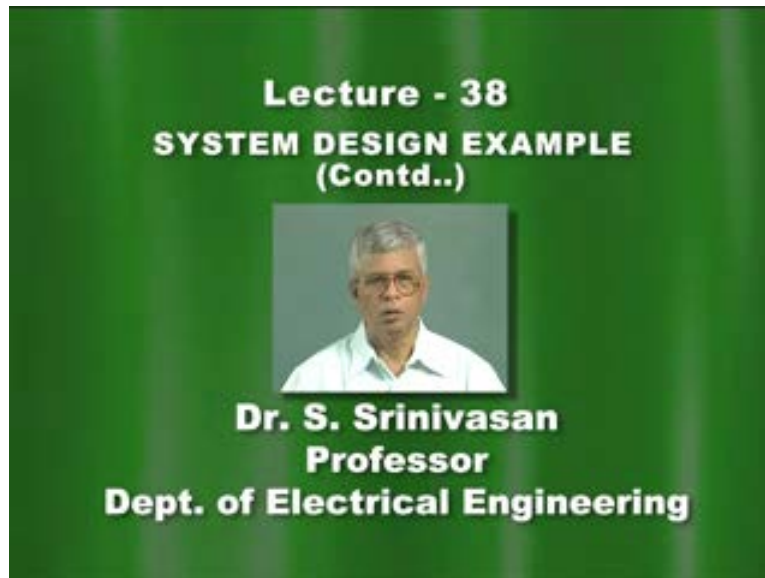
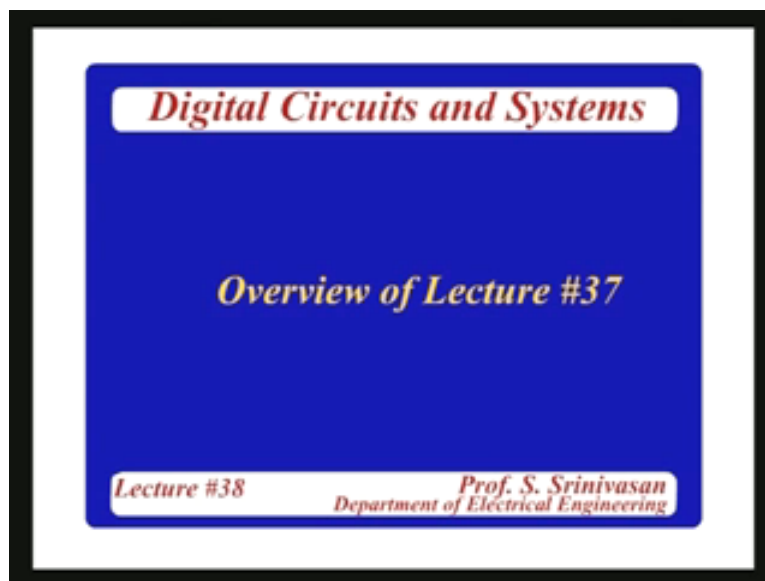


Digital Circuits and Systems
Prof. S. Srinivasan
Department of Electronics and Communication Engineering
Indian Institute of Technology Madras
Lecture - 38
System Design Example (Contd..)

(Refer Slide Time: 00:01:05)



(Refer Slide Time: 00:01:09)



(Refer Slide Time: 00:01:25)

Digital Circuits and Systems

- * *System Design Example*
- * *Traffic Light Controller*

Lecture #38 *Prof. S. Srinivasan*
Department of Electrical Engineering

(Refer Slide Time: 00:01:33)

Digital Circuits and Systems

Contents of Lecture #38

Lecture #38 *Prof. S. Srinivasan*
Department of Electrical Engineering

(Refer Slide Time: 00:01:48)



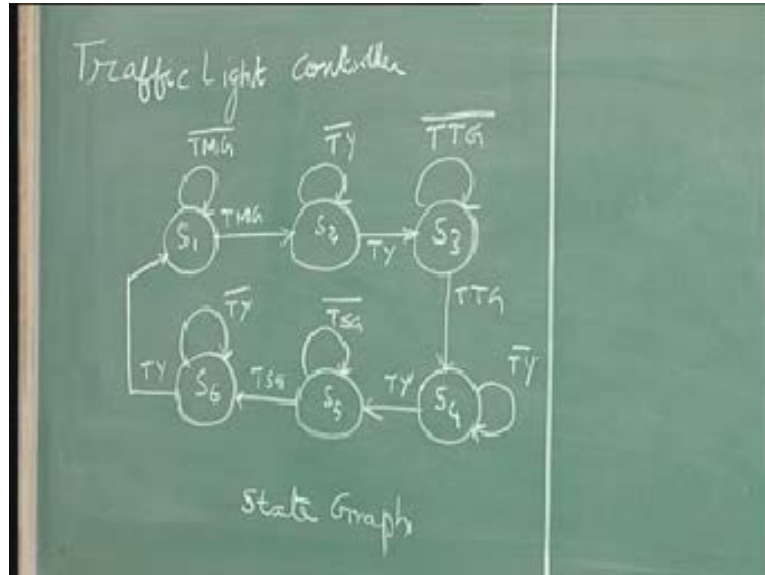
So we were discussing the traffic light controller design, the design of a traffic light controller as an example of a system design. We have earlier seen how to design and implement combinational logic and sequential logic and even circuits that contain both combinational and sequential logic using building blocks such as gates, then multiplexers and decoders, PROM Programmable Logic Arrays, Programmable Logic Devices and likewise in sequential circuits flip-flops, registers, shift registers, [c...2:35] etc. But when you have a digital system to be designed you need to go through a design process starting from specifications to the implementations.

As I have repeatedly said in this course, understanding these specifications clearly and coming up with a requirement in terms of signals and hardware it is the most important step, from then on it is a question of procedure, systematic methods are available. If you want simplification there are steps available and if you do not want simplification there are procedures available depending on the type of hardware you want to use or procedures available.

So the implementation, hardware, putting it together, testing, all those are important, I will not say anything is less important unless the system works there is no use, the design has no meaning unless you can put in the field and test it. But from a designer's point of view, from a design engineer's point of view it is the question of understanding the design specifications and coming up with a blue print of what to do, that is the state graph, the type of examples you have seen. There are other techniques. This one is called state machines, algorithmic state machines, charts and all those.

