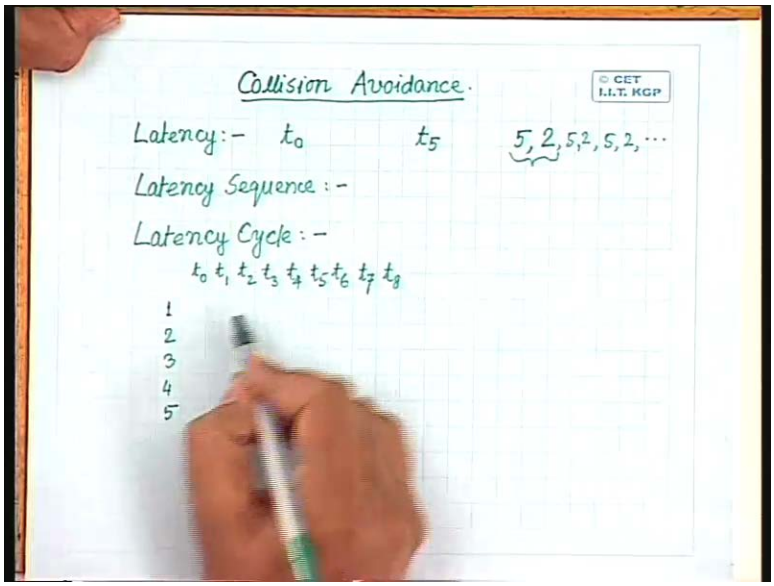
**Digital Computer Organization**
**Prof. P.K. Biswas**
**Department of Electronic and Electrical Communication Engineering**
**Indian Institute of Technology, Kharagpur**
**Lecture No. # 09**
**Pipeline Concept - III**

Now let us see that how to take care of collision and what should be your scheduling strategy in a generalized pipeline, so that collision does not occur.

(Refer Slide Time: 00:01:14 min)

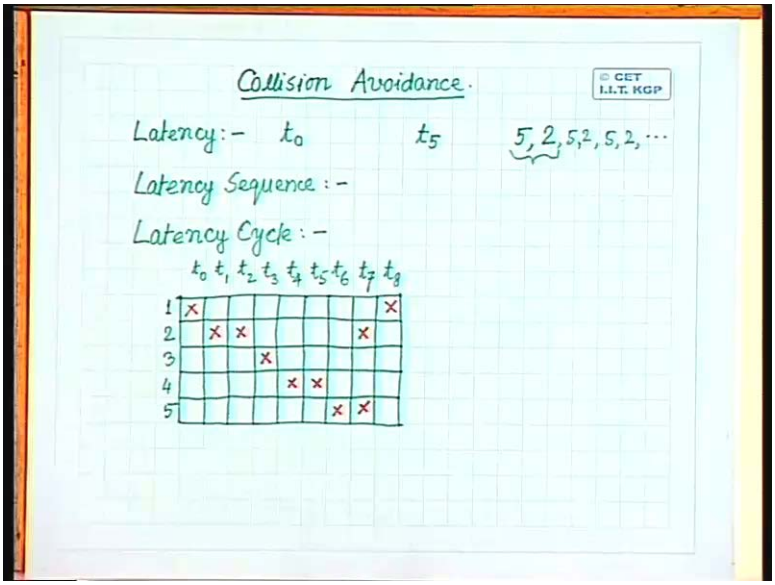



Now we will concentrate on collision avoidance. So before I go for this collision avoidance, let me introduce few terms, a term which is called latency. Latency is the time difference or the time step difference between successive initiation of two tasks. So if I initiate a task at the time $t_0$ and if I find that I can initiate another task at time $t_5$ and this initiation does not lead to any collision. Collision can be at any of the computation stages. It can be at computation stage $s_1$, at computation stage $s_2$ or $s_3$ but I have to find out that I have to initiate the tasks in such a way that it does not lead to collision at any of the computation stages. By analyzing, if I find out that I can initiate a task and if I initiate a task at $t_0$ that is the first task, the second task I can initiate only at $t_5$ nowhere in-between. Then the latency is the time step difference between initiation of these two tasks. So in this case, the latency will be 5 or 5 times steps. So once I defined this latency, I can also define what is called a latency sequence.

