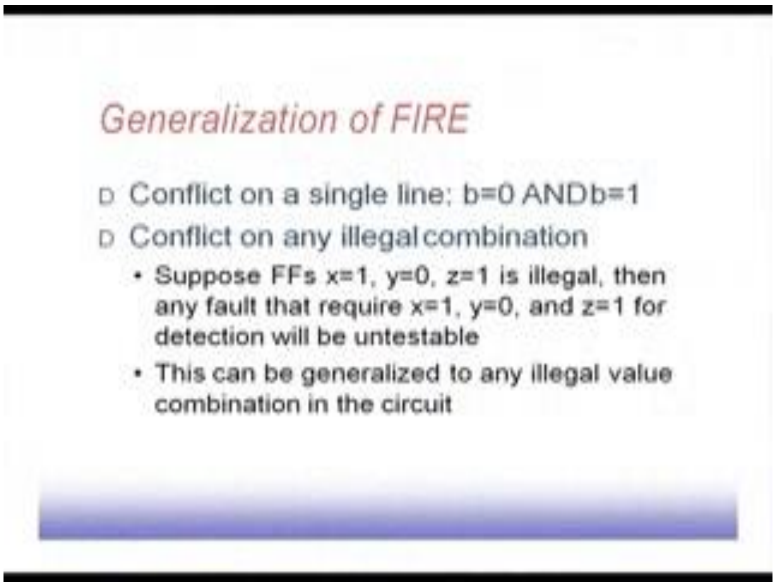
Digital VLSI Testing

Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 21
Test Generation (Contd.)

We can generalize this fire process for fault identification process.

(Refer Slide Time: 00:23)



Generalization of FIRE

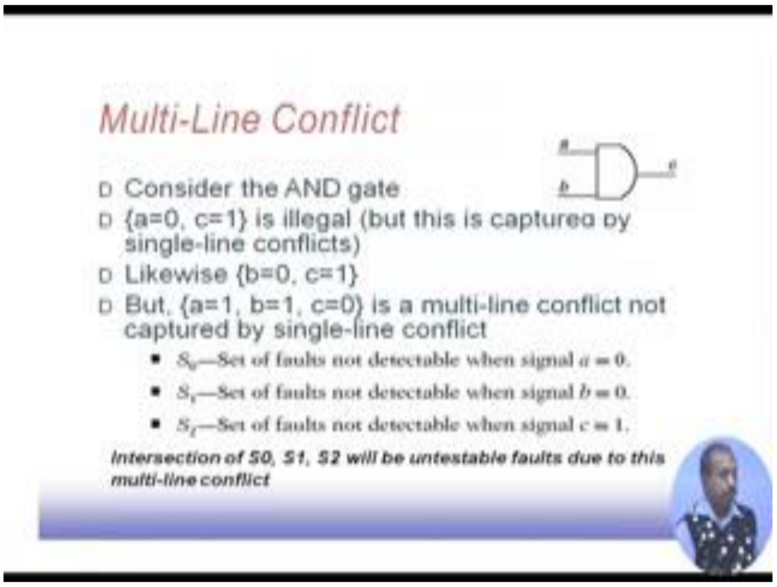
- Conflict on a single line: b=0 ANDb=1
- Conflict on any illegal combination
  - Suppose FFs x=1, y=0, z=1 is illegal, then any fault that require x=1, y=0, and z=1 for detection will be untestable
  - This can be generalized to any illegal value combination in the circuit

Further by considering the cases like conflict on a single line like b equal to 0 and b equal to 1. So, if one particular line requires both of them then this is this is for a single line that is a conflict so that can be removed those type of fault. Similarly, if we have got flip flops, so we have not discussed about sequential ATPGs, but sequential ATPGs can also there and though the algorithms are a bit complex compared to this combinational ATPG.

But in that case also we can award some of the combinations, like say suppose I have got three flip flops x y and z. Now, it may so happen that due to in the operation of the application, so this x equal to 1 y equal to 0 and z equal to 1 is illegal. So, this can happen because this xyz they are flip flop, so they are replace they a combination of them represent some state of the system that it models. Now maybe some of the states of the system are illegal. So, the system will never go to those states.

Now if a test generalization process requires that x should be said to equal to 1 and y equal to 0 and z equal to 1, so in at in reality will never be able to force those force those flip flops to those values. So, any faults that require this particular setting they are untestable fault, because we will never be able to put these values on to those flip flops. So, this is another way of finding out the untestable faults particularly suitable for sequential circuits. Now of course, for scan circuit this is not a problems; scan circuit can always put x equal to 1 y equal to 0 z equal to 1. So that is why this is not there in combinational and scan circuits, but for sequential ATPG with this may be and untestable fault.

(Refer Slide Time: 02:18)



Multi-Line Conflict

- Consider the AND gate
- {a=0, c=1} is illegal (but this is captured by single-line conflicts)
- Likewise {b=0, c=1}
- But, {a=1, b=1, c=0} is a multi-line conflict not captured by single-line conflict
  - S_0—Set of faults not detectable when signal $a = 0$.
  - S_1—Set of faults not detectable when signal $b = 0$.
  - S_2—Set of faults not detectable when signal $c = 1$.

