

Digital System Design
Professor. Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture No. 60
Interacting state machines

Hello everybody, we are going to discuss you interacting state machines today. So, in the previous lectures of this module we have seen that how state machines can be designed, how they can be utilized for different kinds of applications. We have also seen how sequential design can be used for effective design of or more efficient design of arithmetic circuits, and in general different kinds of circuits. And we also seen how pipelining can be utilized or introduced to have efficient throughput.

Now, from now onwards next 3 or 4 lectures, we will try to discuss how can we design complex systems. So, in that relation the first step is that, if we have to design a complex system and let us say there are multiple state machines. How they should interact to each other or how we can design a system which has multiple of the state machines. Why these multiple state machines would be required or useful. So, that is what we are going to discuss in today's lecture.

(Refer Slide Time: 01:33)

Designing complex sequential systems

- Partitioning into sub-systems
- Define clear interface
- Sub-systems
 - Multiple state machines
 - Segregating data-path and control path
 - Design control path using state machines



So, now, this particular question that if we have to design a complex system, most of the complex systems are going to be sequential systems. So, I am writing complex sequential system. So, if we have to design a complex sequential system, how should we start? Usually, the major problem with the complex system is the detail and complexity. So, because of that

sometime designer get overwhelmed with the complexity, with the fine detail, which is there in any of the complex systems.

So, what when we say complex system let us consider your mobile phone your smartphone or any of the application which we see. Let us say, a smart camera or pan tilt zoom PTZ cameras. So, anything which are your D2H or your Smart TV, so, all the applications which we have discussed in our first lecture, anything which is a digital system overall the system will get complex. So, when we are designing such a complex system, what should be the approach?

First thing we have to do is, we have to divide and conquer. So, what does this mean that, let us divide the whole system into multiple subsystems and let us try to design individual subsystem. But, now, the question comes that if the subsystem has to be designed individually, but it is a part of a system that means it must be interacting with the other systems also other subsystems also how to take care of that.

So, whenever we have started designing these subsystems, we have to clearly define the interface, how two subsystems will interact with each other. If we have defined the interfaces clearly, then it would be easy to design each sub-system individually and then we can combine. So, what is the meaning of interface here? Interface would have different meaning in different kinds of system.

So, let us say if you are you are creating a huge software and then you would like to divide things into multiple sub-systems, then the interface is going to be API Application Programming Interface or basically, the function calls their parameters and their arguments list of arguments and their behavior. But in case of a hardware system or a digital system, the interface clearly means that how many inputs, how many outputs, what are the bit width bit width of each input and each output.

What is the functionality expected from each input and output or in other words, when any particular input would be 1 and when any particular input would be 0. So, also try to understand that when we are creating these subsystem interfaces, these interfaces are not visible at the system level. And usually, from the user end specification they will never define the interface of substance. They will only say that in the end, we would like to have this.

So, for example, if you want to have a mobile phone so in the mobile phone they just would like to say that these are the keypad and these are the few buttons, which you will have but

internally how are you going to design this mobile phone or hardware or mobile phone? It is all up to us. So, how these interfaces of sub-system is going to be it depends on the designer. So, and that that makes things easier, as well as complicated. Easier because it is there in our hand complicated because now they are linked with each other unless we know the functionality of each subsystem we will never able to define what is the interface?

So, the first exercise should be that clearly define what subsystems would be, and what would be this, what would be the functionality of each subsystem, what would we expected from this subsystem to from the outside world of the subsystem, and what the subsystem is going to deliver to the outside part of this subsystem.

So today, we will try to understand this concept of sub-system with the example of multiple state machine problems. So, let us say a particular system can be divided into multiple state machines and how those state machines could be segregated and how they could interact with each other. So, in in later lectures, we will see how data path and control path could be segregated? And how we can design the control path using state machines? And how data path can be controlled using these state machines? We will take couple of more examples for these sub-systembased or some relatively complex sequential systems.

(Refer Slide Time: 06:53)

Example

- Single input, single output
- Output is one when pattern “1101” is matched with in block of 256 bits
- Pattern could be overlapping, but should not cross block boundaries

Digital Logic Design Sequential system design

So, to understand this, let us take one problem. Now, this problem is very similar to our sequential matching, sequence matching problem or our pattern matching problems which we had done in previous few lectures. So, like those pattern matching problems, it is also a single input and single output. Single input means it is a serial input, which is coming clock cycle after clock cycle and there is aoutput every clock cycle.

