Digital VLSI System Design

Dr. S. Srinivasan

Department of Electrical Engineering

Indian Institute of Technology, Madras

Lecture - 19

Examples of System Design using Sequential Circuits

Slide – Summary of contents covered in previous lecture.

(Refer Slide Time: 01:06)

Slide – Summary of contents covered in this lecture.

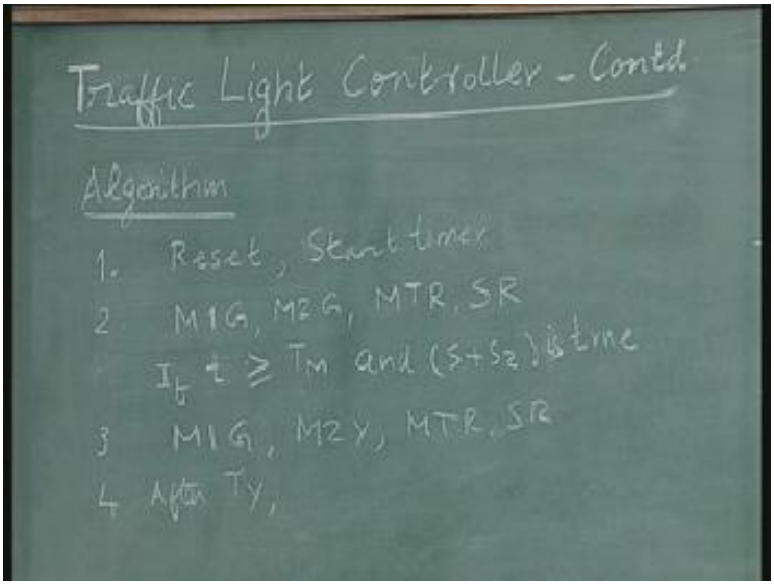
(Refer Slide Time: 01:45)

In this lecture, we will continue the design of traffic light controller, which was introduced in the last lecture. In the last lecture, we specified the problem with all the requirements of the traffic light control. We do the ASM and now the next step, as you know, is to draw the ASM table and get the hardware implementation. The ASM table can be drawn easily from the ASM chart, which had been drawn in the last lecture. Once you have the ASM table as a mapping, the hardware is easy. In order to recapitulate what we did in terms of specifications, we will go over the specifications of the traffic light controller. At first, it is in Reset State. We will say, this is sort of an algorithm, but not in the ASM form. To give you an example, we will give you a quick review as well as a summary of the specifications. Reset State - At Reset State, the timer is started. Remember, we discussed the lights as M1 and M2.

M1 Green, M2 Green (both the main road lights are Green), MTR - Main Turn right (turning from the main road into the side road - the right turn is Red) and SR - Side turn (turning from the side road to the main road, again right, is Red). This will go on for the time t, if t is equal to or greater than T_m and S_1 or S_2 is true. This means, after a specific time has elapsed in the Main road, Green and if one of the sensors S_1 or S_2 is activated, we will go to step 3. Here, we will retain our Main 1 as Green (M1G); Main 2 will become Yellow (M2Y); Main Turn will be Red (MTR); Side road will be Red (SR). T_Y is

a fixed time. This means, when you have a Yellow in any of the lights, then it will allow for a while, for the traffic to slow down and stop.

(Refer Slide Time: 05:20)



T_Y is called Yellow Timing. After T_Y, Main 1 continues to be Green (M1G); Main 2 will become Red (M2R); Main Turn will become Green (MTG); Side road going from the side to the main is still Red (SR), because you can have only one turn at a time and cannot have both turns at the same time, as it will lead to collision. We have a specific time interval T_S, when the Side street will be Green or Turn signal will be Green. After T_S, Main 1 continues to be Green (M1G); Main 2 will be Red (M2R); Main Turn will become Yellow (MTY); Side continues to be Red (SR). now we have a Yellow period again after T_Y.

We are considering turning because of S_1 or S_2 being true. If S_1 was true and S_2 was not true, then there is no requirement of traffic from the side road into the main road, then S_2 will not be on. In which case, we can return to the main sequence without having to spending time on giving the Green signal for turning from side road to main road.

On the other hand, if S_2 happens to be 1 or because both S_1 and S_2 were on, we would like to service the side road turning into the main road, before we go to the main loop. This has to be tested here. After T_Y, if S_2 is 0, then we go back to Step 2. On the other

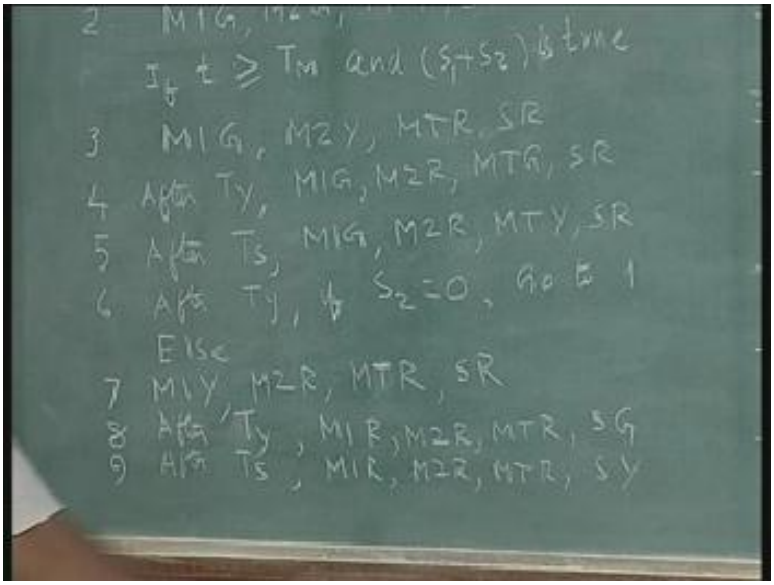
hand, else, after T_Y, if S_2 is 1, then we will have to go and stop the main road traffic on both sides and allow turning from the side to the main road.

