Listing 1: data_tb.vhd

```
1  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
3
   library work;
5  use work.mmu_main;
   use work.minimal_uart_core;
7  use work.txt_util.all;

9  entity data_tb is
   end data_tb;
11
   architecture tb of data_tb is
13   component mmu_main is
       port (
15       -- instruction bus
         inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
17       inst_data : out std_logic_vector(15 downto 0); -- Data lines.
         inst_req  : in  std_logic;                     -- Pulled low to request bus
             usage.
19       inst_ack  : out std_logic;                     -- Pulled high to inform of
             request completion.
         -- data bus
21       data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
         data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
23       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.
25       data_ack  : inout std_logic;                     -- Pulled high to inform of
             request completion.
         -- extras
27       clk          : in  std_logic;
         receive_pin  : in  std_logic;
29       transfer_pin : out std_logic
       );
31   end component;

33   component IO is
       PORT(
35       -- data bus --
         data_add    : IN      std_logic_vector(15 DOWNTO 0);    -- address lines --
37       data_data : INOUT     std_logic_vector(7 DOWNTO 0);   -- data lines --
         data_read : INOUT     std_logic;                 -- pulled high for read, low
             for write --
39       data_req    : INOUT   std_logic;                 -- pulled low to request bus
             usage --
         data_ack    : INOUT   std_logic;                 -- pulled high to inform
             request completion --
41       -- io --
         clk         : IN      std_logic;
43       sw1         : IN      std_logic;
         sw2         : IN      std_logic
45     );
   end component;
47
   component minimal_uart_core is
49   port(
       clock : in    std_logic;
51     eoc   : out   std_logic;
       outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
53     rxd   : in    std_logic;
       txd   : out   std_logic;
55     eot   : out   std_logic;
       inp   : in    std_logic_vector(7 downto 0);
57     ready : out   std_logic;
       wr    : in    std_logic
59   );
   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';
    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 sw1, sw2 : std_logic;

    signal eoc, rxd, txd, eot, ready, wr: std_logic;
    signal outp, inp : 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);
    i : IO       port map (data_add, data_line, data_read, data_req, data_ack, clk,
        sw1, sw2);
    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;

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

        if patterns(i).data_add(0) = '1' then
          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).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;
```

Listing 2: everything.vhd

```
1  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
3
   library work;
5
   entity everything is
7    port (
          clk    : in  std_logic;
9         reset  : in  std_logic;
          tx     : out std_logic;
11        rx     : in  std_logic;
          sw1    : in  std_logic;
13        sw2    : in  std_logic
     );
15 end everything;

17 architecture everything_arch of everything is
     component cpu IS
19     PORT(
       -- instruction bus
21     inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
       inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
23     inst_req  : out std_logic;                     -- Pulled low to request bus
           usage.
       inst_ack  : in  std_logic;                     -- Pulled high to inform of
           request completion.
25     -- data bus
       data_add  : out    std_logic_vector(15 downto 0); -- Address lines.
27     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.
29     data_req  : out    std_logic;                     -- Pulled low to request bus
           usage.
       data_ack  : inout std_logic;                     -- Pulled high to inform of
           request completion.
31     -- extras
       clk       : in  std_logic;
33     reset     : in  std_logic
     );
35   end component;
     component mmu_main is
37     port (
       -- instruction bus
39     inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
       inst_data : out std_logic_vector(15 downto 0); -- Data lines.
41     inst_req  : in  std_logic;                     -- Pulled low to request bus
           usage.
       inst_ack  : out std_logic;                     -- Pulled high to inform of
           request completion.
43     -- data bus
       data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
45     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.
```

```
47      data_req  : in    std_logic;                    -- Pulled low to request bus
            usage.
        data_ack  : inout std_logic;                    -- Pulled high to inform of
            request completion.
49      -- extras
        clk          : in  std_logic;
51      receive_pin  : in  std_logic;
        transfer_pin : out std_logic
53    );
    END component;
55  component IO is
         PORT(
57            -- data bus --
              data_add    : IN      std_logic_vector(15 DOWNTO 0); -- address lines --
59            data_data   : INOUT   std_logic_vector(7 DOWNTO 0);  -- data lines --
              data_read   : INOUT   std_logic;                     -- pulled high for
                  read, low for write --
61            data_req    : INOUT   std_logic;                     -- pulled low to
                  request bus usage --
              data_ack    : INOUT   std_logic;                     -- pulled high to
                  inform request completion --
63            -- io --
              clk         : IN      std_logic;
65            sw1         : IN      std_logic;
              sw2         : IN      std_logic);
67            --leds      : OUT std_logic_vector(7 DOWNTO 0);
    END component;
69  -- instruction bus
    signal inst_add  : std_logic_vector(11 downto 0); -- Address lines.
71  signal inst_data : std_logic_vector(15 downto 0); -- Data lines.
    signal inst_req  : std_logic;                     -- Pulled low to request bus
        usage.
73  signal inst_ack  : std_logic;                     -- Pulled high to inform of
        request completion.
    -- data bus
75  signal data_add  : std_logic_vector(15 downto 0); -- Address lines.
    signal data_line : std_logic_vector(7 downto 0);  -- Data lines.
77  signal data_read : std_logic;                     -- High for a read request, low
        for a write request.
    signal data_req  : std_logic;                     -- Pulled low to request bus
        usage.
79  signal data_ack  : std_logic;                     -- Pulled high to inform of
        request completion.

81  begin
      c : cpu port map(
83      -- instruction bus
        inst_add  => inst_add, -- Instruction address
85      inst_data => inst_data,-- Instruction data
        inst_req  => inst_req, -- Request
87      inst_ack  => inst_ack, -- Instruction obtained
        -- data bus
89      data_add  => data_add, -- Data address
        data_line => data_line,-- Data
91      data_read => data_read,-- 1 for read, 0 for write
        data_req  => data_req, -- Request
93      data_ack  => data_ack, -- Data written to/ read from
        -- extras
95      clk       => clk,
        reset     => reset
97      );
      m : mmu_main port map(
99      -- instruction bus
        inst_add     => inst_add,  -- Address lines.
101     inst_data    => inst_data, -- Data lines.
        inst_req     => inst_req,  -- Pulled low to request bus usage.
103     inst_ack     => inst_ack,  -- Pulled high to inform of request completion.
        -- data bus
105     data_add     => data_add,  -- Address lines.
        data_line    => data_line, -- Data lines.
107     data_read    => data_read, -- High for a read request, low for a write request.
        data_req     => data_req,  -- Pulled low to request bus usage.
109     data_ack     => data_ack,  -- Pulled high to inform of request completion.
```

```
        -- extras
111     clk          => clk,
        receive_pin  => rx,
113     transfer_pin => tx
    );
115 i : io  port map(
            clk          => clk,
117         data_add     => data_add,
            data_data    => data_line,
119         data_read    => data_read,
            data_req     => data_req,
121         data_ack     => data_ack,
            -- io --
123         sw1          => sw1,
            sw2          => sw2
125     );
end architecture everything_arch;
```

Listing 3: IO/debounce.vhd

```
    ----------------------------------------------------------------------------------
2   -- Company:
    -- Engineer:
4   --
    -- Create Date:    16:08:33 09/15/2010
6   -- Design Name:
    -- Module Name:    IO - 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  use IEEE.STD_LOGIC_ARITH.ALL;
    use IEEE.STD_LOGIC_UNSIGNED."+";
24
    library work;
26
    ENTITY debounce IS
28      PORT(clk : IN STD_LOGIC;
        switch : IN STD_LOGIC;
30      switch_state : OUT STD_LOGIC);
    END debounce;
32
    ARCHITECTURE debounced_switch OF debounce IS
34    SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);  --variable or signal???

36  BEGIN
        PROCESS(clk, switch)
38      BEGIN
            IF switch = '0' THEN
40              count <= "000";
            ELSIF rising_edge(clk) THEN
42              IF count /= "111" THEN
                    count <= count + 1;
44              END IF;
            END IF;
46          IF count = "111" AND switch = '1' THEN
                switch_state <= '1';
48          ELSE
                switch_state <= '0';
50          END IF;
        END PROCESS;
52  END debounced_switch;
```

Listing 4: IO/IO.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:08:33 09/15/2010
-- Design Name:
-- Module Name:    IO - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision: Saturday 16 Oct 2010 by Sasha
-- 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.debounce;
use work.switch_reg;
use work.led_io;



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

entity 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);
             --leds       : OUT std_logic_vector(7 DOWNTO 0);
end IO;

architecture io of IO is


COMPONENT led_io
    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 COMPONENT;

