|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  |  |  | | | | | |  | |
|  |  |  | | | | | |  | |
|  |  | | | | | | | | | | | |  |
|  |  | | | | | | | | | | | |  |
|  | **Model Question Paper Answer all the Questions** | | | | | | | | | | | |  |
| **Q.No.** | | **Sub. Sec.** | **Question Description** | | | |  | **Marks** | **Unit No.** | **Level** | **Hot?** | | |
| 1. | |  | **Write a PERL subroutine to convert a decimal number to binary equivalent.**  **Answer:**  sub dec\_to\_bin {  $decimal = @\_[0];  while($decimal > 1)  {  $remainder = $decimal%2;  $decimal /= 2;  unshift (@bin, $remainder);  }  }  @bin = ();  print "Enter a number to convert: ";  chomp($decimal = <STDIN>);  print "Converting $decimal to binary...\n";  dec\_to\_bin($decimal);  print "Bin: @bin"; | | | |  | **6** | **1** | **D** | **Y** | | |
| 2. | |  | **Write PERL commands to (a) add 2 arrays together, (b) splice an array (c) empty an array (d) add 2 elements in the beginning of array (e) print the contents of an array. Explain each with an example.**  **Answer:**  @array\_numeric = (1, 2, 3, 4, 5, 6, 7);  @array\_alpha = (A, B, C, D, E, F);  @alphanumeric = (@array\_numeric, @array\_alpha);  print "1. Merged array: @alphanumeric \n";  splice (@array\_numeric, 0, 3, @array\_alpha[0..3]);  print "2. Splice array: @array\_numeric \n";  unshift (@array\_alpha, "\*", "+");  print "4. Adding 2 extra symbol: @array\_alpha \n";  print "5. Printing content of array \n";  print " Array\_numeric: @array\_numeric  Array\_alpha: @array\_alpha  Alphanumeric: @alphanumeric \n";  print "3. Empting an array \n";  $n = scalar @array\_numeric;  print "Before: New\_array\_numeric: @array\_numeric \n";  for ($i = 0; $i < $n; $i+=1) {  pop (@array\_numeric);  }  print "After: New\_array\_numeric: @array\_numeric \n"; | | | |  | **6** | **1** | **D** | **Y** | | |
| 3. | |  | **Write a TCL procedure to check whether the year is a leap year or not.**  **Answer:**  proc leap\_year {year} {  if {$year%400==0 || $year%100!=0 || $year%4==0} {  puts "Leap year: $year \n"  } else {  puts "Not leap year: $year \n"  }  }  puts "Enter year"  gets stdin years  puts [leap\_year $years] | | | |  | **6** | **2** | **D** | **Y** | | |
| 4. | |  | **Explain briefly the following RTL test strategies: (a) directed tests (b) random tests (c) transaction based tests.** | | | |  | **6** | **4** | **E** | **N** | | |
| 5. | |  | **Explain the advantages of random two-state RTL simulation methods.** | | | |  | **6** | **5** | **D** | **N** | | |
|  | **PART – B (70 Marks) Answer all the Questions** | | | | | | | | | | | | |
| 6. | | (a) | **The government has introduced a road rationing system by imposing the ‘odd-even’ rule. On even dates, only cars with license plates ending with an even number will be allowed on the city roads. On odd dates cars with license plates ending with odd number will be allowed. Write a PERL script that will help to create a file containing records of all cars in the city. The input data should be entered while running the script. The file should contain 20 entries in the following format**  **Name Registration Number**  **Rahul TN02BC9906**  **Ganesh TN01AC1525**  **Answer:**  print "Starting Process \n";  print "Enter number of details you want to enter: ";  chomp($N = <STDIN>);  open(REG, ">Registration\_detail".".txt") or die "Unable to create file \n";  print REG "Name\t\tRegistration number \n";  for ($i=0; $i<$N; $i += 1) {  print "Enter number ${\($i+1)} name: ";  chomp($Name = <STDIN>);  print "Enter the ${\($i+1)} registration number: ";  chomp($Reg\_no = <STDIN>);  print REG "$Name\t\t$Reg\_no \n";  }  close REG; | | | |  | **7** | **1** | **C** | **Y** | | |
|  | | (b) | **Write a PERL script that will accept the date as input and determine from the above file, the cars that would be permitted on the road on that day based on the odd-even rule. The output should display the name and registration number of the permitted vehicles.**  **Answer:**  print "Enter the date in DD/MM/YYYY format: ";  chomp($Date = <STDIN>);  print "Date is: $Date \n";  if ($Date =~ /(\d{2})\/\d{2}\/\d{4}/) {  $day = $1;  print "day: $day \n";  if ($day%2==0) {  print "Date $Date is even \n";  $eo = "E";  } else {  print "Date $Date is odd \n";  $eo = "O";  }  } else {  print "Incorrect format \n";  $eo = "blank";  }  print "eo: $eo \n";  open(EO, "Registration\_detail.txt") or die "Unable to open";  while ($line = <EO>) {  if ($line =~ /\w+\s+[A-Z]{2}[0-9]{2}[A-Z]{2}(\d+)/) {  $digit = $1;  if ($digit%2==0 && $eo eq "E") {  print "Even: $line";  } elsif ($digit%2!=0 && $eo eq "O") {  print "Odd: $line";  } else {  print "";  }  }  }  close EO | | | |  | **8** | **1** | **C** | **Y** | | |
| 7. | | (a) | **A database stores names by reversing the name for a particular security reason. But names that are palindromes would have a problem. Write a TCL script that will accept names during run time and store them in a file with a field that indicates if the name is reversed or not (as it could be a palindrome)**  **Name P-Palindrome/R-Reversed**  **YAJA R**  **NITIN P**  **IVAR R**  **Answer:**  set wDB [open "database.txt" w+]  puts $wDB "Name\t\tP-Palindrome/R-Reversed"  puts "How many name you want to enter"  gets stdin N\_name  for {set i 0} {$i<$N\_name} {incr i} {  puts "Enter name [expr {$i+1}]"  gets stdin name  set rev\_name {}  set N [string length $name]  for {set i $N} {$i>0} {incr i -1} {  append rev\_name [string index $name [expr {$i-1}]]  }  if {[string tolower $name] == [string tolower $rev\_name]} {  set palindrome P  } else {  set palindrome R  }  puts $wDB "$rev\_name\t\t$palindrome"  puts "$rev\_name\t\t$palindrome"  }  close wDB | | | |  | **7** | **2** | **C** | **Y** | | |
|  | | (b) | **Write a TCL script to read the above file containing reversed names and palindromes and list all original names.**  **Answer:**  set Rmode [open "database.txt" r]  set count 0  while {[gets $Rmode data] >= 0} {  set name {}  set len [string first "\t" $data]  set rev\_name [string range $data 0 $len]  for {set i $len} {$i>0} {incr i -1} {  append name [string index $rev\_name [expr {$i-1}]]  }  if {$count==0} {  puts $rev\_name  } else {  puts $name  }  incr count  }  close Rmode | | | |  | **8** | **2** | **C** | **Y** | | |
| 8. | |  | Write a TCL program for automatically generate a Verilog code for a CRC generator to generate a CRC code with a generator polynomial x5+x4+x2+1 and mesage 101000110 . | | | |  | **15** | **3** | **C** | **Y** | | |
