Jiacheng Zhang

Lab 8 Report

ECE 2031 L07

14 Mar 2022

图形用户界面, 文本, 应用程序

描述已自动生成

**Figure 1.** Schematic of a Bit Pusher peripheral implementing functionality of writing and storing 1-bit values in an array. The value of the parameter “num\_bits” represents the length of the array.

APPENDIX A   
VHDL IMPLEMENTATION OF EXTENDED MEMORY PERIPHERAL WITH AUTOMATIC ADDRESS INCREMENT FUNCTION

-- SCOMP peripheral that can store and read data in an

-- additional RAM.

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

library altera\_mf;

use altera\_mf.altera\_mf\_components.all;

entity EXT\_STORAGE is

port(

clk : in std\_logic ;

resetn : in std\_logic ;

io\_write : in std\_logic ;

cs\_addr : in std\_logic ;

cs\_data : in std\_logic ;

data\_word : inout std\_logic\_vector(15 downto 0)

);

end entity;

architecture internals of EXT\_STORAGE is

type write\_states is (idle, storing);

signal wstate: write\_states;

-- signal to hold the current address

signal mem\_addr : std\_logic\_vector(15 downto 0);

-- temporary storage for incoming data

signal mem\_data : std\_logic\_vector(15 downto 0);

-- internal word\_out signal from memory

signal word\_out\_int : std\_logic\_vector(15 downto 0);

-- memory write signal

signal mw : std\_logic;

begin

-- create the memory using altsyncram

altsyncram\_component : altsyncram

GENERIC MAP (

numwords\_a => 65536,

widthad\_a => 16,

width\_a => 16,

intended\_device\_family => "CYCLONE V",

clock\_enable\_input\_a => "BYPASS",

clock\_enable\_output\_a => "BYPASS",

lpm\_hint => "ENABLE\_RUNTIME\_MOD=NO",

lpm\_type => "altsyncram",

operation\_mode => "SINGLE\_PORT",

outdata\_aclr\_a => "NONE",

outdata\_reg\_a => "UNREGISTERED",

power\_up\_uninitialized => "FALSE",

read\_during\_write\_mode\_port\_a => "NEW\_DATA\_NO\_NBE\_READ",

width\_byteena\_a => 1

)

PORT MAP (

wren\_a => mw,

clock0 => clk,

address\_a => mem\_addr,

data\_a => mem\_data,

q\_a => word\_out\_int

);

-- Interface the data output with IO\_DATA, making it hi-Z when

-- not being used

data\_word <= word\_out\_int when ((cs\_data='1') and (io\_write='0')) else "ZZZZZZZZZZZZZZZZ";

process(clk, resetn, cs\_addr)

begin

-- For this implementation, saving the memory address

-- doesn't require anything special. Just latch it when

-- SCOMP sends it.

if resetn = '0' then

wstate <= idle;

mem\_addr <= x"0000";

elsif rising\_edge(clk) then

-- If SCOMP is writing to the address register...

if (io\_write = '1') and (cs\_addr='1') then

mem\_addr <= data\_word;

end if;

--implement address auto-increment feature

--(post-increment)

case wstate is

when idle =>

--read from memory

if (io\_write = '0') and (cs\_data='1') then

mem\_addr <= mem\_addr + 1;

--write to memory

elsif (io\_write = '1') and (cs\_data='1') then

wstate <= storing;

end if;

when storing =>

mem\_addr <= mem\_addr + 1;

wstate <= idle;

when others =>

wstate <= idle;

end case;

end if;

-- The sequence of events needed to store data into memory

-- will be implemented with a state machine.

-- Although there are ways to more simply connect SCOMP's

-- I/O system to an altsyncram module, it would only work

-- with under specific circumstances, and would be limited -- to just simple writes. Since you will probably want to

-- do more complicated things, this is an example of

-- something that could be extended to do more complicated

-- things.

-- Note that 'mw' is \*not\* implemented as a Moore output of

-- this state machine, because Moore outputs are

-- susceptible to glitches, and that's a bad thing for

-- memory control signals.

if resetn = '0' then

wstate <= idle;

mw <= '0';

mem\_data <= x"0000";

-- Note that resetting this device does NOT clear the

-- memory.

-- Clearing memory would require cycling through each

-- address and setting them all to 0.

elsif rising\_edge(clk) then

case wstate is

when idle =>

if (io\_write = '1') and (cs\_data='1') then

-- latch the current data into the temporary

-- storage register,

-- because this is the only time it'll be

-- available

mem\_data <= data\_word;

-- can raise mw on the upcoming transition,

-- because data won't be stored until next clock

-- cycle.

mw <= '1';

-- Change state

wstate <= storing;

end if;

when storing =>

-- All that's needed here is to lower mw. The RAM

-- will be storing data on this clock edge, so mw can

-- go low at the same time.

mw <= '0';

wstate <= idle;

when others =>

wstate <= idle;

end case;

end if;

end process;

end internals;