Lab 02
Jonathan Gusler
1/30/2019

Overview

This lab involves altering a sorting circuit to use SRAM instead of four registers. This change requires alterations to the hardware that selects which register to read from and write to. Some signals will be removed, and one will be added.

Background

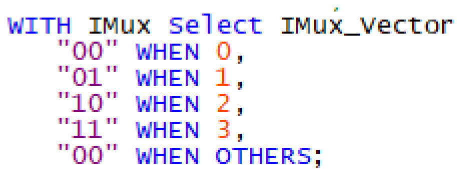
The hardware sorts by sequentially comparing data in one register, register A, which starts at the first register, to all the unsorted registers, which are known as register B in each compare, until a value less than what is in register A is found. Once a lesser value is found, the two registers' contents are swapped and the hardware proceeds to check the rest of the registers, if any registers are left. Once all the registers have been compared against, the position as register A gets assigned to the next register in sequence, and the process starts over. The original code uses four registers for memory, an integer signal that is decoded to a one-hot output to select the registers, and a mux that selects which register's data to put on the data path to the registers which hold the values to be compared. The SRAM hardware is provided by Quartus, so the only changes to be made are adding the SRAM to the components file and changing the original sorter code. All code pertinent to the lab, both original and edited, can be found in Appendix A. This includes all hardware files and the component file. None of the code describing the States needs to be changed and thus will not be discussed or shown. Also in the Appendix is the waveform for the original code to show its functionality.

Discussion

The first step is figuring out how the SRAM behaves. After some testing, the following behavioral points were learned:

-Always outputs data
-Outputs whatever is in the register named by the address line
-Output updates on the rising clock edge
-Data is loaded into memory when the signal wren, write enable, is HIGH
-Can load and output the same data at the same time assuming data is loaded on the rising clock edge

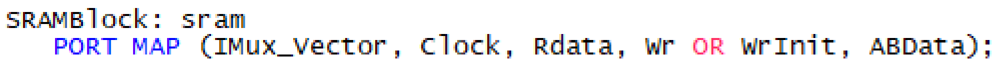
Knowing this behavior, decisions about what code to add, remove, and leave alone can be made. The decoder for selecting the register, the memory registers themselves, and the mux selecting which register to read will all be replaced by the SRAM. Code will be needed for converting from the integer-type data, which originally selected the registers, to vector-type data because the SRAM takes vector-type data for its inputs. The code for converting the data types is shown in *Figure 1*.



```
WITH IMux Select IMux_Vector
    "00" WHEN 0,
    "01" WHEN 1,
    "10" WHEN 2,
    "11" WHEN 3,
    "00" WHEN OTHERS;
```

Figure 1: Code to convert from integer-type IMux signal to vector-type IMux_Vector signal.

The SRAM port mapping is shown in *Figure 2*.



```
SRAMBlock: sram
   PORT MAP (IMux_Vector, Clock, Rdata, Wr OR WrInit, ABData);
```

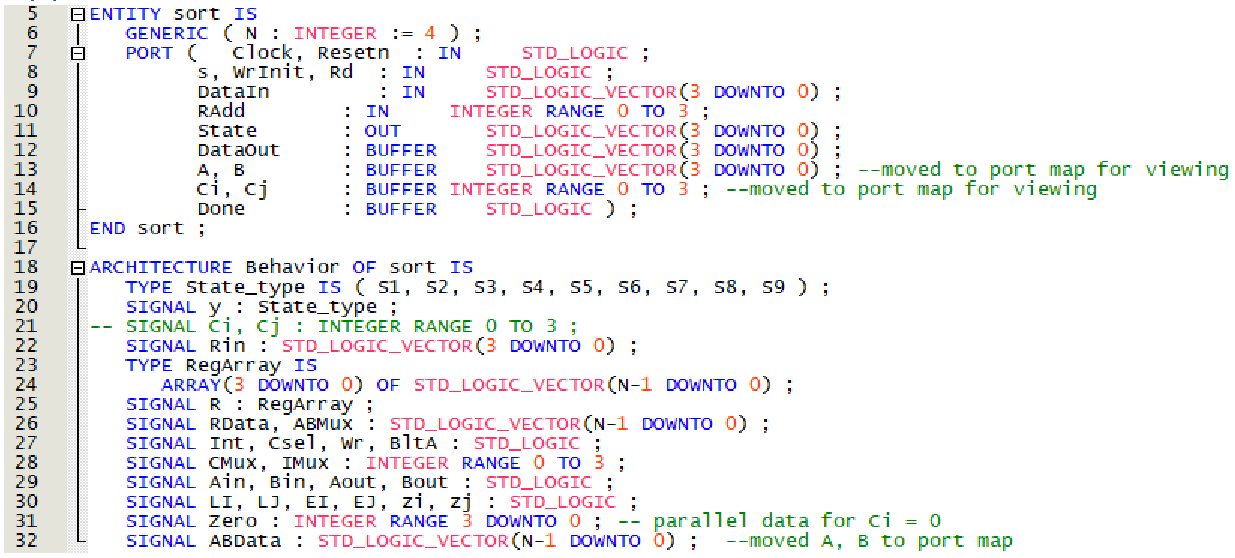
Figure 2: SRAM port map in context of the sorter’s signals.