|  | We are excited to announce that the Application Form for the 2nd Qualifier process is open now.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | | |  |  | | |  |  | | |  | | | | | | |  | | | | | | |  | | | | | | | **Q.No.** | **Sub. Sec.** | **Question Description** | | | **Marks** | | | 1. | a) | **Write a program in PERL to create an array with five elements and print each element in the array.**  **Answer:**  @array = (A, B, C, D, E);  $N = scalar (@array);  for ($i=0; $i<$N; $i+=1) {  print "Element $i of array is @array[$i] \n";  } | | | **[3]** | | |  | b) | **Write a PERL program that makes a modified copy of a text file with every string “Fred” is replaced with “Bob”.**  **Answer:**  $text = "Hey this Fred. Fred from USA. Please welcome Fred in India";  open(Wmode, ">file1".".txt") or die "unable to open file \n";  print Wmode "$text";  close Wmode;  open(Rmode, "file1.txt") or die "unable to open file";  open(Wmode, ">file2".".txt") or die "unable to open file";  while ($line = <Rmode>) {  if ($line =~ s/Fred/Bob/g) {  print Wmode "$line \n";  }  }  close Rmode;  close Wmode; | | | **[3]** | | | 2. |  | **Write a PERL program to find and print the smallest of any three numbers given.**  **Answer:**  print "Enter first number: ";  chomp($number1 = <STDIN>);  print "Enter second number: ";  chomp($number2 = <STDIN>);  print "Enter third number: ";  chomp($number3 = <STDIN>);  if ($number1 < $number2) {  print "Number 1: ($number1) is smallest";  } elsif ($number2 < $number3) {  print "Number 2: $number2 is smallest";  } else {  print "Number 3: $number3 is smallest";  } | | | **[6]** | | | 3. |  | **Write a subroutine in PERL that takes an array of integers, and returns the numbers in the ascending order.**  **Answer**:  sub sorted {  @array = @\_;  @sorted\_array = sort (@array);  print "Before: @array \n";  print "After: @sorted\_array";  }  print "How many number you want to enter: ";  chomp($N = <STDIN>);  @list = ();  for ($i=0; $i<$N; $i+=1) {  print "Enter element number ${\($i+1)}: ";  chomp($entry = <STDIN>);  push (@list, $entry)  }  sorted (@list); | | | **[6]** | | | 4. | a) | **The symbols $, @ and % when prefixing a variable in PERL have specific meaning, explain each with an example.**  **Answer:**  **$:** A scalar variable will precede by a dollar sign ($) and it can store either a number, a string, or a reference.  **@:** An array variable will precede by sign @ and it will store ordered lists of scalars  **%:** The Hash variable will precede by sign % and will be used to store sets of key/value pairs | | | **[3]** | | |  | b) | **Differentiate split and join function in PERL with an example.**  **Answer:**  **Split():** Let's look into one more function called split(), which has the following syntax −  split [ PATTERN [ , EXPR [ , LIMIT ] ] ]  This function splits a string into an array of strings, and returns it. If LIMIT is specified, splits into at most that number of fields. If PATTERN is omitted, splits on whitespace.  **Join():** We can use the join() function to rejoin the array elements and form one long scalar string. This function has the following syntax −  join EXPR, LIST  This function joins the separate strings of LIST into a single string with fields separated by the value of EXPR, and returns the string | | | **[3]** | | | 5. |  | **Write a PERL program to get list of strings from user (enter in order: fred, barney, wilma, betty) and print in order betty, barney fred wilma.**  **Answer:**  print "How many string you want to enter: ";  chomp($N = <STDIN>);  @list\_name = ();  for ($i=0; $i<$N; $i+=1) {  print "Enter string number ${\($i+1)}: ";  chomp($name = <STDIN>);  push (@list\_name, $name);  }  @sorted\_list = sort (@list\_name);  $first = @sorted\_list[1];  @sorted\_list[1] = @sorted\_list[0];  @sorted\_list[0] = $first;  print "Final list: @sorted\_list"; | | | **[6]** | | | 6. |  | **Write a PERL program to open and insert “Welcome to VIT University” as the third line in a text file named vitdata.txt.**  **Answer:**  $data = "Welcome to VIT university";  open(VIT, "> vitdata".".txt") or die "unable to create file";  print "\n\n$data";  print VIT "\n\n$data";  close VIT; | | | **[6]** | | | **PART – B (2 X 10 = 20 Marks) Answer any TWO Questions** | | | | | | | 7. |  | **Write a PERL program to automatically generate VHDL codes for two-input AND, OR, NAND, NOR and Ex-OR gates and save as code.v file.**  **Answer:**  print "Enter name of gate: ";  chomp($gate = <STDIN>);  $GATE = uc($gate);  print "You want to generte $GATE gate\n";  print "Enter module name\n";  chomp($module\_name=<STDIN>);  print "Enter first input\n";  chomp($first\_input=<STDIN>);  print "Enter second input\n";  chomp($second\_input=<STDIN>);  print "Enter output name\n";  chomp($output=<STDIN>);  $compliment="";  if ($GATE eq "AND") {  $symbol="&";  } elsif ($GATE eq "OR") {  $symbol="|";  } elsif ($GATE eq "NAND") {  $symbol="&";  $compliment="~";  } elsif ($GATE eq "NOR") {  $symbol="|";  $compliment="~";  } elsif ($GATE eq "XOR") {  $symbol="";  } else {  $symbol="";  }  open (Wmode, "> $module\_name".".v") or die "Unable to create file\n";  print Wmode "module $module\_name \($output, $first\_input, $second\_input\);\n";  print Wmode "output $output\;\n";  print Wmode "input $first\_input, $second\_input;\n";  print Wmode "assign $output = $compliment\($first\_input $symbol $second\_input\);\n";  print Wmode "endmodule\n";  close Wmode | | | **[10]** | | | 8. | a) | What is netlist patching? Explain with an example. | | | **[3]** | | |  | b) | **Write a PERL program that searches for lines starting with “https://” and ending with “.com” and copy those URLs alone and paste in another file.**  **Answer:**  print "Enter the file name: ";  chomp($file\_name=<STDIN>);  open(Rmode,"$file\_name") or die "Unable to open file\n";  open(Wmode,"> URL".".txt") or die "Unable to create file\n";  while ($line = <Rmode>) {  if ($line=~ /(https\:\/\/.\*\.com)/) {  $url = $1;  print $url;  print Wmode "$url";  }  }  close Wmode;  close Rmode; | | | **[7]** | | | 9. |  | **Write a PERL program to accept details of n students from keyboard and create a menu having students details (reg no., name, total marks, percentage).**  **Answer:**  print "Enter file name: ";  chomp($file\_name=<STDIN>);  open(Wmode,"> $file\_name".".txt") or die "Unable to create file\n";  print "Details of how many student you want to enter: ";  $N=<STDIN>;  printf Wmode ("%-15s %-15s %-15s %-15s\n","Name", "Reg\_no", "Total Mark", "Percentage");  for ($i=0; $i<$N;$i+=1) {  print "Enter Name of student no. ${\($i+1)}: ";  chomp($name=<STDIN>);  print "Enter Reg. no. of student no. ${\($i+1)}: ";  chomp($reg\_no=<STDIN>);  print "Enter Total marks of student no. ${\($i+1)}: ";  chomp($mark=<STDIN>);  print "Enter Percentage of student no. ${\($i+1)}: ";  chomp($percent=<STDIN>);  printf Wmode ("%-15s %-15s %-15s %-15s\n",$name, $reg\_no, $mark, $percent);  }  close Wmode | | | **[10]** | | |  |  | Total Marks | | | **[50]** | | | | | | | | | | | | | |  |

