Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Московский институт электроники и математики им. А.Н.Тихонова национального Исследовательского университета «Высшая школа экономики»
Департамент компьютерной инжерении

Курс «Высокоуровневое и имитационное моделирование цифровых систем»

ОТЧЕТ

о выполнении практической работы № 4

тема работы: **«Высокоуровневое моделирование аппаратных проектов.Верификация HDL проектов. DPI, PLI/VPI»**

Выполнили:

Студент Лоренс А.С. БИВ-186

Студент Петухов Г.А. БИВ-185

_____
Подпись

Принял:

Американов А.А.

_____
Подпись

Москва 2021г.

1. Задания

Часть 1.

1. Ознакомиться с концепцией использования интерфейса **VPI/PLI** для верификации: https://marsohod.org/11-blog/266-verilog-vpi

Ознакомиться с концепцией использования интерфейса **DPI** для верификации: https://www.doulos.com/knowhow/sysverilog/tutorial/dpi/

а также с доп. материалами к работе в папке **HLIMDS_Lab4_Dop**

https://drive.google.com/drive/folders/1TvfSk9LT8MJA082PYpiyCwjRHhA6CrED?usp=sharing

2. Для выполнения работы лучше установить не урезанную (не **Altera edition**) версию **Modelsim** или новую версию данного средства симуляции **QuestaSim** (для данного мануала используется **QuestaSIM_10_4c_x64**).

3. Перейти в папку **examples** вашего симулятора (архив с **examples** содержится в доп. материалах к работе в папке **HLIMDS_Lab4_Dop**). Проанализировать примеры использования **DPI** и **PLI/VPI** в различных папках. Выполнить примеры (использовать **shell**). Основное внимание уделить папкам: **c_windows**, **systemverilog**, **tutorials\systemverilog**, **verilog**.

4. Выполнить пример **examples\tutorials\systemverilog\dpi_basic**, прочитать **README** с пояснениями. Объяснить (написать в отчете ответ!), почему на вейвформе сперва идет Красный цвет, а потом такая последовательность смены цветов:

5. Ознакомиться с примером **examples\systemverilog\dpi\dpivpiperf**. Отразить в отчете, чем отличаются подходы **DPI** и **VPI/PLI.** Какой подход лучше и почему? Почему **VPI/PLI** требует больше кода?

6. Выполнить пример **examples\c_windows\dpi (**или похожий пример **examples\systemverilog\dpi\simple_calls)**. Привести файл результатов, вейвформы, коды с комментариями и пояснениями, что в них делается.

7. Выполнить пример **examples\systemverilog\dpi\openarray**. Привести файл результатов, вейвформы, файлы с кодами с комментариями и пояснениями, что в них делается. Привести (в отчете!) свой пример кода, где используется данный прием.

8. Ознакомиться с примером **examples\systemverilog\dpi\packed_types**. Дать пояснения, для чего в **SystemVerilog** используются упакованные типы и какие особенности их передачи через **DPI**. Привести свой пример кода, использующий данный прием.

Выполнить пример **examples\systemverilog\dpi\unpacked_types**. Проанализировать, чем он отличается от предыдущего. Привести свой пример кода, использующий данный прием.

9. Ознакомиться с примером **examples\systemverilog\dpi\checkpoint**, привести пояснения, для чего может пригодиться прием, описанный в примере.

10. Ознакомиться с примером **examples\systemverilog\dpi\cpackages** вызова команд **TCL Modelsim** из кода **SystemVerilog**. Дать пояснения, для чего может пригодиться данный прием. Привести свой пример кода, где используется данный прием.

11. Ознакомиться с примером **examples\systemverilog\dpi\create_sv_dynarray** создания динамических массивов. Дать пояснения, для чего может пригодиться данный прием. Привести свой пример кода, где используется данный прием.

Часть 2

1. Выполнить пример **examples\systemverilog\dpi\mux81**. Привести файл результатов, вейвформы, файлы с кодами с комментариями и пояснениями, что в них делается. Проверить ситуацию, когда входной сигнал не определен (**‘bx**, **‘bz**).

Сымитировать ошибку кода на Verilog и продемонстрировать, как на это реагирует тест.

Реализовать модуль 12-ти разрядного шифратора.

2. Выполнить пример **examples\systemverilog\dpi\fibonacci**. Привести файл результатов, вейвформы, файлы с кодами с комментариями и пояснениями, что в них делается.

Сымитировать ошибку кода на Verilog и продемонстрировать, как на это реагирует тест.

3. Коды программ загрузить в репозитарий github. Оформить документацию (можно на основе отчета по ЛР). В отчете привести ссылку на репозитарий.

2. Выполнение

Часть 1

Задание 1.2

Для работы нами было установлено средство симуляции Questasim_10_4c_x64. Для установки данного средства симуляции потребовалось дополнительно к действиям, указанным в методичке: скачать набор компиляторов gcc, Поднять приоритет приоритет questasim, по отношению к приоритету modelsim в переменной path, добавить переменную среды QUESTA_HOME, добавить путь к месту установки набора компиляторов gcc в переменную path.