So with that in mind we discuss the requirements, specifications and requirements of a traffic light controller, **I will not repeat those things we have seen in the last lecture.** Finally we ended up with this state graph you remember.

(Refer Slide Time: 4:24)



There are 6 states, state in which the main light is green, side light and turn light are red then it goes to a main light the yellow phase then the turn light green, main light continues to be green from left to right, from here (Refer Slide Time: 4:50) we go to the phase where the turn signal turn light is yellow and then we go the phase where the side light is green in this case we halt the flow of the main road from left to right as well as right to left and then we have the yellow phase and after the yellow phase it goes back to the first state.

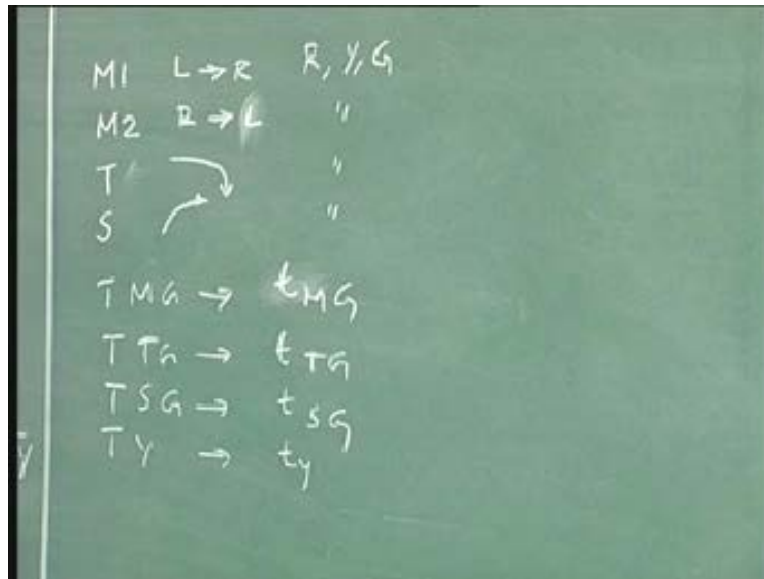
So we said that there are 4 lights M1 M2 these are the two main street lights, this is for left to right traffic, this is for right to left traffic (Refer Slide Time: 5:30), turn light side light, turning is for this turning and side light is for this turning and each one of them can be a red, yellow and green and all those conditions under which this happens we have already discussed in detail based on which we arrive at the state graph. Now the question is to implement this state graph. So we discussed the inputs and outputs. I forgot to mention one important input. What are the inputs and what are the outputs available for this system?

What are the inputs available in the system? What are the outputs expected in this system? The outputs are the lights, the signal which activate the light, of course the digital signal is 0 to 5 volts, 0 volts or 5 volts signal that you get out of the digital gate may be two week in terms of current to drive a light a huge light in the traffic light controller, the traffic light junction. But you can always use drivers. So we are not worried about all those hardware features where how this signal from the traffic light controller IC or a digital circuit is converted into a signal which is strong enough to activate the lights, switch on and switch off so you need to, if required amplify it or you need to boost it either in voltage level or current level depending on the type of the lights you are given. **We will not talk about that point.**

So, when you see here these signals red, green, yellow are all the outputs of a digital circuit that is one thing, of course that is understood. What I was referring to, I said, I forgot to mention something is, when you enter a state you have to watch the period. We discussed the periods. we

said T stands for the time, MG stands for the main green so there are four time intervals of importance in this TMG where duration for which main is green, t main green, TTG is the duration for which turn signal is green, TSG is the duration for which side light is green, this, this and this (Refer Slide Time: 8:19) and then TY - the duration for which the light is yellow, yellow light, lights are yellow.

(Refer Slide Time: 8:29)



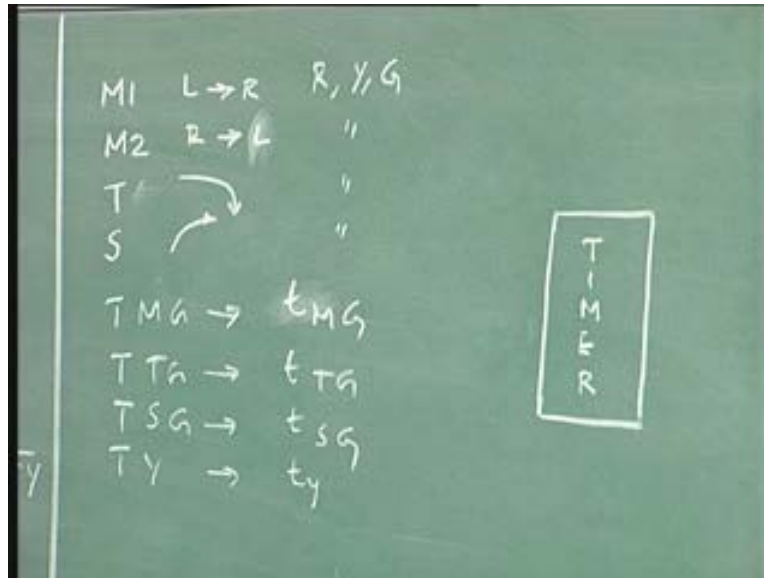
So yellow lights in all the cases, we did not distinguish between yellow for the main and yellow for the side, just to make the problem simple. Of course there are lots of features you can introduce as I said. You can have different timings, you can have pedestrian switches and all those things, and we can have 4-way traffic, 6-way traffic, there are four ways in the sense it is a 2-way street on both sides and then on both sides there can be a left turn and right turn and all those things, make it as practical and interesting as possible.

What I meant was, when you say you are monitoring these timings there must be hardware in the system then hardware has to be monitored for the duration of the time, when the time has elapsed the state has to change. For example, when the circuit is in this state this will remain in this state as long as TMG is not completed, the duration TMG is not completed. When TMG is completed it goes to this state. from here (Refer Slide Time: 9:32) it stays in this state as long as TY is not completed, during TY it is here and at the end of TY it goes here.

So there is a provision for monitoring the signal. These are the inputs to the system; TMG TTG TSG TY the time intervals for which different lights are defined to be green as well as yellow are the inputs to the system. How are you going to get those time? Who is going to give you those timings? So there must be a hardware timer, **we will not go into the details, you know how to do this**. I will tell you how to do this in a simple way. So I will call this as a simple timer. So you can define the timed intervals, like for example, as I said TMG can be 2 minutes or 1 minute may be, this can be 30 seconds, this can be 30 seconds or this can be 20 seconds, this can be 30

seconds or whatever depending on the traffic, may be 30 and 20 and then this may be again another 20.

