Computer Organization
Part – I

Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology

Lecture – 10

Controller Design: Micro programmed and hard wired (contd)

Continuing with whatever we saw in the previous lectures, we are going to discuss or continue to discuss the hardwired logic design.

(Refer Slide Time 00:01:17)

Master-Slave Flip-flop

- Hardwired logic design

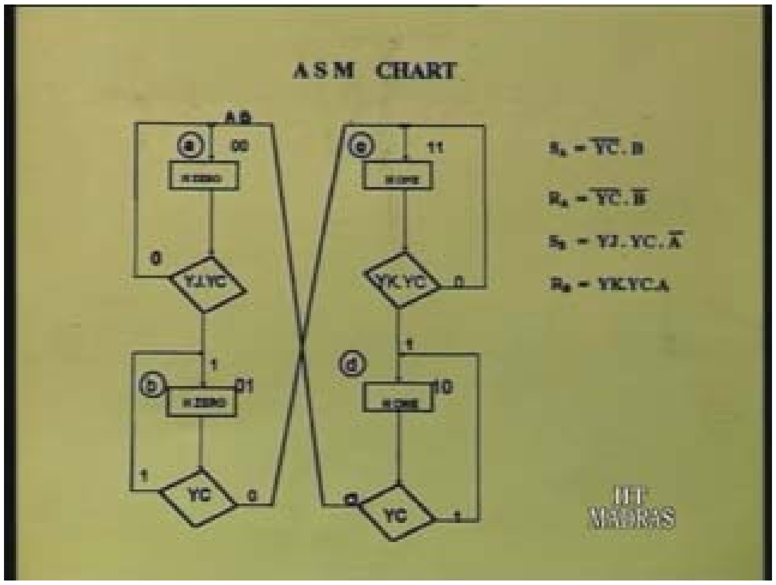
Firmware approach

Earlier we looked at what the firmware approach was; that is a ROM-based design. We have taken master–slave flip-flop as one simple example for which we are designing these, based on these two approaches, that is, the hardware approach as well as the firmware approach, just to see how a controller works. Instead of considering all the states in a computing system, specifically a processor based system, and then seeing the various states and so on, we are just taking a simple example. So let us remember that always.

Now in the hardwired logic design in the previous lectures, we came up to the point of arriving at the Boolean equation for the Boolean functions for the two state variables. What are they? We found that in the implementation of master–slave flip-flop there were essentially four states and, in that particular one, it was accepting three inputs that is YJ, YC and YK. These were the three inputs and it was generating two outputs, namely, H ZERO and H ONE. In the process of this, accepting these three inputs and generating these two outputs, we found that the system goes through four states; that is more important. Essentially the user is concerned with these two only: that is, inputs and outputs; the user is not concerned with anything else.

For instance, the user is not concerned with state; it is actually the designer who introduces the states, mainly because with a state by state approach we will be able to design the system. That is important. Now the starting point for this was ASM chart; the same chart, based on which we also did our firmware design. Now let us take a look at the chart to recall whatever we had seen so far.

(Refer Slide Time 00:03:40)

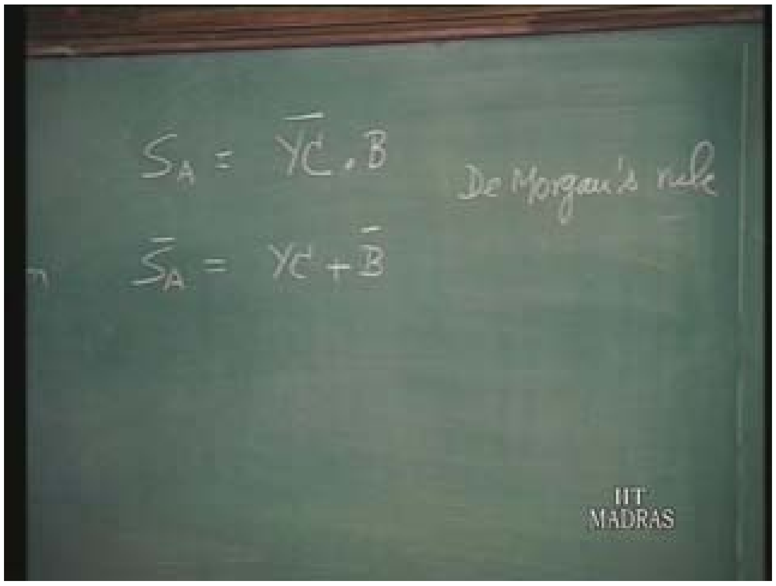


The particular system has four states, and we mark these as a, b, c and d. It generates two outputs: H ZERO and H ONE; so this H ZERO and H ONE and the inputs are there in the diamond boxes YJ, YC and YK. We discussed this in detail; now we found this particular one, this system or this; we may call it a machine. This machine is in one of the four states at any instant: a, b, c, d, which can be represented using two state variables: A and B.

