Advanced VLSI Design

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

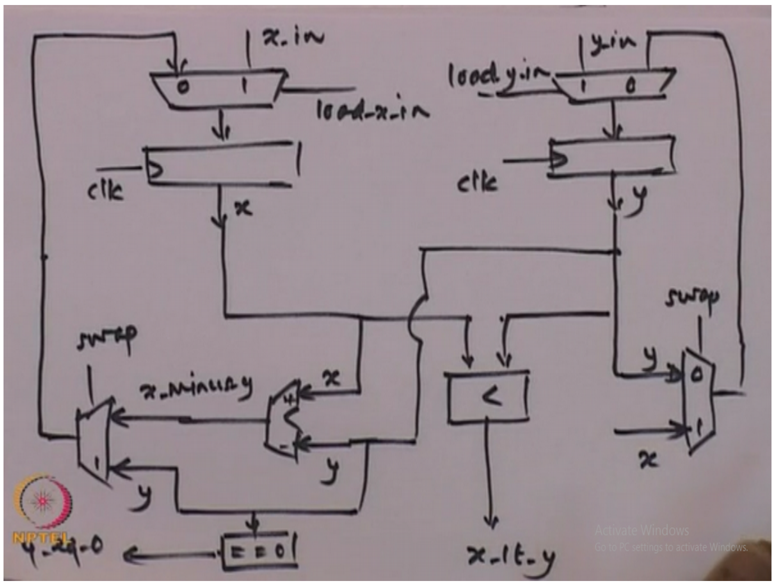
Lecture - 25
FSM + Datapath (Contd.)

So welcome back. We are discussing finite state machine + datapath, controller + datapath and we were discussing a particular example of latest common divisor computation, iterative computation and we had discussed and outlined the datapath, its corresponding Verilog description and the controller for that which turned out to be a singles.

To begin with we designed a purely combination of controller we found a bit of deficiency in that, in the sense that the start signal was having too much priority as long as start signal was higher the system was not at all proceeding to the computations. So then we outlined once if you change the protocol so that we can respond to the start signal as soon as possible. Then we will have to, we need extra states.

And the controller will be a more genuine multistate finite state machine, okay instead of a purely combinational controller. So let us quickly review that and discuss it further.

(Refer Slide Time: 01:36)



So the datapath for GCD just to consolidate the picture in your mind. I will repeat quickly draw bit more neatly. This is the X resistor which is driven by output of a multiplexer and this multiplexer is controlled by output of a controller with a signal load Xin and when this load

Xin is asserted the input from the external sources Xin is loaded into the register X. This is the clock resistor, okay.

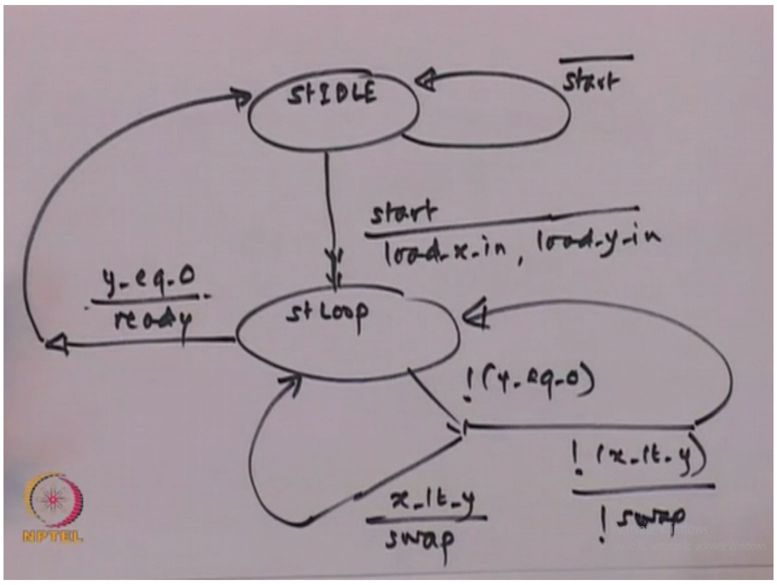
Otherwise if load Xin is 0 then the input to X, X is going to be loaded with something that would depend on whether swap is asserted or not. If swap is asserted X has to be loaded with Y the swap is de-asserted X is to be loaded with something called X–Y, which is the result of arithmetic logic unit which subtracts Y from X, okay. This is the same Y which is going over here coming from somewhere else. This is X which is coming from over here.