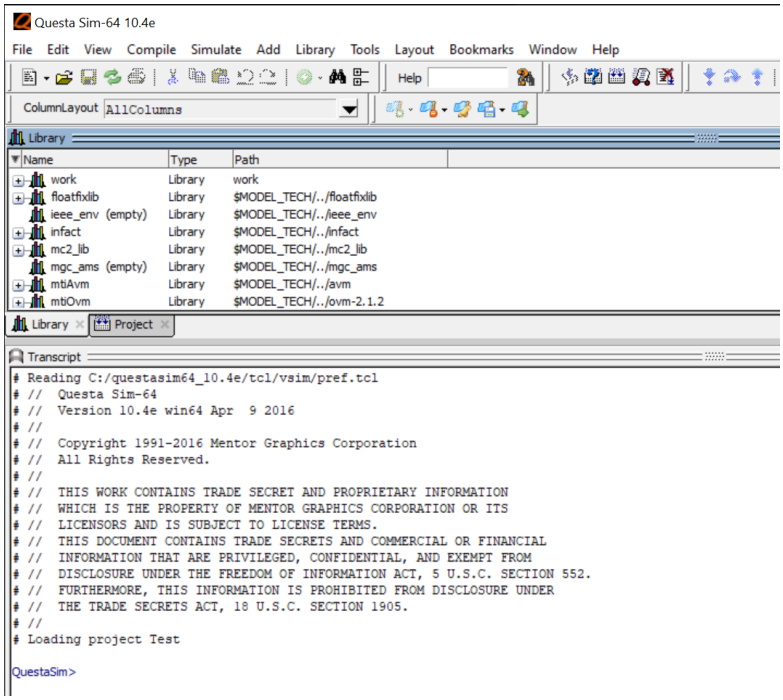

Рис. 1. Главное окно программы QuestaSim после удачной установки.

Задание 1.4

Выполнить пример **examples\tutorials\systemverilog\dpi_basic**, прочитать **README** с пояснениями. Объяснить почему на вейвформе сперва идет Красный цвет, а потом такая последовательность смены цветов:

Рис. 2. Результат симуляции примера dpi_basic в среде QuestaSim.

В данном примере мы можем видеть такую вейвформу, потому что выполнение программы начинается с функции, которая изначально ожидает ввода красного цвета, и только потом начинает работу, и это логично, с учетом того, что данная программа является сильно упрощенной симуляцией работы светофора. Поэтому после красного света идет желтый, потом зеленный, и так далее.

Задание 1.5

Ознакомиться с примером **examples\systemverilog\dpi\dpivpiperf**. Отразить в отчете, чем отличаются подходы **DPI** и **VPI/PLI.** Какой подход лучше и почему? Почему **VPI/PLI** требует больше кода?

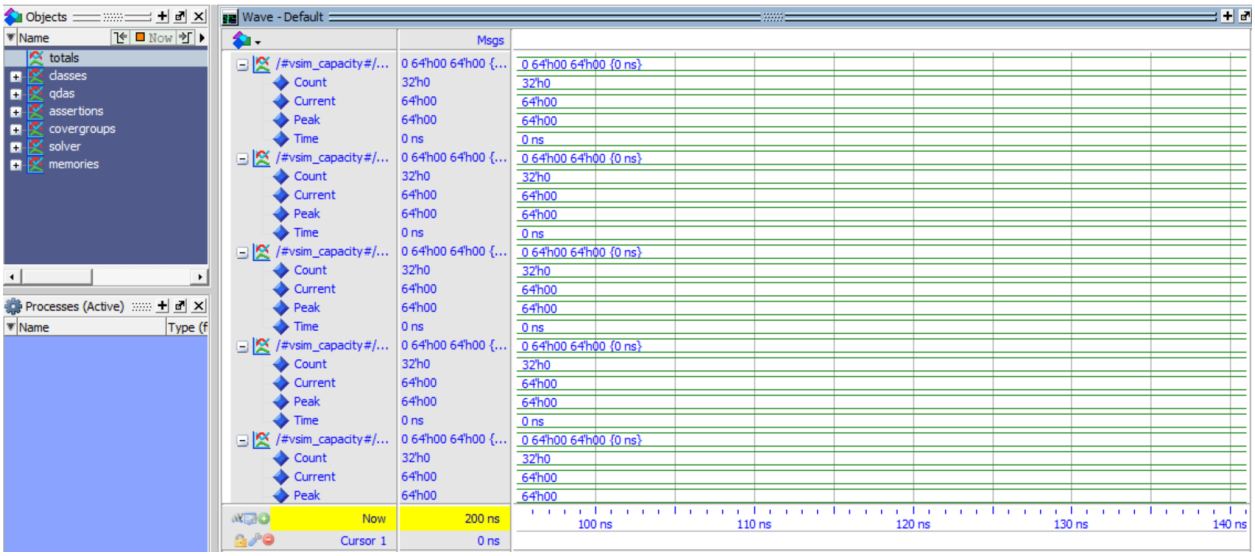

Рис. 3. Результат симуляции примера dpivpiperf в среде QuestaSim.

Принципиальное различие между этими подходами VPI/PLI и DPI состоит в техническом исполнении одной и той же задачи, результат работы двух этих подходов, как правило одинаковый, по этой причине приведен только 1 результат симуляции, так как 2 полностью идентичен с ним. Подход VPI/PLI состоит в расширении языка Verilog, за счет добавления возможности передавать данные из программы, написанной на C, хотя по большому счету пользователь создает себе дополнительную библиотеку на языке C для языка Verilog, и ему надо указывать все состояния и значения переменных. Именно по этой причине подход VPI/PLI требует намного больше кода, чем в случае использования подхода dpi. Который в свою очередь представляет из себя по большому счету интерфейс между языком C и Verilog, в котором пользователь только указывает название функции и данные поступающие или исходящие из этой функции.

