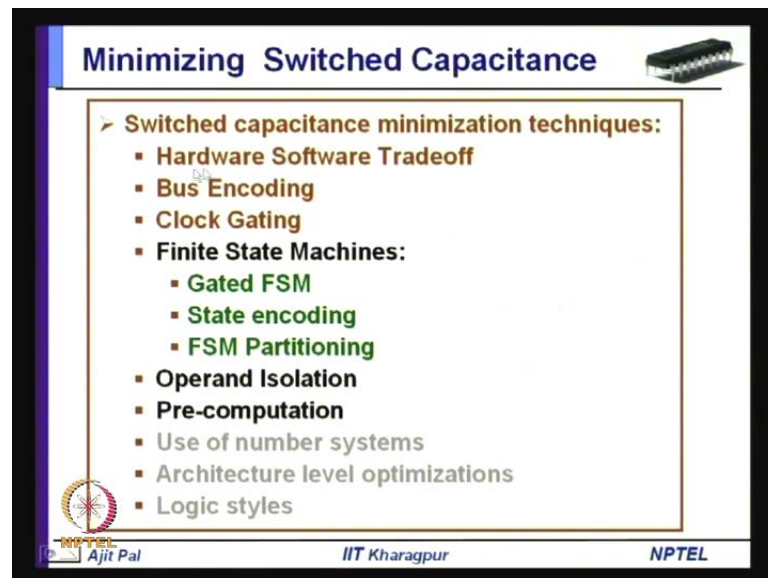


Low Power VLSI Circuits and Systems
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No. # 30
Minimizing Switched Capacitance - IV

Hello, and welcome today is lecture on minimizing switched capacitance, this is the fourth lecture on this topic.

(Refer Slide Time: 00:27)



The slide is titled "Minimizing Switched Capacitance" and features a small image of a microchip in the top right corner. The main content is a list of "Switched capacitance minimization techniques:"

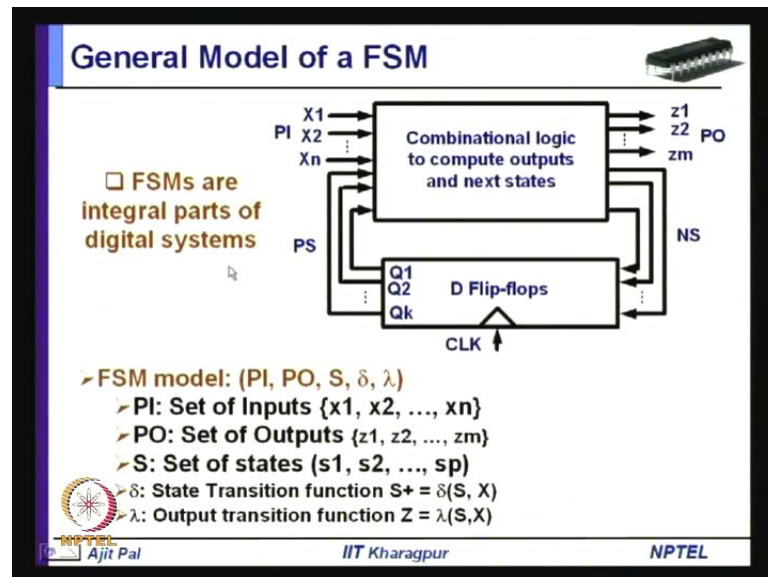
- Hardware Software Tradeoff
- Bus Encoding
- Clock Gating
- Finite State Machines:
 - Gated FSM
 - State encoding
 - FSM Partitioning
- Operand Isolation
- Pre-computation
- Use of number systems
- Architecture level optimizations
- Logic styles

At the bottom of the slide, there are logos for NPTEL, IIT Kharagpur, and the name "Ajit Pal".

And in my earlier lectures, I have discussed about Hardware Software Tradeoff, and we have seen that by replacing hardware by software. We can reduce switched capacitance, particularly capacitance is reduced leading to saving in power dissipation; and in the after that, we have discussed Bus Encoding, where before sending some data over the outside bus, it is encoded to reduce the power dissipation; as a I mean the particularly, the transition activities are reduced by Bus Encoding. And in the last lecture, we have discussed about Clock Gating, where we can introduce getting circuits in the clock tree to reduce power dissipation.

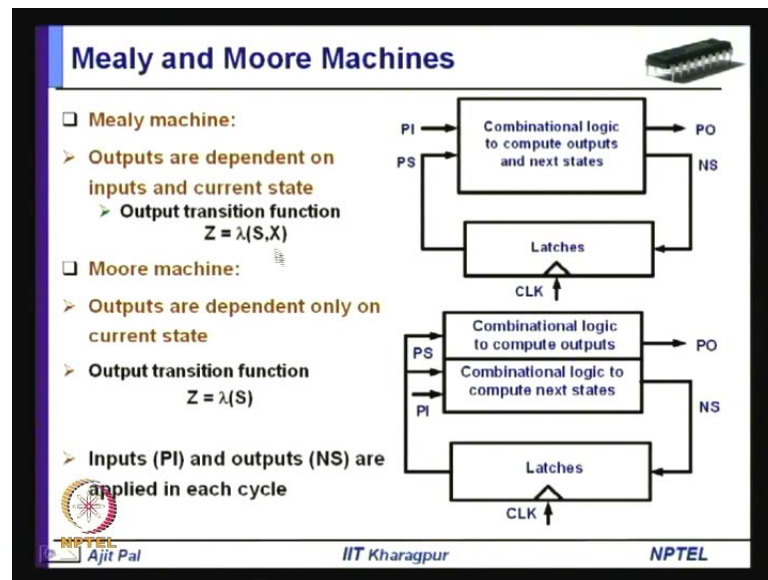
Today we shall focus on finite state machines and three techniques, we shall discuss related to finite state machines that is Gated FSM State encoding and FSM Partitioning; and in addition to that, I shall two more techniques; operand isolation and pre-computation.

(Refer Slide Time: 01:38)



Let me, start with finite state machine as you know finite state machines are integral path of digital systems; whenever you go for synthesizing a digital system, it has got two paths; data path and control path. And essentially, the control path comprises or is implemented with finite state machine. On the other hand, data path comprises ALU, registers, memory bank and so on. And controller path control path is implemented usually, by finite state machines. Now, this is the module of a finite state machine, I have already introduced a finite state machine consist of a set of inputs P I the inputs are X_1 to X_n than it has got a set of outputs z_1 to z_m P O; and a set of states S; and the set of states are s_1 s_2 up to s_p and there are two state transition function. Delta is the state transition function, and as we can see next state is dependent on the present state, as well as the present primary input. And similarly, output transition can also be a function of the present state and the present primary input.

(Refer Slide Time: 03:02)



So, and we have also mentioned about, two different types of machine. One is known as Mealy machine where there where the output transition is dependent both on the present state as well as the present input P I; as well as P S; on the other hand more machines are I mean the output functions is dependent only on the present state not on the inputs. So, as it is shown here, this Combinational logic to compute the output is dependent on the present state, on the other hand next state is dependent of course, on the primary input as well as present state.

Now, whenever we realize a finite state machine conventionally, we apply a clock in each cycle to this latch, and you know the inputs as well as outputs are applied in each cycle; that means, as a at each cycle there is some activation, in this Latches activation of Latches and leading to transitions in inputs and outputs and as a consequence. There is power dissipation in the combinational circuit, as well as in the Latches or registers.