1. A VLSI Design engineer verifies the functionality of the full-adder circuit in structural modeling using individual module files, however, for the EDA simulation, the compilation time is more. Therefore, you are requested to write a Perl script to combine the individual module Verilog files into single. V file to perform the functional simulation very fast. (Note: Consider the full-adder is implemented using two half-adder modules). Then after combining count the number of lines the old file and new file and also display its contents on the screen with a suitable title.

**2. Write a Perl script to check if the given two files are same or not, if not same then filter the common lines from both the files to a new file and then perform the following operations**

1. **Map the first line to Upper case**
2. **In the last line map every first letter in each word to be Uppercase letter**
3. **Also, count the number of digits present in the new file**
4. **Finally, display the values with explanation in the screen**

**Answer:**

use File::Compare;

print "Enter name of first file: ";

chomp($first\_file = <STDIN>);

print "Enter name of second file: ";

chomp($second\_file = <STDIN>);

$compare = compare("$first\_file", "$second\_file");

if ($compare == 0) {

print "Files are equal";

} elsif ($compare == 1) {

print "File are not equal\n";

open(Rmode1, "$first\_file") or die "File does not exist\n";

open(Rmode2, "$second\_file") or die "File does not exist\n";

open(Wmode, "> commanline.txt") or die "Unable to create file\n";

@\_ = <Rmode2>;

while ($file1 = <Rmode1>) {

chomp($file1);

print "1: $file1\n";

foreach $file2 (@\_) {

chomp($file2);

print "2: $file2\n";

if ($file1 eq $file2) {

print Wmode "$file1\n";

}

}

}

close Wmode;

close Rmode1;

close Rmode2;

} else {

print "Error\n";

}

**3. Write a TCL script to create GUI with TWO buttons, as shown in the figure. The GUI has to perform the following operation that if one button is pressed it takes the corresponding input from the button and changes it into another name as VIT to VLSI and vice versa. If Quit is pressed, it will come out of the window.**

****

**Answer:**

grid [button .myButton1 -textvariable Button1 -command "set Button1 VLSI"]

set Button1 "VIT"

grid [button .myButton2 -text "Quit" -command "destroy ."]

grid .myButton1 -row 0 -column 0

grid .myButton2 -row 0 -column 1

4. Implement the following instructions in a Perl program. Take three non-zero digits. Create six two-digit numbers from these digits and add these together. Now add the original three digits together and divide the previous result by this sum. Test your program with three different sets of digits. What results do you obtain? Note: The input should be got during run time.

5. Write a Perl script for the following string descriptions using regular expressions:

1. Strings containing only a's and b's.
2. Strings that do not contain white space.
3. Strings with exactly one word regardless of white space.
4. Strings that end with the same character they start with.
5. Like 1. but the number of a's should be even.
6. Any string.
7. No string, not even the empty string.

You can test the expressions by putting them in a small program that tests whether an input string matches the expression or not.