That is the picture for how X gets loaded either from Xin when load Xin is high or if when load Xin is not high it will be loaded from either Y or the result of this ALU X–Y depending on whether we swap or not. So correspondingly this is Y there is a multiplexer here and again that multiplexer is similar to that for X is going to be controlled by a control input, output from the controller which is input to the datapath's load Yin,

and when that load Yin is asserted Yin which is the input ready at the port of this GCD module is going to be loaded into this block resistor. Otherwise like in this case Y is going to be loaded either with depending on swap either with X or with Y, okay? We also require a comparator and a 0 checker, right other than this.

The comparator will need to compare X and Y and tell us with indicate to us through this signal whether X is less than Y, and a 0 checker is going to look at Y whether Y 0 or not and tell us whether Y=0 or not, okay? This Y is coming from over here, I guess this is slightly neater diagram for the datapath. I have shown all the datapath components that we require.

(Refer Slide Time: 05:32)



So now we have started working, I mean on the protocol like where this start signal is quickly responded to and that required a use of state called state idle which the system is kind of waiting for the start signal to the asserted and when start is not asserted staying this state. Once the start signal is asserted then the system in the next clock cycle would move to a state St Loop, okay.

But before it, by the end of the current cycle before it moves to the state node Xin node Yin would have been asserted by the controllers so as to ensure that at the end of this particular clock cycle while moving from state idle to state loop Xin and Yin would be loaded into X and Y, okay. So at the beginning of the clock cycle when the system has moved to state loop X and Y have been loaded with input vales, okay. So now we are ready to conclude.

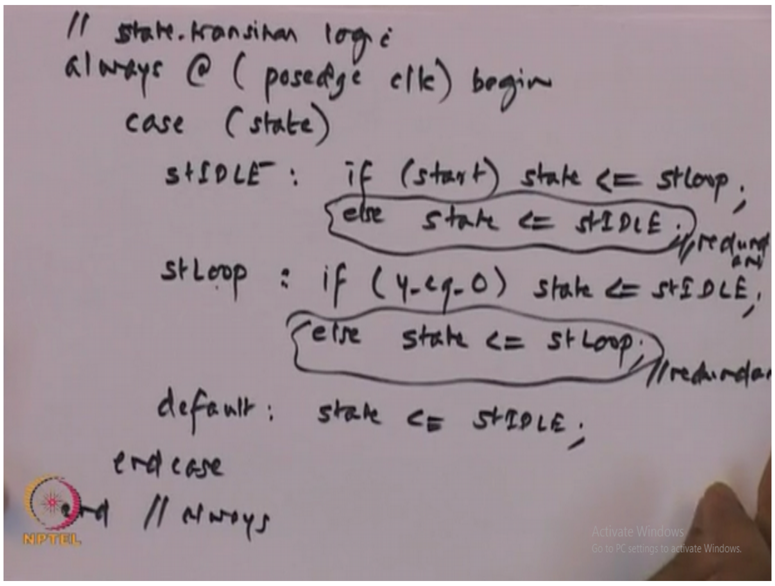
Then we will check, so this is where the system will reside for a few more clock cycles as long as we have interesting X and Y inputs. In the beginning of every clock cycle this will check whether Y=0 or not either Y 0 or Y is not 0. So, if Y is 0 then we know that computation is over the controller asserts the ready signal which is the output of that controllers and goes back to the idle state.

So, that it is ready for like taking in the next input X and Y into X, okay? If Y is not equal to 0 then we are still has some work to do and that would exactly what to do will depend whether X is less than Y or whether X is not less than Y, okay? In this case swap signal will be asserted and in next clock cycle we again go to this same state where we will behave the actions of the datapath would be again dependent on XY=0 or X less than Y and so.

