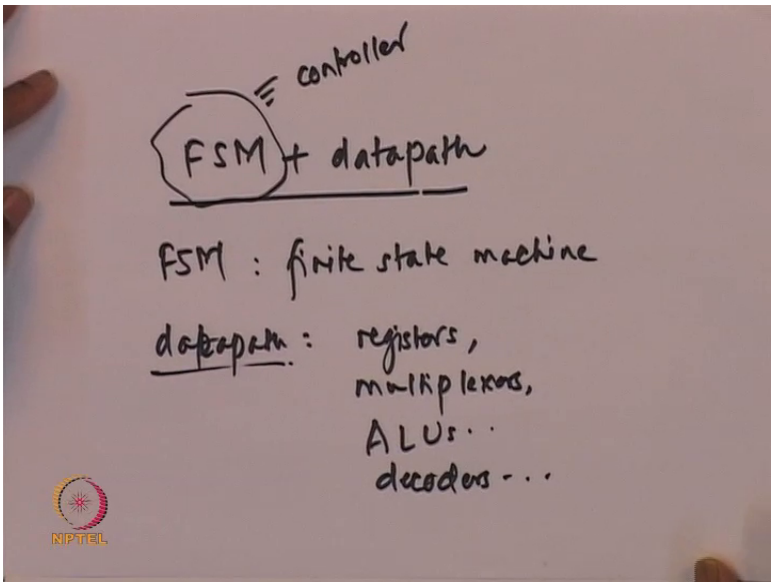
Advanced VLSI Design
Prof. Sachin Patkar
Department of Electrical Engineering
Indian Institute of Technology – Bombay

Lecture - 24
FSM + Datapath (GCD Example)

Welcome to this lecture on FSM + datapath, so we will try to illustrate some of the concepts, the design, the thought process behind the design implementation by through a couple of examples.

(Refer Slide Time: 00:46)

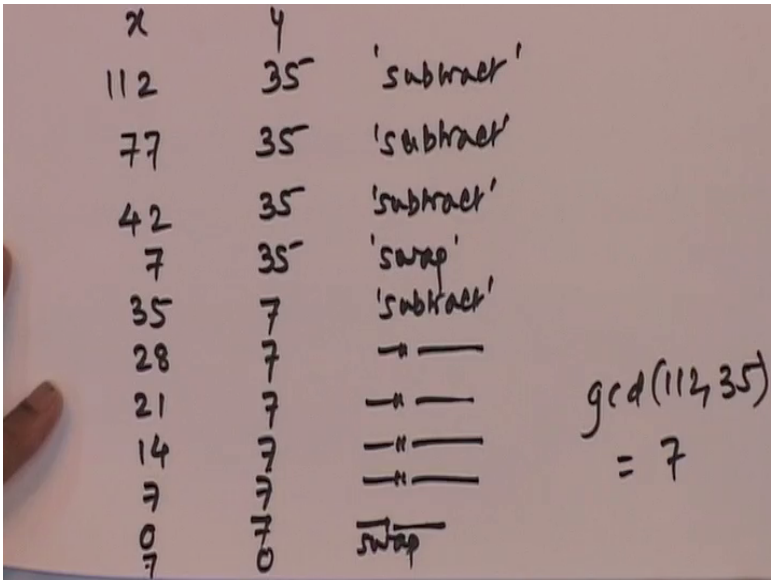


So FSM + datapath, FSM stands for recall finite state machine and datapath is a collection of components which can store data, which can move data, route the data which can operate on the data, so datapath will consist of registers, multiplexers, ALUs extra decoders and so on so forth, okay.

So this some of the components on the datapath like registers, multiplexers specifically multiplexers have control signals based on which the routing gets decided that is the configuration of datapath is achieved by controlling some most of this particular components on datapath and the job of creating generating this control signal appropriate values is of the finite state machines.

So finite state machine is going to work as a controller of the datapath, okay. So finite state machine in the first courses on these digital systems most of you know familiarity with finite state machines, so he here the finite state machine will be studied in the context of datapath how it controls the datapath and the typical like examples of custom computing machines or general purpose computing machines will be illustrated in the course of this lectures.

(Refer Slide Time: 02:43)



So let me focus mostly on the example very well-known example that of GCD greatest common divisor let us for the sake of simplicity of the presentation we will I will focus I will assume that we are given two numbers x and y both non-negative let us say we are given the numbers 112 and number say 35 and we want to compute a gcd of this a using a synchronous sequential system and that synchronous sequential system will be designed using the methodology or notion of FSM and datapath.

Let us see the thought typical thought process which one comes first and how does the rest follow, so let us first recall the gcd example so here we have this two non-negative numbers stored in x and y, loaded into x and y at the beginning of the computation, then let us see what would be happening at every clock cycle, so before I write down the pseudocode let us go through this standard like you know standard computation which most of you will familiar with.

So noticing that x is because then why we can do this so called modulo operation but instead of doing the modulo operation which what we do in case of write when we write a program with the help of powerful libraries C program with the help of C libraries or whatever which you have reminder modular arithmetic I mean remainder function available instead here we will take a more simple approach of repeated subtraction rather than doing the integer division and finding the remainder and the quotient, okay.

So here what we will do is we will subtract this y from x and the result will put it back in x so that gives us 77 which is 112 - 35 and 35 will remain over here, so what we did in this first clock cycle was so called a subtract like operation, okay. I just call it quote unquote subtract, the next clock cycle again we notice that x is > y so again I repeat this step of subtracting y from x which gives me 42 and 35.

So once again in this clock cycle I issue you the command to this synchronous machine subtract. I am deliberately using the words issuing a command that is what we will see the job of the controller would be to issue commands to the datapath which is actually doing the computation of the data as well as the movement of the data. So here we get 7 and 35 okay, 42 - 35 is 7 and 35 remains as it is in here, now x is now < y or x is !> y.