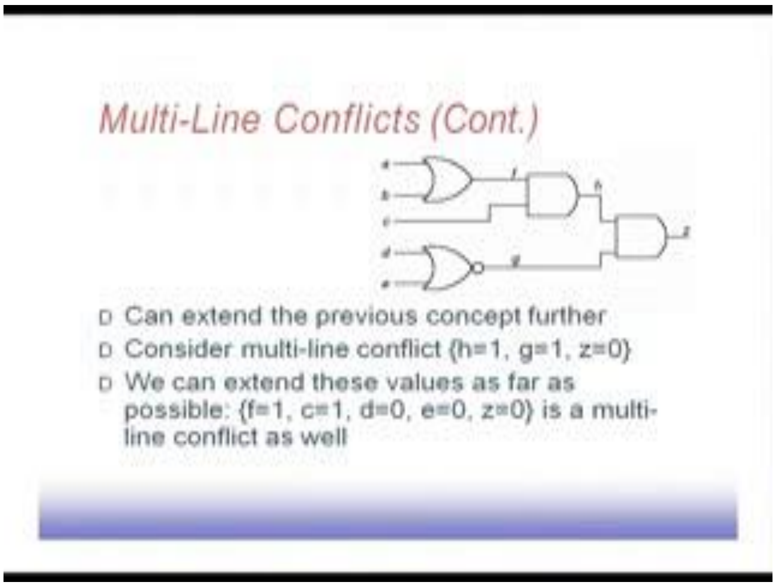
Intersection of S0, S1, S2 will be untestable faults due to this multi-line conflict

Sometimes we need to proceed with multi line conflict, like say this one say this is an AND gate a equal to 0 c equal to 1 is illegal, because this if I said a equal to 0 I cannot get c equal to 1. So, if I find implication of a equal to 0 I will get c equal to 0 as a points, so c equal to 0 and c equal to 1 that is a conflicts. So, it can be found by a single line conflict.

On the other hand, if I have b equal to 0 c equal to 1 that is also a conflict, because this cannot be done. However, a equal to 1, b equal to 1, c equal to 0 this cannot be done by a single line conflict, because this s 0 is the set of faults not detectable with signal a equal to 0; so that is the set S 0. Single S 1 is the set of faults are not detectable with b equal to 0. Now S 2 is the set of faults not detectable under when signal c equal to 1.

So, if we take intersection of these three sets that will give us the untestable fault due to this multi line conflicts. So, a equal to 1, b equal to 1, c equal to 0 is a multi line conflict which is not covered by the single line conflict. If you just take the intersection of this you will only get c equal to 1, we will not get this other fault a equal to 1 b equal to 1 c equal to 0. So, that way this will come only when you have taken the intersection of this three sets S 0 S 1 and S 2. So, this way this multi line conflict can also be used for a determining the faults which are untestable.

(Refer Slide Time: 03:56)



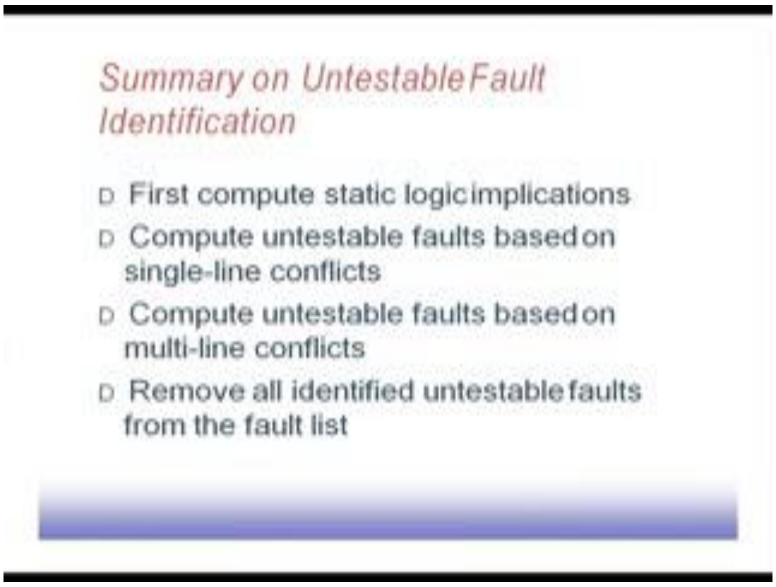
Multi-Line Conflicts (Cont.)

- Can extend the previous concept further
- Consider multi-line conflict {h=1, g=1, z=0}
- We can extend these values as far as possible: {f=1, c=1, d=0, e=0, z=0} is a multi-line conflict as well

Can extend from the previous concert further like consider say h equal to 1, g equal to 1, and z equal to 0; h equal to 1 g equal to 1 and z equal to 0 so it is a multi line conflict. So, we can extend these values as far as possible, like if we do the implication backward implication and on all that so we will get f equal to 1, c equal to 1, d equal to 0, e to equal to 0 z equal to 0. By extending the conflicts that are occurring here we can see that these are the further conflicts that can come. So, this is also a multi line conflict.

So, if we start with multi one multi line conflict and try to is to you may find the logic implications of that then we can find further multi line conflicts. So, that is also very much useful in detecting the untestable faults.

(Refer Slide Time: 04:47)



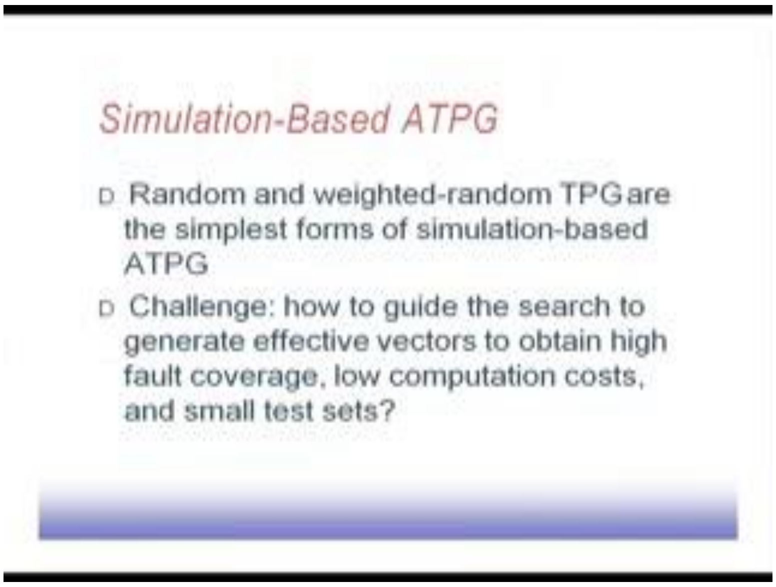
Summary on Untestable Fault Identification

- First compute static logic implications
- Compute untestable faults based on single-line conflicts
- Compute untestable faults based on multi-line conflicts
- Remove all identified untestable faults from the fault list

To summarize this untestable fault identification part: first we compute static logic implications that is the first step, then we compute untestable faults based on single line conflicts, then we compute untestable faults based on multi line conflicts, and then we remove all identified untestable faults from the list.