(Refer Slide Time: 04:14)

Clock-Gated FSM

0110 sequence detector

- There are conditions when the next state and output values do not change (idle condition)
- Clocking the circuit during idle condition leads to unnecessary wastage of power
- The clock can be stopped, if the idle conditions can be detected
- This saves power both in the combinational circuit as well as the registers/latches

➤ Ref: Benini et al

NPTEL Ajit Pal IIT Kharagpur NPTEL

Now, let us consider an example 0110 sequence detectors, which we have already discussed here, as we can see there are some conditions. For example this condition, when the machine is in state s 1 and input is 0. There you can see as long as the input is 0, it remains in the state s 1 and output is also 0. That means, there are conditions: when the next state and output values do not change and similarly, here in there is another state s 4 you can see, as long as the input is 1 it remains in state s 4 and output is also 0.

So, we can say that we can consider these two as idle conditions, and in this condition clocking, the circuit during idle condition leads to unnecessary wastage of time. So, you can see the input and outputs are same. So, there is no nearly no need to change the input are the state, because output is remaining same state (()). So, the clock can be stopped, if the idle conditions can be detected. So, this is the basic idea behind this clock detect finite state machines, and this saves power both in the combinational circuit, as well as in the registers and latches. So, this particular concept was proposed by Benin et al in their paper.

(Refer Slide Time: 05:46)

Clock-Gated FSM

- For a Moore machine, a self loop corresponds to the idle condition, when the clock can be suppressed
- Self fsi: $PI \{0,1\}$, such that self fsi (pi) = 1 iff $\delta(x,si) = si$, $pi \in PI$; $Fa = \sum \text{Selfsi. } xi$, where xi is the decoded state variable corresponding to the state si , i.e. xi is 1 iff FSM is in state si
- Selfsi captures the set on input combinations for which self loop for a state is traversed

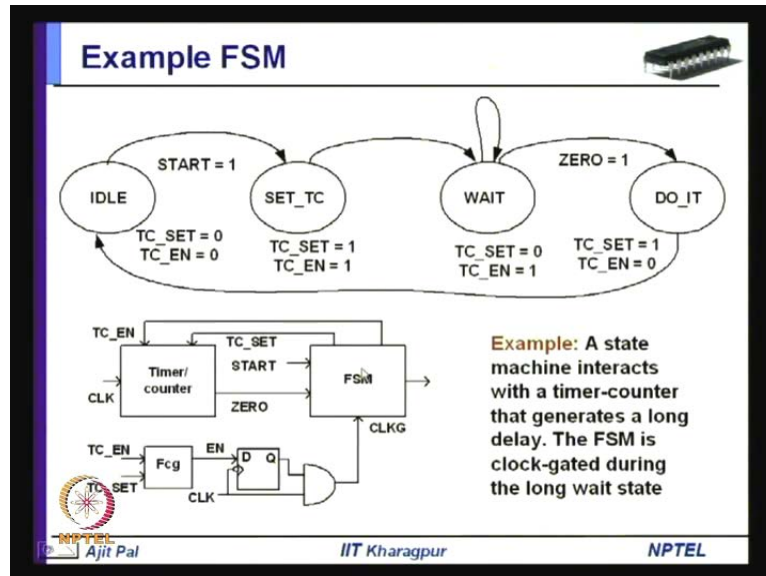
NPTEL Ajit Pal IIT Kharagpur NPTEL

Now, I shall consider a specific case or a Moore machine, for a Moore machine is self loop corresponds to the idle condition, when the clock can be suppressed. So, a self loop like the one that I have shown can be detected, and in that case clock can be suppressed. So, here the self loop self fsi is equal to 1 when the primary input can be 0 or 1. Such that the self fsi, PI is equal to 1, if and only $\delta(x, si) = si$. That means, next state is si and outputs also remain same and PI is part of the primary set of primary ends loop. So, you can find out a function Fa activation function Fa comprising all these self loop conditions, where xi is the **xi the** decoded state variable. Corresponding to the state si and that is all xi is 1, if Finite State Machine is in state si ; that means, if that is xi is 1 if Finite State Machine is in state si .

So, Self si that is the self loop condition, captures all the set of input combinations for which self loop for a state is traversed. Now, what can be done you can identify those self loop conditions, and then you can disable the clock during this period. So, you can apply a gated clock instead of applying, this clock directly to this finite state machine. That means, both the primary input as well as, next state can be gated with the help of this. I mean gated clock can be applied. So, this in this signal this clock will not be applied, when the finite state machine is in self loop, and this Fa identifies all these self loop conditions, and accordingly generates the signal here. So, it is enabled only when the machine is not in the self loop case. That means clock is allowed to applied, on the other hand when this Fa , when the machine is in the idle state.

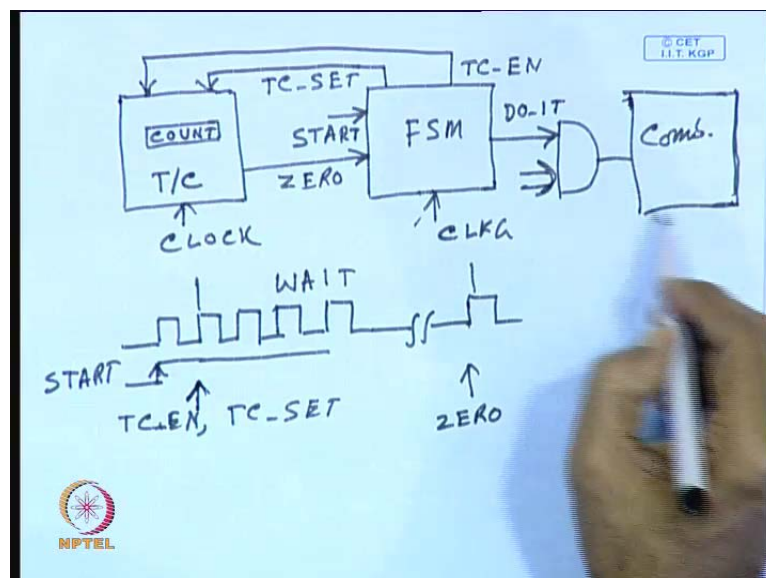
And self loop condition is identified, and then clock is disabled. So, this is the basic idea behind, this clock gated Finite State Machine.

(Refer Slide Time: 08:10)



Now, let me illustrate this with the help of an example. So, here we have taken up an example of a timer counter, which is interacting with a Finite State Machine.

(Refer Slide Time: 08:25)



So, let us assume this is a timer counter as you know a timer counter countdown if some value is loaded in its register, there is an internal register, where the count value can be initialized. Then, it will do the countdown, and this can be this is being interacted with a Finite State Machine. So, this is the FSM, and it is interacting with this FSM can be started with the help of a START signal, and FSM also generates two additional signals. Two signals which are applied to this timer counter, this is a timer counter.

You can say, this is a module and these two signals are one is your timer counter enable TC_EN and another is timer counter set TC_SET, TC_SET means timer counter set. There is a count value can be set, and this timer counter, I mean after timer this timer counter is enabled and TC is SET it will count down the value and it will generate an output, when this countdown becomes ZERO. So, that is why, it identifies the ZERO condition and that ZERO is applied to this Finite State Machine.

Now, it operates in this way the output of the Finite State Machine is generated only, when this ZERO signal is received by the Finite State Machine. And it may really drive a large combinational circuit; you know inputs are applied to this combinational circuit. This is a combinational circuit, and power dissipation in this combinational circuit can be reduced, because only when this ZERO signal is coming then, this input is applied to the combinational circuits otherwise; it is not applied.