These two state variables physically will be represented making use of two latches, and what did we do? We considered two SR latch. Strictly speaking, it is an S bar R bar latch, which we discussed earlier; we may call it loosely as SR latch also, and having considered or discussed the behavior of the system which consists essentially of four states, we have now reduced our problem to consideration of two state variables. That is, we have a whole machine and then we talked about state by state behavior of the machine, and then subsequently, by reducing it to state variables, now we need to consider only how the state variables A and B behave. Then subsequently, we derived the equations for SA and RA, that is, the set input for the state variable A and the reset input for the state variable A. For S_A and R_A we got this equation; similarly we also worked out for S_B and R_B. Now let us go back and since we decided that we will be using S bar R bar latch, now we have to derive the equations corresponding to these: that is S_A is given as – now we can refer to the chart – YC bar and B. So S_A is YC bar and B.

From this, we get the equation that S_A bar – that is, we are actually applying the De Morgan's laws, which you would have learnt in the switching theory in project design course also – the inverse of it, the complement of this will be YC or B. What is this? Just to recall, invert this; that is, complement this, for which what we have to do is we have to complement each of the variables. So YC bar complemented will be YC; B complemented will be B bar; and instead of and you will have or. You get this based on De Morgan's laws, or De Morgan's rule, or whatever you call.

(Refer Slide Time 00:07:15)



$$S_A = \overline{YC} \cdot B$$

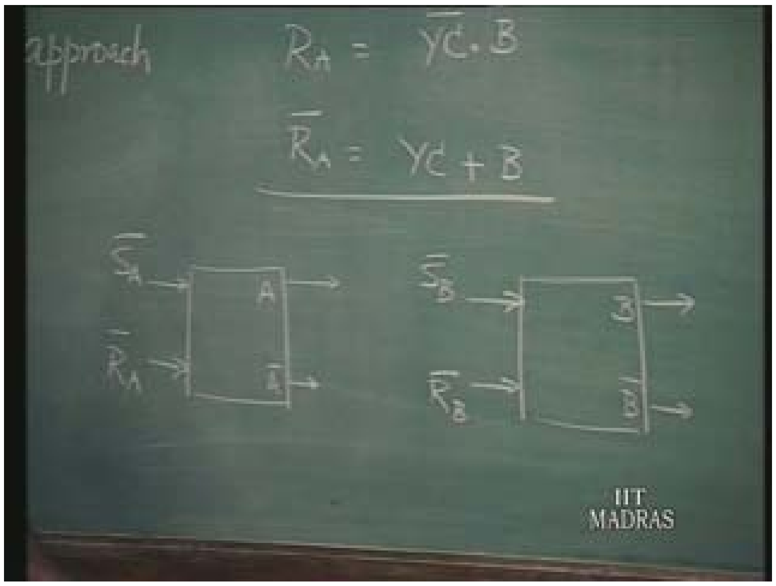
$$\overline{S_A} = YC + \overline{B}$$

De Morgan's rule

Now this is the one equation we get of S_A bar. Similarly, what is R_A? R_A is given – see the chart – R_A is given as YC bar and B bar, from which we get R_A bar as we complement this, which is YC; instead of and you have or instead of B bar, when you complement it, you get B. So that is what you have. For the state variable A, there is one SR latch. There is one memory element, which will develop our output A; the input for that is S_A bar and R_A bar – this particular one.

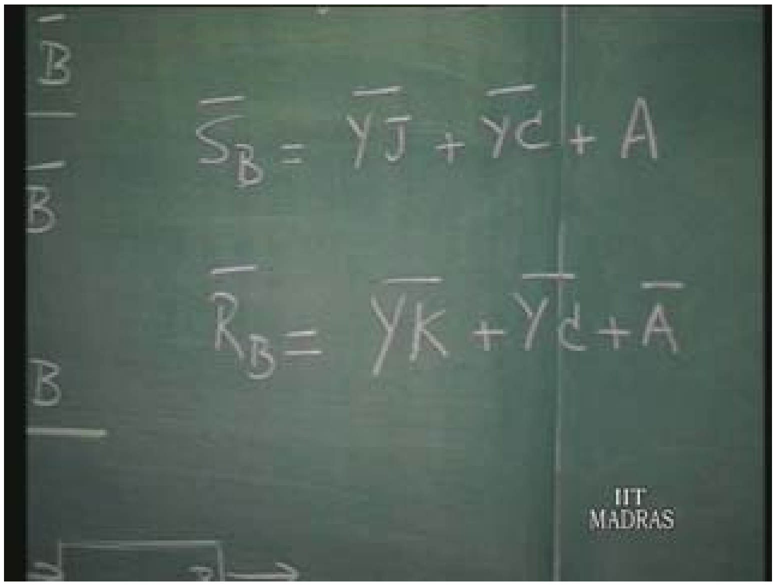
Similarly, we have another one for the memory element B. So we have to work out for that particular one also. We have both the outputs A and A bar, similarly B and B bar from these memory elements. Now the input to these is S_B bar or B bar. So we have two latches, that is, S_A R_A latch, and S_B R_B latch – two things which will represent the state variable respectively, A and B.

