

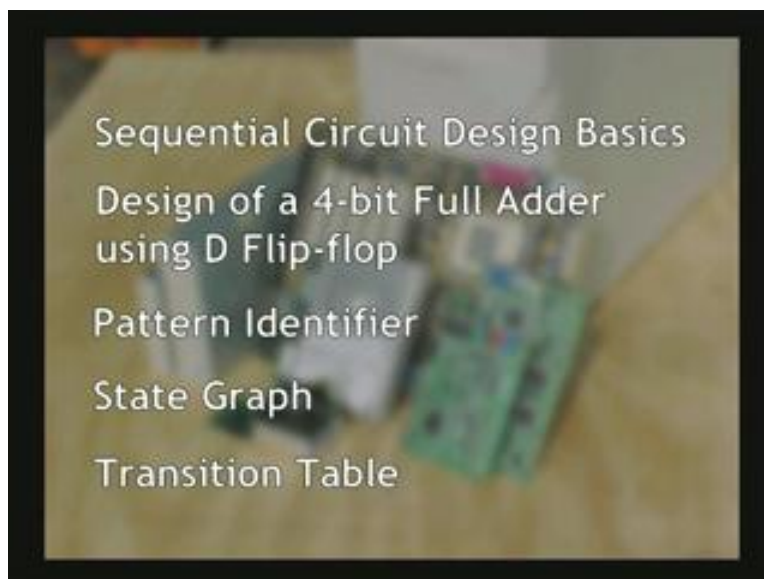
Digital VLSI System Design
Prof. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Lecture No – 7
Sequential Circuit Design

Previous Lecture
Sequential Circuits



Slide – Summary of contents covered in this lecture.

(Refer Slide Time: 01:55)



In the last lecture, we looked at the timings of the Flip-Flops. In the lecture before, we looked at the different types of Flip-Flops. In the last two lectures basically, we have been covering Flip-Flops. As we said earlier, Flip-Flops are essential for sequential circuits. But they alone do not constitute for sequential circuits because, there are exceptions like counters where only Flip-Flops are used. But in general, any digital system will have both combinational and sequential logic; sequential logic uses Flip-Flops. Even though they are total digital systems, whenever a sequential component like a Flip-Flop is involved, the circuit is called sequential circuit even though it will have a combinational logic in it.

Today, we will see the circuits which will use both combinational and sequential elements. As I said they are called sequential circuits but they have both combinational elements and sequential elements; combinational elements can be in terms of gates and multiplexers, PLA's, PLD's; and sequential circuit element will be Flip-Flops. To give an example of this, I will first take a very simple commonly used circuit; we are also familiar with this, a full adder. We have seen a full adder earlier in books, and in an earlier course. Even in this course, for the combinational logic we looked at full adders, implementation using multiplexers, full adder implementation using decoders and even program logic device.

These are 1 bit full adders in the sense, when we give bit a and bit b along with the carry bit from the previous position, we get a sum and carry; single bit combinational logic. If you want to have many bits to be added in the same time, we will have several of these working in parallel. That means, if we want a 4 bit addition, we need 4 full adders connected in cascade. On the other hand, if you have only 1 full adder and you want to add 2 numbers which are more than one bit; multi bit numbers, we go for technical serial adders, that means, the same full adder will be used repeatedly for adding different bits of a and b.

This is how it is going to look like, (Refer Slide Time: 04:40) the 1 bit full adder with 2 elements and sum. You know that 1 bit full adder requires in addition to this a and b. Let us called this a_i , b_i and s_i , this is what we have used in previous lectures for signals of full

adders. In addition we have c_{i-1} the carry for the previous date position and carry from the output for the current position. This completes the full adder but as I said I am going to use the same full adder. First I will add a bit say 4 bit number $a_0, a_1, a_2, a_3; b_0, b_1, b_2, b_3$. First I will add a_0 and b_0 ; then I will add a_1 and b_1 ; then a_2, b_2 ; then a_3, b_3 . While doing so, I should retain the carry from previous position and put it back here (Refer Slide Time: 05:55). This is exactly the concept of sequential circuit we have seen in the last two lectures. We talked about the sequential circuit requirement being a device for a circuit element which will store a value from the previous clock period and give it as input to the system during the next clock period. So the storage element is required which we will not see in a combinational full adder. That is why I want to take this example because all of us are familiar with full adders is a simple circuit. At the same time, how do you introduce a common sequential element in this full adder? What we have to do is, take this c_i , keep it in a Flip-Flop - D Flip-Flop and connect it back as the input; so the carry does not come from anywhere. It comes from within the circuit (Refer Slide Time: 07:00).

