**#Instructions**

* **Please make a copy before you edit it: File -> Make a copy.**
* **Please find the problem statement and detailed template below.**
* **From where the template starts you will be allowed only 3 pages for the solution summary**
* **Please submit the final solution document with an access link in the** [**submission form**](https://forms.gle/oLixmWr6ARbFEkfS9)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Girl Hackathon**  **[Do not edit this section. This is read-only]**   |  |  |  |  | | --- | --- | --- | --- | | **Problem Statement:**  As discussed in workshops, manufactured chips may have structural faults at certain places/nodes, which must be tested before being delivered to end users.  The task is to design an algorithm and write its code to identify the input vector required to identify the fault at a given node in a given circuit. In a case, there would only be a single fault in the design.  The algorithm should be efficient, robust and able to identify faults quickly. Inputs Available inputs are -   1. Circuit file (format provided below) 2. Fault node location 3. Fault type    1. SA0 : stuck-at-0, a fault where node is not able to attain value 1, irrespective of inputs    2. SA1 : stuck-at-1, a fault where node is not able to attain value 0, irrespective of inputs  Outputs The code should print a vector for inputs to test the fault, and the expected value of output to confirm the fault.  The output should be printed to the following file in run directory - output.txt Circuit Format  1. The circuit will have 4 inputs - A, B, C and D. All of which are boolean type (only 0 and 1 are valid inputs) 2. The circuit’s output will always be Z which is also a boolean. 3. The circuit will be built using the following operations -    1. AND ( & ) gate    2. OR ( | ) gate    3. NOT ( ~ ) gate    4. XOR ( ^ ) gate 4. The circuit would purely be a combinational logic. 5. All internal nodes in the circuit would be named as : “net\_<alphanumeric string>” 6. Each input ( A / B / C / D ) would be utilized only once in the circuit.  Example Input: Circuit File:   |  | | --- | | net\_e = A & B  net\_f = C | D  net\_g = ~ net\_f  Z = net\_g ^ net\_e | | | |   Fault:   |  | | --- | | FAULT\_AT = net\_f  FAULT\_TYPE = SA0 |  Example Output:  |  | | --- | | [A, B, C, D] = [0, 0, 0, 1], Z = 1 | |   **Participants need to work on the above problem statement and provide the solution for the same. Goodluck!** |

|  |
| --- |
| Submission: Participants are required to create a PDF document as the final submission. The document should contain the link to a public GitHub repository (accessible and open to all).  The repository should have all the collaterals of the code, along with a README file. The code can be written in any open-source programming language using standard open-source libraries.  The README file should cover how to generate the environment needed to run the code, how to run the code, and any other necessary information.  The document should also cover the following aspects:   1. The approach used to generate the algorithm. 2. Proof of Correctness. 3. Complexity Analysis. |

|  |
| --- |
| Evaluation Criteria:  1. The coverage and correctness of the algorithm across different circuit designs. 2. Space Complexity: extra space used by algorithm to generate the input vector needed. 3. Time Complexity: Time used by the algorithm to generate the input vector required.   Evaluation Rubrics:   * Code Quality (20%) * Circuit Design (10%) * Algorithm Design (50%) - Correctness and Time & Space Complexity * Testing (20%) |

**Find Template to use below**

(3 Pages Maximum from the template below)

