```vhdl
-- Copyright (C) 1991-2013 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.

-- PROGRAM     "Quartus II 64-Bit"
-- VERSION     "Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Full Version"
-- CREATED     "Sat Mar 04 15:01:59 2017"

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY ELEC374 IS
   PORT
   (
      ren0 :  IN  STD_LOGIC;
      ren1 :  IN  STD_LOGIC;
      ren2 :  IN  STD_LOGIC;
      ren3 :  IN  STD_LOGIC;
      ren4 :  IN  STD_LOGIC;
      ren5 :  IN  STD_LOGIC;
      ren6 :  IN  STD_LOGIC;
      ren7 :  IN  STD_LOGIC;
      ren8 :  IN  STD_LOGIC;
      ren9 :  IN  STD_LOGIC;
      renA :  IN  STD_LOGIC;
      renB :  IN  STD_LOGIC;
      renC :  IN  STD_LOGIC;
      renD :  IN  STD_LOGIC;
      renE :  IN  STD_LOGIC;
      renF :  IN  STD_LOGIC;
      clr :  IN  STD_LOGIC;
      clk :  IN  STD_LOGIC;
      R0out :  IN  STD_LOGIC;
      R1out :  IN  STD_LOGIC;
      R2out :  IN  STD_LOGIC;
      R3out :  IN  STD_LOGIC;
      R4out :  IN  STD_LOGIC;
      R5out :  IN  STD_LOGIC;
      R6out :  IN  STD_LOGIC;
      R7out :  IN  STD_LOGIC;
      R8out :  IN  STD_LOGIC;
      R9out :  IN  STD_LOGIC;
      R10out :  IN  STD_LOGIC;
      R11out :  IN  STD_LOGIC;
      R12out :  IN  STD_LOGIC;
      R13out :  IN  STD_LOGIC;
      R14out :  IN  STD_LOGIC;
      R15out :  IN  STD_LOGIC;
      Zhighout :  IN  STD_LOGIC;
      Zlowout :  IN  STD_LOGIC;
```

```
		InPortout :  IN  STD_LOGIC;
		HIin :  IN  STD_LOGIC;
		PCin :  IN  STD_LOGIC;
		LOin :  IN  STD_LOGIC;
		MDRin :  IN  STD_LOGIC;
		HIoutEn :  IN  STD_LOGIC;
		LOoutEn :  IN  STD_LOGIC;
		PCoutEn :  IN  STD_LOGIC;
		MDRoutEn :  IN  STD_LOGIC;
		CoutEn :  IN  STD_LOGIC;
		reny :  IN  STD_LOGIC;
		renz :  IN  STD_LOGIC;
		ReadIn :  IN  STD_LOGIC;
		busmuxout :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		Empty :  IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
		HIout :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		LOout :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		Mdatain :  IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
		MDRout :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		operation :  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
		PCout :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout0 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout1 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout2 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout3 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout4 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout5 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout6 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout7 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout8 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		rout9 :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		routa :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		routb :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		routc :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		routd :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		route :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		routf :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		routy :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		Zhigh :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0);
		Zlow :  INOUT  STD_LOGIC_VECTOR(31 DOWNTO 0)
	);
END ELEC374;

ARCHITECTURE bdf_type OF ELEC374 IS 

COMPONENT alu
	PORT(clk : IN STD_LOGIC;
		 a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
		 b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
		 op : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
		 y : OUT STD_LOGIC_VECTOR(63 DOWNTO 0)
	);
END COMPONENT;

COMPONENT reg
	PORT(clr : IN STD_LOGIC;
		 clk : IN STD_LOGIC;
		 ren : IN STD_LOGIC;
		 rin : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
		 rout : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
	);
END COMPONENT;
```

```

COMPONENT good_mux
