

Listings

Main

Listing 1: main.vhd

```
1  -- Authors:
   --      Wim Looman, Forrest McKerchar, Henry Jenkins, Joel Koh, Sasha Wang, Tracy
       Jackson
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  library work;
8
9  entity main is
10     port (
11         clk      : in  std_logic;
12         reset    : in  std_logic;
13         tx       : out std_logic;
14         rx       : in  std_logic;
15         sw1      : in  std_logic;
16         sw2      : in  std_logic;
17     );
18     end main;
19
20     architecture main_arch of main is
21         component cpu IS
22             PORT(
23                 -- instruction bus
24                 inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
25                 inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
26                 inst_req  : out std_logic;                      -- Pulled low to request bus
27                                     usage.
28                 inst_ack  : in  std_logic;                      -- Pulled high to inform of
29                                     request completion.
30                 -- data bus
31                 data_add  : out std_logic_vector(15 downto 0); -- Address lines.
32                 data_line : inout std_logic_vector(7 downto 0); -- Data lines.
33                 data_read : out std_logic;                      -- High for a read request,
34                                     low for a write request.
35                 data_req  : out std_logic;                      -- Pulled low to request bus
36                                     usage.
37                 data_ack  : inout std_logic;                   -- Pulled high to inform of
38                                     request completion.
39                 -- extras
40                 clk       : in  std_logic;
41                 reset     : in  std_logic;
42             );
43         end component;
44         component mmu_main is
45             port (
46                 -- instruction bus
47                 inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
48                 inst_data : out std_logic_vector(15 downto 0); -- Data lines.
49                 inst_req  : in  std_logic;                      -- Pulled low to request bus
50                                     usage.
51                 inst_ack  : out std_logic;                      -- Pulled high to inform of
52                                     request completion.
```

```

47     -- data bus
data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
data_line : inout std_logic_vector(7  downto 0); -- Data lines.
49     data_read : in    std_logic;                -- High for a read request,
        low for a write request.
data_req  : in    std_logic;                -- Pulled low to request bus
        usage.
51     data_ack  : inout std_logic;                -- Pulled high to inform of
        request completion.
        -- extras
53     clk       : in    std_logic;
receive_pin : in    std_logic;
55     transfer_pin : out std_logic
    );
57 END component;
component IO is
59     PORT(
        -- data bus --
61     data_add  : IN    std_logic_vector(15 DOWNT0 0); -- address lines --
data_data  : INOUT  std_logic_vector(7  DOWNT0 0); -- data lines --
63     data_read : INOUT  std_logic;                -- pulled high for
        read, low for write --
data_req  : INOUT  std_logic;                -- pulled low to
        request bus usage --
65     data_ack  : INOUT  std_logic;                -- pulled high to
        inform request completion --
        -- io --
67     clk       : IN    std_logic;
sw1       : IN    std_logic;
69     sw2       : IN    std_logic);
--leds    : OUT std_logic_vector(7 DOWNT0 0);
71 END component;
-- instruction bus
73 signal inst_add  : std_logic_vector(11 downto 0); -- Address lines.
signal inst_data  : std_logic_vector(15 downto 0); -- Data lines.
75 signal inst_req  : std_logic;                -- Pulled low to request bus
        usage.
signal inst_ack  : std_logic;                -- Pulled high to inform of
        request completion.
77 -- data bus
signal data_add  : std_logic_vector(15 downto 0); -- Address lines.
79 signal data_line : std_logic_vector(7  downto 0); -- Data lines.
signal data_read : std_logic;                -- High for a read request, low
        for a write request.
81 signal data_req  : std_logic;                -- Pulled low to request bus
        usage.
signal data_ack  : std_logic;                -- Pulled high to inform of
        request completion.
83
begin
85     c : cpu port map(
        -- instruction bus
87     inst_add => inst_add, -- Instruction address
inst_data => inst_data, -- Instruction data
89     inst_req => inst_req, -- Request
inst_ack => inst_ack, -- Instruction obtained
91     -- data bus
data_add => data_add, -- Data address
93     data_line => data_line, -- Data
data_read => data_read, -- 1 for read, 0 for write
95     data_req => data_req, -- Request
data_ack => data_ack, -- Data written to/ read from
97     -- extras
clk      => clk,
99     reset  => reset
    );
101    m : mmu_main port map(
        -- instruction bus
103    inst_add  => inst_add, -- Address lines.
inst_data  => inst_data, -- Data lines.
105    inst_req  => inst_req, -- Pulled low to request bus usage.
inst_ack  => inst_ack, -- Pulled high to inform of request completion.
107    -- data bus

```

```

    data_add      => data_add,  -- Address lines.
109    data_line    => data_line, -- Data lines.
    data_read     => data_read, -- High for a read request, low for a write request.
111    data_req     => data_req,  -- Pulled low to request bus usage.
    data_ack      => data_ack,  -- Pulled high to inform of request completion.
113    -- extras
    clk           => clk,
115    receive_pin  => rx,
    transfer_pin  => tx
117 );
i : io  port map(
119     clk           => clk,
    data_add      => data_add,
121     data_data     => data_line,
    data_read     => data_read,
123     data_req      => data_req,
    data_ack      => data_ack,
125     -- io --
    sw1           => sw1,
127     sw2           => sw2
);
129 end architecture main_arch;

```

IO

Listing 2: IO/debounce.vhd

```
1  -----
2  -- Module Name:      debounce
3  -- Description: Entity to debounce a mechanical switch/button
4  -- Authors: Tracy Jackson
5  --                Sasha Wang
6  --
7  -----
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11 use IEEE.STD_LOGIC_ARITH.ALL;
12 use IEEE.STD_LOGIC_UNSIGNED."+";
13
14 library work;
15
16 ENTITY debounce IS
17     PORT(clk : IN STD_LOGIC;
18           switch : IN STD_LOGIC;
19           switch_state : OUT STD_LOGIC);
20 END debounce;
21
22 ARCHITECTURE debounced_switch OF debounce IS
23     SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);
24 BEGIN
25     -- Debounce the switch using a counter
26     PROCESS(clk, switch)
27     BEGIN
28         IF switch = '0' THEN
29             count <= "000";
30         ELSIF rising_edge(clk) THEN
31             IF count /= "111" THEN
32                 count <= count + 1;
33             END IF;
34         END IF;
35         IF count = "111" AND switch = '1' THEN
36             switch_state <= '1';
37         ELSE
38             switch_state <= '0';
39         END IF;
40     END PROCESS;
41 END debounced_switch;
```

Listing 3: IO/IO.vhd

```
1  -----
2  -- Module Name:      IO
3  -- Description: Entity to handle IO
4  -- Authors: Tracy Jackson
5  --                Sasha Wang
6  --
7  -----
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11 use IEEE.STD_LOGIC_ARITH.ALL;
12 --use IEEE.STD_LOGIC_UNSIGNED.ALL;
13
14 library work;
15 use work.debounce;
16 use work.switch_reg;
17 use work.led_io;
18
19
20 ---- Uncomment the following library declaration if instantiating
21 ---- any Xilinx primitives in this code.
22 --library UNISIM;
23 --use UNISIM.VComponents.all;
```

```

24 entity IO is
25     PORT(
26         -- data bus --
27         data_add      : IN          std_logic_vector(15 DOWNTO 0);
28         -- address lines --
29         data_data      : INOUT       std_logic_vector(7  DOWNTO 0);  -- data
30         lines --
31         data_read      : INOUT       std_logic;
32         -- pulled high for read, low for write --
33         data_req        : INOUT       std_logic;
34         -- pulled low to request bus usage --
35         data_ack        : INOUT       std_logic;
36         -- pulled high to inform request completion --
37         -- io --
38         clk             : IN          std_logic;
39         sw1             : IN          std_logic;
40         sw2             : IN          std_logic);
41         --leds          : OUT std_logic_vector(7  DOWNTO 0);
42 end IO;
43
44 architecture io of IO is
45
46     COMPONENT led_io
47     PORT(
48         data_add      : IN          std_logic_vector(15 DOWNTO 0);  --
49         address lines --
50         data_data      : INOUT       std_logic_vector(7  DOWNTO 0);  -- data lines
51         --
52         data_read      : INOUT       std_logic;                      --
53         pulled high for read, low for write --
54         data_req        : INOUT       std_logic;                      --
55         pulled low to request bus usage --
56         data_ack        : INOUT       std_logic;                      --
57         pulled high to inform request completion --
58         clock          : IN          std_logic
59     );
60 END COMPONENT;
61
62 COMPONENT switch_io IS
63     PORT ( data_add      : IN          std_logic_vector(15 DOWNTO 0);
64           data_data      : INOUT       std_logic_vector(7  DOWNTO 0);
65           data_read      : INOUT       std_logic;
66           data_req        : INOUT       std_logic;
67           data_ack        : INOUT       std_logic;
68           clk             : IN          std_logic;
69           sw1             : IN          std_logic;
70           sw2             : IN          std_logic
71     );
72 END COMPONENT;
73
74 BEGIN
75     led: led_io PORT MAP(data_add, data_data, data_read, data_req, data_ack, clk);
76     switch: switch_io PORT MAP(data_add, data_data, data_read, data_req, data_ack,
77                               clk,sw1,sw2);
78
79     -----
80 END io;

