**REPORT ON AI ALGORITHM AND VERILOG IMPLEMENTATION**

# ***INTUITION:***

Q-Learning stands for ***“Quality Learning”.*** Q-Learning is a part of reinforcement learning. Reinforcement learning is the beautiful branch of artificial intelligence which lets machine learn on your own in a way different from traditional machine learning. Throughout our lives, we perform several actions to pursue our dreams some of them bring us good rewards others do not. Along the way, we keep exploring different *paths* and figure out which action might lead to better *rewards.* We work hard towards our dreams utilizing the feedback we get based on our actions to improve our strategies. They help us determine how close we are to achieving our goals. Our mental state teams continuously representing this closeness in that description of how we pursue our goal in the early life we framed for ourselves. Our representative analogy of reinforcement learning reformatting the main points of interest our reality contains ***environment*** in which we perform numerous actions sometimes we get ***good or positive rewards*** for some of these actions, in order to achieve the goals now during the entire course of life our mental and physical states evolve we strengthen our action in order to get as many rewards as possible. Now the *key entities* of interest are ***the environment, the action, reward, and the state***. Now, this whole paradigm of exploring the environment and learning through action rewards and States establishes the foundation of reinforcement learning*. Reinforcement learning solves a particular kind of problem where decision-making is sequential, and the goal is long-term, such as game playing, robotics, resource management or logistics.* Now for a robot and environment is a place where it has been put to use now remember this robot is itself the agent, for example, an automobile factory where a robot is used to move materials from one place to another. Now the task we discussed just now has a property in common these tasks involve an environment and expect the agent to learn from the environment. This is where traditional machine learning fades and hence the need for reinforcement learning now it is good to have an established overview of the problem that is to be solved using the Q- Learning or the reinforcement learning so it helps to define the main components of a reinforcement learning solution that is the environment, action, rewards, and States.

# ***INITIALISATION:***

We have l1, l2, l3, l4, l5, l6, l7, l8, l9 stations number

|  |  |  |
| --- | --- | --- |
| **L1** | **L2** | **L3** |
| **L4** | L5 | L6 |
| **L7** | L8 | L9 |

One thing you might notice here that there are

little obstacle present in between the locations

so l6 is the top priority location enable the robots

so that they can find the shortest route from any given location to another location on their own.

States are the location in which a particular robot is present in the particular instance of time which will denote its states. Now machines understand numbers rather than letters, so map the location codes to number. (i.e. l1->0, l2->1…l9->8/where l1, l2.l9 are locations and 0, 1,8 are states)

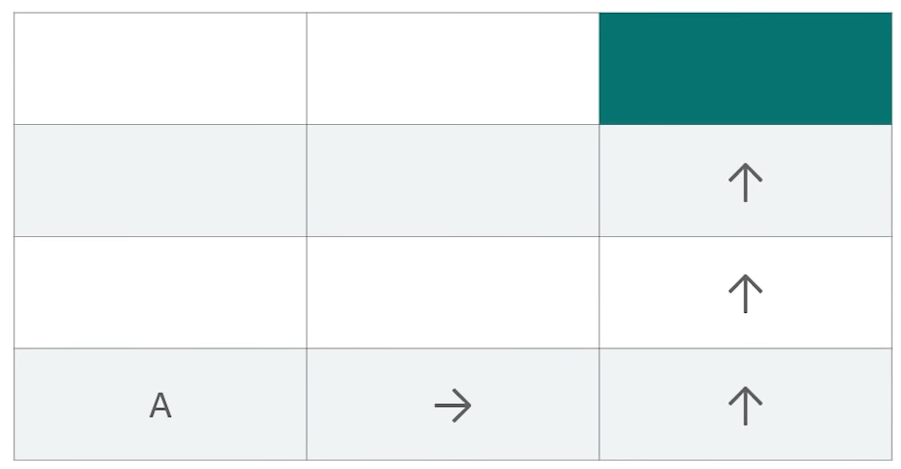
The next step is the actions part. Consider a robot that is at l2 location and the direct locations to which it can move are l5 l1 and l3. Here the set of actions is nothing but the set of all possible states of the robot, for each location, the set of actions that a robot can take will be different.