As each bit is placed here, first of course, we will have to reset the Flip-Flop so that the Flip-Flop Q will be 0. There is no c_{i-1} , so first 2 bits come, get added; there may or may not be a carry that gets stored here (Refer Slide Time: 07:25) and that becomes c_{i-1} second bit a_1, a_2 come a_1 and b_1 come, second bits at the time c_{i-1} will be there. In this case 1 or 0 depending on the previous bits a_0 or b_0 , that gets added generating a new s_i and new c_i that gets stored and so forth, that you can go on. The hardware does not restrict the number of bits to be added.

Whereas in the combinational of full adder if you want add a 4 bit number we need four 1 bit full adders. If you want eight bit adders, 8 bit addition, you need 8, 1 bit full adders. So will have to tell in advance how many things you need. Here I can use this repeatedly as long as I want; of course there are some practical considerations: where is the a_i going to come from; where is b_i is going to come from; where is s_i going to go to; all those questions we have to answer. They have a register to store all these things; we will have the register here (Refer Slide Time: 08:32). I am freely using a lot of these components which are not exactly defined earlier in this course. But as I said this is the second course

and all of you are familiar with these things. So there is no point in trying to be very legalistic about it whatever has not been defined earlier should not be used that concept then we can never proceed.

I know that all of you know registers and shift registers and previous course in digital circuit. We are going to use a shift register here where in originally I am going to put my entire a - whatever is the number of bits; b - whatever is the number of bits. I am going to put this result also in shift register wherein as s_i gets stored and shifted along the way. I will say at the end of its operation, I will take s from here. If it provides sufficient number of bits here, the loss carry also will be taken in the circuit, so I do not have to provide an extra register for a carry. For example, we have a 2, 4 bit numbers I have a 5 bit sum - maximum of 5 bits. That means, for 4 bit shift registers of inputs, I will provide a 5 bits shift register for the output. The last bit also will get added here; the last carry will be added here (Refer Slide Time: 10:10). We do not have to worry about carry being separately saved and used.

All of them have to be clocked properly; we need a clock any way for the Flip-Flop. When does the Flip-Flop know it has to release the previous value or when does it know it has stored the new value c_i and rewrite on the old value? That will be decided by the same clock which can be used for shifting the shift registers. I will say that there will be a clock and then the same clock can also be used for this output shift (Refer Slide Time: 10:45). This is one clock, this shifts bit a into the full adder, bit b into the full adder shifts s_i into the sum register, stores the carry value. We use the c_{i-1} for the next bit addition. This is a sequential circuit design.

In the concept of sequential circuit, I just brought this up because, it is not today full adders are so inexpensive. You know a full adder consists of 2 exclusive OR gates for the sum and a 3, 2 input AND gates and the 3 input OR gates for the carry. It is not a huge hardware and you can multiply it any number of times in order to carry out large addition, plus the facts that these also delay the addition process. You are talking of giga hertz or mega hertz several 100 megahertz operation; that means, the clock speed is very

high. If I have this type of full adder number of bits, the total time takes for me to complete the addition is going to be that many clock cycles.

If there are n bit addition I need n plus 1 really, because the last 1 carry out may be there which is just to be added and put here. There are n bit addition I need n plus 1 clock cycle to be complete this operation (Refer Slide Time: 12:45). It may be too slow for our modern technology, today's technology requirements are: can you trade the speed with the hardware? If I can give you more hardware called hardware accelerator, people call it and you might have heard of this term frequently used today in technology. Hardware accelerator means can I provide hardware and speed of the process? So, trading the hardware for speed; can I get more speed if I give you more hardware? But on the other hand, you want to reduce the hardware for real estate, the size of the chip.

There is conflicting condition; all this we will have to discuss as we go along in this course. There are power requirements, speed requirements, space requirements; all those things will be taken into account. Each of this cannot be analyzed completely like that. Each is a separate concept available for use; if you want to use it, use it and if you do not want to use it, do not. I took the examples simply for two reasons: one is full adder it is a very familiar example to all of us. Second thing is, we talked about sequential circuits and we talked only about Flip-Flops; but I gave you the model of Mealy circuit and Moore circuit yesterday. A Mealy or a Moore circuit will have inputs, state variable, output and state variable.

