**B.M.S COLLEGE OF ENGINEERING**

(Autonomous College under VTU)

****

**Department of Electronics and Instrumentation Engineering**

**LABORATORY REPORT**

ON

**“DATA ACQUISITION AND VIRTUAL INSTRUMENTATION”**

SUBJECT CODE: 16EI6DCDVI

SUBMITTED BY

**STUDENT NAME: KARTHIK GUNALAN**

**USN: 1BM14EI006**

Faculty in charge,

**KUMUDA S**

Assistant Professor

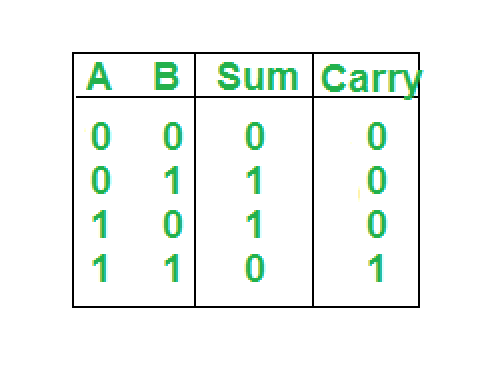
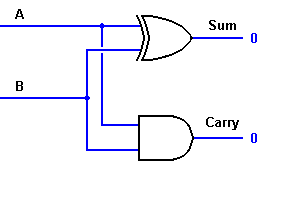
Department of Instrumentation Technology

BMSCE, Bengaluru-560019

**Experiment 1**

**Aim:** To Implement Half Adder Using Basic Gates.

**ALGORITHM:**



Step 1: Take input values from the User.

Step 2: A XOR gate is used to obtain the sum component of the two inputs.

Step 3: An AND gate is used to obtain the carry component of the two inputs.

Step 4: Display the two outputs on the Boolean Indicators.

Step 5: Select the pattern in the connector pane, assign input and output terminals.

Step 6: Save the VI.

**Front panel Implementation:**

Step1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel Place two boolean controller. Right click and go to Control palette >> Boolean>> boolean controller and name them as ‘input 0’ and ‘input 1’.

Step3: Similarly place two boolean indicators instead of controllers and name them as Sum and Carry.



**Block diagram Implementation:**

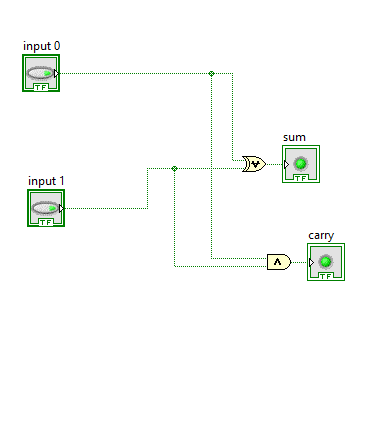
Step1: On the block diagram place one XOR and one AND gate. Right click and go to functions palette >> Boolean >> XOR/AND.

Step2**:** Connect controllers A and B to input of XOR and AND gate and connect their output to Sum and Carry respectively.

Step3: Save the VI and RUN it and observe the output on the front panel.

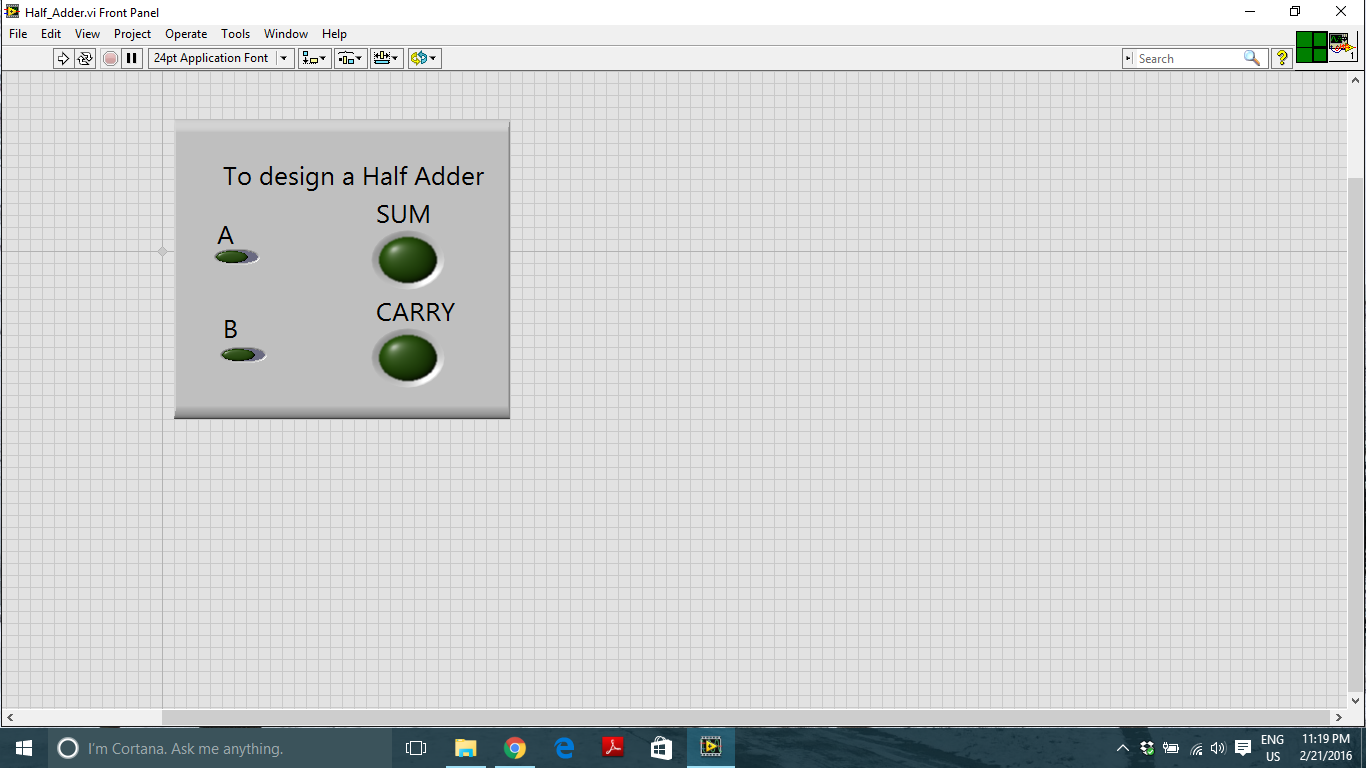
Step4: With continuous RUN mode the controller input can be varied continuously and output is observed.