For example, the set of actions will change if the robot is in l1 rather than l2 so if the robot is in l1 it can only go to l4 and l2 directly. The next stage is the rewards so the states are basically 0 1 2 3 4 and the actions are also 0 1 2 3 4 up till 8 now the rewards now will be given to a robot if a location which is the state is directly reachable from a particular location.

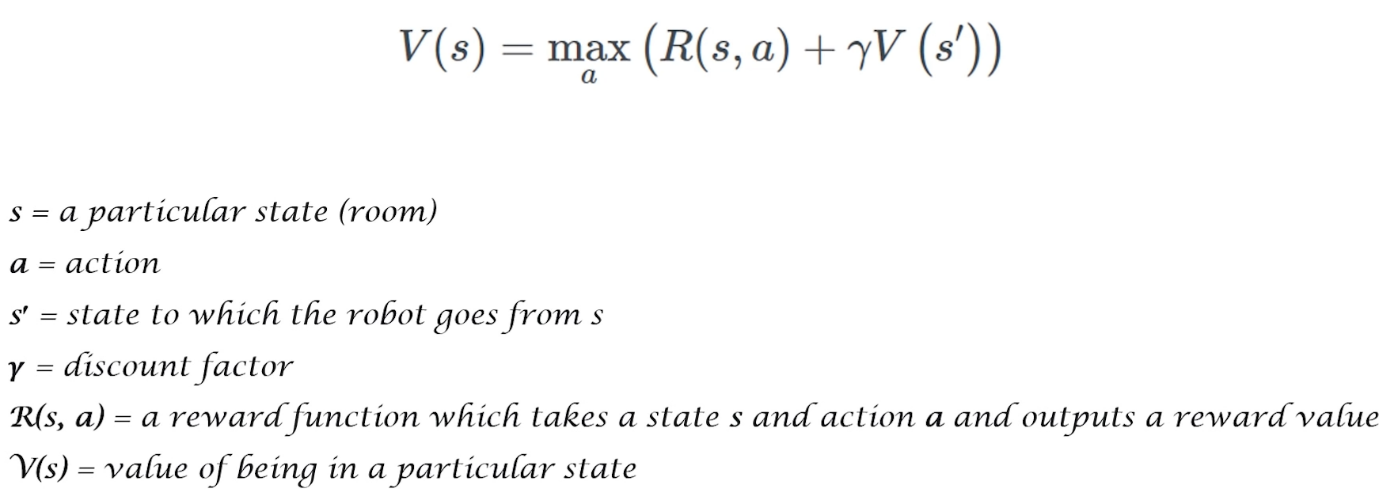
Suppose l9 is directly reachable from L8. If the robot goes from L8 to l9 and vice versa, it will be rewarded by 1 and if a location is not directly reachable from a particular location, we do not give any reward (i.e. a reward of 0). It enables the robots to make sense of the movements helping them in deciding what locations are directly reachable and what are not. Now with this Q, we can construct a reward table that contains all the reward values mapping between all possible states. In the table, the positions which are marked green have a positive that a robot can get by moving in between the different states.

We chose to prioritize L 6 to be the topmost so we associating the topmost priority location with a very high reward than the usual ones, i.e. 999 in the cell L 6.

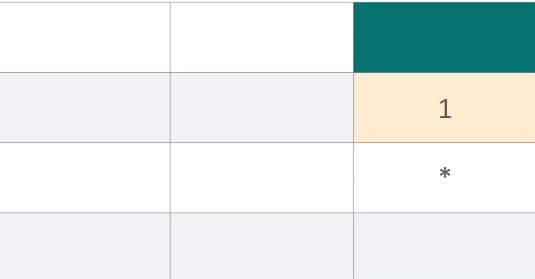
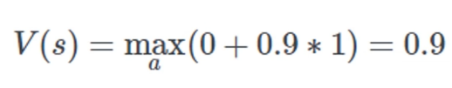
# ***WORKING:***