```

Listing 4: IO/leds.vhd

```

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

```

```

6  --
  -----
8  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_ARITH.ALL;

12 library work;

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

26 ARCHITECTURE led_arch OF led_io IS
   Signal led_enable    : std_logic;
28   Signal led_state    : std_logic_vector(7 DOWNTO 0);
   BEGIN
30
   -- Determine if it is the LEDs being accessed
32   PROCESS(clock, data_req, data_add, data_read)
   BEGIN
34       IF data_req = '0' AND data_add = "0000000000001110" AND data_read = '0'
           THEN
36               led_enable <= '1';
           ELSE
38               led_enable <= '0';
           END IF;
40   END PROCESS;

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

52

54 END led_arch;

```

Listing 5: IO/switch_register.vhd

```

1  -----
   -- Module Name:      switch_reg
3  -- Description: Entity to store switch state (can be extended to more than one)
   -- Authors: Tracy Jackson
5  --          Sasha Wang
   --
7  -----
   library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_ARITH.ALL;
11 use IEEE.STD_LOGIC_UNSIGNED.ALL;

13 library work;

```

```

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

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

```

Listing 6: IO/switches.vhd

```

1  -----
  -- Module Name:      switch_io
3  -- Description: Entity to control input from switches
  -- Authors: Tracy Jackson
  --               Sasha Wang
  --
7  -----
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.STD_LOGIC_ARITH.ALL;

11   library work;
13   use work.debounce;
15   use work.switch_reg;

    entity switch_io is
17       PORT(
19           -- data bus --
           data_add      : IN          std_logic_vector(15 DOWNT0 0);
           -- address lines --
           data_data      : INOUT       std_logic_vector(7 DOWNT0 0);  -- data
           lines --
21           data_read     : INOUT       std_logic;
           -- pulled high for read, low for write --
           data_req       : INOUT       std_logic;
           -- pulled low to request bus usage --
23           data_ack      : INOUT       std_logic;
           -- pulled high to inform request completion --
           -- io --
25           clk           : IN          std_logic;
           sw1            : IN          std_logic;
27           sw2            : IN          std_logic);
    end switch_io;

29   architecture Behavioral of switch_io is
31
33   signal enable1          : std_logic;
35   signal switch1_connection : std_logic;
37   signal enable2          : std_logic;
39   signal switch2_connection : std_logic;
41   signal switch2_output    : std_logic;

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

47   COMPONENT switch_reg

```

```

        PORT( D           : IN STD_LOGIC;
              clk, enable : IN STD_LOGIC;
              Q           : OUT STD_LOGIC);
49
51 END COMPONENT;
53
55 BEGIN
57 sw1_debouncer: debounce PORT MAP(clk, sw1, switch1_connection);
    sw1_status: switch_reg PORT MAP(switch1_connection,clk, enable1, switch1_output);
59
61 sw2_debouncer: debounce PORT MAP(clk, sw2,switch2_connection);
    sw2_status: switch_reg PORT MAP(switch2_connection,clk, enable2, switch2_output);
63
65 PROCESS(clk,switch1_output,switch2_output, data_ack)
    BEGIN
67 IF rising_edge(clk) THEN
        IF switch1_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
            1, disable switch_reg from getting any more info
69             enable1 <= '0';

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

77     IF switch2_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
        1, disable switch_reg from getting any more info
        enable2 <= '0';
79
        --ELSIF data_ack = '0' AND data_add = "0000000000001100" THEN -- when the data is
            sent to the CPU, enable the switch_reg again
81     -- enable2 <= '1';
    ELSE
83         enable2 <= '1';
    END IF;
85

87 END IF;
    END PROCESS;
89

91 PROCESS(clk, data_add, data_read)
    BEGIN
93     IF rising_edge(clk) THEN
        IF data_req = '0' AND data_read = '1' THEN
95             IF data_add = "0000000000001110" THEN -- switch1 address
                IF switch1_output = '1' THEN
97                     data_data <= "00000001";
                ELSE
99                     data_data <= "00000000";
                END IF;
101             data_ack <= '0';
        END IF;
103         IF data_add = "0000000000001100" THEN -- switch2 address
            IF switch2_output = '1' THEN
105                 data_data <= "00000001";
            ELSE
107                 data_data <= "00000000";
            END IF;
109             data_ack <= '0';
        END IF;
111     ELSIF data_req = '1' AND data_ack = '0' THEN
        data_ack <= 'Z';
113     END IF;
    END IF;
115 END PROCESS;

```


117

119

END Behavioral;

CPU

Listing 7: processor/alu.vhd

```
-----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18:59:20 09/18/2010
6  -- Design Name:
7  -- Module Name: alu - alu_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 --
-----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
23 use ieee.std_logic_arith.all;
24 --use ieee.std_logic_unsigned.all;
25
26
27 library work;
28 use work.fulladder8;
29 --use work.cpu.ALL;
30
31 -- Uncomment the following library declaration if using
32 -- arithmetic functions with Signed or Unsigned values
33 --use IEEE.NUMERIC_STD.ALL;
34
35 -- Uncomment the following library declaration if instantiating
36 -- any Xilinx primitives in this code.
37 --library UNISIM;
38 --use UNISIM.VComponents.all;
39
40 entity alu is
41     Port (f : in STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
42           rx : in STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
43           ry : in STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
44           ro : out STD_LOGIC_VECTOR (7 downto 0); -- Output Normaly (Ry)
45           Cin : in STD_LOGIC; -- Carry in
46           sr : out STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
47               C(1), N(2)
48 end alu;
49
50 architecture alu_arch of alu is
51     component fulladder8 IS
52     Port (A : in STD_LOGIC_VECTOR( 7 downto 0);
53           B : in STD_LOGIC_VECTOR( 7 downto 0);
54           Cin : in STD_LOGIC;
55           Sum : out STD_LOGIC_VECTOR( 7 downto 0);
56           Cout : out STD_LOGIC
57     );
58 end component;
59
60 signal A : std_logic_vector(7 downto 0);
61 signal B : std_logic_vector(7 downto 0);
62 signal AdderCin : std_logic;
63 signal Sum : std_logic_vector(7 downto 0);
64 signal AdderCout : std_logic;
65 signal Z,C,N : std_logic; -- Make the code easier to read
66 signal output : std_logic_vector(7 downto 0); -- used to allow reading of ro
67 BEGIN
68     Adder: fulladder8 port map(A, B, AdderCin, Sum, AdderCout);
```

```

68  process(f, rx, ry, Cin, Sum, AdderCout)
    --signal Z,C,N : std_logic; -- Make the code easier to read
70  BEGIN
    -- use case statement to achieve
    -- 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;
96      ELSIF f = "1101" THEN -- Do SUB operation
        AdderCin <= '1';
        A <= ry;
        B <= (not rx);
        output <= Sum;
102     ELSIF f = "1111" THEN -- Do SBB operation
        AdderCin <= (not Cin);
        A <= ry;
        B <= (not rx);
        output <= Sum;
108     ELSIF f = "0100" THEN -- Do NEG operation ( two's complement )
        AdderCin <= '1';
        A <= (others => '0');
        B <= (not rx);
        output <= Sum;
        C <= AdderCout;
        N <= output(7);
114     ELSIF f = "0110" THEN -- Do CMP operation
        AdderCin <= '1';
        A <= rx;
        B <= (not ry);
        output <= Sum;
        C <= AdderCout;
        N <= output(7);
122     ELSE
        AdderCin <= '0';
        A <= (others => '0');
        B <= (others => '0');
        output <= (others => '0');
        C <= '0';
        N <= '0';
128     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))

```

```

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

```

Listing 8: processor/ar.vhd

```

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

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

```

```

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

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

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

```

Listing 9: processor/cpu.vhd

```

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

```

```

-- Revision 0.01 - File Created
17 -- Additional Comments:
--
19 -----
library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
23 --use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

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

39 entity cpu is
41   PORT(
-- instruction bus
43     inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
inst_data  : in  std_logic_vector(15 downto 0); -- Data lines.
45     inst_req  : out std_logic; -- Pulled low to request bus
usage.
inst_ack   : in  std_logic; -- Pulled high to inform of
request completion.
47     -- data bus
data_add   : out std_logic_vector(15 downto 0); -- Address lines.
49     data_line : inout std_logic_vector(7 downto 0); -- Data lines.
data_read  : out std_logic; -- High for a read request,
low for a write request.
51     data_req  : out std_logic; -- Pulled low to request bus
usage.
data_ack   : inout std_logic; -- Pulled high to inform of
request completion.
53     -- extras
clk        : in  std_logic;
55     reset    : in  std_logic
);
57 end cpu;
59
architecture cpu_arch of cpu is
61   component alu IS
Port (f : in STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
63     rx : in STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
ry : in STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
65     ro : out STD_LOGIC_VECTOR (7 downto 0); -- Output Normaly (Ry)
Cin : in STD_LOGIC; -- Carry in
67     sr : out STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
C(1), N(2)
END component;
69   component ar is
Port (clk : in STD_LOGIC;
71     enable : in STD_LOGIC;
Sel8Bit : in STD_LOGIC;
73     SelHighByte : in STD_LOGIC;
ByteInput : in STD_LOGIC_VECTOR (7 downto 0);
75     SelRi : in STD_LOGIC_VECTOR (1 downto 0); -- Select the address
register
SelRo : in STD_LOGIC_VECTOR (1 downto 0); -- Select the address
register
77     Ri : in STD_LOGIC_VECTOR (15 downto 0); -- The input
Ro : out STD_LOGIC_VECTOR (15 downto 0)); -- The output
79   END component;
component cu IS