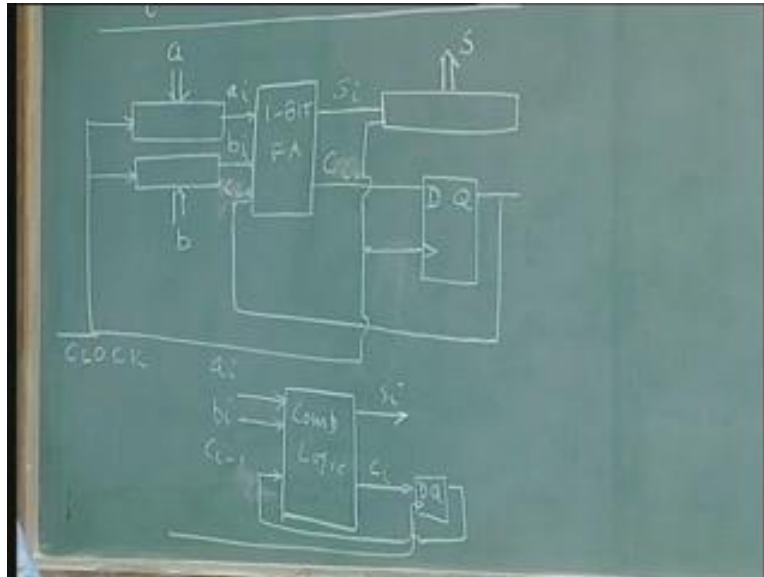
This concept I want to explain and better way to do it than having a very simple circuit, explain it. Now if you take our Mealy model in the last lecture, we had these external inputs. In this case, what are the external inputs a_i and b_i ? I substitute variable input c_{i-1} . Output is c_i s_i and the state variable output is c_i . There is a one state variable input which is again delayed a Flip-Flop. You see this concept of Flip-Flop output, the state variable input resulting in combinational output and state variable outputs. The state variable output used as the state variable input again for next clock period along with a new setup inputs a_i b_i . This combination we discussed in the last lecture.

I thought I could give this example; it will fit into this; this is the combinational logics, this is comb logic. Here in this case 1 bit full adder is a comb logic all of you know that 1 bit full adder is a comb logic with as I said two exclusive OR gates and 3, 2 input AND gates are 1. 3 input OR gate and then the Flip-Flop; the combinational inputs are a_i b_i , combinational output is s_i . State input variable is c_{i-1} , state variable output is c_i . To give this one exact mapping using a very simple concept, I brought this one. Of course, I am not going to derive this state map. I can also prove how to do it by deriving what is known as state graph; all of you know what a state graph is from a previous course on digital systems. We have a state graph drawn for this circuit and design the circuit starting from the conceptual level.

I did not develop the architecture; I just drew this architecture and assumed the circuit is going to be full addition; then explained how it acts as a bit serial adder; and compared it to the standard architecture and proved it is the same. But it is not the way you do it in a circuit, unknown to you. The system is unknown; the problem specification is new to you and do not simply draw the circuit configuration. You have to develop the circuit configuration for that you start from the state graph and you implement the circuit which will also have the same model. Since it is a full adder I did not do that.

I did not want to start with the state machine concept with states a and b as inputs, c as inputs and s_i and c_i output and all that. I do want it because it is an overkill of this problem; it is a very simple problem all of us are familiar with. We will take a slightly more complex example to prove the point. We should also know how to develop the design from a specification, when the circuit is not as obvious as this circuit. When circuit is as obvious as this, you should know how to develop that circuit from the start. We have to understand the problem specification; from the problem specification to the state graph; from state graph go to state table; from state table you get the hardware and that hardware also will have the same architecture (Refer Slide Time: 18:15); same type of connectivity. A modular architecture will be identical; we will prove that with a slightly complex example.

(Refer Slide Time: 18:35)



Take a slightly more involved complex example for this purpose. Another thing which is normally done in digital circuit systems is a compared pattern. This pattern comparison is very useful in many places. Even the computer password checking is a sort of a pattern comparison. You type a password; of course the password is a alpha numeric. Assume for a minute it is only a binary password 0's or 1's pattern. The computer verifies your password, the registered password and then if it matches it will give you access to the machine; otherwise, it will deny the access. Of course it can be done in alpha numeric sense by having each of this alpha numeric characters translate into binary.

Another place is you can have a combinational logic – combinational lock. Suppose I have a key, you know the number locks we are familiar with. I can have a similar number lock digitally. Suppose I have a lock in which I know the pattern. Only when I put the proper pattern and press a switch, a relay will be operated and my door will open. A latch will operate, door will open; otherwise, the door will not open. So this is another place where you can use pattern matching. Sometimes when you are receiving data, you match the data with a given pattern; receive whether there is error in the data. So there are several places, where pattern matching is required in digital circuits. I will take pattern matching as an example for design of a sequential circuit that also will give you a methodology of the design starting with the specification of the problem through this

state graph to the hardware. I will say pattern identifier, call it anything you like; it compares; if a particular pattern has occurred it has to give an output. In the absence of an output, the given pattern not occurring, it should not give an output, the output should be 0. Basically the circuit should consist of the single input, I am making it simple for you, single output I call this x, I call this z and this is a clock (Refer Slide Time: 21:20). With every clock pulse or clock edge because as I said in the last lecture on Flip-Flops, this arrow head points to a positive edge triggered Flip-Flop. For every positive edge of the clock, 1 bit of the x will be the input to the circuit and as inputs occurred, there may be some mechanism inside to get those inputs stored and then, when a particular pattern occurs that pattern will be matched and then output will be set.

