOGIC;
           Sum  : out  STD_LOGIC_VECTOR( 7 downto 0);
52         Cout : out  STD_LOGIC
           );
54 end fulladder8;

56 architecture arch_fulladder8 of fulladder8 is
     component fulladder IS
58   Port (Ax   : in   STD_LOGIC;
           Bx   : in   STD_LOGIC;
60         Ci   : in   STD_LOGIC;
           Sx   : out  STD_LOGIC;
62         Co   : out  STD_LOGIC
           );
64   end component;
     signal Carry   : std_logic_vector(8 downto 0);
66 BEGIN
     Carry(0) <= Cin;
68
       FA0: fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
70     FA1: fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
       FA2: fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
72     FA3: fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
       FA4: fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
74     FA5: fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));
       FA6: fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
76     FA7: fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));

78   Cout <= Carry(8);
   end arch_fulladder8;
80
   ----------------------------------------------------------------------------
82
   library IEEE;
84 use IEEE.STD_LOGIC_1164.ALL;

86 entity fulladder16 is
     Port (A    : in   STD_LOGIC_VECTOR( 15 downto 0);
88         B    : in   STD_LOGIC_VECTOR( 15 downto 0);
           Cin  : in   STD_LOGIC;
90         Sum  : out  STD_LOGIC_VECTOR( 15 downto 0);
           Cout : out  STD_LOGIC
92         );
   end fulladder16;
94
   architecture arch_fulladder16 of fulladder16 is
96   component fulladder IS
     Port (Ax   : in   STD_LOGIC;
98         Bx   : in   STD_LOGIC;
           Ci   : in   STD_LOGIC;
100        Sx   : out  STD_LOGIC;
           Co   : out  STD_LOGIC
102        );
     end component;
104  signal Carry   : std_logic_vector(16 downto 0);
   BEGIN
106  Carry(0) <= Cin;
```

```
108     FA0:  fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
        FA1:  fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
110     FA2:  fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
        FA3:  fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
112     FA4:  fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
        FA5:  fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));
114     FA6:  fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
        FA7:  fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));
116     FA8:  fulladder PORT MAP(A(8), B(8), Carry(8), Sum(8), Carry(9));
        FA9:  fulladder PORT MAP(A(9), B(9), Carry(9), Sum(9), Carry(10));
118     FA10: fulladder PORT MAP(A(10), B(10), Carry(10), Sum(10), Carry(11));
        FA11: fulladder PORT MAP(A(11), B(11), Carry(11), Sum(11), Carry(12));
120     FA12: fulladder PORT MAP(A(12), B(12), Carry(12), Sum(12), Carry(13));
        FA13: fulladder PORT MAP(A(13), B(13), Carry(13), Sum(13), Carry(14));
122     FA14: fulladder PORT MAP(A(14), B(14), Carry(14), Sum(14), Carry(15));
        FA15: fulladder PORT MAP(A(15), B(15), Carry(15), Sum(15), Carry(16));
124
    Cout <= Carry(16);
126 end arch_fulladder16;
```

Listing 12: processor/gpr.vhd

```
    ----------------------------------------------------------------------------------
 2  -- Company:
    -- Engineer:
 4  --
    -- Create Date:  18:59:20 09/18/2010
 6  -- Design Name:
    -- Module Name:  GPR - gpr_arch
 8  -- Project Name:
    -- Target Devices:
10  -- Tool versions:
    -- Description:
12  --
    -- Dependencies:
14  --
    -- Revision:
16  -- Revision 0.01 - File Created
    -- Additional Comments:
18  --
    ----------------------------------------------------------------------------------
20  library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
22
    library work;
24  use work.reg8;

26  entity gpr is
      Port (clk      : in   STD_LOGIC;
28          enable   : in   STD_LOGIC;
            SelRx    : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Rx output selection
                value
30          SelRy    : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Ry output selection
                value
            SelRi    : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Ri input selection
                value
32          SelIn    : in   STD_LOGIC;  -- Select where the input should be from the CU
                or CDB
            RiCU     : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Control
                Unit
34          RiCDB    : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Common Data
                 Bus
            Rx       : out  STD_LOGIC_VECTOR (7 downto 0);  -- The Rx output
36          Ry       : out  STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
    end gpr;
38

40  architecture gpr_arch of gpr is
      component reg8 IS
42      port(I      : in  std_logic_vector(7 downto 0);
             clock  : in  std_logic;
```

```
44          enable : in  std_logic;
            reset  : in  std_logic;
46          Q      : out std_logic_vector(7 downto 0)
           );
48  end component;

50  signal  reset: std_logic := '0';
    signal  input: std_logic_VECTOR (7 downto 0);
52  signal  R0E  : std_logic;  -- Enable signals
    signal  R1E  : std_logic;
54  signal  R2E  : std_logic;
    signal  R3E  : std_logic;
56  signal  R4E  : std_logic;
    signal  R5E  : std_logic;
58  signal  R6E  : std_logic;
    signal  R7E  : std_logic;
60  signal  Q0   : std_logic_VECTOR (7 downto 0);
    signal  Q1   : std_logic_VECTOR (7 downto 0);
62  signal  Q2   : std_logic_VECTOR (7 downto 0);
    signal  Q3   : std_logic_VECTOR (7 downto 0);
64  signal  Q4   : std_logic_VECTOR (7 downto 0);
    signal  Q5   : std_logic_VECTOR (7 downto 0);
66  signal  Q6   : std_logic_VECTOR (7 downto 0);
    signal  Q7   : std_logic_VECTOR (7 downto 0);
68 BEGIN
      reg_0 : reg8 port map(input, clk, R0E, reset, Q0);
70    reg_1 : reg8 port map(input, clk, R1E, reset, Q1);
      reg_2 : reg8 port map(input, clk, R2E, reset, Q2);
72    reg_3 : reg8 port map(input, clk, R3E, reset, Q3);
      reg_4 : reg8 port map(input, clk, R4E, reset, Q4);
74    reg_5 : reg8 port map(input, clk, R5E, reset, Q5);
      reg_6 : reg8 port map(input, clk, R6E, reset, Q6);
76    reg_7 : reg8 port map(input, clk, R7E, reset, Q7);

78  -- Select where the input should come from
      SelectInput: process(SelIn, RiCDB, RiCU)
80    BEGIN
        IF SelIn = '1' THEN
82          input <= RiCDB;
        ELSE
84        input <= RiCU;
        END IF;
86    END process;

88  -- Set Ri the input
    SetInput: process(clk, enable, SelRi)
90  BEGIN
      R0E <= '0';
92    R1E <= '0';
      R2E <= '0';
94    R3E <= '0';
      R4E <= '0';
96    R5E <= '0';
      R6E <= '0';
98    R7E <= '0';
     IF enable = '1' THEN
100     case SelRi IS
          WHEN "000" =>
102         R0E <= '1';
          WHEN "001" =>
104         R1E <= '1';
          WHEN "010" =>
106         R2E <= '1';
          WHEN "011" =>
108         R3E <= '1';
          WHEN "100" =>
110         R4E <= '1';
          WHEN "101" =>
112         R5E <= '1';
          WHEN "110" =>
114         R6E <= '1';
          WHEN "111" =>
116         R7E <= '1';
```

```
          WHEN others =>
118         NULL; -- None of them are enabled
      end case;
120   END IF;
    end process;
122
    -- Set the Rx output
124 WITH SelRx SELECT
    Rx <= Q0 WHEN "000",
126       Q1 WHEN "001",
          Q2 WHEN "010",
128       Q3 WHEN "011",
          Q4 WHEN "100",
130       Q5 WHEN "101",
          Q6 WHEN "110",
132       Q7 WHEN others;

134 -- Set the Ry output
    WITH SelRy SELECT
136 Ry <= Q0 WHEN "000",
          Q1 WHEN "001",
138       Q2 WHEN "010",
          Q3 WHEN "011",
140       Q4 WHEN "100",
          Q5 WHEN "101",
142       Q6 WHEN "110",
          Q7 WHEN others;
144
  end gpr_arch;
```

Listing 13: processor/reg.vhd

```
  ----------------------------------------------------------------------------------
2 -- Company:
  -- Engineer:
4 --
  -- Create Date:    20:08:41 10/11/2010
6 -- Design Name:
  -- Module Name:    register - Behavioral
8 -- Project Name:
  -- Target Devices:
10 -- Tool versions:
  -- Description:
12 --
  -- Dependencies:
14 --
  -- Revision:
16 -- Revision 0.01 - File Created
  -- Additional Comments:
18 --
  ----------------------------------------------------------------------------------
20 library ieee;
  use ieee.std_logic_1164.all;
22
  entity reg8 is
24 port(I      : in  std_logic_vector(7 downto 0);
       clock  : in  std_logic;
26     enable : in  std_logic;
       reset  : in  STD_LOGIC;
28     Q      : out std_logic_vector(7 downto 0)
      );
30 end reg8;

32 architecture behv of reg8 is
  begin
34
    process(I, clock, enable, reset)
36  begin
      IF reset = '1' THEN
38      Q <= (others => '0');
      ELSIF rising_edge(clock) then
40      if enable = '1' then
```

```
          Q <= I;
42       end if;
      end if;
44
    end process;
46
end behv;
48
----------------------------------------------------------------------------------
50
library ieee;
52 use ieee.std_logic_1164.all;

54 entity reg16 is
port(I      : in  std_logic_vector(15 downto 0);
56      clock  : in  std_logic;
       enable : in  std_logic;
58      reset  : in  STD_LOGIC;
       Q      : out std_logic_vector(15 downto 0)
60     );
end reg16;
62
architecture behv of reg16 is
64 begin

66   process(I, clock, enable, reset)
     begin
68     IF reset = '1' THEN
         Q <= (others => '0');
70     ELSIF rising_edge(clock) then
         if enable = '1' then
72         Q <= I;
         end if;
74     end if;

76   end process;

78 end behv;
```

Listing 14: processor/spr.vhd

```
----------------------------------------------------------------------------------
2 -- Company:
-- Engineer:
4 --
-- Create Date:  18:59:20 09/18/2010
6 -- Design Name:
-- Module Name:  sr - sr_arch
8 -- Project Name:
-- Target Devices:
10 -- Tool versions:
-- Description: The Special Purpose Register is 3, 16bit registers. One for the PC,
12 -- another for the SR and the third is the IR.
-- Dependencies:
14 --
-- Revision:
16 -- Revision 0.01 - File Created
-- Additional Comments:
18 --
----------------------------------------------------------------------------------
20 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
22
library work;
24 use work.reg16;

26 entity sr is
  Port (clk      : in  STD_LOGIC;
28        enable   : in  STD_LOGIC;
          reset    : in  STD_LOGIC;
30        Ri       : in  STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
          Ro       : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
```

```
32 end sr;