(Refer Slide Time: 10:36)



How are you going to measure those? You have a timer which is nothing but a counter, you know how to get time intervals from a counter. The number flip-flops have to be decided. I have to give a clock to this, the clock is a system clock, it need not be related to any of these, need not be any of these timings, it can be an independent timing. For example, I can have a clock which is much faster than any of these periods and then I have to count a definite number of these clock periods in order to get each of these signals, is it not.

For example, 2000 counts may be this. That is an example. 1000 counts, 2000 counts for this, 500 counts for this, 400 counts for this, (Refer Slide Time: 11:28) 400 counts for this, something like that you can define, it depends on the duration. It depends on the duration you want for these different lights to be green or yellow and the period of the clock signals you are feeding into the systems.

The same clock will be the clock you will be using in your flip-flops then only there will be synchronism between the state changes and the timer change. But then there is a counter which keeps running all the time with the clock and I have to know which time corresponds to which. Fortunately these are non-overlapping intervals. I want only one time at a particular operation in the sequence. If I am interested in monitoring TMG I don't have to worry about these things because in this state (Refer Slide Time: 12:22) only TMG is important. In this state only TY is important, in this state only TTG is important, in this state again TY, in this state only TSG is important.

So when I am monitoring a time of a particular duration the other time knows the consequences so I can have a simple timer started by clock and give different outputs when the count has reached let us say 1000, 2000, 500, 400 whatever time you fix and when that duration is counted

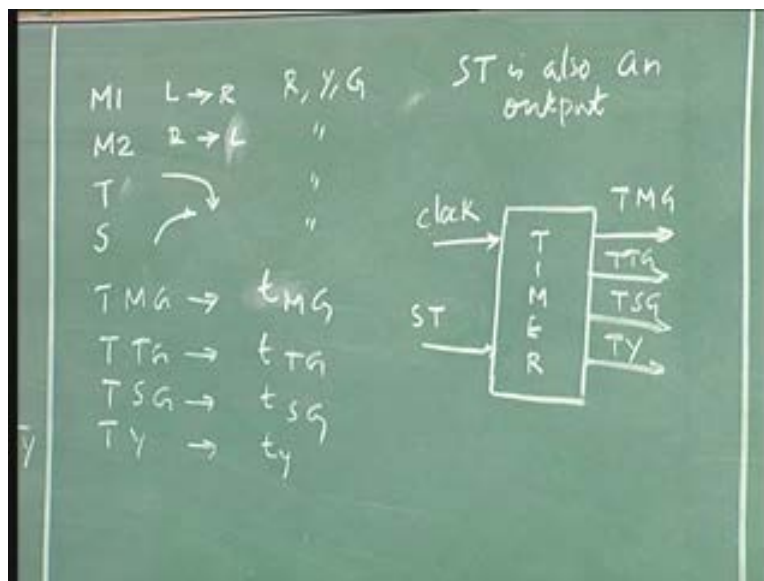
that will be the signal for this state to go from here to here for example, or the circuit to go from this state to this state. So the output of this timer would be; TMG TTG TSG TY all that is fine but when the clock goes on, but I want a starting point also only then I can fix my time. When the clock goes on it goes through all these.

Depending on the count, depending on the number of bits in the clock, number of bits in the counter, when the clock goes on the counter keeps counting to the highest possible count using those many bits goes back to 0 and starts all over again, one of those counts is a TY or one of those counts is TMG etc etc, whether how do you know went to start. So every time I need a fresh count to be started I need a signal called start timer. That means normally I will have a counter which free runs. But in this case I will not have a counter which is free runs. The counter will not count unless there is a count enable.

This start timer can be a count enable. Even though you have fed a clock to the counter it will not count until or unless you give a signal give called count enable. The moment you count enable comes, it starts counting from 0 upwards from wherever you left. So you can reset and start all over again, whatever it is, so it can have a reset and once you go to a state you can reset and when you want to count you start counting so when that particular count is reached that particular signal is...[....14:54].

So this ST is a start signal which is required to be generated also by this. So when I say these are the only light signals required out of this output, it is not really complete since I have forgotten I have left out a start timer. So one more signal to be start timer is also an output. for example, when you want to find the capacity the PROM required to do this we count the number of inputs and number of outputs, we have only counted the number of outputs as four lights, each of them have three states red, yellow, green so 12 outputs. There is really one more output that is ST that also has to be done.

(Refer Slide Time: 15:45)



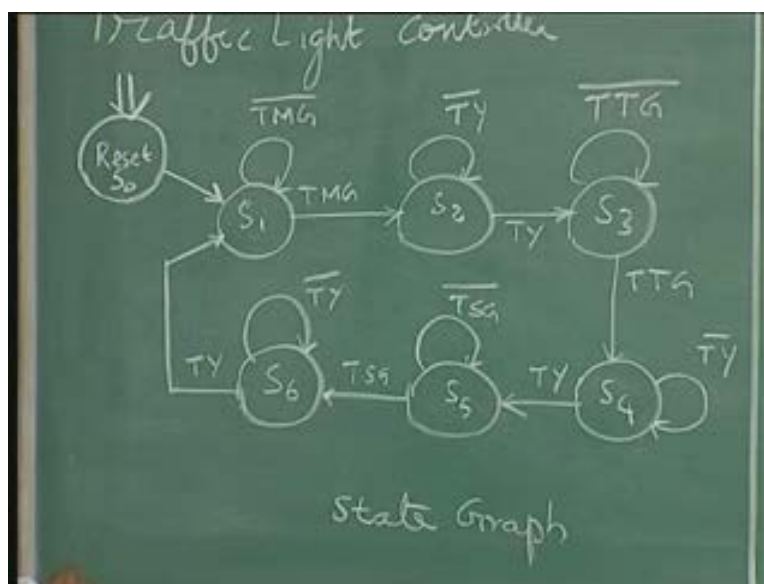
So whenever you want to start a timer you issue a signal 'start timer' and then it starts counting and then when a particular count is reached it goes, then the next time you want another count..... So when you reach the state you can reach that circuit, reset the count and can start all over again to the next sequence.