For example, I want to detect a pattern of 0 1 1 1 or 0 1 1 0 pattern to be identified bit pattern to be identified. 4 bit pattern 0 1 1 0, as the bits keep coming whenever this pattern occurs, output z will go 1 along with the last 0 of that pattern I will give an example let us say x is a serial in time it occurs like this 0 1 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 1 etc., that means each clock pulse one input bit occurs at the input and the output should be, initially we assume the output was 0, this is a reset condition. Initially there is a no output as the bits keep coming there will be no output here the output will be 0 0 0 0 0 0 (Refer Slide Time: 24:17). This is one of the patterns required so 1 should occur; that is what I mean by the last 0. Remember this time it is going this way that means first 0 was received then a 1, then a 1 then a 0; fourth bit corresponds to the output bit along with the fourth bit 0 I should get an output of 1. Then of course it will go 0 0 0 0 0 0 (Refer Slide Time: 25:10); this again will give me 1. I should be make a change in this pattern to give you an interesting concept. I will make it 1 1 0 (Refer Slide Time: 25:21) like this. The fourth bit of the 0, 1 was cut then the next again is a 1. As a designer you should ask this question before you start with design.

If this 0 has already occurred and this 1 is occurring next, should I count this 0 also has a new pattern. There are two ways of doing it, if I say after every output is generated you start afresh, you reset your system and start afresh. If you want to do that way then I do not count this 0. Then I will start with a 1 as a next bit. But sometimes, you may have to do the overlapping. That means, even it is 0 as it is counted in the previous pattern, it

should still be counted in the next pattern. If that is a specification then I should count this 0. Let me assume that model wherein I am going to count the 0 also into my next pattern; this is called overlapping. Even overlapping bit pattern should be identified. So my problem has to be completely and clearly defined; overlapping patterns should also be identified. If you say that then you are very sure of what you should do. If you are not given this then you are not sure whether I should count the 0 as a new pattern or consider this closed and start again here.

Specification is very important. If the specification is not clearly mentioned, your circuit performance has ambiguity. Later on when some input condition occurs and output is not satisfactory, they will blame the designer. So it is the designers fault if the specifications are not completely received, you should ask for the specification. Of course, the person who asks this design should give all these specifications; but in case he does not give, it is your duty as a designer to ask for it and get it.

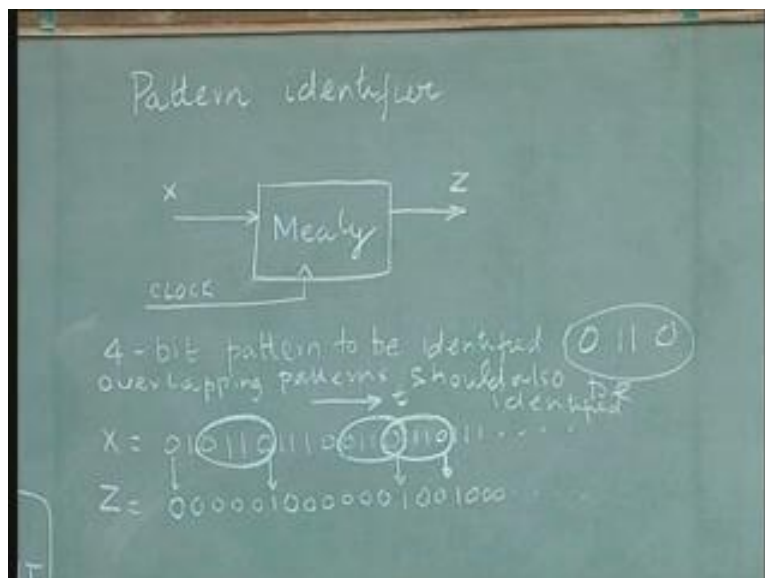
If you also assume something vague and your vagueness is different from the vagueness of the specifier, the specification given has vagueness built into the specification and your design has a vagueness built into a design. But these two vagueness do not match, then comes all the problems. Now 0 1 1 0, I am going to call these as a pattern and give 1 (Refer Slide Time: 28:20). This is what I want to say. So 1 input, 1 output, 1 clock of course, visible 1 clock, inputs, outputs; whenever 0 1 1 0 occurs an output z should be 1 and overlapping also should be occurring. This is the problem statement.

I forgot one small thing about my previous example with the adder. I said all about the adder being a combinational logic and a sequential logic together. We also defined two different types of sequential circuits in the last lecture; one is the Mealy circuit; the other is a Moore circuit. The Mealy circuit output depends on the present states and the inputs. Moore circuit outputs depend only on the state variables. The external inputs do not come in to the picture for the output definition. Output will be decided by the states will be alone which are of course define by the input variables outputs or state variable outputs or functions of input variables and present state variable that is a next state variable that define the output in the Moore circuit.

