

## 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 DOWNT0 0);    -- address lines --
37         data_data : INOUT    std_logic_vector(7 DOWNT0 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
         port(
49         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';
65 signal inst_ack    : std_logic;
signal data_add      : std_logic_vector(15 downto 0);
67 signal data_line   : std_logic_vector(7  downto 0);
signal data_read     : std_logic;
69 signal data_req    : std_logic;
signal data_ack      : std_logic;
71 signal clk         : std_logic;
signal receive_pin   : std_logic;
73 signal transfer_pin : std_logic;
signal sw1, sw2      : std_logic;

75
signal eoc, rxd, txd, eot, ready, wr: std_logic;
77 signal outp, inp : std_logic_vector(7  downto 0);
begin
79 m : mmu_main port map (inst_add, inst_data, inst_req, inst_ack, data_add,
    data_line, data_read, data_req, data_ack, clk, receive_pin, transfer_pin);
    i : IO      port map (data_add, data_line, data_read, data_req, data_ack, clk,
        sw1, sw2);
81 muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);

83 rxd <= transfer_pin;
receive_pin <= txd;

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

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

123         if patterns(i).data_add(0) = '1' then
            wait until eoc'event;
125             assert outp = patterns(i).recv_head
                report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved
                    '" & str(outp) & "'"
127                 severity error;
            wait until eoc'event;
129
            assert false report "passed header" severity note;
131
            wait until eoc'event;
133             assert outp = patterns(i).data_add(8 downto 1)

```

```

    report "Bad address low expected '" & str(patterns(i).data_add(7 downto
        0)) & "' recieved '" & str(otp) & "'"
135     severity error;
    wait until eoc'event;

137
    assert false report "passed address low" severity note;
139

    wait until eoc'event;
    assert otp = "0" & patterns(i).data_add(15 downto 9)
        report "Bad address high expected '" & str(patterns(i).data_add(11 downto
            8)) & "' recieved '" & str(otp) & "'"
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 otp = patterns(i).data_data
            report "Bad data expected '" & str(patterns(i).data_data) & "' recieved
                '" & str(otp) & "'"
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 DOWNT0 0); -- address lines --
59         data_data : INOUT  std_logic_vector(7 DOWNT0 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 DOWNT0 0);
    END component;

-- 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.

```

```

111         -- extras
        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

```

-----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    16:08:33 09/15/2010
6  -- Design Name:
7  -- Module Name:    IO - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision: Saturday 16 Oct 2010 by Sasha
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
-----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 --use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 library work;
26 use work.debounce;
27 use work.switch_reg;
28 use work.led_io;
29
30
31
32 ---- Uncomment the following library declaration if instantiating
33 ---- any Xilinx primitives in this code.
34 --library UNISIM;
35 --use UNISIM.VComponents.all;
36
37 entity IO is
38     PORT(
39         -- data bus --
40         data_add      : IN          std_logic_vector(15 DOWNTO 0);
41             -- address lines --
42         data_data      : INOUT       std_logic_vector(7  DOWNTO 0);  -- data
43             lines --
44         data_read      : INOUT       std_logic;
45             -- pulled high for read, low for write --
46         data_req       : INOUT       std_logic;
47             -- pulled low to request bus usage --
48         data_ack       : INOUT       std_logic;
49             -- pulled high to inform request completion --
50         -- io --
51         clk            : IN          std_logic;
52         sw1            : IN          std_logic;
53         sw2            : IN          std_logic);
54         --leds          : OUT std_logic_vector(7 DOWNTO 0);
55 end IO;
56
57 architecture io of IO is
58
59     COMPONENT led_io
60     PORT(
61         data_add      : IN          std_logic_vector(15 DOWNTO 0);  --
62             address lines --
63         data_data      : INOUT       std_logic_vector(7  DOWNTO 0);  -- data lines
64             --
65         data_read      : INOUT       std_logic;                      --
66             pulled high for read, low for write --
67         data_req       : INOUT       std_logic;                      --
68             pulled low to request bus usage --

```

```

        data_ack      : INOUT      std_logic;          --
        pulled high to inform request completion --
62
        clock         : IN          std_logic
64    );
    END COMPONENT;
66
    COMPONENT switch_io IS
68        PORT ( data_add      : IN          std_logic_vector(15 DOWNT0 0);
                data_data     : INOUT       std_logic_vector(7  DOWNT0 0);
70                data_read    : INOUT       std_logic;
                data_req       : INOUT       std_logic;
72                data_ack     : INOUT       std_logic;
                clk           : IN          std_logic;
74                sw1          : IN          std_logic;
                sw2          : IN          std_logic
76            );
    END COMPONENT;
78

80
    BEGIN
82
    led: led_io PORT MAP(data_add, data_data, data_read, data_req, data_ack, clk);
84    switch: switch_io PORT MAP(data_add, data_data, data_read, data_req, data_ack,
        clk,sw1,sw2);

86 -----

88 END io;

```

#### Listing 5: IO/leds.vhd

```

-----
2  -- Company:
   -- Engineer:
4  --
   -- Create Date:      14:27:55 10/12/2010
6  -- Design Name:
   -- Module Name:      led_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.ALL;
24
   library work;
26
   ---- Uncomment the following library declaration if instantiating
28 ---- any Xilinx primitives in this code.
   --library UNISIM;
30 --use UNISIM.VComponents.all;

32 ENTITY led_io IS
   PORT(
34         data_add      : IN          std_logic_vector(15 DOWNT0 0);    -- address lines --
         data_data     : INOUT       std_logic_vector(7  DOWNT0 0);    -- data lines --
36         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
35     PROCESS(D, enable, clk)
        BEGIN
37         IF rising_edge(clk) THEN --Need else there???
            IF enable = '1' THEN
39                 Q <= D;
                    END IF;
41             END IF;
                END PROCESS;
43 END reg_arch;

```

#### Listing 7: IO/switches.vhd

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:      16:08:33 09/15/2010
6  -- Design Name:
7  -- Module Name:      IO - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision: Saturday 16 Oct 2010 by Sasha
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 --use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 library work;
26 use work.debounce;
27 use work.switch_reg;
28 use work.led_io;
29
30
31 ---- Uncomment the following library declaration if instantiating
32 ---- any Xilinx primitives in this code.
33 --library UNISIM;
34 --use UNISIM.VComponents.all;
35
36 entity switch_io is
37     PORT(
38         -- data bus --
39         data_add      : IN          std_logic_vector(15 DOWNTO 0);
40             -- address lines --
41         data_data      : INOUT       std_logic_vector(7 DOWNTO 0);  -- data
42             lines --
43         data_read      : INOUT       std_logic;
44             -- pulled high for read, low for write --
45         data_req       : INOUT       std_logic;
46             -- pulled low to request bus usage --
47         data_ack       : INOUT       std_logic;
48             -- pulled high to inform request completion --
49         -- io --
50         clk            : IN          std_logic;
51         sw1            : IN          std_logic;
52         sw2            : IN          std_logic;
53         --leds         : OUT std_logic_vector(7 DOWNTO 0);
54     end switch_io;
55
56 architecture Behavioral of switch_io is
57

```

```

55 signal enable1                : std_logic;
signal switch1_connection      : std_logic;
57 signal switch1_output        : std_logic;
--signal switch1_state         : std_logic;
59
signal enable2                : std_logic;
61 signal switch2_connection    : std_logic;
signal switch2_output          : std_logic;
63 --signal switch2_state       : std_logic;

65
COMPONENT debounce
67     PORT(clk, switch : IN STD_LOGIC;
           switch_state: OUT STD_LOGIC);
69 END COMPONENT;

71 COMPONENT switch_reg
    PORT( D           : IN STD_LOGIC;
          clk, enable : IN STD_LOGIC;
          Q           : OUT STD_LOGIC);
73
75 END COMPONENT;

77

79 BEGIN

81 sw1_debouncer: debounce PORT MAP(clk, sw1, switch1_connection);
sw1_status: switch_reg PORT MAP(switch1_connection,clk, enable1, switch1_output);
83

85 sw2_debouncer: debounce PORT MAP(clk, sw2,switch2_connection);
sw2_status: switch_reg PORT MAP(switch2_connection,clk, enable2, switch2_output);
87

89
PROCESS(clk,switch1_output,switch2_output, data_ack)
91 BEGIN
IF rising_edge(clk) THEN
93     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';
95
        --ELSIF data_ack = '0' AND data_add = "0000000000001110" THEN -- when the data is
        sent to the CPU, enable the switch_reg again
97     -- enable1 <= '1';
ELSE
99     enable1 <= '1';
END IF;
101
    IF switch2_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
        1, disable switch_reg from getting any more info
103     enable2 <= '0';

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

111
END IF;
113 END PROCESS;

115

117

119

121 PROCESS(clk, data_add, data_read)
BEGIN
123     IF rising_edge(clk) THEN

```

```

125     IF data_req = '0' AND data_read = '1' THEN
        IF data_add = "0000000000001110" THEN -- switch1 address
            IF switch1_output = '1' THEN
127                 data_data <= "00000001";
            ELSE
129                 data_data <= "00000000";
            END IF;
            data_ack <= '0';
        END IF;
        IF data_add = "0000000000001100" THEN -- switch2 address
            IF switch2_output = '1' THEN
135                 data_data <= "00000001";
            ELSE
137                 data_data <= "00000000";
            END IF;
            data_ack <= '0';
        END IF;
141     ELSIF data_req = '1' AND data_ack = '0' THEN
        data_ack <= 'Z';
143     END IF;
145 END IF;
147 END PROCESS;
149
-----
149 END Behavioral;

```

#### Listing 8: mmu/control\_unit.vhd