(Refer Slide Time 00:08:53)



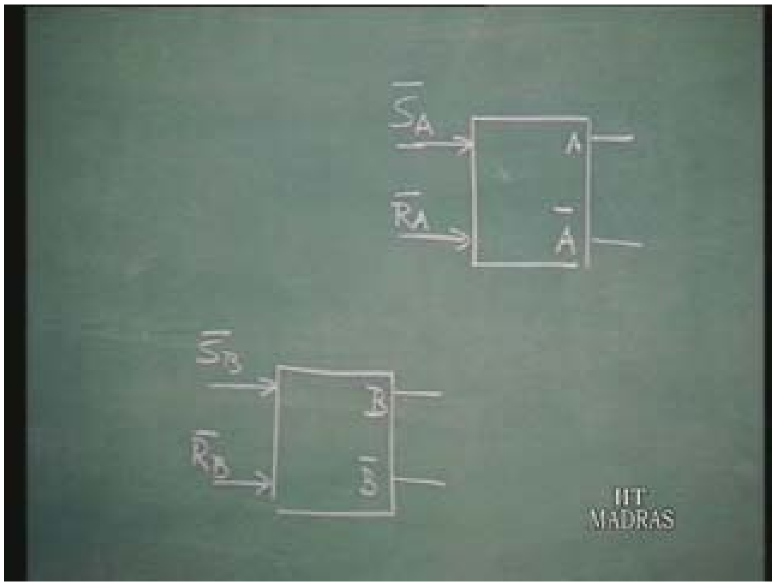
Now the output of these will be A and B; the input for this is what we are working out. So now we know the output–input relationship. Let us continue – just like we did for S_A bar and R_A bar, we have to do for S_B bar and R_B bar also. Now I would straightaway work out S_B bar after taking a look at the chart. S_B is YJ and YC and A bar. SJ is YJ and YC and A bar; so S_B bar will be YJ bar or YC bar or A, that is, YJ bar or YC bar or A.

(Refer Slide Time 00:10:35)


$\overline{S_B} = \overline{YJ} + \overline{YC} + A$
$\overline{R_B} = \overline{YK} + \overline{YC} + \overline{A}$
IIT MADRAS

Similarly, R_B bar would be – what is R_B? Say it is YK and YC and A; so R_B bar will be YK bar or YC bar or A bar. So now we have these four equations here: S_A bar and R_A bar and we have S_B bar and R_B bar. Now, making use of these two memory elements, let us try and draw the whole circuit diagram, that is, the logic circuit diagram. In this, let us take A, S_A bar, R_A bar and the other one, B. Actually we saw that B comes and B bar also comes; this we have discussed earlier – both A output and A bar output are available.

(Refer Slide Time 00:11:59)



Now how do we proceed implementing this? This, as you can see, is an or circuit. Now let us assume that we will make use of NAND gates and implement the whole thing. Now you must recall what is known as NAND circuit synthesis – how do you do that? May be some of you are familiar; I do not expect all of you to be familiar.

(Refer Slide Time 00:12:44)

→ NAND Gates

→ NAND circuit Synthesis

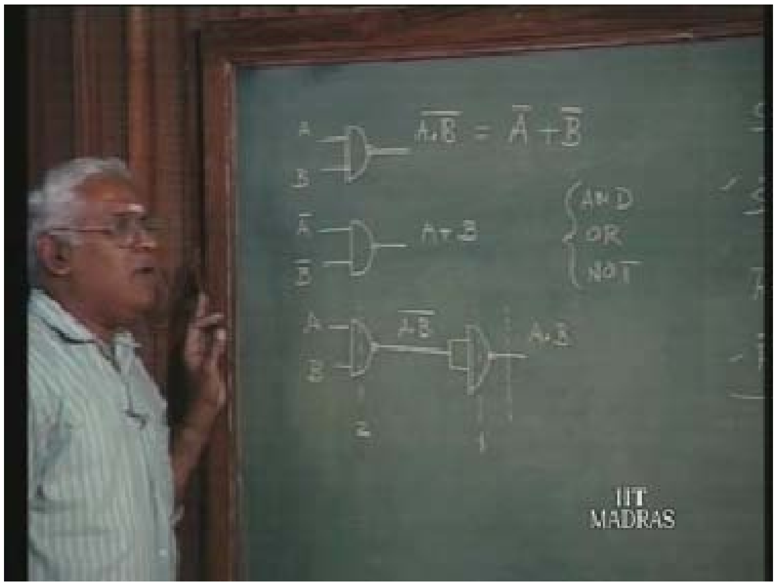
IIT MADRAS

Let us work out the NAND circuit synthesis and then implement this. The NAND circuit synthesizing is about synthesizing a logic system using NAND gates. That is, suppose you have only NAND gates, how would you synthesize the whole logic? We know that if A and B happen to be the inputs the NAND gate output will be A and B the whole bar, which again by De Morgan's rule or law is nothing but A bar or B bar.