So, by doing these first three steps you could find the untestable faults. Then before submitting the faults to the ATPG tool to generate test pattern, so you remove all those faults from the list of faults given to the tool. So, that way this ATPG algorithm will have much less number of faults for visit has to generate the test patterns. So, that will speed up the operation of the ATPG.

(Refer Slide Time: 05:34)



Simulation-Based ATPG

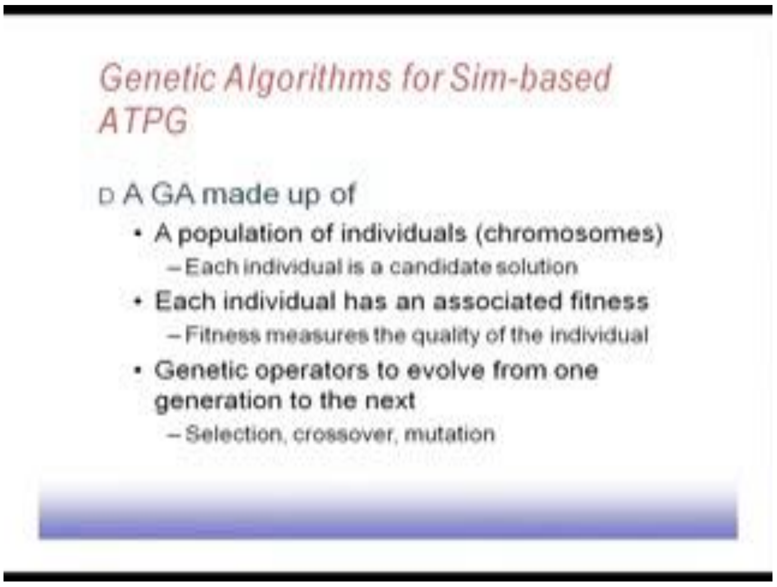
- Random and weighted-random TPG are the simplest forms of simulation-based ATPG
- Challenge: how to guide the search to generate effective vectors to obtain high fault coverage, low computation costs, and small test sets?

Another class of ATPG algorithm they are known as simulation based ATPG techniques. So far whatever technique we have discussed, one of them was the exact method like Boolean difference. Then we have seen a number of heuristic methods, this like d algorithm, podium and etcetera.

Now, another class of ATPG algorithms are known as simulation based ATPG algorithm. So, the random and weighted random test pattern generators are the simplest forms of simulation based ATPG, because they are generating some random pattern doing a circuit simulation to see what are the faults it can detect. And if we detects new fault fine otherwise it will generate another next random pattern. So, that is a simplest type. Now these random generation process, if you can guide it; so if you can guide this random generation process so that it can generate effective vectors. So, instead of being totally random if we can guide it to go to the portion of the search space why it needs to the really generate a new test patterns to improve fault coverage then it is very much useful.

At the same time competition cost should be low, it is not that this exploration process takes lot of kind. And the test say generated should also be small in the compact test. So, so these are actually the desirable features. So, if you can do this thing then we can come up with some good ATPG technique.

(Refer Slide Time: 07:00)



Genetic Algorithms for Sim-based ATPG

- A GA made up of
  - A population of individuals (chromosomes)
    - Each individual is a candidate solution
  - Each individual has an associated fitness
    - Fitness measures the quality of the individual
  - Genetic operators to evolve from one generation to the next
    - Selection, crossover, mutation

One of the very well known ATPG class of algorithm, so they are based on genetic algorithms. So, genetic algorithms you know that this is a stochastic optimization process that me mix the behavior of natural selection and this generations of how the generation improves, the quality of population improves, so it is based on that.

So, in case of this genetic algorithm; any genetic algorithm is made up of these components, a population of individuals which we call chromosomes and each individual is a candidate's solution. Now you see that for a particular problem it may be very difficult to get tail what is the optimum solution, but it may be very easy to give any arbitrary solution. For example, even for test pattern generation problem if you ask me to give a test pattern with detects at very high number of faults, large number of faults it is very difficult to give such a pattern.

But if you ask me to give any test pattern then I can give it very easily. So, I can just say any random in 0, 1 binary sequence, even to the inputs of circuit so that itself is a test pattern it will detect at some faults or the other hopefully. So, this is each individual is a candidate solution. So, it may be possible that I can talk I can give a large number of such individual solutions without bothering about the quality of them. After that each individual has an associated fitness. So, fitness actually tells how good or bad is the particular solution or the particular individual.

So, in the context of test pattern generation, if a pattern can detect large number of faults we can say that its fitness is high. On the other hand, if it detects only few of the undetected faults then its quality is low. And if detects no none then of course its quality is very well. So, that way we can think about this test pattern generation process as a collection of individuals for every pattern is a individual and they have got they are fitness.

Now, genetic operators they evolve from one generation to the next. This will help us from the current generation so if you want to go to the next generation by generating new population new individuals. So, they are done by three mechanisms: one is called selection, another is called crossover, and another is called mutation. So, these are the three genetic operators that are use for evolving over the generations.