```

library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

4 library work;
use work.mmu_types.all;
6
entity mmu_control_unit is
8     port (
        eoc          : in std_logic; -- High on muart has finished collecting data ??
10        eot          : in std_logic; -- High on muart has finished transmitting data.
        ready         : in std_logic; -- ??
12        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.
14        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.
16        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.
18        rw            : in std_logic; -- High if latest input packet had read/write set.
        write          : out std_logic; -- High to start muart writing data.
20        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.
22        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
24        muart_output   : out muart_output_state; -- State to multiplex the muart's output.
        clk             : in std_logic
    );
26 end mmu_control_unit;
28
architecture mmu_control_unit_arch of mmu_control_unit is
30     component data_control_unit is
        port (
32         eoc          : in std_logic; -- High on muart has finished collecting data ??
         eot          : in std_logic; -- High on muart has finished transmitting data.
34         ready         : in std_logic; -- ??

```

```

    data_read      : in std_logic; -- High if the data line is requesting a read, low
    for write.
36    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.
38    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.
40    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.
42    clk          : in std_logic
    );
44    end component;

46    component inst_control_unit is
    port (
48        eoc          : in std_logic; -- High on muart has finished collecting data ??
        eot          : in std_logic; -- High on muart has finished transmitting data.
50        ready        : in std_logic; -- ??
        inst_req       : in std_logic; -- Low when the instruction address is valid and
        should be read.
52        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.
54        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
56        muart_output  : out muart_output_state; -- State to multiplex the muart's output.
        clk            : in std_logic
    );
    end component;

60
    signal data_write, inst_write, inst_inst_or_data_out : std_logic;
62    signal data_muart_input, inst_muart_input : muart_input_state;
    signal data_muart_output, inst_muart_output : muart_output_state;
64    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);
66        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);

68        inst_or_data_out <= inst_inst_or_data_out ;
        write <= inst_write or data_write;
70        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;
72    end mmu_control_unit_arch;

```

Listing 9: mmu/data\_control\_unit.vhd

```

library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

4 library work;
use work.mmu_types.all;

6
entity data_control_unit is
8    port (
        eoc          : in std_logic; -- High on muart has finished collecting data ??
10        eot          : in std_logic; -- High on muart has finished transmitting data.
        ready        : in std_logic; -- ??
12        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.
14        data_add_0    : in std_logic; -- High for memory address, not IO.
        write         : out std_logic; -- High to start muart writing data.
16        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
18        muart_output  : out muart_output_state; -- State to multiplex the muart's output.

```

```

    clk      : in  std_logic
20 );
end data_control_unit;
22
architecture data_control_unit_arch of data_control_unit is
24     type state_type is (idle, get_data, wait_clear);
        type m_state_type is (idle,
26                               send_header, send_add_high, send_add_low, send_data,
                                   get_header, get_add_high, get_add_low, get_data,
28                               finished);
        type read_state_type is (idle, wait_data, read_data, pause, finished);
30     type transmit_state_type is (idle, set_data, trans_data, pause, finished);
    signal state, next_state : state_type := idle;
32     signal get_state, next_get_state : m_state_type := idle;
    signal reader_state, next_reader_state : read_state_type := idle;
34     signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
begin
36     data_fsm : process(state, data_req, clk) begin
        if (rising_edge(clk)) then
38         case state is
            when idle =>
40             if (data_req = '0' and data_add_0 = '1') then
                next_state <= get_data;
42             end if;

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

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

54             when others =>
                NULL;
56             end case;
        end if;
58     end process data_fsm;

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

72             when send_add_low =>
                if transmitter_state = finished then
74                 next_get_state <= send_add_high;
                end if;
76             when send_add_high =>
                if transmitter_state = finished then
78                 if data_read = '1' then
                    next_get_state <= get_header;
80                 else
                    next_get_state <= send_data;
82                 end if;
                end if;
84             end if;

86             when send_data =>
                if transmitter_state = finished then
88                 next_get_state <= finished;
                end if;
90             when get_header =>

```

```

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

96     when get_add_low =>
97         if reader_state = finished then
98             next_get_state <= get_add_high;
99         end if;

100    when get_add_high =>
101        if reader_state = finished then
102            next_get_state <= get_data;
103        end if;

104    when get_data =>
105        if reader_state = finished then
106            next_get_state <= finished;
107        end if;

108    when finished =>
109        next_get_state <= idle;

110    when others =>
111        NULL;
112    end case;
113 end if;
114 end if;
115 end process get_data_fsm;

120 transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
121     if rising_edge(clk) then
122         if ((get_state = send_header) or
123             (get_state = send_add_low) or
124             (get_state = send_add_high) or
125             (get_state = send_data)) then
126             case transmitter_state is
127                 when idle =>
128                     if ready = '1' and eot = '0' then
129                         next_transmitter_state <= set_data;
130                     end if;

131                 when set_data =>
132                     next_transmitter_state <= trans_data;

133                 when trans_data =>
134                     next_transmitter_state <= pause;

135                 when pause =>
136                     if eot = '1' then
137                         next_transmitter_state <= finished;
138                     end if;

139                 when finished =>
140                     next_transmitter_state <= idle;

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

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

161                 when wait_data =>

```

```

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

168
170         when read_data =>
            next_reader_state <= pause;

172
174         when pause =>
            if eoc = '0' then
                next_reader_state <= finished;
            end if;

176
178         when finished =>
            next_reader_state <= idle;

180
182         when others =>
            NULL;
        end case;
    end if;
184 end if;
end process read_fsm;

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

196 -- Outputs
with state select
198     data_ack <= '1' when wait_clear,
        'Z' when idle,
200         '0' when others;

202 with transmitter_state select
    write <= '1' when trans_data,
204         '0' when others;

206 uart_input <= idle when transmitter_state /= set_data and
    transmitter_state /= trans_data else
    header when get_state = send_header else
208     data_add_high when get_state = send_add_high else
    data_add_low when get_state = send_add_low else
210     data_data when get_state = send_data else
        idle;

212
214 uart_output <= clear_data when state = idle else
    idle when reader_state /= read_data else
    header when get_state = get_header else
216     data_data when get_state = get_data else
        idle;
218 end data_control_unit_arch;

```

---

#### Listing 10: mmu/header\_builder.vhd

```

-- builds a header to feed into the RS-232 link
2
library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;

6 library work;

8 entity header_builder is
    port (
10     read_write : in std_logic; -- 1 = read, 0 = write
        inst_data : in std_logic; -- 1 = inst, 0 = data
12     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
                                         -- instructions)
20
   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 1 in 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 =>
            if (get_state = finished) then
43                 next_state <= wait_clear;
            end if;

45         when wait_clear =>
            if (inst_req = '1') then
47                 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 =>
                if transmitter_state = finished then
69                     next_get_state <= send_add_high;
                end if;

71             when send_add_high =>
                if transmitter_state = finished then
73                     next_get_state <= get_header;
                end if;

77             when get_header =>
                if reader_state = finished then
79                     next_get_state <= get_add_low;
                end if;

81             when get_add_low =>
                if reader_state = finished then
83                     next_get_state <= get_add_high;
                end if;

87             when get_add_high =>
                if reader_state = finished then
89                     next_get_state <= get_data_low;
                end if;

91             when get_data_low =>
                if reader_state = finished then
93                     next_get_state <= get_data_high;
                end if;

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

```

```

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

103         when finished =>
            next_get_state <= idle;

105         when others =>
            NULL;
107         end case;
109     end if;
end process get_inst_fsm;

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

121             when set_data =>
                next_transmitter_state <= trans_data;

123             when trans_data =>
                next_transmitter_state <= pause;

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

131             when finished =>
                next_transmitter_state <= idle;

133             when others =>
                NULL;
137             end case;
139         end if;
end process transmit_fsm;

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

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

155             when read_data =>
                next_reader_state <= pause;

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

163             when finished =>
                next_reader_state <= idle;

165             when others =>
                NULL;
167             end case;
169         end if;
end process read_fsm;

```

```

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;
        end process switch_states;

181
    -- 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.
    );
end mmu_main;

```

```

    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 mmu_main;

32
architecture mmu_arch of mmu_main is
34  component mmu_control_unit is
    port (
36      eoc      : in std_logic; -- High on muart has finished collecting data ??
    eot      : in std_logic; -- High on muart has finished transmitting data.
38      ready     : in std_logic; -- ??
    data_read  : in std_logic; -- High if the data line is requesting a read,
        low for write.
40      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.
42      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.
44      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.
46      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.
48      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.
50      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.
52      clk       : in  std_logic
    );
54  end component;

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

64  component header_decoder is
    port (
66      read_write : out  std_logic; -- 1 = read, 0 = write
    fetch_request : out  std_logic;
68      inst_data  : out  std_logic; -- 1 = inst, 0 = data
    header       : in  std_logic_vector(7 downto 0)
70  );
72  end component;

74  component minimal_uart_core is
    port(
76      clock : in  std_logic;
    eoc      : out  std_logic;
    outp     : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
78      rxd   : in  std_logic;
    txd     : out  std_logic;
80      eot   : out  std_logic;
    inp     : in  std_logic_vector(7 downto 0);
82      ready : out  std_logic;
    wr      : in  std_logic
84  );
86  end component;

```

```

component reg8 IS
88     port(
90         I      : in  std_logic_vector(7 downto 0);
          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 ??
99     signal eot      : std_logic; -- High on muart has finished transmitting data.
100    signal ready     : std_logic; -- ??
101    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.
103    signal rw        : std_logic; -- High if latest input packet had read/write
        set.
104
105    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.
107    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.