Block diagram of the VI is as shown below.



**ICON/CONNECTOR PANE:**

* To use VI as sub VI in another program, it must have an icon and connector pane.
* Basically, it’s a graphical representation of a VI which contains images and texts.
* Connector pane is a set of terminals that corresponds to the controls and indicators of that VI.

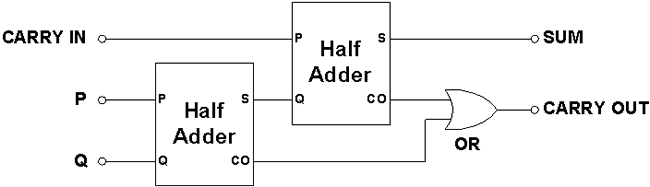


**Result:** The corresponding VI was built successfully and verified for different values.

**Experiment 2**

**Aim:** To Implement Full Adder Using Half Adder.

**ALGORITHM:**



Step 1: The two of the three inputs is given as input to a half adder.

Step 2: Its sum with the third input is given as input to the other half adder.

Step 3: The carry of both the half adders are given to ORgate and the output is displayed using a Boolean Indicator.

Step 4: The sum of the second half adder is the Sum of the Full adder and this is displayed using a Boolean Indicator.

Step 5: This logic is implemented by using two half adder subVI’s

**Front panel Implementation:**

Step1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel Place three boolean controllers. Right click and go to Control palette >> Boolean>> boolean controller and name them as ‘input 0’, ‘input 1’ and ‘input 2’.

Step3: Similarly place two boolean indicators instead of controllers and name them as Sum and Carry.



**Block diagram Implementation:**

Step1: On the block diagram place one OR gate. Right click and go to functions palette >> Boolean >> OR

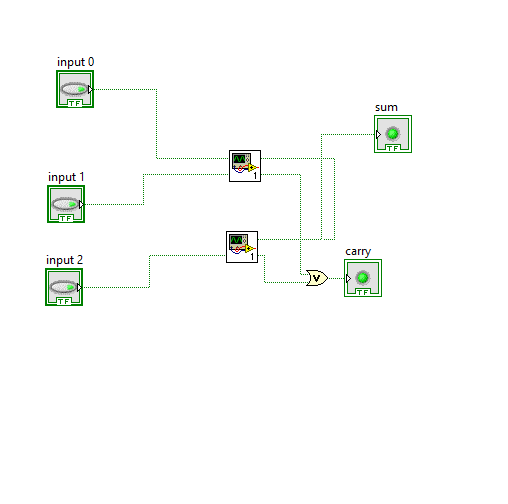
Step2: Place two Half adder sub VIs on the block diagram by going to function palette>> Select a VI and choose the Half\_Adder and select ok.

Step3: Wire the Half\_Adder sub VIs and the or gate.

Step4: Save the VI and RUN it and observe the output on the front panel.

Step5: With continuous RUN mode the controller input can be varied continuously and output is observed.

Block diagram of the VI is as shown below.

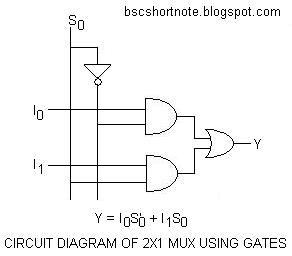


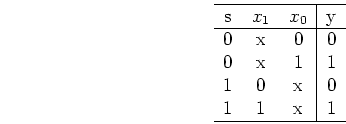
**Result:** The corresponding VI was built successfully and verified for different values.

**Experiment 3**

**Aim:** To Implement 2x1 Mux.

**ALGORITHM:**





Step 1: The output of a 2x1 mux is based upon the select line which chooses the input value that is necessary. This can be done by implementing the above logic circuit by using basic gates.

Step 2: These two outputs from the AND gate are sent to the OR gate and then the result is displayed.

Step 3: The 2x1 mux is to be created as a subVI for the 4x1 Mux. Hence, the icon/ connector pane is used. The pattern is selected in the connector pane and the input and outputs are assigned.

Step 4: The Icon pane can be used to give a specific design on the Icon.

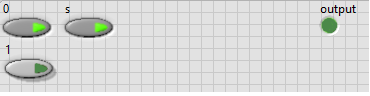
Step 5: Save the VI.

**Front panel Implementation:**

Step1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel Place three boolean controller. To do this go to Control palette >> Boolean>> boolean controller and name them as ‘0’, ‘1’ and ‘S’.

Step3: Similarly place one boolean indicator instead of controllers and name it as output.



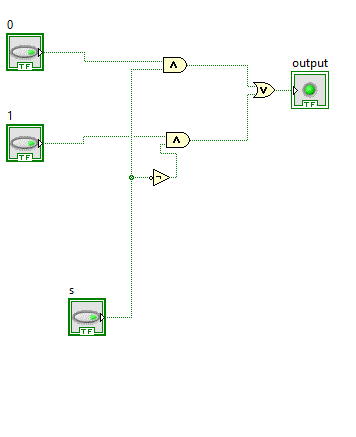
**Block diagram Implementation:**

Step1: On the block diagram place one OR gate two AND gates and one NOT gate. To do this go to functions palette >> Boolean >> OR/AND/NOT.

Step2: Wire the controllers and indicators to the gates as per the logic.