So, you can see a lot of parts we can be done and this Finite State Machine also is clock gated as, I have told you will be **you will be** applying clock gating, and this Finite State Machine will remain in the where wait state as long as, this COUNT down will take place. So, let us have a look **at the look at the look** at the state transition diagram of this Finite State Machine. This is the it remains in the idle condition and only when the START signal is applied it goes to a state called SET_TC, and when the SET_TC state is reached it generates these two outputs, this is a Moore machine.

So, this is its outputs, these two **these two** outputs to the timer counter. So, timer counter is enabled and these two signals are applied, and these two signals will after this is done the countdown will continue and the machine will go to this timer and it will go to the Finite State Machine will go to wait state. That means, you have applied a CLOCK here, and a gated CLOCK is also applied so; that means, gated clock is applied to this Finite State Machine. So, this is the clock you know which is applied to the timer counter.

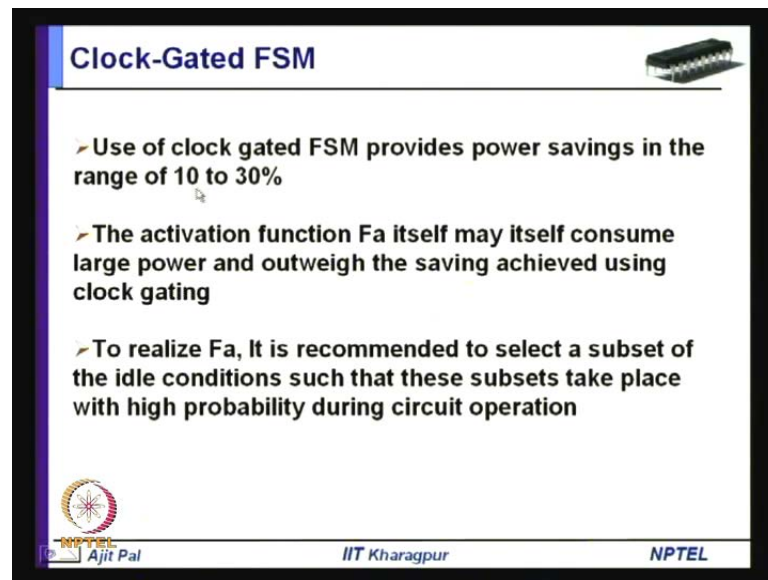
And let us assume, the START signal is applied here. So, this is where the START signal is applied. So, START signal is applied. And that will in the next cycle the Finite State Machine will go to generate this TC_SET and TC enable. So, at this point of time the timer counter is enabled TC enable and TC_SET signals are generated at this point, and now the countdown take place this countdown can take place for a long time. So, you can see this can go on.

And let us assume, this is the point where the this ZERO signal is generated. So, zero signal is generated here, and so this duration is essentially, the WAIT signal during this period. This clock is disabled and Finite State Machine remains in the idle state, and clock is also inhibited, because this output signal is g not generated. So, the output signal can be do it state and this do it state this is generated, and only when this ZERO signal is received the output signal is generated that goes to the Combinational circuit.

So, power is saved here, power is saved in the FSM as well. So, this is being illustrated here, you can see the gating clock gating signal is generated with the help of a hardware F c g which identifies the condition; that means, when the TC_SET is 0 and TC_ENABLE is 1. So, that is why TC_ENABLE and TC_SET with the help of these two signals an enable signal is generated and that is, loaded into this clock getting hardware and this clock is not applied to the FSM unless enable signal comes here.

So, this in this example a state machine interacts with a Timer counter that generates a long delay. So, this Timer counter is essentially generating a long delay that is a COUNT down period and the FSM is clock gated during this long WAIT cycles. So, there is power saving in this Finite State Machine, and also there is power saving in this part of the circuit which the FSM is driving, because this there is no transition here, and this clock the you can inhibit the inputs to this exceeding stage with the help of this output.

(Refer Slide Time: 14:34)



Clock-Gated FSM

- Use of clock gated FSM provides power savings in the range of 10 to 30%
- The activation function F_a itself may itself consume large power and outweigh the saving achieved using clock gating
- To realize F_a , It is recommended to select a subset of the idle conditions such that these subsets take place with high probability during circuit operation

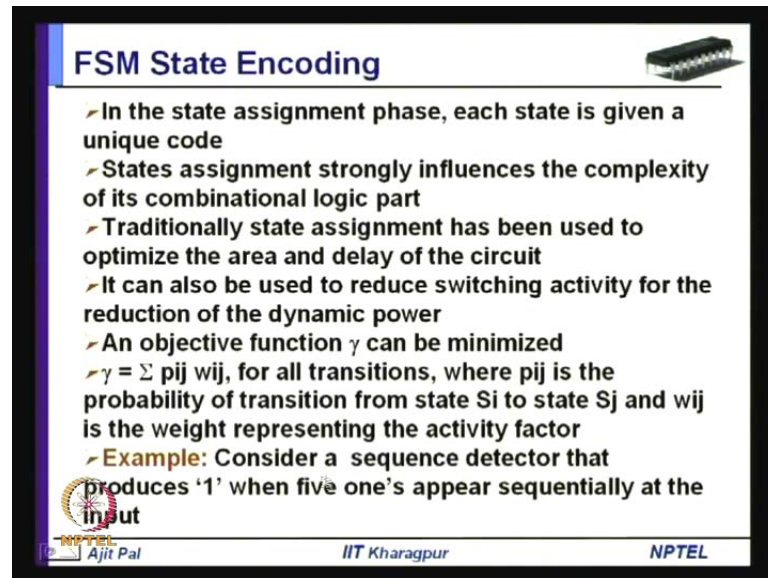
NPTEL Ajit Pal IIT Kharagpur NPTEL

So, this is an example, of a clock gating FSM. So, use of clock gated FSM provides power savings in the range of 10 to 30percent it has been found out by experimental results that this kind of power saving can be obtain by banning and his colleagues and the activation function F_a itself may; however, as you can see you are adding some additional hardware . So, this activation function F_a itself may, it is may consume large power and outweigh the saving achieved using clock gating.

So, this power saving that ,you achieve with the help of this clock gating FSM may be outweigh by the power dissipation of this activation circuit. Which generates F_a that is, the reason why, to realize F_a that is the activation function it is recommended, to select a subset of the idle conditions such that, these subset take place with high probability during circuit operation.

That is number one is this F_a hardware should be small and you have to see that these subsets. I mean this occur at the occurrence of this phenomenon take place with high probability when the circuit is in operation. So, that is how you can achieve reduction of power in this range 10 to 30percent.