34 architecture sr_arch of sr is
     component reg16 IS
36   port(I      : in  std_logic_vector(15 downto 0);
          clock  : in  std_logic;
38        enable : in  std_logic;
          reset  : in  STD_LOGIC;
40        Q      : out std_logic_vector(15 downto 0)
       );
42   end component;
   BEGIN
44   reg_sr : reg16 port map(Ri, clk, enable, reset, Ro);
   end sr_arch;
46
   ----------------------------------------------------------------------------------

48
   library IEEE;
50 use IEEE.STD_LOGIC_1164.ALL;

52 entity pc is
     Port (clk      : in  STD_LOGIC;
54         enable   : in  STD_LOGIC;
           reset    : in  STD_LOGIC;
56         Ri       : in  STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
           Ro       : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
58 end pc;

60
   architecture pc_arch of pc is
62   component reg16 IS
     port(I      : in  std_logic_vector(15 downto 0);
64        clock  : in  std_logic;
          enable : in  std_logic;
66        reset  : in  STD_LOGIC;
          Q      : out std_logic_vector(15 downto 0)
68     );
     end component;
70 BEGIN
     reg_pc : reg16 port map(Ri, clk, enable, reset, Ro);
72 end pc_arch;
```

MMU

Listing 15: mmu/control_unit.vhd

```
1  -- Authors:
   --       Wim Looman, Forrest McKerchar
3
   library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;

7  library work;
   use work.mmu_types.all;
9
   entity mmu_control_unit is
11   port (
       eoc              : in  std_logic; -- High on muart has finished collecting data
13     eot              : in  std_logic; -- High on muart has finished transmitting
       ready            : in  std_logic; -- High if the muart is ready for new transfer
15     data_read        : in  std_logic; -- High if the cpu requests a read, else write
       data_req         : in  std_logic; -- Low to start a transfer
17     data_add_0       : in  std_logic; -- High for memory address, else IO
       inst_req         : in  std_logic; -- Low to start a transfer
19     fr               : in  std_logic; -- Input headers fetch request bit
       inst_or_data_in  : in  std_logic; -- Input headers inst or data bit
21     rw               : in  std_logic; -- Input headers read/!write bit
       write            : out std_logic; -- Pulled high to start muart writing data
23     inst_or_data_out : out std_logic; -- Ouput headers inst or data bit
       inst_ack         : out std_logic; -- Idles high, pulled low when data ready
25     data_ack         : inout std_logic; -- Idles 'Z', high when data not ready,
           pulled low when data ready
       muart_input      : out muart_input_state; -- Signal connected to muart input
27     muart_output     : out muart_output_state; -- Signal connected to muart output
       clk              : in  std_logic
29   );
   end mmu_control_unit;
31
   architecture mmu_control_unit_arch of mmu_control_unit is
33   component data_control_unit is
       port (
35       eoc          : in  std_logic; -- High on muart has finished collecting data
         eot          : in  std_logic; -- High on muart has finished transmitting
37       ready        : in  std_logic; -- High if the muart is ready for new transfer
         data_read    : in  std_logic; -- High if the cpu requests a read, else write
39       data_req     : in  std_logic; -- Low to start a transfer
         data_add_0   : in  std_logic; -- High for memory address, else IO
41       write        : out std_logic; -- Pulled high to start muart writing data.
         data_ack     : inout std_logic; -- Idles 'Z', high when data not ready, pulled
             low when data ready
43       muart_input  : out muart_input_state; -- Signal connected to muart input
         muart_output : out muart_output_state; -- Signal connected to muart output
45       clk          : in  std_logic
       );
47   end component;

49   component inst_control_unit is
       port (
51       eoc          : in  std_logic; -- High on muart has finished collecting data
         eot          : in  std_logic; -- High on muart has finished transmitting
53       ready        : in  std_logic; -- High if the muart is ready for new transfer
         inst_req     : in  std_logic; -- Low to start a transfer
55       write        : out std_logic; -- Pulled high to start muart writing data
         inst_or_data : out std_logic; -- Ouput headers inst or data bit
57       inst_ack     : out std_logic; -- Idles high, pulled low when data ready
         muart_input  : out muart_input_state; -- Signal connected to muart input
59       muart_output : out muart_output_state; -- Signal connected to muart output
         clk          : in  std_logic
61     );
   end component;
63
   signal data_write, inst_write, inst_inst_or_data_out : std_logic;
65 signal data_muart_input, inst_muart_input : muart_input_state;
   signal data_muart_output, inst_muart_output : muart_output_state;
```

```
67 begin
     data_cu : data_control_unit port map (
69     eoc,
       eot,
71     ready,
       data_read,
73     data_req,
       data_add_0,
75     data_write,
       data_ack,
77     data_muart_input,
       data_muart_output,
79     clk
     );
81   inst_cu : inst_control_unit port map (
       eoc,
83     eot,
       ready,
85     inst_req,
       inst_write,
87     inst_inst_or_data_out,
       inst_ack,
89     inst_muart_input,
       inst_muart_output,
91     clk
     );
93
     inst_or_data_out <= inst_inst_or_data_out ;
95   write            <= inst_write or data_write;
     muart_input  <= inst_muart_input  when inst_inst_or_data_out = '1' else
97                   data_muart_input;
     muart_output <= inst_muart_output when inst_inst_or_data_out = '1' else
99                   data_muart_output;
   end mmu_control_unit_arch;
```

Listing 16: mmu/data_control_unit.vhd

```
   -- Authors:
2  --       Wim Looman, Forrest McKerchar

4  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
6
   library work;
8  use work.mmu_types.all;

10 entity data_control_unit is
     port (
12       eoc          : in  std_logic; -- High on muart has finished collecting data
         eot          : in  std_logic; -- High on muart has finished transmitting
14       ready        : in  std_logic; -- High if the muart is ready for new transfer
         data_read    : in  std_logic; -- High if the cpu requests a read, else write
16       data_req     : in  std_logic; -- Low to start a transfer
         data_add_0   : in  std_logic; -- High for memory address, else IO
18       write        : out std_logic; -- Pulled high to start muart writing data.
         data_ack     : inout std_logic; -- Idles 'Z', high when data not ready, pulled
             low when data ready
20       muart_input  : out muart_input_state; -- Signal connected to muart input
         muart_output : out muart_output_state; -- Signal connected to muart output
22       clk          : in  std_logic
     );
24 end data_control_unit;

26 architecture data_control_unit_arch of data_control_unit is
     type m_state_type is (
28     idle,
       send_header, send_add_high, send_add_low, send_data,
30     get_header,  get_add_high,  get_add_low,  get_data,
       finished
32   );
     type state_type          is (idle, get_data, wait_clear);
34   type read_state_type     is (idle, wait_data, read_data, pause, finished);
```

```
    type transmit_state_type is (idle, set_data, trans_data, pause, finished);
36
    signal state,             next_state             : state_type          := idle;
38  signal get_state,         next_get_state         : m_state_type        := idle;
    signal reader_state,      next_reader_state      : read_state_type     := idle;
40  signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
  begin
42  data_fsm : process(state, data_req, clk) begin
      if (rising_edge(clk)) then
44      case state is
          when idle =>
46          if (data_req = '0' and data_add_0 = '1') then
              next_state <= get_data;
48          end if;

50        when get_data =>
            if (get_state = finished) then
52            next_state <= wait_clear;
            end if;
54
          when wait_clear =>
56          if (data_req = '1') then
              next_state <= idle;
58          end if;

60        when others =>
            NULL;
62      end case;
      end if;
64  end process data_fsm;

66  get_data_fsm : process(state, clk) begin
      if rising_edge(clk) then
68      if state = get_data then
          case get_state is
70          when idle =>
              next_get_state <= send_header;
72
            when send_header =>
74            if transmitter_state = finished then
                next_get_state <= send_add_low;
76            end if;

78          when send_add_low =>
              if transmitter_state = finished then
80              next_get_state <= send_add_high;
              end if;
82
            when send_add_high =>
84            if transmitter_state = finished then
                if data_read = '1' then
86                next_get_state <= get_header;
                else
88                next_get_state <= send_data;
                end if;
90            end if;

92          when send_data =>
              if transmitter_state = finished then
94              next_get_state <= finished;
              end if;
96
            when get_header =>
98            if reader_state = finished then
                next_get_state <= get_add_low;
100           end if;

102         when get_add_low =>
              if reader_state = finished then
104             next_get_state <= get_add_high;
              end if;
106
            when get_add_high =>
```

```
108             if reader_state = finished then
                  next_get_state <= get_data;
110             end if;

112           when get_data =>
                if reader_state = finished then
114               next_get_state <= finished;
                end if;
116
              when finished =>
118             next_get_state <= idle;

120           when others =>
                NULL;
122         end case;
          end if;
124     end if;
    end process get_data_fsm;
126
    transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
128     if rising_edge(clk) then
          if ((get_state = send_header) or
130           (get_state = send_add_low) or
              (get_state = send_add_high) or
132           (get_state = send_data)) then
            case transmitter_state is
134           when idle =>
              if ready = '1' and eot = '0' then
136               next_transmitter_state <= set_data;
              end if;
138
              when set_data =>
140             next_transmitter_state <= trans_data;

142           when trans_data =>
                next_transmitter_state <= pause;
144
              when pause =>
146             if eot = '1' then
                  next_transmitter_state <= finished;
148             end if;

150           when finished =>
                next_transmitter_state <= idle;
152
              when others =>
154             NULL;
            end case;
156       end if;
        end if;
158   end process transmit_fsm;

160   read_fsm : process(clk, get_state, reader_state, eoc) begin
        if rising_edge(clk) then
162       if ((get_state = get_header) or
              (get_state = get_add_low) or
164           (get_state = get_add_high) or
              (get_state = get_data)) then
166         case reader_state is
              when idle =>
168             next_reader_state <= wait_data;

170           when wait_data =>
                if eoc = '1' then
172               next_reader_state <= read_data;
              end if;
174
              when read_data =>
176             next_reader_state <= pause;

178           when pause =>
                if eoc = '0' then
180               next_reader_state <= finished;
```

```
            end if;

          when finished =>
            next_reader_state <= idle;

          when others =>
            NULL;
        end case;
      end if;
    end if;
  end process read_fsm;

  switch_states : process(
    clk,
    next_state,
    next_get_state,
    next_reader_state,
    next_transmitter_state
  ) begin
    if rising_edge(clk) then
      state              <= next_state;
      get_state          <= next_get_state;
      reader_state       <= next_reader_state;
      transmitter_state  <= next_transmitter_state;
    end if;
  end process switch_states;

  -- Outputs
  with state select
    data_ack <= '1' when wait_clear,
                'Z' when idle,
                '0' when others;

  with transmitter_state select
    write <= '1' when trans_data,
             '0' when others;

  muart_input <= idle          when transmitter_state /= set_data      else
                 idle          when transmitter_state /= trans_data    else
                 header        when get_state          = send_header   else
                 data_add_high when get_state          = send_add_high else
                 data_add_low  when get_state          = send_add_low  else
                 data_data     when get_state          = send_data     else
                 idle;

  muart_output <= clear_data when state         = idle       else
                  idle       when reader_state /= read_data  else
                  header     when get_state     = get_header else
                  data_data  when get_state     = get_data   else
                  idle;
end data_control_unit_arch;
```

Listing 17: mmu/header_builder.vhd