```

```

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

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

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

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

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

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

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

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

```

```

143         Ry          : out  STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
144     END component;
145     component sr is
146         Port (clk      : in  STD_LOGIC;
147              enable    : in  STD_LOGIC;
148              reset     : in  STD_LOGIC;
149              Ri        : in  STD_LOGIC_VECTOR (15 downto 0); -- The input to the SR
150              Ro        : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
151     END component;
152     component pc is
153         Port (clk      : in  STD_LOGIC;
154              enable    : in  STD_LOGIC;
155              reset     : in  STD_LOGIC;
156              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
158     END component;
159     signal alu_Cin      : std_logic;
160     signal alu_f        : std_logic_vector(3 downto 0);
161     signal alu_rx       : std_logic_vector(7 downto 0);
162     signal alu_ry       : std_logic_vector(7 downto 0);
163
164     signal sr_reset     : std_logic;
165     signal sr_enable    : std_logic;
166     signal sr_Ro        : std_logic_vector(15 downto 0);
167     signal sr_input     : std_logic_vector(15 downto 0);
168
169     signal ar_enable    : STD_LOGIC; -- enable write to AR
170     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
172     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
174     signal ar_sel8Bit    : STD_LOGIC; -- only write half the AR
175     signal ar_selHByte  : STD_LOGIC; -- high or low half of the AR
176         to write
177     signal ar_ByteIn    : STD_LOGIC_VECTOR (7 downto 0); -- 8 bit input to write half
178         of AR
179
180     signal pc_reset     : std_logic;
181     signal pc_enable    : std_logic;
182     signal pc_Ri        : std_logic_vector(15 downto 0);
183     signal pc_Ro        : std_logic_vector(15 downto 0);
184
185     signal gpr_InSel    : std_logic;
186     signal gpr_enable    : std_logic;
187     signal gpr_SelRx    : std_logic_vector(2 downto 0);
188     signal gpr_SelRy    : std_logic_vector(2 downto 0);
189     signal gpr_SelRi    : std_logic_vector(2 downto 0);
190     signal gpr_RiCU     : std_logic_vector(7 downto 0);
191     signal gpr_RiCDB    : std_logic_vector(7 downto 0);
192
193     begin
194         a: alu port map(
195             f      => alu_f,
196             rx     => alu_rx,
197             ry     => alu_ry,
198             ro     => gpr_RiCDB,
199             Cin    => alu_Cin,
200             sr     => sr_input
201         );
202         c: cu port map(
203             reset   => reset, -- '0' for reset
204             clock   => clk, -- clock
205
206             alu_f    => alu_f, -- Function
207             alu_Cin  => alu_Cin, -- Carry into the ALU
208
209             -- General Purpose Registers
210             gpr_InSel => gpr_InSel, -- select the input path (0 - cu, 1 - ALU)
211             gpr_en    => gpr_enable, -- enable write to GPR
212             gpr_SelRx => gpr_SelRx, -- select GPR output x
213             gpr_SelRy => gpr_SelRy, -- select GPR output y
214             gpr_SelRi => gpr_SelRi, -- select GPR input

```



```

215     gpr_Ri    => gpr_RiCU,-- input to GPR
        gpr_Rx    => alu_rx,-- Rx from GPR
        --gpr_Ry    => alu_ry,-- Ry from GPR
217
        -- Status Register
219     sr_en      => sr_enable,-- enable write to SR
        sr_reset  => sr_reset,-- reset SR
221     sr_Ro      => sr_Ro,-- output from SR
        -- control unit doesnt write to SR, the ALU does
223
        -- Program Counter
225     pc_en      => pc_enable,-- enable write to PC
        pc_reset  => pc_reset,-- reset PC
227     pc_Ri      => pc_Ri,-- input to PC
        pc_Ro      => pc_Ro,-- output from PC
229
        -- Address Registers
231     ar_en      => ar_enable,    -- enable write to AR
        ar_SelRi  => ar_SelRi,    -- select AR in
233     ar_SelRo   => ar_SelRo,    -- select AR out
        ar_sel8Bit => ar_sel8Bit,
235     ar_selHByte => ar_selHByte,
        ar_ByteIn  => ar_ByteIn,
237     ar_Ri      => ar_Ri,        -- input to AR
        ar_Ro      => ar_Ro,        -- output from AR
239
        -- Instruction memory
241     inst_add    => inst_add,-- Instruction address
        inst_data  => inst_data,-- Instruction data
243     inst_req    => inst_req,-- Request
        inst_ack   => inst_ack,-- Instruction obtained
245
        data_add   => data_add,-- Data address
247     data_data   => data_line,-- Data
        data_read  => data_read,-- 1 for read, 0 for write
249     data_req    => data_req,-- Request
        data_ack   => data_ack,-- Data written to/ read from
251
    );
    address : ar port map(
253         clk      => clk,
        enable     => ar_enable,
255         Sel8Bit   => ar_Sel8Bit,
        SelHighByte => ar_selHByte,
257         ByteInput => ar_ByteIn,
        SelRi      => ar_SelRi,
259         SelRo     => ar_SelRo,
        Ri         => ar_Ri,
261         Ro        => ar_Ro
    );
263 g : gpr port map(
        clk      => clk,
265     enable     => gpr_enable,
        SelRx    => gpr_SelRx,
267     SelRy    => gpr_SelRy,
        SelRi    => gpr_SelRi,
269     SelIn    => gpr_InSel,
        RiCU     => gpr_RiCU,
271     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 10: processor/cu.vhd

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18:59:20 09/18/2010
6  -- Design Name:
7  -- Module Name: cu - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description: The control unit
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.NUMERIC_STD.ALL;
23 use ieee.std_logic_arith.all;
24 --use ieee.std_logic_unsigned.all;
25
26
27 library work;
28 --use work.fulladder;
29 --use work.cpu.ALL;
30
31 -- Uncomment the following library declaration if using
32 -- arithmetic functions with Signed or Unsigned values
33 --use IEEE.NUMERIC_STD.ALL;
34
35 -- Uncomment the following library declaration if instantiating
36 -- any Xilinx primitives in this code.
37 --library UNISIM;
38 --use UNISIM.VComponents.all;
39
40 entity cu is
41     Port (reset      : in STD_LOGIC;          -- '0' for reset
42           clock       : in STD_LOGIC;          -- clock
43
44           alu_f        : out STD_LOGIC_VECTOR (3 downto 0); -- Function
45           alu_Cin      : out STD_LOGIC;          -- Carry in to ALU
46
47           -- General Purpose Registers
48           gpr_InSel    : out STD_LOGIC;          -- select the input path (0
49               -- cu, 1 - ALU)
49           gpr_en       : out STD_LOGIC;          -- enable write to GPR
50           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
52           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
54           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 ,
56               not used
57
58           -- Status Register
59           sr_en        : out STD_LOGIC;          -- enable write to SR
60           sr_reset     : out STD_LOGIC;          -- reset SR
61           sr_Ro        : in STD_LOGIC_VECTOR (15 downto 0); -- output from SR
62           -- control unit doesnt write to SR, the ALU does
63
64           -- Program Counter
65           pc_en        : out STD_LOGIC;          -- enable write to PC

```

```

65     pc_reset      : out STD_LOGIC;           -- reset PC
66     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

68
69     -- Address Registers
70     ar_en         : out STD_LOGIC;           -- enable write to AR
71     ar_SelRi      : out STD_LOGIC_VECTOR (1 downto 0); -- select AR in
72     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
74     ar_Ro         : in  STD_LOGIC_VECTOR (15 downto 0); -- output from AR
75     ar_sel8Bit    : out STD_LOGIC;           -- only write half the AR
76     ar_selHByte   : out STD_LOGIC;           -- high or low half of the
77     ar_ByteIn     : out STD_LOGIC_VECTOR (7 downto 0); -- 8 bit input to write
78     -- half of AR

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

84
85     data_add      : out STD_LOGIC_VECTOR (15 downto 0); -- Data address
86     data_data     : inout STD_LOGIC_VECTOR (7 downto 0); -- Data
87     data_read     : out STD_LOGIC;           -- 1 for read, 0 for write
88     data_req      : out STD_LOGIC;           -- Request
89     data_ack      : in  STD_LOGIC;           -- Data written to/ read
90     from

91 );
92 end cu;
93
94
95 architecture Behavioral of cu is
96     component fulladder16 IS
97     Port (A      : in  STD_LOGIC_VECTOR(15 downto 0);
98           B      : in  STD_LOGIC_VECTOR(15 downto 0);
99           Cin    : in  STD_LOGIC;
100          Sum    : out STD_LOGIC_VECTOR(15 downto 0);
101          Cout   : out STD_LOGIC
102     );
103 end component;

104
105 type states is (reset_state, fetch, decode, execute);
106 signal state      : states := reset_state;
107 signal next_state : states := reset_state;

108
109 signal opcode      : std_logic_vector(15 downto 0); -- unprocessed instruction

110
111 -- Decoded data
112 signal rx : std_logic_vector(2 downto 0);
113 signal ry : std_logic_vector(2 downto 0);
114 signal ay : std_logic_vector(1 downto 0);

115
116 -- Indicates what needs to be executed
117 signal write_gpr : std_logic;
118 signal write_sr  : std_logic;
119 signal write_pc  : std_logic;
120 signal write_ar  : std_logic;
121 signal write_memory : std_logic;

122
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
130 signal v : STD_LOGIC_VECTOR(7 downto 0); -- 8-bit immediate

131
132 BEGIN
133     Adder16: fulladder16 port map(A16, B16, AdderCin16, Sum16, AdderCout16);