109
110    signal muart_out : std_logic_vector(7 downto 0);
111    signal muart_in  : std_logic_vector(7 downto 0);
112    signal header_in : std_logic_vector(7 downto 0);
113    signal header_out : std_logic_vector(7 downto 0);
114    signal inst_data_high_enable : std_logic;
115    signal inst_data_low_enable  : std_logic;
116    signal data_data_enable      : std_logic;
117    signal data_line_tri        : std_logic_vector(7 downto 0);
118
begin
119
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);
121
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);
123
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));
125
126    dd : reg8 port map (muart_out, clk, data_data_enable, '0', data_line_tri);

127
128    with muart_input select
        muart_in <= header_out                                when header,
129                    "0000" & inst_add(11 downto 8)            when inst_add_high,
130                    inst_add(7 downto 0)                      when inst_add_low,
131                    '0' & data_add(15 downto 9)                when data_add_high,
132                    data_add(8 downto 1)                      when data_add_low,
133                    data_line                                 when data_data,
134                    (others => '0')                          when others;
135
136
137    route_output : process(muart_output, muart_out) begin
138        header_in <= (others => '0');
        inst_data_high_enable <= '0';
139        inst_data_low_enable <= '0';
140        data_data_enable <= '0';
141        case muart_output is
142            when header =>
143                header_in <= muart_out;
144
145            when inst_data_high =>
146                inst_data_high_enable <= '1';
147
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);
53 signal data_read      : std_logic;
54 signal data_req       : std_logic;
55 signal data_ack       : std_logic;
56 signal clk            : std_logic;
57 signal receive_pin    : std_logic;
58 signal transfer_pin   : std_logic;

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

62
63 signal current_rcv : std_logic_vector(7 downto 0);
64 signal current_snd : std_logic_vector(7 downto 0);
begin
65 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);

66
67 rxd <= transfer_pin;
68 receive_pin <= txd;

69
70
71 clk_gen : process begin
72     clk <= '0';
73     wait for 10 ns;
74     clk <= '1';
75     wait for 10 ns;
76 end process;

77
78 inst_test : process
79     type pattern_type is record
80         inst_add      : std_logic_vector(11 downto 0);
81         rcv_head      : std_logic_vector( 7 downto 0);
82         snd_head      : std_logic_vector( 7 downto 0);
83         inst_data      : std_logic_vector(15 downto 0);
84     end record;
85     type pattern_array is array (natural range <>) of pattern_type;
86     constant patterns : pattern_array :=
87 --         inst_add      rcv_head      snd_head      inst_data
88         ((x"000", x"81", x"00", x"83A7"),
89          (x"001", x"81", x"00", x"4F5E"),
90          (x"101", x"81", x"00", x"5937"),
91          (x"051", x"81", x"00", x"A8F2"));
92 begin
93     wr <= '0';
94     for i in patterns'range loop
95         wait for 10000 ns;
96         inst_add <= patterns(i).inst_add;
97         wait for 20 ns;
98         inst_req <= '0';

99
100         wait until eoc'event;
101         assert outp = patterns(i).rcv_head
102             report "Bad header expected '" & str(patterns(i).rcv_head) & "' recieved '"
103                 & str(outp) & "'"
104             severity error;
105         wait until eoc'event;

106         assert false report "passed header" severity note;

107
108         wait until eoc'event;
109         assert outp = patterns(i).inst_add(7 downto 0)
110             report "Bad address low expected '" & str(patterns(i).inst_add(7 downto 0)) &
111                 "' recieved '" & str(outp) & "'"
112             severity error;
113         wait until eoc'event;

114         assert false report "passed address low" severity note;

115
116         wait until eoc'event;
117         assert outp = "0000" & patterns(i).inst_add(11 downto 8)
118             report "Bad address high expected '" & str(patterns(i).inst_add(11 downto
119                 8)) & "' recieved '" & str(outp) & "'"
120             severity error;

```



```

122     wait until eoc'event;

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

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

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

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

174     wait for 100 ns;
176     inp <= patterns(i).inst_data(7 downto 0);
178     wait for 20 ns;
180     wr <= '1';
182     wait for 20 ns;
184     wr <= '0';
186     wait until eot'event;
188     wait until eot'event;

190     assert inst_ack = '1'
192         report "receipt not acknowledged"
194         severity error;

196     assert inst_data = patterns(i).inst_data
198         report "Wrong data recieve expected '" & str(patterns(i).inst_data) & "'
200             recieved '" & str(inst_data) & "'"
202         severity error;

204     assert false report "finished transmission" severity note;

206     wait for 20 ns;

208     inst_req <= '1';
210 end loop;
212 wait;
214 end process;
216 end tb;

```

---

#### Listing 15: mmu/mmu\_types.vhd

```

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

4 package mmu_types is
    type uart_input_state is (header, inst_add_high, inst_add_low, data_add_high,
        data_add_low, data_data, idle);
6    type uart_output_state is (header, inst_data_high, inst_data_low, data_data,
        clear_data, idle);
end mmu_types;

```

#### Listing 16: mmu/uart/BRG.vhd

```

1  -----
2  --* Minimal UART ip core *
3  --* Author: Arao Hayashida Filho      arao@medinovacao.com.br *
4  --* *
5  -----
6  --* *
7  --* Copyright (C) 2009 Arao Hayashida Filho *
8  --* *
9  --* This source file may be used and distributed without *
10 --* restriction provided that this copyright statement is not *
11 --* removed from the file and that any derivative work contains *
12 --* the original copyright notice and the associated disclaimer. *
13 --* *
14 --* This source file is free software; you can redistribute it *
15 --* and/or modify it under the terms of the GNU Lesser General *
16 --* Public License as published by the Free Software Foundation; *
17 --* either version 2.1 of the License, or (at your option) any *
18 --* later version. *
19 --* *
20 --* This source is distributed in the hope that it will be *
21 --* useful, but WITHOUT ANY WARRANTY; without even the implied *
22 --* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *
23 --* PURPOSE. See the GNU Lesser General Public License for more *
24 --* details. *
25 --* *
26 --* You should have received a copy of the GNU Lesser General *
27 --* Public License along with this source; if not, download it *
28 --* from http://www.opencores.org/lgpl.shtml *
29 --* *
30 -----
31
32 library ieee;
33 use ieee.std_logic_1164.all;
34 use ieee.std_logic_arith.all;
35 use ieee.std_logic_unsigned."+";
36
37 entity br_generator is
38     generic (divider_width: integer := 16);
39     port (
40         clock      : in  std_logic;
41         rx_enable   : in  std_logic;
42         clk_txd     : out std_logic;
43         tx_enable   : in  std_logic;
44         clk_serial  : out std_logic
45     );
46 end br_generator;
47
48 architecture principal of br_generator is
49     -- change the following constant to your desired baud rate
50     -- one hz equal to one bit per second
51     signal count_brg      : std_logic_vector(divider_width - 1 downto 0) := (others =>
52         '0');
53     signal count_brg_txd : std_logic_vector(divider_width - 1 downto 0) := (others =>
54         '0');
55     constant brdvd        : std_logic_vector(divider_width - 1 downto 0) := x"0516";
56     -- 38400 bps @ 50MHz
57
58 begin

```

```

txd : process (clock)
57 begin
    if (rising_edge(clock)) then
59         if (count_brg_txd = brdvd) then
            clk_txd <= '1';
            count_brg_txd <= (others => '0');
61         elsif (tx_enable = '1') then
            clk_txd <= '0';
            count_brg_txd <= count_brg_txd + 1;
63         else
            clk_txd <= '0';
            count_brg_txd <= (others => '0');
65         end if;
67     end if;
69 end process txd;

71
rx : process (clock)
73 begin
    if (rising_edge(clock)) then
75         if (count_brg=brdvd) then
            count_brg <= (others => '0');
            clk_serial <= '1';
77         elsif (rx_enable = '1') then
            count_brg <= count_brg+1;
            clk_serial <= '0';
79         else
            count_brg <= '0' & brdvd(divider_width - 1 downto 1);
            clk_serial <= '0';
81         end if;
83     end if;
85 end process rx;
87 end principal;

```

#### Listing 17: mmu/muart/serial.vhd