(Refer Slide Time: 09:50)

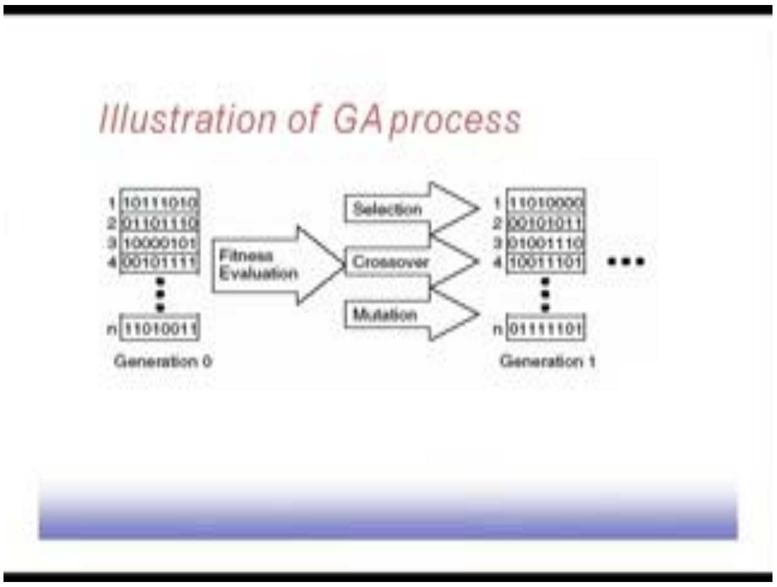
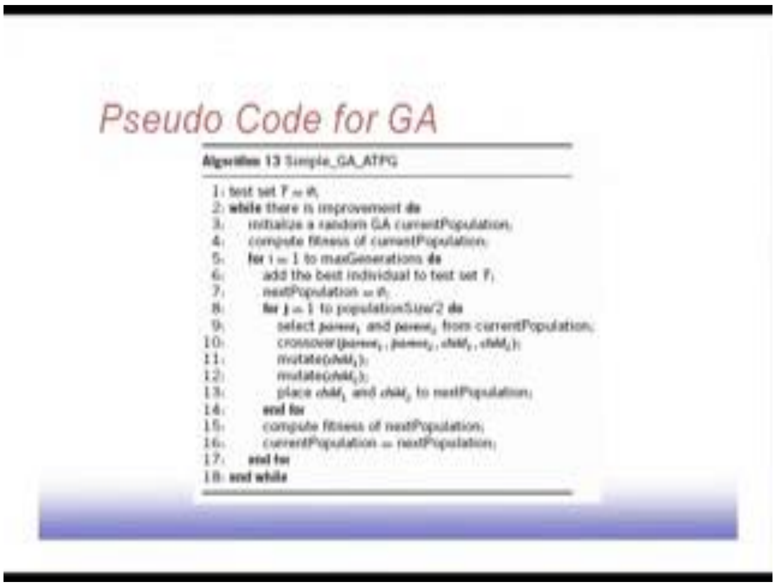


Illustration of GA process

So, in generation 0, so this is the pictorial description of this genetic algorithm process. So, in generation 0 we have got this individual, so one to there are n individuals and each individual here is a sequence of bits string. Each of them goes to a fitness evaluation so every individual has got some fitness measure associated within. Now there is a there are three operators as we have said the selection, crossover, and mutation. So, based on these fitness values of these individuals that we have in generation 0 these three operators it will create new population, it may select some of them from these existing population or it may generate some new population. And accordingly it comes to generation 1.

Now, point is that population size over the generation they remain unaltered. Here it was n, in generation 1 also it is n, and in successive generations also it will be n. So, this actually mimics the behavior of natural genetics, but in natural genetics the advantages that the population's size is virtually infinite. Whereas, in case of this computer base simulation for this genetic algorithms the population size is not infinite due to the limitation on the memory space, due to the limitation of the computational time. So, naturally we do not get the level of variation that you can have in the nature, but if the population size is reasonably good then we will be able to get good variation in the population and possibly it will lead to better of springs.

(Refer Slide Time: 11:32)



Pseudo Code for GA

Algorithm 13 Simple_GA_ATPG

```
1: test set $T = \phi$;
2: while there is improvement do
3:     initialize a random GA currentPopulation;
4:     compute fitness of currentPopulation;
5:     for i = 1 to maxGenerations do
6:         add the best individual to test set T;
7:         nextPopulation = $\phi$;
8:         for j = 1 to populationSize/2 do
9:             select $parent_1$ and $parent_2$ from currentPopulation;
10:            crossover($parent_1$, $parent_2$, $child_1$, $child_2$);
11:            mutate($child_1$);
12:            mutate($child_2$);
13:            place $child_1$ and $child_2$ to nextPopulation;
14:        end for
15:        compute fitness of nextPopulation;
16:        currentPopulation = nextPopulation;
17:    end for
18: end while
```

So, this is the pseudo code of this g a based ATPG techniques. So, initially test set is null and while there is improvement; so improvement may be in terms of fault coverage like new faults getting detected, fault coverage of the test set is improving initially fault coverage is 0. So, what we do? Initialize a random g a current population. So, initially it is randomly done you compute the fitness of current populations so for all the individuals a compute the fitness. So, fitness computation can be done by doing a faults simulation for individual pattern there faults that their detecting and then finding out the what is the overall population, what is the overall quality of the solution.