A latency sequence is the sequence of latencies of successive initiation of tasks. It may so happen that initially I find that I can initiate the second task with latency 5 but it is also possible that the next task that I can initiate after say 2 time steps. That is at $t_0$ I initiate one task, at $t_5$ I initiate the second task then at $t_7$ I can initiate the third task. So in that case the latency sequence becomes 5, 2. So it is the sequence of latencies of initiation of successive tasks. I can also define what is called a latency cycle.

A latency cycle is a latency sequence which repeats that means if I have a sequence like this say 5, 2, 5, 2, 5, 2 like this then this 5, 2 these becomes a latency cycle or I can say 2, 5 is a latency cycle because that is a sequence which repeats. Of course I have to ensure that if I follow this latency cycle for initiation of tasks, this does not lead to any collision. For considering this collision avoidance, I consider another reservation table. Let me assume that I have a pipeline with 5 computation stages. So let us say competition stages 1, 2, 3, 4 and 5. For computation of a particular task, a particular function it makes use of say 9 time steps which are from $t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$ $t_7$ and $t_8$. So my reservation table appears like this.

(Refer Slide Time: 00:06:04 min)

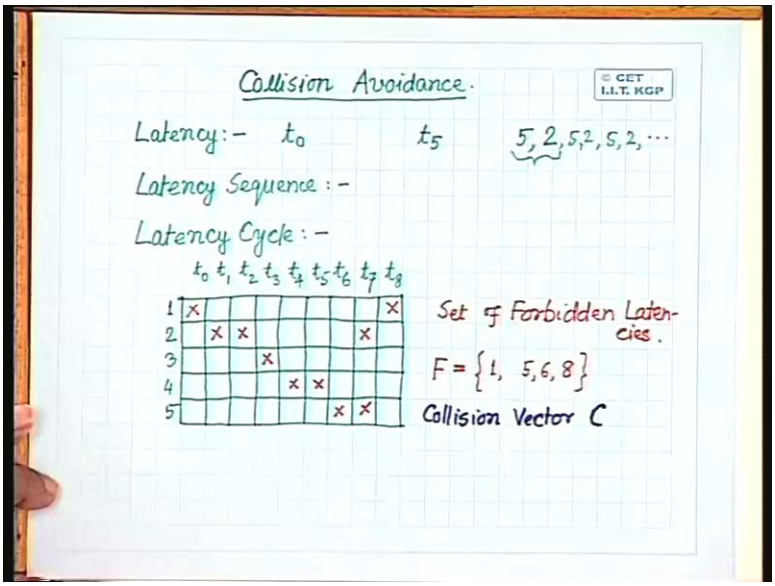



So this is the reservation table for computation of a particular task on a particular pipeline. For simplicity, I will assume that the pipeline is uni-function pipeline that means only one function can be computed on that pipeline. However the concept that I am discussing can also be generalized for a multi function pipeline like the one that we have just discussed where the pipeline can compute two functions a and b with two defined reservation tables. Now even if the pipeline is a uni function pipeline then also the collision can occur. So here at time $t_0$, I initiate a task. Then at time $t_8$, I cannot initiate another task because during time step $t_8$, the first task that was initiated at $t_0$ makes use of the computation stage one. So during time $t_8$, if I initiate the second task then this $t_8$ is nothing but $t_0$ of the second task. So the second task will also make use of the same computation stage one during the same time instant which leads to a collision. If I initiate a task at $t_0$, at $t_8$ I cannot initiate the second task.

Similarly here for two consecutive time steps $t_1$ and $t_2$, the first task make use of computation stage two. So I should ensure that the second task that I initiate should never be initiated in such a way that use of this computation stage two is simultaneous for both task one and task two. I have to identify that what are the time steps, when I can initiate the second and subsequent task by analyzing this reservation table. So for that what I have to do is I have to find out what is called a collision vector from this reservation table.