```
-- Author:
--      Forrest McKerchar

-- builds a header to feed into the RS-232 link

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;

entity header_builder is
  port (
    read_write : in  std_logic; -- 1 = read, 0 = write
    inst_data  : in  std_logic; -- 1 = inst, 0 = data
    header     : out std_logic_vector(7 downto 0)
  );
end header_builder;
```

```
19 architecture header_builder_arch of header_builder is
   begin
21   header(7) <= read_write or inst_data; -- reading from or writing to
                                             -- memory? (can't write
23                                           -- instructions)
     header(6) <= '0'; -- reserved
25   header(5) <= '0'; -- reserved
     header(4) <= '0'; -- reserved
27   header(3) <= '0'; -- diagnostic
     header(2) <= '0'; -- diagnostic;
29   header(1) <= '0'; --  1 if data is being retrieved from RS-232 link
     header(0) <= inst_data; -- instruction data or data data?
31 end header_builder_arch;
```

Listing 18: mmu/header_decoder.vhd

```
   -- Author:
2  --       Forrest McKerchar

4  -- decodes a header received from the RS-232 link

6  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
8
   library work;
10
   entity header_decoder is
12   port (
       read_write    : out  std_logic; -- 1 = read, 0 = write
14     fetch_request : out std_logic;
       inst_data     : out  std_logic; -- 1 = inst, 0 = data
16     header        : in std_logic_vector(7 downto 0)
     );
18 end header_decoder;

20 architecture header_decoder_arch of header_decoder is
   begin
22   read_write <= header(7); -- reading or writing? (should be 1 in this case)
     fetch_request <= header(1); -- fetch request? (should be 1n this case)
24   inst_data <= header(0); -- instruction data or data data?
   end header_decoder_arch;
```

Listing 19: mmu/inst_control_unit.vhd

```
   -- Authors:
2  --       Wim Looman, Forrest McKerchar

4  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
6
   library work;
8  use work.mmu_types.all;

10 entity inst_control_unit is
     port (
12      eoc           : in  std_logic; -- High on muart has finished collecting data
        eot           : in  std_logic; -- High on muart has finished transmitting
14      ready         : in  std_logic; -- High if the muart is ready for new transfer
        inst_req      : in  std_logic; -- Low to start a transfer
16      write         : out std_logic; -- Pulled high to start muart writing data
        inst_or_data  : out std_logic; -- Ouput headers inst or data bit
18      inst_ack      : out std_logic; -- Idles high, pulled low when data ready
        muart_input   : out muart_input_state; -- Signal connected to muart input
20      muart_output  : out muart_output_state; -- Signal connected to muart output
        clk           : in  std_logic
22   );
   end inst_control_unit;
24
   architecture inst_control_unit_arch of inst_control_unit is
26   type m_state_type is (
```

```
    idle,
28    send_header,   send_add_high, send_add_low,
    get_header,    get_add_high,  get_add_low,
30    get_data_high, get_data_low,  finished
  );
32  type state_type          is (idle, get_data, wait_clear);
  type read_state_type     is (idle, wait_data, read_data, pause, finished);
34  type transmit_state_type is (idle, set_data, trans_data, pause, finished);

36  signal state,              next_state             : state_type          := idle;
  signal get_state,          next_get_state         : m_state_type        := idle;
38  signal reader_state,       next_reader_state      : read_state_type     := idle;
  signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
40 begin
  inst_fsm : process(state, inst_req, clk) begin
42    case state is
      when idle =>
44      if (inst_req = '0') then
        next_state <= get_data;
46      end if;

48    when get_data =>
      if (get_state = finished) then
50      next_state <= wait_clear;
      end if;
52
    when wait_clear =>
54      if (inst_req = '1') then
        next_state <= idle;
56      end if;

58    when others =>
      NULL;
60  end case;
  end process inst_fsm;
62
  get_inst_fsm : process(state, clk) begin
64    if state = get_data then
      case get_state is
66      when idle =>
        next_get_state <= send_header;
68
      when send_header =>
70      if transmitter_state = finished then
          next_get_state <= send_add_low;
72      end if;

74    when send_add_low =>
        if transmitter_state = finished then
76        next_get_state <= send_add_high;
        end if;
78
      when send_add_high =>
80      if transmitter_state = finished then
          next_get_state <= get_header;
82      end if;

84    when get_header =>
        if reader_state = finished then
86        next_get_state <= get_add_low;
        end if;
88
      when get_add_low =>
90      if reader_state = finished then
          next_get_state <= get_add_high;
92      end if;

94    when get_add_high =>
        if reader_state = finished then
96        next_get_state <= get_data_low;
        end if;
98
      when get_data_low =>
```

```
100           if reader_state = finished then
                next_get_state <= get_data_high;
102           end if;

104         when get_data_high =>
              if reader_state = finished then
106             next_get_state <= finished;
              end if;
108
            when finished =>
110           next_get_state <= idle;

112         when others =>
              NULL;
114       end case;
      end if;
116   end process get_inst_fsm;

118   transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
      if ((get_state = send_header) or
120       (get_state = send_add_low) or
          (get_state = send_add_high)) then
122     case transmitter_state is
          when idle =>
124         if ready = '1' and eot = '0' then
              next_transmitter_state <= set_data;
126         end if;

128       when set_data =>
            next_transmitter_state <= trans_data;
130
          when trans_data =>
132         next_transmitter_state <= pause;

134       when pause =>
            if eot = '1' then
136           next_transmitter_state <= finished;
            end if;
138
          when finished =>
140         next_transmitter_state <= idle;

142       when others =>
            NULL;
144     end case;
      end if;
146   end process transmit_fsm;

148   read_fsm : process(clk, get_state, reader_state, eoc) begin
      if ((get_state = get_header) or
150       (get_state = get_add_low) or
          (get_state = get_add_high) or
152       (get_state = get_data_low) or
          (get_state = get_data_high)) then
154     case reader_state is
          when idle =>
156         next_reader_state <= wait_data;

158       when wait_data =>
            if eoc = '1' then
160           next_reader_state <= read_data;
            end if;
162
          when read_data =>
164         next_reader_state <= pause;

166       when pause =>
            if eoc = '0' then
168           next_reader_state <= finished;
            end if;
170
          when finished =>
172         next_reader_state <= idle;
```

```
174         when others =>
              NULL;
176       end case;
      end if;
178   end process read_fsm;

180   switch_states : process(
        clk,
182     next_state,
        next_get_state,
184     next_reader_state,
        next_transmitter_state) begin
186     if rising_edge(clk) then
          state <= next_state;
188       get_state <= next_get_state;
          reader_state <= next_reader_state;
190       transmitter_state <= next_transmitter_state;
        end if;
192   end process switch_states;

194   -- Outputs
      with state select
196     inst_ack <= '1' when wait_clear,
                  '0' when others;
198
      with state select
200     inst_or_data <= '0' when idle,
                      '1' when others;
202
      with transmitter_state select
204     write <= '1' when trans_data,
               '0' when others;
206

208   muart_input <= idle          when transmitter_state /= set_data      else
                     idle          when transmitter_state /= trans_data    else
210                  header        when get_state          = send_header   else
                     inst_add_high when get_state          = send_add_high else
212                  inst_add_low  when get_state          = send_add_low  else
                     idle;
214
      muart_output <= idle           when reader_state /= read_data     else
216                   header         when get_state     = get_header    else
                      inst_data_high when get_state     = get_data_high else
218                   inst_data_low  when get_state     = get_data_low  else
                      idle;
220 end inst_control_unit_arch;
```

Listing 20: mmu/mmu.vhd

```
  -- Authors:
2 --       Wim Looman, Forrest McKerchar

4 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
6
  library work;
8 use work.mmu_types.all;
  use work.mmu_control_unit;
10 use work.header_builder;
  use work.header_decoder;
12 use work.reg8;

14 use work.minimal_uart_core;

16   entity mmu_main is
       port (
18       -- instruction bus
         inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
20       inst_data : out std_logic_vector(15 downto 0); -- Data lines.
```

```
    inst_req  : in  std_logic;                    -- Pulled low to request bus
        usage.
    inst_ack  : out std_logic;                    -- Pulled high to inform of
        request completion.
    -- data bus
    data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
    data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
    data_read : in    std_logic;                    -- High for a read request,
        low for a write request.
    data_req  : in    std_logic;                    -- Pulled low to request bus
        usage.
    data_ack  : inout std_logic;                    -- Pulled high to inform of
        request completion.
    -- extras
    clk          : in  std_logic;
    receive_pin  : in  std_logic;
    transfer_pin : out std_logic
  );
end mmu_main;

architecture mmu_arch of mmu_main is
  component mmu_control_unit is
    port (
      eoc              : in  std_logic; -- High on muart has finished collecting
          data
      eot              : in  std_logic; -- High on muart has finished transmitting
      ready            : in  std_logic; -- High if the muart is ready for new
          transfer
      data_read        : in  std_logic; -- High if the cpu requests a read, else
          write
      data_req         : in  std_logic; -- Low to start a transfer
      data_add_0       : in  std_logic; -- High for memory address, else IO
      inst_req         : in  std_logic; -- Low to start a transfer
      fr               : in  std_logic; -- Input headers fetch request bit
      inst_or_data_in  : in  std_logic; -- Input headers inst or data bit
      rw               : in  std_logic; -- Input headers read/!write bit
      write            : out std_logic; -- Pulled high to start muart writing data
      inst_or_data_out : out std_logic; -- Ouput headers inst or data bit
      inst_ack         : out std_logic; -- Idles high, pulled low when data ready
      data_ack         : inout std_logic; -- Idles 'Z', high when data not ready,
          pulled low when data ready
      muart_input      : out muart_input_state; -- Signal connected to muart input
      muart_output     : out muart_output_state; -- Signal connected to muart
          output
      clk              : in  std_logic
    );
  end component;

  component header_builder is
    port (
      read_write : in  std_logic; -- 1 = read, 0 = write
      inst_data  : in  std_logic; -- 1 = inst, 0 = data
      header     : out std_logic_vector(7 downto 0)
    );
  end component;

  component header_decoder is
    port (
      read_write    : out  std_logic; -- 1 = read, 0 = write
      fetch_request : out std_logic;
      inst_data     : out  std_logic; -- 1 = inst, 0 = data
      header        : in std_logic_vector(7 downto 0)
    );
  end component;

  component minimal_uart_core is
    port(
      clock : in    std_logic;
      eoc   : out   std_logic;
      outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
      rxd   : in    std_logic;
      txd   : out   std_logic;
      eot   : out   std_logic;
```