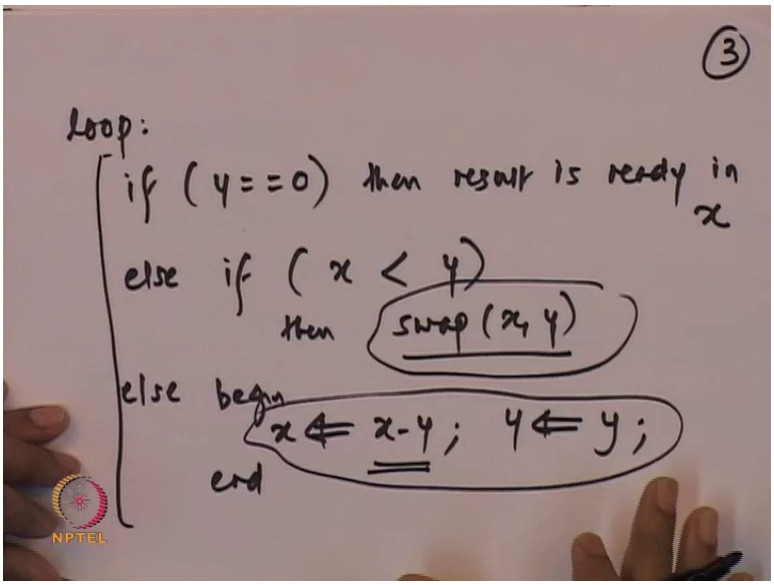
So we have to do this swap operation so that we get bigger number on this side smaller on that side and we can process again, so here at the result of swapping 35 comes here in to x and 7 is loaded into 7 moves to y, here now we are again ready back in the same familiar situation of being the subtract so we get 27 7 again subtract 21 7, 14 7 subtract again 7 7 now we have situation that now still okay x >= y so we will again subtract we get 0 7.

Now we see that x < y so we swap so here we say swap and as a result of swap we get 7 years and 0 over here, once we get 0 in the position in the register variable y then we stop the process sorry okay, so here is notice the last step so we were here say some time ago at the step 7 7, 7 resulted from 14 - 7, so 7 7 now we have I mean x is still >= y so we do not have x < y or is a situation for swapping so we subtract so 7 - 7 is 0 and 7 remains over here.

But now we have x < y so we swap 7 comes over here 0 comes over here, now any subtraction repeated subtraction is not going to help an anyway this is the end of this process and which is indicated by noticing 0 in the variable y, so whatever is in variable x will be the result of this computation which is the gcd, so gcd of 112, 35 is found out to be 7 computed to be 7.

So this is the algorithm a very simple algorithm which works in terms of using a very simple hardware which does comparison, which does a subtraction and which does swapping, movement of data, so we will realize the gcd algorithm by probably like you know wasting number of machine clock cycles doing in a very simple way, but using a very simple hardware which we can illustrate very easily and associated datapath, FSM concepts also.

(Refer Slide Time: 08:53)



So now let me abstract let me capture this what we just discussed in terms of pseudocode here, there is a loop some kind of iterative computation and in every loop we check if we already have a result or not that is if y happens to be 0 then we know that result is ready, then we realize that result is ready where in x okay. Else if the result is not ready that if y != 0 then we check whether this need to swap or not whether x is < y which will make it like you know which will induce us to swap.

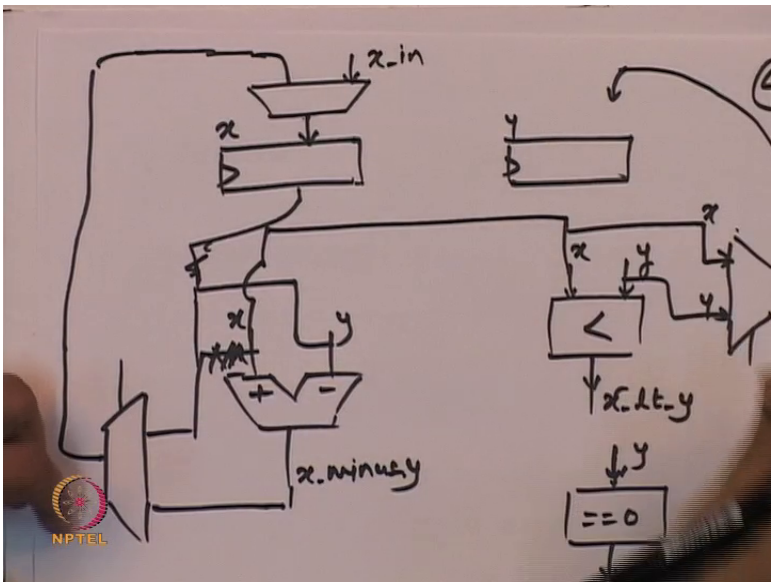
So if x happens to be < y then we will swap x and y okay, else -else what we will do is we will do a subtract operation, the subtract operation will have this result at in the x in the location

storage of x we will load x - y the result of the subtraction and y will be updated with y will not be updated y will remain as it is, x will be updated with the result of subtraction x - y and y will be as it is.

So this is the subtract operation this is very swap and this we keep on doing in the loop, okay. So do not think of it as a C program sequential program it is a description with say like you know using if and else involving condition checks and also involving some operations like swap x and y and this subtraction operation and then moment of loading certain variables with appropriate results.

So from this it will become fairly clear will be that datapath what will be the components on the datapath to support this kind of algorithm well, in every clock cycle we will have every typical clock cycle of computation we will have all these this logic being implemented, so what do we need on the datapath we will need obviously to store variables, we require storage locations, flip-flops, we are going to we have been focusing assuming synchronous computing, so edge triggered flip-flops.

(Refer Slide Time: 11:34)



So let us say this register called x and this registers for y this indicates their edge triggered registers okay, what else do we need, we need some subtractor we will require a subtractor which will subtract y from x, so this is there it is a specialized ALU which is only used for configure for

subtraction, so this signal is say I call it x - y that this is the name I give it because I know it is going to carry the value x - y.

