--- Current MMRSRC "control" VHDL Code with LD/LDR Request Bug Fixed

--- Current file name: control.vhd

--- Last Revised: 7/31/2019; 2:26 p.m.

--- Author: WDR

--- Copyright: William D. Richard, Ph.D.

LIBRARY IEEE ;

USE IEEE.STD\_LOGIC\_1164.ALL ;

USE IEEE.STD\_LOGIC\_ARITH.ALL ;

ENTITY control IS

PORT (clk : IN STD\_LOGIC ;

opcode : IN STD\_LOGIC\_VECTOR(4 DOWNTO 0) ;

con : IN STD\_LOGIC ;

done : IN STD\_LOGIC ;

reset\_l : IN STD\_LOGIC ;

grant : IN STD\_LOGIC ;

request : OUT STD\_LOGIC ;

a\_in : OUT STD\_LOGIC ;

c\_in : OUT STD\_LOGIC ;

c\_out : OUT STD\_LOGIC ;

pc\_in : OUT STD\_LOGIC ;

pc\_out : OUT STD\_LOGIC ;

c1\_out : OUT STD\_LOGIC ;

c2\_out : OUT STD\_LOGIC ;

ir\_in : OUT STD\_LOGIC ;

gra : OUT STD\_LOGIC ;

grb : OUT STD\_LOGIC ;

grc : OUT STD\_LOGIC ;

r\_in : OUT STD\_LOGIC ;

r\_out : OUT STD\_LOGIC ;

ba\_out : OUT STD\_LOGIC ;

md\_bus : OUT STD\_LOGIC ;

md\_rd : OUT STD\_LOGIC ;

md\_wr : OUT STD\_LOGIC ;

md\_out : OUT STD\_LOGIC ;

ma\_in : OUT STD\_LOGIC ;

read : OUT STD\_LOGIC ;

write : OUT STD\_LOGIC ;

add : OUT STD\_LOGIC ;

sub : OUT STD\_LOGIC ;

andx : OUT STD\_LOGIC ;

orx : OUT STD\_LOGIC ;

notx : OUT STD\_LOGIC ;

neg : OUT STD\_LOGIC ;

c\_eq\_b : OUT STD\_LOGIC ;

inc4 : OUT STD\_LOGIC ;

shr : OUT STD\_LOGIC ;

shra : OUT STD\_LOGIC ;

shl : OUT STD\_LOGIC ;

shc : OUT STD\_LOGIC) ;

END control ;

ARCHITECTURE behavioral of control IS

TYPE states IS (s0, s1, s2, s3, s4, s5, s6, s7) ;

SIGNAL state : states ;

SIGNAL nxt\_state : states ;

CONSTANT nop\_op : std\_logic\_vector (4 DOWNTO 0) := "00000" ;

CONSTANT ld\_op : std\_logic\_vector (4 DOWNTO 0) := "00001" ;

CONSTANT ldr\_op : std\_logic\_vector (4 DOWNTO 0) := "00010" ;

CONSTANT st\_op : std\_logic\_vector (4 DOWNTO 0) := "00011" ;

CONSTANT str\_op : std\_logic\_vector (4 DOWNTO 0) := "00100" ;

CONSTANT la\_op : std\_logic\_vector (4 DOWNTO 0) := "00101" ;

CONSTANT lar\_op : std\_logic\_vector (4 DOWNTO 0) := "00110" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "00111" ;

CONSTANT br\_op : std\_logic\_vector (4 DOWNTO 0) := "01000" ;

CONSTANT brl\_op : std\_logic\_vector (4 DOWNTO 0) := "01001" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "01010" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "01011" ;

CONSTANT add\_op : std\_logic\_vector (4 DOWNTO 0) := "01100" ;

CONSTANT addi\_op : std\_logic\_vector (4 DOWNTO 0) := "01101" ;

CONSTANT sub\_op : std\_logic\_vector (4 DOWNTO 0) := "01110" ;