For finding out the collision vector as I said that the time gap, time step difference between two successive initiation of tasks is called a latency. I have to find out that what are the latencies which are not allowed. That means I have to find out what is called a set of forbidden latencies. I can find out the set of forbidden latencies from this reservation table itself, so that is called a set of forbidden latencies. So that set, I term as set F.

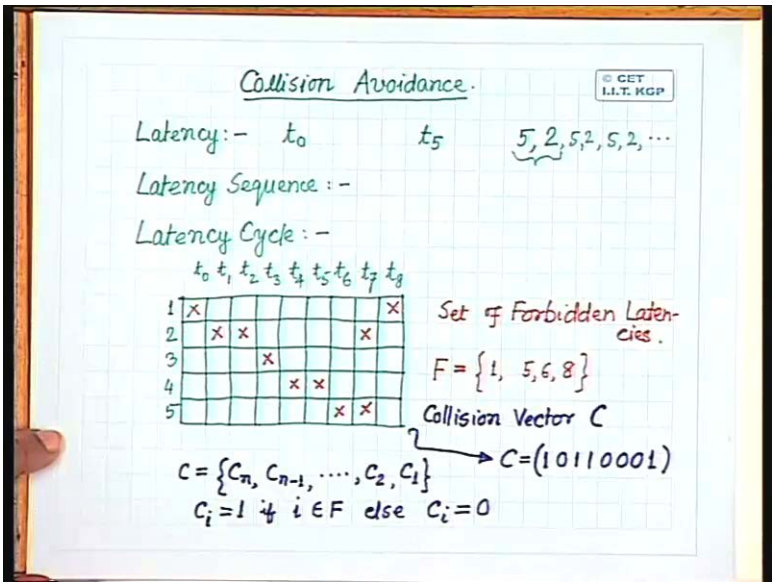
(Refer Slide Time: 00:10:26 min)




Now set of forbidden latencies can be found out very easily. What I have to do is I have to find out that what is the column difference between every pair of crosses in each group. See as I said that in this particular case if I initiate a task at $t_0$, I cannot initiate another task at $t_8$ that means I cannot have a latency which is $t_8$ minus $t_0$ that is 8. So latency of 8 is not allowed that is forbidden. So to find out the set of forbidden latencies, what I have to do is I have to find out the column difference. This 8 is nothing but the column difference between these two crosses in row one. What I have to do is I have to find out the column difference between every pair of crosses in each row of this reservation table. So from the first row, obviously I get a forbidden latency which is 8. From the second row 1, 5 and 6, third row there is only one cross so I don't bother. In the fourth row 1, fifth row is also 1. So the set of forbidden latencies becomes 1, 5, 6 and 8.

From this set of forbidden latencies, what I have to compute is what is called a collision vector. Let us call this collision vector as C, this collision vector is a binary vector. The number of states will be same as the maximum forbidden latency in the set of forbidden latencies. So in this case the maximum forbidden latency is 8. So number of bits in the collision vector C will also be 8. So this collision vector C is of this form C equal to… In general let me put it as $C_n$, $C_{n-1}$ like this, $C_2$ and $C_1$ where obviously n is the maximum forbidden latency.

(Refer Slide Time: 00:13:32 min)