We also require comparator less than comparator to which we are going to have filled in the input x and input y, if x < y we will have a true high value by Boolean value 1 here or Boolean value 0 if it is not a case, so this I denote this signal output signal by x lt y, x I read it as x < y this is the 0 1 signal these are two data items two integers, what else has to be require, we require an equality checker with 0 checker whether this y input to this equality checker is 0 or not.

So again this is a Boolean output I denoted by y eq 0, y eq 0 plus as I mentioned other than registers and ALUs comparators and so on, if you also required multiplexers because datapath is basically a collection of these components, we know where through which in which data is stored or data is operated upon or data is routed, so one important kind of component at we are still missing over here is multiplexer.

Multiplexers are almost invariably there in every kind of datapath okay, omnipresent things other than registers the multiplexers will be more or less always there, so what kind of multiple where do we require multiplexers we notice that like there is in different situations either we in depending on situation either we swap contents of x and y or we load x with x - y and we let y remain with as it is, okay.

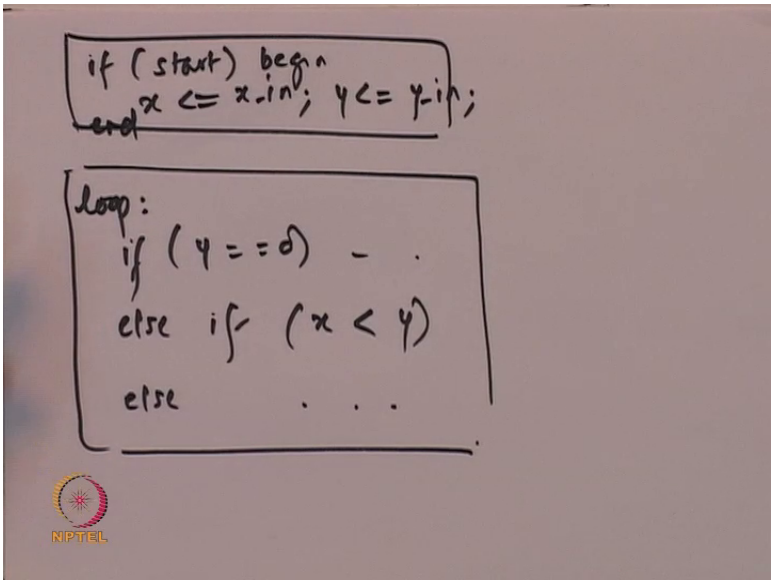
So we will require a multiplexer to which one of the input is x itself and other input is x - y and this the output of this multiplexer will be available to be loaded into x eventually, I am deliberately still not completing this path because we will realize that we will need something more for the algorithm to start, okay. So there we will require a multiplexer which based on a select signal will either route x.

So will either route x - y or in case swap is required in route y okay, so just like we will have a more clear picture of this but just we realize the need that x might have to be load depending on different conditions x would have to be loaded with either with y in case of swap or with x – y. So we require a multiplexer which will who is output will be routed eventually to register x,

similarly for y we will require a multiplexer which will either route x or y towards by register again.

So some multiplexers are required, let us see a couple of things that we have missed out so far.

(Refer Slide Time: 16:33)

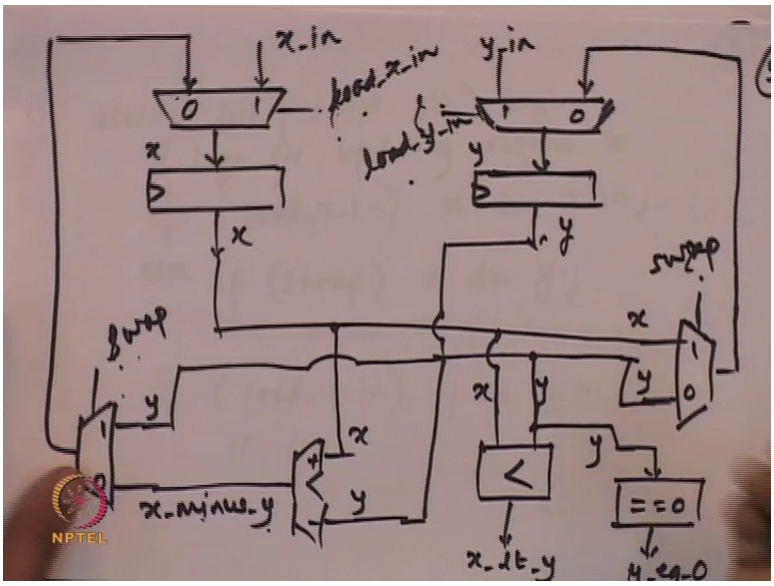


Let us go back to the we just basically I just showed you the main loop of this computation which has something like if y = 0 then something else if x < y then swap else subtract and so on, but before we once settles down into the loop and input has to be loaded into the registers right, so we will have in certain state this system will be doing this computation in every clock cycle, okay.

But before that there will be some kind of state where in we say like supposing we notice a signal called start then we load into x an input meant for it and load into y an input meant for y okay, so this is something extra that we are going to require because of which we will need a multiplexer over here to which one of the inputs is the input from outside x in or the other input is this either y in case of swap or x or x - y in some the other situations.

So we require and output of this multiplexer will be ready as the input to this x register okay, and this x is sorry x is going over here and some other places, let me draw it neatly again.

(Refer Slide Time: 18:29)



We have fair idea of how many components would require so I think it will help me to draw it more neatly, this is x, x is driven by output of this multiplexer where one of the input is something that is coming from outside and the other input is as we said it is the result of the multiplexer output of the multiplexer to which one of the input is x - y and the other input is y in case of swap, okay.

Similarly, for y we will have multiplexer driving the register y are like at the input of register y, one of the choices input choices to the multiplexer will be y in which is the input coming from outside, the other one will be the output of a multiplexer would be either in case of swap y is to be loaded with x or it should remain as it is, okay.

So this is my x signal, this is my y signal okay, this here if I allows this bit of clutter I can draw this y is y carrying the signal from here over here this x from going from here to here, y going from here to here and so on but like this is clear enough, x minus generated by an ALU where the input at the + terminal is x and input at the - terminal is y, x is coming from here, y is coming from here.