CONSTANT neg\_op : std\_logic\_vector (4 DOWNTO 0) := "01111" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "10000" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "10001" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "10010" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "10011" ;

CONSTANT and\_op : std\_logic\_vector (4 DOWNTO 0) := "10100" ;

CONSTANT andi\_op : std\_logic\_vector (4 DOWNTO 0) := "10101" ;

CONSTANT or\_op : std\_logic\_vector (4 DOWNTO 0) := "10110" ;

CONSTANT ori\_op : std\_logic\_vector (4 DOWNTO 0) := "10111" ;

CONSTANT not\_op : std\_logic\_vector (4 DOWNTO 0) := "11000" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "11001" ;

CONSTANT shr\_op : std\_logic\_vector (4 DOWNTO 0) := "11010" ;

CONSTANT shra\_op : std\_logic\_vector (4 DOWNTO 0) := "11011" ;

CONSTANT shl\_op : std\_logic\_vector (4 DOWNTO 0) := "11100" ;

CONSTANT shc\_op : std\_logic\_vector (4 DOWNTO 0) := "11101" ;

---CONSTANT unused\_op: std\_logic\_vector (4 DOWNTO 0) := "11110" ;

CONSTANT stop\_op : std\_logic\_vector (4 DOWNTO 0) := "11111" ;

BEGIN

clkd:PROCESS(clk)

BEGIN

IF (clk'EVENT AND clk='1') THEN

IF (reset\_l = '0') THEN

state <= s0;

ELSE

state <= nxt\_state;

END IF;

END IF;

END PROCESS clkd;

state\_trans:PROCESS(opcode,state,done,grant)

BEGIN

CASE state IS

WHEN s0 => nxt\_state <= s1 ;

WHEN s1 => IF (grant = '1' AND done = '1') THEN

nxt\_state <= s2 ;

ELSE

nxt\_state <= s1 ;

END IF ;

WHEN s2 => nxt\_state <= s3 ;

WHEN s3 => IF ((opcode=nop\_op) OR (opcode=br\_op)) THEN

nxt\_state <= s0 ;

ELSIF (opcode=stop\_op) THEN

nxt\_state <= s3 ;

ELSE

nxt\_state <= s4 ;

END IF;

WHEN S4 => IF (opcode=brl\_op OR opcode=neg\_op OR opcode=not\_op OR

opcode=shr\_op OR opcode=shl\_op OR opcode=shra\_op OR

opcode=shc\_op)THEN

nxt\_state <= s0 ;

ELSE

nxt\_state <= s5 ;

END IF;

WHEN S5 => IF ((opcode=and\_op) OR (opcode=andi\_op) OR

(opcode=or\_op) OR (opcode=ori\_op) OR

(opcode=add\_op) OR (opcode=addi\_op) OR

(opcode=sub\_op) OR (opcode=la\_op) OR

(opcode=lar\_op)) THEN

nxt\_state <= s0 ;

ELSE

nxt\_state <= s6 ;

END IF ;

WHEN S6 => IF ((opcode=ld\_op OR opcode=ldr\_op) AND (NOT(done = '1') OR grant = '0')) THEN

nxt\_state <= s6 ;

ELSE

nxt\_state <= s7 ;

END IF ;

WHEN S7 => IF ((opcode=st\_op OR opcode=str\_op) AND (NOT(done = '1') OR grant = '0')) THEN

nxt\_state <= s7 ;

ELSE

nxt\_state <= s0 ;

END IF ;

END CASE ;

END PROCESS state\_trans ;

output:PROCESS(state,opcode,con,done,grant)