6. Create a Perl script named **lab1.pl** in your user directory. The script must perform the following:

1. Create an array listing six of your favorite foods.
2. Create an array listing six foods you dislike.
3. Print out the first, third and last items in the favorite foods array. Include some descriptive text to clarify your output.
4. Ask the user to input another disliked food but then append this new item to the liked foods array.
5. **Move** the newly added element to the end of the disliked foods array.
6. For the "favorite foods" array, **move** the value of the first element of the array to become the last element of the same array.
7. Print out the first and last elements of each array to show the changes from the previous step.
8. Print (to the terminal) the last two element of each array with some descriptive text.
9. Also print to the terminal the number of elements in each array with some text.
10. Make the script executable with chmod.

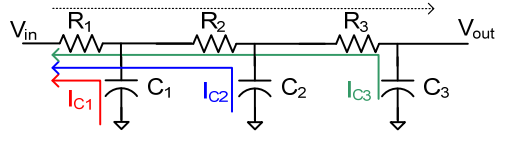
7. Create a Perl script named **lab2.pl** in your user directory. The script that has the following features:

1. Prompt the user to enter at sentence (store in one variable) of at least 5 words.
2. Break the string apart storing the individual words in an array.
3. Print the original sentence to the screen with the first word capitalized and all others in lower case.
4. Print the last 3 letters of each word in the sentence to the screen.
5. Join the second and third words into a string and store in a new variable.
6. Print this new variable to the screen in all UPPER-case letters except for the last letter
7. Prompt the user to enter their full name
8. Create a subroutine that returns a lower-case string comprised of the capitalized entered last name and first initial as a single word (ie. "Mark Tucker" would be returned as "tuckerm") when passed the first and last name as arguments.
9. Print this returned value to the screen.
10. Prompt the user to enter the current day of the week.
11. Convert the day of the week to the first 3 letter in upper case.
12. Print the weekday and the username to the screen with some descriptive text using one print statement.

8. For the given netlist report the number of gate count for each of the gates, wire count and calculate the total area from the synthesized netlist using the following reference table.

|  |  |  |
| --- | --- | --- |
| **S.No** | **Component** | **Area of single component in Sq. µm** |
| **1** | **EDFFX2AD** | **150** |
| **2** | **DFFQXLAD** | **110** |
| **3** | **NAND2BX1AD** | **176** |
| **4** | **OAI2BB1X1AD** | **190** |
| **5** | **INVX1AD** | **220** |
| **6** | **AOI31X1AD** | **80** |
| **7** | **NAND2X1AD** | **140** |
| **8** | **NOR2X1AD** | **120** |
| **9** | **XOR2X1AD** | **70** |
| **10** | **NAND3X1AD** | **20** |
| **11** | **XNOR2X1AD** | **60** |
| **12** | **NOR2BX1AD** | **30** |
| **13** | **NAND4X1AD** | **50** |
| **14** | **Others** | **10** |

9. **Write a Perl script to calculate the delay of the given circuit using Elmore model analysis. (Note: R1, R2 and R3 = 150 K, C1, C2 and C3 = 100 µF)**

****

**Answer:**

print "Enter value of R1 in K: ";

chomp($R1=<STDIN>);

print "Enter value of C1 in micro: ";

chomp($C1=<STDIN>);

print "Enter value of R2 in K: ";

chomp($R2=<STDIN>);

print "Enter value of C2 in micro: ";

chomp($C2=<STDIN>);

print "Enter value of R3 in K: ";

chomp($R3=<STDIN>);

print "Enter value of C1 in micro: ";

chomp($C3=<STDIN>);

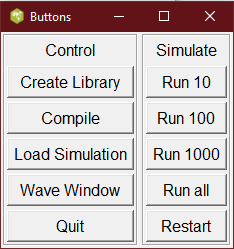
print "Calculating Elmore delay\n";

$elmoreDelay = ($R1\*$C1)+(($R1+$R2)\*$C2)+(($R1+$R2+$R3)\*$C3);

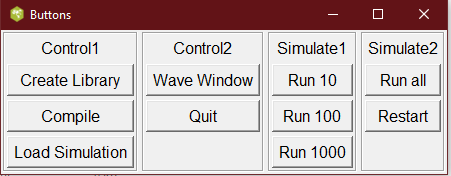
print "Elmore delay is: $elmoreDelay ms\n";

10. Implement a TCL widget-based custom GUI as shown in the figure for simulating a digital circuit as described by the following conditions. The circuit has three inputs and two outputs. Then the circuit will produce output based on the following conditions (Identify the circuit and simulate using GUI)

1. If anyone of the inputs is logic high, then anyone of the output is logic high.
2. If any two of the inputs are logic high, then the other output is logic high.
3. If all the inputs are logic high, then all the outputs are logic high.
4. If all the inputs are logic low, then all the outputs are logic low



11. Write a TCL widget-based custom GUI as shown in the figure for simulating a FIFO read and write operation



**12. Write a TCL widget-based custom GUI using buttons & text entry for module & testbench creations. Note: The TCL widget has to run the corresponding Perl script for .V file creation**

**Answer:**

proc GATE {arg tg} {

set module\_name [.eModule get]

set first\_input [.eFirst get]

set second\_input [.eSecond get]

set output [.eOutput get]

#condition for gate

if {$tg == "gate"} {

set compliment ""

if {$arg == "and"} {

set symbol "&"

.and config -text "Sucessfull" -bg blue

} elseif {$arg == "nand"} {

set symbol "&"

set compliment "~"

.nand config -text "Sucessfull" -bg blue

} elseif {$arg == "or"} {

set symbol "|"

.or config -text "Sucessfull" -bg blue

} elseif {$arg == "nor"} {

set symbol "|"

set compliment "~"

.nor config -text "Sucessfull" -bg blue

} elseif {$arg == "xor"} {

set symbol "^"

.xor config -text "Sucessfull" -bg blue

} else {

set symbol ""

}

#Writing module file

set mod ${module\_name}.v

set Wmode [open "$mod" w+]

puts $Wmode "module $module\_name ($output, $first\_input, $second\_input);"

puts $Wmode "output $output;"

puts $Wmode "input $first\_input, $second\_input;"

puts $Wmode "assign $output = $compliment ($first\_input $symbol $second\_input);"

puts $Wmode "endmodule"