We also mentioned that route we will need a comparator less than to which the inputs are x and y and output is x lt y and an equality checker whether something is 0 or not to which a input is y and output are denoted by y eq 0, so let us route is, so this is x goes to all these places, y that

seems to be more or less the datapath but like I mean this does not describe everything completely.

Because we are not described what will be the control signals to this multiplexers that is what will be the select signal based on whose value whether this either this or this gets routed, similarly the select signal to this multiplexer this is what we yet we have we are yet to describe a multiplexer here this we know this ALU is going to do subtraction so there is no control to be specified here. We assume that x and y registers are going to be ready for loading all the time in every clock cycle so no further enable signal is required here.

In general, we do use it but in this example will turn out that we do not need it okay, so let me start like you know fixing some more details so let us identify this multiplexer inputs by in the standard way, so if this select input is 1 then x in is going to be loaded into x. Similarly, if the select input here for this multiplexer is 1 then y in is going to be loaded over here, so I denote this input port of this multiplexer by 1.

Because it indicates that when the select input is 1 this one will be routed at the output, so this one will be 0 and this one will be 0 okay, and I call this select signal appropriately load x in okay because it is purpose is to facilitate loading of x in into the register x at the end of the clock cycle. Similarly, this one I call it load y in okay, then what do I call let us look at this particular multiplexer what should we call the select input to this multiplexer this I name swap okay.

So if swap is 1 then I must allow route is y - route y in a way so that y can be kept ready at the input of x, so when swap is 1 y will be routed eventually to x so this is the input number - port number one and this is the port number 0, okay. So this select input is multiplexer is going to be called a control signal y, similarly this one also I will call it y call it a same the name swap so when swap is 1 then y is going to be loaded with x so x will be routed over to like y.

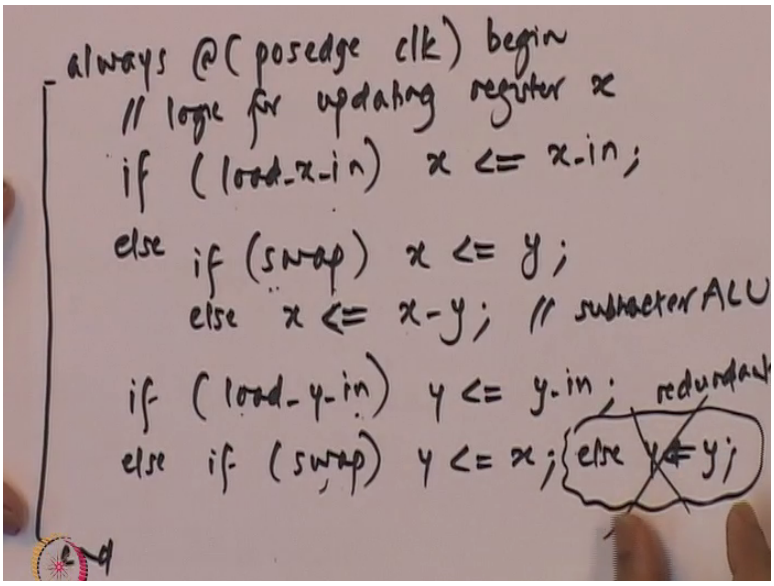
And so if not swapping then y will be routed eventually to y this multiplexer is for the input situation that is when we are loading that fresh new inputs ready inputs into the x and y that is

the start of the computation before the loop begins. And this 0 input to these multiplexers this and this are going to be used in that in the looping computation in the computational loop, okay.

So this is how generally we will kind of once we take us we build inventory of the datapath components we figure out which datapath components we require and how they are to be connected with each other and to wire them up. Then we start figuring out the control signals that are required to control this datapath at different times with different control signals, so it is not going to be any magic.

So we need something which will generate appropriate values of this control signals namely load x in, load y in, swap, I mean such control signals have to be generated by a sequential circuit by itself and that sequential circuit will be a finite state machine typically. So the datapath might look like a bit of clutter but it is quite easy to capture it in the syntax of Verilog, so let me try that now.

(Refer Slide Time: 27:10)



So what we are doing over here in the datapath is specifying that the registers in the datapath's are x and y now I specify how the registers are going to be updated at the end of every clock cycle that is at the triggering edge of the clock the positive edge of the clock. So the logic for now first I will describe the logic for updating register x so logic as follows. If we notice that we saw in the previous just while ago that there is a multiplexer here.

And when we have, when we need to take the input and register it to latch in to x we generate we assert this particular control signal we need to assert its control signal load x in and similarly y in has to be asserted, so and when that happens x in is going to be loaded into x and y in is going to be loaded into y. So we will assume that somebody has helped us assert that particular signal load x in as well as the signal load y.in.

So if load x in is asserted then x is to be loaded with x_in, else again let us look at this if load x in is not asserted then it is definitely not the situation at beginning before the computation begins that needs to be at likely to be in the computational loop. There the route to x, register x is going to be like through this input port of this multiplexer and what we what will be loading into x will be the result of this particular multiplexer which either would be y or x - y which would depend on swap okay.

So if swap = 1 then will x will be loaded in that situation with y and if swap = 0 then x - y the result of this ALU will be routed at the input of x. So else if swap has been asserted then x is to be loaded with y else x is to be loaded with x - y, this x - y is like I am using the power of Verilog the right hand side is x - y in fact implicitly this means there is a arithmetic logic unit subtractor which will be subtracting x and y.

And the result of that subtractor is going to be kept ready at the input of x, so this statement captures the use of this ALU as well as the use of this multiplexer, okay. So that is the power of Verilog is also being illustrated here, so this if else statement which also has another nested if else describes the way x is updated the register x is updated, similarly the way register y is updated is almost the same if load y in which is the input to the multiplexer is asserted is equal to 1.