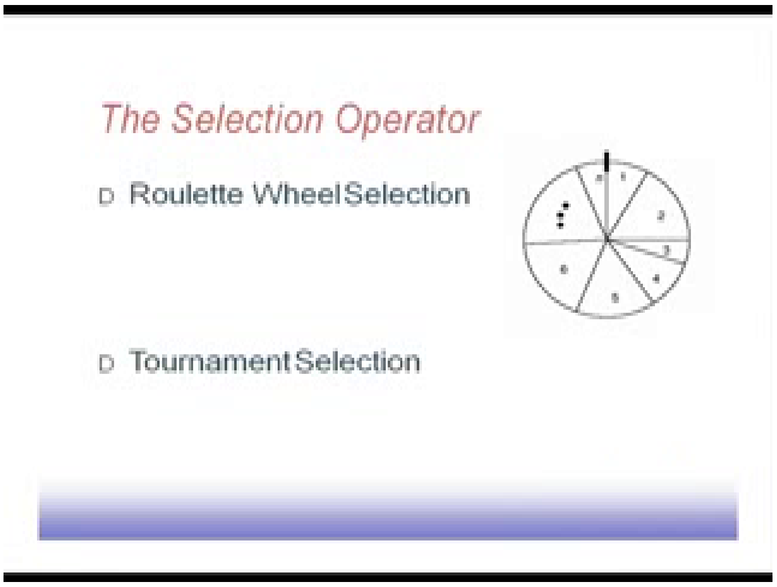
Then, for i equal to 1 to maximum number of generations, we have got some variable maximum generations so that tells for how many generations my g a will run. So, as I have already said in nature the process is infinite so the population goes on evolve evolving over generations. And there is no limit to the number of generations to which this population will evolve. But in case of computer program we have limitations, so we cannot run it infinitely. So, we put a limitation on the maximum number of generations for which the evolution will take place.

So, what we do at the best individual to test set t? Initially in generation 0 we have generated one random population set, now from the random population set after computing fitness so we get the best fit individual. The best fit individual is added to the test set t. So, t is now having one individual or one chromosome (Refer Time: 13:17). Then we compute another set, next populations this set is initialized to null and then what you do we try to create new population new individuals that to put into the next population set.

So, what is done? To select two parents parent 1 and parent 2 from the current population. And then perform crossover operation over this parent 1 and parent 2 to create two children child 1 and child 2. Then we do some mutation operation, so both this child 1 and child 2 they are subjected to some mutation operation and that modifies this child 1 and child 2 a bit. And then this child 1 and child 2 they are put into the next population. So, next population gets this thing, so this process goes on.

So far for every for every iteration of this (Refer Time: 14:13) your generating two such child process two such children individuals. So, after this j iteration is over we have got created populations equal to the population size. And next population has got population size number of individuals. So, again we compute the next population and current population is updated to next population. Then this process continues. So, again this fault rupees oversold this next generation it come starts with the next generation and this process goes on. And while there are improvements means while there are improvement in the say the fault coverage that may be a fitness measure.

(Refer Slide Time: 15:01)



The Selection Operator

- Roulette Wheel Selection
- Tournament Selection

Now the selection operator: now as we have said is in this cases say this crossover then mutation so when you are doing this thing there is a selection process parent 1 and parent 2 from the current population. Now this how do you select? Like if I have got say n number of individuals so we can randomly select two parents from there. But in general observation is that if the parent individuals have got good fitness then the children individuals will also have good fitness; that is a general observation.

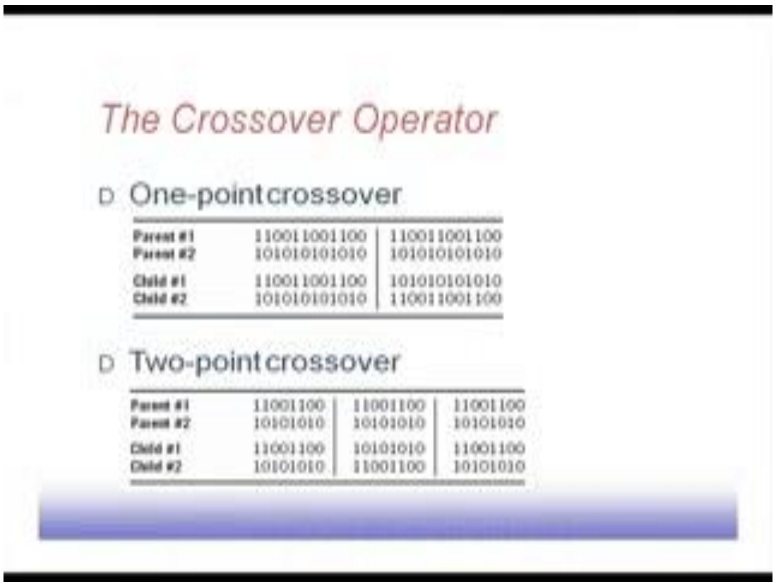
So, that is why how to do this thing this selection process can be made using two different techniques: one is called roulette wheel selection, another is called tournament selection. In roulette wheel selection what is done is that this individuals want to n they are put on a roulette field. Now depending upon the fitness of individuals that we have it is assigned a range of values in this roulette wheel. In this particular case say the individual six has the largest fitness, so it is given this much of region of the roulette wheel.

Similarly two is the next one, so two is given the next the region of the roulette wheel. So, that way they are given individual regions. So, the roulette wheels if this roulette wheel is rotated then the possibility of selecting six is the highest, then possibility of then followed by selection of two selection of five like that. So, in a random number generation process we can mimic this behaviour of roulette wheel. So, we can divide this random number range into different sub ranges and this larger sub range is allocated to individuals that have got better fitness. And now a random number will be uniformly generated and then where if whichever range it faults, so sub range it faults based on that the corresponding individual will get selected. So, the possibility of selecting better individuals will be higher in case of this roulette wheel case.