Задание 1.6

Выполнить пример **examples\c_windows\dpi (**или похожий пример **examples\systemverilog\dpi\simple_calls)**. Привести файл результатов, вейвформы, коды с комментариями и пояснениями, что в них делается.

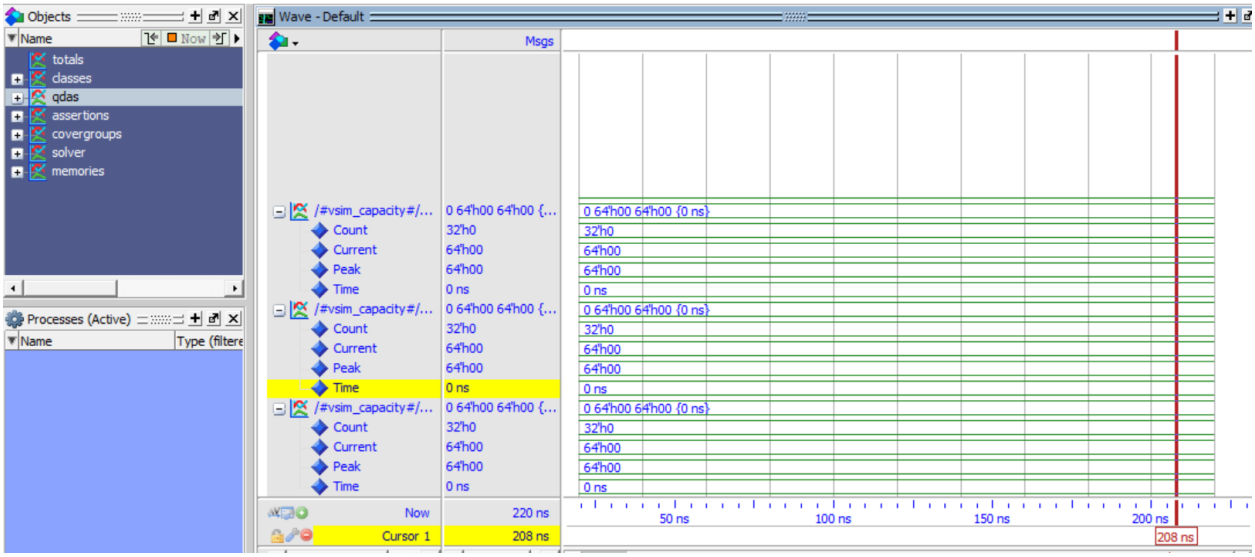

Рис. 4. Результат симуляции примера dpi_examples в среде QuestaSim.

```
 In CTask
    In SVTask
       In CFunction
          In SVFunction
       In CFunction
          In SVFunction
 0: vi1=0002 vi2=0003 vo3=0008 vo4=000c vb5=0006
 In CTask
    In SVTask
       In CFunction
          In SVFunction
       In CFunction
          In SVFunction
 5: vi1=0008 vi2=000c vo3=0014 vo4=0030 vb5=0008
 In CTask
    In SVTask
       In CFunction
          In SVFunction
       In CFunction
          In SVFunction
10: vi1=0014 vi2=0030 vo3=002c vo4=00c0 vb5=000a
 In CTask
    In SVTask
       In CFunction
          In SVFunction
       In CFunction
          In SVFunction
15: vi1=002c vi2=00c0 vo3=005c vo4=0300 vb5=000c
```

Рис. 5. Листинг файла результатов выполнения данного примера.

Задание 1.7

Выполнить пример **examples\systemverilog\dpi\openarray**. Привести файл результатов, вейвформы, файлы с кодами с комментариями и пояснениями, что в них делается. Привести (в отчете!) свой пример кода, где используется данный прием.