In this case, what we are specifically assured is, that the main road from left to right is Green when you allow the turn from main road to side road. The same main road from left to right, will not be Green when you allow the turn from the side road to the main road. The traffic on the main road should be stopped in both directions.

This means, Main 1 is Yellow (M1Y); Main 2 continues to be Red (M2R); Main Turn is Red (MTR); Side Red (SR). After $T_{Y,}$ it becomes - Main 1 Red (M1R); Main 2 Red (M2R); Main Turn Red (MTR); Side Green (SG). Both sides of the main road traffic as well as the turning signals from the main to side are all Red. Side street alone is Green, so that the traffic is allowed to go.

After T_S, the side street period for which it is Green, will go back and continue to be Main 1 Red (M1R); Main 2 Red (M2R); Main Turn Red (MTR); Side is Yellow (SY).

(Refer Slide Time: 10:14)

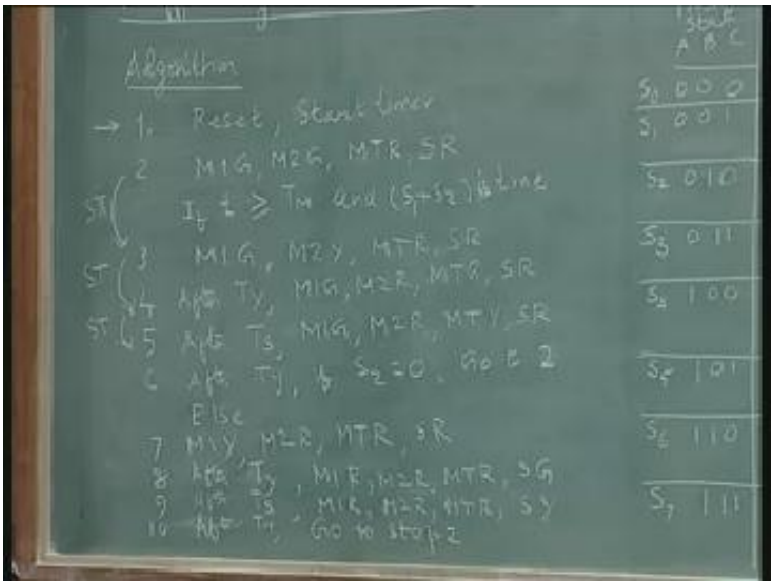


After T_Y, we can restore the original condition of Main 1 Green (M1G); Main 2 Green (M2G); Main Turn Red (MTR); Side Red (SR), which is Step 2, go to step 2. Step 1 is the Reset State, to start the timer, so that the sequence starts. Most of the time, assuming

there is no sensors activated anytime (S_1 and S_2 are both 0s), the system will continue to be in state 2, which is M1G, M2G, MTR, SR. As we said, this is a quick summary and each stage we have not shown something - each stage we have to start the timer. When you go from Step 2 to Step 3, we start the timer. When we go from Step 3 to Step 4, we start the timer. Likewise, when going from one State to the next State, the timer should be started. I have not shown it here. What I should really say? If T_Y is greater than $T_{M,}$ and S_1 is true, start timer and go to M1G. Similarly, I say M1G, M2G, MTR, SR and after $T_{Y,}$ etc, Start Timer. So Start Timer should be put in each stage here; it is assumed that Start Timer has been used in each stage.

This was just to recapitulate the ASM, because it looks a little complex. Now with this background, we want you to look at the ASM table. It is a huge table. Question by the student (Refer Slide Time: 12:27). That is a good question. Since we are testing for S_2 before proceeding or returning back to Step 2, why cannot we do a similar thing for S_1? That is the question. That means, we should check S_1 and S_2 every time and take appropriate action. Yes, it is done. Here, we wanted to introduce some specifications and show how these specifications are properly met in an ASM and the hardware.

(Refer Slide Time: 12:59)



Of course, there are design flaws. Design flaw is different from circuit flaw. It means that, you may not think of a case and as long as it works for the specification to give hundred percent, it is perfectly okay. We deliberately chose this; already the ASM and the number of states and …the first example was a very simple example. This example is in order of magnitude more complex than the first example. We can do that - we can check S_1 first, as we did with R_A and R_B in the first example (Request by System A and Request by System B for the bus).

R_A and R_B were tested and then some sort of priority was set. Here too, we can do the same thing - S_1 and S_2 is tested in sequence. Either S_1 or S_2 can be given priority. We can do that. Then each of these S1 and S2 testing should be done, after every change of signal. ASM becomes more complex and more number of states will be introduced. If that is what you want, you can do it. This example clearly defines that every time we will consider either of these two signals. We are going to give the benefit for main road to side road traffic, which is a reasonable assumption.