Now this particular thing must suggest you something. What is that? Suppose you have only one NAND gate, A bar or B bar. Suppose you have only one NAND gate, what we find is the output of NAND gate is A bar or B bar, which means basically the NAND gate works like an or gate, with its inputs A and B complementary. In other words, suppose you have one NAND gate, if you give A bar with inputs and the other input is B bar, it means that basically A bar will be complemented, which will be A and B bar will be complemented, which is B. In other words, as this says, NAND gate output shows that if you have one NAND gate the one NAND gate works as an or gate, with its inputs complemented.

Now let us work out further. Take another NAND gate; we are just repeating this. Let us say A and B are the two inputs: the output will be A and B bar, that is, a usual NAND gate has A and B bar. Now you give this particular one to another NAND gate; let us assume another NAND gate with two inputs. What is that actually? Both are same. It is nothing but inverting the whole thing; so when inverted, the output will be A and B. You just inverted A and B bar there. There is no logic as such here because the inputs are shorted. It is just inverting it; so when you invert this, A and B bar becomes A and B, in other words, a NAND gate, which is from this output level, a NAND gate, which is two levels away, that particular NAND gate works like an and gate with its inputs true. Now you know that using and, or and not, you can synthesize all the logic functions that you need.

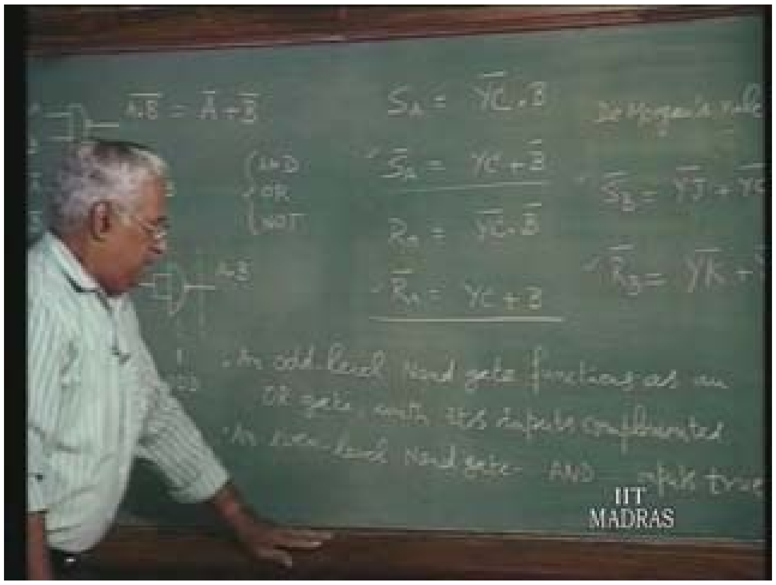
(Refer Slide Time 00:16:35)



Now we see that the NAND gate performs as an or gate or performs as an and gate, and when the input to the NAND gate is shorted, it just works like a not gate. So a NAND gate itself can perform as an and, or, or not, depending on what level it is in. Suppose you want a function with reference to that see at what level it is. Is it level one or level two or level three or level four? Without much discussion, I can just extend this: the level one as an odd level and level two as an even level. You can work it out as any odd level NAND gate. What we mean by that is NAND gates, which are one, three or five steps away from the output. That will work as an or gate. Now this is only one here so this is the odd one; this will work as an or gate. The ones which are two or four or six and so on, even levels away, they will all work as an and gate, but remember the corresponding inputs must be in the case of and, and it must be true; in the case of or, it must be complementary.

So now, let us sum this up. We can say that an odd level – I have just shown you for one level and two level, but we can work out and then see about the generalization of odd and even levels – so an odd level NAND gate functions as an or gate, with its inputs complemented. An even level NAND gate works as an AND gate, and its inputs must be true.

(Refer Slide Time 00:19:38)


$\overline{A \cdot B} = \bar{A} + \bar{B}$
AND
OR
NOT
$S_A = \overline{YC} \cdot B$
$\bar{S}_A = YC + \bar{B}$
$R_A = \overline{YC} \cdot \bar{B}$
$\bar{R}_A = YC + B$
De Morgan's rule
An odd level Nand gate functions as an OR gate with its inputs complemented
An even-level Nand gate - AND - inputs true
IIT MADRAS

That is, an odd level NAND gate works as an or gate with its inputs complemented; an even level NAND gate works like an and gate with its inputs true. Now this in fact is the crux of the NAND circuit synthesis. Now let us work out this particular one: what we have is an or gate. That is, S_A bar is YC or B bar; now we will work out just for S_A bar and then see. We will do this for S_A bar alone. What did we see? We want or function; we just need odd level, which means one gate will do. If you have one NAND gate, we want the output S_A bar, which is YC or B bar. What does it say? An odd level NAND gate works as an or gate with its inputs complemented.