   PORT(data0x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data10x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data11x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data12x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data13x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data14x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data15x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data16x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data17x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data18x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data19x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data1x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data20x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data21x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data22x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data23x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data24x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data25x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data2x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data3x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data4x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data5x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data6x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data7x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data8x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data9x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        sel : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
   );
END COMPONENT;

COMPONENT encoder32to5
   PORT(R0out : IN STD_LOGIC;
        R1out : IN STD_LOGIC;
        R2out : IN STD_LOGIC;
        R3out : IN STD_LOGIC;
        R4out : IN STD_LOGIC;
        R5out : IN STD_LOGIC;
        R6out : IN STD_LOGIC;
        R7out : IN STD_LOGIC;
        R8out : IN STD_LOGIC;
        R9out : IN STD_LOGIC;
        R10out : IN STD_LOGIC;
        R11out : IN STD_LOGIC;
        R12out : IN STD_LOGIC;
        R13out : IN STD_LOGIC;
        R14out : IN STD_LOGIC;
        R15out : IN STD_LOGIC;
        HIout : IN STD_LOGIC;
        LOout : IN STD_LOGIC;
        Zhighout : IN STD_LOGIC;
        Zlowout : IN STD_LOGIC;
        PCout : IN STD_LOGIC;
        MDRout : IN STD_LOGIC;
        InPortout : IN STD_LOGIC;
        Cout : IN STD_LOGIC;
        Sin : OUT STD_LOGIC_VECTOR(4 DOWNTO 0)
   );
END COMPONENT;

```

```
COMPONENT mdmux
   PORT(ReadIn : IN STD_LOGIC;
       BusMuxOut : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
       Mdatain : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
       MDMuxOut : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
   );
END COMPONENT;

COMPONENT reg64
   PORT(clr : IN STD_LOGIC;
       clk : IN STD_LOGIC;
       ren : IN STD_LOGIC;
       rin : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
       rh : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
       rlow : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
   );
END COMPONENT;

SIGNAL   Cout :  STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL   InPortout0 :  STD_LOGIC;
SIGNAL   InPortout1 :  STD_LOGIC;
SIGNAL   InPortout10 :  STD_LOGIC;
SIGNAL   InPortout11 :  STD_LOGIC;
SIGNAL   InPortout12 :  STD_LOGIC;
SIGNAL   InPortout13 :  STD_LOGIC;
SIGNAL   InPortout14 :  STD_LOGIC;
SIGNAL   InPortout15 :  STD_LOGIC;
SIGNAL   InPortout16 :  STD_LOGIC;
SIGNAL   InPortout17 :  STD_LOGIC;
SIGNAL   InPortout18 :  STD_LOGIC;
SIGNAL   InPortout19 :  STD_LOGIC;
SIGNAL   InPortout2 :  STD_LOGIC;
SIGNAL   InPortout20 :  STD_LOGIC;
SIGNAL   InPortout21 :  STD_LOGIC;
SIGNAL   InPortout22 :  STD_LOGIC;
SIGNAL   InPortout23 :  STD_LOGIC;
SIGNAL   InPortout24 :  STD_LOGIC;
SIGNAL   InPortout25 :  STD_LOGIC;
SIGNAL   InPortout26 :  STD_LOGIC;
SIGNAL   InPortout27 :  STD_LOGIC;
SIGNAL   InPortout28 :  STD_LOGIC;
SIGNAL   InPortout29 :  STD_LOGIC;
SIGNAL   InPortout3 :  STD_LOGIC;
SIGNAL   InPortout30 :  STD_LOGIC;
SIGNAL   InPortout31 :  STD_LOGIC;
SIGNAL   InPortout4 :  STD_LOGIC;
SIGNAL   InPortout5 :  STD_LOGIC;
SIGNAL   InPortout6 :  STD_LOGIC;
SIGNAL   InPortout7 :  STD_LOGIC;
SIGNAL   InPortout8 :  STD_LOGIC;
SIGNAL   InPortout9 :  STD_LOGIC;
SIGNAL   SYNTHESIZED_WIRE_0 :  STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL   SYNTHESIZED_WIRE_1 :  STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL   SYNTHESIZED_WIRE_2 :  STD_LOGIC_VECTOR(63 DOWNTO 0);

SIGNAL   GDFX_TEMP_SIGNAL_0 :  STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN

GDFX_TEMP_SIGNAL_0 <= (InPortout31 & InPortout30 & InPortout29 & InPortout28 & InPortout27 ↲
& InPortout26 & InPortout25 & InPortout24 & InPortout23 & InPortout22 & InPortout21 & ↲
InPortout20 & InPortout19 & InPortout18 & InPortout17 & InPortout16 & InPortout15 & ↲
```

```
InPortout14 & InPortout13 & InPortout12 & InPortout11 & InPortout10 & InPortout9 & 
InPortout8 & InPortout7 & InPortout6 & InPortout5 & InPortout4 & InPortout3 & InPortout2 & 