# close Wmode

} elseif {$tg == "testbench"} {

if {$arg == "and"} { .andt config -text "Sucessfull" -bg blue

} elseif {$arg == "nand"} { .nandt config -text "Sucessfull" -bg blue

} elseif {$arg == "or"} { .ort config -text "Sucessfull" -bg blue

} elseif {$arg == "nor"} { .nort config -text "Sucessfull" -bg blue

} elseif {$arg == "xor"} { .xort config -text "Sucessfull" -bg blue

}

set tb\_mod tb\_${module\_name}.v

set Wmode [open "$tb\_mod" w+]

puts $Wmode "module tb\_$module\_name ();"

puts $Wmode "reg $first\_input, $second\_input;"

puts $Wmode "wire $output;\n"

puts $Wmode "$module\_name uut (.$output\($output), \

.$first\_input\($first\_input), .$second\_input\($second\_input));\n"

puts $Wmode "initial begin;"

puts $Wmode "#10 $first\_input=1'b0; $second\_input=1'b0;"

puts $Wmode "#10 $first\_input=1'b0; $second\_input=1'b1;"

puts $Wmode "#10 $first\_input=1'b1; $second\_input=1'b0;"

puts $Wmode "#10 $first\_input=1'b1; $second\_input=1'b1;"

puts $Wmode "end"

puts $Wmode "endmodule"

}

}

label .lModule -text "Enter module name" -relief groove -width 20 -justify left

label .lFirst -text "Enter first input" -relief groove -width 20 -justify left

label .lSecond -text "Enter second input" -relief groove -width 20 -justify left

label .lOutput -text "Enter output" -relief groove -width 20 -justify left

button .and -text "AND Gate" -bg teal -fg white -width 20 -command {GATE "and" "gate"}

button .andt -text "AND testbench" -bg teal -fg white -width 16 -command {GATE "and" "testbench"}

button .nand -text "NAND Gate" -bg teal -fg white -width 20 -command {GATE "nand" "gate"}

button .nandt -text "NAND testbench" -bg teal -fg white -width 16 -command {GATE "nand" "testbench"}

button .or -text "OR Gate" -bg teal -fg white -width 20 -command {GATE "or" "gate"}

button .ort -text "OR testbench" -bg teal -fg white -width 16 -command {GATE "or" "testbench"}

button .nor -text "NOR Gate" -bg teal -fg white -width 20 -command {GATE "nor" "gate"}

button .nort -text "NOR\_testbench" -bg teal -fg white -width 16 -command {GATE "nor" "testbench"}

button .xor -text "XOR Gate" -bg teal -fg white -width 20 -command {GATE "xor" "gate"}

button .xort -text "XOR\_testbench" -bg teal -fg white -width 16 -command {GATE "xor" "testbench"}

entry .eModule -relief groove -width 20

bind .eModule <Return> {.lModule config -text "Module name entered" -bg green -fg white

.eModule config -bg green -fg white}

entry .eFirst -relief groove -width 20

bind .eFirst <Return> {.lFirst config -text "First input name entered" -bg green -fg white

.eFirst config -bg green -fg white}

entry .eSecond -relief groove -width 20

bind .eSecond <Return> {.lSecond config -text "Second input entered" -bg green -fg white

.eSecond config -bg green -fg white}

entry .eOutput -relief groove -width 20

bind .eOutput <Return> {.lOutput config -text "Output entered" -bg green -fg white