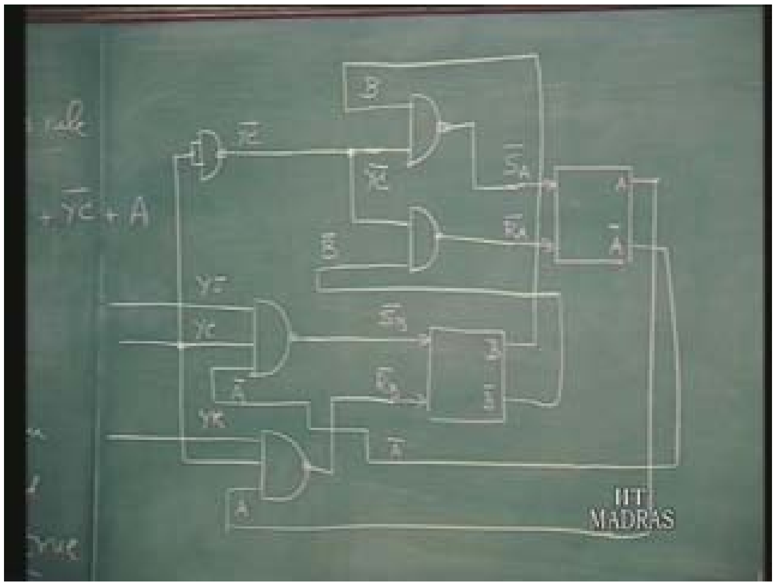
So if you want S_A bar as the output, which in fact is this, its inputs must be complemented, which means the input for this is YC bar and the other input, B bar, you complement it will be B. So just one NAND gate is enough to implement this function, that is, for the output, S_A bar, which is YC or B bar, the rule says to complement the inputs. So for YC, we complement and get YC bar; for B bar we complement, which means we recomplement B bar and get B. So this in fact is the output of the NAND gate, which we can include here. That is, for S_A bar, we just need one NAND gate and the inputs to this are YC bar. I will call this YC bar and the other one B. Similarly for R_A bar, again it is an or gate; so we need just one more NAND gate for R_A bar; the two inputs now we say are YC or B. These are the two inputs.

So again, the input must be complemented. So YC bar is the same as this; so we can give the same YC bar here too. YC bar and the other input also must be complemented. B means B bar; so you must have B bar here. Where do we have B and B bar? We know that you have got it from here. So we can take B bar from here, and we can take B from here. YC bar is one of the inputs; actually our one input will be YC; we have to complement that. Now continuing, let us go to the other thing, SB bar.

Again we have the or function, but this time with three inputs: we need or, which means we need a three input NAND gate and that again must be complemented. So for S_B bar, we will have three inputs, which must be complemented. The first one is YJ: when YJ bar is complemented, it will be YJ; YC bar complemented will be YC; A complemented will be A bar. Then R_B bar, which is or again, and has three inputs or gates. That is, for a three input NAND gate, one gate will do; so we will use a three input NAND gate and generate R_B bar. This input must be complemented. So instead of YK bar, complement and generate YK; then complement YC bar. It is YC, which is the same thing as what we have here, YC. The other input also must be complemented; you complement A bar and you get A.

Where do we have A and A bar? We already have them here. A will be here and we have A bar here. We will run this a bar. So now, look at this: we have YC bar here, we have YC here. And we know that YC is an input of the system, like YJ and YK. Now that we have done it, we do not need these functions anymore. YJ is one input; YC is one input; YK is the third input. Now from YC, you invert and you get a YC bar; it is the YC bar that we need. Suppose you happen to have a two input NAND gate; you just short that and from YC you get YC bar.

(Refer Slide Time 00:27:07)