```

```

135
136 -- Process instruction
137 -- Assumes all instructions are valid
process(clock, state, opcode, gpr_Rx, sr_Ro, pc_Ro, ar_Ro, inst_data, inst_ack,
      data_data, data_ack,
138 rx, ry, ay, v, write_gpr, write_sr, write_pc, write_ar, write_memory)
139 BEGIN
140   if rising_edge(clock) then
141     case state is
142       when reset_state =>
143         sr_reset <= '0';
144         pc_reset <= '0';
145         next_state <= fetch;
146
147       when fetch =>
148         sr_reset <= '1';
149         pc_reset <= '1';
150
151         gpr_en <= '0';
152         sr_en <= '0';
153         pc_en <= '0';
154         ar_en <= '0';
155
156         write_gpr <= '0';
157         write_sr <= '0';
158         write_pc <= '0';
159         write_ar <= '0';
160         write_memory <= '0';
161
162         inst_add <= pc_Ro(11 downto 0);
163         if inst_ack = '0' then
164           inst_req <= '1';
165         else
166           opcode <= inst_data;
167           inst_req <= '0';
168
169           -- increment program counter
170           AdderCin16 <= '1';
171           A16 <= PC_Ro;
172           B16 <= "0000000000000000";
173           pc_Ri <= Sum16;
174           pc_en <= '1';
175
176           next_state <= decode;
177         end if;
178       when decode =>
179         pc_en <= '0';
180
181         -- ALU
182         if opcode(15) = '0' and opcode(10) = '0' and not opcode(14 downto 11) =
183           "0010" then
184
185           ry <= opcode(7 downto 5);
186           rx <= opcode(2 downto 0);
187
188           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
194
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
200           write_sr <= '1';
201           next_state <= execute;
202
203           -- Branching
204           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
236             when others =>
237                 v <= "00000000";
238         end case;
239
240         -- PC <- PC + v
241         AdderCin16 <= '0';
242         A16 <= PC_Ro;
243         B16 <= "00000000" & v;
244         pc_Ri <= Sum16;
245
246     elsif opcode(15 downto 12) = "0111" then -- JMP
247         ay <= opcode(6 downto 5);
248
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;
256
257     write_pc <= '1';
258     next_state <= execute;
259
260 -- Addressing
261 else
262     gpr_Insel <= '0';
263
264     case opcode(12 downto 10) is
265
266         when "001" => -- Load
267             if opcode(15) = '1' then -- immediate
268                 rx <= '0' & opcode(1 downto 0);
269                 v <= opcode(9 downto 2);
270
271                 -- rx <- v
272                 gpr_SelRi <= rx;
273                 gpr_Ri <= v;
274                 write_gpr <= '1';
275                 next_state <= execute;
276             else -- direct

```

```

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

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

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

317     when "101" => -- Store
        if opcode(15) = '1' then -- immediate
319             ay <= opcode(1 downto 0);
            v <= opcode(9 downto 2);
321
            -- [ay] <- v
            ar_selRo <= ay;
            data_add <= ar_Ro;
323             data_read <= '0';
            data_data <= v;
325             write_memory <= '1';
            next_state <= execute;
327
        else -- direct
329             rx <= opcode(2 downto 0);
            ay <= opcode(6 downto 5);
331
            -- [ay] <- rx
            gpr_selRx <= rx;
            ar_selRo <= ay;
333             data_add <= ar_Ro;
            data_read <= '0';
335             data_data <= gpr_Rx;
            write_memory <= '1';
337
339
341         case opcode(14 downto 13) is
            when "01" => -- auto increment
343                 AdderCin16 <= '1';
                 A16 <= ar_Ro;
345                 B16 <= "0000000000000000";
                 ar_selRi <= ay;
347                 ar_sel8bit <= '0';
                 ar_Ri <= Sum16;
349                 write_ar <= '1';
            when "10" => -- auto decrement

```

```

351         AdderCin16 <= '0';
352         A16 <= ar_Ro;
353         B16 <= "1111111111111111";
354         ar_selRi <= ay;
355         ar_sel8bit <= '0';
356         ar_Ri <= Sum16;
357         write_ar <= '1';
358         when others =>
359             -- do nothing
360         end case;
361         next_state <= execute;
362     end if;
363
364     when "100" => -- Move
365         if opcode(9) = '1' then
366             rx <= opcode(2 downto 0);
367             ay <= opcode(6 downto 5);
368
369             -- ayn <- rx
370             gpr_selRx <= rx;
371             ar_selRi <= ay;
372             ar_sel8bit <= '1';
373             ar_ByteIn <= gpr_Rx;
374
375             if opcode(8) = '1' then -- high
376                 ar_selHByte <= '1';
377             else -- low
378                 ar_selHByte <= '0';
379             end if;
380
381             write_ar <= '1';
382             next_state <= execute;
383
384         elsif opcode(4) = '1' then
385             rx <= opcode(7 downto 5);
386             ay <= opcode(1 downto 0);
387
388             -- rx <- ayn
389             gpr_selRi <= rx;
390             ar_selRo <= ay;
391
392             if opcode(3) = '1' then -- high
393                 gpr_Ri <= ar_Ro(15 downto 8);
394             else -- low
395                 gpr_Ri <= ar_Ro(7 downto 0);
396             end if;
397
398             write_gpr <= '1';
399             next_state <= execute;
400
401         else
402             rx <= opcode(2 downto 0);
403             ry <= opcode(7 downto 5);
404
405             -- ry <- rx
406             gpr_selRx <= rx;
407             gpr_selRi <= ry;
408             gpr_Ri <= gpr_Rx;
409
410             write_gpr <= '1';
411             next_state <= execute;
412
413         end if;
414
415         when others =>
416             -- should not reach here
417
418         end case;
419
420     end if;
421
422     when execute =>

```

```

425         if write_memory = '1' then
426             if data_ack = '0' then -- request write
427                 data_req <= '1';
428             else -- data written
429                 data_req <= '0';
430             gpr_en <= write_gpr;
431             sr_en <= write_sr;
432             pc_en <= write_pc;
433             ar_en <= write_ar;
434             next_state <= fetch;
435             end if;
436         else
437             gpr_en <= write_gpr;
438             sr_en <= write_sr;
439             pc_en <= write_pc;
440             ar_en <= write_ar;
441             next_state <= fetch;
442             end if;
443
444         when others =>
445             -- shouldn't reach here
446             next_state <= reset_state;
447         end case;
448     end if;
449 end process;
450
451 process(clock, reset, next_state)
452 BEGIN
453     if reset = '0' then
454         state <= reset_state;
455     elsif rising_edge(clock) then
456         state <= next_state;
457     end if;
458 end process;
459 end Behavioral;

```

Listing 11: processor/fulladder.vhd

```

-----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18:59:20 09/18/2010
6  -- Design Name:
7  -- Module Name: fulladder - Behavioral
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 --
-----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity fulladder is
24     Port (Ax : in STD_LOGIC;
25           Bx : in STD_LOGIC;
26           Ci : in STD_LOGIC;
27           Sx : out STD_LOGIC;
28           Co : out STD_LOGIC
29           );
30 end fulladder;
31
32
33 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
    Port (Ax      : in   STD_LOGIC;
98           Bx      : in   STD_LOGIC;
           Ci      : in   STD_LOGIC;
100          Sx      : out  STD_LOGIC;
           Co      : out  STD_LOGIC
102          );
        end component;
    signal Carry    : std_logic_vector(16 downto 0);
104 BEGIN
106    Carry(0) <= Cin;