Then, y is to be loaded with the fresh input from outside, else what is to be loaded in y. The other option the other signal which is again the result of this multiplexer which is controlled by the controlled input swap, so if swap is 1 then y is going to be loaded with x else y is going to be

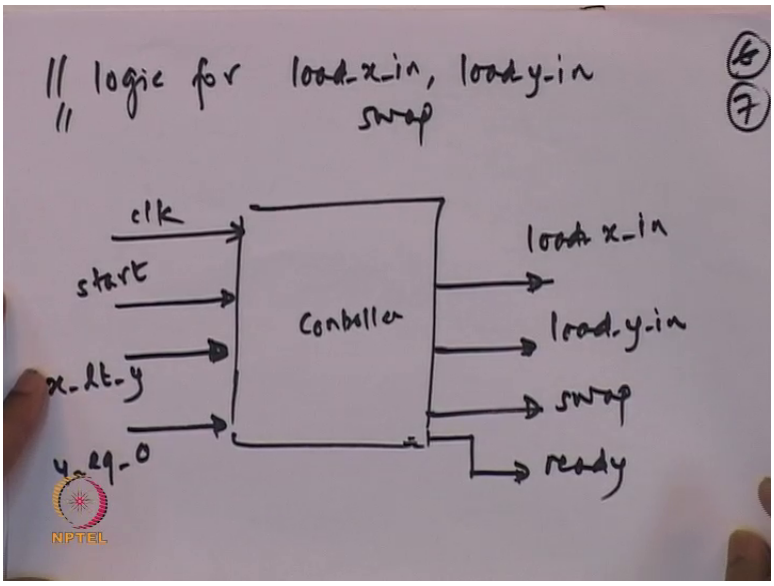
loaded with y itself, swap y okay, else I can say y but with the help of I mean this Verilog synthesizer will find its redundant we do not need to say it will understand it automatically.

We do not need to say it redundant it is not wrong it is redundant, okay. So this is the main part of the logic of update of x and y okay so try and connect it up with multiplexers and arithmetic logic unit and so there multiplexers are kind of implied by this if else statements so there will be a multiplexer with the control input load x in there is a role of multiplexer with the control select input swap, there is the role of multiplexer with select input load y in, and another multiplexer with again the swap as the select input.

Then there is a role of this subtract ALU subtractor ALU I mean in this behavioural statement to be it is implicit that we are making use of the subtractor datapath component and the result of that is to be routed to x using appropriate multiplexer input okay, so that is swap is 0 then x might the result of the subtractor is going to be routed to x, so Verilog synthesizer is going to map this two major portion of the datapath that we just drew.

So with the help of this Verilog we have a fairly like you know a text based and fairly unambiguous description of how the data moves how the data is computed, I mean how is data's processed through subtractors and so on so forth, how that moment of data is controlled by the multiplexers, now what remains obviously the major thing that remains is to understand how this control signals are to be generated and who is going to generate that.

(Refer Slide Time: 34:33)



So what we need is logic for control signals load x in, load y in control signals swap, so the entity that does is this is called controller and output of this controller are these control signals which are being used to drive the datapath, so in particular load x in, load y in and swap we missed once signal that we would like a signal to indicate to the outside world to the external system I mean subsystem to the external Universe whether the result of the computation is ready.

So I believe a controller can help me to do that so let me assume that controller also generates a signal call ready okay, what are the inputs of the controller obviously the clock, clock is an one of the important inputs to the controller I mean it is a synchronous sequential system by itself, so a clock will be input then other inputs to the controller are we assume an input from external sources which triggers this computation to begin.

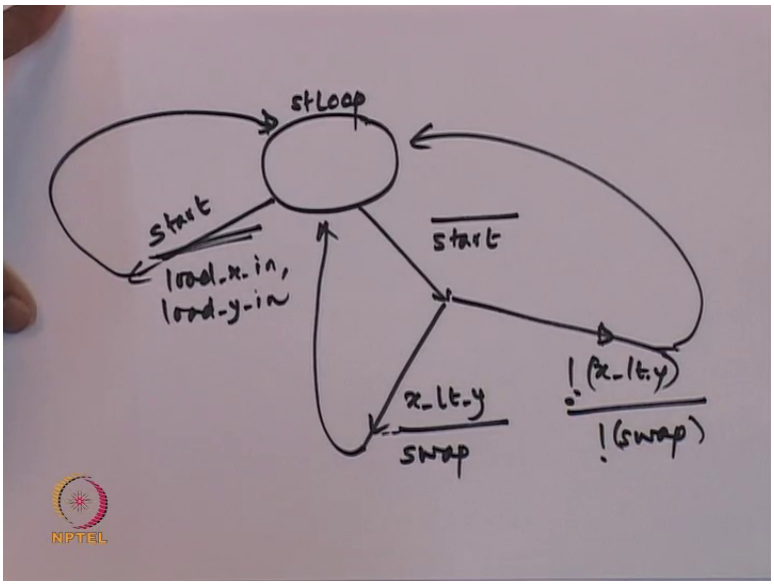
So when start inputs asserted then the x and y registers are to be loaded with the data available at the input ports of this module this design unit, so this is an example of an external input also sometimes some signals from the datapath are also input to the controller, so far in this case we realize that the comparator is output which says whether x is < y and zero checkers output which says whether y is = 0 are also going to be used in controlling this datapath.

So we are just about to discuss the controller, the controller as we just said would have generated control signals which are input to the datapath and also controller would make use of this data

signals from the datapath say x < y, y = 0 as input, other than the input from the external sources first start for triggering the system to do the computation and a clock, because very typical this controller is going to be of synchronous finite state machine.

Further the controller also will generate output for the external world okay say ready indicating that computation is complete, so let us look at the internals of a controller it will turn out that we can make this controller I mean quite often this controller is going to be very simple in fact in this particular case has an illustration will show a controller which is purely combinational it is not even a FSM with multiple states, a single state FSM which is essentially a purely combinational logic.

(Refer Slide Time: 38:05)



So controller is going to be a single state finite state machine which is essentially purely combinational this is the state which is the in which does controller loops performing different types multiplied equations of that gcd computations which involves essentially swapping and subtracting, the highest priority is given to the input signal start, so the controller decides looking at start whether the start is has been asserted or the start is deasserted, okay.