```

1  --*****
2  --* Minimal UART ip core *
3  --* Author: Arao Hayashida Filho      arao@medinovacao.com.br *
4  --* *
5  --*****
6  --*
7  --* Copyright (C) 2009 Arao Hayashida Filho *
8  --*
9  --* This source file may be used and distributed without *
10 --* restriction provided that this copyright statement is not *
11 --* removed from the file and that any derivative work contains *
12 --* the original copyright notice and the associated disclaimer. *
13 --*
14 --* This source file is free software; you can redistribute it *
15 --* and/or modify it under the terms of the GNU Lesser General *
16 --* Public License as published by the Free Software Foundation; *
17 --* either version 2.1 of the License, or (at your option) any *
18 --* later version. *
19 --*
20 --* This source is distributed in the hope that it will be *
21 --* useful, but WITHOUT ANY WARRANTY; without even the implied *
22 --* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *
23 --* PURPOSE. See the GNU Lesser General Public License for more *
24 --* details. *
25 --*
26 --* You should have received a copy of the GNU Lesser General *
27 --* Public License along with this source; if not, download it *
28 --* from http://www.opencores.org/lgpl.shtml *
29 --*
30 --*****
31
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;

34
35 entity minimal_uart_core is
    port(
36         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
    );
    end component;
75
    begin
77        ready <= not(tx_enable);
        brg : br_generator port map (clock, rx_ck_enable, clk_txd, tx_ck_enable,
            clk_serial);
79        rx_ck_enable <= start or receiving;
        tx_ck_enable <= tx_enable or transmitting;
81
        start_detect : process(rxd, eocs)
83        begin
            if (eocs = '1') then
85                start <= '0';
            elsif (falling_edge(rxd)) then
87                start <= '1';
            end if;
89        end process start_detect;

91        rxd_states : process (clk_serial)
        begin
93            if (rising_edge(clk_serial)) then
                atual_state <= next_state;
95            end if;
        end process rxd_states;

97        rxd_state_machine : process(start, atual_state)
99        begin
            if (start = '1' or receiving = '1') then
101                case atual_state is
                    when s0 =>
103                        eocs <= '0';
                        if (start = '1') then
105                            next_state <= s1;
                            receiving <= '1';
107                        else
                            next_state <= s0;
109                            receiving <= '0';

```

```

        end if;
111
        when s1 =>
113            receiving <= '1';
            eocs <= '0';
115            next_state <= s2;

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

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

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

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

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

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

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

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

149        when others =>
            null;
151
153        end case;
155
157        end if;
159        end process rxd_state_machine;

161
163        rxd_shift : process(clk_serial)
165        begin
            if (rising_edge(clk_serial)) then
167                if (eocs = '0') then
                    data <= rxd & data(7 downto 1);
169                end if;
            end if;
171        end process rxd_shift;

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

179        process(atual_state)
181        begin
            if (atual_state=s9) then

```

```

183         outp <= data;
184     end if;
185 end process;

186
187 txd_states : process(clk_txd)
188 begin
189     if (rising_edge(clk_txd)) then
190         atual_state_txd <= next_state_txd;
191     end if;
192 end process txd_states;

193
194 txd_state_machine : process(atual_state_txd, tx_enable)
195 begin
196     case atual_state_txd is
197         when s0 =>
198             inpl <= inp;
199             eots <= '0';
200             if (tx_enable = '1') then
201                 txds <= '0';
202                 transmitting <= '1';
203                 next_state_txd <= s1;
204             else
205                 txds <= '1';
206                 transmitting <= '0';
207                 next_state_txd <= s0;
208             end if;
209
210         when s1 =>
211             txds <= inpl(0);
212             eots <= '0';
213             transmitting <= '1';
214             next_state_txd <= s2;
215
216         when s2 =>
217             txds <= inpl(1);
218             eots <= '0';
219             transmitting <= '1';
220             next_state_txd <= s3;
221
222         when s3 =>
223             txds <= inpl(2);
224             eots <= '0';
225             transmitting <= '1';
226             next_state_txd <= s4;
227
228         when s4 =>
229             txds <= inpl(3);
230             eots <= '0';
231             transmitting <= '1';
232             next_state_txd <= s5;
233
234         when s5 =>
235             txds <= inpl(4);
236             eots <= '0';
237             transmitting <= '1';
238             next_state_txd <= s6;
239
240         when s6 =>
241             txds <= inpl(5);
242             eots <= '0';
243             transmitting <= '1';
244             next_state_txd <= s7;
245
246         when s7 =>
247             txds <= inpl(6);
248             eots <= '0';
249             transmitting <= '1';
250             next_state_txd <= s8;
251
252         when s8 =>
253             txds <= inpl(7);
254             eots <= '0';
255             transmitting <= '1';

```

```

        next_state_txd <= s9;
257
        when s9 =>
259            txds          <= '1';
            eots           <= '1';
261            transmitting  <= '1';
            next_state_txd <= s0;
263
        when others =>
265            null;
267
        end case;
    end process txd_state_machine;
269
    tx_start:process (clock, wr,  eots)
271    begin
        if (eots = '1') then
273            tx_enable <= '0';
        elsif (falling_edge(clock)) then
275            if (wr = '1') then
                tx_enable <= '1';
277            end if;
        end if;
279    end process tx_start;

281    eot<=eots;

283    process (clock)
    begin
285        if (rising_edge(clock)) then
            txd <= txds;
287        end if;
    end process;
289
end principal ;

```

---

#### Listing 18: processor/alu.vhd

---

```

-----
2  -- Company:
   -- Engineer:
4  --
   -- Create Date:   18:59:20 09/18/2010
6  -- Design Name:
   -- Module Name:   alu - alu_arch
8  -- Project Name:
   -- Target Devices:
10 -- Tool versions:
   -- Description:
12 --
   -- Dependencies:
14 --
   -- Revision:
16 -- Revision 0.01 - File Created
   -- Additional Comments:
18 --
-----
20 library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
   use ieee.std_logic_arith.all;
24 --use ieee.std_logic_unsigned.all;

26
   library work;
28 use work.fulladder8;
   --use work.cpu.ALL;
30
   -- Uncomment the following library declaration if using
32 -- arithmetic functions with Signed or Unsigned values
   --use IEEE.NUMERIC_STD.ALL;
34

```

```

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

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

    end alu;
48

50 architecture alu_arch of alu is
    component fulladder8 IS
52        Port (A      : in    STD_LOGIC_VECTOR( 7 downto 0);
              B      : in    STD_LOGIC_VECTOR( 7 downto 0);
54              Cin    : in    STD_LOGIC;
              Sum     : out   STD_LOGIC_VECTOR( 7 downto 0);
56              Cout  : out   STD_LOGIC
              );
58    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
66 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
70    BEGIN
        -- use case statement to achieve
72        -- different operations of ALU

74        AdderCin <= '0';
        A <= (others => '0');
76        B <= (others => '0');
        output <= (others => '0');
78        C <= '0';
        N <= '0';
80        IF f = "0001" THEN -- Do AND operation
            output <= ry and rx;
82        ELSIF f = "0011" THEN -- Do OR operation
            output <= ry or rx;
84        ELSIF f = "0101" THEN
            output <= not rx;
86        ELSIF f = "0111" THEN -- Do XOR operation
            output <= ry xor rx;
88        ELSIF f = "1001" THEN -- Do ADD operation
            AdderCin <= '0';
            A <= ry;
            B <= rx;
            output <= Sum;
92        ELSIF f = "1011" THEN -- Do ADC operation
            AdderCin <= Cin;
            A <= ry;
            B <= rx;
            output <= Sum;
94        ELSIF f = "1101" THEN -- Do SUB operation
            AdderCin <= '1';
            A <= ry;
            B <= (not rx);
102        output <= Sum;
            ELSIF f = "1111" THEN -- Do SBB operation
104        AdderCin <= (not Cin);
            A <= ry;
            B <= (not rx);
106

```



```

    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 Normally (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)
45     );
46 end component;

48 signal ROE      : std_logic; -- Enable signals
49 signal R1E      : std_logic;
50 signal R2E      : std_logic;
51 signal input     : std_logic_VECTOR (15 downto 0);
52 signal Q0       : std_logic_VECTOR (15 downto 0);
53 signal Q1       : std_logic_VECTOR (15 downto 0);
54 signal Q2       : std_logic_VECTOR (15 downto 0);
55 BEGIN
56     reg_0 : reg16 port map(input, clk, ROE, '0', Q0);
57     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)
61 BEGIN
62     ROE <= '0';
63     R1E <= '0';
64     R2E <= '0';
65     IF enable = '1' THEN
66         case SelRi IS
67             WHEN "00" =>
68                 ROE <= '1';
69             WHEN "01" =>
70                 R1E <= '1';
71             WHEN "10" =>
72                 R2E <= '1';
73             WHEN others =>
74                 NULL; -- None of them are enabled
75         END CASE;
76     END IF;
77 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)
81 BEGIN
82     IF Sel8Bit = '0' THEN
83         input <= Ri;
84     ELSE
85         if SelHighByte = '1' THEN
86             input(15 downto 8) <= ByteInput;
87             case SelRi IS
88                 WHEN "00" =>
89                     input(7 downto 0) <= Q0(7 downto 0);
90                 WHEN "01" =>
91                     input(7 downto 0) <= Q1(7 downto 0);
92                 WHEN others =>
93                     input(7 downto 0) <= Q2(7 downto 0);
94             END CASE;
95         else
96             input(7 downto 0) <= ByteInput;
97             case SelRi IS
98                 WHEN "00" =>
99                     input(15 downto 8) <= Q0(15 downto 8);
100                WHEN "01" =>
101                    input(15 downto 8) <= Q1(15 downto 8);
102                WHEN others =>
103                    input(15 downto 8) <= Q2(15 downto 8);
104            END CASE;
105        END IF;
106    END IF;
107 end process;

