VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
IIIT-Delhi

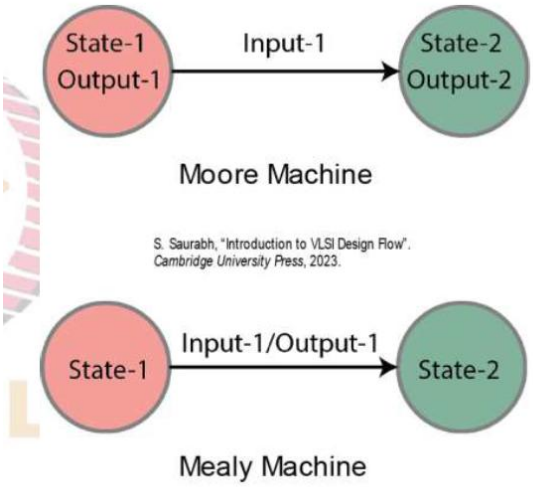
Lecture 20
Logic Optimization: Part III

Hello everybody, welcome to the course VLSI design flow: RTL to GDS. This is the 16th lecture. In this lecture, we will be continuing with logic optimization. In the earlier lectures, we had looked into combinational logic optimization and in today's lecture, we will be looking at sequential logic optimization. Now sequential logic optimization works on the FSM representation of a given circuit. So let us first understand what is an FSM.

FSM is an abstract mathematical model to represent a wide variety of sequential circuits and many other systems, for example, biological systems and many other systems also use FSM for its modeling. Now what does an FSM consist of? An FSM consists of a finite non-empty set of states and that is why it is known as finite state machine. And an FSM also has a finite non-empty set of inputs and outputs and we are given that among the set of states what is the starting state from where the FSM starts. And an FSM also consists of a state transition function and what does the state transition function describe? It describes that given a current state and the input what will be the next state.

So that description is provided by the state transition or state transition function. Now how is the output modeled in an FSM? Now output can be modeled in two ways in an FSM. The first way is that the output depends only on the current state. If that is the case then we say that the FSM is a Moore machine. And the second case is that the output is a function of the current state as well as the current input.

Then if the output depends on both these things then we say that the FSM is a Mealy machine. Now how do we represent an FSM? One of the convenient way to represent an FSM is using state diagram. And what is a state diagram? State diagram is a directed graph. It is a directed graph in which we have vertices which represents the states. So in a state diagram the state corresponds to the vertices in the graph and what does the edge corresponds to? The edge corresponds to the transition from one state to the another state.

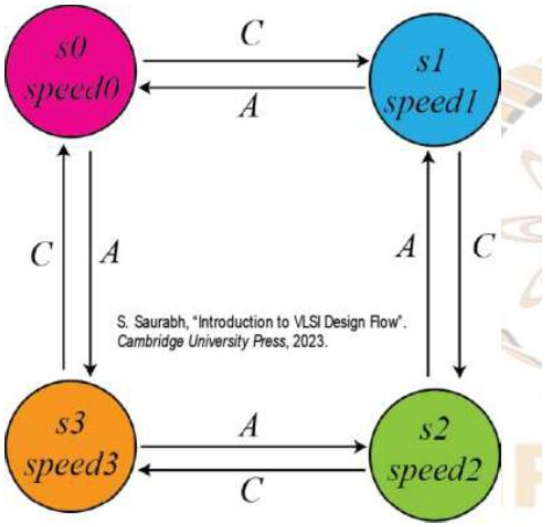


And the edge starts from the current state and it goes to the next state. And at the edge we annotate that what input made this transition from the current state to the next state. For example suppose we have a current state as State-1 and the next state is State-2 when the input is Input-1 then we can represent it in this form. Now how is the output modeled? Now output is modeled by either marking the output at the vertex. For example if this is a Moore machine, so the output depends only on the current vertex or current state and therefore we can mark output directly at the state or the vertex. Now for a Mealy machine the output depends not only on the current state but also on the input. So in that case we mark the output on the edge. So on the edge we say that given a state and the input what will be the output produced and what will be the next state. So now let us look into an example a very simple example to understand what is an FSM. So let us consider a regulator of a fan. Now it can be described as an FSM and we suppose that in this fan regulator there were 4 settings.