In the Mealy circuit, in addition to these outputs they also have outputs which are directly related to the present state and the present state of inputs. And according to that definition, this comes as a Mealy circuit. In any circuit there is a possibility of designing as a Mealy circuit or a Moore circuit. Mealy circuit is slightly easier to implement hardware wise it is a slightly less complex. Moore circuit is one which is slightly more involved in the hardware, of course, it is not complex in the sense it has little more hardware but, later on we will have occasions to again come back to these Mealy and Moore. Why I am emphasizing at this point here is later in our series, we are going to talk of situations where we have to definitely go for Moore circuits for some reasons. This is why I am always finding out whether it is a Mealy or Moore circuit. Since I did not design this, I took the circuit (Refer Slide Time: 30:53) and tried to explain it; I did not ask this question in the beginning. Later on I realized it is a Mealy circuit.

Now in these case, since I am going to design the circuit, before I start with the design, along with the patterns overlapping etc and all the specifications, one more question I should ask the person who gives me the problem is, “do you want it as a Mealy circuit or Moore circuit?”. Again as I said, Mealy is slightly simpler than Moore, the whole idea is to make this first example simple and I am going to call it a Mealy. I am making your life and my life easier by doing this concept.

(Refer Slide Time: 31:44)



Now, with this example and this specification and being a Mealy, we will try to put the state graph that is the first requirement; start with the state graph. The state graph will have several states, again I do not have to explain this state graph to you. I always keep insisting this is not your first course in digital; this is a second course on digital circuits; because of the flavor we are adding in the courses through VLSI circuits plus the fact I am directly going to some design rather than going through all these basic things. Wherever I am assuming things which you do not know, it is your responsibility to catch up by reading a book or practice some exercises, browse through your class notes or look at a video course or whatever; it is up to you to get to speed about these things. Because, if you keep missing all these things, later on, you will find the whole concept so fuzzy; then we will not be able to appreciate the contents of this course. I am not going to explain to you what a state graph is; I am going to directly start doing it.

Assume the first state called s_0 which is a reset state as I said we have to start at the reset state with the output 0. I will make a note here - s_0 is reset state. Then each state the input can be one array 0 that is all. There is only one input; so input is 1 or input is 0. If the input is 1, it does not even fit into my pattern, what is my pattern I am trying to identify? It is 0110. Remember this is the pattern I am trying to identify. The very first bit is 1; it does not fit into the pattern; I do not have to remember this one. This can still remain in reset state.

As long as I am concerned, the 1 received is equal to not receiving anything; I can continue to be in reset state. The 1 occurs (Refer Slide Time: 34:00) I am here - same state not producing an output. First you write the input and next the output. When you say 1 0 that means x slash z, output is shown with a slash in between (Refer Slide Time: 34:26). On the other hand, if it is a 0 it is possible that it is beginning of a pattern. It may not materialize as a pattern but, you should not lose it; 0 has been captured and kept, remember it. If the 0 occurs you still have no output (Refer Slide Time: 35:00). Output will come only with along the fourth bit; output is still 0 but I need to go to a new state which will remember this. Let me call this s_1 ; s_1 is a state will remember 0. I have exhausted, in s_0 what happens if the 0 is input, what happens when 1 is input?

Likewise I do from each state s_1 , in s_1 state if 0 occurs I do not have to store it again. Already I have stored in s_1 is a state where 0 is already been memorized; 0 is recorded when I am already in s_1 state. There is no point in the recording the second 0 because my pattern requires only one 0. The latest 0 can be taken as the 0 of the pattern and discard the previous 0s. That means the 0 occurs, I continue in to be in the state and the 1 occurs then you are in business. Because 01 is a likelihood of a pattern being formed. When 1 occurs I will go to s_2 , output is 0 because I only receive two bit so far. Number output being 1 but I should remember this 1 0 as a distinct pattern which is imagine so far which may be useful to identify my final pattern. So s_2 is state which remembers 0 1. Again in s_2 what happens? In s_2 if 0 comes that means it will be 0 1 0, no point. If a 0 occurs in s_2 I go back here and remember the last 0 (Refer Slide Time: 36:55) which is something like I received 0, 1, I am expecting the next one but 1 has not occurred a 0 has occurred. What do I do? I discard my 0 1 earlier and I now start this with a 0 as a beginning of a pattern.

So when 0 occurs I go back to s_1 the output of 0 and when 1 occurs then it could be 011 could be the 3 bits of the final pattern we are looking at. So if 1 occurs I go to state s_3 which will remember 011. Pattern 011 has occurred in a state 3 means, a pattern 011 has been identified. All it needs is another 0 to complete the operation. If it is not of course we will not get the output as 1. In state s_3 when 0 occurs what happens? You are completing the process; we will output a 1 but also with remember it is a overlapping pattern.