```

```

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

```

Listing 12: processor/gpr.vhd

```

-----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18:59:20 09/18/2010
6  -- Design Name:
7  -- Module Name: GPR - gpr_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 --
-----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 library work;
24 use work.reg8;
25
26 entity gpr is
27     Port (clk      : in  STD_LOGIC;
28           enable   : in  STD_LOGIC;
29           SelRx    : in  STD_LOGIC_VECTOR (2 downto 0); -- The Rx output selection
30           value    :
31           SelRy    : in  STD_LOGIC_VECTOR (2 downto 0); -- The Ry output selection
32           value    :
33           SelRi    : in  STD_LOGIC_VECTOR (2 downto 0); -- The Ri input selection
34           value    :
35           SelIn    : in  STD_LOGIC; -- Select where the input should be from the CU
36           or CDB
37           RiCU     : in  STD_LOGIC_VECTOR (7 downto 0); -- Input from the Control
38           Unit
39           RiCDB    : in  STD_LOGIC_VECTOR (7 downto 0); -- Input from the Common Data
40           Bus
41           Rx       : out STD_LOGIC_VECTOR (7 downto 0); -- The Rx output
42           Ry       : out STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
43 end gpr;
44
45 architecture gpr_arch of gpr is
46     component reg8 IS
47         port(I      : in  std_logic_vector(7 downto 0);
48              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;

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

86 -- Set Ri the input
87 SetInput: process(clk, enable, SelRi)
88 BEGIN
89     R0E <= '0';
90     R1E <= '0';
91     R2E <= '0';
92     R3E <= '0';
93     R4E <= '0';
94     R5E <= '0';
95     R6E <= '0';
96     R7E <= '0';
97     IF enable = '1' THEN
98         case SelRi IS
99             WHEN "000" =>
100                 R0E <= '1';
101             WHEN "001" =>
102                 R1E <= '1';
103             WHEN "010" =>
104                 R2E <= '1';
105             WHEN "011" =>
106                 R3E <= '1';
107             WHEN "100" =>
108                 R4E <= '1';
109             WHEN "101" =>
110                 R5E <= '1';
111             WHEN "110" =>
112                 R6E <= '1';
113             WHEN "111" =>
114                 R7E <= '1';

```

```

        WHEN others =>
118         NULL; -- None of them are enabled
        end case;
120     END IF;
    end process;

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

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

```

Listing 13: processor/reg.vhd

```

-----
2  -- Company:
   -- Engineer:
4  --
   -- Create Date:      20:08:41 10/11/2010
6  -- Design Name:
   -- Module Name:      register - Behavioral
8  -- Project Name:
   -- Target Devices:
10 -- Tool versions:
   -- Description:
12 --
   -- Dependencies:
14 --
   -- Revision:
16 -- Revision 0.01 - File Created
   -- Additional Comments:
18 --
-----

20 library ieee;
   use ieee.std_logic_1164.all;
22
   entity reg8 is
24     port(I          : in  std_logic_vector(7 downto 0);
           clock      : in  std_logic;
           enable     : in  std_logic;
           reset      : in  STD_LOGIC;
28     Q          : out std_logic_vector(7 downto 0)
           );
30 end reg8;

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

```

```

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

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

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

76     end process;

78 end behv;

```

Listing 14: processor/spr.vhd

```

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

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

```

```

32 end sr;

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

46
-----

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

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

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

```

MMU

Listing 15: mmu/control_unit.vhd

```
1  -- Authors:
2  --      Wim Looman, Forrest McKerchar
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  library work;
8  use work.mmu_types.all;
9
10 entity mmu_control_unit is
11     port (
12         eoc          : in  std_logic; -- High on muart has finished collecting data
13         eot          : in  std_logic; -- High on muart has finished transmitting
14         ready        : in  std_logic; -- High if the muart is ready for new transfer
15         data_read     : in  std_logic; -- High if the cpu requests a read, else write
16         data_req      : in  std_logic; -- Low to start a transfer
17         data_add_0    : in  std_logic; -- High for memory address, else IO
18         inst_req      : in  std_logic; -- Low to start a transfer
19         fr           : in  std_logic; -- Input headers fetch request bit
20         inst_or_data_in : in  std_logic; -- Input headers inst or data bit
21         rw           : in  std_logic; -- Input headers read/write bit
22         write         : out std_logic; -- Pulled high to start muart writing data
23         inst_or_data_out : out std_logic; -- Output headers inst or data bit
24         inst_ack      : out std_logic; -- Idles high, pulled low when data ready
25         data_ack      : inout std_logic; -- Idles 'Z', high when data not ready,
26             pulled low when data ready
27         muart_input   : out muart_input_state; -- Signal connected to muart input
28         muart_output  : out muart_output_state; -- Signal connected to muart output
29         clk           : in  std_logic
30     );
31 end mmu_control_unit;
32
33 architecture mmu_control_unit_arch of mmu_control_unit is
34     component data_control_unit is
35         port (
36             eoc          : in  std_logic; -- High on muart has finished collecting data
37             eot          : in  std_logic; -- High on muart has finished transmitting
38             ready        : in  std_logic; -- High if the muart is ready for new transfer
39             data_read     : in  std_logic; -- High if the cpu requests a read, else write
40             data_req      : in  std_logic; -- Low to start a transfer
41             data_add_0    : in  std_logic; -- High for memory address, else IO
42             write         : out std_logic; -- Pulled high to start muart writing data.
43             data_ack      : inout std_logic; -- Idles 'Z', high when data not ready, pulled
44                 low when data ready
45             muart_input   : out muart_input_state; -- Signal connected to muart input
46             muart_output  : out muart_output_state; -- Signal connected to muart output
47             clk           : in  std_logic
48         );
49     end component;
50
51     component inst_control_unit is
52         port (
53             eoc          : in  std_logic; -- High on muart has finished collecting data
54             eot          : in  std_logic; -- High on muart has finished transmitting
55             ready        : in  std_logic; -- High if the muart is ready for new transfer
56             inst_req      : in  std_logic; -- Low to start a transfer
57             write         : out std_logic; -- Pulled high to start muart writing data
58             inst_or_data_in : out std_logic; -- Output headers inst or data bit
59             inst_ack      : out std_logic; -- Idles high, pulled low when data ready
60             muart_input   : out muart_input_state; -- Signal connected to muart input
61             muart_output  : out muart_output_state; -- Signal connected to muart output
62             clk           : in  std_logic
63         );
64     end component;
65
66     signal data_write, inst_write, inst_inst_or_data_out : std_logic;
67     signal data_muart_input, inst_muart_input : muart_input_state;
68     signal data_muart_output, inst_muart_output : muart_output_state;
```

```

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

```

Listing 16: mmu/data_control_unit.vhd

```

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

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
library work;
8 use work.mmu_types.all;

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

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

```



```

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

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

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

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

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

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

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

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

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

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

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

106                 when get_add_high =>

```

```

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

112     when get_data =>
113         if reader_state = finished then
114             next_get_state <= finished;
115         end if;

116     when finished =>
117         next_get_state <= idle;

120     when others =>
121         NULL;
122     end case;
123 end if;
124 end if;
125 end process get_data_fsm;

126 transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
127     if rising_edge(clk) then
128         if ((get_state = send_header) or
129             (get_state = send_add_low) or
130             (get_state = send_add_high) or
131             (get_state = send_data)) then
132             case transmitter_state is
133             when idle =>
134                 if ready = '1' and eot = '0' then
135                     next_transmitter_state <= set_data;
136                 end if;

138             when set_data =>
139                 next_transmitter_state <= trans_data;

142             when trans_data =>
143                 next_transmitter_state <= pause;

144             when pause =>
145                 if eot = '1' then
146                     next_transmitter_state <= finished;
147                 end if;

150             when finished =>
151                 next_transmitter_state <= idle;

152             when others =>
153                 NULL;
154             end case;
155         end if;
156     end if;
157 end process transmit_fsm;

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

170             when wait_data =>
171                 if eoc = '1' then
172                     next_reader_state <= read_data;
173                 end if;

174             when read_data =>
175                 next_reader_state <= pause;

176             when pause =>
177                 if eoc = '0' then
178                     next_reader_state <= finished;

```

```

        end if;
182
        when finished =>
184            next_reader_state <= idle;

186            when others =>
                NULL;
188            end case;
        end if;
190    end process read_fsm;

192    switch_states : process(
194        clk,
        next_state,
196        next_get_state,
        next_reader_state,
198        next_transmitter_state
    ) begin
200        if rising_edge(clk) then
            state <= next_state;
202            get_state <= next_get_state;
            reader_state <= next_reader_state;
204            transmitter_state <= next_transmitter_state;
        end if;
206    end process switch_states;

208    -- Outputs
    with state select
210        data_ack <= '1' when wait_clear,
                                'Z' when idle,
212                                '0' when others;

214    with transmitter_state select
        write <= '1' when trans_data,
216                '0' when others;

218    uart_input <= idle          when transmitter_state /= set_data      else
        idle                    when transmitter_state /= trans_data     else
220        header                  when get_state         = send_header    else
        data_add_high when get_state         = send_add_high else
222        data_add_low when get_state         = send_add_low  else
        data_data    when get_state         = send_data      else
224        idle;

226    uart_output <= clear_data when state = idle      else
        idle              when reader_state /= read_data else
228        header           when get_state = get_header else
        data_data         when get_state = get_data   else
230        idle;
    end data_control_unit_arch;

```

Listing 17: mmu/header_builder.vhd

```

1  -- Author:
   --      Forrest McKerchar
3
   -- builds a header to feed into the RS-232 link
5
   library IEEE;
7  use IEEE.STD_LOGIC_1164.ALL;

9  library work;

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

```

```

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

```

Listing 18: mmu/header_decoder.vhd

```

-- Author:
2 --      Forrest McKerchar

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

6 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

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

```

Listing 19: mmu/inst_control_unit.vhd

```

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

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
library work;
8 use work.mmu_types.all;

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

24
architecture inst_control_unit_arch of inst_control_unit is
26   type m_state_type is (