This is the situation when start is asserted, when start is asserted the controllers job is to assert the control signals load x in and load y in right, so and after it does that it will go back to the

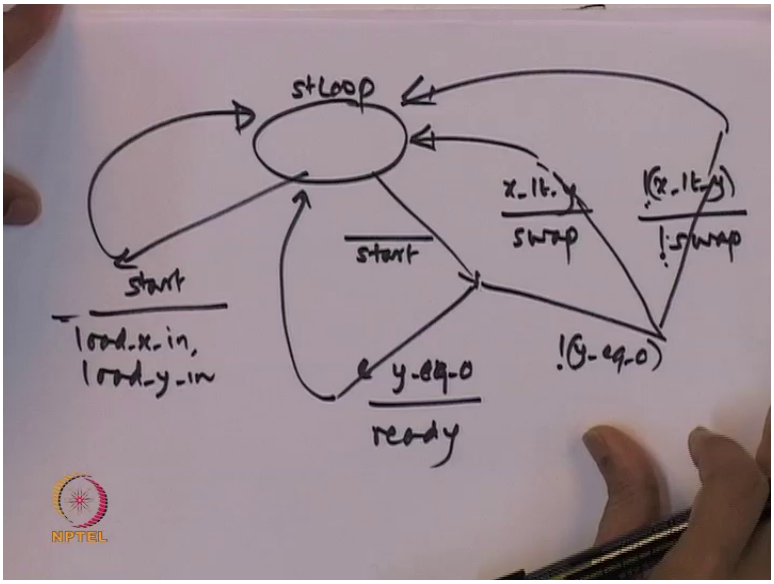
same state, so it just simply asserts this control signals load x in and load y in, when the start signal - start input signal is found to be asserted is found to be high.

On the other hand when the stars signal is found to be the deasserted that means it is like which means the situation is like that we are already inserted a computational loop and we should be doing things based on the result based on the status input from the comparator, the status input from the comparator says x < y or it could tell us that x is !< y okay, either this condition holds or that condition does not hold in both the cases will stay back in the same state.

But different kind of control signals would be generated different values of control signals will be generated, in particular if x < y we know the controllers job is to tell the datapath that it must swap x and y, so the swap signal is asserted in response to noticing that x < y and if x is !< y then swap signal is deasserted whether I am bit casual with the notation sometimes for negation I use over bar sometimes I use this exclamation mark so you that is, okay I think.

You see that this is what is the essential description of the controller it is a single state controller that means this purely combinational it simply generating the output signals load x in, load y in, swap etc. simply by looking at the values of the inputs, okay, I think I missed something let us look at it again.

(Refer Slide Time: 41:17)



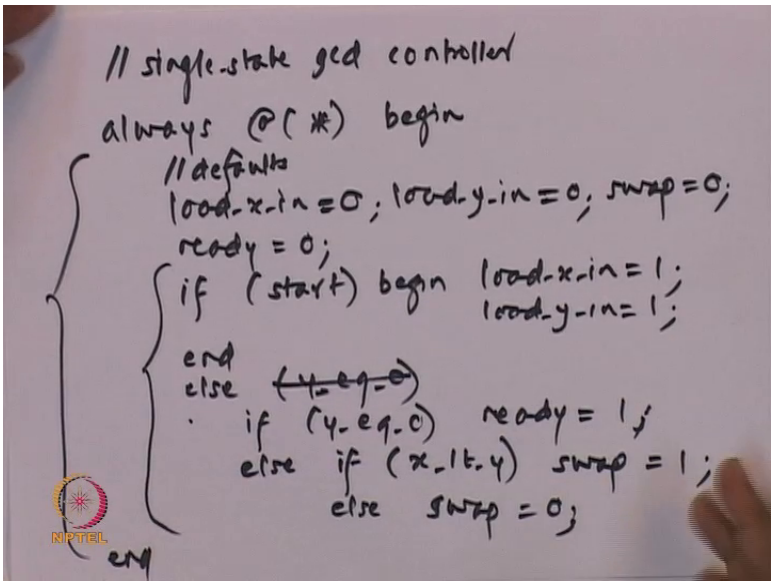
This state loop if start is asserted then x input and y input are to be loaded into the x and y registers at the end of this clock cycle, on the other hand if start is not asserted then it is that means we are in the computational loop, now it is a situation whether now we look at whether y is whether the computation has finished or not that would depend on whether y is already = 0 or not either y is = 0 or y is != 0, okay.

If y is = 0 we will assert the ready signal the controller will assert the ready signal and get ready for the next clock cycle, if y is != 0 then we have two situations which are based which depend on whether x is < y or whether x is !< y okay, if x is < y anyway the controller will next clock cycle will stay in the same state, but the controller signal swap will be asserted that is the output of the controller and if x is !< y then swap will be deasserted swap will be set to 0 and we will go back to the same state.

I missed that the use of y = 0 which is very important because every time in the beginning of the clock cycle while in the computational loop you should the controller will check whether y has become 0 or not, if that happens to be the case that means the output the gcd is ready in x, so the external system has to be told that we are ready with the result in x, if y is != 0 that means we are still not ready with the result that means we have to like compute further.

So that would depend on whether x is < y or x is !< y, if x is < y we should be telling the controller should be telling the datapath that it should swap or it should like in the other case it would not swap it would subtract x subtract y from x and let y remain as it is that is not swap case, okay. So that is the controller this controller can be described in Verilog.

(Refer Slide Time: 44:03)



The this is the single state gcd controller so it is going to be purely combinational no role of clocks here combinational logic can be described using as continuous assign statement or as always blocks which are not sensitive to the edges of the clock like this, so initially the defaults so I will specify that default values for the all control signals are as follows load x in is 0, load y in is 0, swap is deasserted and so is ready, okay.

