Variable Dependency Extractor Development

Rajesh Sangwan (B20CS060)

Cone of Influence :

The Cone of Influence refers to the set of variables that can be influenced by a given variable within a digital circuit. It represents the variables whose values or behavior can be directly or indirectly affected by the given variable. In other words, the COI captures the downstream impact of a variable within a circuit.

For example, if there is a signal A that drives the inputs of multiple gates, and those gates further drive other gates and signals in the circuit, the COI of signal A would include all the gates and signals that the changes can influence in signal A.

The COI is useful for tasks such as determining the fanout of a signal, identifying potential timing or functionality issues, and optimizing the circuit.

Cone of Dependence :

The Cone of Dependency, on the other hand, refers to the set of variables that directly or indirectly influence a given variable within a digital circuit. It represents the variables that contribute to or determine the value or behavior of the given variable. In other words, the COD captures the upstream dependencies of a variable within a circuit.

For example, if there is a signal Z that is driven by multiple inputs, and those inputs are dependent on other gates and signals in the circuit, the COD of signal Z would include all the gates and signals that contribute to the value of signal Z.

The COD is useful for tasks such as determining the sources of data or control signals for a specific variable, identifying the inputs or conditions that affect its behavior, and analyzing the dependencies within the circuit.

Code Explanation in Detail:

- The code includes necessary C++ libraries and defines various sets, maps, and stacks to store variables, operators, keywords, and COI information.
- There are several utility functions to check if a string represents a variable, operator, keyword, or number.

read_line_COI function :

- It takes a temp_line as input, which represents a line of code.
- The function initializes various variables and data structures, including curr (an empty string), n (the size of the input line), lhs (a boolean flag indicating whether the current token is on the left-hand side of an assignment), and vectors for storing left-hand side and right-hand side variables.
- The function tokenizes the input line by iterating through each character and building tokens based on spaces and operators. It stores the tokens in the line vector.
- Next, the function iterates through each token in the line vector.
- If the token is the "module" keyword, the function returns, indicating the end of processing for the current line.
- If the token is a semicolon (";") and the search_begin flag is set to true and the semi_colon is equal to 1, it indicates the end of a block. The function decreases semi_colon, decrements curr_depth, and updates the dependency count for variables in the current block.
- If the token is a left parenthesis ("("), the balanced variable is incremented to keep track of balanced parentheses.
- If the token is a right parenthesis (")"), the balanced variable is decremented. If the balanced variable becomes zero and the ready_to_push flag is set to true, it indicates the end of a block, and the function pushes the current block to the blocks stack.
- If the token is an equals sign ("="), it indicates that the following variables are on the right-hand side of an assignment.
- If the token is a comma (",") or semicolon (";"), it indicates the end of a variable assignment. The function updates the COI map by inserting dependencies between the left-hand side and right-hand side variables.
- If the token passes the variable_check function, it is identified as a variable and added to the variable set.
- If the token passes the number_check function, it is identified as a number and added to the number set.
- If the token is a C++ keyword, such as "if", "else", or "begin", the function handles the corresponding block logic by updating flags and performing necessary operations.

- After processing all the tokens, the function checks if any left-hand side variables are still pending for insertion into the COI map and perform the insertion.

---

read_line_COD function:

- The read_line_COD function is similar to the read_line_COI function, but instead of updating the COI (Connections of Interest) map, it updates the COD (Connections of Declaration) map.
- If the token is a comma (",") or semicolon (";"), it indicates the end of a variable assignment. The function updates the COD map by inserting connections between the left-hand side and right-hand side variables.

COI_bfs function:

- The function initializes a string ans to store the result and a set visited to keep track of visited variables.
- If the given variable s is not found in the variable set, it means the variable is not present in the code file. In this case, the function sets ans to "Variable not found in file" and prints the result.
- If the given variable s is found in the variable set but has no dependencies in the COI map, it means the variable has no other variables depending on it. In this case, the function sets ans to "No dependencies" and prints the result.
- If the given variable s exists in the COI map and has dependencies, the function starts the BFS process.
- It creates a queue qu and adds the starting variable s to the queue. It also adds a special token "module" to indicate the end of a level.
- The variable current_level is set to 1 to keep track of the current level during the BFS process.
- The while loop continues until there is more than one element in the queue.
- If the front element of the queue is not a "module", it means it is a variable to process. The function retrieves the front element, adds it to the ans string (excluding the starting variable s), and removes it from the queue.
- For each dependency i of the current variable curr, if i has not been visited before, it is added to the visited set and pushed into the queue qu.
- If the front element of the queue is "module", it indicates the end of a level. The function removes this special token from the queue, increments the current_level, and adds a level indicator string to the ans string.

- If the queue still contains elements, it means there are more variables to process, so the level indicator is added to the ans string.
- The loop continues until there is only one element left in the queue, which is the special token "module" indicating the end of the BFS process.
- Finally, the ans string containing the variable dependencies organized by levels is printed.

COD_bfs function:

The COD_bfs function performs a breadth-first search (BFS) on the dependencies stored in the COD (Connections of Declaration) map. It starts the BFS from a given variable s and prints the dependencies in a level-wise manner.