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

```

## Listing 21: processor/cpu.vhd

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:      16:09:46 09/15/2010
6  -- Design Name:
7  -- Module Name:      cpu - cpu_arch
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 --use IEEE.STD_LOGIC_ARITH.ALL;
23 --use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 library work;
26 use work.alu;
27 use work.cu;
28 use work.ar;
29 use work.gpr;
30 use work.sr;
31 use work.pc;
32
33
34 ---- Uncomment the following library declaration if instantiating
35 ---- any Xilinx primitives in this code.
36 --library UNISIM;
37 --use UNISIM.VComponents.all;
38
39 entity cpu is
40
41     PORT(
42         -- instruction bus
43         inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
44         inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
45         inst_req  : out std_logic;                    -- Pulled low to request bus
46         usage.
47         inst_ack  : in  std_logic;                    -- Pulled high to inform of
48         request completion.
49         -- data bus
50         data_add  : out std_logic_vector(15 downto 0); -- Address lines.
51         data_line : inout std_logic_vector(7 downto 0); -- Data lines.
52         data_read : out std_logic;                    -- High for a read request,
53         low for a write request.
54         data_req  : out std_logic;                    -- Pulled low to request bus
55         usage.
56         data_ack  : inout std_logic;                  -- Pulled high to inform of
57         request completion.
58         -- extras
59         clk       : in  std_logic;
60         reset     : in  std_logic;
61     );
62
63 end cpu;
64
65 architecture cpu_arch of cpu is
66     component alu IS
67         Port (f : in  STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
68              rx : in  STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
69              ry : in  STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
70              ro : out STD_LOGIC_VECTOR (7 downto 0); -- Output Normaly (Ry)
71              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;
          enable       : in  STD_LOGIC;
          Sel8Bit      : in  STD_LOGIC;
          SelHighByte  : in  STD_LOGIC;
          ByteInput     : in  STD_LOGIC_VECTOR (7 downto 0);
          SelRi        : in  STD_LOGIC_VECTOR (1 downto 0); -- Select the address
                    register
          SelRo        : in  STD_LOGIC_VECTOR (1 downto 0); -- Select the address
                    register
          Ri           : in  STD_LOGIC_VECTOR (15 downto 0); -- The input
          Ro           : out STD_LOGIC_VECTOR (15 downto 0)); -- The output
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
       alu_Cin       : out STD_LOGIC; -- Carry in to ALU
85
       -- 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       : in  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,
198         sr   => sr_input
199     );
200     c: cu port map(
201         reset      => reset, -- '0' for reset
202         clock       => clk,  -- clock
203
204         alu_f       => alu_f, -- Function
205         alu_Cin     => alu_Cin, -- Carry into the ALU
206
207         -- General Purpose Registers
208         gpr_InSel   => gpr_InSel, -- select the input path (0 - cu, 1 - ALU)
209         gpr_en      => gpr_enable, -- enable write to GPR
210         gpr_SelRx   => gpr_SelRx, -- select GPR output x
211         gpr_SelRy   => gpr_SelRy, -- select GPR output y
212         gpr_SelRi   => gpr_SelRi, -- select GPR input
213         gpr_Ri      => gpr_RiCU, -- input to GPR
214         gpr_Rx      => alu_rx, -- Rx from GPR
215         --gpr_Ry    => alu_ry, -- Ry from GPR
216
217         -- Status Register
218         sr_en       => sr_enable, -- enable write to SR
219         sr_reset    => sr_reset, -- reset SR
220         sr_Ro       => sr_Ro, -- output from SR
221         -- control unit doesnt write to SR, the ALU does
222
223         -- Program Counter
224         pc_en       => pc_enable, -- enable write to PC
225         pc_reset    => pc_reset, -- reset PC
226         pc_Ri       => pc_Ri, -- input to PC
227         pc_Ro       => pc_Ro, -- output from PC
228
229         -- Address Registers
230         ar_en       => ar_enable, -- enable write to AR
231         ar_SelRi    => ar_SelRi, -- select AR in
232         ar_SelRo    => ar_SelRo, -- select AR out
233         ar_sel8Bit  => ar_sel8Bit,
234         ar_selHByte => ar_selHByte,
235         ar_ByteIn   => ar_ByteIn,
236         ar_Ri       => ar_Ri, -- input to AR
237         ar_Ro       => ar_Ro, -- output from AR
238
239         -- Instruction memory
240         inst_add    => inst_add, -- Instruction address
241         inst_data   => inst_data, -- Instruction data
242         inst_req    => inst_req, -- Request
243         inst_ack    => inst_ack, -- Instruction obtained
244
245         data_add    => data_add, -- Data address
246         data_data   => data_line, -- Data
247         data_read   => data_read, -- 1 for read, 0 for write
248         data_req    => data_req, -- Request
249         data_ack    => data_ack, -- Data written to/ read from
250     );
251     address : ar port map(
252         clk         => clk,
253         enable      => ar_enable,
254         Sel8Bit     => ar_Sel8Bit,
255         SelHighByte => ar_selHByte,
256         ByteInput   => ar_ByteIn,
257         SelRi       => ar_SelRi,
258         SelRo       => ar_SelRo,
259         Ri          => ar_Ri,
260         Ro          => ar_Ro
261     );
262     g : gpr port map(
263         clk         => clk,
264         enable      => gpr_enable,
265         SelRx       => gpr_SelRx,
266         SelRy       => gpr_SelRy,
267         SelRi       => gpr_SelRi,
268         SelIn       => gpr_InSel,

```



```

271         RiCU    => gpr_RiCU,
           RiCDB   => gpr_RiCDB,
           Rx      => alu_rx,
273         Ry      => alu_ry
           );
275 s : sr port map(
           clk      => clk,
277         enable  => sr_enable,
           reset    => sr_reset,
279         Ri       => sr_input,
           Ro       => sr_Ro
281       );
   programcounter: pc port map(
283         clk      => clk,
           enable  => pc_enable,
285         reset    => pc_reset,
           Ri       => pc_Ri,
287         Ro       => pc_Ro
           );
289 end cpu_arch;

```

## Listing 22: processor/cu.vhd

```

1  -----
   -- Company:
3  -- Engineer:
   --
5  -- Create Date: 18:59:20 09/18/2010
   -- Design Name:
7  -- Module Name: cu - Behavioral
   -- Project Name:
9  -- Target Devices:
   -- Tool versions:
11 -- Description: The control unit
   --
13 -- Dependencies:
   --
15 -- Revision:
   -- Revision 0.01 - File Created
17 -- Additional Comments:
   --
19 -----
   library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.NUMERIC_STD.ALL;
23 use ieee.std_logic_arith.all;
   --use ieee.std_logic_unsigned.all;
25
27 library work;
   --use work.fulladder;
29 --use work.cpu.ALL;
31 -- Uncomment the following library declaration if using
   -- arithmetic functions with Signed or Unsigned values
33 --use IEEE.NUMERIC_STD.ALL;
35 -- Uncomment the following library declaration if instantiating
   -- any Xilinx primitives in this code.
37 --library UNISIM;
   --use UNISIM.VComponents.all;
39
   entity cu is
41     Port (reset      : in STD_LOGIC;           -- '0' for reset
           clock       : in STD_LOGIC;           -- clock
43           alu_f       : out STD_LOGIC_VECTOR (3 downto 0); -- Function
45           alu_Cin     : out STD_LOGIC;           -- Carry in to ALU
47           -- General Purpose Registers
           gpr_InSel    : out STD_LOGIC;           -- select the input path (0
               - cu, 1 - ALU)

```

```

49     gpr_en      : out STD_LOGIC;           -- enable write to GPR
    gpr_SelRx    : out STD_LOGIC_VECTOR (2 downto 0); -- select GPR output x
51     gpr_SelRy   : out STD_LOGIC_VECTOR (2 downto 0); -- select GPR output y
    gpr_SelRi    : out STD_LOGIC_VECTOR (2 downto 0); -- select GPR input
53     gpr_Ri      : out STD_LOGIC_VECTOR (7 downto 0); -- input to GPR
    gpr_Rx       : in STD_LOGIC_VECTOR (7 downto 0);  -- output Rx from GPR
55     --gpr_Ry     : in STD_LOGIC_VECTOR (7 downto 0);  -- output Ry from GPR ,
        not used

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

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

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

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

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

91     );
end cu;
93

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

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

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

111 -- Decoded data
signal rx : std_logic_vector(2 downto 0);
113 signal ry : std_logic_vector(2 downto 0);
signal ay : std_logic_vector(1 downto 0);
115

-- Indicates what needs to be executed
117 signal write_gpr : std_logic;

