|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий

Кафедра вычислительной техники

|  |
| --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**  **Разработка структуры данных (стек)** |
| *Разработка программно-аппаратного обеспечения информационных и автоматизированных систем* |

|  |  |  |
| --- | --- | --- |
| Отчет представлен  к рассмотрению:  Студент группы ИКМО-05-18 | « » декабря 2022 г. | Пленкин А.В. |
|  |  | (подпись и расшифровка подписи) |
|  |  |  |
| Отчет допущен  к защите: | « » декабря 2022 г. |  |
|  |  | (подпись и расшифровка подписи) |
| Руководитель  от кафедры | « » декабря 2022 г. | Унгер А.Ю. |
|  |  | (подпись и расшифровка подписи) |
|  |  |  |

Москва 2022

**Оглавление**

[Введение 3](#_Toc122291451)

[Порядок выполнения. 3](#_Toc122291452)

[Вывод 13](#_Toc122291453)

[Использованная литература 14](#_Toc122291454)

# Введение

Разработка информационных систем ведется на базе стандартных алгоритмов и структур данных. Одной из таких стандартных структур данных является стек. В этой структуре новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала. Стек (англ. stack – стопка) – это область памяти, доступ к которой осуществляется по принципу LIFO (Last Input - First Output, последним пришел – первым вышел). Это значит, что последний элемент, который вы добавили в стек, первым выйдет из него. Стек похож на стопку книг: если вы хотите взглянуть на книгу в середине стека, сперва придется убрать лежащие сверху. Стек – это список, у которого доступен один элемент (одна позиция). Этот элемент называется вершиной стека. Взять элемент можно только из вершины стека, добавить элемент можно только в вершину стека. Например, если записаны в стек числа 1, 2, 3, то при последующем извлечении получим 3,2,1.

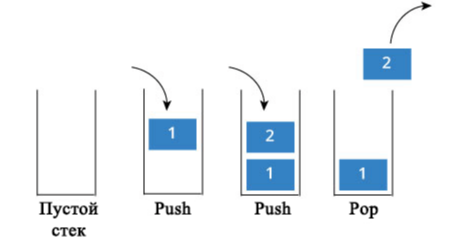


Рисунок 1 – наглядная демонстрация работы стека

**Цель работы:** разработать программно-аппаратный стек на базе программируемой логической интегральной схемы.

# Порядок выполнения.

В структуре данных стек ("LIFO") принято выделять два управляющих сигнала wr и rd для записи и чтения, соответственно. Положим в основу очереди регистровый файл, дополнив его контуром управления. Выделим два указателя: указатель для чтения и указатель для записи, которые указывают на элемент чтения и записи, соответственно. В момент чтения и записи данные указатели смещаются на одну позицию (Листинг 1).