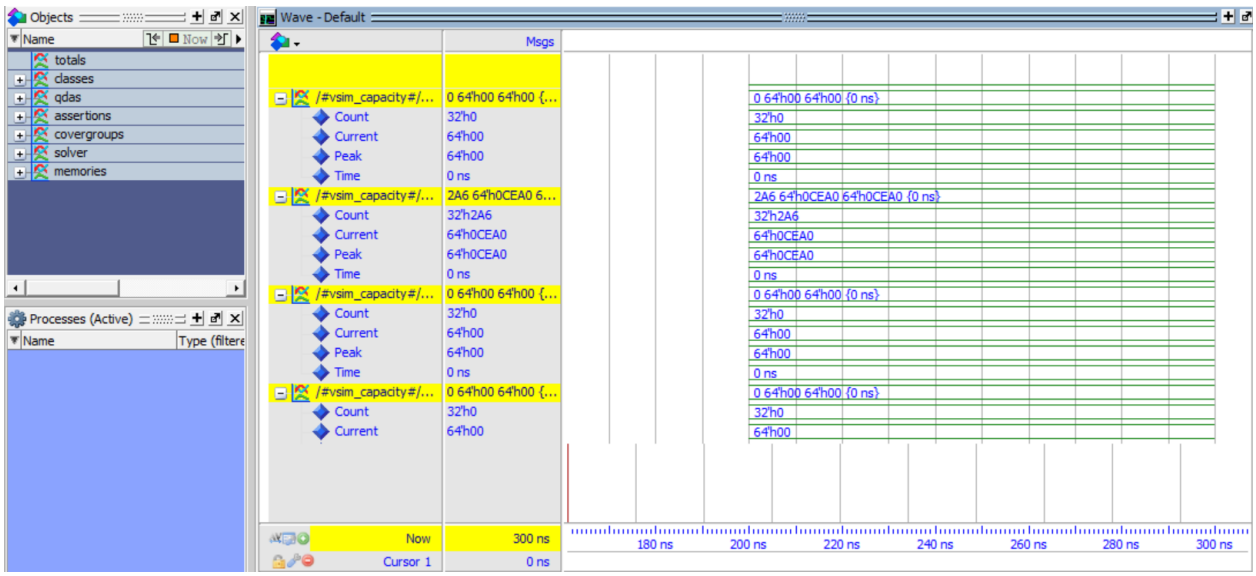

Рис. 6. Результат симуляции примера openarray в среде QuestaSim.

```
# clone_struct_oarr: dim:1, incr:-1, size:10, left:11, right:20, low:11, high:20
# clone_struct_oarr: '{'{x:1, y:9}, '{x:2, y:8}, '{x:3, y:7}, '{x:4, y:6}, '{x:5, y:5}, '{x:6,
y:4}, '{x:7, y:3}, '{x:8, y:2}, '{x:9, y:1}, '{x:0, y:10}}
# clone_struct_oarr_memcpy: '{'{x:1, y:9}, '{x:2, y:8}, '{x:3, y:7}, '{x:4, y:6}, '{x:5, y:5},
'{x:6, y:4}, '{x:7, y:3}, '{x:8, y:2}, '{x:9, y:1}, '{x:0, y:10}}
# clone_struct_oarr_safe: '{'{x:1, y:9}, '{x:2, y:8}, '{x:3, y:7}, '{x:4, y:6}, '{x:5, y:5}, '{x:6,
y:4}, '{x:7, y:3}, '{x:8, y:2}, '{x:9, y:1}, '{x:0, y:10}}
# clone_bitvec_oarr_by_elemptr: dim:1, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_bitvec_oarr_by_elemptr: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_bitvec_oarr_by_elemval: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_bitvec_2D_oarr_by_elemptr: dim:2, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_bitvec_2D_oarr_by_elemptr: dim:1, incr:1, size:8, left:7, right:0, low:0, high:7
# clone_bitvec_2D_oarr_by_elemptr: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_bitvec_2D_oarr_by_elemval: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_bitvec_2D_oarr_by_elemptr: dim:3, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_bitvec_2D_oarr_by_elemptr: dim:2, incr:1, size:8, left:7, right:0, low:0, high:7
# clone_bitvec_2D_oarr_by_elemptr: dim:1, incr:1, size:12, left:11, right:0, low:0, high:11
# clone_bitvec_3D_oarr_by_elemptr: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_bitvec_3D_oarr_by_elemval: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_logicvec_oarr_by_elemptr: dim:1, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_logicvec_oarr_by_elemptr: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_logicvec_oarr_by_elemval: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_logicvec_2D_oarr_by_elemptr: dim:2, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_logicvec_2D_oarr_by_elemptr: dim:1, incr:1, size:8, left:7, right:0, low:0, high:7
# clone_logicvec_2D_oarr_by_elemptr: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_logicvec_2D_oarr_by_elemval: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_logicvec_3D_oarr_by_elemptr: dim:3, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_logicvec_3D_oarr_by_elemptr: dim:2, incr:1, size:8, left:7, right:0, low:0, high:7
# clone_logicvec_3D_oarr_by_elemptr: dim:1, incr:1, size:12, left:11, right:0, low:0, high:11 #
clone_logicvec_3D_oarr_by_elemptr: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_logicvec_3D_oarr_by_elemval: '{8, 7, 6, 5, 4, 3}, expecting '{8, 7, 6, 5, 4, 3}
# clone_bit_oarr: dim:1, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_bit_oarr: '{1, 0, 1, 0, 1, 0}, expecting '{1, 0, 1, 0, 1, 0}
# clone_bit_2D_oarr: '{1, 0, 1, 0, 1, 0}, expecting '{1, 0, 1, 0, 1, 0}
# clone_bit_3D_oarr: '{1, 0, 1, 0, 1, 0}, expecting '{1, 0, 1, 0, 1, 0}
# clone_logic_oarr: dim:1, incr:1, size:6, left:5, right:0, low:0, high:5
# clone_logic_oarr: '{0, 1, 0, 1, 0, 1}, expecting '{0, 1, 0, 1, 0, 1}
# clone_logic_2D_oarr: '{0, 1, 0, 1, 0, 1}, expecting '{0, 1, 0, 1, 0, 1}
# clone_logic_3D_oarr: '{0, 1, 0, 1, 0, 1}, expecting '{0, 1, 0, 1, 0, 1}
```

Рис. 7. Листинг файла результатов выполнения данного примера.

В данном примере происходит работа с открытыми массивами. При этом происходит работа с размерными данными и данными различных типов.

Задание 1.8

Ознакомиться с примером **examples\systemverilog\dpi\packed_types**. Дать пояснения, для чего в **SystemVerilog** используются упакованные типы и какие особенности их передачи через **DPI**. Привести свой пример кода, использующий данный прием.

Выполнить пример **examples\systemverilog\dpi\unpacked_types**. Проанализировать, чем он отличается от предыдущего. Привести свой пример кода, использующий данный прием.