InPortout1 & InPortout0);


b2v_ALU : alu
PORT MAP(clk => clk,
       a => routy,
       b => busmuxout,
       op => operation,
       y => SYNTHESIZED_WIRE_2);


b2v_H : reg
PORT MAP(clr => clr,
       clk => clk,
       ren => HIin,
       rin => busmuxout,
       rout => HIout);


b2v_inst : good_mux
PORT MAP(data0x => Empty,
       data10x => rout9,
       data11x => routa,
       data12x => routb,
       data13x => routc,
       data14x => routd,
       data15x => route,
       data16x => routf,
       data17x => HIout,
       data18x => LOout,
       data19x => Zhigh,
       data1x => rout0,
       data20x => Zlow,
       data21x => PCout,
       data22x => MDRout,
       data23x => GDFX_TEMP_SIGNAL_0,
       data24x => Cout,
       data25x => Empty,
       data2x => rout1,
       data3x => rout2,
       data4x => rout3,
       data5x => rout4,
       data6x => rout5,
       data7x => rout6,
       data8x => rout7,
       data9x => rout8,
       sel => SYNTHESIZED_WIRE_0,
       result => busmuxout);


b2v_inst1 : encoder32to5
PORT MAP(R0out => R0out,
       R1out => R1out,
       R2out => R2out,
       R3out => R3out,
       R4out => R4out,
       R5out => R5out,
       R6out => R6out,
       R7out => R7out,
       R8out => R8out,
		R9out => R9out,
		R10out => R10out,
		R11out => R11out,
		R12out => R12out,
		R13out => R13out,
		R14out => R14out,
		R15out => R15out,
		HIout => HIoutEn,
		LOout => LOoutEn,
		Zhighout => Zhighout,
		Zlowout => Zlowout,
		PCout => PCoutEn,
		MDRout => MDRoutEn,
		InPortout => InPortout,
		Cout => CoutEn,
		Sin => SYNTHESIZED_WIRE_0);


b2v_inst2 : mdmux
PORT MAP(ReadIn => ReadIn,
		BusMuxOut => busmuxout,
		Mdatain => Mdatain,
		MDMuxOut => SYNTHESIZED_WIRE_1);


b2v_LO : reg
PORT MAP(clr => clr,
		clk => clk,
		ren => LOin,
		rin => busmuxout,
		rout => LOout);


b2v_MDR : reg
PORT MAP(clr => clr,
		clk => clk,
		ren => MDRin,
		rin => SYNTHESIZED_WIRE_1,
		rout => MDRout);


b2v_PC : reg
PORT MAP(clr => clr,
		clk => clk,
		ren => PCin,
		rin => busmuxout,
		rout => PCout);


b2v_R0 : reg
PORT MAP(clr => clr,
		clk => clk,
		ren => ren0,
		rin => busmuxout,
		rout => rout0);


b2v_R1 : reg
PORT MAP(clr => clr,
		clk => clk,
		ren => ren1,
		rin => busmuxout,
		rout => rout1);


	b2v_R2 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren2,
		 rin => busmuxout,
		 rout => rout2);


	b2v_R3 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren3,
		 rin => busmuxout,
		 rout => rout3);


	b2v_R4 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren4,
		 rin => busmuxout,
		 rout => rout4);


	b2v_R5 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren5,
		 rin => busmuxout,
		 rout => rout5);


	b2v_R6 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren6,
		 rin => busmuxout,
		 rout => rout6);


	b2v_R7 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren7,
		 rin => busmuxout,
		 rout => rout7);


	b2v_R8 : reg
	PORT MAP(clr => clr,
		 clk => clk,
		 ren => ren8,
		 rin => busmuxout,
		 rout => rout8);


	b2v_R9 : reg
	PORT MAP(clr => clr,
		 clk => clk,
             ren => ren9,
             rin => busmuxout,
             rout => rout9);


      b2v_Ra : reg
      PORT MAP(clr => clr,
             clk => clk,
             ren => renA,
             rin => busmuxout,
             rout => routa);


      b2v_Rb : reg
      PORT MAP(clr => clr,
             clk => clk,
             ren => renB,
             rin => busmuxout,
             rout => routb);


      b2v_Rc : reg
      PORT MAP(clr => clr,
             clk => clk,
             ren => renC,
             rin => busmuxout,
             rout => routc);


      b2v_Rd : reg
      PORT MAP(clr => clr,
             clk => clk,
             ren => renD,
             rin => busmuxout,
             rout => routd);


      b2v_Re : reg
      PORT MAP(clr => clr,
             clk => clk,
             ren => renE,
             rin => busmuxout,
             rout => route);


      b2v_regz : reg64
      PORT MAP(clr => clr,
             clk => clk,
             ren => renz,
             rin => SYNTHESIZED_WIRE_2,
             rh => Zhigh,
             rlow => Zlow);


      b2v_Rf : reg
      PORT MAP(clr => clr,
             clk => clk,
             ren => renF,
             rin => busmuxout,
             rout => routf);


```

```
b2v_RY : reg
PORT MAP(clr => clr,
       clk => clk,
       ren => reny,
       rin => busmuxout,
       rout => routy);


END bdf_type;
```