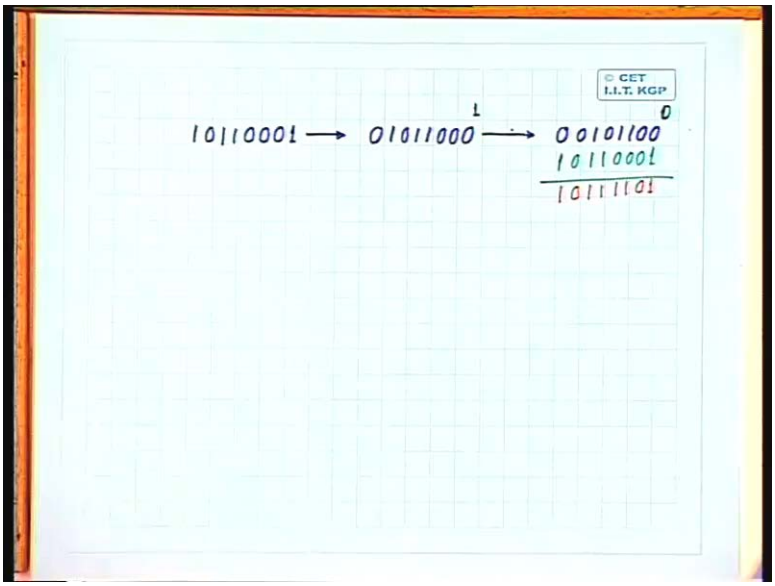

In this collision vector my strategy for deciding, as I said that this collision vector is a bit vector or a binary vector. That means each of these components in this vector can be either be 0 or 1. So my decision will be I will set $C_i$ is equal to 1. A particular bit in this vector $C_i$ equal to 1, if i is a member of F. So in this case 8 is a member of F, so $C_8$ will be set equal to one. 5 is a member of this set, $C_5$ will also be set to 1, $C_6$ will also be 1, $C_1$ will also be 1. Rest of the bits in this collision vector will be equal to 0. So $C_i$ equals to 1, if i belongs to F else $C_i$ will be equal to 0. So in this particular case what will be C, collision vector C? 1 0 1 1 0 0 0 1. So this is the collision vector for this reservation table.

Now see that once I have this collision vector, I can very easily find out that what are the time steps when I can initiate a new task. What I do is whenever I start the first task, this collision vector is loaded in to a shift register. At the end of every time step, you just give a right shift to the shift register. You find that what is coming out of the shift register. If a bit 1 comes out of the shift register then I cannot initiate a new task. If a 0 comes out of the shift register then I can initiate a new task. Now that is not all. This takes care of only initiation of two tasks because whenever I initiate a new task, the collision vector is going to be different because the reservation table of the new task is going to be superimposed with the remaining part of the reservation table of the previous tasks. So I have to re-compute this collision vector to decide when next a new task can be initialized.

So the strategy is simple like this. Whenever you give a right shift to the collision vector, what comes out of the shift register? I will just forget about that. Using that I decide whether I can initiate a task or I cannot initiate a task. The left most bits which are becoming vacant, you fill up those bits with zero. Simply because if this is the only task that is to be executed then when the task is complete say in this case, the task will be completed after 8 shifts. So when the task is complete, at any time I can initiate a new task, that does not lead to a collision. So all the left most bit which becomes vacant because of right shift, you fill up those bits with zero.

So this remaining collision vector that you get, whenever I initiate a new task, I simply bit wise or the collision vector of the new task with whatever is remaining in the shift register and that becomes my new collision vector. So coming to this particular example where I get a collision vector like this 1 0 1 1 0 0 0 1.

(Refer Slide Time: 00:18:02)