В SystemVerilog используются упакованные типы данных так как можно разбить переменные отдельной структуры, представленной в виде вектора на отдельные поля, к которым можно получить доступ в качестве элементов и которые упакованы вместе в памяти без пробелов. Первый член в структуре является наиболее значимым, а последующие члены следуют в порядке убывания значимости. Различия между двумя данными примерами состоит в том, что в первом случае данные упаковываются в памяти последовательно, а во втором случае нет, по этой причине при передаче через dpi в случае упакованного типа данных нам будет удобней передавать данные, так как в памяти они лежат по очереди.

Рис. 8. Результат симуляции примера unpacked_types в среде QuestaSim.

```
4-state vector answer 0 => 0000000000000000
2-state vector answer 0 => 0000000000000000
2-state vector answer 0 => 0000000000000000
       longint answer 0 => 0000000000000000

4-state vector answer 1 => 0000000000000000
2-state vector answer 1 => 0000000000000000
2-state vector answer 1 => 0000000000000000
       longint answer 1 => 0000000000000000

4-state vector answer 2 => 0000ffff0000ffff
2-state vector answer 2 => 0000ffff0000ffff
2-state vector answer 2 => 0000ffff0000ffff
       longint answer 2 => 0000ffff0000ffff
```

Рис. 9. Листинг файла результатов выполнения примера unpacked_types.

Пример использования структуры с упакованным типом данных:

```
typedef struct packed {
bit [3:0] mode;
bit [2:0] cfg;
bit en;
```

```
} st_ctrl;
module tb;
st_ctrl _ctrl_reg;
initial begin
// Initialize packed structure variable
ctrl_reg = '{4'ha, 3'h5, 1};
$display ("ctrl_reg = %p", ctrl_reg);
// Change packed structure member to something else
ctrl_reg.mode = 4'h3;
$display ("ctrl_reg = %p", ctrl_reg);
// Assign a packed value to the structure variable
ctrl_reg = 8'hfa;
$display ("ctrl_reg = %p", ctrl_reg);
end
endmodule
```

Рис. 10. Листинг программы использующей упакованный тип данных.

Задание 1.9

Ознакомиться с примером **examples\systemverilog\dpi\checkpoint**, привести пояснения, для чего может пригодиться прием, описанный в примере.

Прием, описанный в данном примере может пригодиться в случаях, когда в программе может происходить сбой в присваивании значений переменным или в случае ошибок в присваивании переменных между языками программирования C и systemverilog, в таком случае откат к чекпоинту поможет программе не зависнуть, или в случае неправильного результата позволит восстановить состояние на момент применения чекпоинта.

Задание 1.10

Ознакомиться с примером **examples\systemverilog\dpi\cpackages** вызова команд **TCL Modelsim** из кода **SystemVerilog**. Дать пояснения, для чего может пригодиться данный прием. Привести свой пример кода, где используется данный прием.

Само по себе применение TCL Modelsim команд помогает довольно сильно упростить работу с симуляцией проекта, а создание файла с расширением .do может позволить производить симуляцию проекта не открывая modelsim. Такой способ не требует написания объемных тестбенчей, и оптимален, когда надо выводить короткие сообщения о работе того или иного модуля или просто строить вейвформу.

```
#Quit last sim
quit -sim

#Create the work library
vlib work

#vmap the work lib
vmap work work

#include the head files
vlog +incdir+ YOURPATH sfifo_def.v
#eg: vlog +incdir+ sfifo_def.v

# Compile the verilog files
vlog -work work sfifo.v
vlog -work work sfifo_tb.v

#Or you can merge them, but be sure to compile the called file first
vlog -work work sfifo.v sfifo_tb.v
#You can also ignore the '-work work', such as vlog sfifo.v sfifo_tb.v
```

```
      #Run simulation
      vsim sfifo_tb
      vsim -lib work sfifo_tb
      vsim work. sfifo_tb

      #Prohibit to optimize
      vsim -novopt work.sfifo_tb
      vsim -voptargs=+acc -novopt work.sfifo_tb
      #Attention: If you don't add '-novopt' to disable optimization in your commands,
there will be error when add wave to window: # (vish-4014) No objects found matching
'/sfifo_tb/*'.

      #specified time pricision is 1ns
      vsim -novopt work.sfifo_tb -t 1ns

      #Set the window types
      #Open the wave window
      view wave

      #Open the instance structure window
      view structure

      #Open the signals list window
      view signals

      #Open the source window
      view source

      #Open the list window
      view list

      #Open the variables window
      view variables

      #Open the dataflow window
      view dataflow

      #Add all signals to wave window
      add wave -r /*
      #Attention: -r should be followed by a space

      #Add the specific test signal to wave window, eg: add wave sim:/ sfifo_tb /clock
      add wave sim:/ sfifo_tb /signal
      #Attention: 'sim:' can be ignored, which means 'add wave / sfifo_tb /signal' is also
right

      #Add the tb signals to wave window
      add wave sim:/sfifo_tb /*
      add wave /sfifo_tb /*
      add wave *

      #Display in hexadecimal or binary or decimal or unsigned or others
      add wave -hex/-decimal/-unsigned/-binary *

      #format: Logic/Literal/Event/analogautomatic/analogcustom
      #add wave -noupdate -format Logic -radix decimal -color Red /sfifo_tb/sfifo1/clock
      #add wave  -radix unsigned  -color pink -format analogautomatic sim:
sfifo_tb/sfifo1/clock
      add wave  -unsigned sim: sfifo_tb/sfifo1/clock
      add wave -noupdate -color pink -format Analog-Step -height 74 -max 4094.9999999999995
-radix unsigned sfifo_tb/sfifo1/clock

      #Delete signal
      delete  wave  /test/i

      #Restart simulation, then click OK or press ENTER
      restart

      #Run simulation
      run 100 us
      run -all

      #quit -f/-force to quit Modelsim
      #quit -sim to quit simulation
      quit -f/-force/-sim

      #Clear transcript window
      .main clear
```

Рис. 11. Листинг примера программы в котором используются TCL команды.