```
84          inp    : in     std_logic_vector(7 downto 0);
            ready  : out    std_logic;
86          wr     : in     std_logic
          );
88      end component;

90      component reg8 IS
          port(
92          I      : in  std_logic_vector(7 downto 0);
            clock  : in  std_logic;
94          enable : in  std_logic;
            reset  : in  std_logic;
96          Q      : out std_logic_vector(7 downto 0)
          );
98      end component;

100
        signal eoc           : std_logic;
102     signal eot           : std_logic;
        signal ready         : std_logic;
104     signal fr            : std_logic;
        signal inst_or_data_in : std_logic;
106     signal rw            : std_logic;

108     signal write         : std_logic;
        signal inst_or_data_out : std_logic;
110     signal muart_input   : muart_input_state;
        signal muart_output : muart_output_state;
112
        signal muart_out : std_logic_vector(7 downto 0);
114     signal muart_in  : std_logic_vector(7 downto 0);
        signal header_in   : std_logic_vector(7 downto 0);
116     signal header_out  : std_logic_vector(7 downto 0);
        signal inst_data_high_enable : std_logic;
118     signal inst_data_low_enable : std_logic;
        signal data_data_enable : std_logic;
120     signal data_line_tri : std_logic_vector(7 downto 0);

122     begin
        muart : minimal_uart_core port map (
124       clk,
          eoc,
126       muart_out,
          receive_pin,
128       transfer_pin,
          eot,
130       muart_in,
          ready,
132       write
        );
134     cu  : mmu_control_unit port map (
          eoc,
136       eot,
          ready,
138       data_read,
          data_req,
140       data_add(0),
          inst_req,
142       fr,
          inst_or_data_in,
144       rw,
          write,
146       inst_or_data_out,
          inst_ack,
148       data_ack,
          muart_input,
150       muart_output,
          clk
152     );
        hb  : header_builder port map (
154       data_read,
          inst_or_data_out,
156       header_out
```

```
    );
158 hd  : header_decoder port map (
      rw,
160   fr,
      inst_or_data_in,
162   header_in
    );
164 idh : reg8 port map (
      muart_out,
166   clk,
      inst_data_high_enable,
168   '0',
      inst_data(15 downto 8)
170 );
    idl : reg8 port map (
172   muart_out,
      clk,
174   inst_data_low_enable,
      '0',
176   inst_data(7  downto 0)
    );
178 dd  : reg8 port map (
      muart_out,
180   clk,
      data_data_enable,
182   '0',
      data_line_tri
184 );

186 with muart_input select
      muart_in <=  header_out                      when header,
188                "0000" & inst_add(11 downto 8) when inst_add_high,
                   inst_add(7  downto 0)          when inst_add_low,
190                '0' & data_add(15 downto 9)    when data_add_high,
                   data_add(8  downto 1)          when data_add_low,
192                data_line                      when data_data,
                   (others => '0')                when others;
194
    route_output : process(muart_output, muart_out) begin
196   header_in <= (others => '0');
      inst_data_high_enable <= '0';
198   inst_data_low_enable <= '0';
      data_data_enable <= '0';
200   case muart_output is
        when header =>
202       header_in <= muart_out;

204     when inst_data_high =>
          inst_data_high_enable <= '1';
206
        when inst_data_low =>
208       inst_data_low_enable <= '1';

210     when data_data =>
          data_data_enable <= '1';
212     when others =>
          NULL;
214   end case;
    end process;
216
    data_line <= data_line_tri when data_add(0) = '1' and data_read = '1' else
218              (others => 'Z');

220 end mmu_arch;
```

Listing 21: mmu/mmu_types.vhd

```
  -- Authors:
2 --      Wim Looman, Forrest McKerchar

4 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
```

```
6
  package mmu_types is
8    type muart_input_state is (header, inst_add_high, inst_add_low, data_add_high,
         data_add_low, data_data, idle);
     type muart_output_state is (header, inst_data_high, inst_data_low, data_data,
         clear_data, idle);
10 end mmu_types;
```

Listing 22: mmu/muart/BRG.vhd

```
  --*************************************************************************
2 --*  Minimal UART ip core                                                  *
  --* Author: Arao Hayashida Filho        arao@medinovacao.com.br           *
4 --*                                                                        *
  --*************************************************************************
6 --*                                                                        *
  --* Copyright (C) 2009 Arao Hayashida Filho                               *
8 --*                                                                        *
  --* This source file may be used and distributed without                  *
10 --* restriction provided that this copyright statement is not            *
  --* removed from the file and that any derivative work contains           *
12 --* the original copyright notice and the associated disclaimer.         *
  --*                                                                       *
14 --* This source file is free software; you can redistribute it           *
  --* and/or modify it under the terms of the GNU Lesser General            *
16 --* Public License as published by the Free Software Foundation;         *
  --* either version 2.1 of the License, or (at your option) any            *
18 --* later version.                                                        *
  --*                                                                       *
20 --* This source is distributed in the hope that it will be               *
  --* useful, but WITHOUT ANY WARRANTY; without even the implied            *
22 --* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR               *
  --* PURPOSE.  See the GNU Lesser General Public License for more          *
24 --* details.                                                              *
  --*                                                                       *
26 --* You should have received a copy of the GNU Lesser General            *
  --* Public License along with this source; if not, download it            *
28 --* from http://www.opencores.org/lgpl.shtml                              *
  --*                                                                       *
30 --*************************************************************************

32 library ieee;
  use ieee.std_logic_1164.all;
34 use ieee.std_logic_arith.all;
  use ieee.std_logic_unsigned."+";
36
  entity br_generator is
38   generic (divider_width: integer := 16);
    port (
40     clock      : in  std_logic;
       rx_enable  : in  std_logic;
42     clk_txd    : out std_logic;
       tx_enable  : in  std_logic;
44     clk_serial : out std_logic
    );
46 end br_generator;

48 architecture principal of br_generator is
    -- change the following constant to your desired baud rate
50  -- one hz equal to one bit per second
    signal   count_brg     : std_logic_vector(divider_width - 1 downto 0) := (others =>
        '0');
52  signal   count_brg_txd : std_logic_vector(divider_width - 1 downto 0) := (others =>
        '0');
    constant brdvd         : std_logic_vector(divider_width - 1 downto 0) := x"0516";
       -- 38400 bps @ 50MHz
54
    begin
56    txd : process (clock)
      begin
58      if (rising_edge(clock)) then
          if (count_brg_txd = brdvd) then
```

```
60            clk_txd       <= '1';
              count_brg_txd <= (others => '0');
62          elsif (tx_enable = '1') then
              clk_txd       <= '0';
64            count_brg_txd <= count_brg_txd + 1;
            else
66            clk_txd       <= '0';
              count_brg_txd <= (others => '0');
68          end if;
          end if;
70      end process txd;

72      rxd : process (clock)
        begin
74        if (rising_edge(clock)) then
            if (count_brg=brdvd) then
76            count_brg   <= (others => '0');
              clk_serial <= '1';
78          elsif (rx_enable = '1') then
              count_brg   <= count_brg+1;
80            clk_serial <= '0';
            else
82            count_brg   <= '0' & brdvd(divider_width - 1 downto 1);
              clk_serial <= '0';
84          end if;
          end if;
86      end process rxd;
    end principal;
```

Listing 23: mmu/muart/serial.vhd

```
1  --*****************************************************************************
   --*  Minimal UART ip core                                                     *
3  --* Author: Arao Hayashida Filho        arao@medinovacao.com.br               *
   --*                                                                           *
5  --*****************************************************************************
   --*                                                                           *
7  --* Copyright (C) 2009 Arao Hayashida Filho                                   *
   --*                                                                           *
9  --* This source file may be used and distributed without                      *
   --* restriction provided that this copyright statement is not                 *
11 --* removed from the file and that any derivative work contains               *
   --* the original copyright notice and the associated disclaimer.              *
13 --*                                                                           *
   --* This source file is free software; you can redistribute it                *
15 --* and/or modify it under the terms of the GNU Lesser General                *
   --* Public License as published by the Free Software Foundation;              *
17 --* either version 2.1 of the License, or (at your option) any                *
   --* later version.                                                            *
19 --*                                                                           *
   --* This source is distributed in the hope that it will be                    *
21 --* useful, but WITHout ANY WARRANTY; without even the implied                 *
   --* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR                   *
23 --* PURPOSE.  See the GNU Lesser General Public License for more              *
   --* details.                                                                  *
25 --*                                                                           *
   --* You should have received a copy of the GNU Lesser General                 *
27 --* Public License along with this source; if not, download it                *
   --* from http://www.opencores.org/lgpl.shtml                                  *
29 --*                                                                           *
   --*****************************************************************************
31
   library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;

35 entity minimal_uart_core is
     port(
37     clock : in    std_logic;
       eoc   : out   std_logic;
39     outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
       rxd   : in    std_logic;
41     txd   : out   std_logic;
```

```
      eot   : out   std_logic;
43    inp   : in    std_logic_vector(7 downto 0);
      ready : out   std_logic;
45    wr    : in    std_logic
    );
47 end minimal_uart_core;

49 architecture principal of minimal_uart_core  is
    type state is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);
51  signal clk_serial       : std_logic := '0';
    signal start            : std_logic := '0';
53  signal eocs, eoc1, eoc2 : std_logic := '0';
    signal rx_ck_enable     : std_logic := '0';
55  signal receiving        : std_logic := '0';
    signal transmitting     : std_logic := '0';
57  signal clk_txd          : std_logic := '0';
    signal txds             : std_logic := '1';
59  signal eots             : std_logic := '0';
    signal inpl             : std_logic_vector(7 downto 0) := x"00";
61  signal data             : std_logic_vector(7 downto 0) := x"00";
    signal atual_state, next_state, atual_state_txd, next_state_txd: state := s0;
63  signal tx_enable        : std_logic := '0';
    signal tx_ck_enable     : std_logic := '0';
65
    component br_generator
67    port (
        clock      : in  std_logic;
69      rx_enable  : in  std_logic;
        clk_txd    : out std_logic;
71      tx_enable  : in  std_logic;
        clk_serial : out std_logic
73    );
    end component;
75
    begin
77    ready <= not(tx_enable);
      brg : br_generator port map (clock, rx_ck_enable, clk_txd, tx_ck_enable,
          clk_serial);
79    rx_ck_enable <= start or receiving;
      tx_ck_enable <= tx_enable or transmitting;
81
      start_detect : process(rxd, eocs)
83    begin
        if (eocs = '1') then
85        start <= '0';
        elsif (falling_edge(rxd)) then
87        start <= '1';
        end if;
89    end process start_detect;