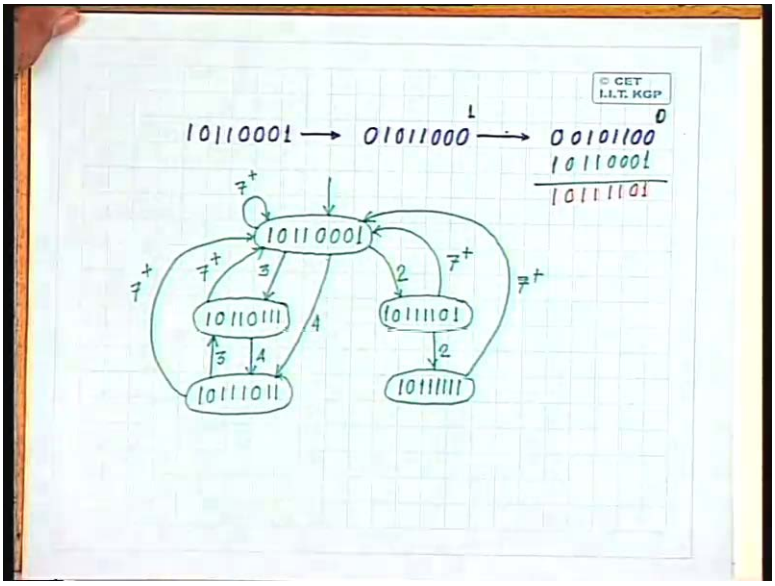



Once the time step $t_0$ is complete, I have to give one bit shift in the right direction to this collision vector. So the new collision vector that I get is 1 0 1 1 0 0 0 and what comes out of the shift register is bit 1 that was this bit. The left most bit which has become vacant, you fill up with 0. Now because a bit of 1 has come out of the shift register, so I cannot initiate a new task. So the new collision vector remains as it is. Then you give the second shift at the end of time $t_1$ and the new collision vector that I get is 0 1 0 1 1 0 0 and a 0 comes out of the shift register from the right most position, the left side I fill up with 0. Now since this is 0, so since a 0 has come out of the shift register from the right most bit so that I can initiate a new task.

The moment I initiate a new task, the new reservation table becomes active and that is superimposed with the remaining part of the reservation table of the first task. That means the collision vector is also going to be changed. So for this what I do is the collision vector for the new task that is initiated will be same as this. So I simply bit wise or these two collision vectors. So the new collision vector that I get is 1 0 1 1 1 1 0 1. So you find that my first latency is 2. The second latency is going to be decided by this. So to find out when a new task, next task can be initiated, I have to perform the same right shift operation on this resultant collision vector. So after one time instant I cannot allocate a new task. I cannot initiate a new task because the right most bit is 1. Only after the two time instants I can initiate a next task because the second bit from the right is 0 and once I do that I initiate a new task. Again this collision vector has to be recomputed and you follow the same logic to find out when again the next task, next to the second one can be initiated. So if I follow this, you find that I can generate a state diagram. If I say that each of this collision vectors represent a state of the pipeline.

I can generate a state diagram of the pipeline and for this particular example, you will find that the state diagram will appear something like this. Our initial collision vector is this one, 1 0 1 1 0 0 0 1. So this is the initial collision vector or the initial state.

(Refer Slide Time: 00:21:58 min)



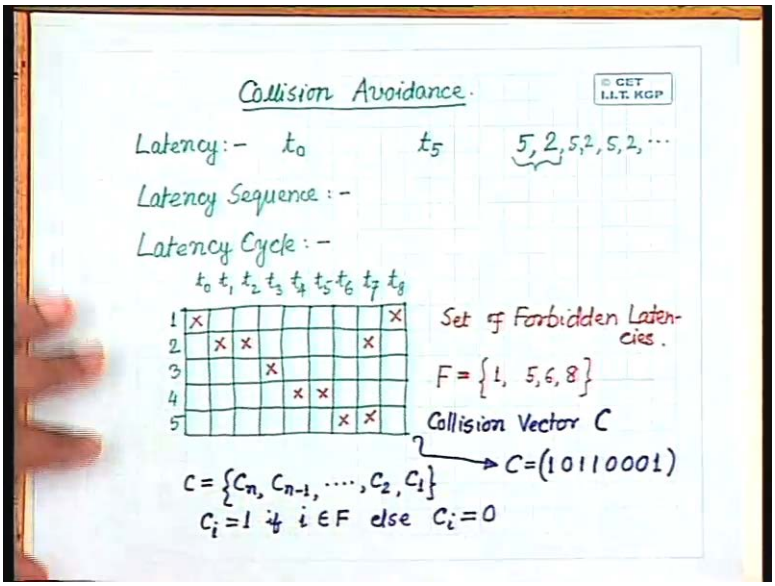

Here we have said that after the latency of two when I initiate the second task, the collision vector becomes 1 0 1 1 1 1 0 1 so that is another state. I can reach from this initial state to this state with a latency two. From here again you find that after a latency two I can initiate another task because the second bit from the right is 0. So if I perform similar thing, you will find that the new state or the new collision vector of this pipeline will be something like this. So this again you get after a latency two. From here after a latency 7 or more, so I put it as 7 plus. 7 or more so I put it as 7 plus. I will come back to the initial latency. Here again after a latency of 7 or more, I will also come back to this initial state.