There is one thing I wanted to mention, it is an important hardware feature, of course concentrically it is not alright, but then as I said, final goal is to make hardware which works. Concepts can only take you so far and beyond that it is the nitty-gritty of the hardware, it has to make it work. That is the reason for all these bugs in your system. When you have a system, when you buy a computer or when you have an electronic gadget it doesn't work because the design is good but then somebody forgot to put on the earth properly, somebody forgot to give a signal which is supposed to clear the whole thing. These are the pit falls.

That is the way the design, these are the design specifications, this phase is very important. Then I can have one more state which I said the other day. If you want you can have a starting state or the reset state, normally when you switch on..... it can..... there are three flip-flops as I said already, a b c and each of these flip-flops go through, there are 6 states only so they will go to six different states and when you switch on for the first time any of these things could be the starting state and then we can go on from there, that is not a problem.

But generally, in any of these circuits we have a reset state because of you want to service or you want to switch off the lights, make all of them zeros, all the lights will be off, you want to keep the lights for a while, remove the clock from the system and then it will remain.....so usually a reset state will be there, we will call it S 0 in this case so this reset is a forced reset, when you switch on the system we can force the circuit to go to the reset state so that from then on it goes to the sequence that you want. So it is always possible to get S synchronous reset so this is what it is.

(Refer Slide Time: 18:27)



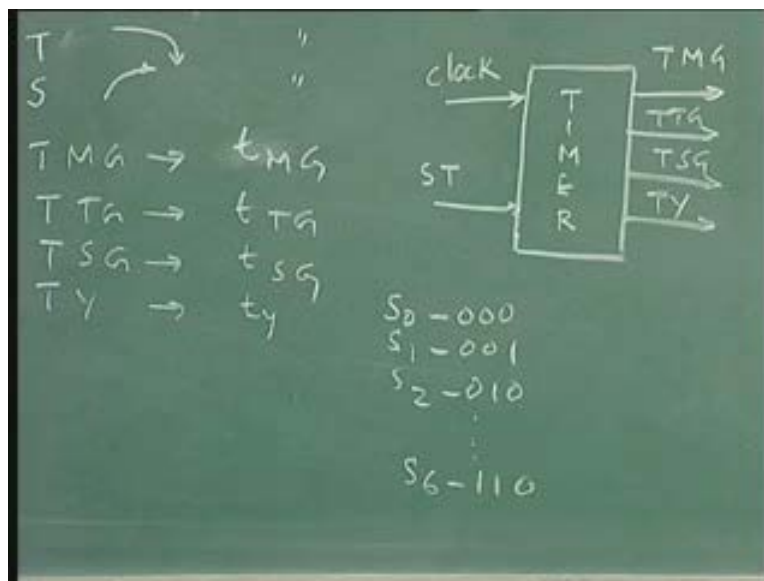
The system can be reset, we force the circuit to reset and start by a push button or something, it can be synchronous, all the counters can be 0 0 0 before you switch on the power for the first time, the circuit in the morning, you want to start the traffic light controller, start the circuit, switch on the power and then reset it by a push button then it gives you the 0 0 0 state and from then on it moves to the proper sequence. So when it is not there we will make that and when the circuit is in reset state switch off the power so when the circuit is in the reset state all the lights are off. That is one option; I am not saying it is the only way to do things.

Design is always open-ended, only one thing you have to make sure is that all the specifications are met, the customer gives you the design problem, he gives you a design assignment and you are supposed to carry it out and meet all the specifications then comes the efficiency the most efficient way you have to do it in a most cost-effective way, do it in a way it reduces power and consumption etc, these are the features. But first we have to satisfy all the requirements and whenever there is any design, design is always open-ended unlike analysis.

You make an analysis there is always one possibility so design is an open-ended problem, this is one possibility, this I am discussing that's all. you may think of some other way, as long as you think of a better way it's good for you, as long as I said it is not very inefficient compared to this, I am not saying it should be always better than this by saving at least one more gate, no. As long as this circuit is similar even by your innovative design even if it consumes a couple of extra gates that is better than taking somebody's circuit which is designed already so innovation is required in a design because some ideas which I have forgotten or overlooked will come out.

So with this background we can now write the state table and from then on you know how to proceed. I will call it state table, we are going to use d flip-flops, state table becomes the transition table, inputs and outputs will also be included. So to make life simple I will map S 0 as 0 0 0, S 1 as 0 0 1 and so forth. S 6 will be...(Refer Slide Time: 21:36), these are the seven states we need to implement.

(Refer Slide Time: 21:41)



So the present state input, next states outputs are as I said ST and then all these lights; main green, main yellow, M1 and M2 we call them, remember, M1 green, M1 yellow, M1 red and so on. In order to save space, what I will do is, I will right M1, M2, T and S.

(Refer Slide Time: 23:43)

Present state	Input	Next state	ST	M1	M2	T	S

We will always follow the red, yellow, green pattern so that I don't have to, so first light is, so this is the pattern, this is the order in which we will write the outputs (Refer Slide Time: 23:55). You may have to help me here to finish the table fast otherwise it will take for ever for me to finish this. We will call this A B C and A plus B plus C plus as usual, present state is 0 0 0, input is nothing it always goes to 0 0 1 and only light is 1, ST we have to get started. Because once move here the timer will start. As soon as I hear I can just start the timer because one clock count does not matter. I have done one clock count earlier than..... after reaching here it should count TMG but once I am there I don't know how to start so I will start it and then go here.

Suppose a 2000 count you want to program it we can count from 2001 or we can program it from 1999 doesn't matter really because one clock is a small, short duration and all other signals are 0. So when I put 0 it is a concatenation of all the signals, nothing else, it is a zero state, then the next state is 0 0 1 and in 0 0 1 there is one signal and TMG is off it stays in 0 0 1 and if it is on it goes to what? I asked you to help me so we can finish the table fast, why are you hesitating so much? Is it so difficult to read this, I am asking you because I can't read parallely you know. It is parallax. Otherwise I would not have asked you for this simple favor. Don't want to commit, why commit and get involved, that is the motto.

ST will be on only when you go the next..... here ST will not be on because we have already switched on ST and we have to wait for TMG so what will be the output, which lights will be on, which lights be off. for M1 it will be 0 0 1, M2 also 0 0 1 then turn 1 0 0 and this is the same thing depending on whichever state it is, in state s one the lights are there, these are what I told you as Moore machine so outputs are defined for the.... Except for this output all other outputs are defined for the state. ST along changes because ST depends on the timing. If the time is over

only I can start the timer. ST is a Moore output and all others are Mealy outputs. **ST is a Moore output because Mealy output**, ST is a Mealy outputs because I can only start the timer when I leave the state to go to the next state.