Now, the pattern which you have to match matches 1101 and this pattern is it could be overlapping in nature. So, that we there could be a sequence of this 1101 and two patterns can overlap. Now, along with that, there is also one more constraint that we have to match in the block of 256 bits. So, what could a real life analogy with this 256 block? So, when, for example, when there is a serial interface like USB, and then there will be a packet.

Packet means, whenever these this sequence of bits are sent, these sequence of bits are sent within certain block. So, all the time the data size is going to be fixed. So, let us say that data says 256 bits, so that means this block does not interact with the other block. So, that means, whatever pattern would match at the boundary will not make sense. So, that is why we have to match it within the block of 256 bits only.

So, when this block will end, that means the pattern will not, has not matched. In other words pattern cannot cross the boundaries. So, if we have to design such a system, it is clearly seen from our experience of this pattern matching and (seek) sequential design. So, we see that we can do it using a state machine.

(Refer Slide Time: 08:56)

Single state machine solution

- Each state should remember which bit of the block it is!
- Each state should remember previous three bits
- Total number of states required $256 \times 4 = 1024$



So, if we had to do it using a state machine, again from our experience, it appears that we have to remember, we have to remember what are the previous 3 bits. So, let us say 1101 is the pattern, so that means, I should know that when none of the bit has mentioned when one bit has matched went two bitshas matched when three bits have matched. So, at least these 4 states has to be there.

And along with that, I also have to remember that how many bits has already elapsed? Or how many bits of the block we have already seen? So, because now number of bits number of bits in a block is 256 so there has to be some 256 states for all of these possibilities. Along with that now, I also have to remember that how many bits of the pattern is matched. So, you see, try to see this information. You will see that these two information is a disjoint piece of information.

So, one information is how many bits has elapsed how many which particular bit of this block it is? Has nothing to do with what pattern it has already matched. So, because of that, because of independent nature of these two pieces of information, total number of states in my single state machine is going to be 4 into 256. So, that means, each state would remember that how many how many information or basically how much bits of the pattern it has mentioned, which particular bit it is. So, total number of states are going to be 1024.

And also you can just look at this and try to understand try to see the complexity of the state machine. How many edges would be there? And who will, when whenever who would design this this particular form it would not be possible onto paper and using paper and pen, we would require some sophisticated software, which would probably automatically generate that which particular state should go next and which particular state should be the new one. And and when the input is there, so based on that input which particular state out of that 1024, which state it has to go.

So, designing itself is a challenge and representing and then (implemented) implementing it using flip flops is whole together a sophisticated issue. So, but the good thing about this particular problem is that these two pieces of information is disjoint. So, can we have two different state machines? One state machine which remember how many how many bits has already passed or the current bit is which particular bit of my block. The second state machine just try to see that which how many bits has already matched. So, these two state machines could be a separate a different state machines.

(Refer Slide Time: 12:21)

Multiple state machines

- State machine M1
 - That remembers the number of bits received. If number of bits is 256th it sends signal to M2
- State machine M2
 - Remembers the pattern of bits

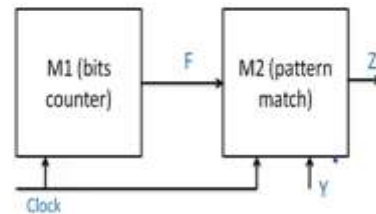


Now, let us look at let us try to say that this the first state machine, which is just counting how many bits has been received, let us call it M1 and M2 is the bits, which is remembering the pattern of the bits, how many bits has already matched. So, now what would be the interaction between these two state machines?

In this case, this M1 state machine which is saying that how many bits has been received, when the last bit or the two fifty sixth bit has been received or is received so at that time, it has to send a signal to M2. It has to tell the other state machine that, yes, this is the last bit, go ahead and do whatever you wish. So, this information is sufficient for the second state machine to act upon and to work in such a way that B is the final problem whatever is given can be solved. So, let us draw it using a diagram.