Листинг 1 – листинг кода реализации стека

|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  entity stack\_module is    generic(B: natural := 8; W: natural := 4);    port (      clk, reset: in STD\_LOGIC;      rd, wr: in STD\_LOGIC;      w\_data: in STD\_LOGIC\_VECTOR(B-1 downto 0);      r\_data: out STD\_LOGIC\_VECTOR(B-1 downto 0);      empty, full: out STD\_LOGIC    );  end stack\_module;  architecture Behavioral of stack\_module is    type reg\_file\_type is array(2\*\*W - 1 downto 0) of STD\_LOGIC\_VECTOR(B - 1 downto 0);    signal w\_ptr\_reg, w\_ptr\_next, w\_ptr\_succ: STD\_LOGIC\_VECTOR(W - 1 downto 0);    signal r\_ptr\_reg, r\_ptr\_next, r\_ptr\_last: STD\_LOGIC\_VECTOR(W - 1 downto 0);    signal array\_reg: reg\_file\_type;    signal full\_reg, full\_next, empty\_reg, empty\_next: STD\_LOGIC;    signal min, max: STD\_LOGIC\_VECTOR(W - 1 downto 0);    signal wr\_op: STD\_LOGIC\_VECTOR(1 downto 0);    signal wr\_en: STD\_LOGIC;  begin    -- register file    process(clk, reset)    begin      if(reset = '1') then -- array nullification         array\_reg <= (others => (others => '0'));      elsif (rising\_edge(clk)) then        if(wr\_en = '1') then -- write          array\_reg(to\_integer(unsigned(w\_ptr\_reg))) <= w\_data;        end if;      end if;    end process;    r\_data <= array\_reg(to\_integer(unsigned(r\_ptr\_reg))); -- read    wr\_en <= wr and (not full\_reg); -- write (only if stack is empty)    -- pointer and read/write controller    process(clk, reset)      begin      if(reset ='1') then -- stack nullification        w\_ptr\_reg <= (others => '0'); -- write pointer reset        r\_ptr\_reg <= (others => '0'); -- read pointer reset        full\_reg <= '0';        empty\_reg <= '1'; -- empty flag true      elsif(rising\_edge(clk)) then        w\_ptr\_reg <= w\_ptr\_next; -- write pointer increment        r\_ptr\_reg <= r\_ptr\_next;        full\_reg <= full\_next;        empty\_reg <= empty\_next;      end if;    end process;    -- increment write pointer if writing    w\_ptr\_succ <= STD\_LOGIC\_VECTOR(unsigned(w\_ptr\_reg)+ 1);    -- decrement write pointer if reading    r\_ptr\_last <= STD\_LOGIC\_VECTOR(unsigned(w\_ptr\_reg)- 1);    min <= (others => '0');    max <= (others => '1');    -- combine read/write flags into register    wr\_op <= wr & rd;    process(w\_ptr\_reg, w\_ptr\_succ, r\_ptr\_reg, wr\_op, full\_reg, empty\_reg, r\_ptr\_last)    begin      w\_ptr\_next <= w\_ptr\_reg; -- default value      r\_ptr\_next <= r\_ptr\_reg;      full\_next <= full\_reg;      empty\_next <= empty\_reg;      case wr\_op is -- decide from read/write register state        when "00" => -- do nothing        when "01" => -- read          if(empty\_reg /= '1') then -- if stack is not empty            r\_ptr\_next <= r\_ptr\_last;            w\_ptr\_next <= r\_ptr\_reg; -- decrement read and write pointers            full\_next <= '0';            if(r\_ptr\_reg = min) then -- take element if it was last              empty\_next <= '1';            end if;          end if;        when "10" => -- write          if(full\_reg /= '1') then -- if stack is not full            w\_ptr\_next <= w\_ptr\_succ;            r\_ptr\_next <= w\_ptr\_reg;            empty\_next <= '0';            if(w\_ptr\_reg = max) then              full\_next <= '1';            end if;          end if;        when others => -- do nothing          w\_ptr\_next <= w\_ptr\_reg;          r\_ptr\_next <= r\_ptr\_reg;      end case;    end process;     -- output    full <= full\_reg;    empty <= empty\_reg;  end Behavioral; |

После разработки стека, необходимо проверить правильность выполнения. Для этого разработаем серию тестов.

В первом тесте заполним стек, затем прочитаем все элементы, опустошив его и после этого, запишем в стек несколько элементов и прочитаем их. Мы изменяем только процесс симуляции (Листинг 2).