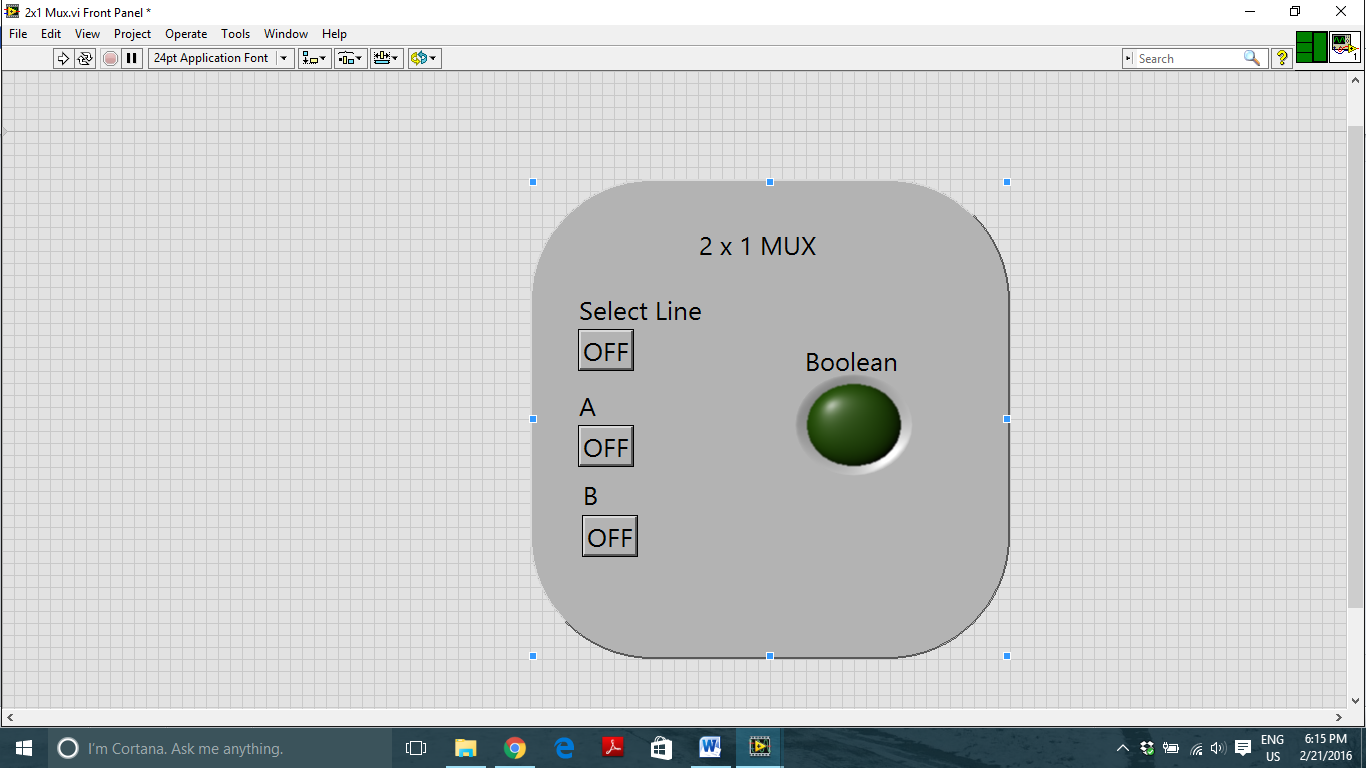
Step3: Save the VI and RUN it and observe the output on the front panel.

Step4: With continuous RUN mode the controller input can be varied continuously and output is observed.



**ICON/CONNECTOR PANE:**

* To use VI as sub VI in another program, it must have an icon and connector pane.
* Basically, it’s a graphical representation of a VI which contains images and texts.
* Connector pane is a set of terminals that corresponds to the controls and indicators of that VI.

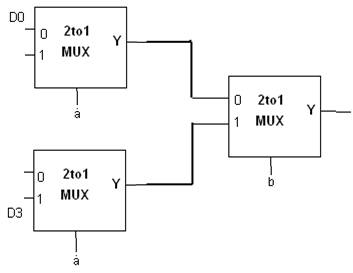
3

**Result:** The corresponding VI was built successfully and verified for different values.

**Experiment 4**

**Aim:** To Implement 4x1 Mux Using 2x1 mux.

**ALGORITHM:**



Step 1: The Input lines 00,01,10 and 11 are selected based on the select lines s1 and s0.

Step 2: The input lines 00 and 01 are sent to the subVI 2x1 Mux and s0 is its select line. Similarly, input lines 10 and 11 are sent to the subVI 2x1 Mux and s0 is its select line.

Step 3: The outputs from these two subVIs is sent to one more subVI and its select line will be s1.

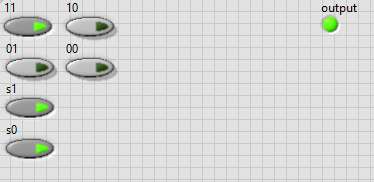
Step 4: The output from this third subVI will be our output. The logic followed is as shown in the truth table above.

**Front panel Implementation:**

Step1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel Place six boolean controller. To do this go to Control palette >> Boolean>> boolean controller and name them as ‘00’, ‘01’, ‘10’, ‘11’, ‘S0’ and ‘S1’.

Step3: Similarly place one boolean indicator instead of controllers and name it as output.



**Block diagram Implementation:**

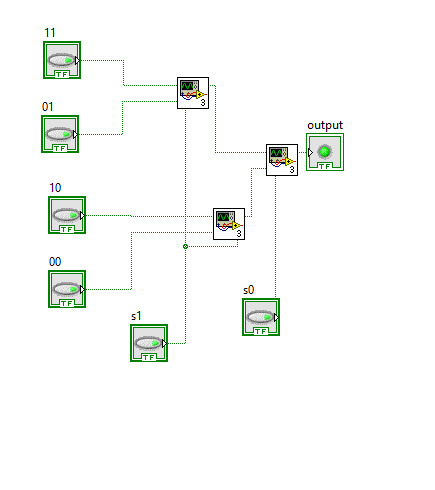
Step1: On the block diagram place three 2x1 mux sub VIs on the block diagram by going to function palette>> Select a VI and choose the Mux\_2x1 and select ok.

Step2: Wire the Mux\_2x1 sub VIs to their respective controllers and indicators.

Step3: Save the VI and RUN it and observe the output on the front panel.