(Refer Slide Time: 13:37)

State machine - interactions

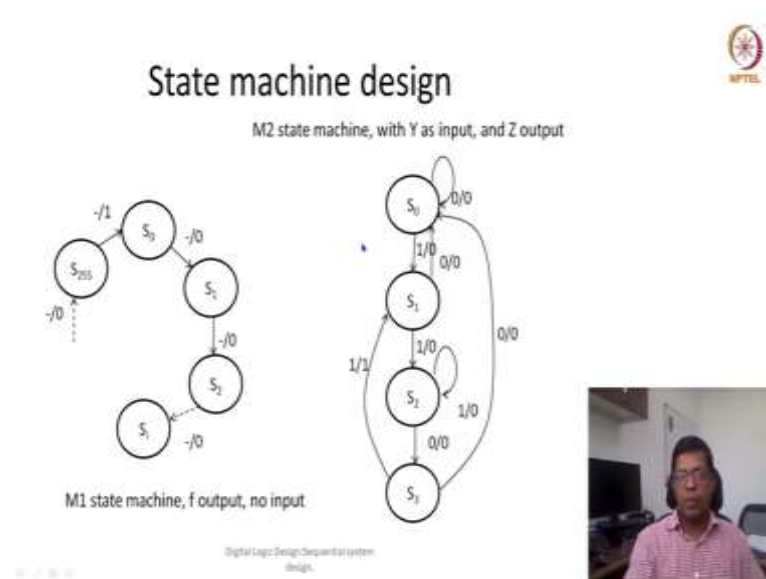


So, when we will say that this M1 is a, these are the two state machines this is M1 and M2. M1 is counting the bits. So, how many bits has already been received. And M2 is a pattern matching, so it is trying to match the pattern. Now, both of them are given clock signal. This is the first one, because it is just trying to remember that how many bits has already come. And because we are assuming here that every cycle there will be a new information so it does not require to know that whether the information is 1 or 0, so, the only input is clock.

So, at every edge of the clock, it can say that which particular bit it is, so it will keep on counting so that this can be implemented using a counter. While the second state machine will have an input, which is which will say that when this this counter is full or when the it has reached the total threshold of the size of the block that means 256 and then it will raise a signal which is F. Then this F signal would be 1 otherwise it is going to be 0. The other input of this pattern matcher would be the original input Y and the output is going to be Z.

So, as said initially in this lecture, so, this F is not visible to the outside system. To the outside system only Y is the input and Z is the output along with this being a clocked system or a synchronous system. So, now, when we have to design so, we can design these M1 and M2 individually. So, let us try to see how can we design.

(Refer Slide Time: 15:34)



So, the M1 is straightforward that because it does not have any input and only F is the output so, what it can we can do is we can simply make it a state machine which is a counter S₀ to S₁ then S₂ then S₃ then S₄ up to S₂₅₅ and in all other cases output is going to be 0 but only when state is the final 255 states then the output is going to be 1. So, that means all the inputs has already received.

So, now, when even we try to design it using Moor's law, Moore machine or, this is essentially a Moore machine because there is no input and output depends on which state it is. So, this can be represented using Moore machine also, but here, the representation is a Mealy machine. But ideally it is a Moore machine. So, just the matter of representation. So, even if we use the method of implementing using our flip flop and using next state equations it will turn out to be a counter.

Now, what about the second machine? So, the second one actually, will take so is going to take F as well as Y as an input and Z as output. So, to make it simple, what I am trying to do first is that, let us ignore the F, let us take a Y as a input in Z as a output. So, when we take Y as an important Z as output. So, it is a traditional state machine, which is trying to identify a pattern 1101, and which is overlapping in nature. So, by doing this, I am trying to segregate the effect because of this M1 state machine onto my M2 state machine.

So, just try to remember the pattern is 1101. So, from S₀ state, which is my initial state, if 1 is there, then I will move to S₁ state output is 0, when another 1 will come I will move to S₂ state output is 0, then 0 will come I will move to S₃ state, and then another one will come,

then I have to move to a S1 state because at least one of the 1 has been identified. So, output is 1. So, output, so this way, we are able to identify our pattern 1101.

So, now this pattern has been there so has been detected. So again, using the way we were designing the state machine, so let us try to complete the state machine means try to identify that what would happen if the input is other inputs are there. So, at S0 if 0 is there, because it is already a reset state if 0 is there it will again go back to S0 state.