(Refer Slide Time: 16:14)



FSM State Encoding

- In the state assignment phase, each state is given a unique code
- States assignment strongly influences the complexity of its combinational logic part
- Traditionally state assignment has been used to optimize the area and delay of the circuit
- It can also be used to reduce switching activity for the reduction of the dynamic power
- An objective function γ can be minimized
- $\gamma = \sum p_{ij} w_{ij}$, for all transitions, where p_{ij} is the probability of transition from state S_i to state S_j and w_{ij} is the weight representing the activity factor
- **Example:** Consider a sequence detector that produces '1' when five one's appear sequentially at the input

NPTEL Ajit Pal IIT Kharagpur NPTEL

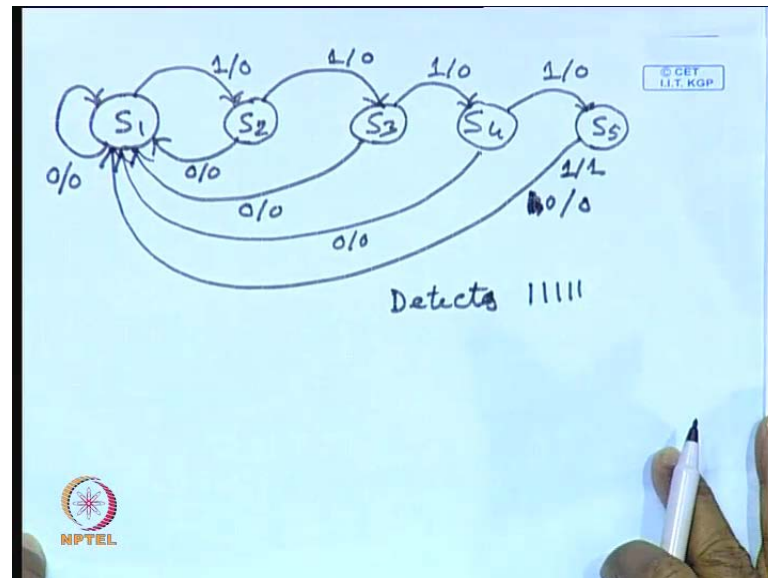
Now, let us consider another example, related to Finite State Machine that is Finite State Machine state on State Encoding. I have already discussed about, Finite State Machine and why when and how state encoding is done as you know, in the state assignment phase each state is given a unique code and state assignments strongly influences the complexity of its combinational logic part and Traditionally, state assignment has been used to optimize the area and delay of the circuit.

As I have told in my earlier lecture that, state assignment is done in such a way that either area is reused or delay is reused in the earlier designs. Now, since power is one of the most important design parameter, and now we can use state assignment for reducing the power consumption in other words. You can use it to reduce switching activity. And so, the basic approach I mean basic requirement here is to do encoding such that, the dynamic power is reduced. So, an objective function γ is decided, which can be minimized.

Where γ is equal to the summation of P_{ij} into w_{ij} for all transitions where P_{ij} is the probability transition from state S_i to state S_j , where w_{ij} is the weight representing the activity F actor. So, we shall try to minimize this objective function that will essentially reduce that switching activity in the Finite State Machine.

Let me, illustrate this FSM encoding for reducing power dissipation with the help of a simple example consider a sequence detector that produces a one, when five one appear sequentially at the input.

(Refer Slide Time: 18:10)



And the state the let me draw, the state transition function of the Finite State Machine. It can start with you can start with state S 1 and the it has to detect the in occurrence of five ones. So, long as it is in 0 it can remain in the state S 1. So, it will produce 0 with input 0 then it will go to state S 2 when a 1 occurs. So, we by producing 0 and it will go to state S 3 if another 1 occurs; however, if a 0 occurs then it will go back to state S 1; that means, 0 by 0 and similarly, from state S 3 it will again go to state S 1 if a 0 occurs. So, 0 by 0; that means, output is 0 for the input 0.

Now, it will go to state S 4 again, if another 1 comes. So, 1 0 and then it will go to state S 5 we will require 5 states for the occurrence of 5 1(s) 1, 2, 3 and 4,4 1 and if, a 0 comes again it will go back to in this case it will go back to 0, 0 by 0 and in this case when it is in a state S 5 a respective of whether a 1 or 0 comes it will produce whenever; 1 comes it will produce 1 and whenever; 0 comes it will produce 0. So, it will go back to state S 1.

So, this is the state transition function of this of this Finite State Machine, which detects a sequence detects occurrence of five 1(s)and whenever; five1(s) occurs as you have seen it produces a 1.

(Refer Slide Time: 20:27)

FSM State Encoding

Table-1: Assignment 1

| State | Encoding |
|-------|----------|
| S1 | 000 |
| S2 | 111 |
| S3 | 001 |
| S4 | 110 |
| S5 | 101 |

Transition Activity: 10.0

Table-2: Assignment 2

| State | Encoding |
|-------|----------|
| S1 | 000 |
| S2 | 001 |
| S3 | 011 |
| S4 | 010 |
| S5 | 100 |

Transition Activity: 5.5

NPTEL
 Ajit Pal IIT Kharagpur NPTEL

So, in this case we can do state, we can **we can** perform state assignment and let us, consider two different state assignments an Assignment 1 let me show it here. So, state this is the Assignment 1 where S 1 is given 0 0 0; S 2 is given 1 1 1; S 3 is gives 0 0 1; s 4 is given 1 1 0 and S 5 is given 1 0 1. So, these are the this is the encoding state Assignment that is done in Assignment 1, and in Assignment 2 for the same states different codes are given for S 1 0 0 0; S 2 0 0 1; S 3 0 1 1 and S 4 0 1 0 and S 5 is 1 0 0 and in this particular, case the switching activity transition activity that parameter that, I have told earlier gamma value for this assignment is 10 and for this assignment is 5.5.

(Refer Slide Time: 21:13)

Assignment 1

| | |
|-----------------------|----------------------|
| $S_1 \rightarrow S_1$ | $0.5 \times 0 = 0.0$ |
| $S_1 \rightarrow S_2$ | $0.5 \times 3 = 1.5$ |
| $S_2 \rightarrow S_1$ | |
| $S_2 \rightarrow S_3$ | |
| $S_3 \rightarrow S_1$ | |
| $S_3 \rightarrow S_4$ | |
| $S_4 \rightarrow S_3$ | |
| $S_4 \rightarrow S_5$ | |
| $S_5 \rightarrow S_4$ | |
| $S_5 \rightarrow S_1$ | |

Assignment 2

Detects 1111

NPTEL
 I.I.T. KGP

Let me show it, how I have obtain that this is, this corresponds to Assignment 1 and let me write down another table this is, this correspond to Assignment 2, as you can see there are 9 possible transition 1, 2, 3, 4, 5 and 6, 7, 8, 9 possible transitions. So, one is your S 1 to S 1 another is S 1 to S 2 then, S 1 to S 1 to S 1 to S 1 S 1 to S 2 then, S 2 to it can go to S 1 and S 2 to it can go to S 3; then, S 3 to it can go to S 3 to S 1 then S 3 to it can go to S 4 and from S 4 it can go to S 1 or S 4 it can go to S 5 and from S 5, it always goes to S 1.

So, these are the five possible transitions and let us, see which is what the switching activity for these 2 assignments is. So, whenever it goes from S 1 to S 1 the probability of transition is .5 and 0.5 and a since, the for the code here, as we have seen the codes are in these case is 0 0 0 and 1 1 1. So, the switching activity I mean. In Fact, are from S 1 to S 1 is 0; that means, it will be 0. So, it will be 0. 0 whether; from S 1 to S 2, when it goes probability of transition is 5 and switching activity is 3 because this 0 0 0 and here, it is 0 0 1. So, it will be 1.5,

So, in this way from S 2 to S 1 the for the code for S 2 is as you have seen 1 1 1.