Step4: With continuous RUN mode the controller input can be varied continuously and output is observed.

Block diagram of the VI is as shown below.



**Result:** The corresponding VI was built successfully and verified for different values.

**Experiment 5**

**Aim:** To create a VI which changes the the colour of the Boolean indicator N number times of between true and false.

**ALGORITHM:**

Step1: The numeric input is given to the for loop and specified number of times the output led should blink either true or false.

Step 2: to toggle the led the iteration values are checked for even and odd continuously, which is alternative in nature.

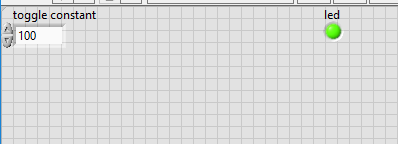
Step 3: the output of the comparator for even and odd is sent to a led and hence the logic is implemented.

**Front panel Implementation:**

Step1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel Place a numeric controller and right click, go to Control palette >> numeric>> numeric controller and name them as input.

Step3: Similarly place one Boolean indicator to view the output on front panel.



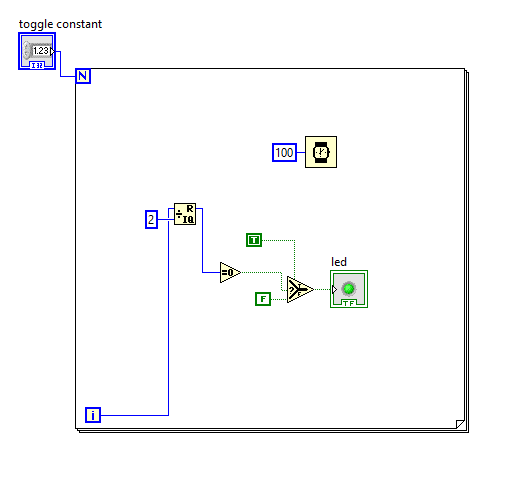
**Block diagram Implementation:**

Step 1: On the block diagram, create a for loop from the functions palette. Structures>>For loop.

Step 2: the iteration value is checked for even and odd and the output of which is sent to a ternary operator

Step 3: the ternary operator sets the value true or false and correspondingly sets the output which is connected to a led

Step 4: a wait function is also implemented, because during execution the toggling of led is distinct able by the human eye.



**Result:** Number of times the colour of Boolean indicator has changed is observed.

**EXPERIMENT 6**

**Aim:** To find a factorial of a given number.

**ALGORITHM:**

Step 1: The Number who’s factorial has to be found is entered. The same number is set as the number of counts in the for loop.

Step 2: Shift registers are used to store the previous values and then the previous and present values are multiplied until it reaches the ‘N’ value. This gives the factorial.

Step 3: In the while loop, The feedback node is used to store the previous values. The feedback node has to be initialized to 0.

Step 4: The stop logic for the while loop is when the number of iterations of the loop equals the number entered by the user.

Step 5: The factorial of the number is displayed using Numeric Indicators in both the cases.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel place one Numeric controller. To do this go to Control palette >> Numeric>> Numeric controller and name it as ‘Number’.

Step3: Similarly place one Numeric indicator instead of controllers and name it as ‘Factorial of Number’.

Step 4: Create another Numeric indicator to show the output which results after loop.



**BLOCK DIAGRAM:**

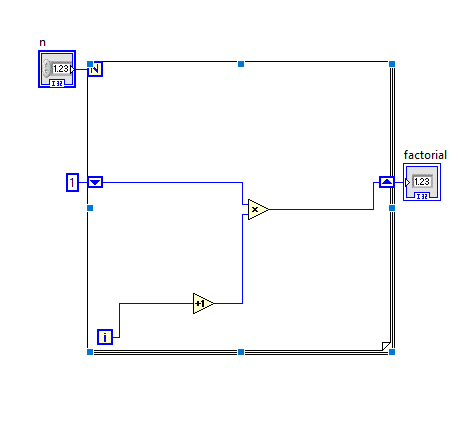
Step 1: On the block diagram, create a for loop from the functions palette. Structures>>For loop.

Step 2: Include a add and multiply from the functions palette>>Numeric>>Add/Multiply.

Step 3: Create a shift register also and connect the components as shown in the block diagram below.

Step 4: Save the VI and RUN it and observe the output on the front panel.

Step 5: With continuous RUN mode the controller input can be varied continuously and output is observed.



**Result:** The factorial of the number was found.

**EXPERIMENT 7**

**Aim:** To build aVI to generate a Fibonacci series.

**ALGORITHM:**

Step 1: The length of the Fibonacci series which is to be generated is specified by placing a numeric controller on the front panel.

Step 2: Shift registers are created and are used to store the previous values. One more left shift register is added to the for loop as we need to store the two previous values which are to be added up.

Step 3: both left shift registers are initialised with 0 and 1 respectively.

Step 4: The values stored in shift registers are added up and given to the numeric indicator.

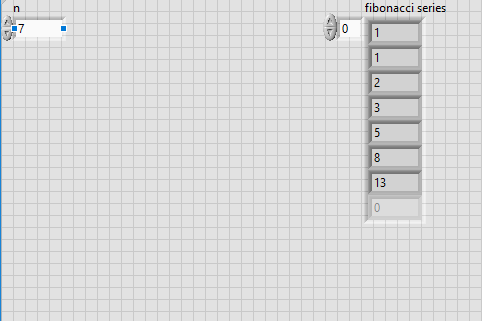
Step 5: The Fibonacci series is displayed on the front panel by numeric indicator for the specified length(N).

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel, place one Numeric controller by right click and go to Control palette >> Numeric>> Numeric controller and name it as ‘Input’.

Step3: Similarly place one Numeric indicator instead of controllers and name it as ‘array output(Fibonacci series)’.