Next one is; in 0 1 0 the only thing to be monitored is TY because there are only two states and what are the two states, again 0 1 0 and 0 1 1. This table is the simplest table you can think of tonight. This state is always one more. The state graph is a simplest state graph and then again you start the timer when you leave the state, as long as you have a yellow state you count the timing, read the timing.

(Refer Slide Time: 28:39)

ABC Present State	Input	ABC Next State	ST	M ₁	M ₂	T	S
000	-	001	1	0	0	0	0
001	TMG	001	0	001	001	100	100
	TMG	010	1				
010	TY	010	0				
	TY	011	1				

So I am going to, for now, assume that ST resets and starts because otherwise I have to define another signal for resetting and then starting. so right now we will assume that ST has the, it will reset and start or it will rest and ok, it has to reset and start ok so we will assume that or otherwise it is going to be difficult to have one more signal to reset and then to start (Refer Slide Time: 29:04). So ST is a signal which means, that means the timer has to be defined with a proper input circuitry, may be flip-flops or counters but a little bit of extra combinational logic, it will take ST signal first for one clock cycle and immediately it will clear it and then it will start, one clock cycle.

All I need from you is this otherwise it is so simple. Tell me what is the lights, main... this is not yellow, it is yellow, M2 is yellow, ok, 0 0 1 then, then, ok, next one is what? TTG **tell me** what is the signal, 0 ok, **even if it is the same you have to tell me because I have no way of verifying looking through that**, then 1 0 0. This is 0 0 1 and this is 1 0 0, then 1 0 0 TY bar TY so TY bar is what? 1 0 0 and 1 0 1 and 0 1, then 0 1 0 and this one is 1 0 0 and this one is 0 1 0 then 1 0 0, (Refer Slide Time: 31:30) 1 0 1 TSG bar..... TSG 1 0 1, 1 1 0, 1 0 0 then 1 0 0 then 1 0 0 then 0 0 1.

(Refer Slide Time: 33:38)

ABC Present State	Input	Next State ST	M1	M2	P	S
000	-	001	1	0	0	0
001	TMG	001	0	001	001	100
	TMG	010	1			
010	TY	010	0	001	010	100
	TY	011	1			
011	TTG	011	0	001	100	001
	TTG	100	1			
100	TY	100	0	010	100	010
	TY	101	1			
101	TSG	101	0	100	100	100
	TSG	110	1			
110	TY	110	0	100	100	010
	TY	001	1			
111	-	000	0	000	000	000

Finally 1 0 1, 1 1 0, TY bar, TY. 1 1 0, 0 0 1, goes back to the starting state 0 1, 1 0 0, 1 0 0, 1 0 0, 0 1 0 and finally it is 1 1 1, **it is not going to happen** so irrespective of that reduce the don't care, we will put it 0 0 0 and all the signals will be 0. You make all zeros so that no problem occurs if it comes to this state. This is the state table or transition table if you want to call it or whatever.

Now if you want the PROM implementation how many inputs are there? Inputs are 3 for the state variables and 4 external inputs namely; TMG, TTG, TSG, TY so 7 inputs and outputs are 3 for the state variables, 12 for this and 1 for this so 13, so 13 plus 3 is 16. Of course as I said that there is a possibility to concatenate or compact the outputs by defining the relationship between....., you can always define a relationship between red light and a green light. For example, in the same street red and green cannot be on at the same time and the main street green and side street green cannot be on at the same time.

We use all those relationships we may be able to come up with a simpler output table and it is called ROM compaction. We want to reduce the size of the ROM the width of the ROM word width we can do that. They used to do that earlier. Today technology is not a big problem, the size of the ROM is not a problem at all especially for small designs like that. Nobody does any of those things it is called ROM compaction using the relationship between the different outputs if they are dependent, naturally here they are dependent. The outputs are not independent of each other, the outputs are dependent on each other, we can always reduce the number of outputs by deriving some relationship between these different outputs, **if you want to do that you can do it as an exercise**, ROM compaction.

(Refer Slide Time: 35:59)

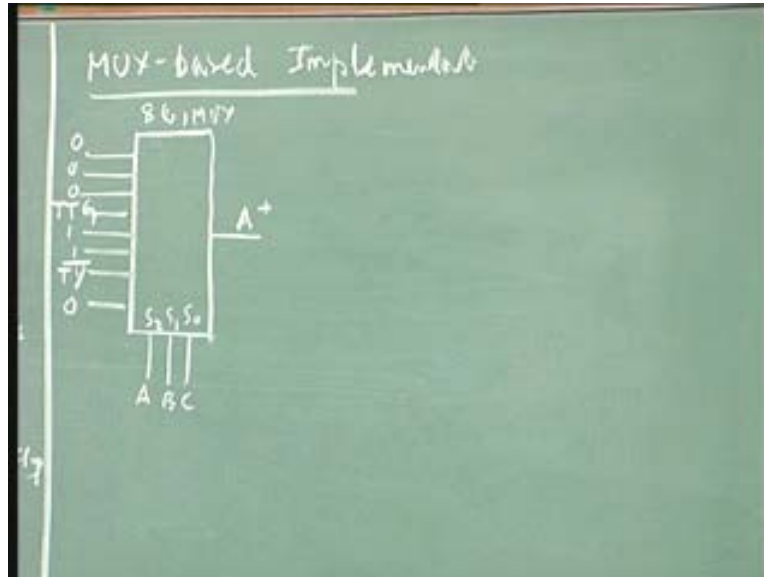
A B C Present State	Input	A ⁺ Next State	ST	M ₀	M ₁	M ₂	S
000	-	001	1	0	0	0	0
001	T ₁ T ₂	001 010	0 1	001	001	100	100
010	T ₁ T ₂	010 011	0 1	001	010	100	100
011	T ₁ T ₂	011 100	0 1	001	100	001	100
100	T ₁ T ₂	100 101	0 1	010	100	010	100
101	T ₁ T ₂	101 110	0 1	100	100	100	001
110	T ₁ T ₂	110 001	0 1	100	100	100	010
111	-	000	0	000	000	000	000

PROM Impl
7-input
16-output
Size
 $2^7 \times 16$ bits
PROM
+
3 D-Flip-Fly

So, the size of the PROM would be 2 power 7 into 16 bits and in addition we need 3 d flip-flops, 2 into 16 bits PROM plus 3 d flip-flops or you want gate solutions, for each one of these you need to draw a Karnaugh Map. For each of the next state outputs you have to draw the Karnaugh Map, next states you have to draw the Karnaugh Map, for each of the output 13 outputs you need it. So 13 plus 3 so 16 Karnaugh Maps you draw and simplify the expressions to get a gate solution. Or you want a multiplexer solution let us do that, that may be interesting.