Another selection process is known as tournament selection. So, tournament selection is like this that it is between two individuals. So, you randomly select two individuals from the population. Now, whichever is better that means that, that one is finally selected. For example, if its speak picks up individual 1 and 5 and 5 fitness is better than 1 then individual 5 qualifies for the crossover operator. The same process is now repeated for selecting another individual. So, again there is a tournament between to randomly picked up individuals and whichever individual means that will take part in crossover along with the individual five that is selected before.

So, this is the tournament selections. So, in both the cases either roulette wheel or tournament selections, so I will the parent chromosomes parent individuals they are getting selected.

(Refer Slide Time: 17:59)



The Crossover Operator

- One-point crossover

Parent #1	110011001100	110011001100
Parent #2	101010101010	101010101010
Child #1	110011001100	101010101010
Child #2	101010101010	110011001100

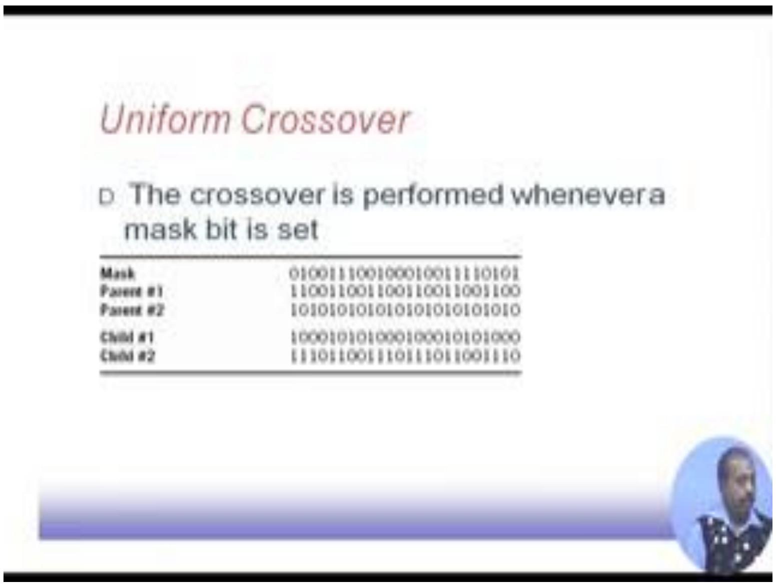
- Two-point crossover

Parent #1	11001100	11001100	11001100
Parent #2	10101010	10101010	10101010
Child #1	11001100	10101010	11001100
Child #2	10101010	11001100	10101010

Now, the crossover operator: so there are different versions of these crossover operators that are followed. Like first one maybe one-point crossover. So, this is one individual and this is the second individual; two parents. Now suppose the crossover is at this point. So, what happens? In the first child, so the first part of parent 1 is copied and the second part of parent 2 is copy into this. So, you see the first part of parent 1 and second part of parent 2 forms the child 1. Similarly second part of parent 2 followed by first part of child 1 forms the child 2. So, this is for one-point crossover.

So, there maybe two-point crossover like over this length of the individual, so you can select two-points along which the crossover will take place. And now it is the first part from the first individual, second part from the second individual and third part from the first individual, so that forms the first child. Similarly first part from second individual, second part from first individual and the third part from second individual that from the second child. So, this way this two-point crossover can be generalized. So, it can go to any point crossover that we can look for.

(Refer Slide Time: 19:19)



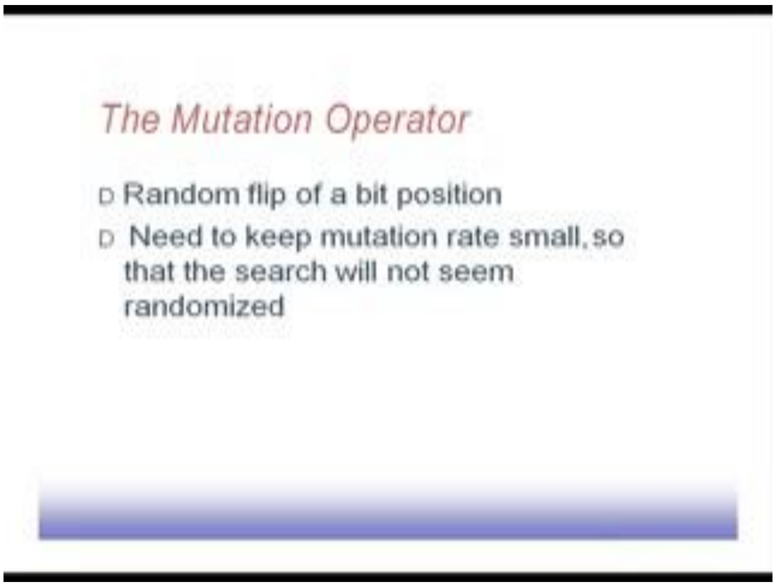
Uniform Crossover

- The crossover is performed whenever a mask bit is set

Mask	010011100100010011110101
Parent #1	110011001100110011001100
Parent #2	101010101010101010101010
Child #1	100010101000100010101000
Child #2	111011001110111011001110

And in the most generalized case we can prepare a mask. Suppose this is the mask that is generated now in this mask if the bit is 0 then say will copy child 1 from parent 1 and copy child 2 from parent 2. On the other hand if this mask bit is 1, so parent 2 will copy this copy parent to value into child 1 and parent 1 value into child 2; this just the reverse. So, this is for individual points based on this mask. So, these bits will get selected and that way that will define another type of crossover. So, it is a number of points that at which crossover is taking place is much higher compared to the one-point or two-point crossover that way have seen. So, this is also known as uniform crossover.