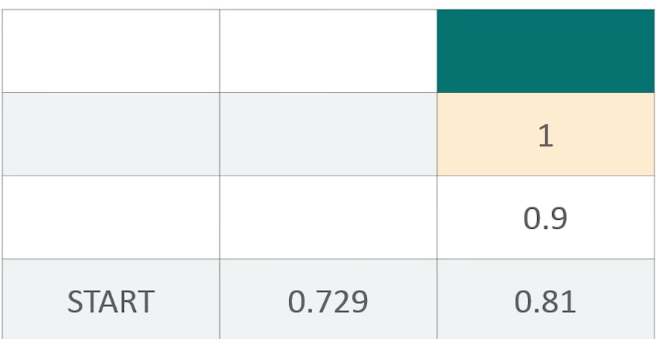


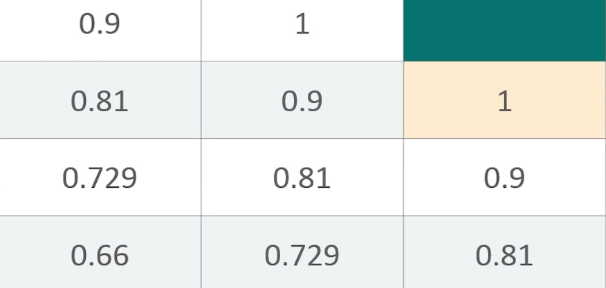
We'll start with the **bellman equation** now consider the square of rooms which is analogous to the actual environment from our original problem but without the barriers. Now suppose a robot needs to go to the room marked in the green from his current position ‘A’ using the specified direction and how can we enable the robot to do this programmatically. One idea would be to introduce some kind of footprint that the robot will be able to follow. Here a constant value is specified in each of the rooms which will come along the robot’s way if it follows the direction specified above. In this way, if it starts at location ‘A’ it will be able to scan through this constant value and will move accordingly but this will only work if the direction is prefixed and the robot always starts at the location ‘A’. Consider the robot starts at some other location rather than its previous one. The robot now sees footprints in two different directions it is, therefore, unable to decide which way to go in order to get the destination which is the green room, it happens primarily because the robot does not have a way to remember the directions to proceed. So, our job now is to enable the robot with a memory. Now, this is where the Bellman equation comes into play. As you can see here the main reason for the bellman equation is to enable the robot with the memory that's the thing, we're going to use so the equation.



V(s) is the value of being in a particular state (footprint)now there is one constraint however regarding the value footprint that is the room marked in yellow just below the green room it will always have the value of 1 to denote that is one of the nearest room adjacent to the green room now this is also to ensure that a robot gets a reward when it goes from a yellow room to the green room let's see how to make sense of the equation which we have.

let's assume a discount factor of 0.9 as gamma is the discount factor for the room which is marked just below the yellow room which is the \* mark for this room. The V(s) that is the value of being in a particular state would be  Here the robot will not get any reward for going to a state marked in yellow hence R(s, a)=0, but the robot knows the value of being in yellow room hence V(s’)= so this is how the table looks with some value footprints computed from the

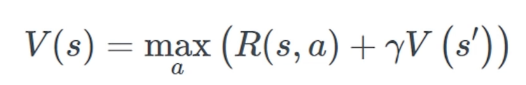
bellman equation. The math function helps the robot to always choose the state that gives the maximum value of being in that state. The discount factor gamma notifies the robot about how far it is from the destination. this is typically specified by the developer of the algorithm that would be installed in the robot. The other states can also be given them respective values in a similar way the boxes adjacent to the green one, it can proceed its way to the green room utilizing the value footprints even if it's dropped at any arbitrary location.



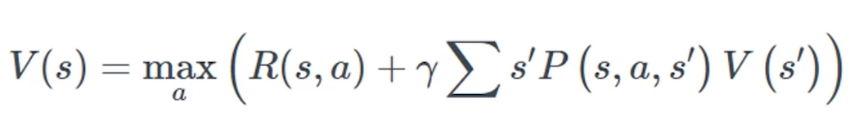
Sometimes the robot may come across some hindrance on its way, which may not be known through it beforehand right and sometimes even if the robot knows that it needs to take the right turn it will not, to solve this, we introduce the stochasticity in our case.