In this itself after a latency of 7 or more, I will remain in the same state. So I will put it as 7 plus. Now from the initial state, one option was that with latency two, I can initiate a new task and I come to this state and here you find that 2 3 4 they are all zeros. That means I can have a latency 2. If I don't want to initiate a task with latency 2, I can initiate a task with latency 3. I can also initiate a task with latency 4. I can also initiate a task with latency 7. So if I initiate a task with latency 3, if you compute you will find that the new collision vector will be 1 0 1 1 0 1 1 1 and this is the state that you can reach from the initial state with latency 3. With latency 4 again from the initial state, I can reach a state of 1 0 1 1 1 0 1 1 and this is a state that I can reach from the initial state with latency 4.

Now from this state I can come to this state, if you analyze with a latency 4 because the fourth bit from the right is 0. From this state I can go back to this state with a latency 3 because you see that the third bit from the right in this collision vector is equal to 0. So once am in this state, if I initiate a task with latency 3, the next state will be this one. From this state with latency 7 plus, I can go back to this initial state. From this state also with latency 7 plus that is 7 or more to this

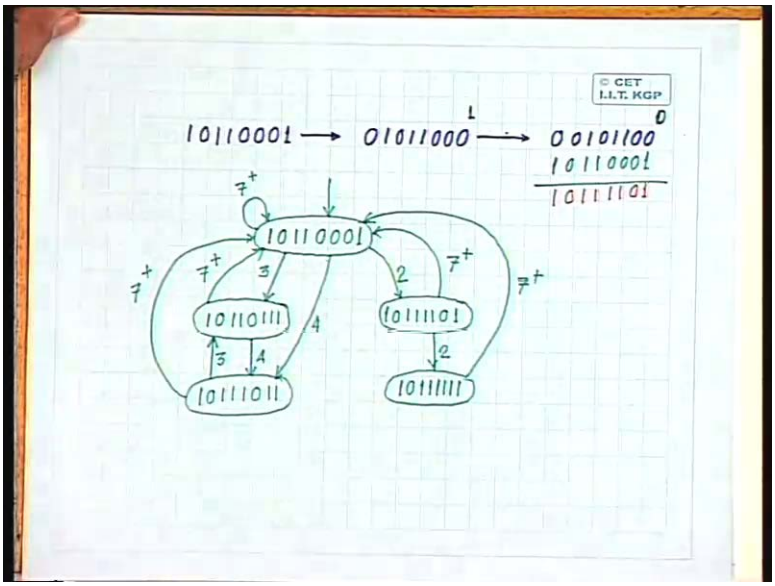
initial state and of course this is the initial state. So as per convention I just put an incoming arrow to this state. So we find that we get what is called a state diagram and this state diagram tells you that all possible latencies or all possible collision vectors that we can have for a pipeline executing a function given by this reservation table.

(Refer Slide Time: 00:26:22 min)




Now this particular state diagram also tells me some more important things because as I said that the latency; as I defined the latency, the latency is the time state difference between two successive initiation of tasks. So naturally our pipeline will be more efficient or the throughput of the pipeline will be more if the latency is less or our latency is less. If the latency is more that indicates that the initiation difference, time difference between initiation of successive tasks is very high that means the throughput of the pipeline will be low. Now you find that from this state diagram I can also find out that what are the states in which I should confine so that throughput of the pipeline will be maximum. You find one more thing that in this state diagram, I have a number of cycles. So those cycles indicate that there is a stable sustainable way in which the tasks can be initiated and that is stable.

(Refer Slide Time: 00:21:58 min)