(Refer Slide Time: 20:12)



The Mutation Operator

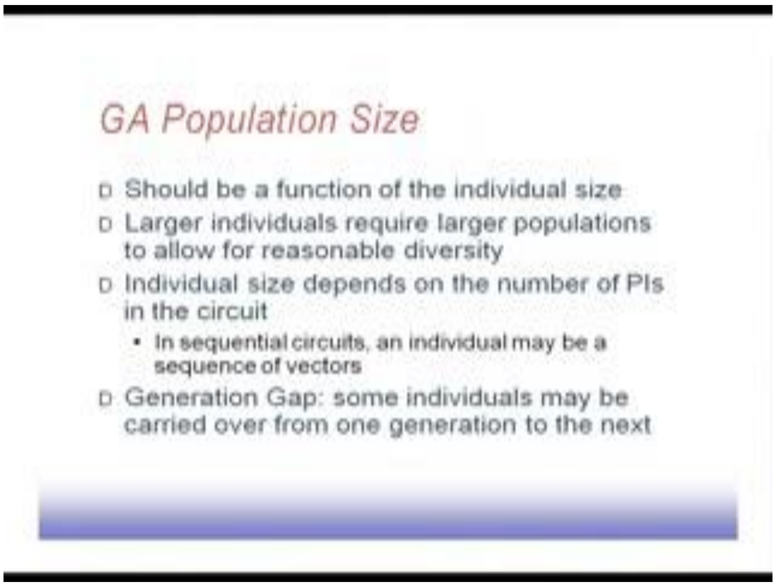
- Random flip of a bit position
- Need to keep mutation rate small, so that the search will not seem randomized

The mutation operators: so mutation operator what happens is that if you just go on doing crossover. Since the varieties are what we have in the population only. So, individuals they are the varieties that we have. Now it does not import any new variety into the system say if you just have the crossover operator.

So, a better operator for this purpose is the mutation operator. So, mutation operation what it does? It randomly flips a bit position. So, it in a chromosome, but an individual it will randomly flip one particular bits, so that will bring variety which was not present in the population so far, so that will be taking in taking the thing. So, it is necessary to keep mutation rate small so that search will not seem randomized. So, if the mutation rate is high then it shows random behaviour, because randomly a large number o bits will change so that is more of a random search this is not a directed search. But what we will be looking for is that two good individuals they will give rise to good children. So, that way that the search is being directed by the genetic algorithm; so if you increase mutation rate high to make mutation rate high then that philosophy will be lost; that will become more of a random behavior.

At the same time when you are in nature actually this mutation rate is very small, so it is very very unlikely that all of a sudden one gene of a chromosome will change. So, it changes over say large number of generations. However, in case of computer programs it may be that we keep this mutation rate a bit high compare to this natural genetics, because the population size is limited. So, we must keep the size as a factor this proper mutation rate at a factor.

(Refer Slide Time: 22:04)



GA Population Size

- Should be a function of the individual size
- Larger individuals require larger populations to allow for reasonable diversity
- Individual size depends on the number of PIs in the circuit
  - In sequential circuits, an individual may be a sequence of vectors
- Generation Gap: some individuals may be carried over from one generation to the next

Now how do we choose a population size? Say what should be population size? It says that it should be a function of the individual size. So, it depends on whatever is the individual size, because individual size will tell us what is the total search space size and out of that search space we should be putting a part of it into the population. So, if larger the individuals they will require larger population to allow reasonable diversity.

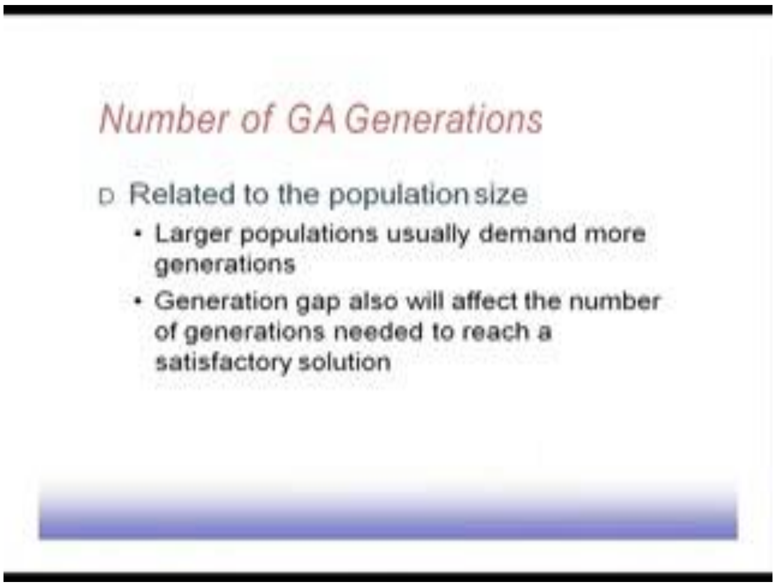
And again for this test generation problem the size depends on the number of primary inputs of the circuit, because every individual is nothing but a test pattern and a test pattern size is equal to the number of primary inputs of the circuit. So, if those numbers of inputs in a circuit is large then naturally this individual size also has to be large. And accordingly to bring large number of variations the population size also needs to be large. So, that is one potential problem of those these particular approach, because we need to have large population for larger circuits.

For sequential circuits an individual may be a sequence of vectors, because in sequential circuits what happens is that we need to apply the test patterns in a sequence. The first pattern will put the flip flops in some proper state and the second pattern will do the test actual testing. So, that way we need to have some sequence of vectors. So, the sequence of vectors is a problem, so the individual that we are talking about so that is basically a sequence of vectors.

Sometimes we encourage generation gap; what it says in the some individuals may be carried over from one generation to the next. So, the algorithm that we have seen it says that from current generation to the next generation with the general population generation process is by means of crossover and mutation operator. But it may so happen in the process that some good chromosome which was present in this generation get lost when you generate the next generation population through crossover and mutation operators.