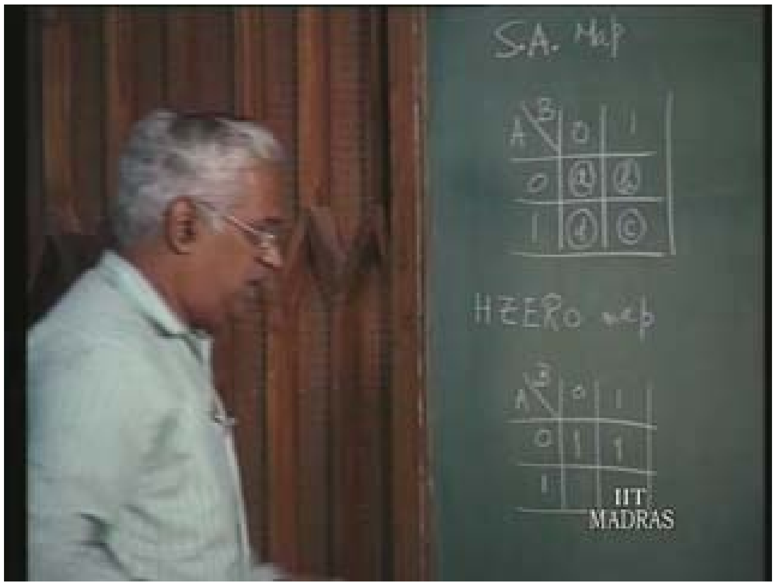
Now we know how exactly, using the three inputs, the two states can be arrived at. What we have here is only for the next state; this whole circuit diagram depicts the next state function. We have not yet worked out the output function. That is what we have to do next. The output will also instantly be derived, making use of the two state variables, A and B. We will do that – we have to refer to the ASM chart. Now we just see YJ, YK and YC are the three inputs I said, and the outputs are H ONE and H ZERO. So you can see that these are the three inputs from the external world, and what we have internally is our digital system or machine or component, whatever you call; and the output will have to be generated, which again will be a function of the state, which means A and B.

Let us go to the ASM chart. In the chart you can see that the two outputs, H ZERO and the other output H ONE are generated respectively in these states. H ZERO is generated once in state a and again in state b. Similarly H ONE is generated once in state c and again in state d. In fact we will be able to directly derive the function of H ZERO and H ONE. What is it? We just take a close look at it without going much into working out the details we can arrive at. What is it? See here – look at state of A. A is 0 here; A is 0 here also – I am talking about the state variable A.

Then state variable A is 1 here; A is 1 here also. So whenever the state variable A is 0, H ZERO is generated; whenever the state variable A is 1, H ONE is generated. This is alright; this is in the case of very simple ASM chart; that means a simple component. But let us work it out in a very formal way. What is it? Let us go to the state assignment map; we need to refer to that again, which I will just call state assignment map. What was that? We had the Karnaugh map with A and B, assuming values 0 and 1. Let us see the chart again and see that A corresponds to A and B being 0. B corresponds to A and B being 01. That is what it is – when A and B both are 0, it corresponds to state a. A 0 B 1 corresponds to state b; and A and B both 1 corresponds to state c; and A 1 B 0 corresponds to state d. This is our state assignment map, which we have done earlier.

Now, like whatever we did for S_A R_A functions, we do similarly for S_B R_B functions; that is as for as the input is concerned. Similarly, we have to work out for the output also. What is it? That is, if you take H ZERO, we have to generate the H ZERO map. What is that H ZERO map? This is an output map. It just says we have to see the ASM chart and work it out. Let us first prepare the map for it. Let us check the ASM chart – H zero is generated in state a and state b. H ZERO is generated in state a and in state b. So the H ZERO map will have the entry corresponding to state a and state b, which is 1.

(Refer Slide Time 00:31:49)



That means, for H ZERO, the particular Boolean function is 1 in state a, which means H ZERO is generated in state a and again it is generated in d. Similarly, we can have H ONE map again without much of difficulty. For A and B, we are assuming value 01. See the ASM chart again – H ONE is generated in state c; H ONE is generated in state c and again in state d. So the H ONE map will show that in state c and d, H ONE will be generated. Now let us work out the Boolean function of H ZERO, that is, encircle this. That gives you the function for H ZERO. According to this, what is H ZERO? It says whenever A is 0, H ZERO is generated; that is what that encirclement shows. That is, whenever A bar is true, that is, A is 0, or A is complemented, this is generated. What about H ONE? H ONE is generated whenever A is 1; so the function of H ONE is whenever A is 1, H ONE is generated. This is what I was telling you earlier. You can arrive at this very simply.

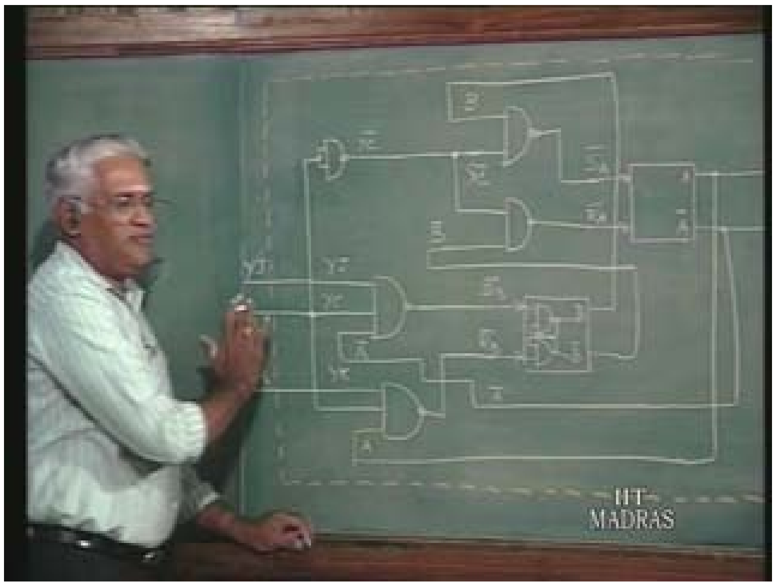
Now H ZERO is A bar and H ONE is A – as simple as that. What happened to our state variable B? It is not figuring in the output, and recall what we were talking about master–slave flip-flop. We said there is one memory element, which will follow the inputs; another memory element will be following the internal state and that is the one, which will generate the output. Now we find that in A the outputs H ZERO and H ONE depend on the state of a. In other words, let us just mark it on this. H ZERO is A bar, that is, the output, and H ONE is A. What happened to B? It is internal. Now we find that the output of this component depends on this element A; depending on this the output is generated, and depending on some combination of some state of B and some other combination, a state will also be arrived at. Now you can see that this element B in fact is the master part and A just follows B; that is the slave one. B as a master follows the inputs and internally prepares the states and decides something about the state of A. Depending on state of A the output follows immediately. So this, in fact, is the master part and that is the slave part, because the slave follows the master.