What is a multiplexer solution going to look like? Mux-based implementation. For each output you need to provide a mux, 16 into 1 mux I mean 8 into 1 mux. So I will take the first one and the next state variable A plus. I will do for one or two and you can complete the rest. So the variables are A B C the present states and the next states be 1 2 3 4 5 6 7 8, this (Refer Slide Time: 37:15) will be A plus, next state variable A plus will be given by this mux, so all you have to do is to look at this column, so wherever there is a 1, this is a 0 0 0, 1 for TTG so this is a 0 0 0, if TTG is high this is high so this is TTG, this is 1, this is 1, this is TY bar, if TY bar is 1 if TY is 0 then this 1 goes to TY bar. All of know how to do this is it not from our earlier mux-based design. Look at the present state, the next state and what condition will make it 1 that we have to put for each present state. The present state is 0 0 0, no condition will make it 1 so put a 0, for 0 0 1 no condition will make it 1 so put a 0, in this case 1 0 1 present state whatever is the condition the next state is 1. If I take here the present state is 0 0 1, TTG will make it 1, TTG bar will not make it 1 so put a TTG.

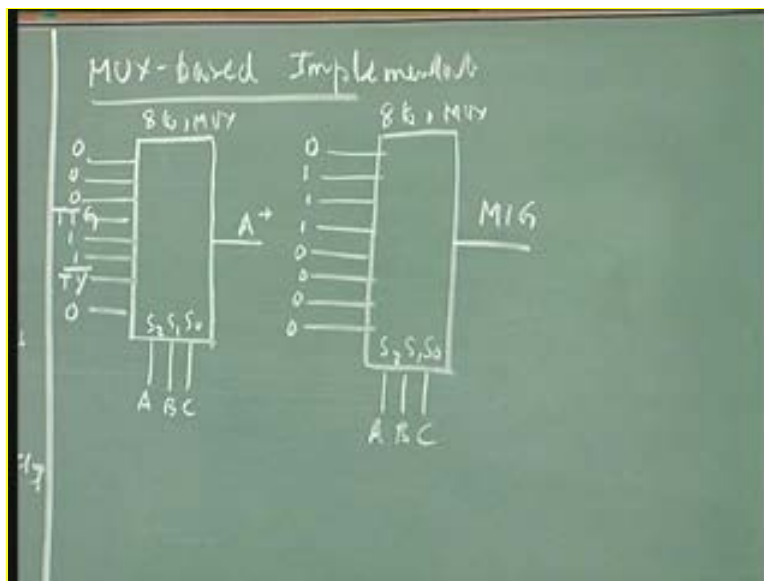
(Refer Slide Time: 38:41)



We have done this before, yes or no? My God, I thought you are dead, right. Class is alive and well. Now A plus B plus C plus let me not do A plus B plus C plus, you do it yourself, finish the design, let us do some of the outputs.

Main green M1 green M1 green, what are the conditions at which M1 is green? Let us look at these columns. In state S 2 it is 0, S 1 it is 1, S 2, S 3 first three states, rest of the states it is 0 so this is a 0, in S 1 M1 green, S 2 I am sorry, S 2 M1 is green, S 1 M1 is green, S 2 M1 is green, S 3 M1 is green. Put 1 1 1 rest of them are 0s.

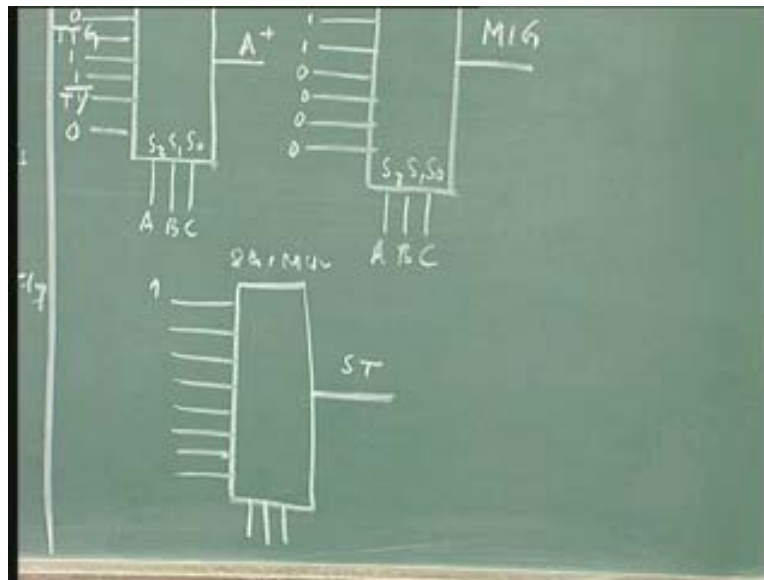
(Refer Slide Time: 40:50)



Is it too much task to do these things yourself? Please do this quickly. You see you have come on after a long holiday and early morning, many of you went home, why can't you be little more lively today? Then let us do the ST. Somebody's going to tell me what ST is. I am tired of doing it all by myself.

S T outputs, ok who is going to tell me ST outputs? You there, yeah. Quickly just tell me what are the inputs I need to connect to the mux. Look at S T column. No, we have these present states and under what condition S T will be 1 in each of the present states? Present state is 0 0 0 but under what condition S T will be 1? No condition, unconditional 1 so I put a 1. Now complete it.