(Refer Slide Time: 23:27)

Transition Activity

| State | Encoding |
|-------|----------|
| S1 | 000 |
| S2 | 111 |
| S3 | 001 |
| S4 | 110 |
| S5 | 101 |

| State | Encoding |
|-------|----------|
| S1 | 000 |
| S2 | 001 |
| S3 | 011 |
| S4 | 010 |
| S5 | 100 |

| Transitions | Assignment-1 | Assignment-2 |
|-------------|--------------|--------------|
| S1→S1 | 0.0 | 0.0 |
| S1→S2 | 1.5 | 0.5 |
| S2→S1 | 1.5 | 0.5 |
| S2→S3 | 1.0 | 0.5 |
| S3→S1 | 0.5 | 1.0 |
| S3→S4 | 1.5 | 0.5 |
| S4→S1 | 1.0 | 0.5 |
| S4→S5 | 1.0 | 1.0 |
| S5→S1 | 2.0 | 1.0 |
| Total | 10.0 | 5.5 |

NPTEL IIT Kharagpur

Let me show it here, itself where you have got all, I mean codes as well as the transitions. So, you can see here, whenever; it goes from S 1 to S 2 probability is .5, and since it is going from S 1 to S 2 there is no transition. So, it is there is no switching activity. So, it will be 0. And S 1 to S 2 the number of transitions is three and probability

is 0.5. So, 0.5 into three is 1.5 on the other hand, for this code you can see the hamming distance is only one.

So, transition is only one. So, 0.5 into 1.5, So for this transition for Assignment 2. The transition activity is 0.5 similarly, from S2 to S2 S2 to S1 S2 to S1 the S2 to S1, the in this case transition activity is three. So, as for Assignment 1.5 into 3 is 1.5 and here, it is 1 into 0.5, 0.5, and similarly from S2 to S3 S2 to S3 here, the number of transition is 2. So, 2 into 0.5 is equal to 1.


And in this case, S2 to S3 number of transition is only 1 0 to 1 other bits it do not change and. So, it is 0.5 S2 to S3 and S3 to S1 S3 to S1 in this case the number of transition is two and In this case number of transition is also two. So, S3 to S1 S3 to S1 the number of transition is 1. So, it is 0.5, but here, it is 2 S3 to S1. So, it is 1.5 into 2 is 1 S3 to S4 S3 to S4 in this case S3 to S4 number of transition is 3. So, 3 into 0.5 is 1.5.

So, in this way you can find out for all these transitions and for from S5 a, I have told it always goes to S1. So, you can see from S5 to S1 either for 0 input or for 1 put it will go to S1. So, probability is 1 and in this case it goes the number of transition is 2. So, 2 into 1 is 2 and on other hand, in this particular case 0 0 0 and this is 1 0 0. So, the number of transition is 1 into 2, 1 into 2 is 1. So, one into this is 0.5 plus 0.5 into 1 so, number of transition is 1s. So, it is 1

(Refer Slide Time: 26:28)

FSM State Encoding

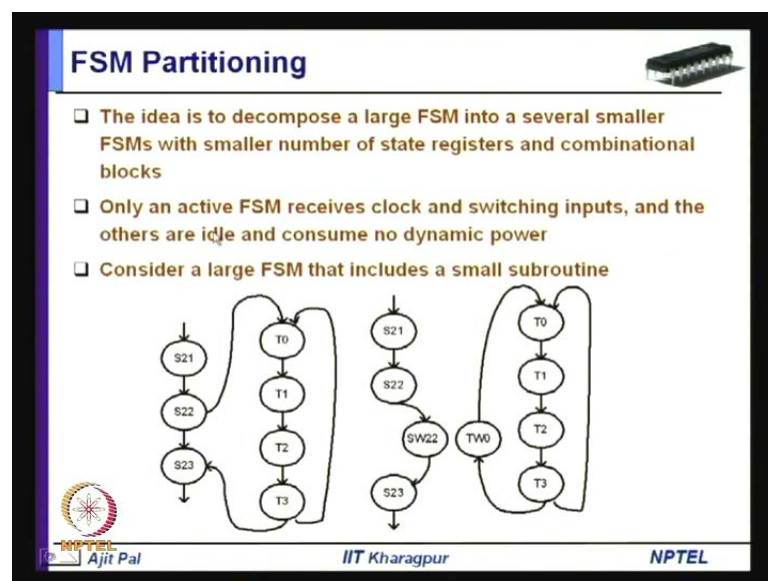
- In the state assignment phase, each state is given a unique code
- States assignment strongly influences the complexity of its combinational logic part
- Traditionally state assignment has been used to optimize the area and delay of the circuit
- It can also be used to reduce switching activity for the reduction of the dynamic power
- An objective function γ can be minimized
- $\gamma = \sum p_{ij} w_{ij}$, for all transitions, where p_{ij} is the probability of transition from state S_i to state S_j and w_{ij} is the weight representing the activity factor
- **Example:** Consider a sequence detector that produces '1' when five one's appear sequentially at the input


Ajit Pal
IIT Kharagpur
NPTEL

So, we have taken into account the possible all possible transitions and multiplied with the probability along with the at transition weight age that is p_{ij} and S_{ij} . Which I have already mention this p_{ij} gamma p_{ij} into w_{ij} and found out this table. And if, we can take this summation, we get gamma is equal to 10 for the first assignment and for the second assignment it is 5.5. So, obviously, this assignment 2 is superior.

So, this illustrates that by doing is state assignment, you can reduce the transition activity of a finite state machine, and I am although it cannot be claimed that the this particular assignment is best one, but because this state assignment is a (O) problem and you have to find out all possible assignments to find out which one gives you minimum transition activity, but this is just an example, to illustrate that this particular assignment is reducing the transition activity to 5 .5. And obviously, the dynamic power dissipation for this assignment will be about 55 percent of the previous case for this assignment.

(Refer Slide Time: 27:40)



So you have discussed the state assignment now we shall come to another important concept that is FSM Partitioning, again this was introduced by barning and his colleagues. So, here the idea is to decompose a large FSM into several smaller finite state machines with smaller number of state register and combinational blocks. And only an active s m FSM receives clock, and switching inputs and the others are idle and consume no dynamic power. So, in this case as you can see, this is the original finite state machine and we can say this finite state machine. This state transition tag now this particular part

corresponds to the main loop main program and this can be **can be** considered as a kind of subroutine.

So, this from here, it can go to the subroutine it is starting state T 0 and it can remain the subroutine for some time and then of course, it will return to the main process. So, you can see, this particular this is a large FSM. It can be decomposed into two FSM and by adding to intermediate states as you can traditional states. So, we have I introduced twostates SW22 in the main sub-route main program and in the subroutine part. We have introduced another state two.

So, these two states actually leads to kind of transition from this machine to another machine; that means, the activation of one machine to other machine. So normally, when this FSM this particular FSM is running this particular FSM remains in the idle state similarly, when this particular FSM second FSM is running the main FSM remains in idle state and of course, this transition from idle condition to active condition take place through these two states SW22 to and two; that means, when the machine can go cannot send signals to this FSM and it can go to this subroutine and this can become active and this can remain in the this can remain in the idle state.

So, we find that in this case by decomposing a large FSM in to a number of smaller FSM you can save power, because at a time one of the FSM will be will remain in active conditions others will remain in idle conditions and; however, these two these FSM may be interacting with each other as, you have seen they will send signals, one FSM can put another FSM to active a condition or can put and whenever; an active I mean whenever it goes from one FSM to another FSM; obviously, it will go to idle state and the other FSM will become active. So, that is what will happen in this particular case.

So, this is illustrated with the help of a simple example, by decomposing a single FSM into two FSM, but you can have several. I mean, you can decompose into several FSM and basic idea is that, you can identify a large FSM and enclose, and find outs smaller subroutines; and which have high probability of transitions, and then you can divide them into several FSM in this way by adding some extra states.