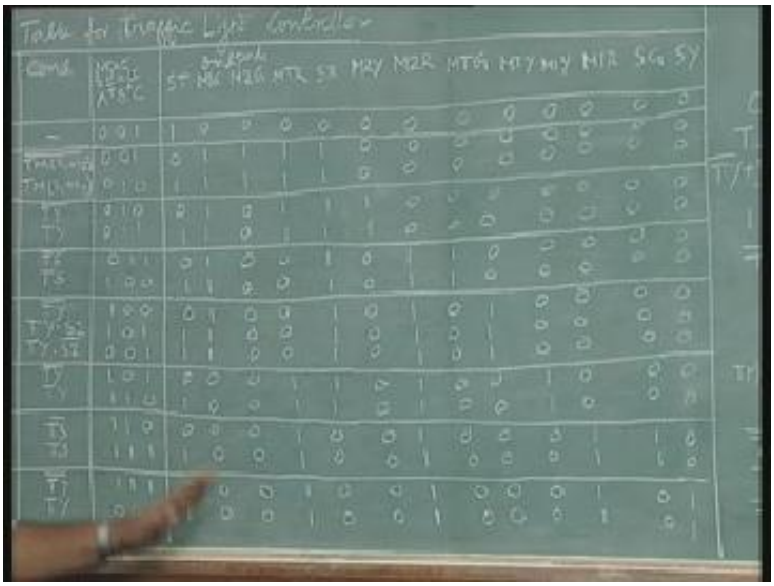
Some of the traffic will go from main road to side road is an inherent assumption in the problem. We are not making this assumption after the problem is given to me. We are assuming this specifications that main road will always be given a chance. In fact, it would have been easier for me to not go through this test, if we did not have this assumption. There are several possibilities. Suppose, we did not have a sensor and allow the main road to be green for a while and after this time, we allow traffic from main to side for some time and then from side to main for a while and then go back to the original - this is one simple way of doing. The number of states will be reduced. Another modification will be - one of those sensors will be put in a place where it is less likely to occur. Third is to put both sensors and then verify each sensor every time. This will become more complex. We thought of taking an intermediate path - we wanted to make the problem reasonably complex and another reason is also make it slightly different from the routine problem that you find in any Traffic Light Controller example. Traffic Light Controller has been a problem of choice for many authors. You will find the main road - side road traffic problem in almost every book in digital design from gate level to ASM to FPGA to ASIC to microprocessor base.

Of course, we will understand the standard example when we read it, but when we give you a slightly different specification and explain it and you understand that we will be confident that you have understood the concept. Now, we go to the ASM table. We are not expected to memorize this table - it is a huge table. As you can see, there are 8 states here S_0 to S_7. For each state, we have given a binary assignment. It is called a natural binary assignment without any specific guideline for states.

S_0 corresponds to 0 0 0; S_1 to 0 0 1; and similarly S_7 to 1 1 1, hence 8 states and 3 variables A, B, and C. The conditions are very simple - Timer Yellow (T_Y) not completed, T_Y completed, Timer Side street (T_S) not completed, T_S completed, Timer Main (T_M) completed, T_M not completed. Sensors S_1 and S_2 are both marked at the first stage, whereas only S_2 is monitored in the second stage. Based on this, they go from one state to the next. You can correlate with the ASM that we drew. The condition from one state to the next state and what is the output in each of these states.

For example, in state S_2, which is 0 1 0, all we are looking at is, yellow. This is M2Y, where we are allowing for the yellow time to be over before it turns to red to allow the main traffic to side road. If it is not there, you continue to be in the same state. After this, you go to the next state and so the lights are correspondingly marked here. There is no start timer. The start timer will come only when you want to go to the next state. Whenever you want to go to the next state, you start the timer. ST stands for the Start Timer. These are output signals, which correspond to the lights – Main 1 Green (M1G), Main 2 Green (M2G), Main Turn Red (MTR), Side Red (SR), Main 2 Yellow (M2Y), Main 2 Red (M2R), Main Turn Green (MTG), Main Turn Yellow (MTY), Main 1 Yellow (M1Y), Main 1 Red (M1R), Side Green (SG), and Side Yellow (SY). The reason it is written in random order and not in the order you would normally like to see – M1G, M1Y, M1R, M2G, M2Y, M2R, which is one way of writing is because, this is the way in which these outputs appear in the ASM chart. Hence, we put them in this order.

(Refer Slide Time: 19:00)



Corresponding to this state of 0 1 0, you are waiting for the yellow time to be over. As long as the yellow timer is not over, you continue in the same state, keeping the Main 1 Green (M1G), Main Turn Red (MTR), Side Red (SR), Main 2 Yellow (M2Y), all others are 0s. When the turn occurs and the time is over, everything is same except that we start the timer. It is the same set of outputs with Start Timer (ST) 1 and then it goes to next state 0 1 1, wherein the yellow has completely turned to red and the turn has become green. It will become: Main 1 Green (M1G), Main 2 Red (M2R), Main Turn Green (MTG), all others are 0s. This is how you write the table.