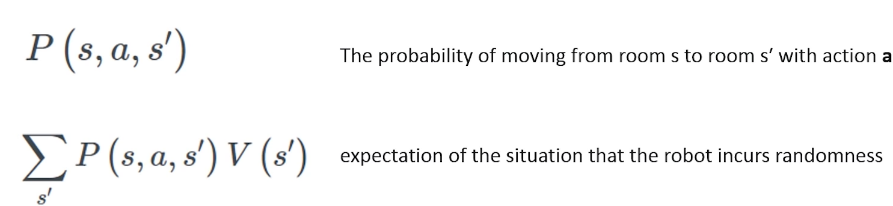
***The Markov Decision Process:***

now consider the robot is currently in the Red Room and it needs to go to the green room. let's consider the robot has a slight chance of misfunctioning and might take the left or the right or the bottom turn instead of taking upper turn in order to get to the green room from where it is. The question is how do we enable the robot to handle this? .A situation where the decision-making regarding which turn is to be taken is partly random and partly another control of the robot now partly random because we are not sure when exactly the robot and is functional and partly under the control of the robot because it is still making a decision of taking a turn right on its own and with the help of the program embedded into it . Markov decision process is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision-making in situations where the outcomes are partly random and partly under the control of the decision-maker. Let's have a look at the original bellman equation

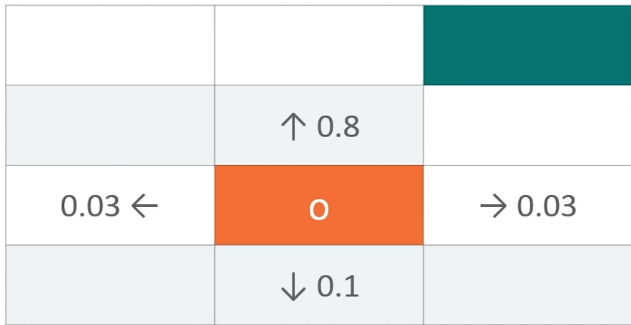


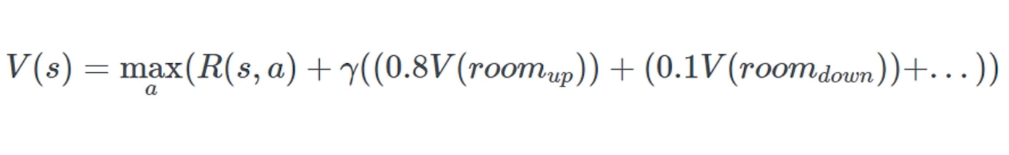
We are not sure when the robot might not take the expected turn, we are also not sure in which room it might end up. We are not sure of the s - which is the next state or the room but we do know all the probable turn the robot might take. In order to incorporate each of these probabilities into the above equation, we need to associate a probability with each of the turns to quantify the robot if it has got any explicitness chance of taking this turn. now if we do so we get,





let's take a look at this example. here when we associate the probabilities to each of these states, we essentially mean that there is an 80% chance that the robot will take the upper turn now.





note that the value footprints will not change due to the fact that we are incorporating stochastically here. We will not calculate those value footprints instead we will let the robot figure it out. Ideally, there should be a reward for each action the robot takes to help it better assess the quality of the actions. This idea is known as the ***living penalty***. In reality

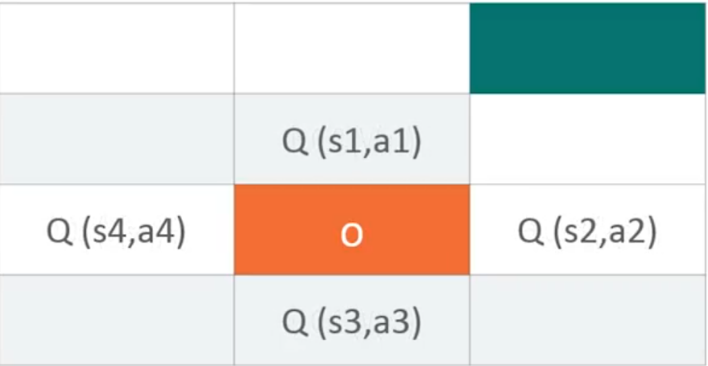
the reward system can be very complex and particularly modeling sparse rewards is an active area of research in the domain of reinforcement learning.