**BLOCK DIAGRAM:**

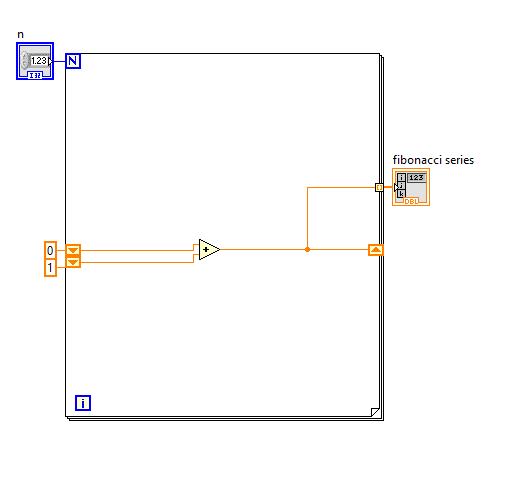
Step 1: On the block diagram, create a for loop from the functions palette. Right click>> Structures>>For loop.

Step 2: place a adder from the functions palette>>Numeric>>Adder.

Step 3: Create a shift registers, also initialise them and connect the components as shown in the block diagram below.

Step 4: Save the VI and RUN it and observe the output on the front panel.

Step 5: With continuous RUN mode the controller input can be varied continuously and output is observed.



**Result:** Fibonacci series is generated and observed for the required different length(N) on the front panel.

**EXPERIMENT 8**

**AIM:**  To build a VI to find the sum of N natural numbers.

**ALGORITHM:**

Step 1: The number of natural numbers whose sum is to be found is specified on the front panel.

Step 2: Shift registers are created and are used to store the previous values. Initialisation is done to the shift register.

Step 3: As shift registerd will be initialised to 0,the value of the first iteration(i) should be starting from 1.hence an incrementor is placed to increment the values in each iteration.

Step 4: The values stored in shift registers and new value from each iteration are added up and given to the numeric indicator.

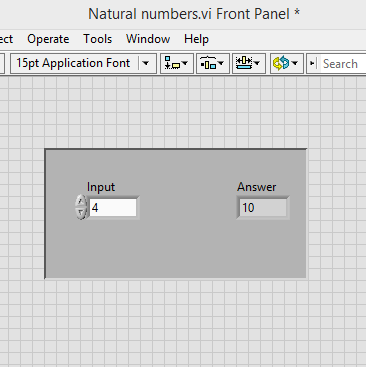
Step 5: The sum of specified length(N) of natural nymbers is displayed on the front panel by numeric indicator.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel, place one Numeric controller by right click and go to Control palette >> Numeric>> Numeric controller and name it as ‘Input’.

Step3: Similarly place one Numeric indicator instead of controllers and name it as ‘Answer’.



**BLOCK DIAGRAM:**

Step 1: On the block diagram, create a for loop from the functions palette. Right click>> Structures>>For loop.

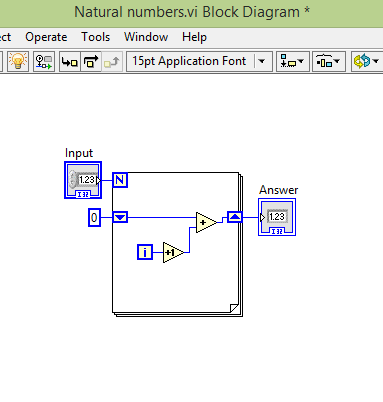
Step 2: place an incrementor to increment i from the functions palette>>Numeric>>incrementor.

Step3:place an adder from the functions palette>>Numeric>>Adder.

Step 4: Create a shift registers, also initialise them and connect the components as shown in the block diagram below.

Step 5: Save the VI and RUN it and observe the output on the front panel.

Step 6: With continuous RUN mode the controller input can be varied continuously and output is observed.



**Result:** The sum of N natural numbers is generated on the front panel.

**EXPERIMENT 9**

**AIM:** To build a VI to display numbers from 0 to 9 on seven segment display LED

**ALGORITHM:**

Step 1: The number which we need to display on seven segment LED is given from numeric controller on the front panel.

Step 2: case structure is created from function palette and we can create as many cases we need(to be displayed from 0 to 9).

Step 3: place seven Boolean indicators on the front panel named as A,B,C,D,E,F and G.

Step 4: consider the each case and place seven true/false functions which are required to display the specified number given by the user and connected to respective indicators.

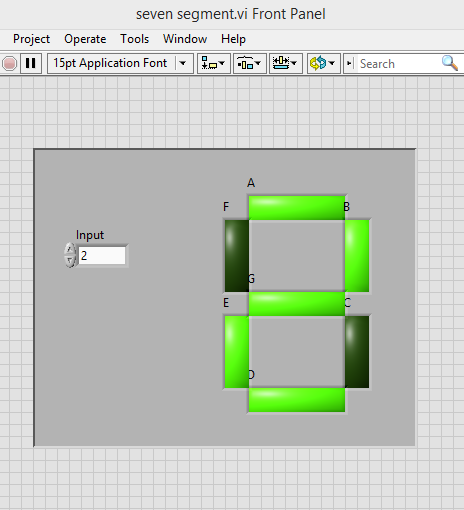
Step 5: Now observe the output in the seven segment LED’s on front panel for the given input number.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step2: On the front panel, place one Numeric controller by right click and go to Control palette >> Numeric>> Numeric controller and name it as ‘Input’.

Step3: Similarly place seven boolean indicators as seven segment display and name them as A,B,C,D,E,F and G.



**BLOCK DIAGRAM:**

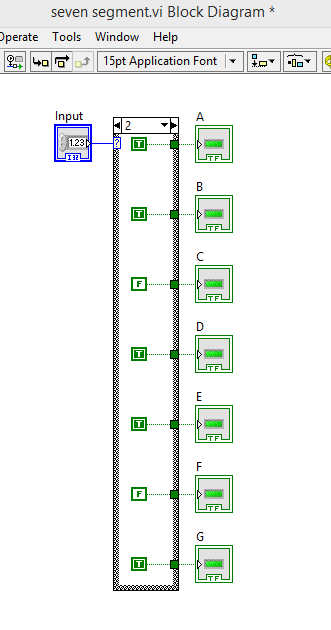
Step 1: On the block diagram, create a case structure from the functions palette. Right click>> Structures>>Case structure.