.eOutput config -bg green -fg white}

grid .lModule -row 0 -column 0

grid .lFirst -row 1 -column 0

grid .lSecond -row 2 -column 0

grid .lOutput -row 3 -column 0

grid .eModule -row 0 -column 1

grid .eFirst -row 1 -column 1

grid .eSecond -row 2 -column 1

grid .eOutput -row 3 -column 1

grid .and -row 4 -column 0

grid .andt -row 4 -column 1

grid .nand -row 5 -column 0

grid .nandt -row 5 -column 1

grid .or -row 6 -column 0

grid .ort -row 6 -column 1

grid .nor -row 7 -column 0

grid .nort -row 7 -column 1

grid .xor -row 8 -column 0

grid .xort -row 8 -column 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | |  |  | |
|  |  | |
|  | | | | | |
|  | | | | | |
| **PART – A (4 X 5 = 20 Marks)** | | | | | |
| **Q.No.** | **Sub. Sec.** | **Question Description** | | | **Marks** |
| 1. | A)  B) | **Explain briefly about pattern matching in Perl with three examples.**  **Answer:**  A regular expression is a string of characters that defines the pattern or patterns you are viewing. The syntax of regular expressions in Perl is very similar to what you will find within other regular expression.supporting programs, such as sed, grep, and awk.  The basic method for applying a regular expression is to use the pattern binding operators =~ and !~. The first operator is a test and assignment operator.  There are three regular expression operators within Perl.  **Match Regular Expression - m//**  **Substitute Regular Expression - s///**  **Transliterate Regular Expression - tr///**  The forward slashes in each case act as delimiters for the regular expression (regex) that you are specifying. If you are comfortable with any other delimiter, then you can use in place of forward slash.  **The Match Operator**  The match operator, m//, is used to match a string or statement to a regular expression. For example, to match the character sequence "foo" against the scalar $bar, you might use a statement like this  #!/usr/bin/perl  $bar = "This is foo and again foo";  if ($bar =~ /foo/) {  print "First time is matching\n";  } else {  print "First time is not matching\n";  }  $bar = "foo";  if ($bar =~ /foo/) {  print "Second time is matching\n";  } else {  print "Second time is not matching\n";  }  **The Substitution Operator**  The substitution operator, s///, is really just an extension of the match operator that allows you to replace the text matched with some new text. The basic form of the operator is −  **s/PATTERN/REPLACEMENT/;**  The PATTERN is the regular expression for the text that we are looking for. The REPLACEMENT is a specification for the text or regular expression that we want to use to replace the found text with. For example, we can replace all occurrences of dog with cat using the following regular expression −  #/user/bin/perl  $string = "The cat sat on the mat";  $string =~ s/cat/dog/;  print "$string\n";  **The Translation Operator**  Translation is similar, but not identical, to the principles of substitution, but unlike substitution, translation (or transliteration) does not use regular expressions for its search on replacement values. The translation operators are −  **tr/SEARCHLIST/REPLACEMENTLIST/cds**  **y/SEARCHLIST/REPLACEMENTLIST/cds**  The translation replaces all occurrences of the characters in SEARCHLIST with the corresponding characters in REPLACEMENTLIST. For example, using the "The cat sat on the mat." string we have been using in this chapter  #/user/bin/perl  $string = 'The cat sat on the mat';  $string =~ tr/a/o/;  print "$string\n";  Write a Perl Script to generate a HDL Testbench wrapper file Module for the design you are starting and save it in a .v file extension. | | | **[3]**  **[2]** |
| 2. | A)  B) | Explain Briefly about the TCL commands with an example   * Switch   **Answer:**  A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.  **Syntax**  The syntax for unquoted **switch** statement in Tcl language is as follows −  **switch switchingString matchString1 {body1} matchString2 {body2} ... matchStringn {bodyn}**  The syntax for unquoted **switch** statement in Tcl language is as follows –  switch switchingString {  matchString1 {  body1  }  matchString2 {  body2  }  ...  matchStringn {  bodyn  }  }  The following rules apply to a **switch** statement −   1. The **switching String** is used in a **switch** statement; used between the different blocks by comparing to the match String. 2. You can have any number of match String blocks within a switch. 3. A **switch** statement can have an optional **default** block, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true.  * **Eval**   **Answer:**  **Eval** takes one or more arguments, which together comprise a Tcl script containing one or more commands. **Eval** concatenates all its arguments in the same fashion as the **[concat](https://www.tcl.tk/man/tcl8.6/TclCmd/concat.htm)** command, passes the concatenated string to the Tcl interpreter recursively, and returns the result of that evaluation (or any error generated by it). Note that the [**list**](https://www.tcl.tk/man/tcl8.6/TclCmd/list.htm) command quotes sequences of words in such a way that they are not further expanded by the **eval** command.  Write the command to get input from a text file in TCL and Perl  Answer:  **TCL**:  set Rmode [open "file\_name.txt" r] {  while {[gets $Rmode data] >= 0} {  puts "line: $data"  }  }  close $Rmode  **Perl**:  open(Rmode, "< file\_name.txt") or die "unable to open the file";  while ($list = <Rmode>) {  print "Line: $list \n";  }  close Rmode; | | | **[4]**  **[1]** |