```

```

119     signal write_sr      : std_logic;
120     signal write_pc      : std_logic;
121     signal write_ar      : std_logic;
122     signal write_memory  : std_logic;

123     -- full adders
124     signal A16           : std_logic_vector(15 downto 0);
125     signal B16           : std_logic_vector(15 downto 0);
126     signal AdderCin16    : std_logic;
127     signal Sum16         : std_logic_vector(15 downto 0);
128     signal AdderCout16   : std_logic;

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

131

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

134
135     -- Process instruction
136     -- Assumes all instructions are valid
137     process(clock, state, opcode, gpr_Rx, sr_Ro, pc_Ro, ar_Ro, inst_data, inst_ack,
138             data_data, data_ack,
139             rx, ry, ay, v, write_gpr, write_sr, write_pc, write_ar, write_memory)
140     BEGIN
141         if rising_edge(clock) then
142             case state is
143                 when reset_state =>
144                     sr_reset <= '0';
145                     pc_reset <= '0';
146                     next_state <= fetch;
147
148                 when fetch =>
149                     sr_reset <= '1';
150                     pc_reset <= '1';
151
152                     gpr_en <= '0';
153                     sr_en <= '0';
154                     pc_en <= '0';
155                     ar_en <= '0';
156
157                     write_gpr <= '0';
158                     write_sr <= '0';
159                     write_pc <= '0';
160                     write_ar <= '0';
161                     write_memory <= '0';
162
163                     inst_add <= pc_Ro(11 downto 0);
164                     if inst_ack = '0' then
165                         inst_req <= '1';
166                     else
167                         opcode <= inst_data;
168                         inst_req <= '0';
169
170                         -- increment program counter
171                         AdderCin16 <= '1';
172                         A16 <= PC_Ro;
173                         B16 <= "0000000000000000";
174                         pc_Ri <= Sum16;
175                         pc_en <= '1';
176
177                         next_state <= decode;
178                     end if;
179                 when decode =>
180                     pc_en <= '0';
181
182                     -- ALU
183                     if opcode(15) = '0' and opcode(10) = '0' and not opcode(14 downto 11) =
184                         "0010" then
185
186                         ry <= opcode(7 downto 5);
187                         rx <= opcode(2 downto 0);
188
189                         gpr_SelRy <= ry;

```

```

189     gpr_SelRx <= rx;
190     gpr_SelRi <= ry;
191     gpr_InSel <= '1';
192     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
196         others do
197             write_gpr <= '1';
198         end if;

199     write_sr <= '1';
200     next_state <= execute;

201 -- Branching
202 elsif opcode(11 downto 10) = "11" then

205     v <= "00000000"; -- initialise v

207     if opcode(15) = '1' then
208         case opcode(14 downto 12) is
209             when "000" => -- BEQ
210                 if sr_Ro(0) = '1' then -- Z=1
211                     v <= opcode(9 downto 2);
212                 end if;
213             when "001" => -- BNE
214                 if sr_Ro(0) = '0' then -- Z=0
215                     v <= opcode(9 downto 2);
216                 end if;
217             when "010" => -- BLT
218                 if sr_Ro(0) = '0' and sr_Ro(2) = '1' then -- Z=0 and N=1
219                     v <= opcode(9 downto 2);
220                 end if;
221             when "011" => -- BGT
222                 if sr_Ro(0) = '0' and sr_Ro(2) = '0' then -- Z=0 and N=0
223                     v <= opcode(9 downto 2);
224                 end if;
225             when "100" => -- BC
226                 if sr_Ro(1) = '1' then -- C=1
227                     v <= opcode(9 downto 2);
228                 end if;
229             when "101" => -- BNC
230                 if sr_Ro(1) = '0' then -- C=0
231                     v <= opcode(9 downto 2);
232                 end if;
233             when "110" => -- RJMP
234                 v <= opcode(9 downto 2);

235             when others =>
236                 v <= "00000000";
237         end case;

239     -- PC <- PC + v
240     AdderCin16 <= '0';
241     A16 <= PC_Ro;
242     B16 <= "00000000" & v;
243     pc_Ri <= Sum16;

245     elsif opcode(15 downto 12) = "0111" then -- JMP
246         ay <= opcode(6 downto 5);

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

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

```

```

261     -- Addressing
262 else
263     gpr_Insel <= '0';
264
265     case opcode(12 downto 10) is
266
267         when "001" => -- Load
268             if opcode(15) = '1' then -- immediate
269                 rx <= '0' & opcode(1 downto 0);
270                 v <= opcode(9 downto 2);
271
272                 -- rx <- v
273                 gpr_SelRi <= rx;
274                 gpr_Ri <= v;
275                 write_gpr <= '1';
276                 next_state <= execute;
277             else -- direct
278                 rx <= opcode(2 downto 0);
279                 ay <= opcode(6 downto 5);
280
281                 -- rx <- [ay]
282                 gpr_SelRi <= rx;
283                 ar_selRo <= ay;
284                 data_add <= ar_Ro;
285                 data_read <= '1';
286                 if data_ack = '0' then -- request data
287                     data_req <= '1';
288                 else -- data obtained
289                     gpr_Ri <= data_data;
290                     data_req <= '0';
291                     write_gpr <= '1';
292
293                     case opcode(14 downto 13) is
294                         when "01" => -- auto increment
295                             AdderCin16 <= '1';
296                             A16 <= ar_Ro;
297                             B16 <= "0000000000000000";
298                             ar_selRi <= ay;
299                             ar_sel8bit <= '0';
300                             ar_Ri <= Sum16;
301                             write_ar <= '1';
302                         when "10" => -- auto decrement
303                             AdderCin16 <= '0';
304                             A16 <= ar_Ro;
305                             B16 <= "1111111111111111";
306                             ar_selRi <= ay;
307                             ar_sel8bit <= '0';
308                             ar_Ri <= Sum16;
309                             write_ar <= '1';
310                         when others =>
311                             -- do nothing
312                     end case;
313                     next_state <= execute;
314                 end if;
315             end if;
316
317         when "101" => -- Store
318             if opcode(15) = '1' then -- immediate
319                 ay <= opcode(1 downto 0);
320                 v <= opcode(9 downto 2);
321
322                 -- [ay] <- v
323                 ar_selRo <= ay;
324                 data_add <= ar_Ro;
325                 data_read <= '0';
326                 data_data <= v;
327                 write_memory <= '1';
328                 next_state <= execute;
329             else -- direct
330                 rx <= opcode(2 downto 0);
331                 ay <= opcode(6 downto 5);
332
333                 -- [ay] <- rx

```

```

335     gpr_selRx <= rx;
336     ar_selRo <= ay;
337     data_add <= ar_Ro;
338     data_read <= '0';
339     data_data <= gpr_Rx;
340     write_memory <= '1';
341
342     case opcode(14 downto 13) is
343     when "01" => -- auto increment
344         AdderCin16 <= '1';
345         A16 <= ar_Ro;
346         B16 <= "0000000000000000";
347         ar_selRi <= ay;
348         ar_sel8bit <= '0';
349         ar_Ri <= Sum16;
350         write_ar <= '1';
351     when "10" => -- auto decrement
352         AdderCin16 <= '0';
353         A16 <= ar_Ro;
354         B16 <= "1111111111111111";
355         ar_selRi <= ay;
356         ar_sel8bit <= '0';
357         ar_Ri <= Sum16;
358         write_ar <= '1';
359     when others =>
360         -- do nothing
361     end case;
362     next_state <= execute;
363 end if;
364
365 when "100" => -- Move
366     if opcode(9) = '1' then
367         rx <= opcode(2 downto 0);
368         ay <= opcode(6 downto 5);
369
370         -- ayn <- rx
371         gpr_selRx <= rx;
372         ar_selRi <= ay;
373         ar_sel8bit <= '1';
374         ar_ByteIn <= gpr_Rx;
375
376         if opcode(8) = '1' then -- high
377             ar_selHByte <= '1';
378         else -- low
379             ar_selHByte <= '0';
380         end if;
381
382         write_ar <= '1';
383         next_state <= execute;
384
385     elsif opcode(4) = '1' then
386         rx <= opcode(7 downto 5);
387         ay <= opcode(1 downto 0);
388
389         -- rx <- ayn
390         gpr_selRi <= rx;
391         ar_selRo <= ay;
392
393         if opcode(3) = '1' then -- high
394             gpr_Ri <= ar_Ro(15 downto 8);
395         else -- low
396             gpr_Ri <= ar_Ro(7 downto 0);
397         end if;
398
399         write_gpr <= '1';
400         next_state <= execute;
401
402     else
403         rx <= opcode(2 downto 0);
404         ry <= opcode(7 downto 5);
405
406         -- ry <- rx
407         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;
22
    entity fulladder is
24        Port (Ax      : in    STD_LOGIC;
                Bx      : in    STD_LOGIC;
26                Ci      : in    STD_LOGIC;
                Sx      : out   STD_LOGIC;
28                Co      : out   STD_LOGIC
                );
30    end fulladder;

32
    architecture arch_fulladder of fulladder is
34    BEGIN
        process(Ax, Bx, Ci)
36        BEGIN
            Sx <= (Ax XOR Bx) XOR Ci;
38            Co <= (Ax and Bx) or (Ax and Ci) OR (Bx AND Ci);
            end process;
40    end arch_fulladder;

42 -----

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

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

78        Cout <= Carry(8);
    end arch_fulladder8;
80
82 -----

82    library IEEE;
84    use IEEE.STD_LOGIC_1164.ALL;

86    entity fulladder16 is
        Port (A      : in    STD_LOGIC_VECTOR( 15 downto 0);
88                B      : in    STD_LOGIC_VECTOR( 15 downto 0);
                Cin   : in    STD_LOGIC;
90                Sum     : out   STD_LOGIC_VECTOR( 15 downto 0);
                Cout  : out   STD_LOGIC
92                );
    end fulladder16;