As you can see here, a look at the inputs a clock is there and then the state of B alone decides what the state of A is. So a follows B, and B follows the inputs. If you take a look these inputs, they will be following the inputs and A just follows B. So A is the slave and B is the master. So the master keeps an eye on the inputs, changes its states, and the slave follows the master. It is a very simple output function; nevertheless, even in the case of complex output functions, this is the way to proceed.

Now do you recall whatever we did on the ROM-based design? What we have here is the hardwired logic design. I will just put it as simply hardware design. Earlier we did a ROM based design and I said that is a ROM-based design and that particular one is the firmware design. So why did we do all these designs? Not to really design the flip-flop, but just to get an idea of the hardware design and the firmware design, and now we link this up with whatever we had done earlier. We had taken a look at the firmware design of a controller of a processor, CPU. Earlier, while discussing the data path control, we did the firmware design. Similarly, there is a way to do the hardware design, but the number of states, inputs and outputs in the controller of a CPU will be too many to discuss.

So we took up this example, and through this example, just know the relationship between a hardware design and a firmware design, and extend or extrapolate your idea about a firmware based control, which is nothing but a micro-programmed controller, and the hardware control. Now this is hardware because everything is wired; you have the two memory elements and again these two memory elements are two NAND gates. Remember they were connected back to back? I was working out like this. That is, essentially this will also consist of two NAND gates like this. Similarly here too there will be two NAND gates; this also is D bar. So the entire thing consists of NAND gates, nothing else. So it is possible to have any logic system or a digital system designed only with NAND gates and we have just seen it for this master–slave flip-flop completely. Now we have to extend for any other system with any number of inputs, outputs and states.

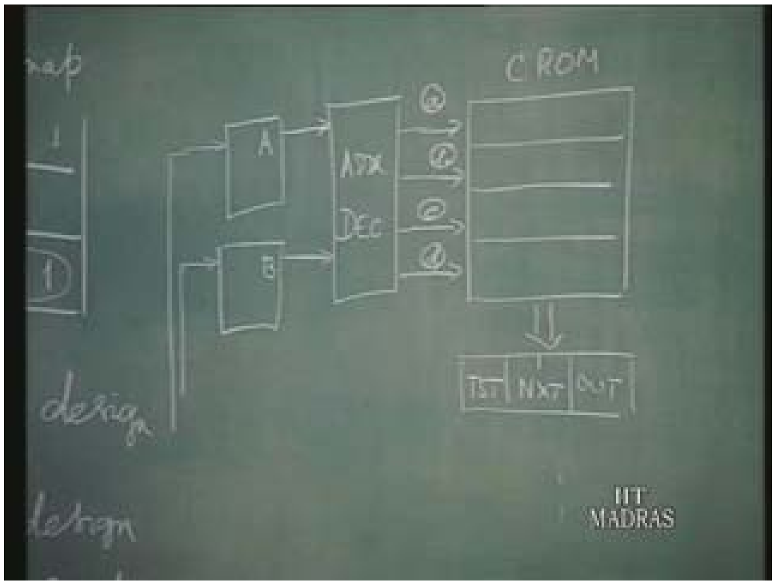
(Refer Slide Time 00:39:50)



What is that we had earlier in the firmware design? Let us take a quick look at the ASM chart and see again, because the ASM chart was the starting point for both hardware as well as firmware design; that is, the chart describes the behavior of the system and the system we see consists of four states; in this case, a, b, c, and d, and in each state, it checks for some input and for the output. Then, state by state, we saw that in state a – that is the state code for a – the system generates output H ZERO and it checks for inputs YJ and YC. As long as YJ and YC are 0, the next state is a itself; if YJ or YC becomes 1, the next state becomes b. This is what we did for one state, which is a. With a as present state the possible next state is a itself or b; the inputs to be checked are YJ and YC; and the output to be generated is H ZERO. So this is for one state; similarly, we do for the other three states. Essentially we had four such micro-words. These four micro-words we put in a read only memory and we got the firmware design. Now instead, here what we have is a hardwired logic. You remember in the read only memory we had four words, each for each state: one for a; one for b; one for state c; one for state d.