This 0 is last 0 of the first pattern (Refer Slide Time: 38:20); it also will serve as the first 0 of the next pattern. So this 0 not only should produce an output of 1, should also be remembered as the first 0 of the next pattern. What is the state which remembers the first 0? s_1 remembers the first 0; so when 0 occurs in s_3 , you go to s_1 , so the 0 will be remembered as the fresh 0 of the next pattern and at the same time I should get an output of 1 not a 0. This is the first time, the only time the output will come; 0 will occur, the output will become 1. On the other hand, when you are in s_3 , the 1 occurs already in 0 11; the 1 occurs has no meaning because I cannot use any of these things; because 0111 has no place in our pattern to the extent; that means, it as good as going to the reset state; 1

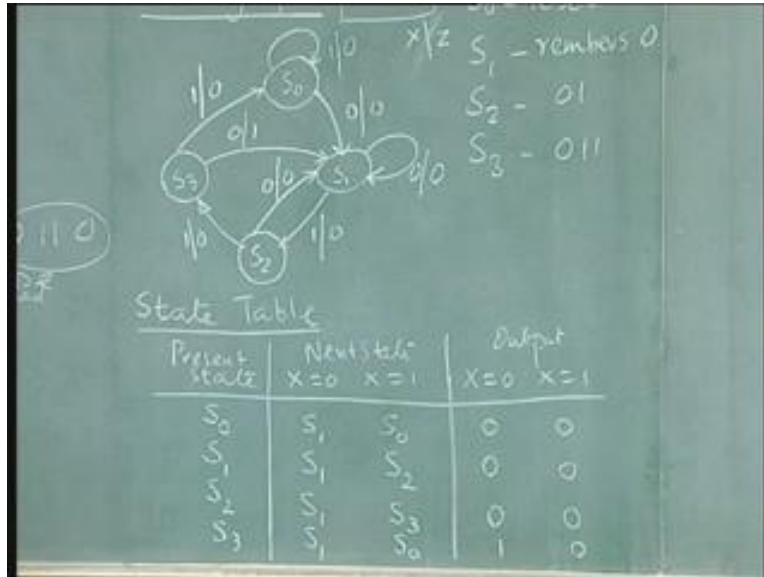
occurs in s_3 is a wasted effort. Something like the whole thing was almost there and we missed it.

I made the circuit specifically simple, so that you can understand the concept well. We can practice more such problems; any standard text books and digital design will give you problems for this type of exercises. You can go and do more complex ones with more number of states and more number of inputs and outputs, to practice the design of the sequential circuit. What we are having is a sequential circuit in which there are 4 states s_0, s_1, s_2, s_3 and this is how you write the state table (Refer Slide Time: 40:18).

What is the present state in which the circuit is? What will happen to the circuit in next state, if the input is 0 and if the input is 1? What will be the output in each case? That can be written now on this state graph. The present state is s_0 , 0 occurs we go to s_1 . If 1 occurs we remain in s_0 ; either case the output is 0. Whereas state s_1 , if input is 0, we remain in s_1 ; input is 1, you go to s_2 . Both cases, output is 0. s_1 to 0, s_2, s_1, s_1 . You are in state s_2 input occurs 0. Go back to s_1 and if it is 1, go to s_3 again. Both cases the output is 0. Finally when the circuit is in s_3 state input 0 occurs you go to s_1 ; 1 occurs, circuit goes to s_0 . In the first case the output is a 1, second case output is 0.

This is the state table; again a concept well known to you from previous course on digital systems. I thought it is better to do it; this is the graph; this is the table. From this table you can get the circuit. To do that you have to assign variables for each of these states; you can arbitrarily assign 0 0, 0 1, 1 0, 1 1 as state variables.

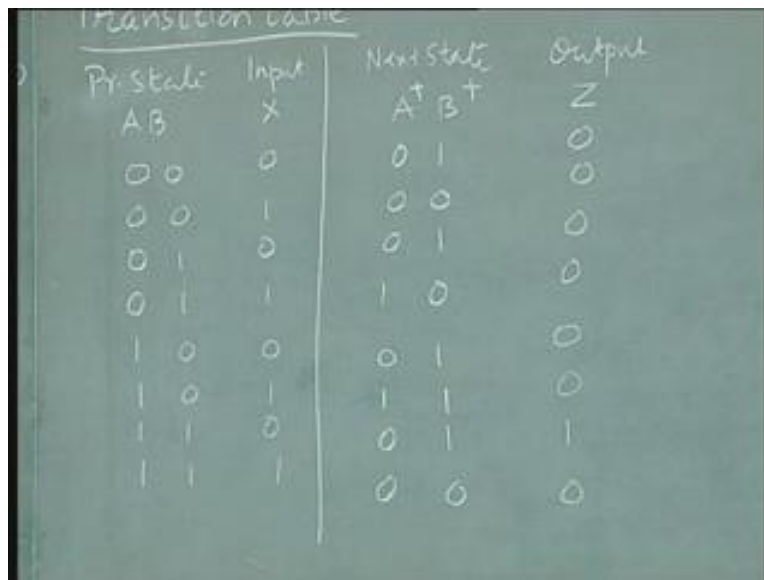
(Refer the Slide Time: 42:47)



We can translate this into a table called transition table. The difference between the state table and transition table, if you do not know already, is the fact that in state table, you refer to the states as symbols s_0, s_1, s_2 or a, b, c, p, q, r, x, y, z. In a transition table, you give the actual values of 0 0 1 1 - binary values for these states. So these are the binary values, let us call these binary variables A and B, present state, input, next state output, present state I will call A B, input is X, next state is A B again. In order to differentiate between the next state and present state, it is usual to put a plus on this (A plus B plus). Output is z. Now the present state is 00 input is 00 0 input is (Refer Slide Time: 44:18) 1, 01 0, 01 1, 10 0, 10 1, 11 0, 11 1.