|  |
| --- |
| **2023 Girl Hackathon Ideathon Round: Solution Submission** |
| Project Name: Algorithm to identify fault at single node of a circuit. |
| Pariticipant Name: Ritika Sharma |
| Participant Email ID: ritikadayama |
| ReadMe File Links (Eg Github): |
| **Summary:**  The assignment is to create an algorithm and develop its code to locate a structural flaw at a certain node in a specific circuit. The circuit file, the position of the problem node, and the fault type (SA0 or SA1) are the accessible inputs. The circuit is represented as a file format.  **Solution:** A C++ example implementation is provided in the provided code. It analyses the circuit using input vectors after reading the circuit file to find the desired problem. The procedure begins with a starting input vector, adjusts the value of the fault node as necessary, and then assesses the circuit. The code prints the test inputs and the anticipated output if the issue is found. If the error is not found, the algorithm loops through the input vectors, flipping each individual input until the error is found. The "output.txt" file is updated with the final test inputs and the anticipated output. |
| **Problem Statement:**  Reading the circuit file and building the circuit using the given logic gates constitutes the solution. The algorithm sets the fault node value based on the type of defect, initialises an input vector, and evaluates the circuit. The algorithm prints the test inputs and anticipated result if the error is found. If the error is not found, it iterates through the input vectors, flipping each individual input until it is. An output file is created with the final test inputs and predicted output.  The technique is made to locate defects quickly and reliably in a particular circuit, supplying useful data for quality control and debugging. |
| **The approach used to generate the algorithm.**  I utilised a typical method for fault identification in digital circuits when creating the algorithm in the previous responses. To comprehend the circuit architecture and the relevant logic gates, the algorithm first scans the circuit file. Following that, it adjusts the fault node value to reflect the fault type provided and analyses the circuit using input vectors.  The output is examined by the algorithm to see if the fault has been found. If not, it repeatedly flips one input at a time through the input vectors to test various combinations and locate the problem. This procedure keeps going until either the error is found, or all input vectors have been examined.  The strategy seeks to effectively identify the defect by reducing the quantity of necessary test inputs. The search is guided by the logic of the circuit as well as the traits of the fault type (stuck-at-0 or stuck-at-1).  While the algorithm is based on standard fault detection principles, it was important to keep in mind that the actual implementation and efficiency may vary depending on the particular circuit characteristic, fault types, and other factors. |
| **Proof of Correctness.**  While the algorithm is designed based on standard fault detection techniques, it's important to thoroughly test and validate the implementation for specific circuit files and fault scenarios. Rigorous testing and verification can help ensure the correctness of the algorithm and address any potential edge cases or limitations.  **Design Specification:** A precise specification outlining the desired behaviour and needs should serve as the basis for the algorithm's design. This includes being aware of the fault kinds, circuit structure, and intended fault detection results.  **Test Cases:** A variety of fault scenarios and circuit configurations should be covered by a collection of test cases. Both scenarios of normal operation and cases with known flaws should be included in these test cases.  **Expected Output:** The expected output for each test case should be identified either through human analysis or by consulting findings that are known to be accurate. Understanding circuit behaviour and fault consequences is necessary for this.  **Validation of the algorithm's implementation:** Using the specified test cases, the algorithm's implementation should be extensively tested. The algorithm's actual results are contrasted with its predicted results. If they regularly match across various test situations, there is some assurance that the algorithm is valid.  **Verification & Validation:** Experts or peers in the fields of defect detection and digital circuits should review and validate the algorithm. They can appraise the test cases, the effectiveness of the implementation, and the design of the algorithm.  **Formal Analysis:** To mathematically analyse the algorithm's correctness in some circumstances, formal techniques like formal verification or model checking may be used. These techniques entail exacting mathematical justifications and in-depth examinations of the behaviour of the circuit.  Provided below the compiler input and output statement. |
| **Complexity Analysis**  The major operations and their frequency within the algorithm must be considered in order to conduct a complexity analysis of the aforementioned method. Let's dissect the algorithm into its constituent parts:  The size of the file and the method used to read it both affect how difficult it is to read the circuit file. The temporal complexity could be regarded as O(m) if the circuit file contains ‘m' lines.  Circuit Evaluation: The circuit evaluation includes determining the values of the net nodes and the output as well as evaluating the logic gates. This procedure normally involves completing the corresponding logic gate operations for each line in the circuit file as you iterate through the circuit file. The time complexity for circuit evaluation could be O(n) if the circuit file has 'n' lines.  Until the defect is found, several input vectors are systematically tested as part of the fault detection process. The method examines the circuit while going over each input vector one at a time to look for errors. The fault detection process's temporal complexity could be regarded as O(k) if the input vector count is 'k'.  Considering these key elements, the algorithm's total time complexity would be the sum of the time complexities of each element:  **Time complexity overall: O (m + n + k)** |
| **Alternatives considered** |
| **References and appendices**  **Attaching the circuital diagram using draw.io.** |