So, this addition of states will; obviously, requires some additional hardware, but since one of them will be active at a time they will be saving in power dissipations. So that means, without with additional area you are able to reduce the switching activity, in this

FSM partitioning. So, we have discussed FSM partitioning and this is a third technique that can be used in the synthesis of finite state machines to reduce switching activity. Now, we shall consider another very important concept that is used to reduce switching activity that is called operand isolation.

(Refer Slide Time: 32:09)

Operand Isolation

- Operand isolation is a technique for power reduction in the combinational part of the circuit
- Here the basic concept is to 'shut-off' logic blocks when they do not perform any useful computation
- Shutting-off is done by not allowing the inputs to toggle in clock cycles when the block output is not used.

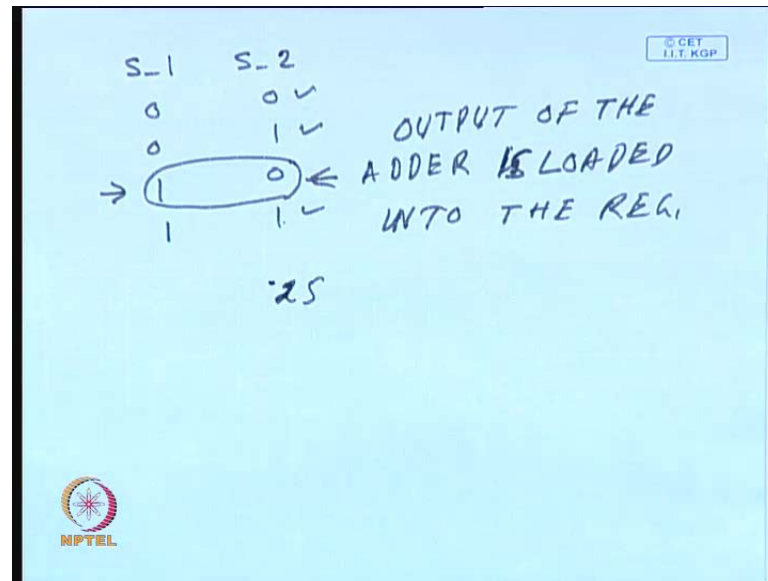
The diagram shows a circuit for operand isolation. It consists of an adder block with inputs A and B, a multiplexer with select inputs S₁ and S₂, and a register D. The adder output is connected to the multiplexer. The multiplexer output is connected to the register D. The clock input CLK is shared by both the multiplexer and the register. The diagram shows the multiplexer selecting the adder output when S₁ is 1 and S₂ is 0.

NPTEL
Ajit Pal
IIT Kharagpur
NPTEL

What is operand isolation? Operand isolation is a technique for power reduction in the combinational part of the circuit. So, in a combinational part of a circuit, you can reduce power dissipation. So, for we have discussed about power reduction in sequential circuits or registers, now we shall consider, how power reduction can be done in combinational circuit and in this case the basic concept is to shut off logic blocks when they do not perform any useful computation.

So in this case, **it will** we shall find out the condition. When it is not doing useful computation for example, this particular adder is producing some result and it is receiving inputs A and B. And this result is loaded into a register shown, by D and only when S₁ is 1 and S₂ is 0.

(Refer Slide Time: 33:28)



That means, when this condition is satisfied only then, the output of this adder is loaded into this register otherwise; it is not used at all; that means, if we consider three possible three four possible condition say S 1 S 2 can have four possibilities 0 0 0 1 1 0 and 1 1. Now, we find that when S 1 is 1 and S 2 is 0 only during this condition the output of the adder is loaded output of the adder is loaded into the register. So, this is happening only when this is the condition.

That means the probability of loading the output of the adder into the register is only 0.25 or 25 percent, because out of four possibilities only in one possibility the output is loaded. That means, if, you can prevent the inputs to reach the adder, during the other conditions then there will be no unnecessary consumption of power in the adder for these three conditions. Only when the S 1 is 1 and S 2 is 0 then, occurrence can be loaded into the adder and it will do the computation and result will be loaded. So, this is the basic idea.

So, shutting off is done by not allowing the inputs to the inputs to toggle the clock cycles toggling clock cycles, when the block output is not used. So, this is the basic idea.

(Refer Slide Time: 35:19)

Operand Isolation

- The output of the adder is loaded into the latch only when S_1 is 1 and S_2 is 0
- To perform addition only when the output of the adder is loaded into the latch, an activation signal AS is generated to isolate the operands using AND gates

Although operand isolation reduces power, it introduces timing, area and power overhead

NPTEL
Ajit Pal
IIT Kharagpur
NPTEL

And let us see, how this is realized. So, the output of the adder is loaded into the latch. When S_1 is 1 and S_2 is 0 as, I have explained. So, to perform addition only when the output of the adder is loaded into the latch an activation signal as is generated to isolate the operands using AND gates.

So, you can see a simple hardware is shown, this is the adder and gate and an inverter which can generate the activation function, and you can see this operands A AND B are ended although only two ends are shown; obviously, if it is end bit you will require a I mean end two AND gate end. I mean you will require two AND gates for each bit. So, you will require two end AND gates to apply the input to this adder. And you can see, this activated signal is active only for S_1 is equal to 1 and S_2 is equal to 0, and only then these two operands will be applied to this adder and; obviously, this signal will go to this point.

For all other conditions inputs are not allowed to reach this adder, and obviously, unnecessary computation is not done and unnecessary power dissipation do not take place. However, whenever you do operand isolation it reduces power, it introduces timing area and power overhead as you can see, if it this particular adder is in the critical path. Then addition of these AND gates will introduced some delay. So, this may disturb the timing conditions and obviously, it will also add some area you are adding this

activation signal hardware and also, you are adding this AND gates; obviously, this will add to some area and power overhead.

So here, the tradeoff is the power saving that is achieved by using operand isolation is more than the power that is, consumed by the hardware that generates the activation signals and the additional hardware that you are using; obviously, it is a this tradeoff has to be satisfied. Otherwise, it will not have any benefit. So, you have to find out this condition using very simple hardware, so that this tradeoff is satisfied and obviously, you are adding some additional area to achieve reduction in the power dissipation. So, it is your trading area for lower power and by reducing the switching activity. So, this is the technique of operand isolation. Now, we shall consider another very interesting technique that is called pre-computation

(Refer Slide Time: 38:20)

Pre-Computation

- ❑ Pre-computation is a technique in which selective computation of output values is done one or few cycles in advance using a much simpler circuit than the original circuit
- ❑ Pre-computed values are used to reduce switching activity in the subsequent cycles
- ❑ Consider a combinational logic block sandwiched between two registers R1 and R2

The diagram shows a data path starting with an 'INPUT' arrow pointing to a register 'R1'. An enable signal 'E1' is shown as an upward arrow to the bottom of R1. The output of R1 goes into a 'Combinational logic block'. The output of this block goes into another register 'R2'. An enable signal 'E2' is shown as an upward arrow to the bottom of R2. The final output of R2 is labeled 'f'. There is a small image of a microchip in the top right corner of the slide.