```

```

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

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

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

52     when wait_clear =>
       if (inst_req = '1') then
54         next_state <= idle;
56         end if;

58     when others =>
       NULL;
60     end case;
   end process inst_fsm;

62  get_inst_fsm : process(state, clk) begin
44     if state = get_data then
       case get_state is
66         when idle =>
           next_get_state <= send_header;

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

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

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

84         when get_header =>
           if reader_state = finished then
86             next_get_state <= get_add_low;
           end if;

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

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

98         when get_data_low =>

```

```

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

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

108     when finished =>
109         next_get_state <= idle;

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

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

128         when set_data =>
129             next_transmitter_state <= trans_data;

130         when trans_data =>
131             next_transmitter_state <= pause;

132         when pause =>
133             if eot = '1' then
134                 next_transmitter_state <= finished;
135             end if;

136         when finished =>
137             next_transmitter_state <= idle;

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

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

158         when wait_data =>
159             if eoc = '1' then
160                 next_reader_state <= read_data;
161             end if;

162         when read_data =>
163             next_reader_state <= pause;

164         when pause =>
165             if eoc = '0' then
166                 next_reader_state <= finished;
167             end if;

168         when finished =>
169             next_reader_state <= idle;

```

```

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

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

194 -- Outputs
195 with state select
196     inst_ack <= '1' when wait_clear,
197               '0' when others;

198 with state select
199     inst_or_data <= '0' when idle,
200                 '1' when others;

201 with transmitter_state select
202     write <= '1' when trans_data,
203           '0' when others;

204

206

208 uart_input <= idle          when transmitter_state /= set_data      else
209                 idle          when transmitter_state /= trans_data   else
210                 header        when get_state       = send_header    else
211                 inst_add_high  when get_state       = send_add_high  else
212                 inst_add_low   when get_state       = send_add_low   else
213                 idle;

214
215 uart_output <= idle          when reader_state /= read_data      else
216                 header        when get_state    = get_header     else
217                 inst_data_high when get_state    = get_data_high  else
218                 inst_data_low  when get_state    = get_data_low   else
219                 idle;

220 end inst_control_unit_arch;

```

Listing 20: mmu/mmu.vhd

```

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

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

6
7 library work;
8 use work.mmu_types.all;
9 use work.mmu_control_unit;
10 use work.header_builder;
11 use work.header_decoder;
12 use work.reg8;

13
14 use work.minimal_uart_core;

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

```

```

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

36 architecture mmu_arch of mmu_main is
    component mmu_control_unit is
38    port (
        eoc : in std_logic; -- High on muart has finished collecting
        data
40        eot : in std_logic; -- High on muart has finished transmitting
        ready : in std_logic; -- High if the muart is ready for new
        transfer
42        data_read : in std_logic; -- High if the cpu requests a read, else
        write
        data_req : in std_logic; -- Low to start a transfer
44        data_add_0 : in std_logic; -- High for memory address, else I0
        inst_req : in std_logic; -- Low to start a transfer
46        fr : in std_logic; -- Input headers fetch request bit
        inst_or_data_in : in std_logic; -- Input headers inst or data bit
48        rw : in std_logic; -- Input headers read!/write bit
        write : out std_logic; -- Pulled high to start muart writing data
50        inst_or_data_out : out std_logic; -- Output headers inst or data bit
        inst_ack : out std_logic; -- Idles high, pulled low when data ready
52        data_ack : inout std_logic; -- Idles 'Z', high when data not ready,
        pulled low when data ready
        muart_input : out muart_input_state; -- Signal connected to muart input
54        muart_output : out muart_output_state; -- Signal connected to muart
        output
        clk : in std_logic
    );
56 end component;

58 component header_builder is
60    port (
        read_write : in std_logic; -- 1 = read, 0 = write
62        inst_data : in std_logic; -- 1 = inst, 0 = data
        header : out std_logic_vector(7 downto 0)
64    );
    end component;

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

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

```



```

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

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

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

114 signal write     : std_logic;
116 signal inst_or_data_out : std_logic;
118 signal muart_input  : muart_input_state;
120 signal muart_output : muart_output_state;

122 signal muart_out : std_logic_vector(7 downto 0);
124 signal muart_in  : std_logic_vector(7 downto 0);
126 signal header_in : std_logic_vector(7 downto 0);
128 signal header_out : std_logic_vector(7 downto 0);
130 signal inst_data_high_enable : std_logic;
132 signal inst_data_low_enable : std_logic;
134 signal data_data_enable : std_logic;
136 signal data_line_tri : std_logic_vector(7 downto 0);

138 begin
140 muart : minimal_uart_core port map (
142     clk,
144     eoc,
146     muart_out,
148     receive_pin,
150     transfer_pin,
152     eot,
154     muart_in,
156     ready,
158     write
    );
160 cu : mmu_control_unit port map (
162     eoc,
164     eot,
166     ready,
168     data_read,
170     data_req,
172     data_add(0),
174     inst_req,
176     fr,
178     inst_or_data_in,
180     rw,
182     write,
184     inst_or_data_out,
186     inst_ack,
188     data_ack,
190     muart_input,
192     muart_output,
194     clk
    );
196 hb : header_builder port map (
198     data_read,
200     inst_or_data_out,
202     header_out

```

```

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

186  with muart_input select
      muart_in <= header_out          when header,
188                  "0000" & inst_add(11 downto 8) when inst_add_high,
                  inst_add(7  downto 0)         when inst_add_low,
190                  '0' & data_add(15 downto 9)    when data_add_high,
                  data_add(8  downto 1)         when data_add_low,
192                  data_line                  when data_data,
                  (others => '0')              when others;

194  route_output : process(muart_output, muart_out) begin
196      header_in <= (others => '0');
      inst_data_high_enable <= '0';
198      inst_data_low_enable <= '0';
      data_data_enable <= '0';
200      case muart_output is
          when header =>
202          header_in <= muart_out;

204          when inst_data_high =>
              inst_data_high_enable <= '1';

206          when inst_data_low =>
              inst_data_low_enable <= '1';

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

220  end mmu_arch;

```

Listing 21: mmu/mmu_types.vhd

```

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

4 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;

```

```

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

```

Listing 22: mmu/muart/BRG.vhd

```

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

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

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

54   begin
56     txd : process (clock)
       begin
58       if (rising_edge(clock)) then
         if (count_brg_txd = brdvd) then

```

```

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

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

```

Listing 23: mmu/muart/serial.vhd

```

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

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

```

```

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

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

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

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

```

```

115         eocs          <= '0';
        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     end case;

153     end if;
155     end process rxd_state_machine;

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

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

179     process(atual_state)
181     begin
        if (atual_state=s9) then
183             outp <= data;
        end if;
185     end process;

```

```

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';
256               next_state_txd <= s9;
257
258           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 ;

```

Test Benchs

Listing 24: processor/alu_tb.vhd

```

library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

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

8 architecture behav of alu_tb is
    -- Declaration of the component that will be instantiated.
10 component alu
    Port (f      : in    STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
12         rx     : in    STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
         ry     : in    STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
14         ro     : out   STD_LOGIC_VECTOR (7 downto 0); -- Output Normaly (Ry)
         Cin    : in    STD_LOGIC; -- Carry in
16         sr     : out   STD_LOGIC_VECTOR (2 downto 0)); -- Status register out Z(0),
                C(1), N(2)
    end component;

18 -- Specifies which entity is bound with the component.
for alu_0: alu use entity work.alu;
20     signal f      : STD_LOGIC_VECTOR (3 downto 0);
    signal rx, ry, ro : STD_LOGIC_VECTOR (7 downto 0);
22     signal Cin    : STD_LOGIC;
    signal sr       : STD_LOGIC_VECTOR (2 downto 0);
24 begin
    -- Component instantiation.
26     alu_0: alu port map (f => f, rx => rx, ry => ry, ro => ro, Cin => Cin, sr => sr);

28     -- This process does the real job.
    process
30     type pattern_type is record
        f      : STD_LOGIC_VECTOR (3 downto 0);
32     rx, ry    : STD_LOGIC_VECTOR (7 downto 0);
        ro     : STD_LOGIC_VECTOR (7 downto 0);
34     Cin      : STD_LOGIC;
        sr     : STD_LOGIC_VECTOR (2 downto 0);
36     end record;
    -- The patterns to apply.
38     type pattern_array is array (natural range <>) of pattern_type;
    constant patterns : pattern_array :=
40     -- f      rx      ry      ro      Cin  sr
    (( "0001", "00000000", "00000000", "00000000", '0', "001"), --AND tests - 1ns
42     ("0001", "00000001", "00000001", "00000001", '0', "000"), --AND tests
     ("0001", "00000000", "00000001", "00000000", '0', "001"), --AND tests
44     ("0001", "10101010", "10101010", "10101010", '0', "100"), --AND tests
     ("0001", "01010101", "01010101", "01010101", '0', "000"), --AND tests - 5ns
46     ("0001", "11111111", "00000000", "00000000", '0', "001"), --AND tests
     ("0001", "11111111", "11111111", "11111111", '0', "100"), --AND tests
48     ("0001", "00000000", "01010101", "00000000", '0', "001"), --AND tests
     ("0001", "00000000", "10101010", "00000000", '0', "001"), --AND tests
50     ("0001", "11111111", "01010101", "01010101", '0', "000"), --AND tests - 10 ns
     ("0001", "11111111", "10101010", "10101010", '0', "100"), --AND tests
52     ("0001", "10000011", "10110010", "10000010", '0', "100"), --AND tests
     ("0001", "00000011", "00110010", "00000010", '0', "000"), --AND tests

54     ("0011", "00000000", "00000000", "00000000", '0', "001"), --OR tests - 14 ns
     ("0011", "00000001", "00000001", "00000001", '0', "000"), --OR tests
     ("0011", "00000000", "00000001", "00000001", '0', "000"), --OR tests
58     ("0011", "10101010", "10101010", "10101010", '0', "100"), --OR tests
     ("0011", "01010101", "01010101", "01010101", '0', "000"), --OR tests
60     ("0011", "11111111", "00000000", "11111111", '0', "100"), --OR tests
     ("0011", "11111111", "11111111", "11111111", '0', "100"), --OR tests - 20 ns
62     ("0011", "00000000", "01010101", "01010101", '0', "000"), --OR tests
     ("0011", "00000000", "10101010", "10101010", '0', "100"), --OR tests
64     ("0011", "11111111", "01010101", "11111111", '0', "100"), --OR tests
     ("0011", "11111111", "10101010", "11111111", '0', "100"), --OR tests
66     ("0011", "10000011", "10110010", "10110011", '0', "100"), --OR tests - 25 ns
     ("0011", "00000011", "00110010", "00110011", '0', "000"), --OR tests