All other changes are marked in the edited code.

Analysis and Results

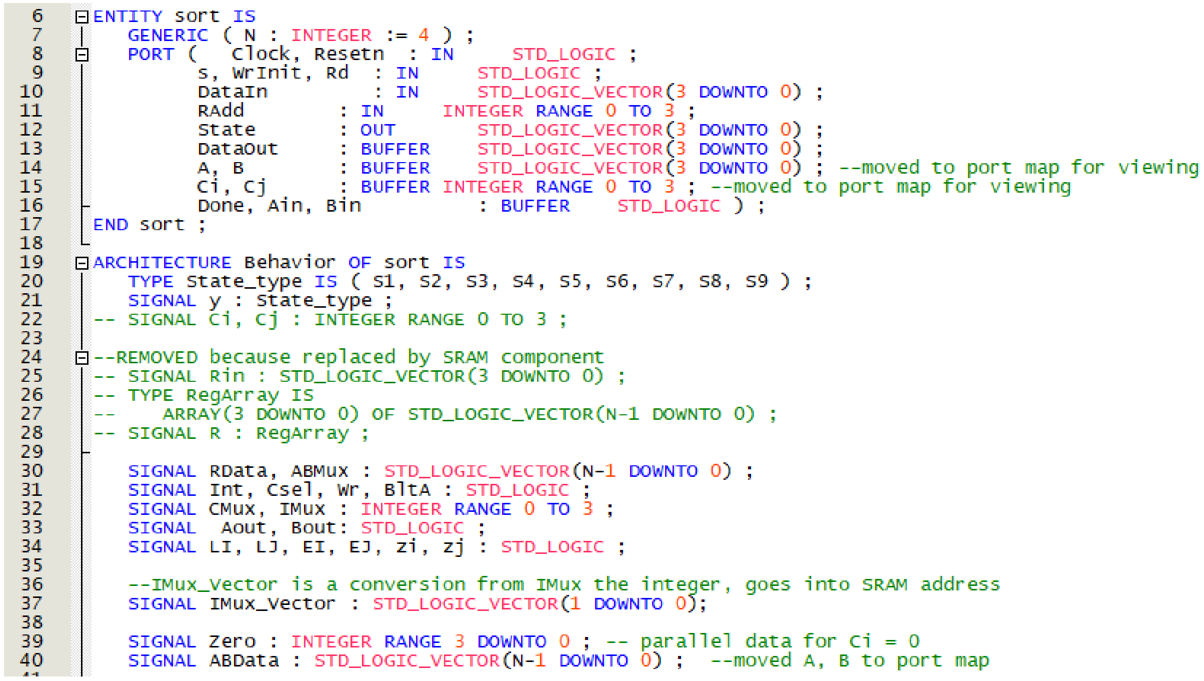
The original code did run as expected. Regrettably, with the edited code, a data type error presented itself while trying to run the waveform and the source of it could not be found. However, the altered code did at least compile.