BEGIN

request <= '0' ;

a\_in <= '0' ;

c\_in <= '0' ;

c\_out <= '0' ;

pc\_in <= '0' ;

pc\_out <= '0' ;

c1\_out <= '0' ;

c2\_out <= '0' ;

ir\_in <= '0' ;

gra <= '0' ;

grb <= '0' ;

grc <= '0' ;

r\_in <= '0' ;

r\_out <= '0' ;

ba\_out <= '0' ;

md\_bus <= '0' ;

md\_rd <= '0' ;

md\_wr <= '0' ;

md\_out <= '0' ;

ma\_in <= '0' ;

read <= 'Z' ;

write <= 'Z' ;

add <= '0' ;

sub <= '0' ;

andx <= '0' ;

orx <= '0' ;

notx <= '0' ;

neg <= '0' ;

c\_eq\_b <= '0' ;

inc4 <= '0' ;

shr <= '0' ;

shra <= '0' ;

shl <= '0' ;

shc <= '0' ;

CASE state IS

WHEN s0 =>

request <= '1' ;

pc\_out <= '1' ;

ma\_in <= '1' ;

inc4 <= '1' ;

c\_in <= '1' ;

WHEN s1 =>

IF (grant = '0') THEN

request <= '1' ;

ELSE --- grant = '1'

read <= '1' ;

c\_out <= '1' ;

pc\_in <= '1' ;

IF NOT(done = '1') THEN

request <= '1' ;

ELSE

md\_rd <= '1' ;

END IF ;

END IF ;

WHEN s2 =>

md\_out <= '1' ;

ir\_in <= '1' ;

WHEN s3 =>

IF (opcode=add\_op OR opcode=addi\_op OR opcode=and\_op OR opcode=andi\_op OR

opcode=or\_op OR opcode=ori\_op or opcode=sub\_op) THEN

grb <= '1' ;

r\_out <= '1' ;

a\_in <= '1' ;

ELSIF (opcode=br\_op) THEN

grb <= '1' ;

r\_out <= '1' ;

IF (con = '1') THEN

pc\_in <= '1' ;

END IF ;

ELSIF (opcode=brl\_op) THEN

gra <= '1' ;

r\_in <= '1' ;

pc\_out <= '1' ;

ELSIF (opcode=st\_op OR opcode=la\_op OR opcode=ld\_op) THEN

grb <= '1' ;

ba\_out <= '1' ;

a\_in <= '1' ;

ELSIF (opcode=ldr\_op OR opcode=str\_op OR opcode=lar\_op) THEN

pc\_out <= '1' ;

a\_in <= '1' ;

ELSIF (opcode=shc\_op OR opcode=shl\_op OR opcode=shr\_op OR opcode=shra\_op) THEN

grb <= '1' ;

r\_out <= '1' ;

c\_in <= '1' ;

IF (opcode=shc\_op) THEN

shc <= '1' ;

ELSIF (opcode=shl\_op) THEN

shl <= '1' ;

ELSIF (opcode=shr\_op) THEN

shr <= '1' ;

ELSE

shra <= '1' ;

END IF ;

ELSIF (opcode=neg\_op OR opcode=not\_op) THEN

grc <= '1' ;

r\_out <= '1' ;

c\_in <= '1' ;

IF (opcode=not\_op) THEN

notx <= '1' ;

ELSIF (opcode=neg\_op) THEN

neg <= '1' ;

END IF ;

END IF ; -- (nop and stop make no assignments)

WHEN s4 =>

IF (opcode=add\_op OR opcode=and\_op OR opcode=or\_op OR opcode=sub\_op) THEN

grc <= '1' ;

r\_out <= '1' ;

c\_in <= '1' ;

IF (opcode=and\_op) THEN

andx <= '1' ;

ELSIF (opcode=or\_op) THEN

orx <= '1' ;

ELSIF (opcode=sub\_op) THEN

sub <= '1' ;

ELSE

add <= '1' ;

END IF ;

ELSIF (opcode=addi\_op OR opcode=andi\_op OR opcode=ori\_op) THEN

c2\_out <= '1' ;

c\_in <= '1' ;