***Transition to Q Learning:***

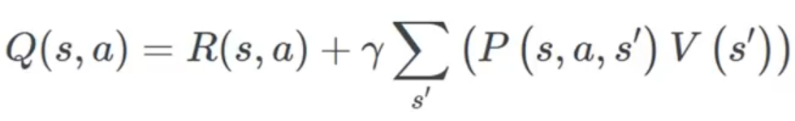
This equation gives us the value of going to a particular state, taking the stochastic city of the environment into account. the idea of the living penalty which deals with associating each move of the robot with a reward. *Q learning possesses an idea of assessing the quality of an action, that is taking to move to a state and rather than determining the possible value of the state which is being moved to*. if you incorporate the idea of assessing the quality of the

action for moving to a certain state. The environment with the agent and the

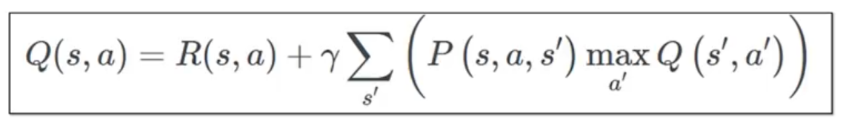
quality of the action,



If we take all possible states from the current state that robot is in and then we are taking the maximum value caused by taking a certain action and the equation produces a value footprint which is for just one possible action. We can say this *the quality of the action.*

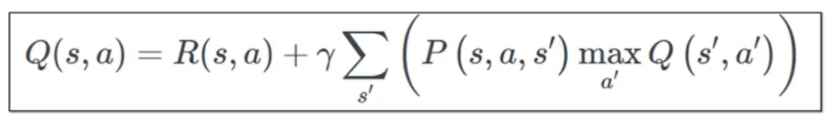


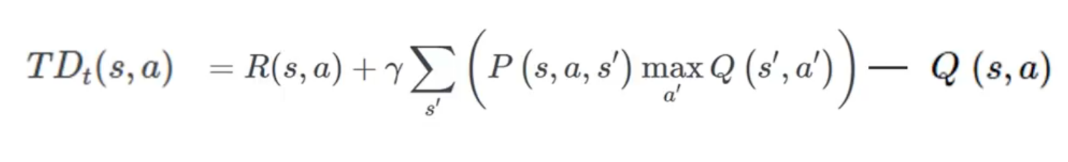
we have got an equation to quantify the quality of a particular action we are going to make a little adjustment in the equation. we can now say that V(s) is the maximum of all the possible values of Q(s). Now, replace V(s’) as a function of The resulting equation would be



The equation of V is now turned into an equation of Q which is the quality. We do that to ease our calculations because now we have only one function Q which is also the core of the dynamic programming language. We have only one function Q to calculate and R (s, a) is a quantified metric that produces the reward of moving to a certain state.

The qualities of the actions are called the Q values and from now on we will refer to the value footprints as the Q values, an important piece of the puzzle is the temporal difference. Temporal difference is the component that will help the robot calculate the Q values with respect to the changes in the environment over time.

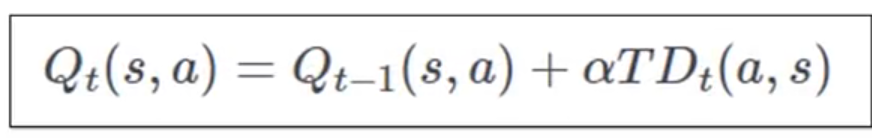




Consider our robot is currently in the mark state and it wants to move to the upper state, one thing to note that , the robot already knows the q-value of making the action that is moving to the upper state and we know that the environment is stochastic in nature and the reward that the robot will get after moving to the upper state might be different from an earlier observation.