Appendix A



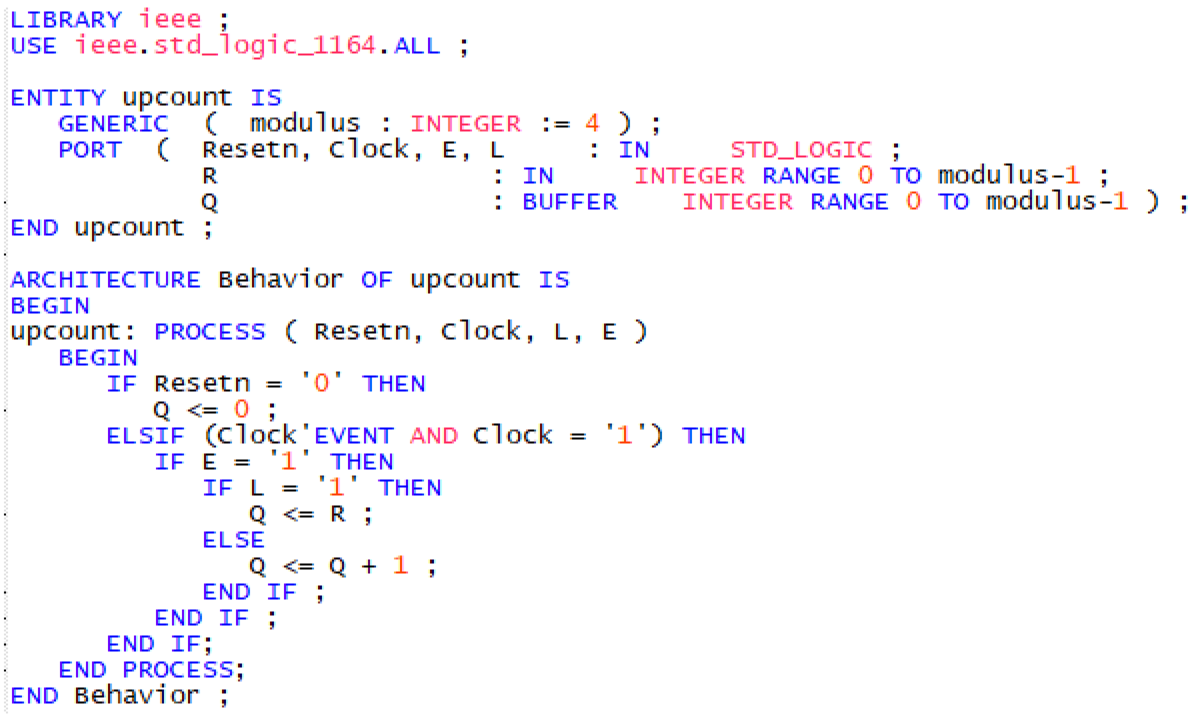
```vhdl
ENTITY sort IS
    GENERIC ( N : INTEGER := 4 ) ;
    PORT (   Clock, Resetn  : IN      STD_LOGIC ;
          s, WrInit, Rd  : IN     STD_LOGIC ;
          DataIn         : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          RAdd        : IN     INTEGER RANGE 0 TO 3 ;
          State       : OUT       STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          DataOut     : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          A, B        : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ; --moved to port map for viewing
          Ci, Cj      : BUFFER INTEGER RANGE 0 TO 3 ; --moved to port map for viewing
          Done        : BUFFER    STD_LOGIC ) ;
END sort ;

ARCHITECTURE Behavior OF sort IS
    TYPE State_type IS ( S1, S2, S3, S4, S5, S6, S7, S8, S9 ) ;
    SIGNAL y : State_type ;
--  SIGNAL Ci, Cj : INTEGER RANGE 0 TO 3 ;
    SIGNAL Rin : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
    TYPE RegArray IS
        ARRAY(3 DOWNTO 0) OF STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL R : RegArray ;
    SIGNAL RData, ABMux : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL Int, Csel, Wr, BltA : STD_LOGIC ;
    SIGNAL CMux, IMux : INTEGER RANGE 0 TO 3 ;
    SIGNAL Ain, Bin, Aout, Bout : STD_LOGIC ;
    SIGNAL LI, LJ, EI, EJ, zi, zj : STD_LOGIC ;
    SIGNAL Zero : INTEGER RANGE 3 DOWNTO 0 ; -- parallel data for Ci = 0
    SIGNAL ABData : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;   --moved A, B to port map
```

Figure 3: Port Map and Signals for original code.



