Jiacheng Zhang
Lab 8 Report
ECE 2031 L07
14 Mar 2022

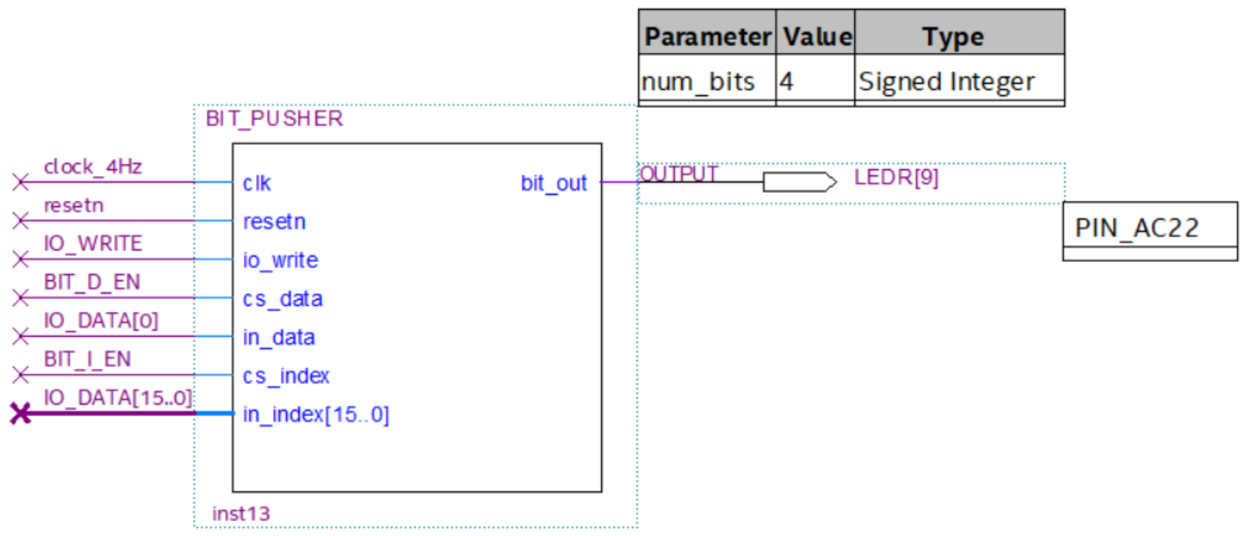

**Figure 1.** Schematic of a Bit Pusher peripheral implementing functionality of writing and storing 1-bit values in an array. The value of the parameter "num_bits" represents the length of the array.

# 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;
```

```vhdl
-- 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;
```