```

```

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

```

Listing 25: processor/fulladder_tb.vhd

```

library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

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

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

```

```

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

26     -- This process does the real job.
    process
28     type pattern_type is record
        A      : STD_LOGIC_VECTOR( 7 downto 0);
30        B      : STD_LOGIC_VECTOR( 7 downto 0);
        Cin     : STD_LOGIC;
32        Sum     : STD_LOGIC_VECTOR( 7 downto 0);
        Cout     : STD_LOGIC;
34    end record;
    -- The patterns to apply.
36    type pattern_array is array (natural range <>) of pattern_type;
    constant patterns : pattern_array :=
38    -- A      B      Cin     Sum     Cout
    (( "00000000", "00000000", '0', "00000000", '0'), --AND tests - 1ns
40     ("11111111", "11111111", '1', "11111111", '1'), --AND tests
     ("00000000", "00000000", '1', "00000001", '0'), --AND tests
42     ("00000000", "11111111", '0', "11111111", '0'), --AND tests
     ("11111111", "00000000", '0', "11111111", '0'), --AND tests
44     ("11111111", "00000000", '1', "00000000", '1'), --AND tests
     ("10101010", "01010101", '0', "11111111", '0'), --AND tests
46     ("10101010", "01010101", '1', "00000000", '1'), --AND tests
     ("11111111", "11111111", '0', "11111110", '1') --XOR tests
48 );
begin
50     -- Check each pattern.
    for i in patterns'range loop
52         -- Set the inputs.
        A <= patterns(i).A;
54         B <= patterns(i).B;
        Cin <= patterns(i).Cin;
56         -- Wait for the results.
        wait for 1 ns;
58         -- Check the outputs.
        assert Sum = patterns(i).Sum
60         report "The sum check failed" severity error;
        assert Cout = patterns(i).Cout
62         report "The carry out is wrong" severity error;
    end loop;
64     assert false report "end of test" severity note;
    -- Wait forever; this will finish the simulation.
66     wait;
end process;
68 end behav;

```

Listing 26: 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

   architecture TB of spr_TB is

11     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;
        enable    : in    STD_LOGIC;                -- Enable write
27        reset    : in    STD_LOGIC;                -- Reset the register
        Ri        : in    STD_LOGIC_VECTOR (15 downto 0); -- The input to the SPR
29        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";

```

```

85     wait for 20 ns;
    assert (sr_Ro = "1001100101110100") report "Wrote to sr while disabled" severity
        error;
    assert (pc_Ro = "1010101010101010") report "Read pc #3 failed" severity error;
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 <= "1111111000000000";
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 = "1111111000000000") 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
    for TB
117     end for;
    end CFG_TB;

```

Listing 27: 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
12             : IN std_logic; -- Clock
            enable
                : IN std_logic; -- Enable input
            (output is always enabled)
            SelRx, SelRy, SelRi : IN std_logic_vector(2 DOWNTO 0); -- Selecti which
                registers to use
14             Ri
                : IN std_logic_vector(7 DOWNTO 0); -- Input
            Rx, Ry
                : OUT std_logic_vector(7 DOWNTO 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 DOWNTO 0);
        signal Ri, Rx, Ry
            : std_logic_vector(7 DOWNTO 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;
31     clk <= '1';
32     wait for 5 ns;
33 end process;
34
35 -- This process does the real job.
36 process
37 begin
38
39     -- Write to R0
40     SelRi <= "000";
41     Ri <= "00010100";
42     enable <= '1';
43     wait for 20 ns;
44
45     -- Read R0 from Rx
46     SelRx <= "000";
47     wait for 20 ns;
48     assert (Rx = "00010100") report "Read from Rx failed #1" severity error;
49
50     -- Read R0 from Ry
51     SelRy <= "000";
52     wait for 20 ns;
53     assert (Ry = "00010100") report "Read from Ry failed #1" severity error;
54
55     -- Disable write
56     enable <= '0';
57     wait for 20 ns;
58
59     -- Change Ri (should not write as it is disabled)
60     Ri <= "00101010";
61     wait for 20 ns;
62     assert (Rx = "00010100") report "Wrote to register while disabled #1" severity
        error;
63     assert (Ry = "00010100") report "Wrote to register while disabled #2" severity
        error;
64
65     -- Enable write
66     enable <= '1';
67     wait for 20 ns;
68     assert (Rx = "00101010") report "Read from Rx failed #2" severity error;
69     assert (Ry = "00101010") report "Read from Ry failed #2" severity error;
70
71     -- Write to R2
72     SelRi <= "010";
73     Ri <= "01010001";
74     wait for 20 ns;
75
76     -- Read R2 from Rx
77     SelRx <= "010";
78     wait for 20 ns;
79     assert (Rx = "01010001") report "Read from Rx failed #3" severity error;
80
81     -- Read R2 from Ry
82     SelRy <= "010";
83     wait for 20 ns;
84     assert (Ry = "01010001") report "Read from Ry failed #3" severity error;
85
86     -- Read R0 from Rx again (should not have changed from previous results)
87     SelRx <= "000";
88     wait for 20 ns;
89     assert (Rx = "00101010") report "Read from Rx failed #4" severity error;
90
91     -- Read R0 from Ry again (should not have changed from previous results)
92     SelRy <= "000";
93     wait for 20 ns;
94     assert (Ry = "00101010") report "Read from Ry failed #4" severity error;
95
96     -- Wait for a long time
97     wait for 1 ms;
98     assert (Rx = "00101010") report "Read from Rx failed #5" severity error;
99     assert (Ry = "00101010") report "Read from Ry failed #5" severity error;
100

```

```

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

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

112 end process;
end behav;

```

Listing 28: mmu/mmu_tb.vhd

```

-- Author:
2 --      Wim Looman

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
library work;
8 use work.mmu_main;
use work.minimal_uart_core;
10 use work.txt_util.all;

12 entity mmu_tb is
end mmu_tb;

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

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

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

```

```

52  signal inst_req      : std_logic := '1';
    signal inst_ack     : std_logic;
54  signal data_add     : std_logic_vector(15 downto 0);
    signal data_line    : std_logic_vector(7  downto 0);
56  signal data_read    : std_logic;
    signal data_req     : std_logic;
58  signal data_ack     : std_logic;
    signal clk          : std_logic;
60  signal receive_pin  : std_logic;
    signal transfer_pin : std_logic;
62
    signal eoc, rxd, txd, eot, ready, wr: std_logic;
64  signal outp, inp : std_logic_vector(7  downto 0);

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

72  rxd <= transfer_pin;
    receive_pin <= txd;
74
    clk_gen : process begin
76      clk <= '0';
        wait for 10 ns;
78      clk <= '1';
        wait for 10 ns;
80  end process;

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

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

110     assert false report "passed header" severity note;

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

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

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

```



```

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

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

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

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

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

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

168     wait for 100 ns;
    inp <= patterns(i).inst_data(15 downto 8);
170     wait for 20 ns;
    wr <= '1';
172     wait for 20 ns;
    wr <= '0';
174     wait until eot'event;
    wait until eot'event;
176

    assert inst_ack = '1'
178         report "receipt not acknowledged"
            severity error;
180

    assert inst_data = patterns(i).inst_data
182         report "Wrong data recieve expected '" & str(patterns(i).inst_data) & "'
            recieved '" & str(inst_data) & "'
            severity error;
184

    assert false report "finished transmission" severity note;
186

    wait for 20 ns;
188
    inst_req <= '1';
190 end loop;
    wait;
192 end process;

```

```
end tb;
```

Listing 29: 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
49   port(
      clock : in  std_logic;
51       eoc : out std_logic;
      outp : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
53       rxd : in  std_logic;
      txd : out  std_logic;
55       eot : out std_logic;
      inp : in  std_logic_vector(7 downto 0);
57       ready : out std_logic;
      wr : in  std_logic
59   );
  end component;
```