- The function initializes a string ans to store the result and a set visited to keep track of visited variables.
- If the given variable s is not found in the variable set, it means the variable is not present in the code file. In this case, the function sets ans to "Variable not found in file" and prints the result.
- If the given variable s is found in the variable set but has no dependencies in the COD map, it means the variable has no dependencies. In this case, the function sets ans to "No dependencies" and prints the result.
- If the given variable s exists in the COD map and has dependencies, the function starts the BFS process.
- It creates a queue qu and adds the starting variable s to the queue. It also adds a special token "module" to indicate the end of a level.
- The variable current_level is set to 1 to keep track of the current level during the BFS process.
- The while loop continues until there is more than one element in the queue.
- If the front element of the queue is not "module", it means it is a variable to process. The function retrieves the front element, adds it to the ans string (excluding the starting variable s), and removes it from the queue.
- For each entry i in the COD map, the function checks if the current variable curr is found in the set of dependencies for that entry. If it is, the dependent variable dependent_var is added to the visited set and pushed into the queue qu.
- If the front element of the queue is "module", it indicates the end of a level. The function removes this special token from the queue, increments the current_level, and adds a level indicator string to the ans string.

- If the queue still contains elements, it means there are more variables to process, so the level indicator is added to the ans string.
- The loop continues until there is only one element left in the queue, which is the special token "module" indicating the end of the BFS process.
- Finally, the ans string containing the variable dependencies organized by levels is printed.

---

removeComments function:

- The provided code defines a function removeComments that takes a string fileContent as input and removes both single-line and multi-line comments from it.

Main function:

- The program prompts the user to enter the name of a Verilog file.
- The file is opened, and if there's an error opening it, an error message is displayed, and the program exits.
- The content of the file is read into a string variable called fileContent.
- A function called removeComments is called with fileContent as an argument. This function is not shown in the provided code but is likely intended to remove comments from the Verilog code.
- The temp_file string is initialized with the modified file content after removing comments.
- The line vector is created to store individual lines of the Verilog code.
- The code iterates through each character of temp_file to parse the lines and separate them into the line vector.
- Various flags (last_space, line_start, last_semi) are used to determine the state of parsing.
- The parsed lines are stored in the line vector.

- The read_line_COD function is called for each line in line to process the Control of Data dependencies.
- The read_line_COI function is called for each line in line to process the Control of Information dependencies.
- The variables present in the Verilog file are printed to the console.
- The program prompts the user to enter the name of a variable for which they want to see dependencies.
- The COI_bfs function is called with the specified variable name to calculate the Control of Information dependencies for that variable.
- The COD_bfs function is called with the specified variable name to calculate the Control of Data dependencies for that variable.
- The results of the COI and COD calculations are printed on the console

---

Output Results:

```
Enter the name of the Verilog file: sample_input5.v
Below is the list of variables present in verilog file:
ACTIVE DATA HEMANT IDLE TOKEN USBF_ASYNC_RESET USBF_T_PID_ACK USBF_T_PID_DATA0 USBF_T_PID_DATA1 USBF_T_PID_DATA2 USBF_T_PID_ERR USBF_T_PID_
IN USBF_T_PID_MDATA USBF_T_PID_NACK USBF_T_PID_NYET USBF_T_PID_OUT USBF_T_PID_PING USBF_T_PID_PRE USBF_T_PID_RES USBF_T_PID_SETUP USBF_T_PI
D_SOF USBF_T_PID_SPLIT USBF_T_PID_STALL clk crc16_clr crc16_err crc16_out crc16_sum crc5_err crc5_out crc5_out2 crc_in crc_out d0 d1 d2 dat
a_done data_valid0 data_valid_d din endif frame_no got_pid_ack h1f h800d hf0 hffff ifdef include next_state pid pid_ACK pid_DATA pid_DATA0
pid_DATA1 pid_DATA2 pid_ERR pid_IN pid_MDATA pid_NACK pid_NYET pid_OUT pid_PING pid_PRE pid_RES pid_SETUP pid_SOF pid_SPLIT pid_STALL pid_T
OKEN pid_cks_err pid_ld_en pid_le_sm rst rx_active rx_active_r rx_data rx_data_done rx_data_st rx_data_valid rx_err rx_valid rxv1 rxv2 seq_
err state token0 token1 token_crc5 token_endp token_fadr token_le_1 token_le_2 token_valid token_valid_r1 token_valid_str1 u0 u1 usbf_crc16
 usbf_crc5 usbf_defines v
For which variable above mentioned you want to see dependencies.
Enter the name of variable: TOKEN
Below is answer of COI of variable TOKEN:
Level1
clk
crc16_clr
data_valid_d
pid_ACK
pid_TOKEN
rst
rx_active
rx_err
rx_valid
rxv1
state
Level2
rx_active_r
pid_DATA
USBF_T_PID_ACK
pid
pid_ld_en
pid_IN
pid_OUT
pid_PING
pid_SETUP
```

```
pid_SOF
data_done
IDLE
next_state
Level3
pid_DATA0
pid_DATA1
pid_DATA2
pid_MDATA
hf0
rx_data
pid_le_sm
USBF_T_PID_IN
USBF_T_PID_OUT
USBF_T_PID_PING
USBF_T_PID_SETUP
USBF_T_PID_SOF
ACTIVE
DATA
Level4
USBF_T_PID_DATA0
USBF_T_PID_DATA1
USBF_T_PID_DATA2
USBF_T_PID_MDATA

Below is answer of COD of variable TOKEN:
Level1
next_state
Level2
ACTIVE
DATA
IDLE
state
Note : if you are getting clear screen, it means that your entered variable has no dependecies
PS C:\Users\91962\Desktop\rajesh>
```