Задание 1.11

Ознакомиться с примером **examples\systemverilog\dpi\create_sv_dynarray** создания динамических массивов. Дать пояснения, для чего может пригодиться данный прием. Привести свой пример кода, где используется данный прием.

Динамический массив-это одно измерение распакованного массива, размер которого может быть установлен или изменен во время выполнения. Динамический массив объявляется с использованием подстрочного индекса пустого слова [ ].

Пространство для динамического массива не существует до тех пор, пока массив не будет явно создан во время выполнения, пространство выделяется при вызове нового[числа]. число указывает количество выделяемого пространства/элементов.

```
module dynamic_array;
  //dynamic array declaration
  bit [7:0] d_array1[];
  int       d_array2[];

  initial begin
    $display("Before Memory Allocation");
    $display("\tSize of d_array1 %0d",d_array1.size());
    $display("\tSize of d_array2 %0d",d_array2.size());

    //memory allocation
    d_array1 = new[4];
    d_array2 = new[6];

    $display("After Memory Allocation");
    $display("\tSize of d_array1 %0d",d_array1.size());
    $display("\tSize of d_array2 %0d",d_array2.size());

    //array initialization
    d_array1 = {0,1,2,3};
    foreach(d_array2[j])  d_array2[j] = j;

    $display("--- d_array1 Values are ---");
    foreach(d_array1[i])   $display("\td_aaray1[%0d] = %0d",i, d_array1[i]);
    $display("---------------------------------");

    $display("--- d_array2 Values are ---");
    foreach(d_array2[i])   $display("\td_aaray2[%0d] = %0d",i, d_array2[i]);
    $display("---------------------------------");
  end

endmodule
```

Рис. 12. Листинг примера программы в котором используются динамические массивы.

Часть 2

Задание 2.1

Выполнить пример **examples\systemverilog\dpi\mux81**. Привести файл результатов, вейвформы, файлы с кодами с комментариями и пояснениями, что в них делается. Проверить ситуацию, когда входной сигнал не определен (**‘bx**, **‘bz**).

Сымитировать ошибку кода на Verilog и продемонстрировать, как на это реагирует тест.

Реализовать 12-ти разрядный шифратор.

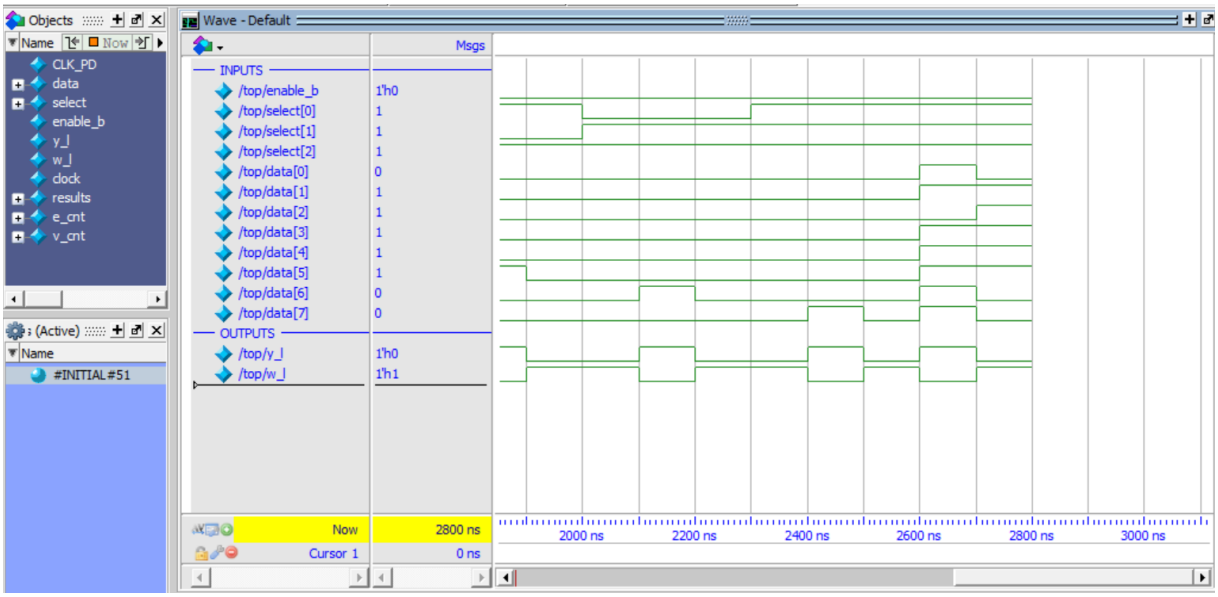

Рис. 13. Результат симуляции примера mux_81 в среде QuestaSim.