So, we will go back in this state which will have some more computation to do. If X is not less than Y then we do not swap and in the next clock cycle we go back to the state. So this is the state diagram that we have arrived at, okay? To make this and I will be guess or like to connect this up, correlate this with Verilog to understand a Verilog semantics and synthesis process it would be instructed to write the Verilog code for this controller.

Note that datapath has not changed, okay? It is just a controller changing and with the help of that change in the controller we have been able to implement a different protocol of the way we respond to start signal and so.

(Refer Slide Time: 08:40)



```
// state.transition logic
always @ ( posedge clk) begin
    case (state)
        stIDLE :  if (start) state <= stLoop;
                  else  state <= stIDLE;  //redundant
        stLoop :  if (y.eq.0) state <= stIDLE;
                  else  state <= stLoop;  //redundan
        default:  state <= stIDLE;
    endcase
end  // always
```

So the Verilog for generation of this I mean like the state machine. So, first let me describe the state transition logic. State transition is taking place at the triggering edges of the clock that is a passage of positive edge at the clock, okay? So, typically it is going to be described using a case statement depending on what the current state is we had different cases. The current state is state idle then we stay in the same state as long as start is 0.

But it starts becomes one start is asserted then in the next clock cycle we will be ready to move to state, state loop, okay? Hence we state this is not required else we stay back in the same state, state idle, okay? That is how we respond while we are in the state idle. So the transition in the next clock cycle we will move to state loop if start has been found to be asserted and stable at the end of this clock cycle and otherwise we stay back in this loop.

So this is corresponding to this particular picture. If start is asserted we are going to move to state loop, if start is not asserted we are going to stay back in this. Now the logic, if the system is in state loop, okay the state transition logic not the logic for generation of the control signal that we will describe in a separate always block.

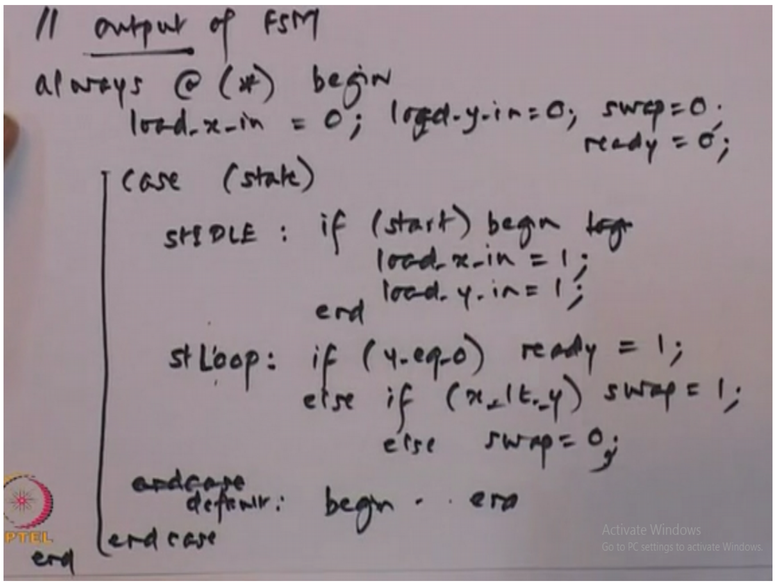
So, here it would depend on whether first the system will look at the status signal Y=0, which is coming from the datapath. This controller system will look at Y=0. If Y=0 then we know that next at we are going to move to is idle state. So, waiting for next triggering, next trigger for the system to start, okay? Else the computation would depend on whether X is less than Y or X is not less than Y but the next state is going to be the same, okay?

You notice that although in this diagram, we have if Y is not equal to 0, we have two further situations with X less than Y or X not less than Y but that is for the output control signals, okay? Control signals which are being generated by this controller. But the next state is going to be independent of whether this or this next state is going to be ST loops. So, as far as next state logic is concerned, state transition logic is concerned this is sufficient to say this.

In fact, this is also redundant because Verilog semantics will tell the synthesizer that in the next clock cycle we are staying back in this states with this would have been similar to this. This else block is also redundant. Okay then just for clarity I am putting it here. What else, these are the only 2 states, right? We have completely captured how these transitions take place, okay?