COMPONENT switch_io IS
    PORT ( data_add         : IN            std_logic_vector(15 DOWNTO 0);
           data_data        : INOUT         std_logic_vector(7 DOWNTO 0);
           data_read        : INOUT         std_logic;
           data_req         : INOUT         std_logic;
           data_ack         : INOUT         std_logic;
           clk              : IN            std_logic;
           sw1              : IN            std_logic;
           sw2              : IN            std_logic
           );
END COMPONENT;


BEGIN

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

----------------------------------------------------------------------------------

END io;
```

Listing 5: IO/leds.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:27:55 10/12/2010
-- Design Name:
-- Module Name:    led_io - 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;
use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;

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

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 --
```

```
38          data_ack    : INOUT   std_logic;                -- pulled high to inform
                request completion --
            --
40          clock     : IN      std_logic;
            led_state :OUT std_logic_vector(7 DOWNTO 0) --just to make diagram work!!!!!
42          );
    END led_io;
44
    ARCHITECTURE led_arch OF led_io IS
46      Signal led_enable   : std_logic;
    --    Signal led_state    : std_logic_vector(7 DOWNTO 0);
48  BEGIN

50      PROCESS(clock, data_req, data_add, data_read)
        BEGIN
52          IF data_req = '0' AND data_add = "0000000000001110" AND data_read = '0' THEN
                led_enable <= '1';
54          ELSE
                led_enable <= '0';
56          END IF;
        END PROCESS;
58

60    PROCESS(clock, led_enable) -- process of read data from the CPU and display LEDS
      BEGIN
62          IF rising_edge(clock) THEN
              IF led_enable = '1' THEN
64                led_state <= data_data;
                  data_ack <= '0';
66            END IF;
            END IF;
68    END PROCESS;

70

72  END led_arch;
```

Listing 6: IO/switch_register.vhd

```
1   ----------------------------------------------------------------------------------
    -- Company:
3   -- Engineer:
    --
5   -- Create Date:    16:08:33 09/15/2010
    -- Design Name:
7   -- Module Name:    IO - Behavioral
    -- Project Name:
9   -- Target Devices:
    -- Tool versions:
11  -- Description:
    --
13  -- Dependencies:
    --
15  -- Revision: Saturday, 16 Oct 2010 by Sasha
    -- Revision 0.01 - File Created
17  -- Additional Comments:
    --
19  ----------------------------------------------------------------------------------
    library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;

25  library work;

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

33  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 7: IO/switches.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:08:33 09/15/2010
-- Design Name:
-- Module Name:    IO - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision: Saturday 16 Oct 2010 by Sasha
-- 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.debounce;
use work.switch_reg;
use work.led_io;


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

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);
            --leds      : OUT std_logic_vector(7 DOWNTO 0);
end switch_io;

architecture Behavioral of switch_io is

```

```
signal enable1                  : std_logic;
signal switch1_connection   : std_logic;
signal switch1_output       : std_logic;
--signal switch1_state      : std_logic;

signal enable2                  : std_logic;
signal switch2_connection   : std_logic;
signal switch2_output       : std_logic;
--signal switch2_state      : 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;


-------------------------------------------------------------------------------------------------

END Behavioral;
```

Listing 8: mmu/control_unit.vhd

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

library work;
use work.mmu_types.all;

entity 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 data.
     ready         : in std_logic; -- ??
     data_read     : in std_logic; -- High if the data line is requesting a read, low
         for write.
     data_req      : in std_logic; -- Low when the data address is valid and should be
          read.
     data_add_0    : in std_logic; -- High for memory address, not IO.
     inst_req      : in std_logic; -- Low when the instruction address is valid and
         should be read.
     fr            : in std_logic; -- High if latest input headers fetch request was
         set.
     inst_or_data_in : in std_logic; -- High if latest input packet was an
         instruction packet.
     rw            : in std_logic; -- High if latest input packet had read/write set.
     write         : out std_logic; -- High to start muart writing data.
     inst_or_data_out : out std_logic; -- High if current output packet is an
         instruction packet.
     inst_ack      : out std_logic; -- Low when the inst is ready to be read by CPU.
         High otherwise.
     data_ack      : inout std_logic; -- Low when the data is ready to be read by CPU.
          High impedance otherwise.
     muart_input   : out muart_input_state; -- State to multiplex the muart's input
     muart_output  : out muart_output_state; -- State to multiplex the muart's output.
    clk     : in  std_logic
  );
end mmu_control_unit;

architecture mmu_control_unit_arch of mmu_control_unit is
  component data_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 data.
     ready         : in std_logic; -- ??
```

```
      data_read    : in std_logic; -- High if the data line is requesting a read, low
          for write.
      data_req     : in std_logic; -- Low when the data address is valid and should be
           read.
      data_add_0   : in std_logic; -- High for memory address, not IO.
      write        : out std_logic; -- High to start muart writing data.
      data_ack     : inout std_logic; -- Low when the data is ready to be read by CPU.
           High impedance otherwise.
      muart_input  : out muart_input_state; -- State to multiplex the muart's input
      muart_output : out muart_output_state; -- State to multiplex the muart's output.
    clk    : in  std_logic
    );
  end component;

  component inst_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 data.
      ready        : in std_logic; -- ??
      inst_req     : in std_logic; -- Low when the instruction address is valid and
          should be read.
      write        : out std_logic; -- High to start muart writing data.
      inst_or_data : out std_logic; -- High if current output packet is an instruction
           packet.
      inst_ack     : out std_logic; -- Low when the inst is ready to be read by CPU.
          High otherwise.
      muart_input  : out muart_input_state; -- State to multiplex the muart's input
      muart_output : out muart_output_state; -- State to multiplex the muart's output.
    clk    : in  std_logic
    );
  end component;

  signal data_write, inst_write, inst_inst_or_data_out : std_logic;
  signal data_muart_input, inst_muart_input : muart_input_state;
  signal data_muart_output, inst_muart_output : muart_output_state;
begin
  data_cu : data_control_unit port map (eoc, eot, ready, data_read, data_req,
      data_add_0, data_write, data_ack, data_muart_input, data_muart_output, clk);
  inst_cu : inst_control_unit port map (eoc, eot, ready, inst_req, inst_write,
      inst_inst_or_data_out, inst_ack, inst_muart_input, inst_muart_output, clk);

  inst_or_data_out  <= inst_inst_or_data_out ;
  write <= inst_write or data_write;
  muart_input  <= inst_muart_input  when inst_inst_or_data_out = '1' else
      data_muart_input;
  muart_output <= inst_muart_output when inst_inst_or_data_out = '1' else
      data_muart_output;
end mmu_control_unit_arch;
```

Listing 9: mmu/data_control_unit.vhd

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

library work;
use work.mmu_types.all;

entity data_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 data.
      ready        : in std_logic; -- ??
      data_read    : in std_logic; -- High if the data line is requesting a read, low
          for write.
      data_req     : in std_logic; -- Low when the data address is valid and should be
           read.
      data_add_0   : in std_logic; -- High for memory address, not IO.
      write        : out std_logic; -- High to start muart writing data.
      data_ack     : inout std_logic; -- Low when the data is ready to be read by CPU.
           High impedance otherwise.
      muart_input  : out muart_input_state; -- State to multiplex the muart's input
      muart_output : out muart_output_state; -- State to multiplex the muart's output.
```

```
    clk    : in  std_logic
  );
end data_control_unit;

architecture data_control_unit_arch of data_control_unit is
  type state_type is (idle, get_data, wait_clear);
  type m_state_type is (idle,
                        send_header, send_add_high, send_add_low, send_data,
                        get_header,  get_add_high,  get_add_low,  get_data,
                        finished);
  type read_state_type is (idle, wait_data, read_data, pause, finished);
  type transmit_state_type is (idle, set_data, trans_data, pause, finished);
  signal state, next_state : state_type := idle;
  signal get_state, next_get_state : m_state_type := idle;
  signal reader_state, next_reader_state : read_state_type := idle;
  signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
begin
  data_fsm : process(state, data_req, clk) begin
    if (rising_edge(clk)) then
      case state is
        when idle =>
          if (data_req = '0' and data_add_0 = '1') then
            next_state <= get_data;
          end if;

        when get_data =>
          if (get_state = finished) then
            next_state <= wait_clear;
          end if;

        when wait_clear =>
          if (data_req = '1') then
            next_state <= idle;
          end if;

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

  get_data_fsm : process(state, clk) begin
    if rising_edge(clk) then
      if state = get_data then
        case get_state is
          when idle =>
            next_get_state <= send_header;

          when send_header =>
            if transmitter_state = finished then
              next_get_state <= send_add_low;
            end if;

          when send_add_low =>
            if transmitter_state = finished then
              next_get_state <= send_add_high;
            end if;

          when send_add_high =>
            if transmitter_state = finished then
              if data_read = '1' then
                next_get_state <= get_header;
              else
                next_get_state <= send_data;
              end if;
            end if;

          when send_data =>
            if transmitter_state = finished then
              next_get_state <= finished;
            end if;

          when get_header =>
```

```
92            if reader_state = finished then
                next_get_state <= get_add_low;
94            end if;

96          when get_add_low =>
              if reader_state = finished then
98              next_get_state <= get_add_high;
              end if;
100
            when get_add_high =>
102           if reader_state = finished then
                next_get_state <= get_data;
104           end if;

106         when get_data =>
              if reader_state = finished then
108             next_get_state <= finished;
              end if;
110
            when finished =>
112           next_get_state <= idle;

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

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

144         when finished =>
              next_transmitter_state <= idle;
146
            when others =>
148           NULL;
          end case;
150     end if;
      end if;
152 end process transmit_fsm;

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

164         when wait_data =>
```

```
            if eoc = '1' then
              next_reader_state <= read_data;
            end if;

          when read_data =>
            next_reader_state <= pause;

          when pause =>
            if eoc = '0' then
              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 and
      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 10: mmu/header_builder.vhd