Then we had the decoder. That is nothing but a simple address decoder, that is, the read only memory. We have also said that its controller has read only memory and any time only one word is taken. That consists of three fields, that is, test field, next state field – there we are talking about two next states possible – and out field, and extra logic. Here we had two memory elements, A and B. These two states A and B will decide which one of these to use. We were getting input to A and B from here; we were having some test logic and so on and so forth.

(Refer Slide Time 00:43:26)

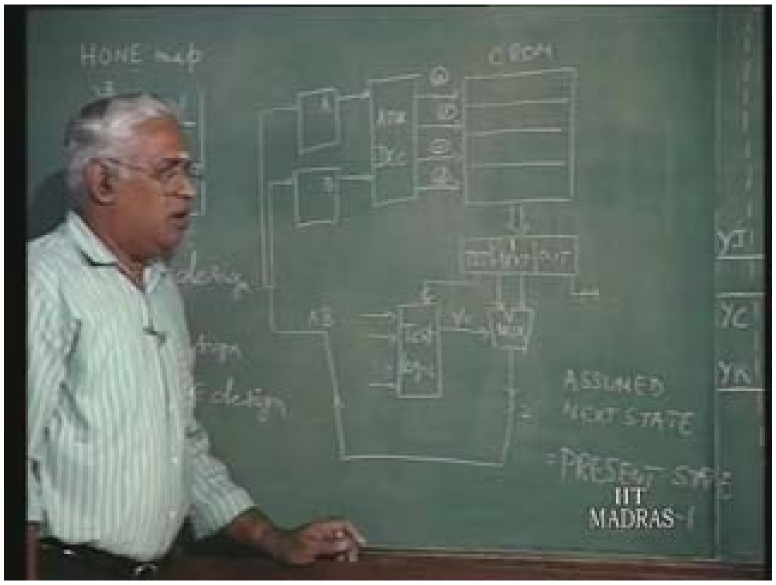


And if you see the ASM chart, we have just reduced the next state as only two next states; that is, we call one a false next state, the other a true next state. It is possible that in any practical problem, there can be more than two next states, but everything can be reduced to always only two next states. May be I will just work out here because there is a point I want to make here. We are talking about a true or a false next state; where one of these will be true next state, the other will be a false next state. One of these will be selected depending on some test logic and what exactly is to be tested depends on what you get from that word. Here are the actual inputs. One of these is passed on, and that is the one which forms these two codes – for A B, there will be only 2 bits.

So one of these will be selected because that, in fact, is a multiplexer and outputs will be generated. Now something can be assumed here. This field can be reduced instead of two next states; we can call one a true or false next state. One of these can always be assumed: what is it? That is, we can talk about either a true or false state; we can have, let us say, true. We can always have only true next state, and then we can always assume the next state is nothing but the present state plus one, in which case we need not have two fields; we can just have only one field. One of these can be removed. Let us take a look at the chart – suppose I am assuming this.

Now you can see here that for AB, the code is 0001. I am assuming the true next state is assumed, which means after a the next state will be a plus one, which will be 01 itself, true next state, in which case, I have to give only one next state field, which is, in this case, 00 itself.

(Refer Slide Time 00:46:17)



That is, I have to give only the false next state, so that a simple counter can be made use of here; so the assumed next state is the present state plus one. But we have to decide. Now as you can see the ASM chart, we have assumed for the true next state, in which case we can reduce this particular one also; that is what we call reducing the size of this micro-word. Now that is in fact the firmware approach and here we have the hardware approach.

Now as I said earlier, for both of these, ASM chart is the starting point; but here in the read only memory we have this entire state by state information. In the present state it says about the inputs to be tested; the possible next states; and the outputs to be generated. Starting from the same consideration, we had derived hardware also, is it not? The same thing, that is, ASM chart, was the starting point for both.

Now, in instead of this particular read only memory, which gives this information as per the ASM chart as we discussed earlier, you pull out this read only memory, put some other code, and put it back. Then, the behavior of this machine changes. As long as the memory remains the same, it is going to behave as per that ASM chart whatever we have discussed. Instead if you put some other code another one comes; so we talk about the firmware approach; that is, as long as we have the code, it is okay. If we change the code, it is going to behave differently. Compare that with the hardware part – if you want to change anything the entire connections will have to be changed.

So that in fact brings about the flexibility, that means, the firmware approach brings in a flexibility, and that in fact what we saw as the micro-programmed controller for the process. Now it is up to you to compare this with whatever we had discussed for the processor. In connection with processor, we discussed the controller –I took only one word in that particular one, which was giving the data path control. Now here, we have for the entire flip-flop.