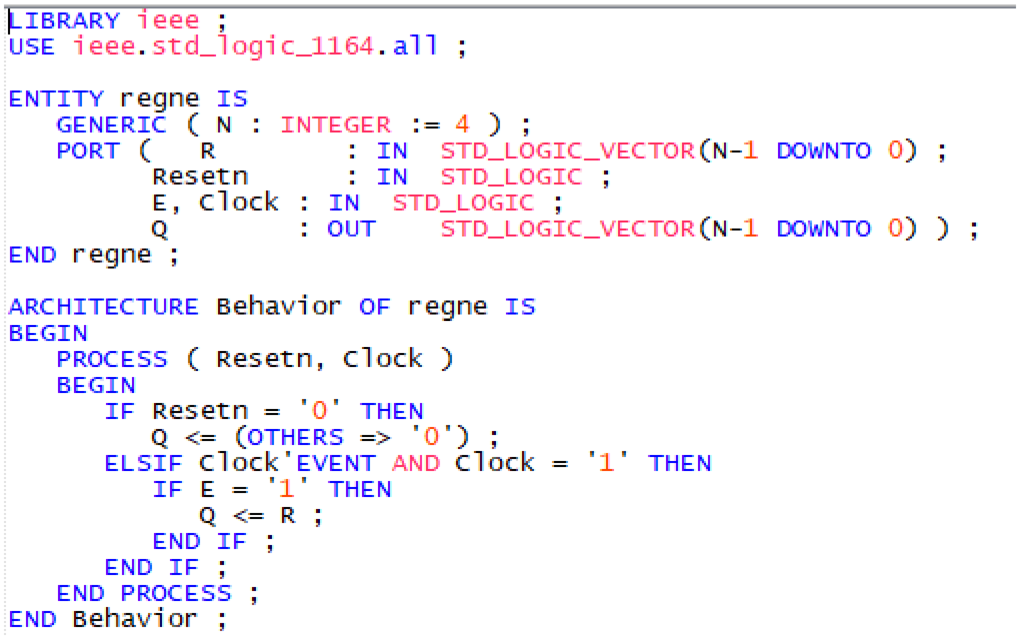
```vhdl
ENTITY sort IS
    GENERIC ( N : INTEGER := 4 ) ;
    PORT (   Clock, Resetn  : IN      STD_LOGIC ;
          s, WrInit, Rd  : IN     STD_LOGIC ;
          DataIn         : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          RAdd        : IN     INTEGER RANGE 0 TO 3 ;
          State       : OUT       STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          DataOut     : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          A, B        : BUFFER    STD_LOGIC_VECTOR(3 DOWNTO 0) ; --moved to port map for viewing
          Ci, Cj      : BUFFER INTEGER RANGE 0 TO 3 ; --moved to port map for viewing
          Done, Ain, Bin          : BUFFER    STD_LOGIC ) ;
END sort ;

ARCHITECTURE Behavior OF sort IS
    TYPE State_type IS ( S1, S2, S3, S4, S5, S6, S7, S8, S9 ) ;
    SIGNAL y : State_type ;
--  SIGNAL Ci, Cj : INTEGER RANGE 0 TO 3 ;

--REMOVED because replaced by SRAM component
--  SIGNAL Rin : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
--  TYPE RegArray IS
--      ARRAY(3 DOWNTO 0) OF STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
--  SIGNAL R : RegArray ;

    SIGNAL RData, ABMux : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL Int, Csel, Wr, BltA : STD_LOGIC ;
    SIGNAL CMux, IMux : INTEGER RANGE 0 TO 3 ;
    SIGNAL  Aout, Bout: STD_LOGIC ;
    SIGNAL LI, LJ, EI, EJ, zi, zj : STD_LOGIC ;

    --IMux_Vector is a conversion from IMux the integer, goes into SRAM address
    SIGNAL IMux_Vector : STD_LOGIC_VECTOR(1 DOWNTO 0);

    SIGNAL Zero : INTEGER RANGE 3 DOWNTO 0 ; -- parallel data for Ci = 0
    SIGNAL ABData : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;  --moved A, B to port map
```

Figure 4: Port Map and Signals for revised code.