```
-- 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)
  );
```

```
14 end header_builder;

16 architecture header_builder_arch of header_builder is
   begin
18   header(7) <= read_write or inst_data; -- reading from or writing to
                                             -- memory? (can't write
20                                           -- instructions)
     header(6) <= '0'; -- reserved
22   header(5) <= '0'; -- reserved
     header(4) <= '0'; -- reserved
24   header(3) <= '0'; -- diagnostic
     header(2) <= '0'; -- diagnostic;
26   header(1) <= '0'; --  1 if data is being retrieved from RS-232 link
     header(0) <= inst_data; -- instruction data or data data?
28 end header_builder_arch;
```

Listing 11: mmu/header_decoder.vhd

```
1  -- decodes a header received from the RS-232 link

3  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
5
   library work;
7
   entity header_decoder is
9    port (
       read_write    : out  std_logic; -- 1 = read, 0 = write
11     fetch_request : out std_logic;
       inst_data     : out  std_logic; -- 1 = inst, 0 = data
13     header        : in std_logic_vector(7 downto 0)
     );
15 end header_decoder;

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

Listing 12: mmu/inst_control_unit.vhd

```
1  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
3
   library work;
5  use work.mmu_types.all;

7  entity inst_control_unit is
     port (
9      eoc          : in std_logic; -- High on muart has finished collecting data ??
       eot          : in std_logic; -- High on muart has finished transmitting data.
11     ready        : in std_logic; -- ??
       inst_req     : in std_logic; -- Low when the instruction address is valid and
           should be read.
13     write        : out std_logic; -- High to start muart writing data.
       inst_or_data : out std_logic; -- High if current output packet is an instruction
            packet.
15     inst_ack     : out std_logic; -- Low when the inst is ready to be read by CPU.
           High otherwise.
       muart_input  : out muart_input_state; -- State to multiplex the muart's input
17     muart_output : out muart_output_state; -- State to multiplex the muart's output.
      clk     : in  std_logic
19   );
   end inst_control_unit;
21
   architecture inst_control_unit_arch of inst_control_unit is
23   type state_type is (idle, get_data, wait_clear);
     type m_state_type is (idle,
```

```
25                        send_header,   send_add_high,   send_add_low,
                          get_header,    get_add_high,    get_add_low,
27                        get_data_high, get_data_low,    finished);
    type read_state_type is (idle, wait_data, read_data, pause, finished);
29  type transmit_state_type is (idle, set_data, trans_data, pause, finished);
    signal state, next_state : state_type := idle;
31  signal get_state, next_get_state : m_state_type := idle;
    signal reader_state, next_reader_state : read_state_type := idle;
33  signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
begin
35  inst_fsm : process(state, inst_req, clk) begin
      case state is
37      when idle =>
          if (inst_req = '0') then
39          next_state <= get_data;
          end if;
41
        when get_data =>
43        if (get_state = finished) then
            next_state <= wait_clear;
45        end if;

47      when wait_clear =>
          if (inst_req = '1') then
49          next_state <= idle;
          end if;
51
        when others =>
53        NULL;
      end case;
55  end process inst_fsm;

57  get_inst_fsm : process(state, clk) begin
      if state = get_data then
59      case get_state is
          when idle =>
61          next_get_state <= send_header;

63        when send_header =>
            if transmitter_state = finished then
65            next_get_state <= send_add_low;
            end if;
67
          when send_add_low =>
69          if transmitter_state = finished then
              next_get_state <= send_add_high;
71          end if;

73        when send_add_high =>
            if transmitter_state = finished then
75            next_get_state <= get_header;
            end if;
77
          when get_header =>
79          if reader_state = finished then
              next_get_state <= get_add_low;
81          end if;

83        when get_add_low =>
            if reader_state = finished then
85            next_get_state <= get_add_high;
            end if;
87
          when get_add_high =>
89          if reader_state = finished then
              next_get_state <= get_data_low;
91          end if;

93        when get_data_low =>
            if reader_state = finished then
95            next_get_state <= get_data_high;
            end if;
97
```

```
        when get_data_high =>
          if reader_state = finished then
            next_get_state <= finished;
          end if;

        when finished =>
          next_get_state <= idle;

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

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

        when set_data =>
          next_transmitter_state <= trans_data;

        when trans_data =>
          next_transmitter_state <= pause;

        when pause =>
          if eot = '1' then
            next_transmitter_state <= finished;
          end if;

        when finished =>
          next_transmitter_state <= idle;

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

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

        when wait_data =>
          if eoc = '1' then
            next_reader_state <= read_data;
          end if;

        when read_data =>
          next_reader_state <= pause;

        when pause =>
          if eoc = '0' then
            next_reader_state <= finished;
          end if;

        when finished =>
          next_reader_state <= idle;

        when others =>
          NULL;
      end case;
```

```
171     end if;
    end process read_fsm;
173
    switch_states : process(clk, next_state, next_get_state, next_reader_state,
        next_transmitter_state) begin
175     if rising_edge(clk) then
          state <= next_state;
177       get_state <= next_get_state;
          reader_state <= next_reader_state;
179       transmitter_state <= next_transmitter_state;
        end if;
181   end process switch_states;

183   -- Outputs
    with state select
185     inst_ack <= '1' when wait_clear,
                        '0' when others;
187
    with state select
189     inst_or_data <= '0' when idle,
                            '1' when others;
191
    with transmitter_state select
193     write <= '1' when trans_data,
                     '0' when others;
195

197   muart_input <= idle           when transmitter_state /= set_data and
        transmitter_state /= trans_data else
                       header         when get_state = send_header          else
199                    inst_add_high  when get_state = send_add_high        else
                       inst_add_low   when get_state = send_add_low         else
201                    idle;

203   muart_output <= idle            when reader_state /= read_data    else
                        header          when get_state = get_header     else
205                     inst_data_high  when get_state = get_data_high  else
                        inst_data_low   when get_state = get_data_low   else
207                     idle;
  end inst_control_unit_arch;
```