```



```

94  architecture arch_fulladder16 of fulladder16 is
96      component fulladder IS
100          Port (Ax      : in    STD_LOGIC;
102                 Bx      : in    STD_LOGIC;
104                 Ci      : in    STD_LOGIC;
106                 Sx      : out   STD_LOGIC;
108                 Co      : out   STD_LOGIC
110             );
112      end component;
114      signal Carry      : std_logic_vector(16 downto 0);
116  BEGIN
118      Carry(0) <= Cin;

120      FA0:  fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
122      FA1:  fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
124      FA2:  fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
126      FA3:  fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
128      FA4:  fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
130      FA5:  fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));
132      FA6:  fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
134      FA7:  fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));
136      FA8:  fulladder PORT MAP(A(8), B(8), Carry(8), Sum(8), Carry(9));
138      FA9:  fulladder PORT MAP(A(9), B(9), Carry(9), Sum(9), Carry(10));
140      FA10: fulladder PORT MAP(A(10), B(10), Carry(10), Sum(10), Carry(11));
142      FA11: fulladder PORT MAP(A(11), B(11), Carry(11), Sum(11), Carry(12));
144      FA12: fulladder PORT MAP(A(12), B(12), Carry(12), Sum(12), Carry(13));
146      FA13: fulladder PORT MAP(A(13), B(13), Carry(13), Sum(13), Carry(14));
148      FA14: fulladder PORT MAP(A(14), B(14), Carry(14), Sum(14), Carry(15));
150      FA15: fulladder PORT MAP(A(15), B(15), Carry(15), Sum(15), Carry(16));

152      Cout <= Carry(16);
154  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
12      Port (A      : in    STD_LOGIC_VECTOR( 7 downto 0);
14             B      : in    STD_LOGIC_VECTOR( 7 downto 0);
16             Cin    : in    STD_LOGIC;
18             Sum     : out   STD_LOGIC_VECTOR( 7 downto 0);
20             Cout    : out   STD_LOGIC
22         );
24  end component;
26  -- Specifies which entity is bound with the component.
28  for fulladder8_0: fulladder8 use entity work.fulladder8;
30      signal A,B,Sum      : STD_LOGIC_VECTOR (7 downto 0);
32      signal Cin,Cout     : STD_LOGIC;
34  begin
    -- Component instantiation.
    fulladder8_0: fulladder8 port map (A => A, B => B, Cin => Cin, Sum => Sum, Cout
    => Cout);

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

```

```

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
34      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
36      Ry        : out   STD_LOGIC_VECTOR (7 downto 0); -- The Ry output
37 end gpr;

39
40 architecture gpr_arch of gpr is
41   component reg8 IS
42     port(I      : in    std_logic_vector(7 downto 0);
43          clock   : in    std_logic;
44          enable  : in    std_logic;
45          reset   : in    std_logic;
46          Q       : out   std_logic_vector(7 downto 0)
47     );
48   end component;
49
50   signal reset: std_logic := '0';
51   signal input: std_logic_VECTOR (7 downto 0);
52   signal R0E  : std_logic; -- Enable signals
53   signal R1E  : std_logic;
54   signal R2E  : std_logic;
55   signal R3E  : std_logic;
56   signal R4E  : std_logic;
57   signal R5E  : std_logic;
58   signal R6E  : std_logic;
59   signal R7E  : std_logic;
60   signal Q0   : std_logic_VECTOR (7 downto 0);
61   signal Q1   : std_logic_VECTOR (7 downto 0);
62   signal Q2   : std_logic_VECTOR (7 downto 0);
63   signal Q3   : std_logic_VECTOR (7 downto 0);
64   signal Q4   : std_logic_VECTOR (7 downto 0);
65   signal Q5   : std_logic_VECTOR (7 downto 0);
66   signal Q6   : std_logic_VECTOR (7 downto 0);
67   signal Q7   : std_logic_VECTOR (7 downto 0);
68 BEGIN
69   reg_0 : reg8 port map(input, clk, R0E, reset, Q0);
70   reg_1 : reg8 port map(input, clk, R1E, reset, Q1);
71   reg_2 : reg8 port map(input, clk, R2E, reset, Q2);
72   reg_3 : reg8 port map(input, clk, R3E, reset, Q3);
73   reg_4 : reg8 port map(input, clk, R4E, reset, Q4);
74   reg_5 : reg8 port map(input, clk, R5E, reset, Q5);
75   reg_6 : reg8 port map(input, clk, R6E, reset, Q6);
76   reg_7 : reg8 port map(input, clk, R7E, reset, Q7);
77
78   -- Select where the input should come from
79   SelectInput: process(SelIn, RiCDB, RiCU)
80   BEGIN
81     IF SelIn = '1' THEN
82       input <= RiCDB;
83     ELSE
84       input <= RiCU;
85     END IF;
86   END process;
87
88   -- Set Ri the input
89   SetInput: process(clk, enable, SelRi)
90   BEGIN
91     R0E <= '0';
92     R1E <= '0';
93     R2E <= '0';
94     R3E <= '0';
95     R4E <= '0';
96     R5E <= '0';
97     R6E <= '0';
98     R7E <= '0';
99     IF enable = '1' THEN
100      case SelRi IS
101      WHEN "000" =>
102        R0E <= '1';
103      WHEN "001" =>

```

```

105         R1E <= '1';
        WHEN "010" =>
            R2E <= '1';
107         WHEN "011" =>
            R3E <= '1';
109         WHEN "100" =>
            R4E <= '1';
111         WHEN "101" =>
            R5E <= '1';
113         WHEN "110" =>
            R6E <= '1';
115         WHEN "111" =>
            R7E <= '1';
117         WHEN others =>
            NULL; -- None of them are enabled
119     end case;
    END IF;
121 end process;

123 -- Set the Rx output
    WITH SelRx SELECT
125     Rx <= Q0 WHEN "000",
            Q1 WHEN "001",
127            Q2 WHEN "010",
            Q3 WHEN "011",
129            Q4 WHEN "100",
            Q5 WHEN "101",
131            Q6 WHEN "110",
            Q7 WHEN others;
133
    -- Set the Ry output
135     WITH SelRy SELECT
        Ry <= Q0 WHEN "000",
137            Q1 WHEN "001",
            Q2 WHEN "010",
139            Q3 WHEN "011",
            Q4 WHEN "100",
141            Q5 WHEN "101",
            Q6 WHEN "110",
143            Q7 WHEN others;

145 end gpr_arch;

```

Listing 26: processor/gpr\_tb.vhd

```

library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

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

8 architecture behav of gpr_tb is
    -- Declaration of the component that will be instantiated.
10     component gpr
        Port(clk          : IN std_logic;                -- Clock
12            enable       : IN std_logic;                -- Enable input
            (output is always enabled)
            SelRx, SelRy, SelRi : IN std_logic_vector(2 DOWNT0 0); -- Selecti which
            registers to use
14            Ri            : IN std_logic_vector(7 DOWNT0 0); -- Input
            Rx, Ry         : OUT std_logic_vector(7 DOWNT0 0)); -- Outputs
16     end component;
    -- Specifies which entity is bound with the component.
18     for gpr_0: gpr use entity work.gpr;
        signal clk, enable      : std_logic;
20        signal SelRx, SelRy, SelRi : std_logic_vector(2 DOWNT0 0);
        signal Ri, Rx, Ry       : std_logic_vector(7 DOWNT0 0);
22     begin
        -- Component instantiation.
24     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.
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
    SelRy <= "010";
82    wait for 20 ns;
    assert (Ry = "01010001") report "Read from Ry failed #3" severity error;

84    -- 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)
    SelRy <= "000";
92    wait for 20 ns;
    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)
       begin
36           IF reset = '1' THEN
               Q <= (others => '0');
38           ELSIF rising_edge(clock) then
               if enable = '1' then
40                 Q <= I;
42             end if;
           end if;
44
       end process;

46 end behv;
48
-----
50 library ieee;

```

```

52 use ieee.std_logic_1164.all;

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

62
architecture behv of reg16 is
64 begin

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

76     end process;

78 end behv;

```

---

#### Listing 28: processor/spr.vhd

```

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

20 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

22
library work;
24 use work.reg16;

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

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

```

```

BEGIN
44   reg_sr : reg16 port map(Ri, clk, enable, reset, Ro);
end sr_arch;
46
-----
48   library IEEE;
50   use IEEE.STD_LOGIC_1164.ALL;

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

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

```

#### Listing 29: processor/spr\_tb.vhd

```

1  -----
   library ieee;
3  use ieee.std_logic_1164.all;
   --use ieee.std_logic_unsigned.all;
5  --use ieee.std_logic_arith.all;

7  entity spr_TB is          -- entity declaration
   end spr_TB;

9

11  architecture TB of spr_TB is

   component sr
13   Port (clk      : in  STD_LOGIC;
         enable   : in  STD_LOGIC;          -- Enable write
15         reset   : in  STD_LOGIC;          -- Reset the register
         Ri       : in  STD_LOGIC_VECTOR (15 downto 0); -- The input to the SPR
17         Ro      : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SPR
   end component;

19

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

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

33   signal pc_enable : std_logic;
   signal pc_reset   : std_logic;
35   signal pc_Ri      : std_logic_vector(15 downto 0);
   signal pc_Ro      : std_logic_vector(15 downto 0);
37

   signal T_clk      : std_logic;
39