91    rxd_states : process (clk_serial)
      begin
93      if (rising_edge(clk_serial)) then
          atual_state <= next_state;
95      end if;
      end process rxd_states;
97
      rxd_state_machine : process(start, atual_state)
99    begin
        if (start = '1' or receiving = '1') then
101       case atual_state is
            when s0 =>
103           eocs <= '0';
              if (start = '1') then
105             next_state <= s1;
                receiving  <= '1';
107           else
                next_state <= s0;
109             receiving  <= '0';
              end if;
111
            when s1 =>
113           receiving  <= '1';
```

```
            eocs       <= '0';
            next_state <= s2;

          when s2   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s3;

          when s3   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s4;

          when s4   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s5;

          when s5   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s6;

          when s6   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s7;

          when s7   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s8;

          when s8   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s9;

          when s9   =>
            receiving  <= '1';
            eocs       <= '1';
            next_state <= s0;

          when others =>
            null;

        end case;
      end if;
    end process rxd_state_machine;

    rxd_shift : process(clk_serial)
    begin
      if (rising_edge(clk_serial)) then
        if (eocs = '0') then
          data <= rxd & data(7 downto 1);
        end if;
      end if;
    end process rxd_shift;

    process (clock)
    begin
      if (rising_edge(clock)) then
        eoc <= eocs;
      end if;
    end process;

    process(atual_state)
    begin
      if (atual_state=s9) then
        outp <= data;
      end if;
    end process;
```

```
187     txd_states : process(clk_txd)
        begin
189       if (rising_edge(clk_txd)) then
            atual_state_txd <= next_state_txd;
191       end if;
        end process txd_states;
193
        txd_state_machine : process(atual_state_txd, tx_enable)
195     begin
          case atual_state_txd is
197         when s0 =>
              inpl <= inp;
199           eots <= '0';
              if (tx_enable = '1') then
201             txds           <= '0';
                transmitting   <= '1';
203             next_state_txd <= s1;
              else
205             txds           <= '1';
                transmitting   <= '0';
207             next_state_txd <= s0;
              end if;
209
            when s1 =>
211           txds           <= inpl(0);
              eots           <= '0';
213           transmitting   <= '1';
              next_state_txd <= s2;
215
            when s2 =>
217           txds           <= inpl(1);
              eots           <= '0';
219           transmitting   <= '1';
              next_state_txd <= s3;
221
            when s3 =>
223           txds           <= inpl(2);
              eots           <= '0';
225           transmitting   <= '1';
              next_state_txd <= s4;
227
            when s4 =>
229           txds           <= inpl(3);
              eots           <= '0';
231           transmitting   <= '1';
              next_state_txd <= s5;
233
            when s5 =>
235           txds           <= inpl(4);
              eots           <= '0';
237           transmitting   <= '1';
              next_state_txd <= s6;
239
            when s6 =>
241           txds           <= inpl(5);
              eots           <= '0';
243           transmitting   <= '1';
              next_state_txd <= s7;
245
            when s7 =>
247           txds           <= inpl(6);
              eots           <= '0';
249           transmitting   <= '1';
              next_state_txd <= s8;
251
            when s8 =>
253           txds           <= inpl(7);
              eots           <= '0';
255           transmitting   <= '1';
              next_state_txd <= s9;
257
            when s9 =>
259           txds           <= '1';
```

```
            eots           <= '1';
            transmitting   <= '1';
            next_state_txd <= s0;

          when others =>
            null;

      end case;
    end process txd_state_machine;

    tx_start:process (clock, wr,  eots)
    begin
      if (eots = '1') then
        tx_enable <= '0';
      elsif (falling_edge(clock)) then
        if (wr = '1') then
          tx_enable <= '1';
        end if;
      end if;
    end process tx_start;

    eot<=eots;

    process (clock)
    begin
      if (rising_edge(clock)) then
        txd <= txds;
      end if;
    end process;

end principal ;
```

Test Benchs

Listing 24: processor/alu_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--  A testbench has no ports.
entity alu_tb is
  end alu_tb;

architecture behav of alu_tb is
  --  Declaration of the component that will be instantiated.
  component alu
   Port (f   : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
         rx  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
         ry  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
         ro  : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
         Cin : in   STD_LOGIC;                      -- Carry in
         sr  : out  STD_LOGIC_VECTOR (2 downto 0)); -- Status register out Z(0),
             C(1), N(2)
  end component;
  --  Specifies which entity is bound with the component.
  for alu_0: alu use entity work.alu;
    signal f          : STD_LOGIC_VECTOR (3 downto 0);
    signal rx, ry, ro : STD_LOGIC_VECTOR (7 downto 0);
    signal Cin        : STD_LOGIC;
    signal sr         : STD_LOGIC_VECTOR (2 downto 0);
  begin
    --  Component instantiation.
    alu_0: alu port map (f => f, rx => rx, ry => ry, ro => ro, Cin => Cin, sr => sr);

    --  This process does the real job.
    process
    type pattern_type is record
      f          : STD_LOGIC_VECTOR (3 downto 0);
      rx, ry     : STD_LOGIC_VECTOR (7 downto 0);
      ro         : STD_LOGIC_VECTOR (7 downto 0);
      Cin        : STD_LOGIC;
      sr         : STD_LOGIC_VECTOR (2 downto 0);
    end record;
  --  The patterns to apply.
  type pattern_array is array (natural range <>) of pattern_type;
  constant patterns : pattern_array :=
  -- f       rx           ry           ro          Cin   sr
  (("0001", "00000000", "00000000", "00000000", '0', "001"), --AND tests - 1ns
   ("0001", "00000001", "00000001", "00000001", '0', "000"), --AND tests
   ("0001", "00000000", "00000001", "00000000", '0', "001"), --AND tests
   ("0001", "10101010", "10101010", "10101010", '0', "100"), --AND tests
   ("0001", "01010101", "01010101", "01010101", '0', "000"), --AND tests - 5ns
   ("0001", "11111111", "00000000", "00000000", '0', "001"), --AND tests
   ("0001", "11111111", "11111111", "11111111", '0', "100"), --AND tests
   ("0001", "00000000", "01010101", "00000000", '0', "001"), --AND tests
   ("0001", "00000000", "10101010", "00000000", '0', "001"), --AND tests
   ("0001", "11111111", "01010101", "01010101", '0', "000"), --AND tests - 10 ns
   ("0001", "11111111", "10101010", "10101010", '0', "100"), --AND tests
   ("0001", "10000011", "10110010", "10000010", '0', "100"), --AND tests
   ("0001", "00000011", "00110010", "00000010", '0', "000"), --AND tests

   ("0011", "00000000", "00000000", "00000000", '0', "001"), --OR tests - 14 ns
   ("0011", "00000001", "00000001", "00000001", '0', "000"), --OR tests
   ("0011", "00000000", "00000001", "00000001", '0', "000"), --OR tests
   ("0011", "10101010", "10101010", "10101010", '0', "100"), --OR tests
   ("0011", "01010101", "01010101", "01010101", '0', "000"), --OR tests
   ("0011", "11111111", "00000000", "11111111", '0', "100"), --OR tests
   ("0011", "11111111", "11111111", "11111111", '0', "100"), --OR tests - 20 ns
   ("0011", "00000000", "01010101", "01010101", '0', "000"), --OR tests
   ("0011", "00000000", "10101010", "10101010", '0', "100"), --OR tests
   ("0011", "11111111", "01010101", "11111111", '0', "100"), --OR tests
   ("0011", "11111111", "10101010", "11111111", '0', "100"), --OR tests
   ("0011", "10000011", "10110010", "10110011", '0', "100"), --OR tests - 25 ns
   ("0011", "00000011", "00110010", "00110011", '0', "000"), --OR tests
```

```
68
    ("0101", "00000000", "00000000", "11111111", '0', "100"), --NOT tests - ry should
        not matter
70  ("0101", "00000001", "00000001", "11111110", '0', "100"), --NOT tests
    ("0101", "00000000", "00000001", "11111111", '0', "100"), --NOT tests
72  ("0101", "10101010", "10101010", "01010101", '0', "000"), --NOT tests - 30 ns
    ("0101", "01010101", "01010101", "10101010", '0', "100"), --NOT tests
74  ("0101", "11111111", "00000000", "00000000", '0', "001"), --NOT tests
    ("0101", "11111111", "11111111", "00000000", '0', "001"), --NOT tests
76  ("0101", "00000000", "01010101", "11111111", '0', "100"), --NOT tests
    ("0101", "00000000", "10101010", "11111111", '0', "100"), --NOT tests - 35 ns
78  ("0101", "11111111", "01010101", "00000000", '0', "001"), --NOT tests
    ("0101", "11111111", "10101010", "00000000", '0', "001"), --NOT tests
80  ("0101", "10000011", "10110010", "01111100", '0', "000"), --NOT tests
    ("0101", "00000011", "00110010", "11111100", '0', "100"), --NOT tests - 39 ns
82
    ("0111", "00000000", "00000000", "00000000", '0', "001"), --XOR tests - 40 ns
84  ("0111", "00000001", "00000001", "00000000", '0', "001"), --XOR tests
    ("0111", "00000000", "00000001", "00000001", '0', "000"), --XOR tests
86  ("0111", "10101010", "10101010", "00000000", '0', "001"), --XOR tests
    ("0111", "01010101", "01010101", "00000000", '0', "001"), --XOR tests
88  ("0111", "11111111", "00000000", "11111111", '0', "100"), --XOR tests - 45 ns
    ("0111", "11111111", "11111111", "00000000", '0', "001"), --XOR tests
90  ("0111", "00000000", "01010101", "01010101", '0', "000"), --XOR tests
    ("0111", "00000000", "10101010", "10101010", '0', "100"), --XOR tests
92  ("0111", "11111111", "01010101", "10101010", '0', "100"), --XOR tests
    ("0111", "11111111", "10101010", "01010101", '0', "000"), --XOR tests - 50 ns
94  ("0111", "10000011", "10110010", "00110001", '0', "000"), --XOR tests
    ("0111", "00000011", "00110010", "00110001", '0', "000")  --XOR tests
96  );
  begin
98  --  Check each pattern.
    for i in patterns'range loop
100   --  Set the inputs.
      Cin <= patterns(i).Cin;
102   f <= patterns(i).f;
      rx <= patterns(i).rx;
104   ry <= patterns(i).ry;
      --  Wait for the results.
106   wait for 1 ns;
      --  Check the outputs.
108   assert ro = patterns(i).ro
      report "bad output register value" severity error;
110   assert sr = patterns(i).sr
      report "bad status register value" severity error;
112   assert sr(0) = patterns(i).sr(0)
      report " *Zero is incorrect" severity error;
114   assert sr(1) = patterns(i).sr(1)
      report " *Carry is incorrect" severity error;
116   assert sr(2) = patterns(i).sr(2)
      report " *Negitive is incorrect" severity error;
118 end loop;
    assert false report "end of test" severity note;
120 --  Wait forever; this will finish the simulation.
    wait;
122 end process;
  end behav;
```