We just label them as s0, s1, s2 and s3 these are 4 settings and these we consider them as states s0 state, s1 state, s2 state and s3 state and we say that the initial state is s0 for example we start with the state s0. Now input to the fan regulator is either we rotate it clockwise or anti-clockwise. So the input set contains 2 elements clockwise movement or anti-clockwise movement. And suppose the we say that at state 0 the speed is speed0 and that is the output and at state 1 the speed is speed1, state 2 speed is speed2 and state speed3 speed is 3. So we define the output at each of the states and what are the transitions from one state to another.

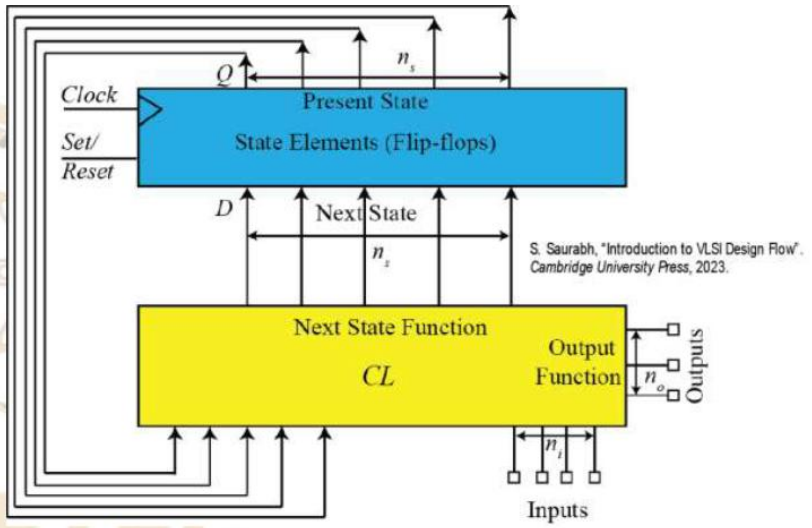
So when we rotate say clockwise then we move from s0 to s1 that is from one setting to the next setting then from s1 to again to s2 to the next state or next setting of the speed and then to the third and again if we rotate clockwise then we again reach S0. So this is

clockwise movement. Similarly there will be a rotate anti-clockwise so whenever from a given state if we rotate anti-clockwise we go from say from s0 to s3 then s2 then s1 and then again reach s0. So these are the state transitions that is defined for a fan regulator. Now how can we represent this in terms of state diagram let us look into that.



So these are very simple representation of the FSM for the fan regulator we have 4 set of states s0, s1, s2, and s3 and if we rotate clockwise we go to s1, if you rotate again clockwise, we go to s2, clockwise, s3, clockwise again reached s0 and similarly we go anti-clockwise. And on each vertices we mark that what is the corresponding output for s0 we have the output as speed0 for s1 we have the speed1 for s2 we have speed2 and s3 we have speed3 so these are the outputs. So this is the finite representation of the FSM of a fan regulator as a state transition graph. Now which kind of FSM is this? Is it a Moore machine or a Mealy machine. Now since the output depends only on the current state we can say that it is a Moore machine.

So now having understood what an FSM is now let us look into that how actually FSM can be implemented in a circuit. So FSM is implemented using flip-flops and combinational circuit elements. So this is a representation of an FSM as it will be implemented in a sequential circuit. So we represent the states by the combination of Q pin value of the flip-flops. So we will have the flip-flops and the output of the flip-flops so this big chunk represents a set of flip-flops and the output of them of this set of flip-flops are denoted as Q.



Now this output of this flip-flop will represent the state or the current state of the FSM and there is a transition function implemented using a combinational logic CL. Now what this transition function is doing is that given the current state and given the current input what will be the next state and that is going to the D input of the flip-flop meaning that whenever the clock will come at the next clock edge the next state will be reached and so on and also the output is being produced by this combinational logic. So now this will be like the way in which an FSM will be implemented in our circuit. Now what kind of optimizations can we do in this circuit. So there are two things we can optimize the first is this flip-flop, set of flip-flops and second is this combinational logic which implement the next state function and the output function.

So the first thing of first optimization that is optimization of the state elements or the flip-flops that is done using a technique which is known as state minimization. So when we minimize the number of states in an FSM then we will be reducing the number of flip-flops required in this implementation and we can save area. And the other thing is that if we choose or find a good encoding of the states. What we mean by encoding, we will just see. We choose a good encoding of our states then we can optimize or improve the combinational logic. We can improve the combinational logic which is required to implement the next state function and the output function. So let us look into both these techniques: the state minimization technique and the state encoding. How we can optimize by doing state minimization and choosing a good state encoding, the FSM implementation and therefore the sequential circuit representation will improve or the area will reduce.