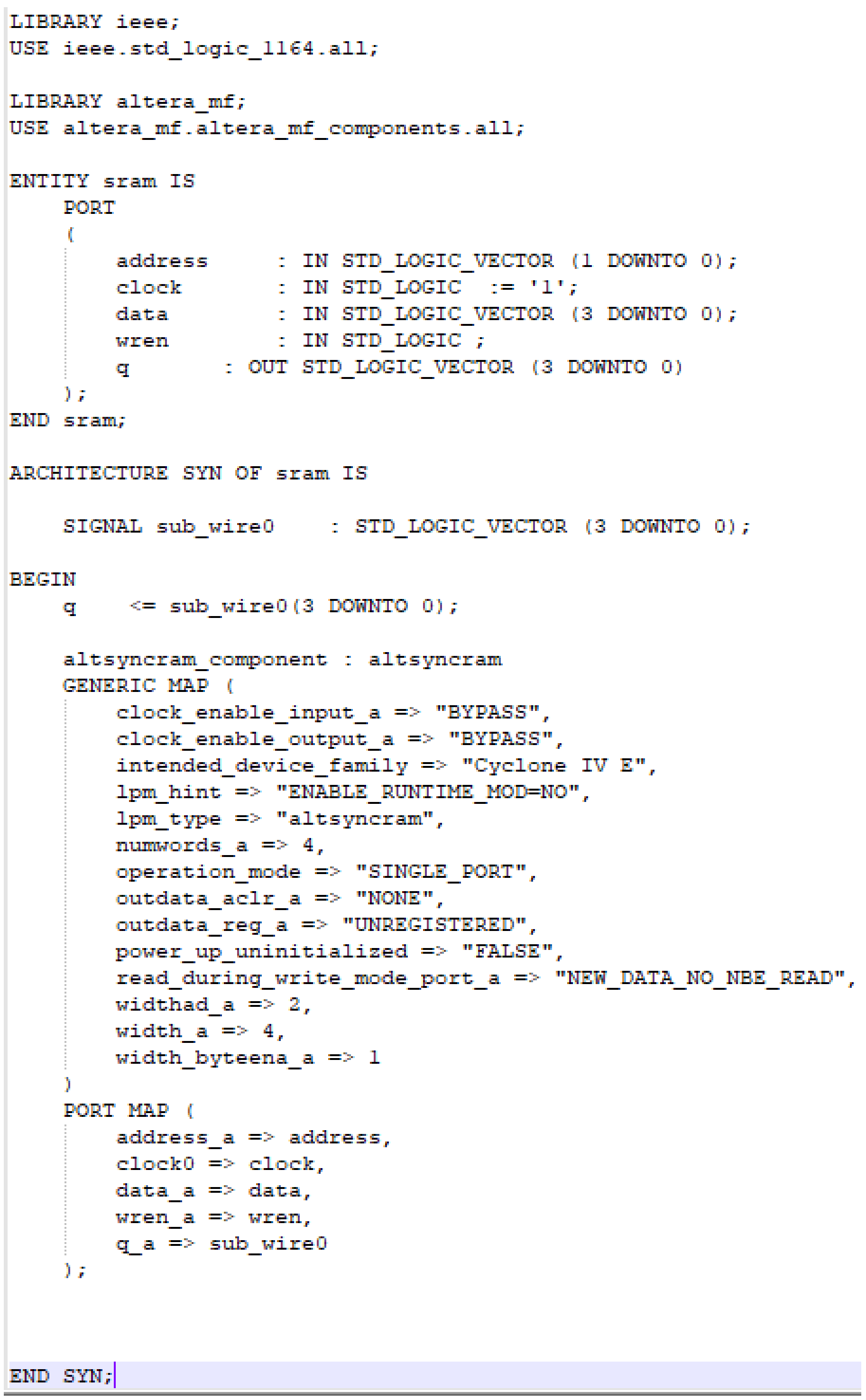
```
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;

ENTITY upcount IS
   GENERIC  (  modulus : INTEGER := 4 ) ;
   PORT  (  Resetn, Clock, E, L      : IN      STD_LOGIC ;
            R                       : IN      INTEGER RANGE 0 TO modulus-1 ;
            Q                       : BUFFER     INTEGER RANGE 0 TO modulus-1 ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
upcount: PROCESS ( Resetn, Clock, L, E )
   BEGIN
      IF Resetn = '0' THEN
         Q <= 0 ;
      ELSIF (Clock'EVENT AND Clock = '1') THEN
         IF E = '1' THEN
            IF L = '1' THEN
               Q <= R ;
            ELSE
               Q <= Q + 1 ;
            END IF ;
         END IF ;
      END IF;
   END PROCESS;
END Behavior ;
```

Figure 5: The two counters used.