IF (opcode=addi\_op) THEN

add <= '1' ;

ELSIF (opcode=andi\_op) THEN

andx <= '1' ;

ELSE

orx <= '1' ;

END IF ;

ELSIF (opcode=brl\_op) THEN

grb <= '1' ;

r\_out <= '1' ;

IF (con = '1') THEN

pc\_in <= '1' ;

END IF ;

ELSIF (opcode=st\_op OR opcode=la\_op OR opcode=ld\_op) THEN

c2\_out <= '1' ;

add <= '1' ;

c\_in <= '1' ;

ELSIF (opcode=ldr\_op OR opcode=str\_op OR opcode=lar\_op) THEN

c1\_out <= '1' ;

add <= '1' ;

c\_in <= '1' ;

ELSIF (opcode=neg\_op OR opcode=not\_op OR opcode=shc\_op OR opcode=shl\_op OR opcode=shr\_op OR opcode=shra\_op) THEN

c\_out <= '1' ;

gra <= '1' ;

r\_in <= '1' ;

END IF ;

WHEN s5 =>

IF (opcode=add\_op OR opcode=addi\_op OR opcode=and\_op OR opcode=andi\_op OR

opcode=or\_op OR opcode=ori\_op or opcode=sub\_op) THEN

c\_out <= '1' ;

gra <= '1' ;

r\_in <= '1' ;

ELSIF (opcode=st\_op) THEN

c\_out <= '1' ;

ma\_in <= '1' ;

ELSIF (opcode=ld\_op) THEN

request <= '1' ;

c\_out <= '1' ;

ma\_in <= '1' ;

ELSIF (opcode=la\_op OR opcode=lar\_op) THEN

c\_out <= '1' ;

gra <= '1' ;

r\_in <= '1' ;

ELSIF (opcode=ldr\_op) THEN

request <= '1' ;

c\_out <= '1' ;

ma\_in <= '1' ;

ELSIF (opcode=str\_op) THEN

c\_out <= '1' ;

ma\_in <= '1' ;

END IF ;

WHEN s6 =>

IF (opcode=st\_op) THEN

request <= '1' ;

gra <= '1' ;

r\_out <= '1' ;

md\_bus <= '1' ;

ELSIF (opcode=ld\_op) THEN

IF (grant = '0') THEN

request <= '1' ;

ELSE --- grant = '1'

read <= '1' ;

IF NOT(done = '1') THEN

request <= '1' ;

ELSE

md\_rd <= '1' ;

END IF ;

END IF ;

ELSIF (opcode=ldr\_op) THEN

IF (grant = '0') THEN

request <= '1' ;

ELSE --- grant = '1'

read <= '1' ;

IF NOT(done = '1') THEN

request <= '1' ;

ELSE

md\_rd <= '1' ;

END IF ;

END IF ;

ELSIF (opcode=str\_op) THEN

request <= '1' ;

gra <= '1' ;

r\_out <= '1' ;

md\_bus <= '1' ;

END IF ;

WHEN s7 =>

IF (opcode=st\_op) THEN

IF (grant = '0') THEN

request <= '1' ;

ELSE --- grant = '1'

write <= '1' ;

md\_wr <= '1' ;

IF NOT(done = '1') THEN

request <= '1' ;

END IF ;

END IF ;

ELSIF (opcode=ld\_op) THEN

md\_out <= '1' ;

gra <= '1' ;

r\_in <= '1' ;

ELSIF (opcode=ldr\_op) THEN

md\_out <= '1' ;

gra <= '1' ;

r\_in <= '1' ;

ELSIF (opcode=str\_op) THEN

IF (grant = '0') THEN

request <= '1' ;

ELSE --- grant = '1'

write <= '1' ;

md\_wr <= '1' ;

IF NOT(done = '1') THEN

request <= '1' ;

END IF ;

END IF ;

END IF ;

END CASE ;

END PROCESS output ;

END behavioral ;