So how do we capture this change and the real difference?

We calculate the new Q (s, a) with the same formula and subtract the previously known Q (s, a) from it. This gives us the new Q (s, a). The equation that we just tried gives the temporal difference in the Q values which further helps to capture the random changes in the environment which may impose. now the new Q (s, a) is updated as the following

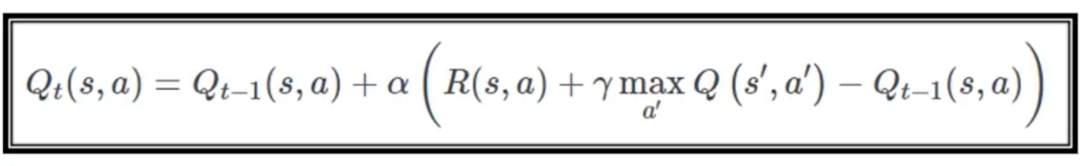


here **alpha is the learning rate** that controls how quickly the robot adapts to the random changes imposed by the environment.

**Qt (s, a)** is the current state Q value and

**Qt-1(s, a)** is the previously recorded Q value.

If you replace the TD (s, a) with its full-form equation we should get

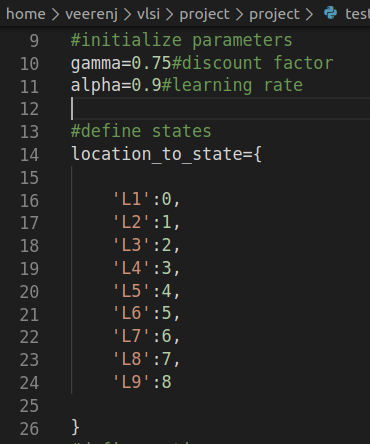


We have all the little pieces of Q-Learning together we can move forward to its implementation part.

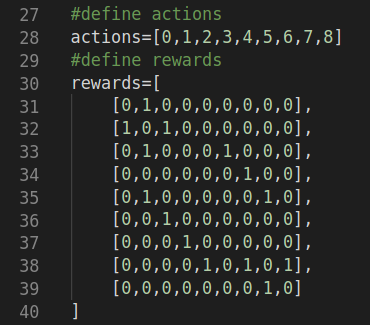
# ***IMPLEMENTATION:***

The above is the final equation of the Q learning rate, we implement this to obtain the best path for any robot to take. To implement the algorithm, we need to understand the location and how that can be mapped to different states. Define the reward table it's the same matrix

that we created.



There isn't any real barrier limitation as depicted in the image, for example, the transition l4 to l1 is allowed, but the reward will be zero to discourage that path or in a tough situation. we add a minus one so that it gets a negative reward.



In the above code snippet, you can see that each state has one in the respective state that are directly reachable from a certain state. Refer to that reward table we created to understand.

Note that we did not consider the top priority location l6 yet.

We need an inverse mapping from the states back to its original location and it will be cleaner when we reach the utter depths of the algorithms.

The inverse map location state the location we will take the distinct state and

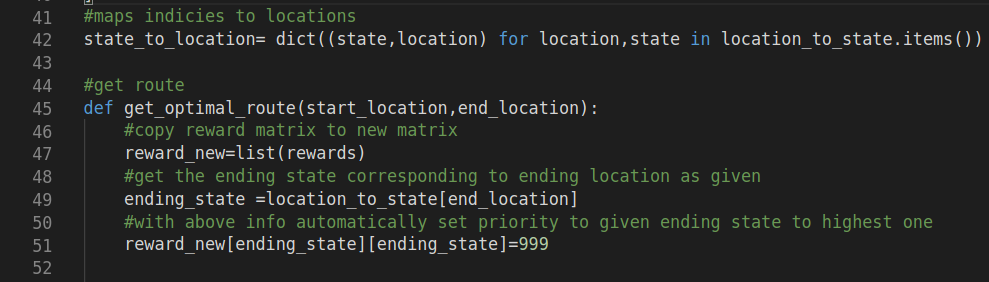
location and convert it back.