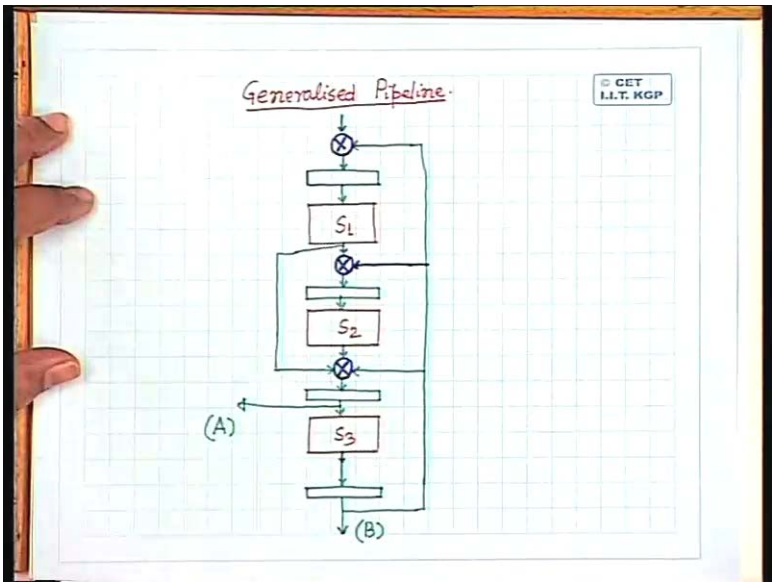

So here I can initiate a task starting from the initial state with latency 2. From this I can also initiate a task with latency 2. After that with latency 7 or more, I can initiate a task and I come back to this initial state and this can be repeated again and again. This also can be repeated again and again. If first I initiate a task with latency 2, after that I initiate tasks with latency 7 or more. This also can be sustained and this is a stable cycle. This is also a stable cycle 3, 7, 3, 7; 4, 3, 7 that is also a stable cycle. 4, 7 this is also a stable cycle, 4, 3 that is also a stable cycle. So from this state diagram, I can find out that if there is a cycle then what is the average latency in that cycle.

Now what is the average latency? I have to find out a cycle with few number of states. I have to find out what is the total latency in that cycle divided by the number of states that is involved in the cycle that gives me average latency of that cycle. So if you do that, you find that if I take just this cycle, here the latency is 2 plus 7 or more that means minimum latency involved with cycle is 9 divided by 2. So average latency is 4.5. If I come here 4 plus 3, 7 divided by 2 that is 3.5 and you will find that the average latency, minimum average latency for any cycle that you get in this is 3.5. So if I decide a scheduling strategy such that I will confine only within these two states. I will not make use of any other state in this state diagram. Of course the initial state I cannot avoid because the first task that has to be initiated puts the pipeline in this state that I cannot avoid. After that with latency 3, I come to this particular state.

Once I am in this particular state, I will initiate the tasks following this latency sequence of this latency cycle 4, 3, again 4, again 3, again 4, again 3. So this way I will continue which gives me an average latency of 3.5 and that is the minimum. So if I follow this scheduling policy on this pipeline, the throughput of the pipeline is going to be maximum. So that is an important property of this state diagram which can be used for optimum scheduling. Now in this particular case I have used only one function that is assumed that the pipeline is designed for execution of one function only and such pipelines are called uni functional pipeline. Whereas the other example

that I have taken or I have said that in this generalized pipeline, I can compute two functions either A, I can compute A, I can also compute B.

(Refer Slide Time: 00:31:14 min)




These kind of pipeline are called multi-functional pipelines. Now as I have seen that in case of uni functional pipeline, I can avoid the collisions following this state diagrams of collision vectors. Similarly for a multi-functional pipeline also, I can generate a similar such thing but that is more complicated. I will not go in to that but only one hint that I will give you that if I use a multi functional pipeline and because it is a multi functional pipeline, any function can be initiated at any time. I can initiate function A, next time I can initiate a function A, I can initiate a function B also. So there what you have to take care is instead of considering the collision vector of one function, I have to merge both the reservation tables and what I have to find out is what is called a cross collision vector. So there I can have collision for two reasons, for initiations of A followed by initiation of A. I can have collision for initiation of B followed by initiation of B. I can have collision for initiation of A followed by initiation of B or I can have collision for initiation of B followed by initiation of A. So all these four different combinations I have to consider and I have to generate what is called cross collision vectors.

So by considering all those cross collision vectors and because there are so many options, in that case I will not have a vector. What I will have is a matrix that becomes a cross collision matrix. So by analyzing the cross collision matrix, I have to determine that what are the latencies which are permitted for which sequence of jobs. So that is a much more complicated task. For this course I am not going to go in to that. So this is a simple situation with an uni functional pipeline, of this tells you that what are the advantages of pipeline that you can extract and what are the problems that you can face and how to solve those problems. So with this I will stop today.