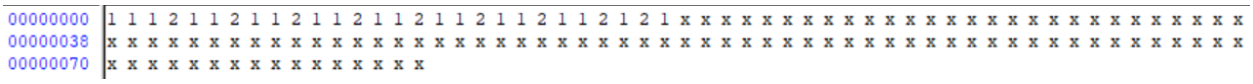

Рис. 14. Содержание переменной memory data.

Следует заметить, что содержание переменной memory data полностью схожа со значениями в файле mux81.dat, что говорит об удачной симуляции проекта.

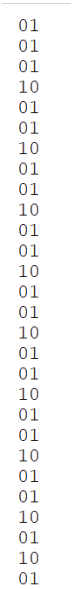

Рис. 15. Содержание файла mux81.dat.

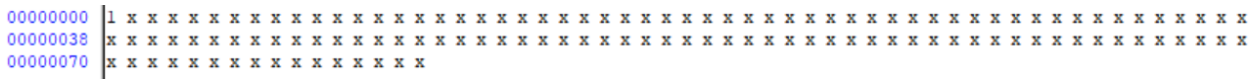

Рис. 16. . Содержание переменной memory data после изменения файла с заданными переменными.

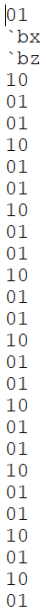

Рис. 17. Измененное содержание файла mux81.dat.

Как мы можем заметить при подаче неопределенного сигнала, все идущие после него сигналы тоже становятся неопределенными, таким образом мы выяснили что будет происходить в подобных случаях.

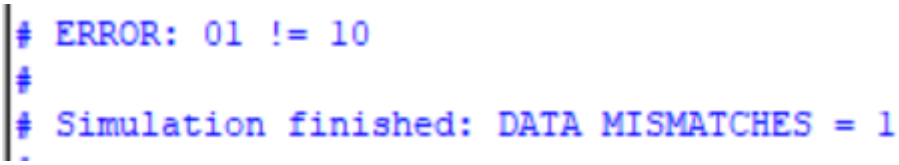

Рис. 18. Вывод ошибки в консоли QuestaSim.

В данном случае мы сымитировали ошибку в коде Verilog, на что тест на стадии верификации вполне закономерно выдает ошибку о неправильно присвоенном значении переменной.

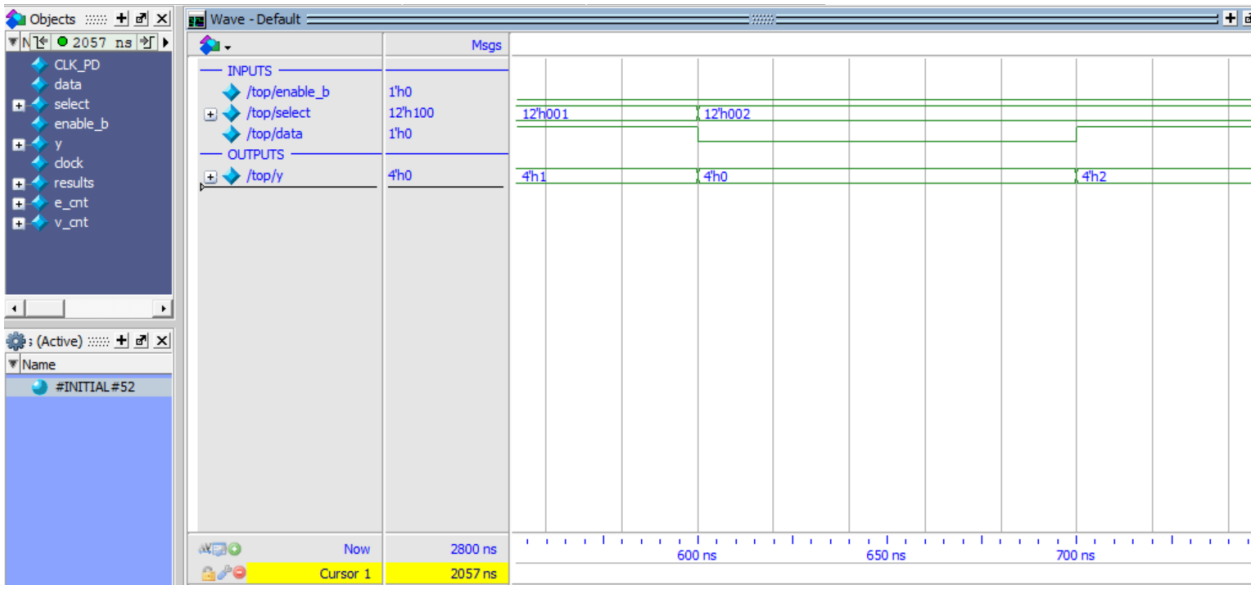

Рис. 19. Результат симуляции примера coder_12 в среде QuestaSim.

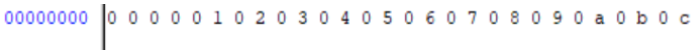

Рис. 20. Значения переменной memory data.

Задание 2.2

Выполнить пример **examples\systemverilog\dpi\fibonacci**. Привести файл результатов, вейвформы, файлы с кодами с комментариями и пояснениями, что в них делается.

Сымитировать ошибку кода на Verilog и продемонстрировать, как на это реагирует тест. Написать модуль разворачиваюсь число.