The next states are s_1 , (Refer Slide Time: 44:40) 01, 00, 01, 10, 01, 11, 01, 00. These are the next states, the corresponding outputs are (Refer Slide Time: 45:00) 0, 0,0,0,0,0,1,0. This is a transition table from which you can draw the karnaugh maps for each of these variables A and B and the output z and then implement in terms of a circuit.

(Refer Slide Time: 45:10)



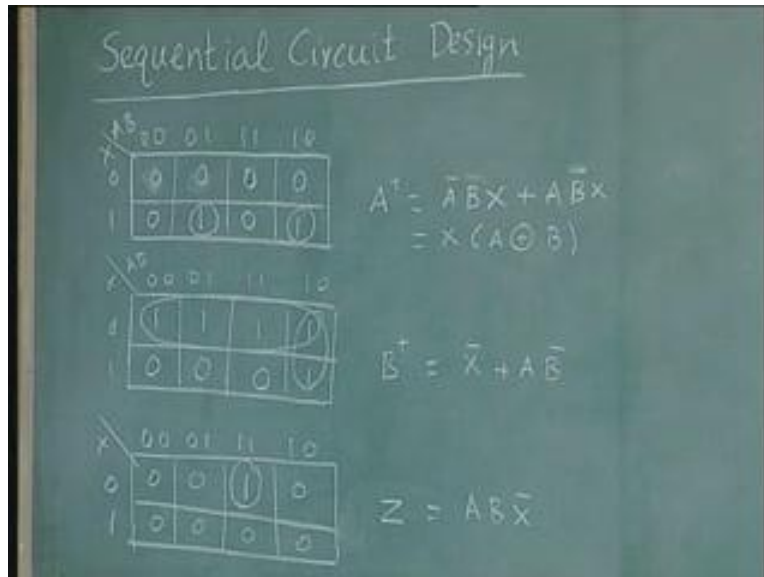
A handwritten transition table on a chalkboard. The table is titled 'Transition Table' and has four columns: 'Pr. State', 'Input', 'Next state', and 'Output'. The 'Pr. State' column has two sub-columns 'A' and 'B'. The 'Next state' column has two sub-columns 'A+' and 'B+'. The 'Output' column has one sub-column 'Z'. The table contains 8 rows of data.

| Pr. State | | Input | Next state | | Output |
|-----------|---|-------|----------------|----------------|--------|
| A | B | X | A ⁺ | B ⁺ | Z |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |

After drawing the transition table we need to realise the circuit and the best way to do it is to use karnaugh maps. If you are going to go for gate implementation. This being the first example, go for gate implementation of combinational logics, which means I will go for karnaugh map. Minimize the karnaugh map to get the minimum expressions for the values, next state variables A plus, B plus and the output z. You do not have spend lot of time on this because, you know how to do this. We have the transition table here (Refer Slide Time: 46:01) and all you do is to map this transition table into 3 karnaugh maps.

One for A-plus, second for B-plus, third for z. That has been done here and I got the expression. These are the two terms for which the output is 1, the rest are 0s (Refer Slide Time: 46:22). A plus is this, this is $A \bar{B} X$, this is $A B \bar{X}$; so these **X OR X and A OR A exclusive OR B**. This is the much simpler B plus the whole thing is 1 which is nothing but X bar and this is $A B \bar{X}$ and z there is only one term. For z, there is only one term is 1, all of these are 0.

(Refer the Slide Time: 46:03)