Listing 13: mmu/mmu.vhd

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

4 library work;
  use work.mmu_types.all;
6 use work.mmu_control_unit;
  use work.header_builder;
8 use work.header_decoder;
  use work.reg8;
10
  use work.minimal_uart_core;
12
    entity mmu_main is
14    port (
        -- instruction bus
16      inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
        inst_data : out std_logic_vector(15 downto 0); -- Data lines.
18      inst_req  : in  std_logic;                     -- Pulled low to request bus
            usage.
        inst_ack  : out std_logic;                     -- Pulled high to inform of
            request completion.
20      -- data bus
        data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
22      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.
24      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 data.
      ready        : in std_logic; -- ??
      data_read    : in std_logic; -- High if the data line is requesting a read,
          low for write.
      data_req     : in std_logic; -- Low when the data address is valid and should
           be read.
      data_add_0   : in std_logic; -- High for memory address, not IO.
      inst_req     : in std_logic; -- Low when the instruction address is valid and
           should be read.
      fr           : in std_logic; -- High if latest input headers fetch request
          was set.
      inst_or_data_in : in std_logic; -- High if latest input packet was an
          instruction packet.
      rw           : in std_logic; -- High if latest input packet had read/write
          set.
      write        : out std_logic; -- High to start muart writing data.
      inst_or_data_out : out std_logic; -- High if current output packet is an
          instruction packet.
      inst_ack     : out std_logic; -- Low when the inst is ready to be read by
          CPU. High otherwise.
      data_ack     : inout std_logic; -- Low when the data is ready to be read by
          CPU. High impedance otherwise.
      muart_input  : out muart_input_state; -- State to multiplex the muart's input
      muart_output : out muart_output_state; -- State to multiplex the muart's
          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;
      inp   : in    std_logic_vector(7 downto 0);
      ready : out   std_logic;
      wr    : in    std_logic
    );
  end component;

```

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

98  signal eoc          : std_logic; -- High on muart has finished collecting data ??
    signal eot          : std_logic; -- High on muart has finished transmitting data.
100 signal ready        : std_logic; -- ??
    signal fr           : std_logic; -- High if latest input headers fetch request
        was set.
102 signal inst_or_data_in : std_logic; -- High if latest input packet was an
        instruction packet.
    signal rw           : std_logic; -- High if latest input packet had read/write
        set.
104
    signal write        : std_logic; -- High to start muart writing data.
106 signal inst_or_data_out : std_logic; -- High if current output packet is an
        instruction packet.
    signal muart_input  : muart_input_state; -- State to multiplex the muart's input
108 signal muart_output : muart_output_state; -- State to multiplex the muart's
        output.

110 signal muart_out : std_logic_vector(7 downto 0);
    signal muart_in  : std_logic_vector(7 downto 0);
112 signal header_in   : std_logic_vector(7 downto 0);
    signal header_out  : std_logic_vector(7 downto 0);
114 signal inst_data_high_enable : std_logic;
    signal inst_data_low_enable : std_logic;
116 signal data_data_enable : std_logic;
    signal data_line_tri : std_logic_vector(7 downto 0);
118
    begin
120 muart : minimal_uart_core port map (clk, eoc, muart_out, receive_pin,
        transfer_pin, eot, muart_in, ready, write);
    cu  : mmu_control_unit port map (eoc, eot, ready, data_read, data_req,
        data_add(0), inst_req, fr, inst_or_data_in, rw, write, inst_or_data_out,
        inst_ack, data_ack, muart_input, muart_output, clk);
122 hb  : header_builder port map (data_read, inst_or_data_out, header_out);
    hd  : header_decoder port map (rw, fr, inst_or_data_in, header_in);
124 idh : reg8 port map (muart_out, clk, inst_data_high_enable, '0', inst_data(15
        downto 8));
    idl : reg8 port map (muart_out, clk, inst_data_low_enable,  '0', inst_data(7
        downto 0));
126 dd  : reg8 port map (muart_out, clk, data_data_enable,      '0', data_line_tri);

128 with muart_input select
      muart_in <=  header_out                              when header,
130                "0000" & inst_add(11 downto 8)          when inst_add_high,
                   inst_add(7  downto 0)                   when inst_add_low,
132                '0' & data_add(15 downto 9)             when data_add_high,
                   data_add(8  downto 1)                   when data_add_low,
134                data_line                               when data_data,
                   (others => '0')                         when others;
136
    route_output : process(muart_output, muart_out) begin
138   header_in <= (others => '0');
      inst_data_high_enable <= '0';
140   inst_data_low_enable <= '0';
      data_data_enable <= '0';
142   case muart_output is
        when header         =>
144       header_in <= muart_out;

146     when inst_data_high =>
          inst_data_high_enable <= '1';
148
        when inst_data_low  =>
```

```
150           inst_data_low_enable <= '1';

152         when data_data      =>
              data_data_enable <= '1';
154         when others =>
              NULL;
156       end case;
      end process;
158
      data_line <= data_line_tri when data_add(0) = '1' and data_read = '1' else
160                  (others => 'Z');

162   end mmu_arch;
```

Listing 14: mmu/mmu_tb.vhd

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

4 library work;
  use work.mmu_main;
6 use work.minimal_uart_core;
  use work.txt_util.all;
8
  entity mmu_tb is
10 end mmu_tb;

12 architecture tb of mmu_tb is
    component mmu_main is
14    port (
        -- instruction bus
16      inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
        inst_data : out std_logic_vector(15 downto 0); -- Data lines.
18      inst_req  : in  std_logic;                     -- Pulled low to request bus
            usage.
        inst_ack  : out std_logic;                     -- Pulled high to inform of
            request completion.
20      -- data bus
        data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
22      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.
24      data_req  : in    std_logic;                     -- Pulled low to request bus
            usage.
        data_ack  : inout std_logic;                     -- Pulled high to inform of
            request completion.
26      -- extras
        clk          : in  std_logic;
28      receive_pin  : in  std_logic;
        transfer_pin : out std_logic
30    );
    end component;
32
    component minimal_uart_core is
34    port(
        clock : in    std_logic;
36      eoc   : out   std_logic;
        outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
38      rxd   : in    std_logic;
        txd   : out   std_logic;
40      eot   : out   std_logic;
        inp   : in    std_logic_vector(7 downto 0);
42      ready : out   std_logic;
        wr    : in    std_logic
44    );
    end component;
46
    signal inst_add     : std_logic_vector(11 downto 0);
48  signal inst_data    : std_logic_vector(15 downto 0);
    signal inst_req     : std_logic := '1';
50  signal inst_ack     : std_logic;
    signal data_add     : std_logic_vector(15 downto 0);
```

```
52    signal data_line    : std_logic_vector(7 downto 0);
      signal data_read    : std_logic;
54    signal data_req     : std_logic;
      signal data_ack     : std_logic;
56    signal clk          : std_logic;
      signal receive_pin  : std_logic;
58    signal transfer_pin : std_logic;

60    signal eoc, rxd, txd, eot, ready, wr: std_logic;
      signal outp, inp : std_logic_vector(7 downto 0);
62
      signal current_recv : std_logic_vector(7 downto 0);
64    signal current_send : std_logic_vector(7 downto 0);
  begin
66    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);
68
      rxd <= transfer_pin;
70    receive_pin <= txd;

72    clk_gen : process begin
        clk <= '0';
74      wait for 10 ns;
        clk <= '1';
76      wait for 10 ns;
      end process;
78
      inst_test : process
80      type pattern_type is record
          inst_add      : std_logic_vector(11 downto 0);
82        recv_head     : std_logic_vector( 7 downto 0);
          send_head     : std_logic_vector( 7 downto 0);
84        inst_data     : std_logic_vector(15 downto 0);
        end record;
86      type pattern_array is array (natural range <>) of pattern_type;
        constant patterns : pattern_array :=
88  --      inst_add       recv_head   send_head   inst_data
        ((x"000", x"81", x"00", x"83A7"),
90       (x"001", x"81", x"00", x"4F5E"),
         (x"101", x"81", x"00", x"5937"),
92       (x"051", x"81", x"00", x"A8F2"));
      begin
94      wr <= '0';
        for i in patterns'range loop
96        wait for 10000 ns;
          inst_add <= patterns(i).inst_add;
98        wait for 20 ns;
          inst_req <= '0';
100
          wait until eoc'event;
102       assert outp = patterns(i).recv_head
            report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved '"
                & str(outp) & "'"
104         severity error;
          wait until eoc'event;
106
          assert false report "passed header" severity note;
108
          wait until eoc'event;
110       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) & "'"
112         severity error;
          wait until eoc'event;
114
          assert false report "passed address low" severity note;
116
          wait until eoc'event;
118       assert outp = "0000" & patterns(i).inst_add(11 downto 8)
            report "Bad address high expected '" & str(patterns(i).inst_add(11 downto
                8)) & "' recieved '" & str(outp) & "'"
120         severity error;
```

```
        wait until eoc'event;
122
        assert false report "passed address high" severity note;
124
        wait for 100 ns;
126     inp <= patterns(i).send_head;
        wait for 20 ns;
128     wr <= '1';
        wait for 20 ns;
130     wr <= '0';
        wait until eot'event;
132     wait until eot'event;

134
        wait for 100 ns;
136     inp <= "0000" & patterns(i).inst_add(11 downto 8);
        wait for 20 ns;
138     wr <= '1';
        wait for 20 ns;
140     wr <= '0';
        wait until eot'event;
142     wait until eot'event;

144
        wait for 100 ns;
146     inp <= patterns(i).inst_add(7 downto 0);
        wait for 20 ns;
148     wr <= '1';
        wait for 20 ns;
150     wr <= '0';
        wait until eot'event;
152     wait until eot'event;