(Refer Slide Time: 41:49)

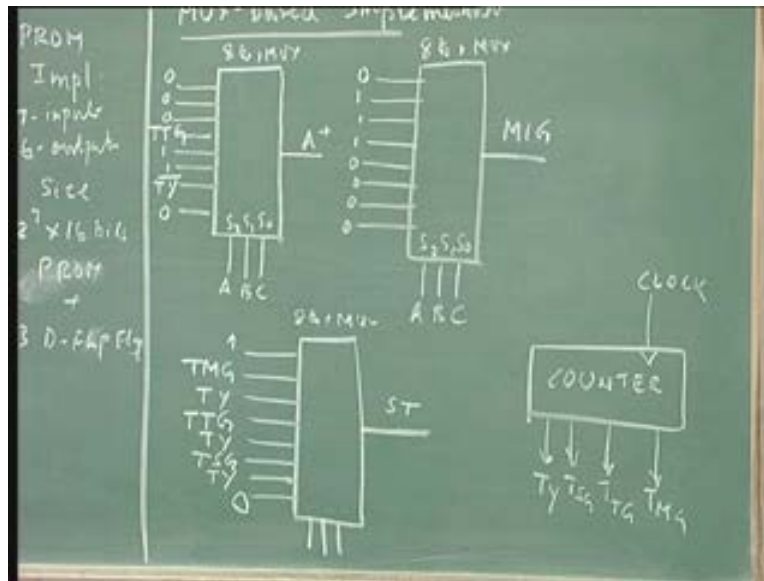


Then TMG, then TY, then say it again TTG, let him say because if too many people talk at the same time I can't hear properly, then TY, then TSG, and then finally it is 0, very simple. So we can do a multiplexer-based solution PROM based solution, gate based solution or PAL based solution, PLA based solution whatever hardware your boss wanted to use based on the stock and availability, sometimes the boss decides the type of hardware you need to [bec...42:48] that is the type of inventory you have in your company and you can use it or the same product line of your company, you don't want to use your rival's, competitor's hardware. These are the many reasons, practical and commercial reasons but you know how to do it whatever. Only one minor itch, I have not really completely taken care of how to convert my S T.

The S T has two roles; S T sets a.... this is a timer, the clock runs continuously the counter keeps running continuously. I want to start it from 0 and when a count reach..... So when S T has to start but the count let us say TMG I have used I need to keep counting, will keep counting so when S T comes it will, when S T is removed it will not.... the count is frozen and then the next time S T comes it starts from there. But I want to start or count always from 0 only then my timings will be right. My timings will be right TMG, TTG, TSG, TY will be right only every time I start with 0 then a particular count has reached.

So ST has a role of resetting the count. That is also possible because actually inside the count enable S T can do a resetting that is enough. Let the clock run continuously, if it resets it is enough. All S T should do may be, this is the counter (Refer Slide Time: 44:40) clock, **you understand the problem here**, and the counter is programmed to give different count as outputs; TY, TSG I am assuming **this in the order as this is**, the lowest count, next higher count, this is the order, this is not necessarily true but.....

(Refer Slide Time: 45:08)



There are two ways. The count can be enabled and disabled by S T rather than that let the count be on all the time but S T will always reset it to 0, from then on it starts counting it keeps counting, every time S T comes it restarts and then from 0 it starts so when the count is reached it is taken and then it will continue but next S T is anyway going to reset it again. So this could be my S T which is counted to reset. Actually it should not be called start timer, start timer in the sense it starts the timing of that particular interval S T stands for start timing but for that interval you start. So this could be the reset, once S T comes the first counter resets to 0 0 0 and then on it counts as I said, it will be one count less than what you want but it doesn't matter. Supposing I have put 1 KHz clock that means every millisecond it counts and I want 2 minutes or even 1 minute or I want 60 seconds so 1000 in 1 second and 60 into 1000 so 60000 counts I am going to count for TMG and if it is minus 1 it doesn't matter or you program it for 2001 that is ok, 1001 that is a minor.....so that also has been sorted out.

Now you have a complete solution for this. **The object of this exercise, the object of this exercise is all of these things are known to us. I don't think I have introduced any thing new.** You know how to design using multiplexers, you know how to design using PROM, you know how to design state graphs, and you know how to translate a state graph into a state table or a transition table. But what is new is the systematic understanding of the problem and defining the various states and the transitions, what are the transitions allowed and not allowed, when the transitions should happen and so on, go to the nitty-gritty in details of the signal that is required, how to generate those signals, what hardware will be required so that we will be able to come up with

the inventory of parts that we need to build this then the design becomes complete because the designer, the engineer is the designer, the role is to make sure that all i's are dotted and all t's are crossed and until that time it is not complete, anything left.... because it could be, you may have something in mind but then somebody else might interpret in some other way so it becomes ambiguous design and that is what we want to avoid.

So I want to show you only one more example. In this example what I did was I have used a very simple design in the sense the hardware required is very little, all I need is lights which are not under my control, we only generate signal for turning on; the green light, blue light, yellow light, red light and then if it is not there you have to boost the signal voltage and current levels and then get a huge thing circular disc which you will get in your traffic light stations here. Other than that there is hardly any hardware except the counter. This hardware (Refer Slide Time: 48:40) is called the controlling hardware.

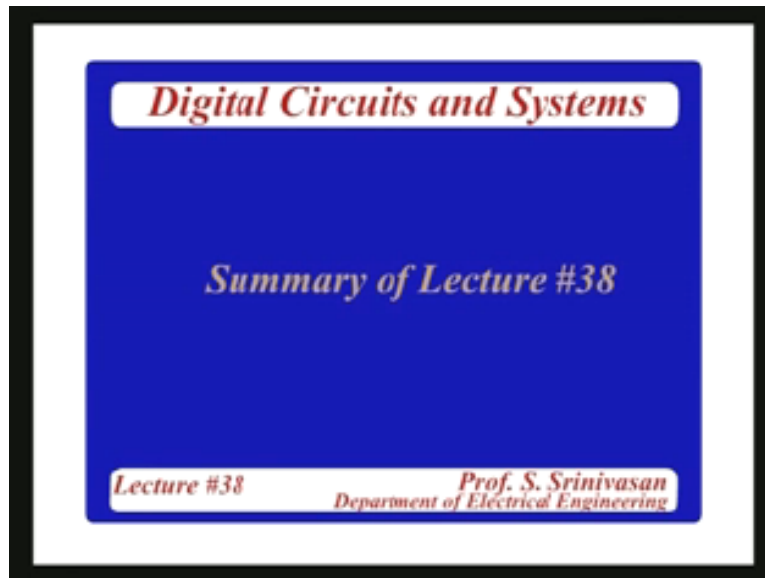
The state gate implementation is the sequencing, I told you this earlier, it is an algorithm, it is a program. Whether it is a programmable device you are using or fixed device you are using like multiplexers or gates it is the sequence through which you want the circuit to go through. So this table gives you the sequence of events and that is unique to the problem. Each problem has its own state graph and a state table and an implementation of the state table, you cannot do it in advance although the problem is known to you.