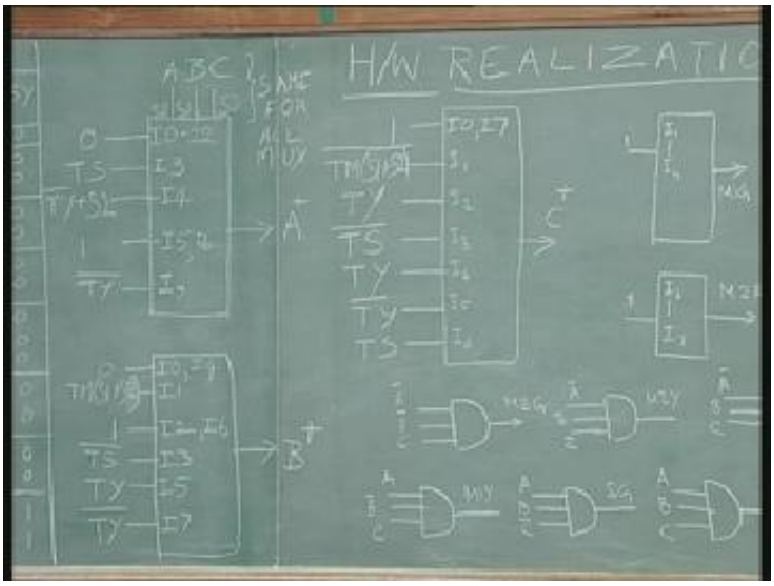
Once you have handled this,… purposely chosen problem with so many things, you know that you can handle systems. The system complexity does not change the design complexity. The design and procedure are the same. With more number of variables and more number of conditions, the ASM becomes larger and the number of inputs and outputs become large. So, you have to handle them carefully. You have to access with extreme care in drawing it. Once you draw this, this is going to be implemented in hardware. If this is all right here and you want to do a verilog coding, the whole thing will be written in a verilog and then you have to synthesize this and get into FPGA or whatever it is. You have to be extremely carefully, because if you make a mistake then

you have to redo it all over again and then you have to recompile and do everything again and again until you are satisfied.

We are not going to explain this. We have already told you how to get this table and the hardware blocks. These are A, B, C - present state and next state we have already seen how to do a multiplexer based solution for this. We have 8 to 1 multiplexers with 3 selector variables A, B, C. Hence, there are 3 multiplexers required here - one for A, one for B and one for C. There is a multiplexer for A, a multiplexer for B and a multiplexer for C (Refer Slide Time: 21:35). We call it A plus, B plus and C plus, indicating they are next state variables. Their states are given by these inputs. ABC inputs are given to, selector 2, selector 1 and selector 0. An 8 to 1 multiplexer will have 3 selectors – S_2, S_1 and S_0**,** to which we are applying the present state ABC.

Present state A B C is given, next state variable A plus is the output. Present state ABC is not shown here, as the drawings are already too many and cluttered. We do not want to increase the complexity of the drawings. A B C are the inputs and B plus is the output. A B C is input and C plus is the output. What to give to each of these inputs depend on this table.

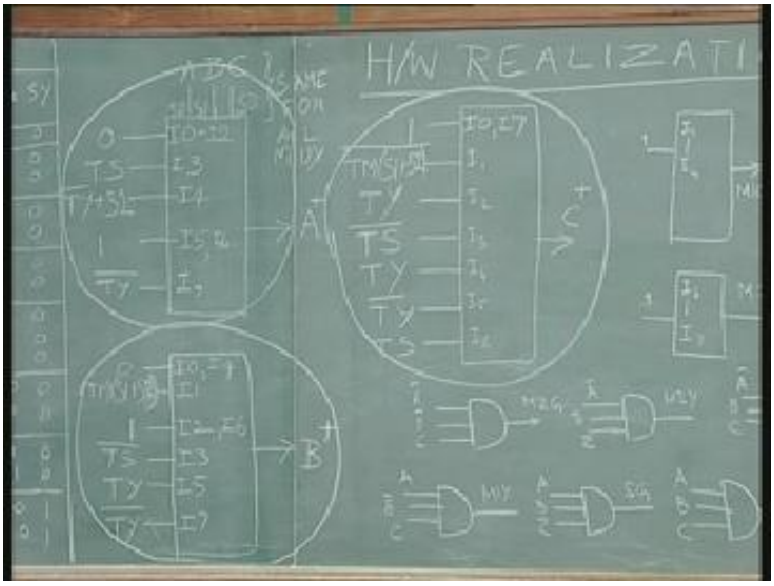
(Refer Slide Time: 23:15)



For example, when you are in state S_0, the condition under which the next state is 0. For the first input to the first multiplexer, corresponding to A, you give 0 as input, I_0. Wherever it is 0, we have combined them – I_0, I_1, I_2 are all 0s, because first 3 entries are all 0s. I_1 corresponding to the first state; I_2 corresponds to the second state; I_3 corresponds to the third state.

For all these states, the next state variable of A is 0. Therefore, I_0 has to be 0, I_1 has to be 0, I_2 has to be 0. All of them have been combined again to reduce the drawing. Like this, you can write the input conditions for each of these inputs I_0 to I_7, for each of the multiplexers A, B and C. The outputs can also be given by multiplexers. We will give you one example and you can see the rest. For example, we will see how to get this start as an output. These inputs - the next state variables, A plus, B plus and C plus (Refer Slide Time: 23:38). Each one is an 8 to 1 multiplexer with 3 selector variables S_2, S_1 and S_0 connected to A, B and C. 8 inputs - I_0 to I_7 are connected to corresponding conditions based on the table as already explained.