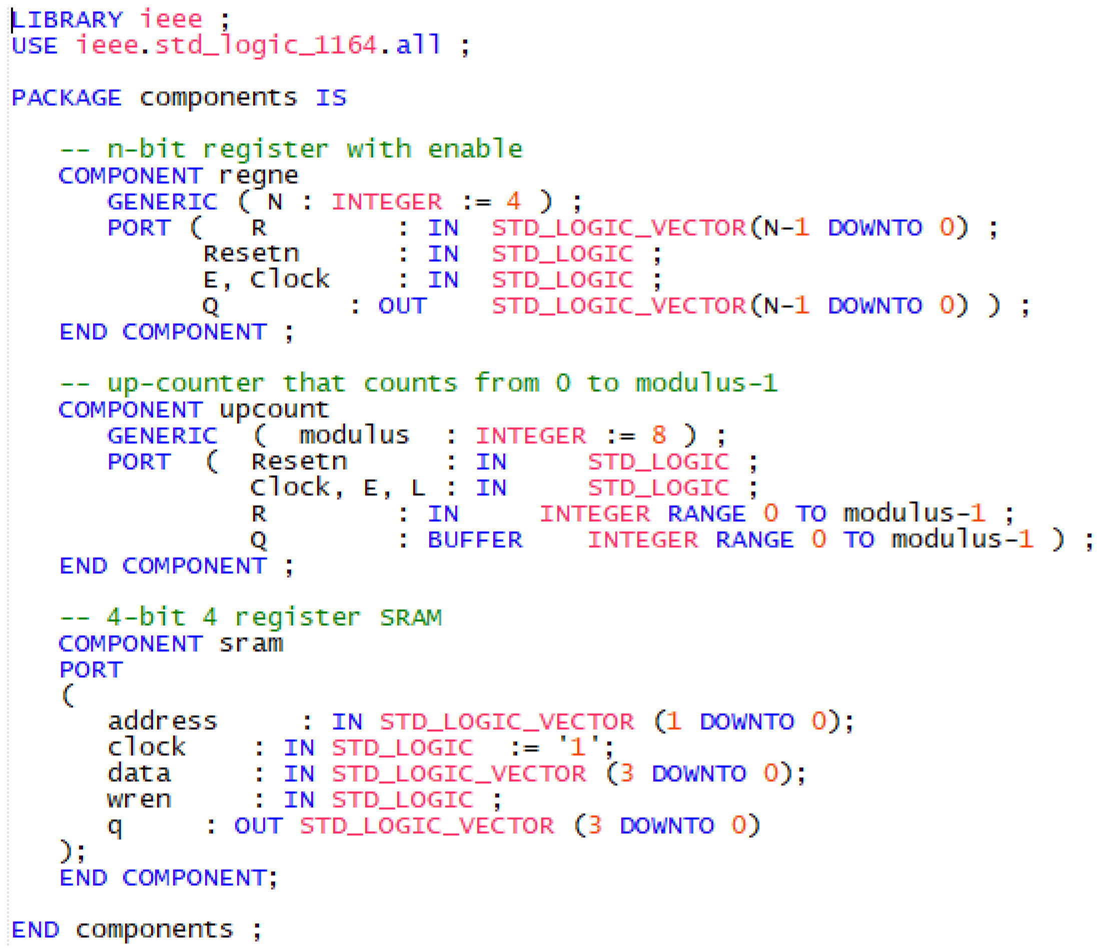
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY regne IS
   GENERIC ( N : INTEGER := 4 ) ;
   PORT (    R          : IN   STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
           Resetn       : IN   STD_LOGIC ;
           E, Clock : IN   STD_LOGIC ;
           Q          : OUT     STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regne ;