Listing 25: processor/fulladder_tb.vhd

```
  library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

4 --  A testbench has no ports.
  entity fulladder8_tb is
6   end fulladder8_tb;

8 architecture behav of fulladder8_tb is
    --  Declaration of the component that will be instantiated.
10  component fulladder8
    Port (A    : in   STD_LOGIC_VECTOR( 7 downto 0);
12        B    : in   STD_LOGIC_VECTOR( 7 downto 0);
```

```
        Cin  : in   STD_LOGIC;
        Sum  : out  STD_LOGIC_VECTOR( 7 downto 0);
        Cout : out  STD_LOGIC
        );
  end component;
  --  Specifies which entity is bound with the component.
  for fulladder8_0: fulladder8 use entity work.fulladder8;
    signal A,B,Sum    : STD_LOGIC_VECTOR (7 downto 0);
    signal Cin,Cout   : STD_LOGIC;
  begin
    --  Component instantiation.
    fulladder8_0: fulladder8 port map (A => A, B => B, Cin => Cin, Sum => Sum, Cout
        => Cout);

    --  This process does the real job.
    process
    type pattern_type is record
      A    :  STD_LOGIC_VECTOR( 7 downto 0);
      B    :  STD_LOGIC_VECTOR( 7 downto 0);
      Cin  :  STD_LOGIC;
      Sum  :  STD_LOGIC_VECTOR( 7 downto 0);
      Cout :  STD_LOGIC;
    end record;
  --  The patterns to apply.
  type pattern_array is array (natural range <>) of pattern_type;
  constant patterns : pattern_array :=
  -- A           B           Cin   Sum      Cout
  (("00000000", "00000000", '0', "00000000", '0'), --AND tests - 1ns
   ("11111111", "11111111", '1', "11111111", '1'), --AND tests
   ("00000000", "00000000", '1', "00000001", '0'), --AND tests
   ("00000000", "11111111", '0', "11111111", '0'), --AND tests
   ("11111111", "00000000", '0', "11111111", '0'), --AND tests
   ("11111111", "00000000", '1', "00000000", '1'), --AND tests
   ("10101010", "01010101", '0', "11111111", '0'), --AND tests
   ("10101010", "01010101", '1', "00000000", '1'), --AND tests
   ("11111111", "11111111", '0', "11111110", '1')  --XOR tests
  );
begin
  --  Check each pattern.
  for i in patterns'range loop
    --  Set the inputs.
    A   <= patterns(i).A;
    B   <= patterns(i).B;
    Cin <= patterns(i).Cin;
    --  Wait for the results.
    wait for 1 ns;
    --  Check the outputs.
    assert Sum = patterns(i).Sum
    report "The sum check failed" severity error;
    assert Cout = patterns(i).Cout
    report "The carry out is wrong" severity error;
  end loop;
  assert false report "end of test" severity note;
  --  Wait forever; this will finish the simulation.
  wait;
end process;
end behav;
```

Listing 26: processor/spr_tb.vhd

```
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_unsigned.all;
--use ieee.std_logic_arith.all;

entity spr_TB is      -- entity declaration
  end spr_TB;

architecture TB of spr_TB is

  component sr
```

```
13    Port (clk      : in   STD_LOGIC;
          enable   : in   STD_LOGIC;                          -- Enable write
15        reset    : in   STD_LOGIC;                          -- Reset the register
          Ri       : in   STD_LOGIC_VECTOR (15 downto 0);   -- The input to the SPR
17        Ro       : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SPR
    end component;
19
    signal sr_enable : std_logic;
21  signal sr_reset  : std_logic;
    signal sr_Ri     : std_logic_vector(15 downto 0);
23  signal sr_Ro     : std_logic_vector(15 downto 0);

25  component pc
    Port (clk      : in   STD_LOGIC;
27        enable   : in   STD_LOGIC;                          -- Enable write
          reset    : in   STD_LOGIC;                          -- Reset the register
29        Ri       : in   STD_LOGIC_VECTOR (15 downto 0);   -- The input to the SPR
          Ro       : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SPR
31  end component;

33  signal pc_enable : std_logic;
    signal pc_reset  : std_logic;
35  signal pc_Ri     : std_logic_vector(15 downto 0);
    signal pc_Ro     : std_logic_vector(15 downto 0);
37
    signal T_clk  : std_logic;
39
begin
41
    U_sr: sr port map (clk => T_clk, enable => sr_enable, reset => sr_reset, Ri =>
        sr_Ri, Ro => sr_Ro);
43  U_pc: pc port map (clk => T_clk, enable => pc_enable, reset => pc_reset, Ri =>
        pc_Ri, Ro => pc_Ro);

45    -- concurrent process to offer the clk signal
    process
47  begin
      T_clk <= '0';
49    wait for 5 ns;
      T_clk <= '1';
51    wait for 5 ns;
    end process;
53
    process
55
      variable err_cnt: integer :=0;
57
    begin
59
      -- Write
61    sr_enable <= '1';
      sr_reset  <= '0';
63    sr_Ri     <= "0100011001011001";
      pc_enable <= '1';
65    pc_reset  <= '0';
      pc_Ri     <= "0101011010110100";
67    wait for 20 ns;

69    -- Read
      assert (sr_Ro="0100011001011001") report "Read sr #1 failed" severity error;
71    assert (pc_Ro="0101011010110100") report "Read pc #1 failed" severity error;

73    -- Change Ri
      sr_Ri <= "1001100101110100";
75    pc_Ri <= "0001010001110000";
      wait for 20 ns;
77    assert (sr_Ro = "1001100101110100") report "Read sr #2 failed" severity error;
      assert (pc_Ro = "0001010011110000") report "Read pc #2 failed" severity error;
79
      -- Disable sr, pc still enabled
81    sr_enable <= '0';
      sr_Ri <= "0101010101010101";
83    pc_Ri <= "1010101010101010";
```

```
    wait for 20 ns;
    assert (sr_Ro = "1001100101110100") report "Wrote to sr while disabled" severity
        error;
    assert (pc_Ro = "1010101010101010") report "Read pc #3 failed" severity error;

    -- Enable sr
    sr_enable <= '1';
    wait for 20 ns;
    assert (sr_Ro = "0101010101010101") report "Read sr #3 failed" severity error;

    -- Disable pc, sr still enabled
    pc_enable <= '0';
    sr_Ri <= "0000000011111111";
    pc_Ri <= "1111111100000000";
    wait for 20 ns;
    assert (sr_Ro = "0000000011111111") report "Read sr #4 failed" severity error;
    assert (pc_Ro = "1010101010101010") report "Wrote to pc while disabled" severity
        error;

    -- Enable pc
    pc_enable <= '1';
    wait for 20 ns;
    assert (pc_Ro = "1111111100000000") report "Read pc #4 failed" severity error;


    assert false report "End of test" severity note;
    wait; -- wait forever to end the test

  end process;

end TB;

--------------------------------------------------------------------------
configuration CFG_TB of spr_TB is
  for TB
  end for;
end CFG_TB;
```

Listing 27: processor/gpr_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--  A testbench has no ports.
entity gpr_tb is
  end gpr_tb;

architecture behav of gpr_tb is
  --  Declaration of the component that will be instantiated.
  component gpr
  Port(clk                  : IN std_logic;                     -- Clock
       enable               : IN std_logic;                     -- Enable input
           (output is always enabled)
       SelRx, SelRy, SelRi  : IN std_logic_vector(2 DOWNTO 0);  -- Selecti which
           registers to use
       Ri                   : IN std_logic_vector(7 DOWNTO 0);  -- Input
       Rx, Ry               : OUT std_logic_vector(7 DOWNTO 0)); -- Outputs
  end component;
  --  Specifies which entity is bound with the component.
  for gpr_0: gpr use entity work.gpr;
  signal clk, enable          : std_logic;
  signal SelRx, SelRy, SelRi  : std_logic_vector(2 DOWNTO 0);
  signal Ri, Rx, Ry           : std_logic_vector(7 DOWNTO 0);
begin
  --  Component instantiation.
  gpr_0: gpr port map (clk => clk, enable => enable, SelRx => SelRx, SelRy => SelRy,
      SelRi => SelRi, Ri => Ri, Rx => Rx, Ry => Ry);

  -- Does the clock signal
  process
  begin
    clk <= '0';
```

```
30      wait for 5 ns;
        clk <= '1';
32      wait for 5 ns;
    end process;
34
    --  This process does the real job.
36  process
    begin
38
        -- Write to R0
40      SelRi <= "000";
        Ri <= "00010100";
42      enable <= '1';
        wait for 20 ns;
44
        -- Read R0 from Rx
46      SelRx <= "000";
        wait for 20 ns;
48      assert (Rx = "00010100") report "Read from Rx failed #1" severity error;

50      -- Read R0 from Ry
        SelRy <= "000";
52      wait for 20 ns;
        assert (Ry = "00010100") report "Read from Ry failed #1" severity error;
54
        -- Disable write
56      enable <= '0';
        wait for 20 ns;
58
        -- Change Ri (should not write as it is disabled)
60      Ri <= "00101010";
        wait for 20 ns;
62      assert (Rx = "00010100") report "Wrote to register while disabled #1" severity
            error;
        assert (Ry = "00010100") report "Wrote to register while disabled #2" severity
            error;
64
        -- Enable write
66      enable <= '1';
        wait for 20 ns;
68      assert (Rx = "00101010") report "Read from Rx failed #2" severity error;
        assert (Ry = "00101010") report "Read from Ry failed #2" severity error;
70
        -- Write to R2
72      SelRi <= "010";
        Ri <= "01010001";
74      wait for 20 ns;

76      -- Read R2 from Rx
        SelRx <= "010";
78      wait for 20 ns;
        assert (Rx = "01010001") report "Read from Rx failed #3" severity error;
80
        -- Read R2 from Ry
82      SelRy <= "010";
        wait for 20 ns;
84      assert (Ry = "01010001") report "Read from Ry failed #3" severity error;

86      -- Read R0 from Rx again (should not have changed from previous results)
        SelRx <= "000";
88      wait for 20 ns;
        assert (Rx = "00101010") report "Read from Rx failed #4" severity error;
90
        -- Read R0 from Ry again (should not have changed from previous results)
92      SelRy <= "000";
        wait for 20 ns;
94      assert (Ry = "00101010") report "Read from Ry failed #4" severity error;

96      -- Wait for a long time
        wait for 1 ms;
98      assert (Rx = "00101010") report "Read from Rx failed #5" severity error;
        assert (Ry = "00101010") report "Read from Ry failed #5" severity error;
100
```

```
    -- Read R2 after a long time
102 SelRx <= "010";
    SelRy <= "010";
104 wait for 1 ms;
    assert (Rx = "01010001") report "Read from Rx failed #6" severity error;
106 assert (Ry = "01010001") report "Read from Ry failed #6" severity error;

108
    assert false report "End of test" severity note;
110 wait; -- wait forever to end the test