So what is the motivation of doing state minimization? Why do we want to minimize a

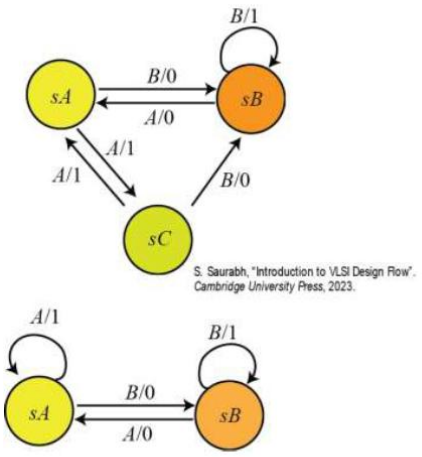
state in our circuit? So to represent an FSM we use a sequence of bits. For example suppose there were three states s1, s2 and s3. Now how do we represent this? We represent it using sequence of bits in this we can say that we represent s1 as 00, s2 as 01 and s3 as 11. So we represent each state as a sequence of bits and for n_s states, n_s is the number of states. For a given number of states how many minimum bits do we require? So the minimum number of bits will be required will be $\log_2 n_s$ and then we take the ceiling of it. In this case there were three states so the minimum number of bits required for this case will be 2.

It will be $\log_2 2=1$ and $\log_2 4=2$. So this $\log_2 3$ will be something between 1 and 2 and if we take the ceiling it will turn out to be 2. So the number of states and the number of bits required are related. So by reducing the number of states we can reduce the number of bits required in an FSM representation. For example if there are say 100 states and we reduce it to say 50 states then the number of bits required for the representation will reduce and therefore we can save in the number of flip flops and area why because the flip flops are used to represent the state bits. For each bit for example in this case there are two state bits.
So for each of them will have one flip flop that represent each state bits. So if we reduce the number of states we will be reducing the number of bits required to represent a state and if we reduce the number of bits required to represent a state then we will be able to reduce the number of flip flops required for an FSM. So that is the motivation of state minimization we want to reduce the number of flip flops in our circuit. So what is the objective of statement minimization? The objective is to derive an FSM that has the minimum number of states and exhibit the same behavior as the original FSM. So we have got an FSM in which there are certain number of states we want to minimize it, reduce it and such that the new FSM that we are getting is functionally equivalent or is functionally same as the original FSM.

And to do state minimization what we rely on is we rely on determining equivalent states. Now what is an equivalent state? We consider two states suppose there are two states s1 and s2 we consider them to be equivalent if these two conditions are fulfilled. The first is that they produce identical outputs for the same inputs. If we give same input to both these states, both these states should produce identical output. And the second thing is that the corresponding next states are the same or equivalent meaning that there are two states. Now if we give same input to these two states then the next state should be the same or should be an equivalent state.

If these two conditions are satisfied then we say that the two states are equivalent and when we find that two states are equivalent then what we can do is that we can retain any one of the equivalent states and remove others and then we update the transition function

to maintain the same behavior as the original FSM. So let us take an example to understand this. Now consider this FSM which is shown here there are three states sA, sB and sC and there are two inputs A and B and two outputs 0 and 1. Now let us consider whether the states sA and sC are equivalent or not. So let us examine whether they are equivalent or not.