154
        wait for 100 ns;
156     inp <= patterns(i).inst_data(7 downto 0);
        wait for 20 ns;
158     wr <= '1';
        wait for 20 ns;
160     wr <= '0';
        wait until eot'event;
162     wait until eot'event;

164
        wait for 100 ns;
166     inp <= patterns(i).inst_data(15 downto 8);
        wait for 20 ns;
168     wr <= '1';
        wait for 20 ns;
170     wr <= '0';
        wait until eot'event;
172     wait until eot'event;

174     assert inst_ack = '1'
          report "receipt not acknowledged"
176       severity error;

178     assert inst_data = patterns(i).inst_data
          report "Wrong data recieve expected '" & str(patterns(i).inst_data) & "'
              recieved '" & str(inst_data) & "'"
180       severity error;

182     assert false report "finished transmission" severity note;

184     wait for 20 ns;

186     inst_req <= '1';
      end loop;
188   wait;
    end process;
190 end tb;
```

Listing 15: mmu/mmu_types.vhd

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

package mmu_types is
    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);
end mmu_types;
```

Listing 16: mmu/muart/BRG.vhd

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

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned."+";

entity br_generator is
  generic (divider_width: integer := 16);
  port (
    clock      : in  std_logic;
    rx_enable  : in  std_logic;
    clk_txd    : out std_logic;
    tx_enable  : in  std_logic;
    clk_serial : out std_logic
  );
end br_generator;

architecture principal of br_generator is
  -- change the following constant to your desired baud rate
  -- one hz equal to one bit per second
  signal   count_brg     : std_logic_vector(divider_width - 1 downto 0) := (others =>
      '0');
  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

  begin
```

```
      txd : process (clock)
57    begin
        if (rising_edge(clock)) then
59        if (count_brg_txd = brdvd) then
            clk_txd       <= '1';
61          count_brg_txd <= (others => '0');
          elsif (tx_enable = '1') then
63          clk_txd       <= '0';
            count_brg_txd <= count_brg_txd + 1;
65        else
            clk_txd       <= '0';
67          count_brg_txd <= (others => '0');
          end if;
69      end if;
      end process txd;
71
      rxd : process (clock)
73    begin
        if (rising_edge(clock)) then
75        if (count_brg=brdvd) then
            count_brg  <= (others => '0');
77          clk_serial <= '1';
          elsif (rx_enable = '1') then
79          count_brg  <= count_brg+1;
            clk_serial <= '0';
81        else
            count_brg  <= '0' & brdvd(divider_width - 1 downto 1);
83          clk_serial <= '0';
          end if;
85      end if;
      end process rxd;
87  end principal;
```

Listing 17: 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;

          when s1 =>
            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
```

```
183         outp <= data;
        end if;
185     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;

          when s9 =>
            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 ;
```

Listing 18: 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);
  process(f, rx, ry, Cin, Sum, AdderCout)
    --signal Z,C,N  : std_logic; -- Make the code easier to read
  BEGIN
    -- use case statement to achieve
    -- different operations of ALU

      AdderCin <= '0';
      A <= (others => '0');
      B <= (others => '0');
      output <= (others => '0');
      C <= '0';
      N <= '0';
      IF f = "0001" THEN -- Do AND operation
         output <= ry and rx;
      ELSIF f = "0011" THEN -- Do OR  operation
        output <= ry or rx;
      ELSIF f = "0101" THEN
        output <= not rx;
      ELSIF f = "0111" THEN -- Do XOR operation
        output <= ry xor rx;
      ELSIF f = "1001" THEN -- Do ADD operation
        AdderCin <= '0';
        A <= ry;
        B <= rx;
        output <= Sum;
      ELSIF f = "1011" THEN -- Do ADC operation
        AdderCin <= Cin;
        A <= ry;
        B <= rx;
        output <= Sum;
      ELSIF f = "1101" THEN -- Do SUB operation
        AdderCin <= '1';
        A <= ry;
        B <= (not rx);
        output <= Sum;
      ELSIF f = "1111" THEN -- Do SBB operation
        AdderCin <= (not Cin);
        A <= ry;
        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 19: processor/alu_tb.vhd

```
1   library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
3
    --  A testbench has no ports.
5   entity alu_tb is
      end alu_tb;
7
    architecture behav of alu_tb is
9     --  Declaration of the component that will be instantiated.
      component alu
11     Port (f   : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
            rx  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
13          ry  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
            ro  : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
15          Cin : in   STD_LOGIC;                      -- Carry in
            sr  : out  STD_LOGIC_VECTOR (2 downto 0)); -- Status register out Z(0),
                C(1), N(2)
17    end component;
      --  Specifies which entity is bound with the component.
19    for alu_0: alu use entity work.alu;
        signal f          : STD_LOGIC_VECTOR (3 downto 0);
21      signal rx, ry, ro : STD_LOGIC_VECTOR (7 downto 0);
        signal Cin        : STD_LOGIC;
23      signal sr         : STD_LOGIC_VECTOR (2 downto 0);
      begin
25      --  Component instantiation.
        alu_0: alu port map (f => f, rx => rx, ry => ry, ro => ro, Cin => Cin, sr => sr);
27
```

```
    --  This process does the real job.
29  process
    type pattern_type is record
31    f          : STD_LOGIC_VECTOR (3 downto 0);
      rx, ry     : STD_LOGIC_VECTOR (7 downto 0);
33    ro         : STD_LOGIC_VECTOR (7 downto 0);
      Cin        : STD_LOGIC;
35    sr         : STD_LOGIC_VECTOR (2 downto 0);
    end record;
37  --  The patterns to apply.
  type pattern_array is array (natural range <>) of pattern_type;
39  constant patterns : pattern_array :=
  -- f       rx          ry          ro          Cin   sr
41  (("0001", "00000000", "00000000", "00000000", '0', "001"), --AND tests - 1ns
    ("0001", "00000001", "00000001", "00000001", '0', "000"), --AND tests
43  ("0001", "00000000", "00000001", "00000000", '0', "001"), --AND tests
    ("0001", "10101010", "10101010", "10101010", '0', "100"), --AND tests
45  ("0001", "01010101", "01010101", "01010101", '0', "000"), --AND tests - 5ns
    ("0001", "11111111", "00000000", "00000000", '0', "001"), --AND tests
47  ("0001", "11111111", "11111111", "11111111", '0', "100"), --AND tests
    ("0001", "00000000", "01010101", "00000000", '0', "001"), --AND tests
49  ("0001", "00000000", "10101010", "00000000", '0', "001"), --AND tests
    ("0001", "11111111", "01010101", "01010101", '0', "000"), --AND tests - 10 ns
51  ("0001", "11111111", "10101010", "10101010", '0', "100"), --AND tests
    ("0001", "10000011", "10110010", "10000010", '0', "100"), --AND tests
53  ("0001", "00000011", "00110010", "00000010", '0', "000"), --AND tests

55  ("0011", "00000000", "00000000", "00000000", '0', "001"), --OR tests - 14 ns
    ("0011", "00000001", "00000001", "00000001", '0', "000"), --OR tests
57  ("0011", "00000000", "00000001", "00000001", '0', "000"), --OR tests
    ("0011", "10101010", "10101010", "10101010", '0', "100"), --OR tests
59  ("0011", "01010101", "01010101", "01010101", '0', "000"), --OR tests
    ("0011", "11111111", "00000000", "11111111", '0', "100"), --OR tests
61  ("0011", "11111111", "11111111", "11111111", '0', "100"), --OR tests - 20 ns
    ("0011", "00000000", "01010101", "01010101", '0', "000"), --OR tests
63  ("0011", "00000000", "10101010", "10101010", '0', "100"), --OR tests
    ("0011", "11111111", "01010101", "11111111", '0', "100"), --OR tests
65  ("0011", "11111111", "10101010", "11111111", '0', "100"), --OR tests
    ("0011", "10000011", "10110010", "10110011", '0', "100"), --OR tests - 25 ns
67  ("0011", "00000011", "00110010", "00110011", '0', "000"), --OR tests

69  ("0101", "00000000", "00000000", "11111111", '0', "100"), --NOT tests - ry should
        not matter
    ("0101", "00000001", "00000001", "11111110", '0', "100"), --NOT tests
71  ("0101", "00000000", "00000001", "11111111", '0', "100"), --NOT tests
    ("0101", "10101010", "10101010", "01010101", '0', "000"), --NOT tests - 30 ns
73  ("0101", "01010101", "01010101", "10101010", '0', "100"), --NOT tests
    ("0101", "11111111", "00000000", "00000000", '0', "001"), --NOT tests
75  ("0101", "11111111", "11111111", "00000000", '0', "001"), --NOT tests
    ("0101", "00000000", "01010101", "11111111", '0', "100"), --NOT tests
77  ("0101", "00000000", "10101010", "11111111", '0', "100"), --NOT tests - 35 ns
    ("0101", "11111111", "01010101", "00000000", '0', "001"), --NOT tests
79  ("0101", "11111111", "10101010", "00000000", '0', "001"), --NOT tests
    ("0101", "10000011", "10110010", "01111100", '0', "000"), --NOT tests
81  ("0101", "00000011", "00110010", "11111100", '0', "100"), --NOT tests - 39 ns

83  ("0111", "00000000", "00000000", "00000000", '0', "001"), --XOR tests - 40 ns
    ("0111", "00000001", "00000001", "00000000", '0', "001"), --XOR tests
85  ("0111", "00000000", "00000001", "00000001", '0', "000"), --XOR tests
    ("0111", "10101010", "10101010", "00000000", '0', "001"), --XOR tests
87  ("0111", "01010101", "01010101", "00000000", '0', "001"), --XOR tests
    ("0111", "11111111", "00000000", "11111111", '0', "100"), --XOR tests - 45 ns
89  ("0111", "11111111", "11111111", "00000000", '0', "001"), --XOR tests
    ("0111", "00000000", "01010101", "01010101", '0', "000"), --XOR tests
91  ("0111", "00000000", "10101010", "10101010", '0', "100"), --XOR tests
    ("0111", "11111111", "01010101", "10101010", '0', "100"), --XOR tests
93  ("0111", "11111111", "10101010", "01010101", '0', "000"), --XOR tests - 50 ns
    ("0111", "10000011", "10110010", "00110001", '0', "000"), --XOR tests
95  ("0111", "00000011", "00110010", "00110001", '0', "000")  --XOR tests
  );
97 begin
    --  Check each pattern.
99  for i in patterns'range loop
```

```
        --  Set the inputs.
101     Cin <= patterns(i).Cin;
        f <= patterns(i).f;
103     rx <= patterns(i).rx;
        ry <= patterns(i).ry;
105     --  Wait for the results.
        wait for 1 ns;
107     --  Check the outputs.
        assert ro = patterns(i).ro
109     report "bad output register value" severity error;
        assert sr = patterns(i).sr
111     report "bad status register value" severity error;
        assert sr(0) = patterns(i).sr(0)
113     report " *Zero is incorrect" severity error;
        assert sr(1) = patterns(i).sr(1)
115     report " *Carry is incorrect" severity error;
        assert sr(2) = patterns(i).sr(2)
117     report " *Negitive is incorrect" severity error;
    end loop;
119   assert false report "end of test" severity note;
      --  Wait forever; this will finish the simulation.
121   wait;
  end process;
123 end behav;
```