112   end process;
  end behav;
```

Listing 28: mmu/mmu_tb.vhd

```
  -- Author:
2 --       Wim Looman

4 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
6
  library work;
8 use work.mmu_main;
  use work.minimal_uart_core;
10 use work.txt_util.all;

12 entity mmu_tb is
  end mmu_tb;
14
  architecture tb of mmu_tb is
16   component mmu_main is
       port (
18       -- instruction bus
         inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
20       inst_data : out std_logic_vector(15 downto 0); -- Data lines.
         inst_req  : in  std_logic;                     -- Pulled low to request bus
             usage.
22       inst_ack  : out std_logic;                     -- Pulled high to inform of
             request completion.
         -- data bus
24       data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
         data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
26       data_read : in    std_logic;                     -- High for a read request,
             low for a write request.
         data_req  : in    std_logic;                     -- Pulled low to request bus
             usage.
28       data_ack  : inout std_logic;                     -- Pulled high to inform of
             request completion.
         -- extras
30       clk          : in  std_logic;
         receive_pin  : in  std_logic;
32       transfer_pin : out std_logic
       );
34   end component;

36   component minimal_uart_core is
       port(
38       clock : in    std_logic;
         eoc   : out   std_logic;
40       outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
         rxd   : in    std_logic;
42       txd   : out   std_logic;
         eot   : out   std_logic;
44       inp   : in    std_logic_vector(7 downto 0);
         ready : out   std_logic;
46       wr    : in    std_logic
       );
48   end component;

50   signal inst_add     : std_logic_vector(11 downto 0);
     signal inst_data    : std_logic_vector(15 downto 0);
```

```
  signal inst_req     : std_logic := '1';
  signal inst_ack     : std_logic;
  signal data_add     : std_logic_vector(15 downto 0);
  signal data_line    : std_logic_vector(7 downto 0);
  signal data_read    : std_logic;
  signal data_req     : std_logic;
  signal data_ack     : std_logic;
  signal clk          : std_logic;
  signal receive_pin  : std_logic;
  signal transfer_pin : std_logic;

  signal eoc, rxd, txd, eot, ready, wr: std_logic;
  signal outp, inp : std_logic_vector(7 downto 0);

  signal current_recv : std_logic_vector(7 downto 0);
  signal current_send : std_logic_vector(7 downto 0);
begin
  m : mmu_main port map (inst_add, inst_data, inst_req, inst_ack, data_add,
      data_line, data_read, data_req, data_ack, clk, receive_pin, transfer_pin);
  muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);

  rxd <= transfer_pin;
  receive_pin <= txd;

  clk_gen : process begin
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
  end process;

  inst_test : process
    type pattern_type is record
      inst_add      : std_logic_vector(11 downto 0);
      recv_head     : std_logic_vector( 7 downto 0);
      send_head     : std_logic_vector( 7 downto 0);
      inst_data     : std_logic_vector(15 downto 0);
    end record;
    type pattern_array is array (natural range <>) of pattern_type;
    constant patterns : pattern_array :=
--      inst_add       recv_head   send_head   inst_data
    ((x"52E", x"81", x"83", x"83A7"),
     (x"96F", x"81", x"83", x"4F5E"),
     (x"8F1", x"81", x"83", x"5937"),
     (x"65A", x"81", x"83", x"A8F2"));
  begin
    wr <= '0';
    for i in patterns'range loop
      wait for 10000 ns;
      inst_add <= patterns(i).inst_add;
      wait for 20 ns;
      inst_req <= '0';

      wait until eoc'event;
      assert outp = patterns(i).recv_head
        report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved '"
            & str(outp) & "'"
        severity error;
      wait until eoc'event;

      assert false report "passed header" severity note;

      wait until eoc'event;
      assert outp = patterns(i).inst_add(7 downto 0)
        report "Bad address low expected '" & str(patterns(i).inst_add(7 downto 0)) &
             "' recieved '" & str(outp) & "'"
        severity error;
      wait until eoc'event;

      assert false report "passed address low" severity note;

      wait until eoc'event;
      assert outp = "0000" & patterns(i).inst_add(11 downto 8)
```

```
122         report "Bad address high expected '" & str(patterns(i).inst_add(11 downto
                8)) & "' recieved '" & str(outp) & "'"
            severity error;
124       wait until eoc'event;

126       assert false report "passed address high" severity note;

128       wait for 100 ns;
          inp <= patterns(i).send_head;
130       wait for 20 ns;
          wr <= '1';
132       wait for 20 ns;
          wr <= '0';
134       wait until eot'event;
          wait until eot'event;
136

138       wait for 100 ns;
          inp <= "0000" & patterns(i).inst_add(11 downto 8);
140       wait for 20 ns;
          wr <= '1';
142       wait for 20 ns;
          wr <= '0';
144       wait until eot'event;
          wait until eot'event;
146

148       wait for 100 ns;
          inp <= patterns(i).inst_add(7 downto 0);
150       wait for 20 ns;
          wr <= '1';
152       wait for 20 ns;
          wr <= '0';
154       wait until eot'event;
          wait until eot'event;
156

158       wait for 100 ns;
          inp <= patterns(i).inst_data(7 downto 0);
160       wait for 20 ns;
          wr <= '1';
162       wait for 20 ns;
          wr <= '0';
164       wait until eot'event;
          wait until eot'event;
166

168       wait for 100 ns;
          inp <= patterns(i).inst_data(15 downto 8);
170       wait for 20 ns;
          wr <= '1';
172       wait for 20 ns;
          wr <= '0';
174       wait until eot'event;
          wait until eot'event;
176
          assert inst_ack = '1'
178         report "receipt not acknowledged"
            severity error;
180
          assert inst_data = patterns(i).inst_data
182         report "Wrong data recieve expected '" & str(patterns(i).inst_data) & "'
                recieved '" & str(inst_data) & "'"
            severity error;
184
          assert false report "finished transmission" severity note;
186
          wait for 20 ns;
188
          inp_req <= '1';
190     end loop;
        wait;
192   end process;
```

```
end tb;
```

Listing 29: data_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.mmu_main;
use work.minimal_uart_core;
use work.txt_util.all;

entity data_tb is
end data_tb;

architecture tb of data_tb is
  component mmu_main is
    port (
      -- instruction bus
      inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
      inst_data : out std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : in  std_logic;                     -- Pulled low to request bus
          usage.
      inst_ack  : out std_logic;                     -- Pulled high to inform of
          request completion.
      -- data bus
      data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read : in    std_logic;                      -- High for a read request,
          low for a write request.
      data_req  : in    std_logic;                      -- Pulled low to request bus
          usage.
      data_ack  : inout std_logic;                      -- Pulled high to inform of
          request completion.
      -- extras
      clk          : in  std_logic;
      receive_pin  : in  std_logic;
      transfer_pin : out std_logic
    );
  end component;

  component IO is
    PORT(
      -- data bus --
      data_add    : IN      std_logic_vector(15 DOWNTO 0);    -- address lines --
      data_data : INOUT     std_logic_vector(7 DOWNTO 0);   -- data lines --
      data_read : INOUT     std_logic;                -- pulled high for read, low
          for write --
      data_req    : INOUT   std_logic;                -- pulled low to request bus
          usage --
      data_ack    : INOUT   std_logic;                -- pulled high to inform
          request completion --
      -- io --
      clk         : IN      std_logic;
      sw1         : IN      std_logic;
      sw2         : IN      std_logic
    );
  end component;

  component minimal_uart_core is
    port(
      clock : in    std_logic;
      eoc   : out   std_logic;
      outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
      rxd   : in    std_logic;
      txd   : out   std_logic;
      eot   : out   std_logic;
      inp   : in    std_logic_vector(7 downto 0);
      ready : out   std_logic;
      wr    : in    std_logic
    );
  end component;
```

```
61
    signal inst_add     : std_logic_vector(11 downto 0);
63  signal inst_data    : std_logic_vector(15 downto 0);
    signal inst_req     : std_logic := '1';
65  signal inst_ack     : std_logic;
    signal data_add     : std_logic_vector(15 downto 0);
67  signal data_line    : std_logic_vector(7 downto 0);
    signal data_read    : std_logic;
69  signal data_req     : std_logic;
    signal data_ack     : std_logic;
71  signal clk          : std_logic;
    signal receive_pin  : std_logic;
73  signal transfer_pin : std_logic;
    signal sw1, sw2 : std_logic;
75
    signal eoc, rxd, txd, eot, ready, wr: std_logic;
77  signal outp, inp : std_logic_vector(7 downto 0);
begin
79  m : mmu_main port map (inst_add, inst_data, inst_req, inst_ack, data_add,
        data_line, data_read, data_req, data_ack, clk, receive_pin, transfer_pin);
    i : IO       port map (data_add, data_line, data_read, data_req, data_ack, clk,
        sw1, sw2);
81  muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);

83  rxd <= transfer_pin;
    receive_pin <= txd;
85
    clk_gen : process begin
87    clk <= '0';
      wait for 10 ns;
89    clk <= '1';
      wait for 10 ns;
91  end process;

93  data_test : process
      type pattern_type is record
95      data_add      : std_logic_vector(15 downto 0);
        recv_head     : std_logic_vector( 7 downto 0);
97      send_head     : std_logic_vector( 7 downto 0);
        data_data     : std_logic_vector( 7 downto 0);
99      switch_data   : std_logic_vector( 1 downto 0);
        rw            : std_logic;
101   end record;
      type pattern_array is array (natural range <>) of pattern_type;
103   constant patterns : pattern_array :=
--  data_add, recv_head, send_head, data_data, switch_data, rw
105   ((x"0581", x"80", x"00", x"A7", "00", '1' ),
       (x"0273", x"00", x"00", x"5E", "00", '0' ));
107 begin
      wr <= '0';
109   data_req <= '1';
      for i in patterns'range loop
111     wait for 10000 ns;
        data_add <= patterns(i).data_add;
113     data_read <= patterns(i).rw;
        wait for 20 ns;
115     if patterns(i).rw = '0' then
          data_line <= patterns(1).data_data;
117     else
          data_line <= (others => 'Z');
119     end if;
        wait for 20 ns;
121     data_req <= '0';

123     if patterns(i).data_add(0) = '1' then
          wait until eoc'event;
125       assert outp = patterns(i).recv_head
            report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved
                '" & str(outp) & "'"
127         severity error;
          wait until eoc'event;
129
          assert false report "passed header" severity note;
```

```
131
          wait until eoc'event;
133       assert outp = patterns(i).data_add(8 downto 1)
            report "Bad address low expected '" & str(patterns(i).data_add(7 downto
                0)) & "' recieved '" & str(outp) & "'"
135         severity error;
          wait until eoc'event;
137
          assert false report "passed address low" severity note;
139
          wait until eoc'event;
141       assert outp = "0" & patterns(i).data_add(15 downto 9)
            report "Bad address high expected '" & str(patterns(i).data_add(11 downto
                8)) & "' recieved '" & str(outp) & "'"
143         severity error;
          wait until eoc'event;
145
          assert false report "passed address high" severity note;
147       if patterns(i).rw = '0' then
            wait until eoc'event;
149         assert outp = patterns(i).data_data
              report "Bad data expected '" & str(patterns(i).data_data) & "' recieved
                  '" & str(outp) & "'"
151           severity error;
            wait until eoc'event;
153
            assert false report "passed data" severity note;
155       else
            wait for 100 ns;
157         inp <= patterns(i).send_head;
            wait for 20 ns;
159         wr <= '1';
            wait for 20 ns;
161         wr <= '0';
            wait until eot'event;
163         wait until eot'event;

165
            wait for 100 ns;
167         inp <= patterns(i).data_add(8 downto 1);
            wait for 20 ns;
169         wr <= '1';
            wait for 20 ns;
171         wr <= '0';
            wait until eot'event;
173         wait until eot'event;

175
            wait for 100 ns;
177         inp <=  "0" & patterns(i).data_add(15 downto 9);
            wait for 20 ns;
179         wr <= '1';
            wait for 20 ns;
181         wr <= '0';
            wait until eot'event;
183         wait until eot'event;

185
            wait for 100 ns;
187         inp <= patterns(i).data_data;
            wait for 20 ns;
189         wr <= '1';
            wait for 20 ns;
191         wr <= '0';
            wait until eot'event;
193         wait until eot'event;
          end if;
195     else

197     end if;

199     assert data_ack = '1'
          report "receipt not acknowledged"
```

```
201           severity error;

203         if  patterns(i).rw = '1' then
              assert data_line = patterns(i).data_data
205             report "Wrong data recieve expected '" & str(patterns(i).data_data) & "'
                    recieved '" & str(data_line) & "'"
                severity error;
207         end if;

209         assert false report "finished transmission" severity note;

211         wait for 20 ns;

213         data_req <= '1';
          end loop;
215       wait;
        end process;
217   end tb;
```