At S1, S1 actually represent that first bit of the pattern 1 has been matched. So, that means another one will come it will go here, but if 0 will come then it has to go to S0 state. So, it has to go to 0 state with the output 0. So, in case of S2 if 0 will come it is going to S3 state, but if 1 is coming then it has matched at least two bits 11. So, that means it will still remain at S2 state, output is going to be 0.

In case of S3 if we have 0 as a input or 1 as a input, then we are going to S1 state if 0 is the input so that means it will become 1100. So, that means we will have to again go back to S0 state that means initial state with the output 0. So, this exercise we have done multiple times for the other state machine. So, this is the way we will do this also. Now the next thing is that we have to consider if this F, which is generated by M1 state machine that also is given as input here.

(Refer Slide Time: 19:51)

State machine design

M2 state machine, with Y and F as input, and Z output

If F is '1', next state will be S0, irrespective of previous state and value of Y

If F is '1', present state is S3, if Y is 1, Output Z = 1

If F is '0', state machine will work like State machine without F

Digital Logic Design: Sequential system design

So, if that is given as a input, what would be the effect? So, first thing is that because first thing is that F is 1, so, that means I have reached 255 states. So, that means it is the end of the block. So, that means, whatever is my present state it does not matter. Whatever is my present

state I should write here present state. So, irrespective of the present state whatever is the present state whether it is S0, S1, S2, S3 next state is always have to be 0 because that would be the starting the next block. Because it is starting the next block so, that means we have to start afresh.

So, and also whatever is the value of Y, Y could be 0, Y could be 1, but the next state is always going to be S0 if F is 1. So, that is one thing. The other thing is that if F is 1 and the present state is S3 also Y is 1 then output could be 1, but the next state is going to be S0. So, there could be different outputs, but the next state is going to be always S0 if F is. But if F is not 1, then the state machine will work like a regular state machine whatever we have decided or whatever we have designed in our previous slide. So, that means it would look like a regular state machine even without F.

So, with this kind of a discussion now, we can redraw our state machine, and see how will it look like. So, now instead of we will start with again S0 state, now, there are two inputs. Remember, one input is F the other input is Y, and we I am writing it like F first and then Y. So, that means if my input is 0, 0 corresponds to F and 1 corresponds to Y. So, if it is 01, then the next set is going to be S1, if it is 01 then again it is going to be S2 and if it is 00 then S3. And from S3 if it is 01 then it is S1 with the output of 1.

So, this particular combination forms the state when the output is going to be 1. Now, let us try to see what are the other possibilities? What are the let us try to complete the state machine. Now, also try to remember now since there are two inputs, Y and F. So, at every state we have to see what would be the possibilities for all the input, all the four input combinations 000111 and 10.

So, from S0 we have already seen that what would happen when 01 is there, what would happen if 00 is there 10 is there and 11 is there as a input. So, as these would be our guiding words. So, if F is 1, it does not matter, that what is the value of what is the value of Y, the output state is going to be S0. So, we can say if it is 1 and independent of value of Y, it is going to be this.

So, the other possibility is this that if it is 00, so that means my F value is 0, then 00 would also mean that it is we are going to stay there in S0 state. So, when we are seeing 00 or when we are seeing X0 so X0 would have both the states 00 as well as 10, 10 is there in this this equation this expression also, but yes, it makes our life simpler. So, which means that we can

write this, the condition as $F + Y'$. So, $F + Y'$ if the condition is $F + Y'$ then we can say that it is going to be there in a S_0 state.

So, let us see this $F + S + Y + F_0$ state is there in S_1 also. So, let us see. Again in both these cases, it is going to be there in the S_0 state. So in, so we have we have seen this particular condition that in case F is 1 irrespective of if it is 1 irrespective of value of Y it is going to be the S_0 state and this represents 00 scenario that we F is 0 and Y is also 0, then it is going to be in S_0 state. So, let us look at the S_2 scenario.