Listing 20: processor/ar.vhd

```
  ----------------------------------------------------------------------------------
2 -- Company:
  -- Engineer:
4 --
  -- Create Date:  18:59:20 09/18/2010
6 -- Design Name:
  -- Module Name:  ar - 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
  library work;
24 use work.reg16;

26 entity ar is
    Port (clk         : in   STD_LOGIC;
28        enable      : in   STD_LOGIC;
          Sel8Bit     : in   STD_LOGIC;
30        SelHighByte : in   STD_LOGIC;
          ByteInput   : in   STD_LOGIC_VECTOR (7 downto 0);
32        SelRi       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
              register
          SelRo       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
              register
34        Ri          : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input
          Ro          : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
36 end ar;

38 architecture Behavioral of ar is
    component reg16 IS
40    port(I      : in  std_logic_vector(15 downto 0);
           clock  : in  std_logic;
42         enable : in  std_logic;
           reset  : in  std_logic;
```

```
44          Q       : out std_logic_vector(15 downto 0)
          );
46    end component;

48    signal  R0E    : std_logic;  -- Enable signals
      signal  R1E    : std_logic;
50    signal  R2E    : std_logic;
      signal  input  : std_logic_VECTOR (15 downto 0);
52    signal  Q0     : std_logic_VECTOR (15 downto 0);
      signal  Q1     : std_logic_VECTOR (15 downto 0);
54    signal  Q2     : std_logic_VECTOR (15 downto 0);
  BEGIN
56      reg_0 : reg16 port map(input, clk, R0E, '0', Q0);
        reg_1 : reg16 port map(input, clk, R1E, '0', Q1);
58      reg_2 : reg16 port map(input, clk, R2E, '0', Q2);

60    SetInput: process(clk, enable, SelRi, Ri)
      BEGIN
62      R0E <= '0';
        R1E <= '0';
64      R2E <= '0';
        IF enable = '1' THEN
66        case SelRi IS
            WHEN "00" =>
68            R0E <= '1';
            WHEN "01" =>
70            R1E <= '1';
            WHEN "10" =>
72            R2E <= '1';
            WHEN others =>
74            NULL; -- None of them are enabled
          END CASE;
76      END IF;
      end process;
78
      -- Select if 1 or 2 Bytes is to be written and if
80    SetNumBytes: process(clk, Ri, SelRi, ByteInput, Sel8Bit, SelHighByte, Q0, Q1, Q2)
      BEGIN
82      IF Sel8Bit = '0' THEN
          input <= Ri;
84      ELSE
          if SelHighByte = '1' THEN
86          input(15 downto 8) <= ByteInput;
            case SelRi IS
88            WHEN "00" =>
                input(7 downto 0) <= Q0(7 downto 0);
90            WHEN "01" =>
                input(7 downto 0) <= Q1(7 downto 0);
92            WHEN others =>
                input(7 downto 0) <= Q2(7 downto 0);
94          END CASE;
          else
96          input(7 downto 0) <= ByteInput;
            case SelRi IS
98            WHEN "00" =>
                input(15 downto 8) <= Q0(15 downto 8);
100           WHEN "01" =>
                input(15 downto 8) <= Q1(15 downto 8);
102           WHEN others =>
                input(15 downto 8) <= Q2(15 downto 8);
104         END CASE;
          END IF;
106     END IF;
      end process;
108
      -- Set the output Ro
110   WITH SelRo SELECT
      Ro <= Q0 WHEN "00",
112         Q1 WHEN "01",
            Q2 WHEN others;
114 end Behavioral;
```

Listing 21: 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
17 -- Additional Comments:
   --
19 ----------------------------------------------------------------------------------
   library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
   --use IEEE.STD_LOGIC_ARITH.ALL;
23 --use IEEE.STD_LOGIC_UNSIGNED.ALL;

25 library work;
   use work.alu;
27 use work.cu;
   use work.ar;
29 use work.gpr;
   use work.sr;
31 use work.pc;

33
   ---- Uncomment the following library declaration if instantiating
35 ---- any Xilinx primitives in this code.
   --library UNISIM;
37 --use UNISIM.VComponents.all;

39 entity cpu is

41   PORT(
        -- instruction bus
43      inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
        inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
45      inst_req  : out std_logic;                     -- Pulled low to request bus
            usage.
        inst_ack  : in  std_logic;                     -- Pulled high to inform of
            request completion.
47      -- data bus
        data_add  : out    std_logic_vector(15 downto 0); -- Address lines.
49      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.
51      data_req  : out    std_logic;                     -- Pulled low to request bus
            usage.
        data_ack  : inout std_logic;                     -- Pulled high to inform of
            request completion.
53      -- extras
        clk       : in  std_logic;
55      reset     : in  std_logic
     );
57
   end cpu;
59
   architecture cpu_arch of cpu is
61   component alu IS
       Port (f   : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
63           rx  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
             ry  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
65           ro  : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
             Cin : in   STD_LOGIC;                      -- Carry in
```