```

61     signal inst_add      : std_logic_vector(11 downto 0);
63     signal inst_data    : std_logic_vector(15 downto 0);
64     signal inst_req     : std_logic := '1';
65     signal inst_ack     : std_logic;
66     signal data_add     : std_logic_vector(15 downto 0);
67     signal data_line    : std_logic_vector(7 downto 0);
68     signal data_read    : std_logic;
69     signal data_req     : std_logic;
70     signal data_ack     : std_logic;
71     signal clk          : std_logic;
72     signal receive_pin  : std_logic;
73     signal transfer_pin : std_logic;
74     signal sw1, sw2    : std_logic;
75
76     signal eoc, rxd, txd, eot, ready, wr : std_logic;
77     signal outp, inp : std_logic_vector(7 downto 0);
78 begin
79     m : mmu_main port map (inst_add, inst_data, inst_req, inst_ack, data_add,
80         data_line, data_read, data_req, data_ack, clk, receive_pin, transfer_pin);
81     i : IO port map (data_add, data_line, data_read, data_req, data_ack, clk,
82         sw1, sw2);
83     muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);
84
85     rxd <= transfer_pin;
86     receive_pin <= txd;
87
88     clk_gen : process begin
89         clk <= '0';
90         wait for 10 ns;
91         clk <= '1';
92         wait for 10 ns;
93     end process;
94
95     data_test : process
96         type pattern_type is record
97             data_add      : std_logic_vector(15 downto 0);
98             recv_head     : std_logic_vector( 7 downto 0);
99             send_head     : std_logic_vector( 7 downto 0);
100            data_data      : std_logic_vector( 7 downto 0);
101            switch_data    : std_logic_vector( 1 downto 0);
102            rw              : std_logic;
103        end record;
104        type pattern_array is array (natural range <>) of pattern_type;
105        constant patterns : pattern_array :=
106            -- data_add, recv_head, send_head, data_data, switch_data, rw
107            ((x"0581", x"80", x"00", x"A7", "00", '1' ),
108             (x"0273", x"00", x"00", x"5E", "00", '0' ));
109    begin
110        wr <= '0';
111        data_req <= '1';
112        for i in patterns'range loop
113            wait for 10000 ns;
114            data_add <= patterns(i).data_add;
115            data_read <= patterns(i).rw;
116            wait for 20 ns;
117            if patterns(i).rw = '0' then
118                data_line <= patterns(i).data_data;
119            else
120                data_line <= (others => 'Z');
121            end if;
122            wait for 20 ns;
123            data_req <= '0';
124
125            if patterns(i).data_add(0) = '1' then
126                wait until eoc'event;
127                assert outp = patterns(i).recv_head
128                    report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved
129                        '" & str(outp) & '"
130                    severity error;
131                wait until eoc'event;
132
133                assert false report "passed header" severity note;

```

```

131
132     wait until eoc'event;
133     assert outp = patterns(i).data_add(8 downto 1)
134         report "Bad address low expected '" & str(patterns(i).data_add(7 downto
135             0)) & "' recieved '" & str(outp) & "'"
136         severity error;
137     wait until eoc'event;
138
139     assert false report "passed address low" severity note;
140
141     wait until eoc'event;
142     assert outp = "0" & patterns(i).data_add(15 downto 9)
143         report "Bad address high expected '" & str(patterns(i).data_add(11 downto
144             8)) & "' recieved '" & str(outp) & "'"
145         severity error;
146     wait until eoc'event;
147
148     assert false report "passed address high" severity note;
149     if patterns(i).rw = '0' then
150         wait until eoc'event;
151         assert outp = patterns(i).data_data
152             report "Bad data expected '" & str(patterns(i).data_data) & "' recieved
153                 '" & str(outp) & "'"
154             severity error;
155         wait until eoc'event;
156
157         assert false report "passed data" severity note;
158     else
159         wait for 100 ns;
160         inp <= patterns(i).send_head;
161         wait for 20 ns;
162         wr <= '1';
163         wait for 20 ns;
164         wr <= '0';
165         wait until eot'event;
166         wait until eot'event;
167
168
169         wait for 100 ns;
170         inp <= patterns(i).data_add(8 downto 1);
171         wait for 20 ns;
172         wr <= '1';
173         wait for 20 ns;
174         wr <= '0';
175         wait until eot'event;
176         wait until eot'event;
177
178
179         wait for 100 ns;
180         inp <= "0" & patterns(i).data_add(15 downto 9);
181         wait for 20 ns;
182         wr <= '1';
183         wait for 20 ns;
184         wr <= '0';
185         wait until eot'event;
186         wait until eot'event;
187
188         wait for 100 ns;
189         inp <= patterns(i).data_data;
190         wait for 20 ns;
191         wr <= '1';
192         wait for 20 ns;
193         wr <= '0';
194         wait until eot'event;
195         wait until eot'event;
196     end if;
197 else
198
199 end if;
200
201 assert data_ack = '1'
202     report "receipt not acknowledged"

```

```

201         severity error;

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

209     assert false report "finished transmission" severity note;

211     wait for 20 ns;

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

```

Tools

Listing 30: assembler.rb

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

3 # Author:
4 #     Wim Looman
5 # Copyright:
6 #     Copyright (c) 2010 Wim Looman
7 # License:
8 #     GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)
9
10 def assert(error=nil)
11   raise (error || "Assertion Failed!") unless yield
12 end
13
14 # For 8-bit twos complement
15 def twos_complement(num)
16   return 256 + num
17 end
18
19
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
27
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 < 0
42       v = twos_complement(v)
43     end
44     return v
45   end
46 end
47
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
55
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
65
66
67 def convert(lines)
68   table = first_pass(lines)
```

```

69  return second_pass(lines, table)
end
71

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

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

105  def convert_line(line, symbols, line_no)
    chunks = line.sub(" ", " ").split#.partition(";")[0].split
107
    case chunks[0]
109      when "LDI"
        instruction = 0x21
111        x = register(chunks[1], 4)
        v = immediate(chunks[2])
113        operands = (v << 2) + x

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

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

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

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

```

```

139     operands = (y << 5) + x
140 elif 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 elif 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)
203
204 when "BLT"
205     instruction = 0x2B
206     v = immediate(chunks[1], symbols, line_no)
207     operands = (v << 2)
208
209 when "BGT"
210     instruction = 0x2F
211     v = immediate(chunks[1], symbols, line_no)

```



```

211     operands = (v << 2)

213     when "BC"
214         instruction = 0x33
215         v = immediate(chunks[1], symbols, line_no)
216         operands = (v << 2)

217     when "BNC"
218         instruction = 0x37
219         v = immediate(chunks[1], symbols, line_no)
220         operands = (v << 2)

221     when "RJMPP"
222         instruction = 0x3B
223         v = immediate(chunks[1], symbols, line_no)
224         operands = (v << 2)

225     when "JMP"
226         instruction = 0x2F
227         y = register(chunks[1], 3)
228         operands = (y << 5)
229     end
230     opcode = (instruction << 10) + operands
231     return [(opcode >> 8), (opcode & 0xFF)]
232 end

233 if __FILE__ == $0
234     if !(1..2).include?(ARGV.length) || !File.exist?(ARGV[0])
235         p "Usage: ruby #{__FILE__} <input_file> [<output_file>]"
236         exit
237     end

238     input = IO.readlines(ARGV[0])

239     output = convert(input)
240     if ARGV.length == 2:
241         File.open(ARGV[1], "wb") do |file|
242             output.each do |char|
243                 file.putc(char)
244             end
245         end
246     else
247         output.each do |char|
248             $stdout.putc(char)
249         end
250     end
251 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
13 def serve(program, data_file, sp)
14     while 1
15         header = sp.getc
16         diagnostic_mode = (header >> 2) & 0x03
17         instruction = header & 0x01 == 0x01
18         address = sp.getc + (sp.getc << 8)

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

```

```

22     header = 0x82
23     header += 0x01 if instruction
24     sp.putc(header)
25     instruction ? sp.write(program[address]) :
26         sp.write(data_file[address])
27     p "Sending data for #{instruction ? "Instruction" : "Data"} bus, address:" +
28         "#{address}, data: #{instruction ? program[address] : data_file[address]}"
29     when 0x00 # write, doesn't support writing to instruction memory
30         data = sp.getc
31         data_file[address] = data
32     p "Writing data, address: " + address + ", data: " + data_file[address]
33     end
34 end
35 end
36
37 if __FILE__ == $0
38
39     if ARGV.size < 3
40         STDERR.print "Usage: ruby #{$0} <device> <baud_rate> <program_file>
41             [<data_file>]\n"
42         exit
43     end
44
45     device = ARGV[0]
46     baud_rate = ARGV[1].to_i
47
48     program = Array.new(2**12, 0x00)
49     i = 0
50     File.open(ARGV[2], 'rb') do |input|
51         input.each_byte do |byte|
52             program[i] = byte
53             i += 1
54         end
55     end
56
57     data = Array.new(2**15, 0x00)
58     File.open(ARGV[3], 'rb') do |input|
59         input.each_byte do |byte|
60             data += byte
61         end
62     end if ARGV.size > 3
63
64     sp = SerialPort.new(device, baud_rate, 8, 1, SerialPort::NONE)
65     serve(program, data, sp)
66 end

```