(Refer Slide Time: 23:49)



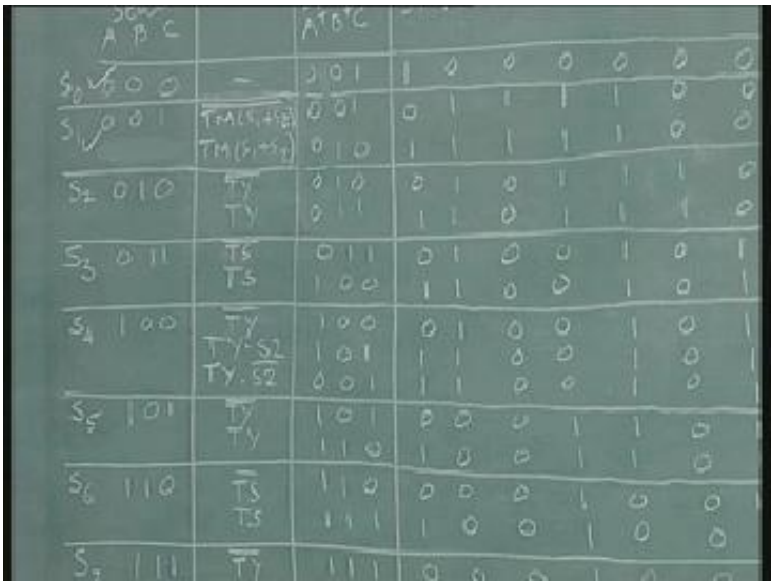
One or two are here and this is how the rest has to be drawn too for output. Before we go to the output, we will tell you how to get the output, by taking one example. Let us take start signal as example – the rest you can follow based on this. Again, there are only 8

states. Hardware Implementation for ST (Start Timer): - 8 inputs, 1 output, MUX. A B C, present state variables, three selectors – S_2 S_1 S_0. This is the start output F, which is the output of the multiplexer; 8 inputs - I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7. The condition under which the start signal is 1, is all you have to map in this multiplexer. We have to put this by looking at the conditions under which start is 1. Start is 1 for the present state 0 0 0.

If you are in present state 0 0 0, then start is 1. Start is also 1 in S_1, provided the condition T_M (S_1 or S_2) is valid. Corresponding to I_1, we have to put T_M (S_1 or S_2), which means after elapse of the time T_M - the Main road Green Time and if one of the sensors is true, then start timer will be initiated, when you are in state 1. This is how you will map it. We have taken care of this by simply looking at the table and writing. For everything else, it is the condition. For S_2, the output Start Timer will be 1, if T_Y is 1. Hence, it is T_Y.

In S_3, it will be 1, if it is T_S. Usually, the start timer is after elapse of the time. After the elapse of T_S, in this case, it is one of these 2 conditions - T_Y S_2 or T_Y S_2 bar, which can be combined and written as T_Y because S_2 and S_2 bar get cancelled, because S_2 plus S_2 bar is 1. It is again T_Y. When you are in S_4, S_5, it is T_Y again. Almost all of it is T_Y. S_6 is T_S. S_7 is T_Y.

(Refer Slide Time: 27:10)

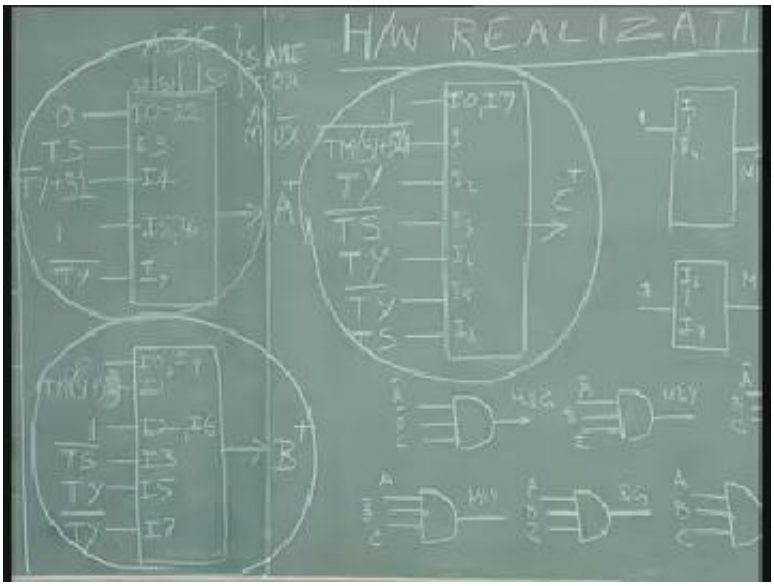


Remember, we had a timer in the functional block. The timer started with a Start Timer signal. We had the clock and then these conditions T_M, T_Y, T_S. The functional units of these things are available here and depending on which state you are in after this. Depending on which state you are in, the corresponding input will be given, so that the start timer will get started again. You are resetting the timer again and again. The output and the input are same, which means 1; that the start timer is reset and gets started again.