```
67          sr  : out  STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
                C(1), N(2)
    END component;
69  component ar is
      Port (clk         : in   STD_LOGIC;
71          enable      : in   STD_LOGIC;
            Sel8Bit     : in   STD_LOGIC;
73          SelHighByte : in   STD_LOGIC;
            ByteInput   : in   STD_LOGIC_VECTOR (7 downto 0);
75          SelRi       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
                register
            SelRo       : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
                register
77          Ri          : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input
            Ro          : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
79  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
215             gpr_Rx    => alu_rx,-- Rx from GPR
                --gpr_Ry    => alu_ry,-- Ry from GPR
217
                -- Status Register
219             sr_en     => sr_enable,-- enable write to SR
                sr_reset  => sr_reset,-- reset SR
221             sr_Ro     => sr_Ro,-- output from SR
                -- control unit doesnt write to SR, the ALU does
223
                -- Program Counter
225             pc_en     => pc_enable,-- enable write to PC
                pc_reset  => pc_reset,-- reset PC
227             pc_Ri     => pc_Ri,-- input to PC
                pc_Ro     => pc_Ro,-- output from PC
229
                -- Address Registers
231             ar_en       => ar_enable,   -- enable write to AR
                ar_SelRi    => ar_SelRi,    -- select AR in
233             ar_SelRo    => ar_SelRo,    -- select AR out
                ar_sel8Bit  => ar_sel8Bit,
235             ar_selHByte => ar_selHByte,
                ar_ByteIn   => ar_ByteIn,
237             ar_Ri       => ar_Ri,       -- input to AR
                ar_Ro       => ar_Ro,       -- output from AR
239
                -- Instruction memory
241             inst_add  => inst_add ,-- Instruction address
                inst_data => inst_data,-- Instruction data
243             inst_req  => inst_req ,-- Request
                inst_ack  => inst_ack ,-- Instruction obtained
245
                data_add  => data_add ,-- Data address
247             data_data => data_line,-- Data
                data_read => data_read,-- 1 for read, 0 for write
249             data_req  => data_req ,-- Request
                data_ack  => data_ack  -- Data written to/ read from
251           );
    address : ar port map(
253             clk         => clk,
                enable      => ar_enable,
255             Sel8Bit     => ar_Sel8Bit,
                SelHighByte => ar_selHByte,
257             ByteInput   => ar_ByteIn,
                SelRi       => ar_SelRi,
259             SelRo       => ar_SelRo,
                Ri          => ar_Ri,
261             Ro          => ar_Ro
              );
263 g : gpr port map(
                clk    => clk,
265             enable => gpr_enable,
                SelRx  => gpr_SelRx,
267             SelRy  => gpr_SelRy,
                SelRi  => gpr_SelRi,
269             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,
            Ro       => pc_Ro
          );
end cpu_arch;
```

Listing 22: processor/cu.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  cu - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: The control unit
--
-- 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.fulladder;
--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 cu is
  Port (reset       : in STD_LOGIC;                       -- '0' for reset
        clock       : in STD_LOGIC;                       -- clock

        alu_f       : out STD_LOGIC_VECTOR (3 downto 0);  -- Function
        alu_Cin     : out STD_LOGIC;                      -- Carry in to ALU

        -- 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;
    signal write_pc     : std_logic;
    signal write_ar     : std_logic;
    signal write_memory : std_logic;

    -- full adders
    signal   A16         : std_logic_vector(15 downto 0);
    signal   B16         : std_logic_vector(15 downto 0);
    signal   AdderCin16  : std_logic;
    signal   Sum16       : std_logic_vector(15 downto 0);
    signal   AdderCout16 : std_logic;

    signal v    : STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit immediate


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

    -- Process instruction
    -- 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,
            rx, ry, ay, v, write_gpr, write_sr, write_pc, write_ar, write_memory)
    BEGIN
      if rising_edge(clock) then
        case state is
          when reset_state =>
            sr_reset <= '0';
            pc_reset <= '0';
            next_state <= fetch;

          when fetch =>
            sr_reset <= '1';
            pc_reset <= '1';

            gpr_en <= '0';
            sr_en <= '0';
            pc_en <= '0';
            ar_en <= '0';

            write_gpr <= '0';
            write_sr <= '0';
            write_pc <= '0';
            write_ar <= '0';
            write_memory <= '0';

            inst_add <= pc_Ro(11 downto 0);
            if inst_ack = '0' then
              inst_req <= '1';
            else
              opcode <= inst_data;
              inst_req <= '0';

              -- increment program counter
              AdderCin16 <= '1';
              A16 <= PC_Ro;
              B16 <= "0000000000000000";
              pc_Ri <= Sum16;
              pc_en <= '1';

              next_state <= decode;
            end if;
          when decode =>
            pc_en <= '0';

            -- ALU
            if opcode(15) = '0' and opcode(10) = '0' and not opcode(14 downto 11) =
                "0010" then

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

              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 =>
              gpr_en <= write_gpr;
425           sr_en <= write_sr;
              pc_en <= write_pc;
427           ar_en <= write_ar;
              if write_memory = '1' then
429             if data_ack = '0' then  -- request write
                  data_req <= '1';
431             else                    -- data written
                  data_req <= '0';
433               next_state <= fetch;
                end if;
435           else
                next_state <= fetch;
437           end if;

439         when others =>
              -- shouldnt reach here
441           next_state <= reset_state;
        end case;
443     end if;
    end process;
445
    process(clock, reset, next_state)
447 BEGIN
      if reset = '0' then
449     state <= reset_state;
      elsif rising_edge(clock) then
451     state <= next_state;
      end if;
453 end process;

455 end Behavioral;
```

Listing 23: processor/fulladder.vhd

```
  ----------------------------------------------------------------------------------
2 -- Company:
  -- Engineer:
4 --
  -- Create Date:  18:59:20 09/18/2010
6 -- Design Name:
  -- Module Name:  fulladder - 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;

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

--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity 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 fulladder8;

architecture arch_fulladder8 of fulladder8 is
  component fulladder IS
  Port (Ax    : in   STD_LOGIC;
        Bx    : in   STD_LOGIC;
        Ci    : in   STD_LOGIC;
        Sx    : out  STD_LOGIC;
        Co    : out  STD_LOGIC
        );
  end component;
  signal Carry   : std_logic_vector(8 downto 0);
BEGIN
  Carry(0) <= Cin;

    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));
    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));
    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));
    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));

  Cout <= Carry(8);
end arch_fulladder8;

--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder16 is
  Port (A     : in   STD_LOGIC_VECTOR( 15 downto 0);
        B     : in   STD_LOGIC_VECTOR( 15 downto 0);
        Cin   : in   STD_LOGIC;
        Sum   : out  STD_LOGIC_VECTOR( 15 downto 0);
        Cout  : out  STD_LOGIC
        );
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 24: 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;
14         Sum  : out   STD_LOGIC_VECTOR( 7 downto 0);
           Cout : out   STD_LOGIC
16         );
     end component;
18   --  Specifies which entity is bound with the component.
     for fulladder8_0: fulladder8 use entity work.fulladder8;
20     signal A,B,Sum    : STD_LOGIC_VECTOR (7 downto 0);
       signal Cin,Cout   : STD_LOGIC;
22   begin
       --  Component instantiation.
24     fulladder8_0: fulladder8 port map (A => A, B => B, Cin => Cin, Sum => Sum, Cout
           => Cout);

26     --  This process does the real job.
       process
28     type pattern_type is record
         A    :  STD_LOGIC_VECTOR( 7 downto 0);
30       B    :  STD_LOGIC_VECTOR( 7 downto 0);
         Cin  :  STD_LOGIC;
32       Sum  :  STD_LOGIC_VECTOR( 7 downto 0);
         Cout :  STD_LOGIC;
34     end record;
     --  The patterns to apply.
```

```
36    type pattern_array is array (natural range <>) of pattern_type;
      constant patterns : pattern_array :=
38    -- A            B           Cin    Sum      Cout
      (("00000000", "00000000", '0', "00000000", '0'), --AND tests - 1ns
40     ("11111111", "11111111", '1', "11111111", '1'), --AND tests
       ("00000000", "00000000", '1', "00000001", '0'), --AND tests
42     ("00000000", "11111111", '0', "11111111", '0'), --AND tests
       ("11111111", "00000000", '0', "11111111", '0'), --AND tests
44     ("11111111", "00000000", '1', "00000000", '1'), --AND tests
       ("10101010", "01010101", '0', "11111111", '0'), --AND tests
46     ("10101010", "01010101", '1', "00000000", '1'), --AND tests
       ("11111111", "11111111", '0', "11111110", '1')  --XOR tests
48    );
    begin
50    --  Check each pattern.
      for i in patterns'range loop
52      --  Set the inputs.
        A   <= patterns(i).A;
54      B   <= patterns(i).B;
        Cin <= patterns(i).Cin;
56      --  Wait for the results.
        wait for 1 ns;
58      --  Check the outputs.
        assert Sum = patterns(i).Sum
60      report "The sum check failed" severity error;
        assert Cout = patterns(i).Cout
62      report "The carry out is wrong" severity error;
      end loop;
64    assert false report "end of test" severity note;
      --  Wait forever; this will finish the simulation.
66    wait;
    end process;
68  end behav;
```