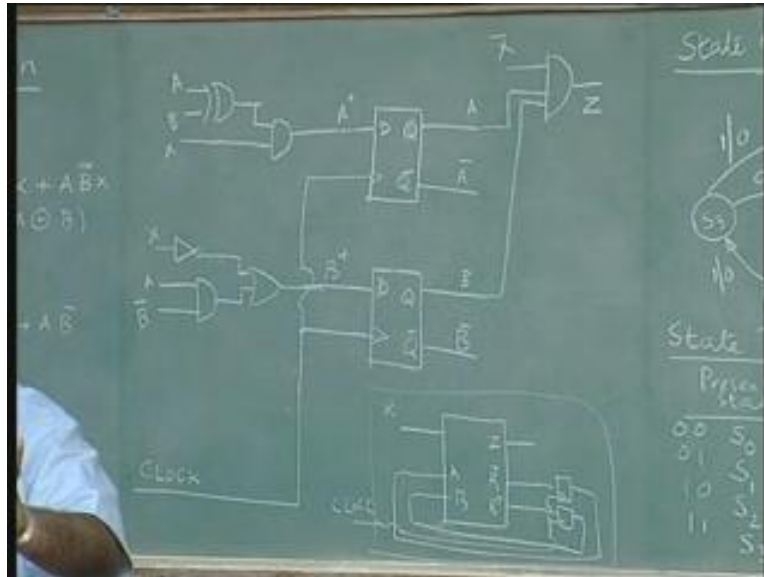
This happens to be $A B X \bar{}$; with this I can draw the gate structure for this there are two Flip-Flops. One of them is going to be called A, the other is going to be called B. What I mean by $A \bar{}$ is, the input to the Flip-Flop A that is A plus; the output is A here. Similarly, input is A Flip-Flop B is called B-plus; output of this is B. What is A plus? It is X, there A exclusive OR B ANDed with X; B is X bar OR A and B bar, B bar is here.

You do not need inverter of that; put them in AND gate, combine them two in an OR gate; you get B-plus. These are Flip-Flops, you need to clock them and output z is $A B X \bar{}$, A is here, B is here, X is here, X has to be put through an inverter (Refer Slide Time: 49:02). We already have an inverter here if you want to use it directly $X \bar{}$ this is z; this $X \bar{}$ comes from here. I am not showing it as a separate inverter, if you want you can show x as an inverter. But that inverter can be saved because we already used an inverter for X. This is the sequential circuit implementation of the original pattern generated we started with, we want to detect a pattern occurring as 0 1 1 0.

If it is overlapping, we need extra including overlapping pattern should detect and whenever the pattern is detected along with the last bit 0 we need to get a 1. This whole thing is now the Mealy circuit, inputs are A B and X. A B are the state variables, X is input variable which is going here (Refer Slide Time: 50:14). A and B are the state

variables we combine $X A B$ to get A -plus B -plus z . It is a typical Mealy circuit because output combines the present state variables and the present input. And the next state variables are obtained here using Flip-Flops which are fed back as a clock. This clock will be going here and here, that is all right (Refer Slide Time: 51:00).

(Refer Slide Time: 51:34)



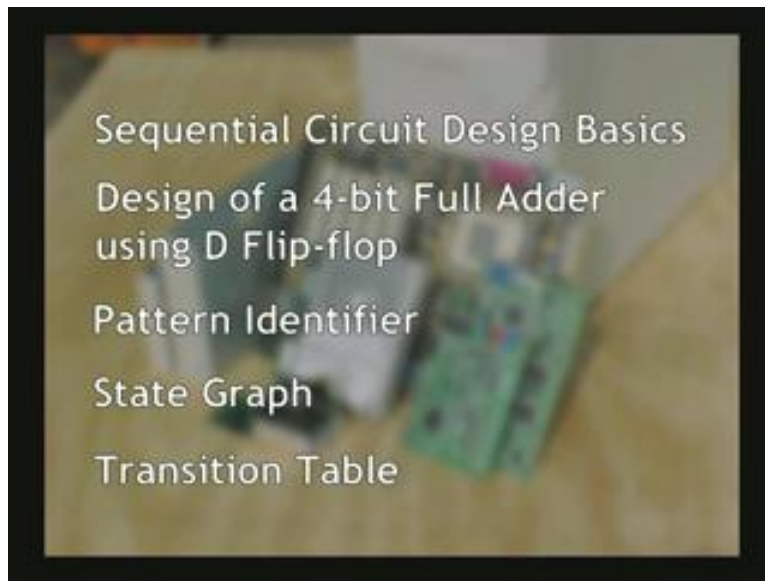
This shows exactly the pattern of this architecture. The configuration of this circuit is same as the configuration of the circuit which we started with (Refer Slide Time: 51:15). Whatever the design problems, a simple serial adder or a pattern generated or any other complex problem, the whole thing will go through the design procedure. Remember to have the specifications complete in detail, very clearly written. No ambiguity of the specifications and then from the specification we draw the state graph. From the state graph you go the state table; from the state table to the transition table; transition table you map into hardware the options. This is only a simple gate oriented option.

The combinational logic can be implemented using multiplexers. We know the programmable logic devices. There are so many other ways you make the choice and implement it. There are some other techniques by which, without drawing the state graph by equivalent method, not the same procedure, we can do that. Conceptually the procedure is the same. There are certain ways in which you can do this without having to

do that; we will see that later on. But right now this procedure is complete. We will see more examples in future lectures.

Summary of lecture 7

(Refer Slide Time: 52:54)



MSI Implementation of Sequential Circuits:

(Refer Slide Time: 53:00)