This is an example of one of the outputs. In all, there are 13 outputs, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13. Among 13 outputs, one is here and all other 12 outputs correspond to the 4 lights. There are only four lights - Main 1, Main 2, Main Turn and Side. Each of the 4 lights will have a Green, Yellow and Red. 4 times 3 is 12 lights. Start Timer is 13^{th} signal, we have already shown here. The 12 lights are very easily written. Whenever the input is in the state I_1, I_2, I_3, I_4, then M1G is on and rest of the inputs are 0. All inputs not shown are 0s. Similarly for MTR its 1 for: I_1, I_2, I_5 to I_7, and I_1 to I_5 is also 1 for SR. You can write like this. Sometimes, we do not even have to use a multiplexer. These are 8 to 1 multiplexers – 8 inputs, 1 output, 3 selectors. When the number of outputs is not large, you do not need to use a multiplexer - you can use a simple gate. For example, M2G is A bar, B bar, C - only one case, so put that here. M1R is only A, B – put a 2 input gate.

If you do not want these gates here, you can also replace them by multiplexers. You know how to do that. In summary, we have taken an example. Question by the student (Refer Slide Time: 29:58). The question is I have not shown the state flip-flops. All of you know that in a multiplexer based system, there will be state flip-flops. We will now remove this table, which you do not need.

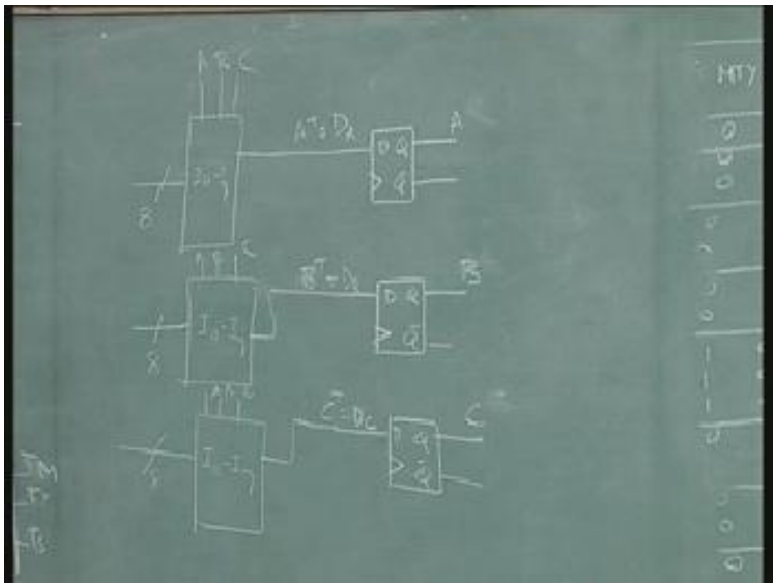
(Refer Slide Time: 30:10)



We are not drawing the full block diagram. There is the question about where to include the state flip-flops - there are only 3 flip-flops. We think this is something which we already described in detail in previous state graph technique as well as the ASM technique using multiplexers. Each variable will have a flip-flop; it is a D flip-flop. Q is connected. This will be my A plus (Refer Slide Time: 30:55). A and A plus are the same, because the next state becomes present state and then the complements. We will have A, because A plus is the input.

This is the input corresponding to D_A, which is also called A plus; B plus or D_B - we think we have done all these things again and again. You cannot continue doing this in every problem. It is assumed architecture. Each of this is given by a multiplexer, which is 8 to 1 and this is connected here. A, B, C are the inputs selectors. I_0 to I_7 will be connected as shown in this diagram.

(Refer Slide Time: 32:45)



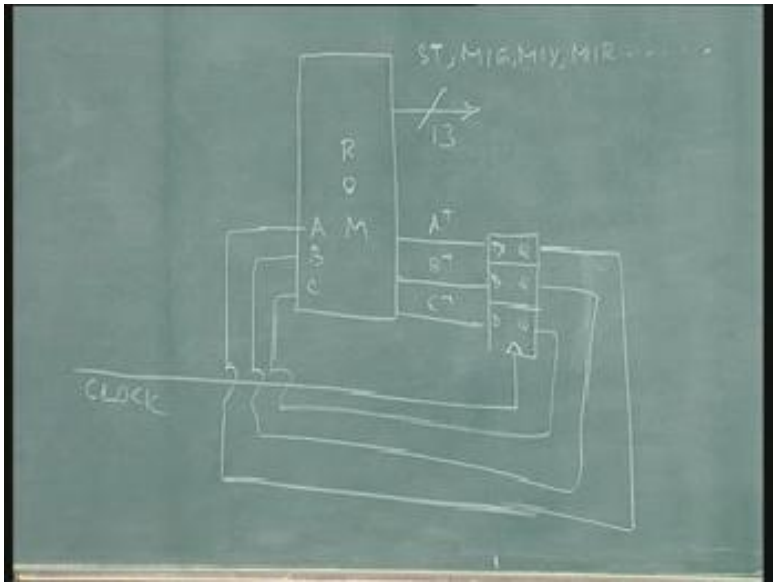
8 inputs of this will be connected according to A, B and C. We have extra circuitry using multiplexers for output functions. To summarise, we have taken an example. Of course, it will not work without the clock. We have clock here and all of these things (Refer Slide Time: 33:09). Given a problem, understand its specifications, systematic step by step procedure called the algorithm, and then write an ASM chart, identify the signal inputs and outputs for the controller, get the ASM table, get the implementation using flip-flops or multiplexer. Go to the output, also using multiplexers or gates, do the same thing for the outputs in multiplexers or gates, and then connect the whole problem together like this.