Now depending on different situations these values are to be like appropriate values of the could this outputs of the controller as to - are to be specified, so if start is asserted then since we know that load x in will be 1 and load y in will be 1 will be asserted okay, that is the end of the case start is asserted. If start is not asserted, then we are in the computational loop and first we will check for whether y is = 0 or not.

If y is = 0 then we will further check else sorry else if, if y is = 0 will further check okay will image at least I mean this necessarily means that ready signal has to be asserted in this clock cycle, else if y is != 0 then it would depend on whether x is < y in which case the swap signal has to be asserted else the swap signal will be deasserted okay, letting x - y subtract subtractors result go to x and y remain as it is, okay.

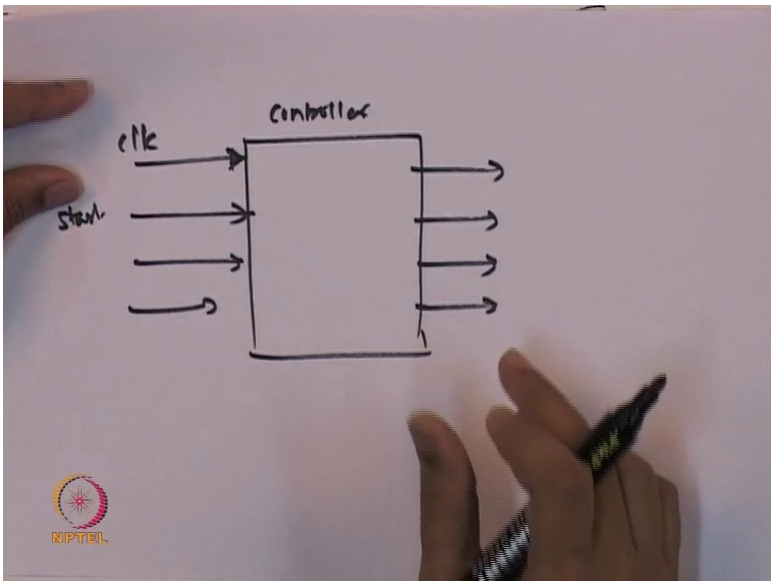
So this is the logic for generation of this control signals and output signal ready, so this always block which is purely combinational you see that with the help of this default blocking

assignments we have made sure that in for every possible situation depending on the input values start y = 0 x < y we have in all situations specified values of this for signals load x in, load y in swap and ready, okay.

Then default values are this and not default values happened to be like specific get specified for specific conditions okay, so is exactly it is a Verilog description of this particular control state diagram a single state diagram can check it start is asserted then load x in load y in are asserted, if start is not 0 so if start is not asserted that start is 0 in that else block it would depend on whether y = 0 or if y = 0 ready signals will be asserted and if y is != 0.

Then it will further depend on x whether x is < y or x is !< y in which case the swap will be asserted or deasserted which is described over here okay, if y is != 0 then it would depend on whether x is < y okay so its x, this captures that particular state single state gcd controller which is purely combinational, okay.

(Refer Slide Time: 48:41)

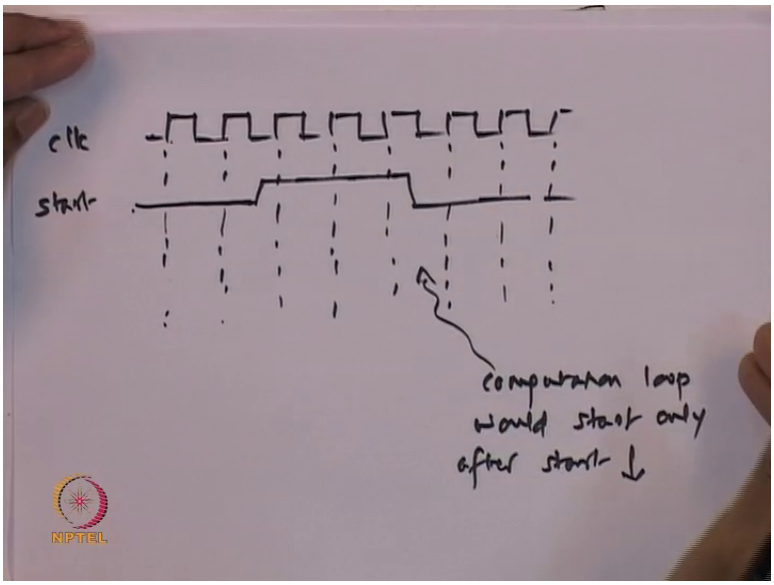


So now it is a bit surprising that this particular controller that we discussed although it had this I mean with thought this is also an example of a synchronous system it is in general it is in fact indeed going to be in a very typical finite state machine, so it is going to depend on the clock signal for synchronizing it state updates and so on, so this was a controller which turned out to be a single state controller.

So just to illustrate that like most typically because we will have some protocol on the inputs some protocol about the outputs what the external Universe wants I mean how the external world wants to interact with this particular computing machine custom computing machine for say greatest common divisor computations. So we will just modify the specifications will understand what is the limitation of that single state controller machine.

And try to understand what could be the modification and how that modification of the specification would result in a slightly more general controller.

(Refer Slide Time: 50:01)



So note that in this, this is the clock let us see the waveforms sample scenario this is start signal this we are using rising edge triggered discipline, so the clock periods clock cycle starts at the rising edge and ends at the rising edge of the next clock period so let us say the start signal got asserted sometime here, okay and state asserted for whatever let us say couple of clock cycles and got deasserted here, okay.