Step 2: Create 10 cases to display 10 digits from 0 to 9.Input number which is to be displayed is given to the case structure.

Step3: Also consider the each case and place seven true/false functions which are required to display the specified number given by the user and connected to respective indicators.

Step 4: Save the VI and RUN it and observe the output on the front panel.

Step 5: With continuous RUN mode the controller input can be varied continuously and output is observed.



**Result:** The output is observed in the seven segment LED display for the given input number.

**EXPERIMENT 10**

**AIM:** To create a VI to find the sum of positive and negative numbers in an array.

**ALGORITHM:**

Step 1: The Array elements are first checked for positive or negative.

Step 2: If the element is positive, it enters the case structure in True case and gets added with the positive sum.

Step 3: Similarly, if the element is negative, it enters the case structure in False case and gets added with the negative sum.

Step 4: This is done for all the elements in an array, using a For loop.

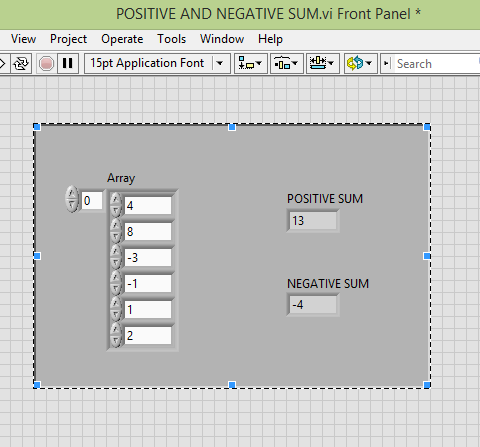
Step 5: Sum of positive numbers and negative numbers is found and displayed.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step 2: Create a Numeric array and initialize the array.

Step 3: Create two Numeric Indicators, One for negative sum and the other for Positive Sum.

****

**BLOCK DIAGRAM:**

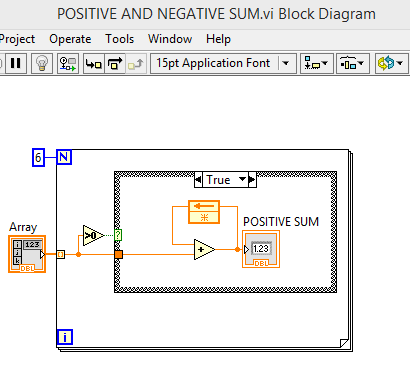
Step 1: Create a For loop from the Function palette.Right click>>Structures Palette>>for loop

Step 2: Connect the initialized array from the outside through a tunnel to a greater than equal to 0 function.

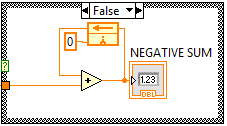
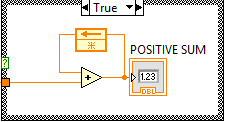
Step 3: This output is then connected to a case structure.

Step 4: If the case is true then the element is added to the positive sum, using feedback loop.

Step 5: If the case is false then the element is added to the negative sum, using feedback loop.

****

**Case structures:**



**Result:** The Sum of Positive numbers and Negative numbers was found and displayed.

**EXPERIMENT 11**

**AIM:** To check if the elements of an array are odd or even.

**ALGORITHM:**

Step 1: Input array of elements is created and Initialise the array before running the program.

Step 2: The each array elements are now checked if they are divisible by 2 or not.

Step 3: The Elements whose’s remainders are 0 after division is passed on to the ‘True’ case of the Case structure and the output LED for that element is made to glow.

Step 4: The Elemets whoose’s remainders are not 0 after division is passed on to the ‘False’ case of the Case structure and LED for that element should not glow.

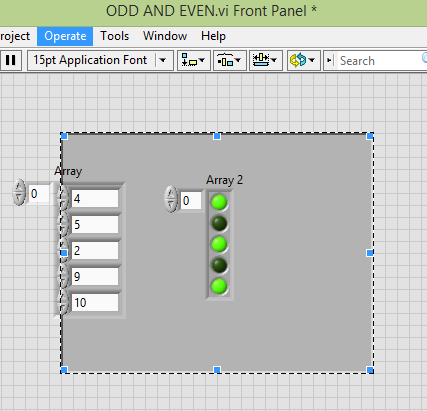
Step 5: The output LED array for the given input array elements is displayed.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step 2: Create a Numeric array and initialize the array.

Step 3: Create a Boolean array to display the output for each element of the array.



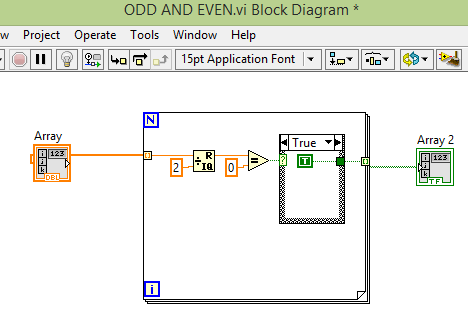
**BLOCK DIAGRAM:**

Step 1: Create a For loop from the Structures Palette.

Step 2: Connect the initialized array from the outside through a tunnel to a quotient and remainder function. Divide each element with 2.

Step 3: Compare the Remainder with 0.

Step 4: Use a case structure to decide if the element is odd or even.



**Result:** The Elements entered were displayed as odd or even in the output LED array on the front panel.

**EXPERIMENT 12**

**AIM:** To find square roots of a given quadratic equation.

**ALGORITHM:**

Step 1: Input array of elements is created and Initialise the array before running the program.

Step 2: The each array elements are now checked if they are divisible by 2 or not.

Step 3: The Elements whose’s remainders are 0 after division is passed on to the ‘True’ case of the Case structure and the output LED for that element is made to glow.

Step 4: The Elemets whoose’s remainders are not 0 after division is passed on to the ‘False’ case of the Case structure and LED for that element should not glow.