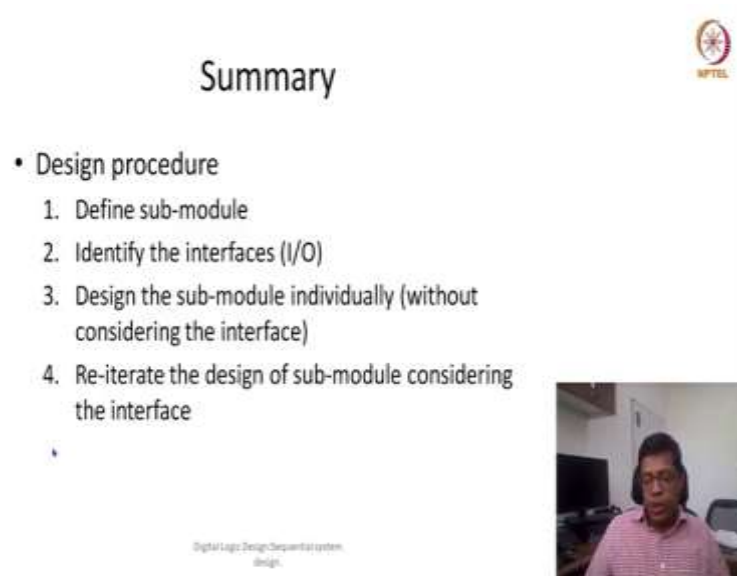
In S_2 scenario we have to see we have already seen that 00 will take it to the S_3 state; 01 will take you to the S_2 state, but if there is 1, F is equal to 1 and irrespective of value of Y , it is going to the S_0 state. So, that means there has to be an arrow from S_2 to S_0 . When F is 1 and Y is irrespective Y is do not care, output is 0. So, what about S_3 ? So, S_3 case is little interesting because, here, if my although my F could be 1, but if my input is 1, then it can go to S_0 state with the output of 1. So, if F is 1, but input Y is also 1, then the output is going to be 1, but it is going to be, going to the S_0 state.

If, Y is 0, so if F is 0, then if F is 0, that means it is a do not care, then input is 0. Input is 0, but it does not depend on the F , but it is always going to be the output is going to be 0. So, this is how we have to design this state machine. So, since we had already designed the state machine without considering F , it was a little easier in this case to redesign that and to rethink about the scenarios what would be the impact of F and then redesigning the interaction.

So, this is how, we would design if there are multiple state machines or multiple sub modules. So, in summary so, before the summary. So, essentially, what kind of advantage we have we have got by designing these two state machines independently. First of all, the total number of states has been reduced from 1024 states we are reduced it to 256 plus 4 states. Along with that, the design of 256 states was easier straightforward, because we were, we have designed using counter and design of this sequence detector was easier, because the first design of sequence detector was independent of F .

So, and based on that we have inferred couple of rules. And those rules when we implied on our state machine, then we could design the effective state machines with both the interface input as well as the original input. So, overall design was easier first of all, efficient in terms of area and in terms of designing also it was much easier than designing it using a single state machine.

(Refer Slide Time: 28:08)



The slide is titled "Summary" and features the NPTEL logo in the top right corner. It contains a bulleted list under the heading "Design procedure" with four numbered steps. A small video inset in the bottom right shows a man speaking. At the bottom center, there is a small text label: "Digital Logic Design: Sequential system design."

- Design procedure
 1. Define sub-module
 2. Identify the interfaces (I/O)
 3. Design the sub-module individually (without considering the interface)
 4. Re-iterate the design of sub-module considering the interface

So, the overall summary is that whenever we are designing any complex system, if we define them into sub-system, the key in defining sub-system is any two parts which seems to be independent, which has very less interaction can or which has very definite well defined interaction can form two different sub modules.

So, because they have well defined interaction, so that well defined interaction can be formed using interfaces. We can clearly define what are going to be the inputs what are going to be the outputs of each sub module, and how they would interact with each other? What could be the behavior semantics? Or semantics means that whenever input is 1 what is its meaning and whenever input is 0, what does it mean?

So, that interface semantics and behavior if we identify then we can design all these sub modules individually. So, this particular information, sometimes it helps sometimes it does not help, but in our case, which we example which we took, it was quite effective that we can design this sub module without considering the interface.

So, once it is designed, so that means we know, if it is a standard kind of a design, then we can design it without considering interface. And after that, considering interface, we can reiterate the design process. So, because our first level of design without an interface is already there, so adding the interface part of signals could be easier. So, that is why it can help many a times.

So, this is end of this particular topic. So, let us take try to do couple of tutorial problems related to this kind of interactions. And we will see you in the next lecture. Thank you very much.