On the other hand, the circuit requires a lot of other hardware which is in this case a counter. Suppose when I give you a design in which large number of these hardware is available, take a computer for example, you may give the sequencing of the various function units but then there is a standard arithmetic logic unit, that is a memory, that is an input device, output device, I don't think we should design each one of them every time, this design has to be done, this is an algorithmic design, state graph has to be designed, you draw the state graph program it for proper use. This is called the algorithm, **as I said the other day**, you can call it sequence if you want to call it or procedure you want to call it procedure for going from state to state the sequencing or algorithm. But the other type of hardware which is a commonly used hardware which may be required for your design in this case is very little except the counter may be and lights which are not under our control.

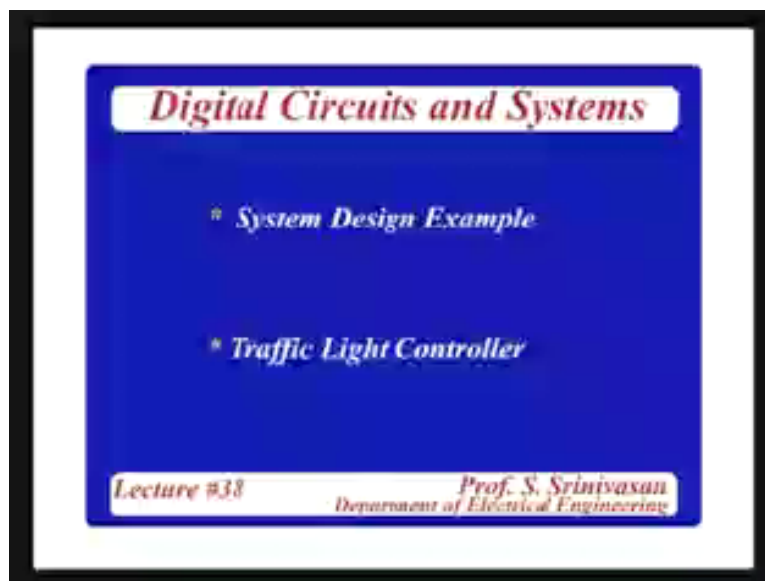
Circuits like a computer, microprocessor, calculator which may need lot of off-the-shelf components available components I need to get them, I need to connect them together, I need to functionally define what I want, I need to connect them together, I need to operate on them, I need to give signals for them to start, signals for them to clear, stop. So if the problem is little more involved I will use two types of hardware: One type of hardware is what is known as off-the-shelf components or available components which can be bought any standard store, usually a 4-bit ALU which is all you need to say and other hardware is the hardware which will result because of the state graph you will implement for that particular problem. For each problem you need to draw a state graph, state table, mux design, PROM design or PAL design. In addition, this extra hardware like the counter, like the flip-flop, like the shift registers, like ALUs, like memory chips those things I don't have to design again and again and again these are off-the-shelf available.

So the next example I want to take the case of a design in which we will view some of those, we will have a mix of some of the off-the-shelf components, again a simpler design of how much you can do in a class, a simple design but conceptually you will have to partition the problem into two parts, the parts which are available you should know how to control them, the other part is designed by us, the controller, which issue the signals for controlling them to get the signal if you want, that will be the last example we will be working out in this class.

(Refer Slide Time: 51:50)



(Refer Slide Time: 00:51:59)



(Refer Slide Time: 00:52:13)

Digital Circuits and Systems

End of Lecture #38

Lecture #38 *Prof. S. Srinivasan*
Department of Electrical Engineering

This slide features a blue background with a white border. At the top, the course title "Digital Circuits and Systems" is written in a red, italicized serif font. In the center, the text "End of Lecture #38" is displayed in a yellow, italicized serif font. At the bottom, a white horizontal bar contains the text "Lecture #38" in black, followed by "Prof. S. Srinivasan" and "Department of Electrical Engineering" in a red, italicized serif font.

(Refer Slide Time: 00:52:20)

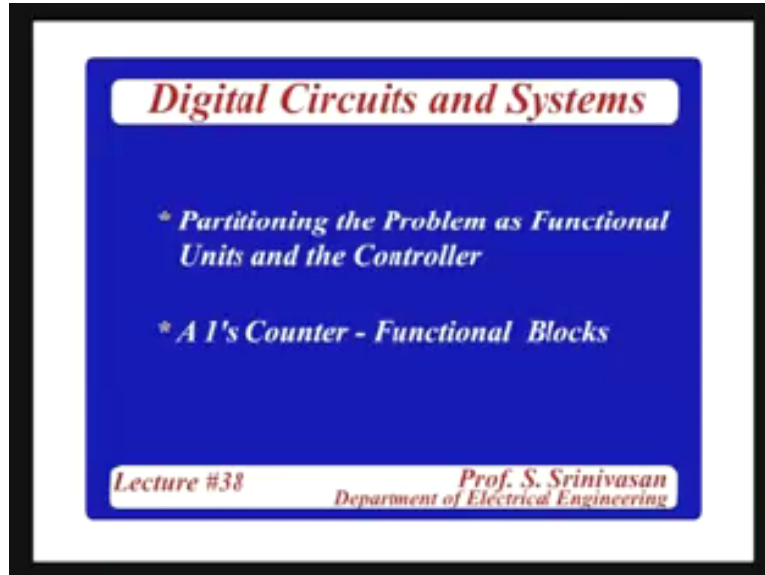
Digital Circuits and Systems

Next Lecture

Lecture #38 *Prof. S. Srinivasan*
Department of Electrical Engineering

This slide features a blue background with a white border. At the top, the course title "Digital Circuits and Systems" is written in a red, italicized serif font. In the center, the text "Next Lecture" is displayed in a yellow, italicized serif font. At the bottom, a white horizontal bar contains the text "Lecture #38" in black, followed by "Prof. S. Srinivasan" and "Department of Electrical Engineering" in a red, italicized serif font.

(Refer Slide Time: 00:52:30)



Digital Circuits and Systems

- * *Partitioning the Problem as Functional Units and the Controller*
- * *A 1's Counter - Functional Blocks*

Lecture #38 *Prof. S. Srinivasan*
Department of Electrical Engineering