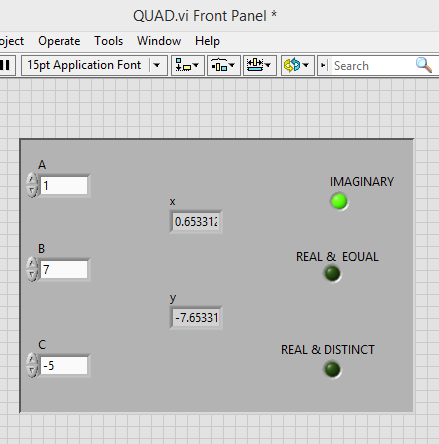
Step 5: The output LED array for the given input array elements is displayed.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step 2: Create a Numeric array and initialize the array.

Step 3: Create a Boolean array to display the output for each element of the array.

****

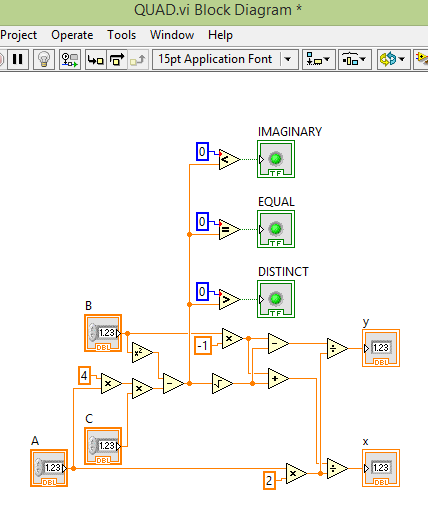
**BLOCK DIAGRAM:**

Step 1: Create a For loop from the Structures Palette.

Step 2: Connect the initialized array from the outside through a tunnel to a quotient and remainder function. Divide each element with 2.

Step 3: Compare the Remainder with 0.

Step 4: Use a case structure to decide if the element is odd or even.

****

**EXPERIMENT 13**

**AIM:** To find whether a given number is prime or composite by using a flat sequence

**ALGORITHM:**

Step 1:create a flat sequence and divide into two frames.

Step 2: in the first frame create a loop to input a number and to check if the number is prime or composite.

Step 2: check if the number is divisible by all the numbers starting from 1.

Step 3: if the number is divisible then the remainder is 0.

Step 4: count the number of 0’s for each iteration.

Step 5: the output count is obtained in the first frame and sent to the second frame where the output whether the number is prime or not is displayed .

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step 2: Create a Numeric constant and a Boolean indicator.



**BLOCK DIAGRAM:**

Step 1: create a flat sequence from structures

Step 2: create two frames, in the first frame create a for loop from structure.

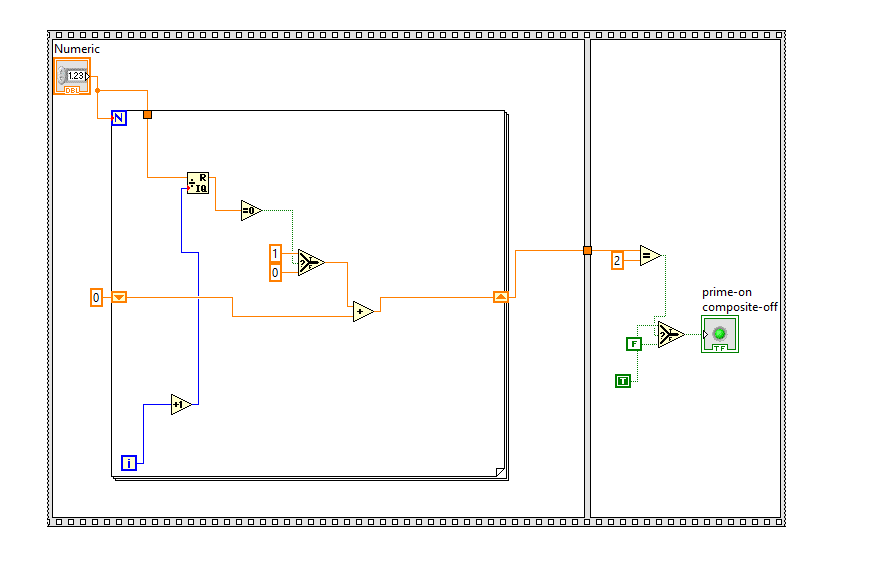
Step 3: a numeric controller is used to set the loop count and also as a constant to check whether the number is prime or not.

Step 4: the iteration value is incremented by 1 so that 0 is skipped and the number is divided by the iteration value.

Step 5: an equal to 0 function is implemented from the comparison function pallete which is used to check if the remainder is 0.

Step 6: a shift register is used to keep adding the number of 0’s whenever true and the output is sent to the second frame

Step 7: in the second frame the count variable is checked if it is equal to 2 or not so as to indicate whether the number is prime or composite