| 3. |  | **Write a TCL script to automate a simulation in modelsim using buttons for compiling the module Fulladder**  **Answer:**  proc include\_these\_buttons {} {  add\_frame "Control"  add\_button "Create Library" {vlib work; vmap work work}  add\_button "Compile" { vlog Adder.v  vlog Adder\_tb.v }  add\_button "Load Simulation" {vsim work.first\_counter\_tbvsim -voptargs=+acc work.Adder\_tb  vsim -voptargs=+acc work.Adder\_tb}  add\_button "Wave Window" {add wave -r /\*}  add\_button "Quit" {quit -force}  add\_frame "Simulate"  add\_button "Run 10" {run 10}  add\_button "Run 100" {run 100}  add\_button "Run 1000" {run}  add\_button "Run all" {run -all}  add\_button "Restart" {restart}  }  proc add\_frame title {  global buttons  set buttons(frame) .frame$buttons(widget\_count)  frame $buttons(frame) -border 2 -relief groove  label $buttons(frame).label -text $title -font $buttons(font)  pack $buttons(frame) -side left -padx 2 -pady 2 -anchor n -fill both -expand 1  pack $buttons(frame).label -side top -padx 2 -pady 2  incr buttons(widget\_count)  }  proc add\_button {title command} {  global buttons  button $buttons(frame).b$buttons(widget\_count) -text $title -font $buttons(font) -command "puts \"$command\""  pack $buttons(frame).b$buttons(widget\_count) -side top -pady 2 -padx 2 -fill x  incr buttons(widget\_count)  }  proc respond\_to\_buttons {} {  global buttons  if [eof $buttons(pipe\_id)] {  catch {close $buttons(pipe\_id)}  } elseif { [gets $buttons(pipe\_id) line] >= 0 } {  puts $line  eval $line  }  }  if [string compare $argv buttons\_gui] {  global buttons  if [catch {set buttons(pipe\_id) [open "|wish buttons.tcl buttons\_gui" r+]}] {  puts "Couldn't start wish for the buttons GUI\n"  } else {  fconfigure $buttons(pipe\_id) -blocking 0 -buffering line  fileevent $buttons(pipe\_id) readable respond\_to\_buttons  }  } else {  wm title . Buttons  set buttons(font) [font create -family {Arial Helvetica "Courier New"} -size 12]  set buttons(frame) {}  set buttons(widget\_count) 1  include\_these\_buttons  } | | | **[5]** |
| **PART – B (5 X 16 = 80 Marks) Answer any FIVE Questions** | | | | | |
| 6. | A)  B) | In the puzzle  **LEARN - PERL = HERE**  Each character represents a number (**0-9**). Different characters represent different numbers and H, L and P are not equal to zero. Furthermore the subtraction is correct. Write a Perl program that solves this puzzle. What solutions do you get?  Write a Perl script to create a simple database using hash declaration. In that program explain briefly about the creating, appending, and reading & writing operation by tying the hash to database. | | | **[8]**  **[8]** |
| 7. | A)  B) | **Write the Output for the following TCL Program with explanation**  **proc SetP {variable value } {**  **upvar $variable myvar**  **if {$value < 0} {**  **set myvar [expr {-$value}]**  **} else {**  **set myvar $value**  **}**  **return $myvar**  **}**  **SetP x 5**  **SetP y -5**  **puts "X : $x Y: $y\n"**  **Answer:** X : 5 Y: 5  Discuss the following TCL Commands with an example.   1. string index 2. toupper 3. expr 4. linsert. 5. lappend | | | **[6]**  **[10]** |
| 8. | A)  B) | **Create a TCL Script for creating a GUI window for checkbox for font changing operation as shown below and explain each commands using a procedure call**  **Capture**  **Answer:**  set font helvetica  proc applyIt { } {  global bold italics font  if {$bold} {set weight bold} {set weight normal}  if {$italics} {set slant italic} {set slant roman}  .b configure -font "-family $font -weight $weight -slant $slant"  }  checkbutton .c1 -text Bold -variable bold -anchor w  checkbutton .c2 -text Italics -variable italics -anchor w  radiobutton .r1 -text Helvetica -variable font -value helvetica  radiobutton .r2 -text Courier -variable font -value courier  button .b -text Apply -command "applyIt"  grid .c1 -row 0 -column 1 -sticky w  grid .c2 -row 1 -column 1 -sticky w  grid .r1 -row 0 -column 0 -sticky w  grid .r2 -row 1 -column 0 -sticky w  grid .b -row 2 -column 0 -columnspan 2  **Create a Simple GUI window using Label Creation and display the operation result of adding 3+5 in the output window.**  **Answer:**  proc sum {label} {  set add [expr 3 + 5]  set button [tk\_messageBox  -icon question \  -type yesno \  -title "Sum of 3 and 5" \  -parent . \  -message "Sum is $add"]  $label configure -text $button  }  label .l -text "Sum of 3 and 5" \  -background teal \  -foreground white \  -relief ridge \  -borderwidth 3  button .b -text "Click me to show result" \  -command "sum .l" \  -background teal \  -foreground white \  -relief ridge \  -borderwidth 3  grid .b -row 0 -column 0  grid .l -row 0 -column 1 | | | **[8]**  **[8]** |
| 11. |  | Write a Perl script to do the following operations   * Create an array listing of six of your favourite foods * Create an array listing of six foods you dislike. * Print out the first, third and last items in the favourite foods array. Include some descriptive text to clarify your output. * Ask the user to input another disliked food but then append this new item to liked foods array. * Move the newly added element to the end of the disliked foods array. * For the favourite foods array move the value of the first element of the array to become the last element of the same array. * Print out the first and last elements of each array to show the changes from previous step. * Print the last two elements of each array with some descriptive text. * Also print the number of elements in each array with some text. | | | **[16]** |
|  |  | Total Marks | | | **100** |
| ⇔⇔⇔   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | | |  |  | | | |  |  | | | |  | | | | | | |  | | | | | | | **PART – A (4 X 5 = 20 Marks)** | | | | | | | **Q.No.** | **Sub. Sec.** | **Question Description** | | | **Marks** | | 1. | A)  B) | Explain briefly about scalar, array and hash in Perl  Discuss foreach and while loop in Perl with an example | | | **[3]**  **[2]** | | 2. | A)  B) | Explain Briefly about the TCL commands with an example   * Proc * Dict   Write the command to get input from user while during the runtime in TCL | | | **[4]**  **[1]** | | 3. | A)  B) | Write a TCL script to automate a synthesis in RTL Compiler using buttons for synthesizing the module Halfadder.  Enumerate the difference between verification and testing. | | | **[3]**  **[2]** | | **PART – B (5 X 16 = 80 Marks) Answer any FIVE Questions** | | | | | | | 6. | A)  B) | **Rot13 is an encryption method in which each of the 26 characters a-z or A-Z is replaced by the character 13 places ahead or 13 places back in the alphabet. Write a Perl program that reads a series of text and outputs the rot13 equivalent.