We do not want to do this architecture repeatedly. We did this in the case of state graph and in the case of first example as multiplexer. To avoid repetition and as you are familiar with this by now, output of multiplexer drives the flip-flops. That is it. This architecture is standard. The size of the multiplexer depends on the number of states - up to 8 states multiplexer will be 8 to 1; up to 4 states multiplexer will be 4 to 1; up to 16 states multiplexer will be 16 to 1. The inputs of the multiplexer are the only thing that is different.

The inputs of the multiplexers will be the only thing different from problem to problem. To that extent, it is a very simple design - it is all fixed, except to find that will be returned directly from the ASM. Of course, we want to have a ROM based solution for this. Yes, it is possible. What will be the size of the ROM it will require - supposing we have 8 states?

The ROM keeps everything and drives the flip-flops. It is called the flip-flops register clock, which is common to all the 3. The outputs of this are - A plus, B plus, C plus - D, D, D, Q, Q, Q; clock is common in all the 3 flip-flops. This will be your A, B and C. In addition, we should give all the inputs and get all the 13 outputs starting with ST, M1G, M1Y, M1R, etc.

(Refer Slide Time: 36:27)

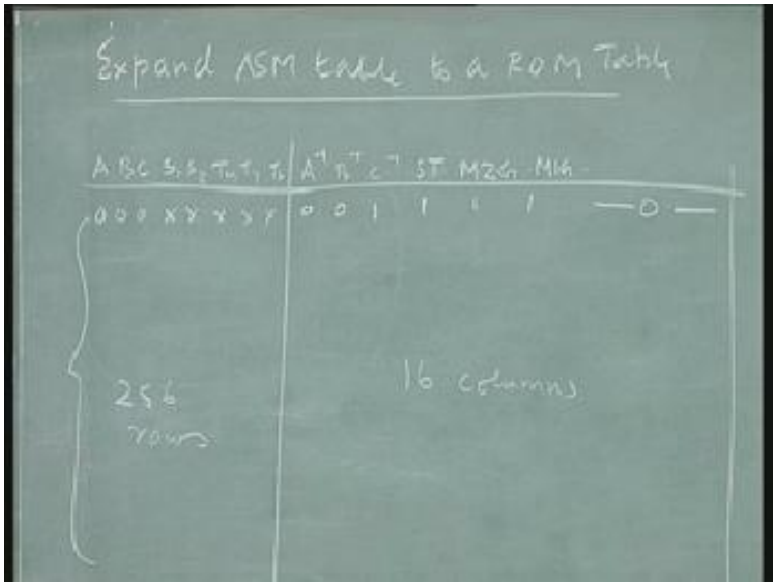


How many inputs will we have here? The input will be T_M, T_Y, T_S, S_1 and S_2. These are the inputs to this. The size of this ROM would be 5 plus 3 equals 8 inputs and 13 plus 3 equals 16 outputs. Hence, its size is - 2 to the power of 8 into 16 or 256 into16 bits or it is 512 bytes. We have a 512 bytes organized ROM. We can realize all this with additional 3 flip-flops. This is a ROM based solution. It is very simple. Our ASM chart can be expanded. In the ASM chart, we had only rows corresponding to a condition. There were 8 states 0 0 0 to 1 1 1.

When you expand each of these states, there will be 256 rows - one row for each condition. The size of the truth table would be 256 rows and there are 3 inputs, 5 states and 5 conditions. We will have to write a row for each of these states and each of these conditions. Our ROM truth table will have 256 rows with each row having 13. Expand the ASM table to a ROM table. The ROM table will have 256 entries. A, B, C, S_1, S_2, T_M, T_Y, T_S are the 8 inputs, which mean 256 rows and there will be 16 columns. This is how you got this number, 256 times 16; there are 256 rows and 16 columns.

The ASM table has to be expanded - for example, 0 0 0 state goes to the next state 0 0 1, whatever the condition may be. You can refer the notes and ASM. In A B C, there is no condition; we go to in S_0 there is no condition; you will always go to 0 0 1. The output in the state is start timer ST. The lights can be main lights.

(Refer Slide Time: 40:55)

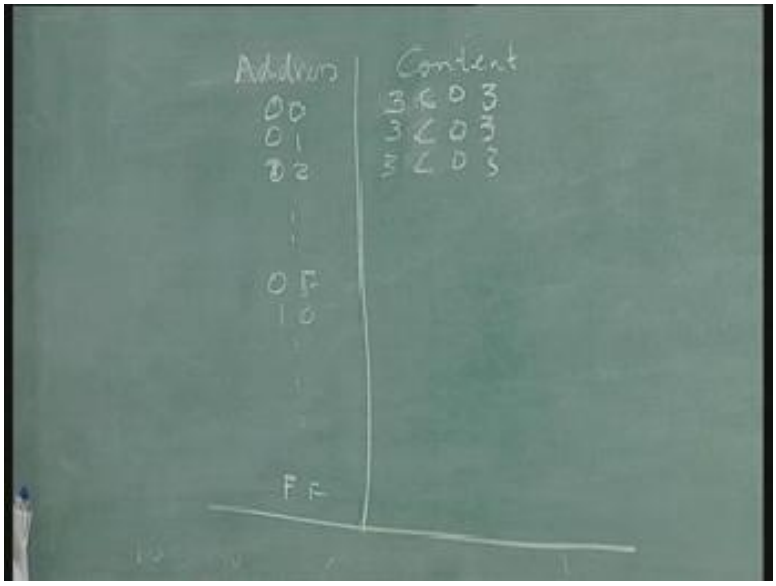


Irrespective of the condition, this will be M2G and M1G and rest are all 0s. You want to write red and that all, we are not going to fill as this one is an example. Now, this x x x will not hold in a ROM table, whereas in an ASM table we can have a x x x, because we are going to look at conditions for next transition.