**Experiment 14**

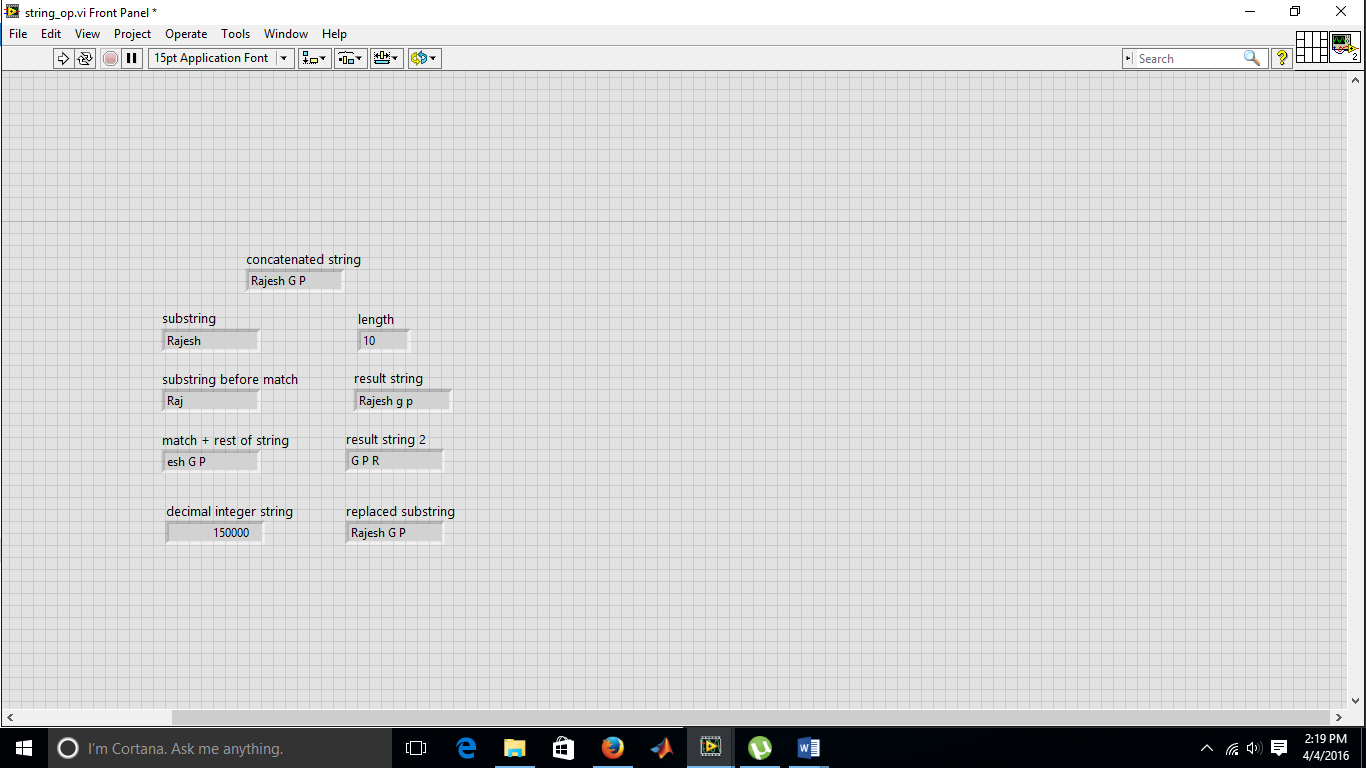
**AIM:** To explore the various functions of the String palette.

**FRONT PANEL:**

Step 1: Open a new VI to create blank front panel and block diagram.

Step 2: Create a string controller to give the Input string.

Step 3: Create indicators for the various outputs of the string functions.



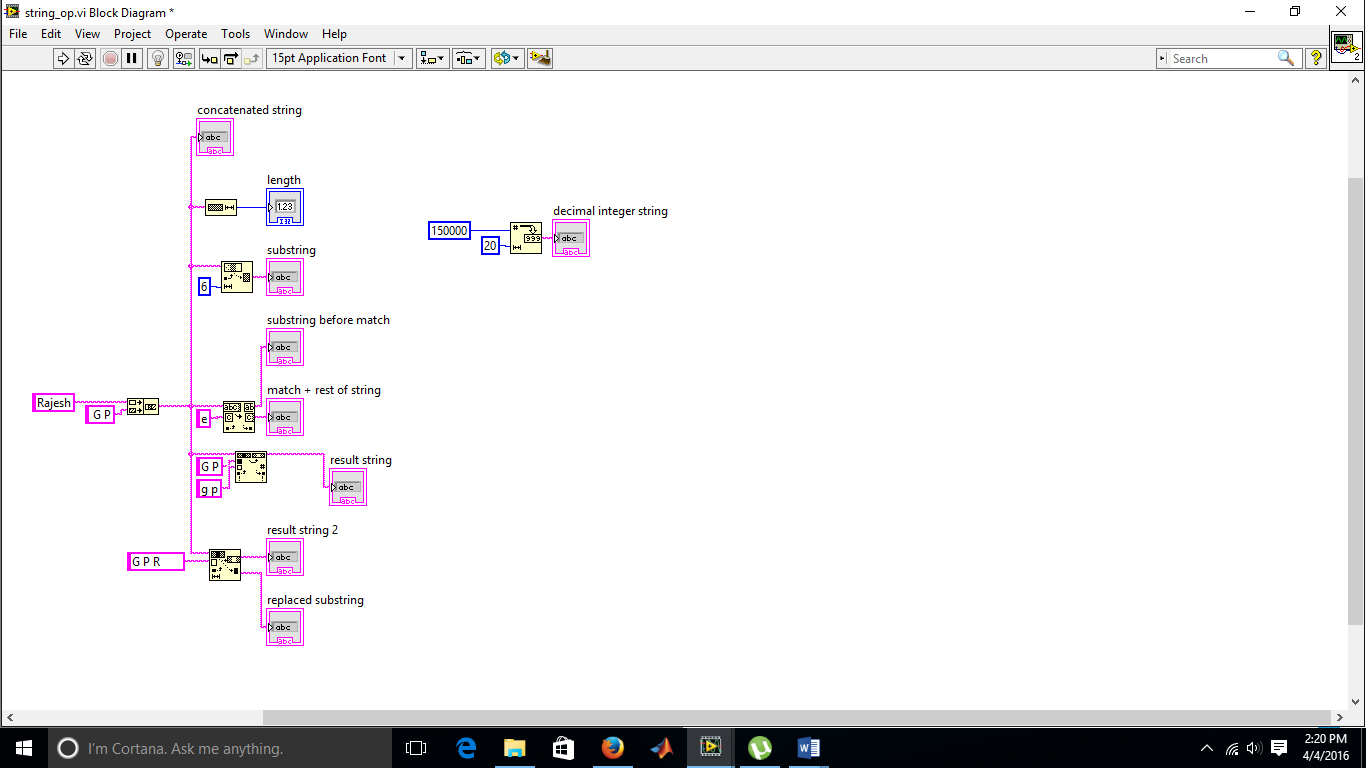
**BLOCK DIAGRAM:**

Step 1: Use the String functions ‘String Subset’to get the subset of a string who’s length is specified as 5.

Step 2: Use the String function ‘Replace Substring’ and specify the substring, offset and the length.

Step 3: Use the String function ‘Format into String’and specify the input and the format.

Step 4: Use indicators to display outputs from all these functions.



**RESULT:** The functions and its working were explored.

**SIGNATURE: \_\_\_\_\_\_\_\_\_\_\_**