Листинг 2 – листинг кода Test Bench 1

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY l3\_stack\_tb IS  END l3\_stack\_tb;  ARCHITECTURE behavior OF l3\_stack\_tb IS      COMPONENT stack\_module      PORT(           clk : IN  std\_logic;           reset : IN  std\_logic;           rd : IN  std\_logic;           wr : IN  std\_logic;           w\_data : IN  std\_logic\_vector(7 downto 0);           r\_data : OUT  std\_logic\_vector(7 downto 0);           empty : OUT  std\_logic;           full : OUT  std\_logic          );      END COMPONENT;     --Inputs     signal clk : std\_logic := '0';     signal reset : std\_logic := '0';     signal rd : std\_logic := '0';     signal wr : std\_logic := '0';     signal w\_data : std\_logic\_vector(7 downto 0) := (others => '0');    --Outputs     signal r\_data : std\_logic\_vector(7 downto 0);     signal empty : std\_logic;     signal full : std\_logic;     -- Clock period definitions     constant clk\_period : time := 10 ns;  BEGIN    -- Instantiate the Unit Under Test (UUT)     uut: stack\_module PORT MAP (            clk => clk,            reset => reset,            rd => rd,            wr => wr,            w\_data => w\_data,            r\_data => r\_data,            empty => empty,            full => full          );     -- Clock process definitions     clk\_process :process     begin      clk <= '0';      wait for clk\_period/2;      clk <= '1';      wait for clk\_period/2;     end process;    -- Stimulus process    stim\_proc: process    begin      reset <= '1';      wait until falling\_edge(clk);      reset <= '0';      wait until falling\_edge(clk);        wr <= '1';        w\_data <= "00000001";      wait until falling\_edge(clk);        w\_data <= "00000010";      wait until falling\_edge(clk);        w\_data <= "00000011";      wait until falling\_edge(clk);        w\_data <= "00000100";      wait until falling\_edge(clk);        w\_data <= "00000101";      wait until falling\_edge(clk);        wr <= '0';        rd <= '1';        for i in 1 to 6 loop          wait until falling\_edge(clk);        end loop;      wait until falling\_edge(clk);        rd <= '0';        wr <= '1';        w\_data <= "00000110";      wait until falling\_edge(clk);        w\_data <= "00000111";      wait until falling\_edge(clk);        w\_data <= "00000000";      wait until falling\_edge(clk);        wr <= '0';        rd <= '1';        for i in 1 to 10 loop          wait until falling\_edge(clk);        end loop;      assert false report "Simulation completed" severity failure;      end process;  END; |

Результат симуляции представлен на рисунках 1-4:

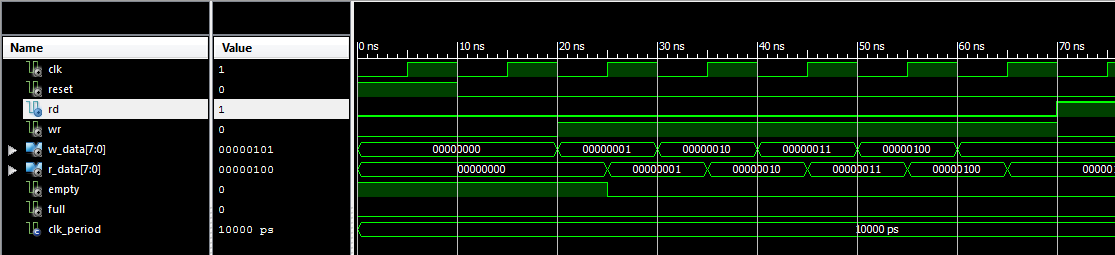


Рисунок 1 – тест 1, фаза записи 1

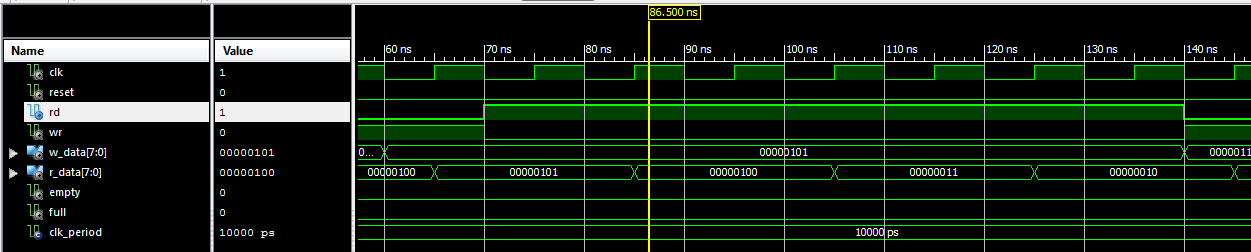


Рисунок 2 – тест 1, фаза чтения 1

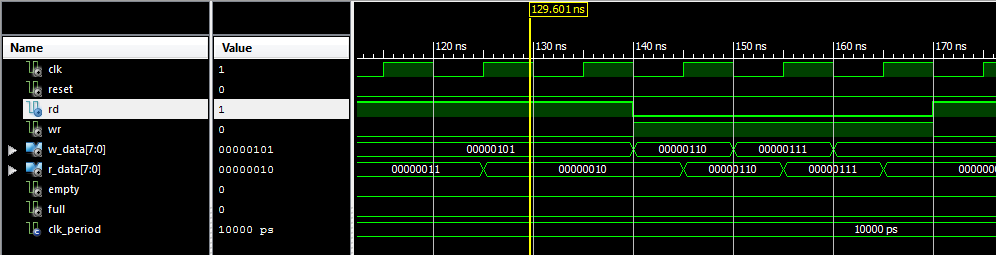


Рисунок 3 – тест 1, фаза записи 2

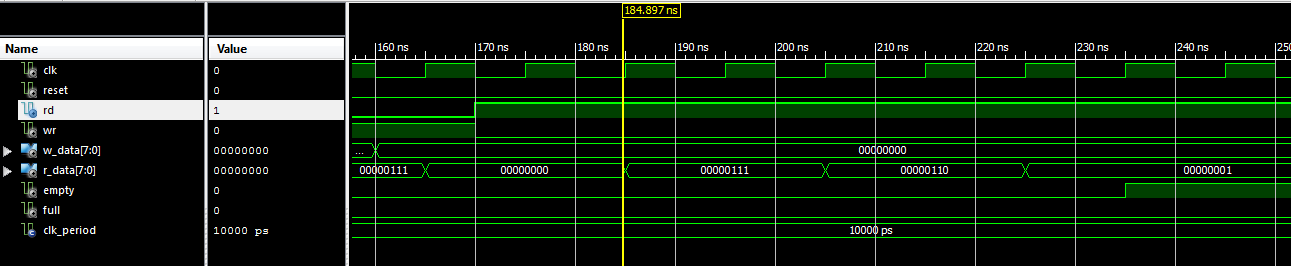


Рисунок 4 – тест 1, фаза чтения 2

Во втором тесте будем чередовать фазы чтения и записи, а в последствии произведем чтение в цикле, пока стек не окажется пустым. Код для Test Bench 2 представлен в листинге 2.