```



```

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

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

53   process

55       variable err_cnt: integer :=0;

57   begin

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

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

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

79       -- Disable sr, pc still enabled
81       sr_enable <= '0';
        sr_Ri <= "0101010101010101";
83       pc_Ri <= "1010101010101010";
        wait for 20 ns;
85       assert (sr_Ro = "1001100101110100") report "Wrote to sr while disabled" severity
            error;
        assert (pc_Ro = "1010101010101010") report "Read pc #3 failed" severity error;

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

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

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

105

107      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
116     for TB
117     end for;
118 end CFG_TB;

```

---

### Listing 30: assembler.rb

```

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

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

14 # For 8-bit twos complement
15 def twos_complement(num)
16     return 256 + num
17 end
18

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

28
29 def immediate(chunk, symbols=nil, move_from=nil)
30     v = chunk.to_i
31     if v == 0 && chunk != "0" && symbols != nil
32         assert(chunk + " is not a valid symbol") {symbols.include?(chunk)}
33         move_to = symbols[chunk]
34         diff = move_to - move_from
35         if diff < 1
36             diff = twos_complement(diff)
37         end
38         return diff
39     else
40         assert(chunk + " is not a valid immediate") {v >= -127 && v < 128}
41         if v < 1
42             v = twos_complement(v)
43         end
44         return v
45     end
46 end

48
49 def register(chunk, num_registers)
50     x = chunk[1..1].to_i
51     assert(chunk + " is not a valid register") {x >= 0 && x < num_registers}
52     return x
53 end
54

56 def auto(chunk)
57     if chunk[-1..-1] == "+"
58         return 0x08
59     elsif chunk[-1..-1] == "-"

```

```

60     return 0x10
61   else
62     return 0x00
63   end
64 end

66
67 def convert(lines)
68   table = first_pass(lines)
69   return second_pass(lines, table)
70 end

72
73 def first_pass(lines)
74   instruction = 0
75   symbols = {}
76   lines.each do |line|
77     chunks = line.sub(",", " ").split
78     case chunks[0]
79       when "LDI", "LD", "STI", "ST", "MV", "AND", "OR", "NOT", "XOR", "ADD", "ADC",
80         "SUB", "SBB", "NEG", "CMP", "BEQ", "BNE", "BLT", "BGT", "BC", "BNC", "RJMP",
81         "JMP"
82         instruction += 1
83       when "label:"
84         symbols[chunks[1]] = instruction
85       end
86     end
87   return symbols
88 end

90 def second_pass(lines, symbols)
91   line_no = 0
92   output = []
93   lines.each do |line|
94     label = line.sub(",", " ").split[0]
95     case label
96       when "LDI", "LD", "STI", "ST", "MV", "AND", "OR", "NOT", "XOR", "ADD", "ADC",
97         "SUB", "SBB", "NEG", "CMP", "BEQ", "BNE", "BLT", "BGT", "BC", "BNC", "RJMP",
98         "JMP"
99         line_no += 1
100        output.push(convert_line(line, symbols, line_no))
101      end
102    end
103  return output.flatten
104 end

106 def convert_line(line, symbols, line_no)
107   chunks = line.sub(",", " ").split#.partition(";")[0].split

108   case chunks[0]
109     when "LDI"
110       instruction = 0x21
111       x = register(chunks[1], 4)
112       v = immediate(chunks[2])
113       operands = (v << 2) + x
114
115     when "LD"
116       instruction = 0x01 + auto(chunks[2])
117       x = register(chunks[1], 8)
118       y = register(chunks[2], 3)
119       operands = (y << 5) + x
120
121     when "STI"
122       instruction = 0x25
123       y = register(chunks[1], 3)
124       v = immediate(chunks[2])
125       operands = (v << 2) + y
126
127     when "ST"
128       instruction = 0x05 + auto(chunks[1])

```

```

130     y = register(chunks[1], 3)
131     x = register(chunks[2], 8)
132     operands = (y << 5) + x
133
134 when "MV"
135     instruction = 0x04
136     if chunks[1][0] == 'r'[0] && chunks[2][0] == 'r'[0]
137         y = register(chunks[1], 8)
138         x = register(chunks[2], 8)
139         operands = (y << 5) + x
140     elsif chunks[1][0] == 'a'[0]
141         y = register(chunks[1], 3)
142         x = register(chunks[2], 8)
143         n = chunks[1][-1] == 'H' ? 1 : 0
144         operands = (1 << 9) + (n << 8) + (y << 5) + x
145     elsif chunks[2][0] == 'a'[0]
146         y = register(chunks[1], 8)
147         x = register(chunks[2], 3)
148         n = chunks[2][-1] == 'H' ? 1 : 0
149         operands = (1 << 4) + (n << 3) + (y << 5) + x
150     else
151         # explode
152     end
153
154 when "AND"
155     instruction = 0x02
156     operands = logical_operands(chunks)
157
158 when "OR"
159     instruction = 0x06
160     operands = logical_operands(chunks)
161
162 when "NOT"
163     instruction = 0x0A
164     operands = logical_operands(chunks)
165
166 when "XOR"
167     instruction = 0x0E
168     operands = logical_operands(chunks)
169
170 when "ADD"
171     instruction = 0x12
172     operands = logical_operands(chunks)
173
174 when "ADC"
175     instruction = 0x16
176     operands = logical_operands(chunks)
177
178 when "SUB"
179     instruction = 0x1A
180     operands = logical_operands(chunks)
181
182 when "SBB"
183     instruction = 0x1E
184     operands = logical_operands(chunks)
185
186 when "NEG"
187     instruction = 0x08
188     operands = logical_operands(chunks)
189
190 when "CMP"
191     instruction = 0x0C
192     operands = logical_operands(chunks)
193
194 when "BEQ"
195     instruction = 0x23
196     v = immediate(chunks[1], symbols, line_no)
197     operands = (v << 2)
198
199 when "BNE"
200     instruction = 0x27
201     v = immediate(chunks[1], symbols, line_no)
202     operands = (v << 2)

```

```

202
203   when "BLT"
204       instruction = 0x2B
205       v = immediate(chunks[1], symbols, line_no)
206       operands = (v << 2)
207
208   when "BGT"
209       instruction = 0x2F
210       v = immediate(chunks[1], symbols, line_no)
211       operands = (v << 2)
212
213   when "BC"
214       instruction = 0x33
215       v = immediate(chunks[1], symbols, line_no)
216       operands = (v << 2)
217
218   when "BNC"
219       instruction = 0x37
220       v = immediate(chunks[1], symbols, line_no)
221       operands = (v << 2)
222
223   when "R JMP"
224       instruction = 0x3B
225       v = immediate(chunks[1], symbols, line_no)
226       operands = (v << 2)
227
228   when "JMP"
229       instruction = 0x2F
230       y = register(chunks[1], 3)
231       operands = (y << 5)
232   end
233   opcode = (instruction << 10) + operands
234   return [(opcode >> 8), (opcode & 0xFF)]
235 end
236
237 if __FILE__ == $0
238     if !(1..2).include?(ARGV.length) || !File.exist?(ARGV[0])
239         p "Usage: ruby #{$0} <input_file> [<output_file>]"
240         exit
241     end
242
243     input = IO.readlines(ARGV[0])
244
245     output = convert(input)
246     if ARGV.length == 2:
247         File.open(ARGV[1], "wb") do |file|
248             output.each do |char|
249                 file.putc(char)
250             end
251         end
252     else
253         output.each do |char|
254             $stdout.putc(char)
255         end
256     end
257 end
258 end

```

---

### Listing 31: memory.rb

```

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

10 require 'rubygems'
11 require 'serialport'
12

```

```

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

20     case (header >> 7) & 0x01 # read/write bit
      when 0x01 # read
22         header = 0x82
         header += 0x01 if instruction
24         sp.putc(header)
         instruction ? sp.write(program[address]) :
26             sp.write(data_file[address])
         p "Sending data for " + instruction ? "Instruction" : "Data" + " bus,
           address: " + address + ", data: " + instruction ? program[address] :
           data_file[address]
28     when 0x00 # write, doesn't support writing to instruction memory
         data = sp.getc
30         data_file[address] = data
         p "Writing data, address: " + address + ", data: " + data_file[address]
32     end
      end
34 end

36 if __FILE__ == $0

38   if ARGV.size < 3
      STDERR.print "Usage: ruby #{__FILE__} <device> <baud_rate> <program_file>
        [<data_file>]\n"
40     exit
      end

42     device = ARGV[0]
44     baud_rate = ARGV[1].to_i

46     program = Array.new(2**12, 0x00)
      i = 0
48     File.open(ARGV[2], 'rb') do |input|
        input.each_byte do |byte|
50           program[i] = byte
           i += 1
52         end
      end

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

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

64     serve(program, data, sp)
      end

```

---

### Listing 32: test.S

```

1 LDI r0 10
  MV a0H r0
3 LDI r0 98
  MV a0L r0
5 LDI r0 8
  LDI r1 -9
7 ADD r0 r1
  ST a0 r0

```

---