**  **Answer:**  print "Enter sentence: ";  chomp($sentence=<STDIN>);  @rotarray =();  @character = split(//,$sentence);  $N=scalar(@character);  for ($i=0;$i<$N;$i=$i+1) {  rot13 ($character[$i]);  }  $rotsen = join("",@rotarray);  print "Rot13 of $sentence is: $rotsen";  sub rot13 {  chomp($char=$\_[0]);  $ascai\_num = ord($char);  $rot13\_num = $ascai\_num + 13;  if ($ascai\_num > 64 && $ascai\_num < 91) {  if ($rot13\_num>90) {  $rot13\_char = chr($rot13\_num - 26);  #print "Rot13 of $char is $rot13\_char\n";  push (@rotarray,"$rot13\_char");  } else {  $rot13\_char = chr($rot13\_num);  #print "Rot13 of $char is $rot13\_char\n";  push (@rotarray,"$rot13\_char");  }  } elsif ($ascai\_num > 96 && $ascai\_num < 123) {  if ($rot13\_num>122) {  $rot13\_char = chr($rot13\_num - 26);  #print "Rot13 of $char is $rot13\_char\n";  push (@rotarray,"$rot13\_char");  } else {  $rot13\_char = chr($rot13\_num);  #print "Rot13 of $char is $rot13\_char\n";  push (@rotarray,"$rot13\_char");  }  } else {  $rot13\_char = chr($ascai\_num);  # print "Rot13 of $char is $rot13\_char\n";  push (@rotarray,"$rot13\_char");  }  }  Create a Hash using a Perl script by reading a text file. | | | **[8]**  **[8]** | | 7. | A)  B) | **Write the Output for the following TCL Scripts with explanation**  **set sample "Where there is a will, There is a way."**  **set result [regexp {[a-z]+} $sample match]**  **puts "Result: $result match: $match"**  **set result [regexp {([A-Za-z]+) +([a-z]+)} $sample match sub1 sub2 ]**  **puts "Result: $result Match: $match 1: $sub1 2: $sub2"**  **regsub "way" $sample "lawsuit" sample2**  **puts "New: $sample2"**  **puts "-----------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*------------------\n\n"**  **set list1 [list {/dev/wd0a 17086 10958 5272 68% /}\**  **{/dev/wd0f 179824 127798 48428 73% /news}\**  **{/dev/wd0h 1249244 967818 218962 82% /usr}\**  **{/dev/wd0g 98190 32836 60444 35% /var}]**  **foreach line $list1 {**  **regexp {[^ ]\* \*([0-9]+)[^/]\*(/[a-z]\*)} $line match size mounted;**  **puts "$mounted is $size blocks"**  **}**  **set line {Interrupt Vector? [32(0x20)]}**  **regexp "\[^\t]+\t\\\[\[0-9]+\\(0x(\[0-9a-fA-F]+)\\)]" $line match hexval**  **puts "Hex Default is: 0x$hexval "**  **Answer:**  Result: 1 match: here  Result: 1 Match: Where there 1: Where 2: there  New: Where there is a will, There is a lawsuit.  -----------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*------------------  / is 17086 blocks  /news is 179824 blocks  /usr is 1249244 blocks  /var is 98190 blocks  Hex Default is: 0x20  Discuss the following TCL Commands with an example.   * lindex * trimright & trimleft * exists * To open and close a file.   Write the output for the below two statements & comment on it.  set y [set x "def"] def  set y {[set x "def"]} [set x “def] | | | **[6]**  **[10]** | | 8. | A)  B) | **Create a TCL Script for creating a GUI window for List-box for selecting operation as shown below and explain each commands**  **Capture1**  **Answer:**  proc setLabel {color} {  .label configure -text $color -background $color  }  scrollbar .s -command ".l yview"  listbox .l -yscroll ".s set"  label .label -text "Nothing Selected"  bind .l <Double-B1-ButtonRelease> {setLabel [.l get active]}  grid .l -row 0 -column 0 -sticky news  grid .s -row 0 -column 1 -sticky news  grid .label -row 1 -column 0 -columnspan 2  .l insert 0 gray60 gray70 gray80 gray85 gray90 gray95 \  snow1 snow2 snow3 snow4 seashell1 seashell2 \  seashell3 seashell4 AntiqueWhite1 AntiqueWhite2 AntiqueWhite3 \  DarkSlateGray1 DarkSlateGray2 DarkSlateGray3 \  aquamarine4 DarkSeaGreen1 DarkSeaGreen2 DarkSeaGreen3 \  PaleGreen1 PaleGreen2 PaleGreen3 PaleGreen4 SpringGreen1 \  green3 green4 chartreuse1 chartreuse2 chartreuse3 \  chartreuse4 OliveDrab1 OliveDrab2 OliveDrab3 OliveDrab4 \  coral2 coral3 coral4 tomato1 tomato2 tomato3 tomato4 \  red4 DeepPink1 DeepPink2 DeepPink3 DeepPink4 HotPink1 \  HotPink2 HotPink3 HotPink4 pink1 pink2 pink3 pink4 \  PaleVioletRed2 PaleVioletRed3 PaleVioletRed4 maroon1 \  VioletRed4 magenta1 magenta2 magenta3 magenta4 orchid1 \  orchid2 orchid3 orchid4 plum1 plum2 plum3 plum4 \  DarkOrchid1 DarkOrchid2 DarkOrchid3 DarkOrchid4 purple1 \  MediumPurple3 MediumPurple4 thistle1 thistle2 thistle3 \  **Create a TCL/TK GUI window with buttons as shown below which does the following operation as follows if the hello button is presses it is changed to goodbye and vice versa. Whereas the quit button is pressed it is coming out of the process using TCL scripts.**  **Capture2**  **Answer:**  set text Hello  proc doIt {widget} {  global text  if {$text == "Hello"} {  set text "Goodbye"  } else {  set text "Hello"  }  $widget configure -text $text  }  button .b1 -text "Hello" \  -command "doIt .b1"  button .b2 -text "Quit" \  -command "destroy ."  grid .b1 -row 0 -column 0  grid .b2 -row 0 -column 1 | | | **[8]**  **[8]** | | 11. |  | **Write a Perl script to do the following operations**   * **Prompt the user to enter a sentence of atleast 5 words.** * **Break the string apart storing the individual strings in an array.** * **Print the original sentence to the screen with first word capitalized and all others in lowercase.** * **Print the last 3 letters of each word in the sentence to the screen** * **Join the second and third words into a string and store in a new variable.** * **Print this new variable to the screen in all uppercase letters except for the last letter** * **Prompt the user to enter their full name** * **Create a subroutine that returns a lowercase string comprised of the capitalized entered lastname and first initial as a single word (i.e “MARK TUCKER” would be returned as “tuckerm”) when passed the first and last name as arguments.** * **Prompt the user to enter the day of the week** * **Convert the day of the week to the first 3 letters in uppercase.** * **Print the weekday and the username to the screen with some descriptive text using one print statement.**   **Answer:**  print "Enter sentence with atleast 5 word \n";  chomp($sentence = <STDIN>);  @array = split(" ", $sentence);  $N=scalar(@array);  print "array: @array\n";  if ($N<5) {  print "I asked you to enter 5 word and you are entering just $N word\nAre you mad!";  } else {  $FC=$sentence;  $FC=~ s/^(\w+)/uc($1)/e;  print "FC: $FC\n";  $three = $sentence;  $three =~ s/(\w+)/substr($1,-3)/eg;  print "three: $three\n";  $joined=join("","$array[1]","$array[2]");  print "Joied: $joined\n";  $len=length($joined)-1;  $joined=~ s/(\w{$len})/uc($1)/e;  print "Modified joined: $joined\n";  print "Enter you full name: ";  chomp($name = <STDIN>);  $modname=username ($name);  print "Enter day of week: ";  chomp($day=<STDIN>);  $day=dayweek($day);  print "Name of user: $modname\nName of day: $day";  }  sub username {  $name=$\_[0];  if ($name=~ m/(\w+)\s+(\w+)/) {  $modname = join("",lc($2), substr(lc($1),0,1));  return $modname  }  }  sub dayweek {  $day = $\_[0]; v5r  $day =~ s/(\w{3})/uc($1)/e;  return $day  } | | | **[16]** | |  |  | Total Marks | | | **100** | | ⇔⇔⇔ | | | | | | | | | | | |