Листинг 2 – листинг кода Test Bench 2

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY l3\_stack\_tb2 IS  END l3\_stack\_tb2;  ARCHITECTURE behavior OF l3\_stack\_tb2 IS      COMPONENT stack\_module      PORT(           clk : IN  std\_logic;           reset : IN  std\_logic;           rd : IN  std\_logic;           wr : IN  std\_logic;           w\_data : IN  std\_logic\_vector(7 downto 0);           r\_data : OUT  std\_logic\_vector(7 downto 0);           empty : OUT  std\_logic;           full : OUT  std\_logic          );      END COMPONENT;     --Inputs     signal clk : std\_logic := '0';     signal reset : std\_logic := '0';     signal rd : std\_logic := '0';     signal wr : std\_logic := '0';     signal w\_data : std\_logic\_vector(7 downto 0) := (others => '0');    --Outputs     signal r\_data : std\_logic\_vector(7 downto 0);     signal empty : std\_logic;     signal full : std\_logic;     -- Clock period definitions     constant clk\_period : time := 10 ns;    BEGIN    -- Instantiate the Unit Under Test (UUT)     uut: stack\_module PORT MAP (            clk => clk,            reset => reset,            rd => rd,            wr => wr,            w\_data => w\_data,            r\_data => r\_data,            empty => empty,            full => full          );     -- Clock process definitions     clk\_process :process     begin      clk <= '0';      wait for clk\_period/2;      clk <= '1';      wait for clk\_period/2;     end process;     -- Stimulus process     stim\_proc: process     begin      reset <= '1';      wait until falling\_edge(clk);      reset <= '0';        wait until falling\_edge(clk);      wr <= '1';      w\_data <= "00000001";      wait until falling\_edge(clk);      w\_data <= "00000010";      wait until falling\_edge(clk);      w\_data <= "00000011";      wait until falling\_edge(clk);      w\_data <= "00000100";      wait until falling\_edge(clk);      w\_data <= "00000101";      wait until falling\_edge(clk);      wr <= '0';      rd <= '1';      for i in 1 to 4 loop        wait until falling\_edge(clk);      end loop;      wait until falling\_edge(clk);      rd <= '0';      wr <= '1';      w\_data <= "00000111";      wait until falling\_edge(clk);      w\_data <= "00001000";      wait until falling\_edge(clk);      w\_data <= "00001001";      wait until falling\_edge(clk);      w\_data <= "00001010";      wait until falling\_edge(clk);      w\_data <= "00001011";      wait until falling\_edge(clk);      wr <= '0';      rd <= '1';      for i in 1 to 4 loop        wait until falling\_edge(clk);      end loop;      wait until falling\_edge(clk);      rd <= '0';      wr <= '1';      w\_data <= "00001100";      wait until falling\_edge(clk);      w\_data <= "00001101";      wait until falling\_edge(clk);      w\_data <= "00001110";      wait until falling\_edge(clk);      w\_data <= "00001111";      wait until falling\_edge(clk);      wr <= '0';      rd <= '1';      for i in 1 to 20 loop        wait until falling\_edge(clk);      end loop;      assert false report "Simulation completed" severity failure;        wait;     end process;  END; |

Результат симуляции представлен на рисунках 5-8:

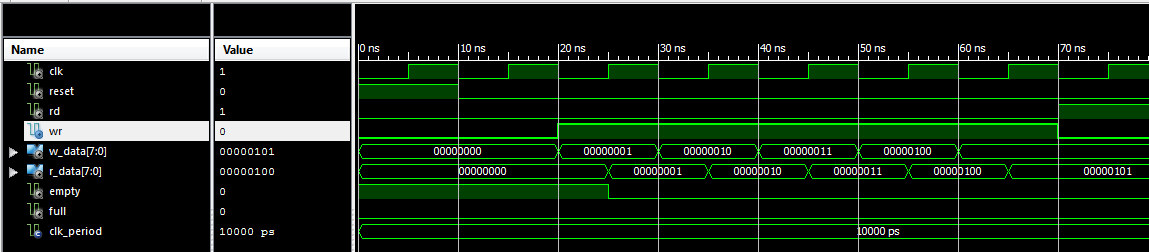


Рисунок 5 – тест 2 фаза записи 1

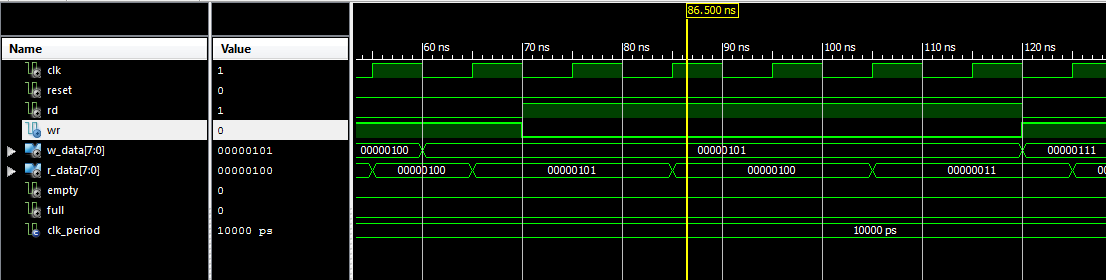


Рисунок 6 – тест 2 фаза чтения 1

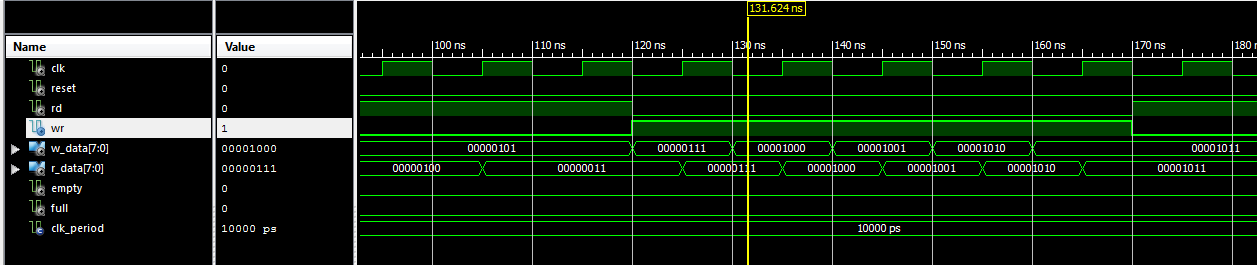


Рисунок 7 – тест 2 фаза записи 2

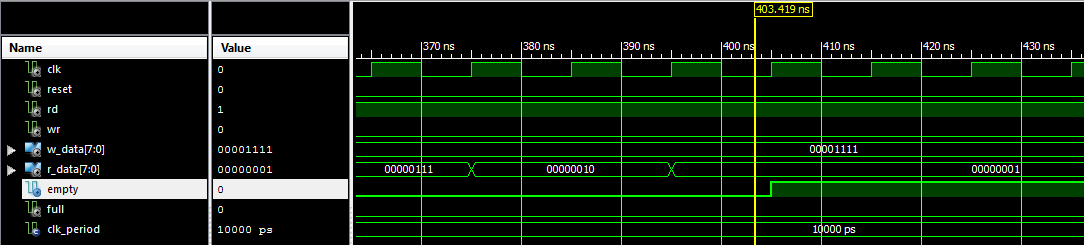


Рисунок 8 – тест 2 фаза чтения до опустошения стека

# Вывод

На базе программируемой логической интегральной схемы (ПЛИС) была спроектирована структура данных стек. Работоспособность разработанной схемы была проверена с помощью серии тестов.

# Использованная литература

1. Потехин Д. С., Тарасов И. Е. Разработка программно-аппаратного обеспечения информационных и автоматизированных систем [Электронный ресурс]:конспект лекций. – М.: РТУ МИРЭА, 2020.

2. Строгонов А. В. Цифровая обработка сигналов в базисе программируемых логических интегральных схем [Электронный ресурс]:. - Санкт-Петербург: Лань, 2021.

3. Зайцев Е. И., Халабия Р. Ф. Организация ЭВМ и систем:учебное пособие. - М.: РТУ МИРЭА, 2019.

4. Певцов Е. Ф., Тарасов И. Е., Миннебаев В. М. Автоматизированное проектирование цифровых схем [Электронный ресурс]:учебное пособие. - М.: МИРЭА, 2016.

5. Красников А. К., Волосенков В. О., Никончук Я. С. Программирование микропроцессорных систем [Электронный ресурс]:учебное пособие. - М.: МГТУ МИРЭА, 2014.

6. Деменкова Т. А., Яровов Д. Д. Аппаратная реализация алгоритмов на FPGA [Электронный ресурс]:монография. - М.: МИРЭА, 2016.