Рис. 21. Результат симуляции примера fibonacci в среде QuestaSim.

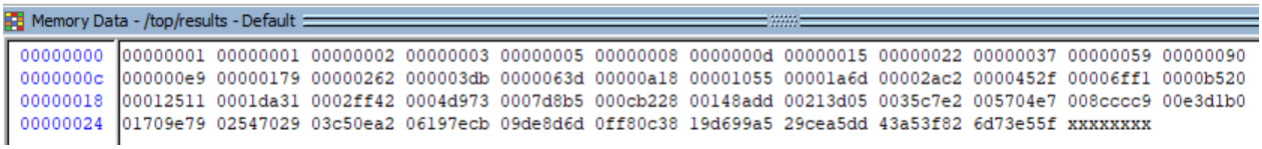

Рис. 22. Значения переменной memory data.

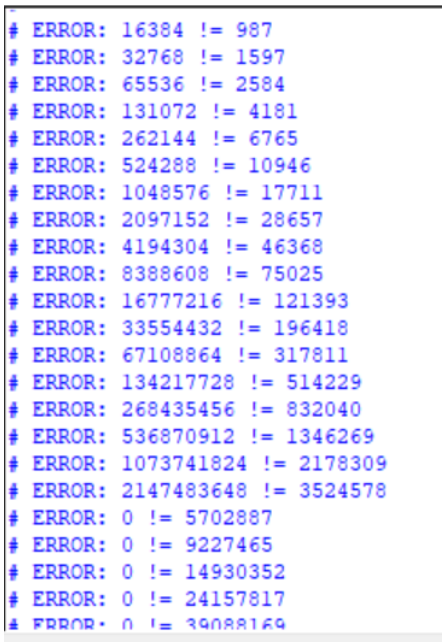

Рис. 23. Вывод ошибки в консоли QuestaSim.

Как мы можем заметить при имитировании ошибки присваивания значения в коде Verilog, на стадии верификации, в консоль выводятся ошибки о несоответсствии предполагаемого и фактического значения.

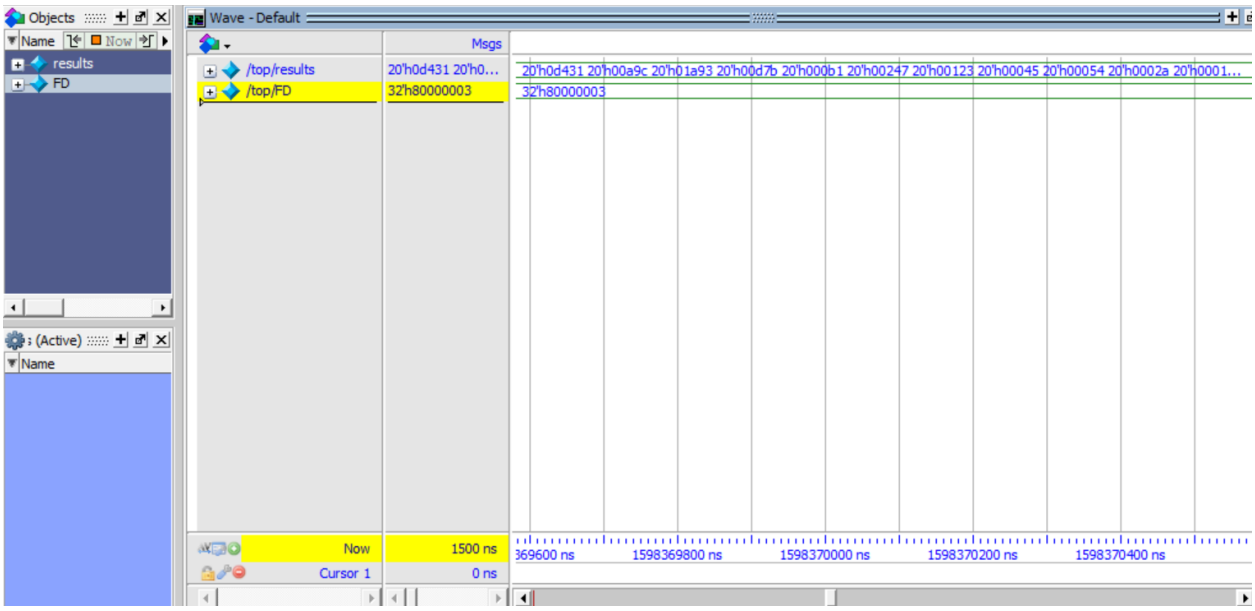

Рис. 24. Результат симуляции примера reverse в среде QuestaSim.

Данная программа переворачивает число наоборот, и выводит новое число в ячейку памяти.

Задание 2.3

Загрузить все материалы в репозитарий github (коды + *.md мануал на основе сокращенной версии отчета).

Все необходимые материалы, а именно сокращенная версия отчета и весь код данной лабораторной работы были загружены в репозиторий по данной ссылке: https://github.com/Gorg122/HLIMDS/tree/Lab_4

Выводы:

В ходе данной лабораторной работы мы научились работать с интерфейсами DPI VPI/PLI. Научились пользоваться файлами с расширением .do для симуляции проекта с помощью .bat файлов или из командной строки. Научились пользоваться TCL командами. Поработали с средством симуляции QuetsaSim, и научились настраивать его работу. Научились писать модули на языке C и встраивать эти модули в код на Verilog.