Here, these five Xs have to be expanded in 32 rows. There are 5 inputs - we have to start with 0 0 0 0 0; 0 0 0 0 1 and so on for 32 rows, where all of them have the same output.

This block will have the same output. You have to store the same 16-bit word in the first 32 locations in the ROM. This is repeated for the second and then exhausts all the 8 states. This is how you get the ROM table. All you need is to get an expanded ROM table and program the ROM for this application. Of course, you do not give it in this form. In a ROM table, you know you will do in hexadecimal form. ROM table is not returned in binary, because it is so huge to write. ASM table, state table or truth table can be written in binary. ROM table is generally written in hexadecimal form to compress the size of the table. In hexadecimal, this will be written as 0 0, 0 1, 0 2 and so on.

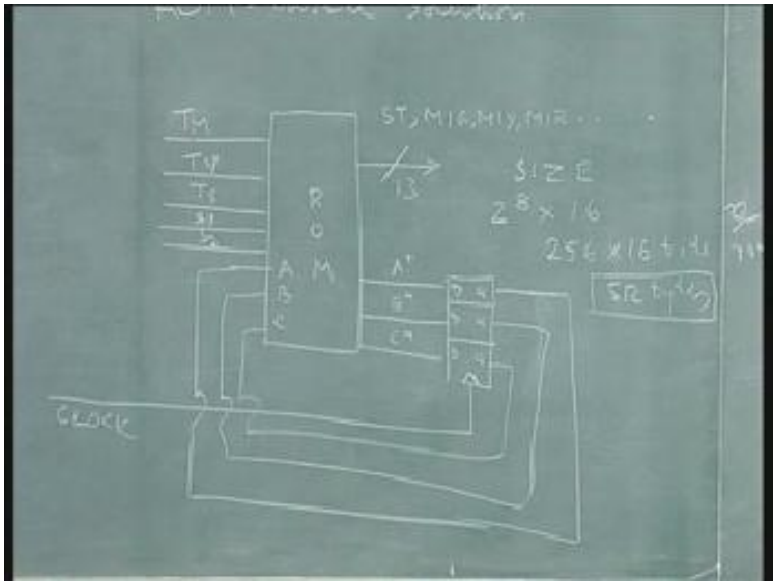
(Refer Slide Time: 45:05)



The hexadecimal ROM table will look like this. Address and content - for each address a corresponding content. Address will be 0 0, 0 1, 0 2, 0 F, 1 0, and so on until F F. What do we put in each of these rows? In the first table, for example, first is 3 and something – we write it as 3 times x and continue. This is not x - do not put x in your table. We are writing x, because we have not fully written all the rows. Instead, we can do it and make sure that the way in which it is appearing; it will become ST, M1G, M2G, MTG, SG and then we will repeat it for yellow and red. If we take this case, the condition here will be - Main will be Green, M2G will be Green, MTG will be 0, SG will be 0 and then, in the case of Yellow it will be 0 0 0 0 and in the case of Red, it will be 0 0 1 1.

The first word will be - the first 8 which is 3C and the next is 03. The first word is 3C03 and it is repeated for the first 32 entries. You can go to the next step and have a Programmable Array Logic (PAL). We will not go over all of this again, because we know the difference already - we have discussed the difference between PAL and the ROM approach. PAL approach will have a PAL table, which is a simplified form of a ROM table, where the redundant entries and entries with no activity will be removed. PAL will have fewer rows or fewer product terms. Another advantage is that the registers are included in the PAL - we can have a PAL or a PLA, usually PAL, in which register is part of the device.

(Refer Slide Time: 46:00)



In a single device, all we need is to look for a device with 5 plus 3 equals 8 inputs. We have seen the design of PAL earlier, so we are not going to spend time on it. In a PAL based design, there are 8 inputs and 16 outputs, out of which at least 3 should be registered outputs, which are required.

Of course, there should be a clock input as well. We should have sufficient number of product terms, sufficient number of AND gates for each output to fit the design. We will leave it here, as there are standard things like AND gates which we are not going to design. We are going to look through the library of the manufacturer and then decide the

appropriate device for this purpose. We have already seen PAL based design in earlier examples and those will be used here.

In summary, we now have an elegant method of drawing a chart and an algorithm based on specifications, identifying the functional blocks, identifying the signal, drawing the ASM, implementing it in various forms like gate form or you can write Karnaugh maps or using number of gates or multiplexer based form or ROM based or PAL based. With this, we will stop today's lecture here and we will see one more example of a slightly different nature in the next lecture.