Listing 25: processor/gpr.vhd

```
1   ----------------------------------------------------------------------------------
    -- Company:
3   -- Engineer:
    --
5   -- Create Date:  18:59:20 09/18/2010
    -- Design Name:
7   -- Module Name:  GPR - gpr_arch
    -- 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.reg8;
25
    entity gpr is
27    Port (clk       : in    STD_LOGIC;
          enable    : in    STD_LOGIC;
29        SelRx     : in    STD_LOGIC_VECTOR (2 downto 0);  -- The Rx output selection
              value
          SelRy     : in    STD_LOGIC_VECTOR (2 downto 0);  -- The Ry output selection
              value
31        SelRi     : in    STD_LOGIC_VECTOR (2 downto 0);  -- The Ri input selection
              value
          SelIn     : in    STD_LOGIC;  -- Select where the input should be from the CU
              or CDB
```

```
33          RiCU      : in    STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Control
                Unit
            RiCDB     : in    STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Common Data
                 Bus
35          Rx        : out   STD_LOGIC_VECTOR (7 downto 0);  -- The Rx output
            Ry        : out   STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
37  end gpr;

39
    architecture gpr_arch of gpr is
41    component reg8 IS
        port(I      : in  std_logic_vector(7 downto 0);
43           clock  : in  std_logic;
             enable : in  std_logic;
45           reset  : in  std_logic;
             Q      : out std_logic_vector(7 downto 0)
47          );
      end component;
49
      signal  reset: std_logic := '0';
51    signal  input: std_logic_VECTOR (7 downto 0);
      signal  R0E  : std_logic;  -- Enable signals
53    signal  R1E  : std_logic;
      signal  R2E  : std_logic;
55    signal  R3E  : std_logic;
      signal  R4E  : std_logic;
57    signal  R5E  : std_logic;
      signal  R6E  : std_logic;
59    signal  R7E  : std_logic;
      signal  Q0   : std_logic_VECTOR (7 downto 0);
61    signal  Q1   : std_logic_VECTOR (7 downto 0);
      signal  Q2   : std_logic_VECTOR (7 downto 0);
63    signal  Q3   : std_logic_VECTOR (7 downto 0);
      signal  Q4   : std_logic_VECTOR (7 downto 0);
65    signal  Q5   : std_logic_VECTOR (7 downto 0);
      signal  Q6   : std_logic_VECTOR (7 downto 0);
67    signal  Q7   : std_logic_VECTOR (7 downto 0);
    BEGIN
69      reg_0 : reg8 port map(input, clk, R0E, reset, Q0);
        reg_1 : reg8 port map(input, clk, R1E, reset, Q1);
71      reg_2 : reg8 port map(input, clk, R2E, reset, Q2);
        reg_3 : reg8 port map(input, clk, R3E, reset, Q3);
73      reg_4 : reg8 port map(input, clk, R4E, reset, Q4);
        reg_5 : reg8 port map(input, clk, R5E, reset, Q5);
75      reg_6 : reg8 port map(input, clk, R6E, reset, Q6);
        reg_7 : reg8 port map(input, clk, R7E, reset, Q7);
77
      -- Select where the input should come from
79      SelectInput: process(SelIn, RiCDB, RiCU)
        BEGIN
81        IF SelIn = '1' THEN
              input <= RiCDB;
83        ELSE
            input <= RiCU;
85        END IF;
        END process;
87
      -- Set Ri the input
89    SetInput: process(clk, enable, SelRi)
      BEGIN
91      R0E <= '0';
        R1E <= '0';
93      R2E <= '0';
        R3E <= '0';
95      R4E <= '0';
        R5E <= '0';
97      R6E <= '0';
        R7E <= '0';
99     IF enable = '1' THEN
          case SelRi IS
101         WHEN "000" =>
              R0E <= '1';
103         WHEN "001" =>
```

```
            R1E <= '1';
        WHEN "010" =>
            R2E <= '1';
        WHEN "011" =>
            R3E <= '1';
        WHEN "100" =>
            R4E <= '1';
        WHEN "101" =>
            R5E <= '1';
        WHEN "110" =>
            R6E <= '1';
        WHEN "111" =>
            R7E <= '1';
        WHEN others =>
            NULL; -- None of them are enabled
    end case;
  END IF;
end process;

-- Set the Rx output
WITH SelRx SELECT
Rx <= Q0 WHEN "000",
      Q1 WHEN "001",
      Q2 WHEN "010",
      Q3 WHEN "011",
      Q4 WHEN "100",
      Q5 WHEN "101",
      Q6 WHEN "110",
      Q7 WHEN others;

-- Set the Ry output
WITH SelRy SELECT
Ry <= Q0 WHEN "000",
      Q1 WHEN "001",
      Q2 WHEN "010",
      Q3 WHEN "011",
      Q4 WHEN "100",
      Q5 WHEN "101",
      Q6 WHEN "110",
      Q7 WHEN others;

end gpr_arch;
```

Listing 26: 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);
```

```
26  -- Does the clock signal
    process
28  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 27: 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;
```

```
use ieee.std_logic_1164.all;

entity reg16 is
port(I      : in  std_logic_vector(15 downto 0);
     clock  : in  std_logic;
     enable : in  std_logic;
     reset  : in  STD_LOGIC;
     Q      : out std_logic_vector(15 downto 0)
    );
end reg16;

architecture behv of reg16 is
begin

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

  end process;

end behv;
```

Listing 28: processor/spr.vhd

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

library work;
use work.reg16;

entity sr is
  Port (clk      : in  STD_LOGIC;
        enable   : in  STD_LOGIC;
        reset    : in  STD_LOGIC;
        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
end sr;

architecture sr_arch of sr is
  component reg16 IS
  port(I      : in  std_logic_vector(15 downto 0);
       clock  : in  std_logic;
       enable : in  std_logic;
       reset  : in  STD_LOGIC;
       Q      : out std_logic_vector(15 downto 0)
      );
  end component;
```

```
BEGIN
  reg_sr : reg16 port map(Ri, clk, enable, reset, Ro);
end sr_arch;

--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pc is
  Port (clk      : in  STD_LOGIC;
        enable   : in  STD_LOGIC;
        reset    : in  STD_LOGIC;
        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
end pc;

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

Listing 29: 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
  Port (clk      : in   STD_LOGIC;
        enable   : in   STD_LOGIC;                       -- Enable write
        reset    : in   STD_LOGIC;                       -- Reset the register
        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
  end component;

  signal sr_enable : std_logic;
  signal sr_reset  : std_logic;
  signal sr_Ri     : std_logic_vector(15 downto 0);
  signal sr_Ro     : std_logic_vector(15 downto 0);

  component pc
  Port (clk      : in   STD_LOGIC;
        enable   : in   STD_LOGIC;                       -- Enable write
        reset    : in   STD_LOGIC;                       -- Reset the register
        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
  end component;

  signal pc_enable : std_logic;
  signal pc_reset  : std_logic;
  signal pc_Ri     : std_logic_vector(15 downto 0);
  signal pc_Ro     : std_logic_vector(15 downto 0);

  signal T_clk  : std_logic;

```

```
begin

  U_sr: sr port map (clk => T_clk, enable => sr_enable, reset => sr_reset, Ri =>
      sr_Ri, Ro => sr_Ro);
  U_pc: pc port map (clk => T_clk, enable => pc_enable, reset => pc_reset, Ri =>
      pc_Ri, Ro => pc_Ro);

    -- concurrent process to offer the clk signal
  process
  begin
    T_clk <= '0';
    wait for 5 ns;
    T_clk <= '1';
    wait for 5 ns;
  end process;

  process

    variable err_cnt: integer :=0;

  begin

    -- Write
    sr_enable <= '1';
    sr_reset  <= '0';
    sr_Ri     <= "0100011001011001";
    pc_enable <= '1';
    pc_reset  <= '0';
    pc_Ri     <= "0101011010110100";
    wait for 20 ns;

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

    -- Change Ri
    sr_Ri <= "1001100101110100";
    pc_Ri <= "0001010001110000";
    wait for 20 ns;
    assert (sr_Ro = "1001100101110100") report "Read sr #2 failed" severity error;
    assert (pc_Ro = "0001010001110000") report "Read pc #2 failed" severity error;

    -- Disable sr, pc still enabled
    sr_enable <= '0';
    sr_Ri <= "0101010101010101";
    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
```

```
109
      end process;
111
    end TB;
113
    ------------------------------------------------------------------
115 configuration CFG_TB of spr_TB is
      for TB
117   end for;
    end CFG_TB;
```