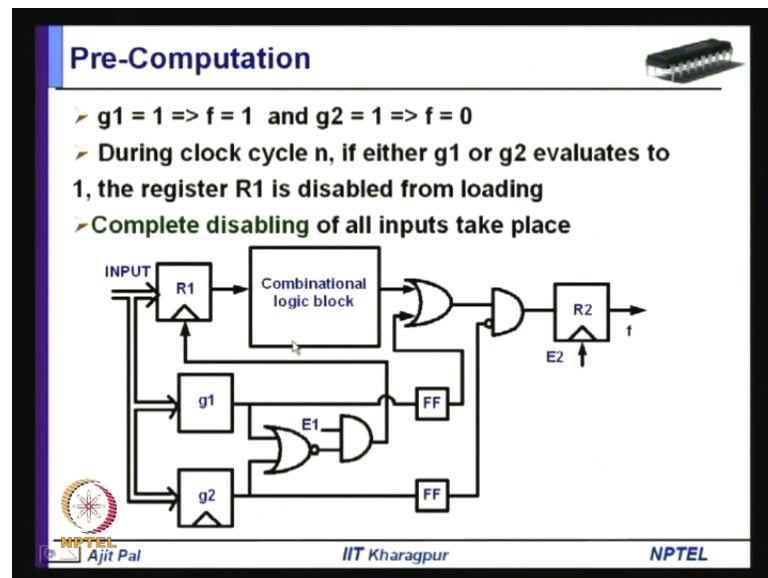
NPTEL IIT Kharagpur

So, Pre-computation is a technique, where you know you perform computation in which selective computation of the output values is done one or few cycles in advance using a much simpler circuit than the original circuit. So, your original circuits which may be performing very complex operation say, it may be performing multiplication of end bits addition of end bits or it may be something more complex function. So, it will identify some simpler hardware, and which will do some Pre-computation little in advance, and then, what you will do you will use this Pre-computed values to reduce switching activity

in subsequent cycles. That is, the basic idea of Pre-computation and Pre-computation is done by simpler circuit as, I have told.

And to illustrate this example let us, consider a Combinational logic block sandwiched between two registers R 1 and R 2. So, this is a simple generalized circuit to illustrate the concept of Pre-computation. So, here, is a large Combinational block which is receiving input through a register R 1 and it has got enable signal E 1, and output is it is generating is f through; another register R 2 which has enable signal E 2. So, this is the original hardware.

(Refer Slide Time: 39:50)

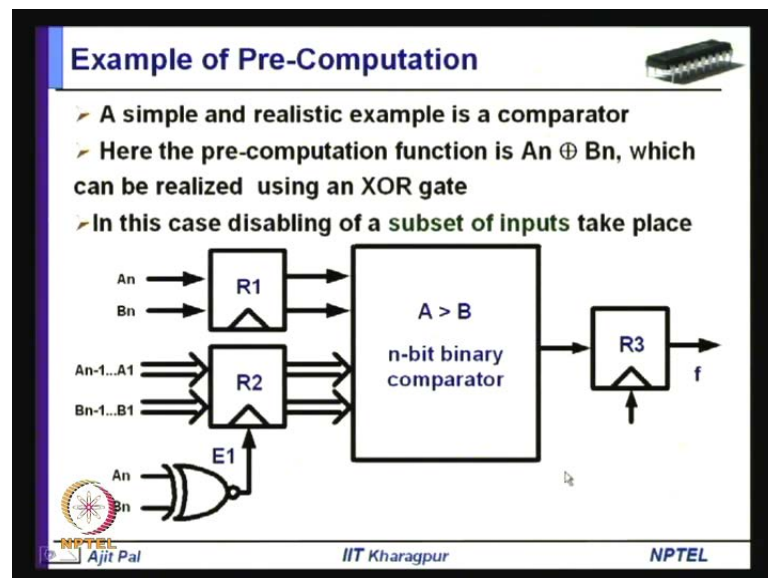


Now, pre-computation can be done by adding to additional hardware much simpler than this Combinational logic block g 1 and g 2 g 1 is 1 when f 1 is 1. So that means, it will this Combinational circuit, when it is 1 this g 1 is 1. So, by using a very simple hardware you are able to find out that and g 2 is 1; when f is equal to 0. So, g 1 is equal to 1 and g 2 is equal to when this Combinational circuit is 1 or 0. Now, what you can see, this you can find out by using simpler hardware. So, these outputs from g 1 or from g 2 can be channelized to the output depending on the input combination; that means, **which condition is satisfied** whether this condition is satisfied g 1 is equal to 1 for when f 1 is equal to 1 or g 2 is equal to 1.

When f is equal to 0, and that can be transmitted and send to this output, and these two signals. These two output g_1 and g_2 ; can be used to generate some activation signal for enabling, this latch or register R_1 . So that means, you can disable this input to reach when computation is done, when either this is one or this is one. That means, when g_1 is 1 or g_2 is 1. You are disabling this input to reach, this Combinational block and in those situation outputs. You are taking from the output of g_1 or g_2 instead of taking the output from this Combinational logic block.

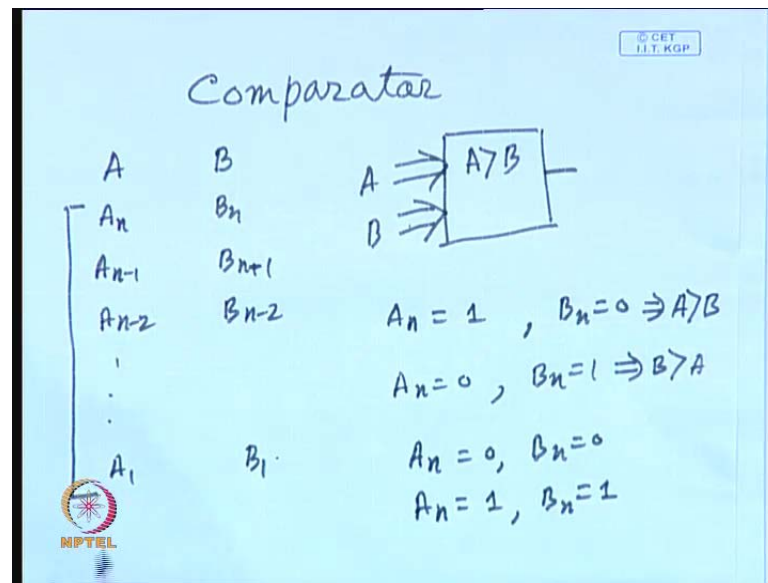
That is, the reason why, we are using this additional hardware to take outputs either from the output of g_1 , or from the output of g_2 and that is, taken out a through another register R_2 , which has enable signal E_2 .

(Refer Slide Time: 41:55)



Now, this particular Pre-computation can be illustrated, with the help of a simple example, very simple, but interesting example you are Familiar with the function of function of comparator.

(Refer Slide Time: 42:11)



You are performing comparison of two numbers A and B. So, both A and B are of n bits. So, $A_n, A_{n-1}, A_{n-2}, \dots, A_1$; $B_n, B_{n-1}, B_{n-2}, \dots, B_1$. In this way, it is up to A_1 and here, you have got B_1 . So, both A and B are 2 inputs to a comparator. So, this comparator will be receiving normally, two inputs A and B may be through two registers, A and B and it will perform the comparison A greater than B or not. Now, you can see whether A is greater than B or not that can be found out from these two bits. If they are 1 0 or 1 1; that means, suppose A_n is equal to 1 and B_n is equal to 0 then, you know that number A is greater than B without looking into these bits the remaining bits.

Similarly, if A_n is equal to 0 and in this particular case A is greater than B and when A_n is equal to 0 and B_n is equal to 1 that implies B is greater than A. So, you can see these two conditions can be found out without looking at these bits; remaining bits; these most significant bits are sufficient to find out the conditions. However, when A_n is equal to 0 and B_n is equal to 0 or A_n is equal to 1 and B_n is equal to 1 in these two cases; obviously, whether A and B is greater A is greater than B or B is greater than A, for that you have to we have to consider, you have to take into account the other bits.