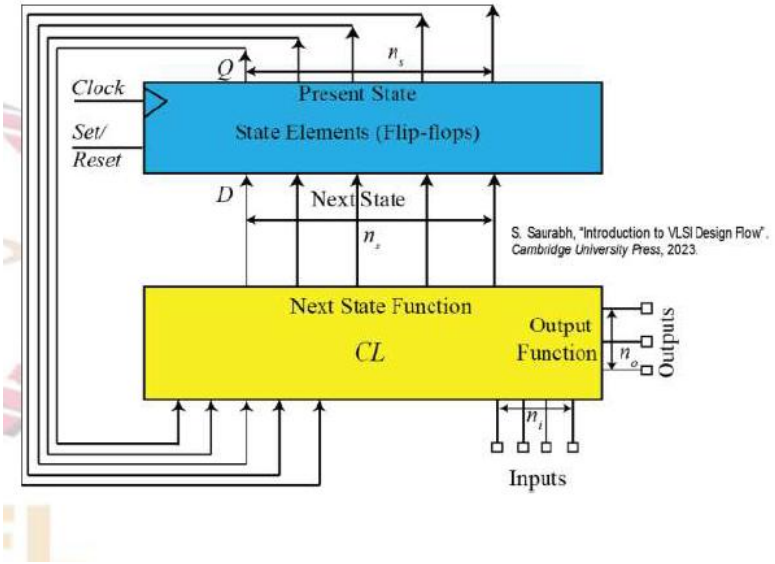


Now first let us look at the output for the state sA and sB. Now at the output for the state sA if we give the input as A then the output is 1 and for the state sC if we give an input as A the output is again 1 so that is same. So when input is A both states produce 1 and when the input is B this produces a 0 and the state sC also produces a 0. So when input is B both states produce 0 and let us look at the next state when input is A and the current state is sA then it goes to sC and we are considering sA and sC as equivalent so it is going to an equivalent state sC. And if we consider the current state as sC and we give an input as A then it goes to sA.

Now sA and sC are equivalent. So when we are giving input as A, sA transition to sC and sC transitions to sA and we are considering sA and sC to be equivalent. So they are transitioning basically to the equivalent state for the input A what about when the input is B. If the input is B then sA is going to sB and sC is also going to sB. So both sA and sC are transitioning to sB and therefore they are transitioning to the same state and therefore the next state condition is also satisfied and therefore these two states are equivalent. Now if these two states are found to be equivalent what we can do is that we can retain one of them one of the states say let us assume that we retain sA and we can remove sC and we can get this kind of FSM. So for the case A when input is A then it is going to the same state.

So this extra edge is created and sC is eliminated. So we could have as well retained sC and eliminated sA that behavior of the FSM would remain the same. So there are efficient algorithms, polynomial time algorithms which can be used to find equivalent set of states for a given FSM and those algorithms are used for state minimization in the logic synthesis. Now let us look into the other problem of an FSM that is state encoding. Now what is state encoding? State encoding basically assigns a binary representation to each state of an FSM.

For example let us consider again a three states s0, s1, s2 there are three states we can assign it as: s0 as as 00, s1 as 01 and this s2 as 11 we could have as well assigned this (s0) as 01, this (s1) as 11 and this one (s2) as 00. So we can assign binary representation to each state in many different ways. Now the problem of state encoding is that how should we assign it such that our FSM implementation is optimal. So how does the state encoding impact the behavior or the area of our circuit. So changing the state encoding can change the next state function and the output function.



So next state function and the output function can change as we change for example this encoding and this encoding are different and for these two different encoding this function CL will be different and one of them will be actually giving us a more compact combinational logic CL compared to the other. So combinational circuit block CL and the associated QoR can change with encoding. So that is what the motivation is for trying out various or looking at various encoding schemes and finding a good implementation or good encoding for our FSM. Now we can also consider the encoding length of our FSM

representation. For example if we consider say the three states s0, s1, s2, we could have represented for example as I said 00, 01, and 11 this is one of the encoding scheme.

So what we have seen that if the number of state is n_s then we need at least $\log_2 n_s$ and the ceiling of that, that many number of bits minimum, that is for N states we require $\log_2 N$ and the ceiling of that, that many number of bits in the FSM implementation or in FSM encoding. But we can always go more than that we can always choose a longer encoding length than what this minimum is. For example for the three case the minimum turns out to be two, that is we need to have at least two bits for the three states, but we can also choose a longer encoding or the encoding length can be longer for example we could say that we do a one hot encoding. So in one hot encoding we reserve one bit of one state so in this there are three states, so there are three bits that will be used. For example will use 001 for the state0, 010 for state1 and 100 for state2 so we are reserving bit, the lowest bit for this state, this bit for this state and this bit for this state. So this kind of encoding scheme is known as one hot encoding, so one hot encoding reserves one bit for each state. Now why will we want to change the encoding length why not always the minimum? So there are complicated dependencies on the encoding length and the number of flip-flops and the input output of the logic block CL.