Define a function get optimal to obtain the optimal route which will have a start location

and an end location. The function takes two arguments (the starting location and the end

location and it will return the optimal route for reaching the end location from the

starting location in the form of an ordered list.



# ***VERILOG INCLUSION:***

# 

# ***VERILOG MODULES:***

1. Full Adder has been used 4 times for each iteration.
2. Wallace Multiplier has been used 2 times for each iteration.
3. The number of iterations is 1000.
4. Therefore, Full Adder has been used 4000 times,
5. Wallace Multiplier has been used 2000 times.

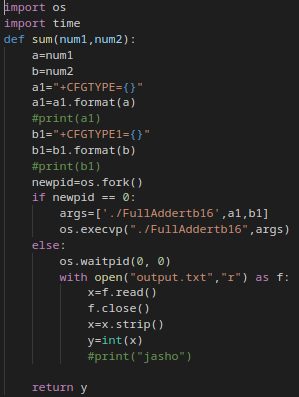
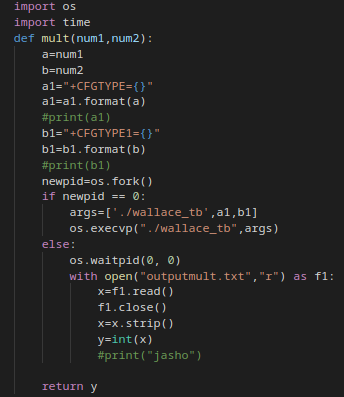
# ***Parameter passing:***

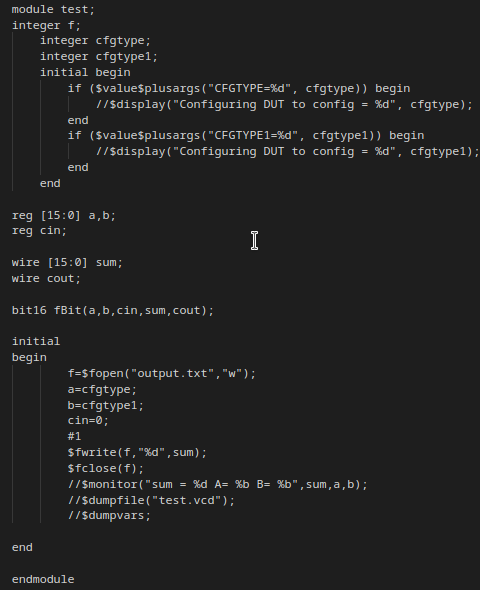
CFGTYPE is used to pass integer parameter to compiled testbench file.

For example, ./FullAddertb16 +CFGTYPE=255 +CFGTYPE=246

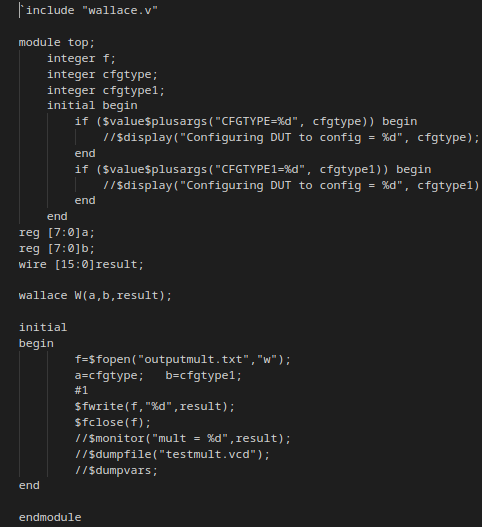
Filename:

1. Addition: add.py, FullAddertb16.v
2. Multiplication: mult.py, wallace\_tb.v

1.)  2.) 

1.)

2.)



When the function is called, the process gets forked and inputs are given to testbench file in child process, opens testbench file and pass the given command line values into the testbench file.

i.e. child process takes input.

The parent process saves the then values into output.txt file and returns the value to the python file in which these functions are called.

i.e. parent process returns values.