So, in this way from this you can say that, you can do Pre-computation with the values of A_n and B_n and whenever A_n is equal to 1 B_n is equal to 0 you all you know that A is greater than B.

And so, you need not apply the remaining bits to the A comparator. Similarly, when A_n is equal to 0 and B_n is equal to 1 then also, you that b is greater than A. So, you need not really apply the remaining bits to this comparator, and you can save power consumption. So, this is the basic idea of this Pre-computation illustrated with the help of this comparator example.

So, here you can see the circuit that can implemented very easily. So, A_n and B_n are applied through a register R_1 and A_{n-1} to A_1 and B_{n-1} to B_1 these inputs are applied through a separate register with a enable signal E_1 .

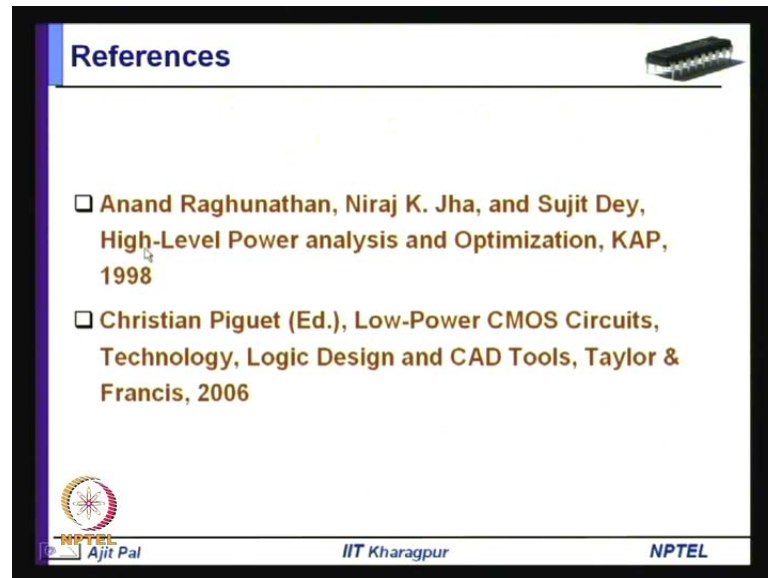
Now, this signal E_1 is generated, which is a function of A_n and B_n . So, this is enabled only when A_n and B_n are 1 are same; that means, when A_n and B_n are same 0 0 0 1 only than these bits A_{n-1} to A_1 and B_{n-1} to B_1 are to be applied to this comparator to find out. Whether A is greater than B or B is greater than A in all other situations; that means, when A_n is equal to 1 and B_n is equal to 0 or B_n is equal to 1, A_n is equal to 0 than you can prevent these bits to be applied to do to this n-bit binary comparator.

So, this is what is done with the help of this hardware simple hardware. So, a simple exclusive OR gate is used to generate the enable signals; that means, whenever; both of them are 0 0 or 1 1. I mean, than this 0 0 or 1 1 only than this enable signal is generated otherwise, this signal is 0 and output is not applied. And then, you take the final output from another register R_3 . So, here in this example pre-computation function is A_n exclusive of B_n , which can be realized by an or gate simple circuit.

So, in this case disabling of a subset of inputs take places. So, earlier we have seen in the earlier case, you are able to disable the entire set of inputs; that means, all the inputs are disabled. So, this is the case where the complete disabling of the inputs is taking place. But in the in the example, that we have illustrated comparator example, only a subset of the inputs are disabled from reaching the inputs. So, you can obviously, the saving will be much more in this particular case, but it may be little complex. You will require little complex hardware to identify this condition, and generate the activation hardware for this particular latch.





On the other hand, in this case although you are able to disable a subset of inputs, there will be significant saving in power dissipation. In this comparator power circuit, and the activation circuit, it is also very simple in this particular example.

(Refer Slide Time: 48:12)



References

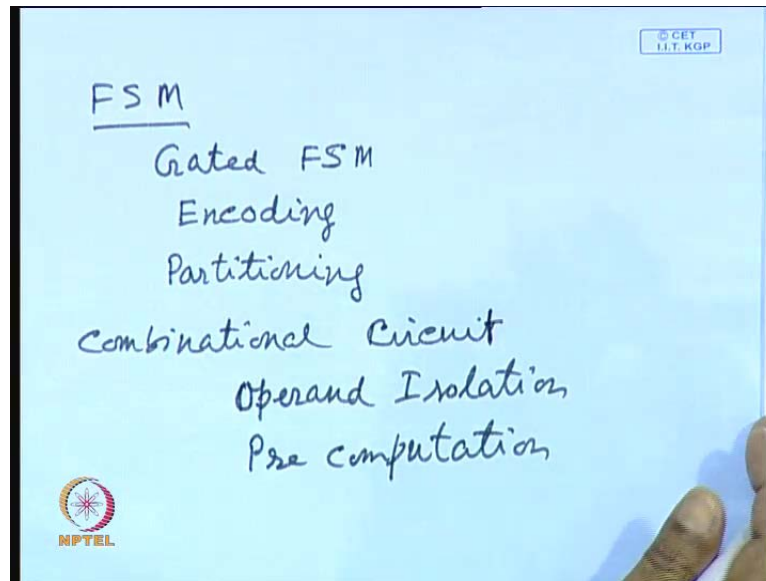
- ❑ Anand Raghunathan, Niraj K. Jha, and Sujit Dey, **High-Level Power analysis and Optimization**, KAP, 1998
- ❑ Christian Piguet (Ed.), **Low-Power CMOS Circuits, Technology, Logic Design and CAD Tools**, Taylor & Francis, 2006

  Ajit Pal  IIT Kharagpur  NPTEL

So, this example of Pre-computation has been shown, in this with this help of this comparator. And these topics I have taken from, these two references one by Anand Raghunathan, Niraj jha and Sujit Dey. I mean a book written by them High Level Power analysis and Optimization, by a published by Kluar Academic Publisher in the year 1988.

And sec and another reference is also a book edited by Christian piquet, and the book is low power comes circuit technology logic design, and CAD Tools published by Taylor and Francis in the year 2006. So, I can conclude this lecture by summarizing, what we have discussed today, first of all I have discussed about, three techniques for reducing switching activity in Finite State Machine.

(Refer Slide Time: 49:12)



The first example was in FSM, you can do you can use gated FSM. You can do? You can need not apply the clock all the time, you can identify conditions idle conditions, and by finding out idle conditions, you can get the clock and say switching power dissipation.

Second was you know Encoding you can do straight Encoding such that, the switching activity is reduced, and then we have used another technique, that is called you know that is essentially a technique. Where you do FSM Partitioning, and after that with the help of these three techniques, you can reduce power dissipation in finite state machines and in combination logic blocks also, you can reduce power dissipation by using two techniques, that are two techniques and these two techniques are number one is Operand Isolation, and second one was Pre-Computation. So in these two cases, when you are driving a large combinational circuit, you can inhibit the input to reach these combinational circuits and unnecessary transitions.

So, with this we let us conclude our lecture in the next lecture, we shall discuss about, various architectural techniques that can be used to reduce, the switch can be used to minimize the switch capacitance in a circuit. And also, we shall discuss of our different logic styles that, we can use to reduce power dissipation. Thank you.