Tools

Listing 30: assembler.rb

```
#!/usr/bin/env ruby

# Author:
#       Wim Looman
# Copyright:
#       Copyright (c) 2010 Wim Looman
# License:
#       GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)

def assert(error=nil)
  raise (error || "Assertion Failed!") unless yield
end

# For 8-bit twos complement
def twos_complement(num)
  return 256 + num
end


def logical_operands(chunks)
      y = chunks[1][1..1].to_i
      assert(chunks[1] + " is not a valid register") {y >= 0 && y < 8}
      x = chunks[2][1..1].to_i
      assert(chunks[2] + " is not a valid register") {x >= 0 && x < 8}
      return (y << 5) + x
end

def immediate(chunk, symbols=nil, move_from=nil)
  v = chunk.to_i
  if v == 0 && chunk != "0" && symbols != nil
    assert(chunk + " is not a valid symbol") {symbols.include?(chunk)}
    move_to = symbols[chunk]
    diff = move_to - move_from
    if diff < 1
      diff = twos_complement(diff)
    end
    return diff
  else
    assert(chunk + " is not a valid immediate") {v >= -127 && v < 128}
    if v < 0
      v = twos_complement(v)
    end
    return v
  end
end

def register(chunk, num_registers)
  x = chunk[1..1].to_i
  assert(chunk + " is not a valid register") {x >=0 && x < num_registers}
  return x
end

def auto(chunk)
  if chunk[-1..-1] == "+"
    return 0x08
  elsif chunk[-1..-1] == "-"
    return 0x10
  else
    return 0x00
  end
end

def convert(lines)
  table = first_pass(lines)
```

```
69    return second_pass(lines, table)
  end
71
73  def first_pass(lines)
      instruction = 0
75    symbols = {}
      lines.each do |line|
77      chunks = line.sub(",", " ").split
        case chunks[0]
79        when "LDI", "LD", "STI", "ST", "MV", "AND", "OR", "NOT", "XOR", "ADD", "ADC",
              "SUB", "SBB", "NEG", "CMP", "BEQ", "BNE", "BLT", "BGT", "BC", "BNC", "RJMP",
               "JMP"
            instruction += 1
81
          when "label:"
83          symbols[chunks[1]] = instruction
        end
85    end
      return symbols
87  end
89
  def second_pass(lines, symbols)
91    line_no = 0
      output = []
93    lines.each do |line|
        label = line.sub(",", " ").split[0]
95      case label
          when "LDI", "LD", "STI", "ST", "MV", "AND", "OR", "NOT", "XOR", "ADD", "ADC",
              "SUB", "SBB", "NEG", "CMP", "BEQ", "BNE", "BLT", "BGT", "BC", "BNC", "RJMP",
               "JMP"
97          line_no += 1
            output.push(convert_line(line, symbols, line_no))
99      end
      end
101   return output.flatten
  end
103
105 def convert_line(line, symbols, line_no)
      chunks = line.sub(",", " ").split#.partition(";")[0].split
107
      case chunks[0]
109     when "LDI"
          instruction = 0x21
111       x = register(chunks[1], 4)
          v = immediate(chunks[2])
113       operands = (v << 2) + x

115     when "LD"
          instruction = 0x01 + auto(chunks[2])
117       x = register(chunks[1], 8)
          y = register(chunks[2], 3)
119       operands = (y << 5) + x

121     when "STI"
          instruction = 0x25
123       y = register(chunks[1], 3)
          v = immediate(chunks[2])
125       operands = (v << 2) + y

127     when "ST"
          instruction = 0x05 + auto(chunks[1])
129       y = register(chunks[1], 3)
          x = register(chunks[2], 8)
131       operands = (y << 5) + x

133     when "MV"
          instruction = 0x04
135       if chunks[1][0] == 'r'[0] && chunks[2][0] == 'r'[0]
            y = register(chunks[1], 8)
137         x = register(chunks[2], 8)
```

```
        operands = (y << 5) + x
      elsif chunks[1][0] == 'a'[0]
        y = register(chunks[1], 3)
        x = register(chunks[2], 8)
        n = chunks[1][-1] == 'H' ? 1 : 0
        operands = (1 << 9) + (n << 8) + (y << 5) + x
      elsif chunks[2][0] == 'a'[0]
        y = register(chunks[1], 8)
        x = register(chunks[2], 3)
        n = chunks[2][-1] == 'H' ? 1 : 0
        operands = (1 << 4) + (n << 3) + (y << 5) + x
      else
        # explode
      end

    when "AND"
      instruction = 0x02
      operands = logical_operands(chunks)

    when "OR"
      instruction = 0x06
      operands = logical_operands(chunks)

    when "NOT"
      instruction = 0x0A
      operands = logical_operands(chunks)

    when "XOR"
      instruction = 0x0E
      operands = logical_operands(chunks)

    when "ADD"
      instruction = 0x12
      operands = logical_operands(chunks)

    when "ADC"
      instruction = 0x16
      operands = logical_operands(chunks)

    when "SUB"
      instruction = 0x1A
      operands = logical_operands(chunks)

    when "SBB"
      instruction = 0x1E
      operands = logical_operands(chunks)

    when "NEG"
      instruction = 0x08
      operands = logical_operands(chunks)

    when "CMP"
      instruction = 0x0C
      operands = logical_operands(chunks)

    when "BEQ"
      instruction = 0x23
      v = immediate(chunks[1], symbols, line_no)
      operands = (v << 2)

    when "BNE"
      instruction = 0x27
      v = immediate(chunks[1], symbols, line_no)
      operands = (v << 2)

    when "BLT"
      instruction = 0x2B
      v = immediate(chunks[1], symbols, line_no)
      operands = (v << 2)

    when "BGT"
      instruction = 0x2F
      v = immediate(chunks[1], symbols, line_no)
```

```
211       operands = (v << 2)

213     when "BC"
          instruction = 0x33
215       v = immediate(chunks[1], symbols, line_no)
          operands = (v << 2)
217
        when "BNC"
219       instruction = 0x37
          v = immediate(chunks[1], symbols, line_no)
221       operands = (v << 2)

223     when "RJMP"
          instruction = 0x3B
225       v = immediate(chunks[1], symbols, line_no)
          operands = (v << 2)
227
        when "JMP"
229       instruction = 0x2F
          y = register(chunks[1], 3)
231       operands = (y << 5)
      end
233   opcode = (instruction << 10) + operands
      return [(opcode >> 8), (opcode & 0xFF)]
235 end

237
    if __FILE__ == $0
239   if !(1..2).include?(ARGV.length) || !File.exist?(ARGV[0])
        p "Usage: ruby #{$0} <input_file> [<output_file>]"
241     exit
      end
243
      input = IO.readlines(ARGV[0])
245
      output = convert(input)
247   if ARGV.length == 2:
        File.open(ARGV[1], "wb") do |file|
249       output.each do |char|
            file.putc(char)
251       end
        end
253   else
        output.each do |char|
255       $stdout.putc(char)
        end
257   end
    end
```

Listing 31: memory.rb

```
    #!/usr/bin/env ruby
 2
    # Author:
 4  #       Wim Looman
    # Copyright:
 6  #       Copyright (c) 2010 Wim Looman
    # License:
 8  #       GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)

10  require 'rubygems'
    require 'serialport'
12
    def serve(program, data_file, sp)
14    while 1
        header = sp.getc
16      diagnostic_mode = (header >> 2) & 0x03
        instruction = header & 0x01 == 0x01
18      address = sp.getc + (sp.getc << 8)

20      case (header >> 7) & 0x01 # read/write bit
          when 0x01               # read
```

```
22          header = 0x82
            header += 0x01 if instruction
24          sp.putc(header)
            instruction ? sp.write(program[address]) :
26                        sp.write(data_file[address])
            p "Sending data for " + instruction ? "Instruction" : "Data" + " bus,
                address: " + address + ", data: " + instruction ? program[address] :
                data_file[address]
28        when 0x00               # write, doesn't support writing to instruction memory
            data = sp.getc
30          data_file[address] = data
            p "Writing data, address: " + address + ", data: " + data_file[address]
32      end
    end
34  end

36  if __FILE__ == $0

38    if ARGV.size < 3
        STDERR.print "Usage: ruby #{$0} <device> <baud_rate> <program_file>
            [<data_file>]\n"
40      exit
      end
42
      device = ARGV[0]
44    baud_rate = ARGV[1].to_i

46    program = Array.new(2**12, 0x00)
      i = 0
48    File.open(ARGV[2], 'rb') do |input|
        input.each_byte do |byte|
50        program[i] =  byte
          i += 1
52      end
      end
54
      data = Array.new(2**15, 0x00)
56    File.open(ARGV[3], 'rb') do |input|
        input.each_byte do |byte|
58        data += byte
        end
60    end if ARGV.size > 3

62    sp = SerialPort.new(device, baud_rate, 8, 1, SerialPort::NONE)

64    serve(program, data, sp)
    end
```