ARCHITECTURE Behavior OF regne IS
BEGIN
   PROCESS ( Resetn, Clock )
   BEGIN
      IF Resetn = '0' THEN
         Q <= (OTHERS => '0') ;
      ELSIF Clock'EVENT AND Clock = '1' THEN
         IF E = '1' THEN
            Q <= R ;
         END IF ;
      END IF ;
   END PROCESS ;
END Behavior ;
```

Figure 6: The Registers.



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY sram IS
    PORT
    (
        address     : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        clock       : IN STD_LOGIC  := '1';
        data        : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        wren        : IN STD_LOGIC ;
        q       : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
END sram;

ARCHITECTURE SYN OF sram IS

    SIGNAL sub_wire0    : STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN
    q    <= sub_wire0(3 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        intended_device_family => "Cyclone IV E",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 4,
        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",
        widthad_a => 2,
        width_a => 4,
        width_byteena_a => 1
    )
    PORT MAP (
        address_a => address,
        clock0 => clock,
        data_a => data,
        wren_a => wren,
        q_a => sub_wire0
    );



END SYN;
```

Figure 7: The SRAM.



```vhdl
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

PACKAGE components IS

   -- n-bit register with enable
   COMPONENT regne
      GENERIC ( N : INTEGER := 4 ) ;
      PORT (    R           : IN   STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
            Resetn      : IN   STD_LOGIC ;
            E, Clock    : IN   STD_LOGIC ;
            Q         : OUT    STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
   END COMPONENT ;

   -- up-counter that counts from 0 to modulus-1
   COMPONENT upcount
      GENERIC  (  modulus   : INTEGER := 8 ) ;
      PORT   (  Resetn      : IN      STD_LOGIC ;
                Clock, E, L : IN      STD_LOGIC ;
                R           : IN     INTEGER RANGE 0 TO modulus-1 ;
                Q           : BUFFER     INTEGER RANGE 0 TO modulus-1 ) ;
   END COMPONENT ;

   -- 4-bit 4 register SRAM
   COMPONENT sram
   PORT
   (
      address      : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
      clock     : IN STD_LOGIC   := '1';
      data      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      wren      : IN STD_LOGIC ;
      q      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
   );
   END COMPONENT;

END components ;
```

Figure 8: The components file.

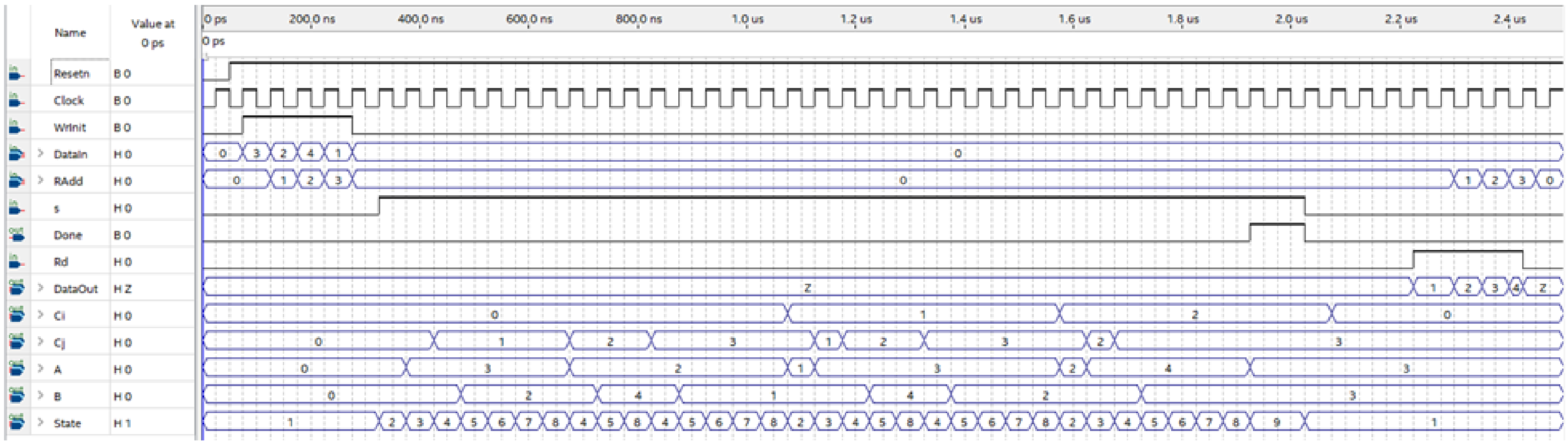


Figure 9: Waveform showing functionality of the original code.