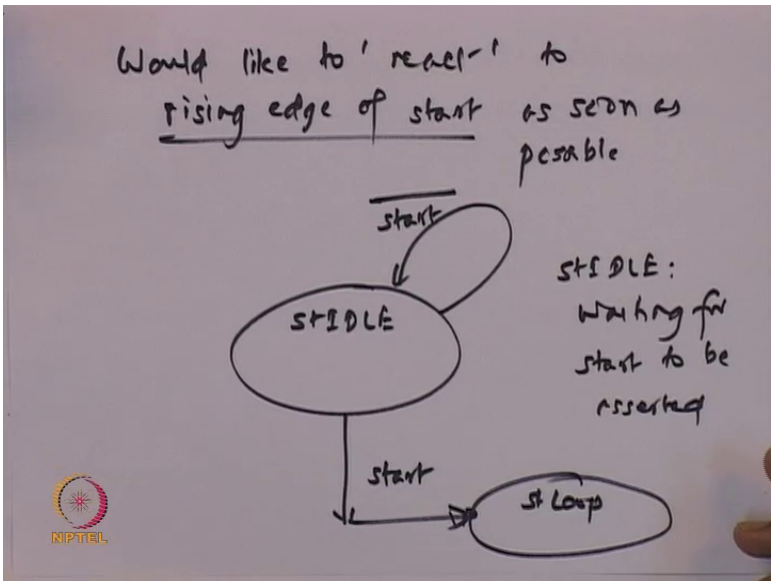
Now we note that while because the way the logic has been specified in this if else statement the priority is given to start signal if start is asserted then we load inputs x input and y input into x and y registers and we simply do not proceed to the computations until start is found to be the deasserted right. So if some it is if the external circuitry which is driving or gcd keeps the start

signal high for long enough time hoping like you know just feel be doing that it has figured this system.

But until the start is deasserted the computation will not start okay, well start is remains asserted the system will the gcd subsystem will keep on like loading the x and y registers from the input. So the computation would not start, computation would start only over here okay, only after start is goes low, so at the end of towards the end of this clock cycle something will start happening in terms of swapping and subtracting and so on.

And it will take a few more clock cycles for the datapath to yield a result and system would stay in the same state that is a that is what behaviour of that is a purely combinational controller rate. So the drawback of this single state control is that it does not respond to start at the earliest it responds to start sort of like you know when the start goes down the computation begins, so let us try and modified work around this little bit of problem.

(Refer Slide Time: 52:58)

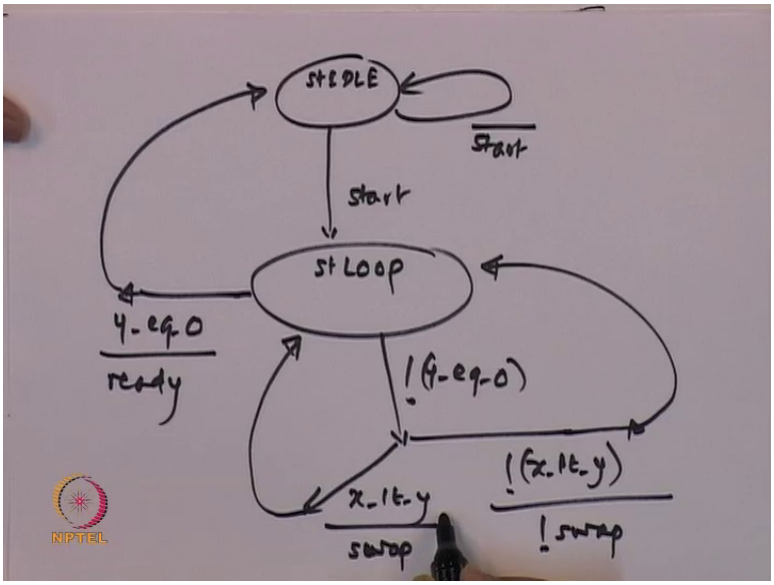


So we would like to react to rising edge of start as soon as possible okay, so the FSM will be in a state called state idle, in state idle we are the system will be essentially waiting for start to be asserted okay, so the system would stay in this controller would stay in the state as long as long as start is 0 okay, movement start becomes 1 the system would go to a new state let us call it state loop and in this state it will do the computation, okay.

So you notice that moment start becomes when start becomes when its noticed by the controller, controller would have been in that state idle state, but it is going to in the next clock cycle it is going to a new state in that new state the controller will start responding to like will start will ignore like the start signal.

And would simply like pace its actions like based on the status from the datapath that is whether y is = 0 or not that is when the whether computation is done or not or whether if computation is not yet done whether x is < y, so which is when we should swap or we should not swap okay, so that is what is the interesting things are going to happen in this state loop okay, let us see what exactly happens in this state loop.

(Refer Slide Time: 55:01)



So this is the state idle the system controller stays here as long as start is 0 and start is found out to be asserted the system moves to a state called state loop it is an arbitrary name that we have given to it, in this state again first we are going to look at this controller would look at whether y is = 0 or not or y != 0, if y is = 0 then that means the computation is done, so it is a case it is the situation for to wait for the next stat trigger.

So immediately we will assert the ready flag and in the next clock cycle we will go to the state idle waiting for the start to be asserted okay, if y is != 0 that means the computation has to

continue then it will depend on whether x is < y or whether x is, whether it is not the case that is x is !< y in either case we know the computation has to carry we done for one more iteration at least, so we go back into come I mean we go back to the same state to I mean you can see the we are still in the computation so we will go back to the same state.

But different control signals will be generated, so here swap will be asserted and here swap will be deasserted, so it is essentially like we just split that particular earlier single state controller removed created to one extra state to respond to which is job is to only kind of wait for the start signal as a main state which is where the computation is happening. So now we see that this kind of controller will ensure that the start signals is responded to as soon as possible we do not have to as in the previous case.

We will not wait for the start signal to die and then the computational will begin, okay. This is the power of finite state machine just a minor changes in the finite state machine note that datapath would not have changed at all it is just the way it is just that we have chosen a different protocol which decided to react to start like focus on start signal in a state idle state which is the initial state and once start signal is noticed we immediately moved into a different state.

If you had stayed back in the same state priority to start would still would have been there and it would have like you know inhibited this swap and subtract operations. So by moving into a different state we have sort of like you know help the system controllers ignore the start signal note that in this particular state there is no role of start signal okay completely ignore the start signal.

It only depends on whether y is = 0 in which case it is going to know that computation is finished ready signals asserted and if is y != 0 it would depend further on whether x is < y or x is !< y, okay.