So, what is desirable is? If some individuals are really good, they may be directly copy to the next generation. They do not go through this crossover mutation channel; they directly get copied on to the next generation. So, this creates generation gap in the evaluation process, but that may have to be encouraged.

(Refer Slide Time: 24:36)

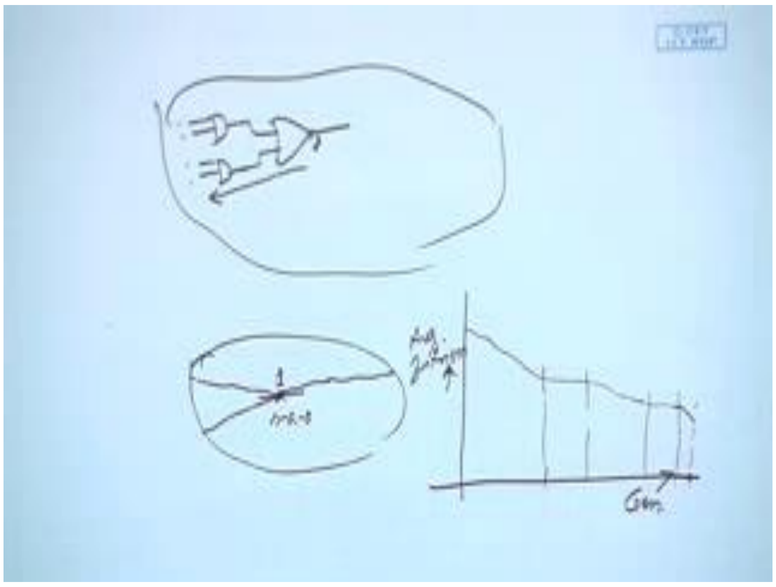


Number of GA Generations

- Related to the population size
  - Larger populations usually demand more generations
  - Generation gap also will affect the number of generations needed to reach a satisfactory solution

How many generations that we run in GA? Again it is related to population size. So, larger the population, so you should demand more number of generation.

(Refer Slide Time: 24:54)



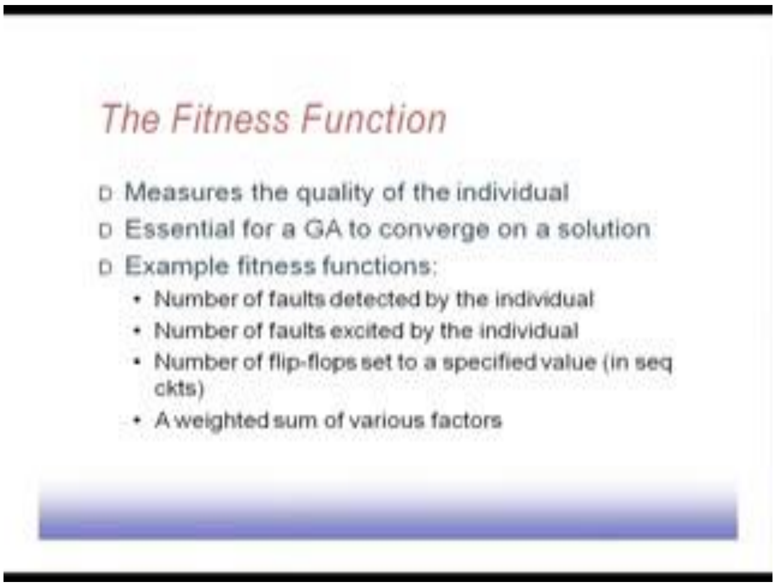
So, basically what happens is that if you just look into the operation of any genetic algorithm; this side we have got the generations and this side we plot the say the best fitness or average fitness whatever, average fitness of individuals. Then we will see that for first few generations it will start coming, it will be improving very fast, but after that this improvement rate becomes more or less flat. And again after sometime it will decrease bit and then it will go like this.

So, what happens is that for large number of generations in between like say this region so the quality does not improve. Similarly in this region quality does not improve, that can happen. So, we have to take those things into consideration. So, we have to see that though those parts are taken care of, so until analyze we make our; so after there is a drop if we are not allowing to the GA to run up to this generation this particular solution will not be obtained.

So, this problem is there. As a result larger the population size, larger will be this generation that will that will required to run. So, larger population will demand more generation. And this generation gap will affect the number of generation needed to reach a satisfactory solution. So that is also a factor, because generate so that will how many chromosomes your directly copying from one generation to the next generation. And actually when do it terminate; one termination is the of course a maximum number of generations that we can allow the GA to run.

And another termination is that if it does if the best solution does not improve over last few generations then we can say that possibly we have these take good solution and we stop at that point so that can be done.

(Refer Slide Time: 26:41)



The Fitness Function

- Measures the quality of the individual
- Essential for a GA to converge on a solution
- Example fitness functions:
  - Number of faults detected by the individual
  - Number of faults excited by the individual
  - Number of flip-flops set to a specified value (in seq ckts)
  - A weighted sum of various factors

About the fitness function: it measures the quality of the individual essential for a GA to converge to a solution, so a fitness function it should be good it should reflect the actual fitness of the individuals. So, for this test pattern generation problem you can have several such fitness functions, like number of faults detected by the individual. So, that is one type of fitness function. The number of faults exited by the individuals, so that is also be a fitness function.

Number of flip flops set to a specified value for sequentiality which is important. The number of flip flops set to some specified value so that may be another fitness function. Or we may have a weighted some of this factors, as when we go to some other test pattern generation technique like say power award testing and all that you may find that we have to have some other factors to come in the fitness function. So, this fitness function this very important and this has to be chosen carefully. We will continue in the next class.