So of course when we increase the number of bits then the number of flip-flops will increase but what effect it will have on the combinational logic that is not very easy or straightforward to think in terms of the figures of merit for example the area and also the timing. For example sometimes combinational logic block can be simplified by changing the encoding length. For example one hot encoding identifies a state or if instead of say minimal bit, which was say this encoding scheme is minimum and this encoding scheme is one hot. Now in one hot encoding if you want to identify that what is the current state we can just look at one of the bit and find out and identifying whether we are in a current state or not is very easy in one hot encoding scheme. In contrast if you want to identify which state we are in, in other kinds of encoding scheme.

For example in this encoding scheme we have to look at two bits and take the corresponding say min term and then based on that min term we can say that we are in a given state. For example if in this encoding scheme we denote the first bit as a and this as b then the min term which will identify the state s0 will be a'b'. And therefore identifying whether we are in the current state will require more logic in other encoding scheme than in one hot encoding scheme. And as a result what can happen is that one hot encoding scheme can have fewer logic levels between flip-flops and be faster in some cases. And therefore we can try with different encoding length also to optimize our FSM.

Now given an encoding length how to find or choose a good encoding scheme. So there

the challenge is that number of possibilities is huge. Say there are 50 states and there are say 50 state bits then there will be lots of possibilities exponential number of possibilities in which we can assign or encode states with 50 bits. So now brute force cannot be of course applied in these cases to examine all the possibilities and come up with a solution. And the other challenge with this is that though we have discussed first the combinational logic optimization and then sequential logic optimization.

We have done this because it is easy to understand sequential logic optimization after understanding combinational logic optimization. But in practice sequential logic optimization is typically done before the combinational logic optimization. So when we are actually doing sequential logic optimization, at that stage the combinational logic which implements the next state function and the output function that is not yet implemented. So we cannot easily assess the effect of an encoding scheme on the QoR or the figures of merit of the combinational logic block. So we have to choose an encoding scheme in FSM optimization such that in future when combinational logic optimization is done for the block CL that is combinational logic block that is implementing the next state function and the output function that gets optimized.

So during FSM optimization and choosing an encoding scheme we must encode such that in future when this logic block CL is optimized, the optimizer sees lots of possibilities of optimizing and makes this CL as compact as possible. So what does the tool do or how tools approach this problem? So the CL needs to produce the next state function and the output function. So this combinational logic is being used by next state function as well as output function and therefore the same logic elements or circuit elements will be used in multiple outputs, multiple functions. It will be implementing many different next state functions and many output functions and therefore the same CL logic block is used for implementing different logic functions. So we can do the encoding of the states such that the sharing of this logic elements inside this CL element is maximized.

If that happens a lot of sharing happens then this CL can be very very compact and so we should choose an encoding scheme that allows more common cubes and common sub expressions sharing among the various logic elements. If we are able to do that successfully then our encoding scheme will yield an FSM which is compact or having a good figure of merit. So what tools do they take they follow some heuristics algorithm they define what is known as gain which somehow quantifies the possibility of common cube extraction or common sub expression extraction for a given encoding scheme. So given encoding scheme it tries to formulate it or it tries to formulate an expression or function called gain which somehow captures that how much sharing of logic can happen in the element CL for a given encoding scheme. And then it tries to determine an

encoding scheme that maximizes this.

So at a very broad level this is what the tools will do while finding a good encoding scheme. Now if you want to go deeper into these topics I will suggest that please go through these references which are shown in this slide. And just to summarize what we have done in today's lectures we have looked into sequential logic optimization and we have in particular looked into an FSM representation of a sequential circuit and how it can be optimized using state minimization and state encoding. And also let me summarize what we have done in last three lectures. In last three lectures we have looked into logic optimization of a netlist consisting of generic logic gates.

Now given an RTL we did RTL synthesis and we also did logic optimization. Now while doing this transformations in our RTL representation was all this transformation correct in terms of functionality? So initially when we have written an RTL we have done a functional verification using simulation and other methods and we have verified that the initial RTL is functionally correct. Now once we have done this transformations that is RTL synthesis and logic optimization are we still sure that the functionality remains the same? We need to actually ensure it that the logic optimization and RTL synthesis task that was performed that did not introduce any errors.

So we have to somehow ensure the equivalence of the initial RTL and the netlist that we have after RTL synthesis and logic optimization. So this kind of equivalence between these two models can be established using technique which is known as equivalence checking and these equivalence checking is done using formal methods. So in the next lecture we will be looking at various formal verification techniques. Thank you very much.