But for completing the case statement we have to have some default and in which we really do not care but just to be on the safer side we say that if some funny situation is there and anticipated like on clear values on the states signal then we moved to state idle. So, this Verilog code fragment captures this state transition logic.

(Refer Slide Time: 13:17)



```
// output of FSM
always @ (*) begin
    load_x_in = 0; load_y_in=0; swap=0;
                                  ready = 0;
  case (state)
    stIDLE : if (start) begin ~~log~~
                 load_x_in = 1;
                 load_y_in = 1;
             end
    stLoop: if (y_eq_0) ready = 1;
            else if (x_lt_y) swap = 1;
                 else swap= 0;
    ~~endcase~~
      default: begin . . end
  endcase
end
```

Now the logic for generating the control signal is also typically described using always block, oaky. So this is the output logic output of, okay? So the outputs of FSM, the control signals to the datapath and also a signal ready which is output of this GCD sub system, okay? So, this is combination logic. So, this is going to depend purely on what the current state is and within that depending on current state.

You also look at the current input values to the controller which will be the signals X less than Y or signal Y=0 and so on, okay? So, we have to specify about what will be the control signals. The control signals are load Xin like in the case for this purely computational controller which we seen in the last class. I used the default values. I said a default load Xin to 0, load Yin to 0 swap to 0 ready towards a 0.

And here I am only going to show the non-default values. So, if in the state ideal then only when start signal is asserted will be doing something to this load Xin and load Yin signals will be asserted. Load Xin will be 1 load Yin will be 1, okay and we would have moved to that state loop as described in the other always block which is the state transitional description. So, nothing else as far as the output signals are concerned in state ideal.

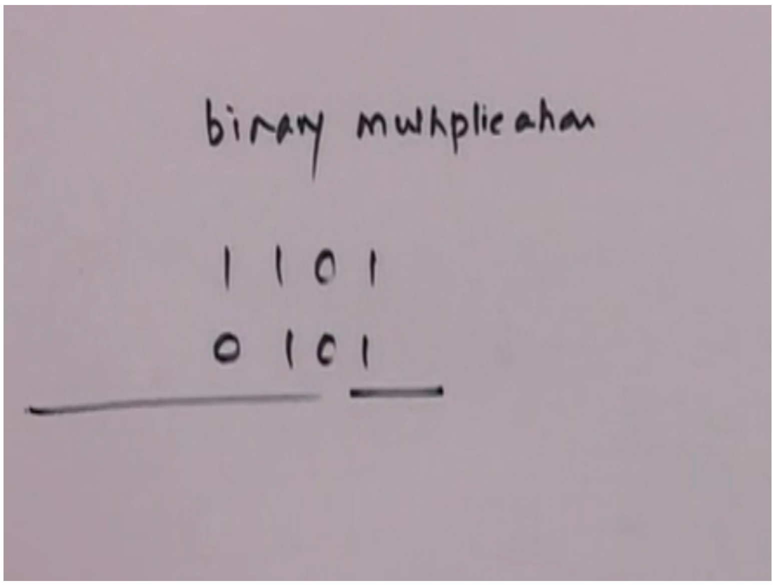
State loop just look at the diagram for state loop. In state loop the output signals will depend on whether Y=0 then the ready signal will be asserted and if Y is not equal to 0 it would depend on whether X is less than Y or X is not less than Y. So, if Y=0 then ready is 1 else if X less than Y then swap is 1 else swap is at the default value 0, okay? Sorry, the default is also required but default whatever.

So, I do not care really in case so, I could just have an empty statement here let the default values remain as it is, okay? So, using the blocking assignments and specifying the defaults early enough and specifying with the non-default values in this block of code, I have completely described purely combination logic for generation of output, okay?

And this logic depends on the current state as well as the current inputs of the controller, which are start Y=0 X less than Y old signals are playing a part in deciding the values output values of this controller which are some of them are input to the datapath some of them are output to the external world. So having seen the GCD example quite and a variation of that to illustrate the typical need for multi-state finite state machine, rather than purely combinational controller.

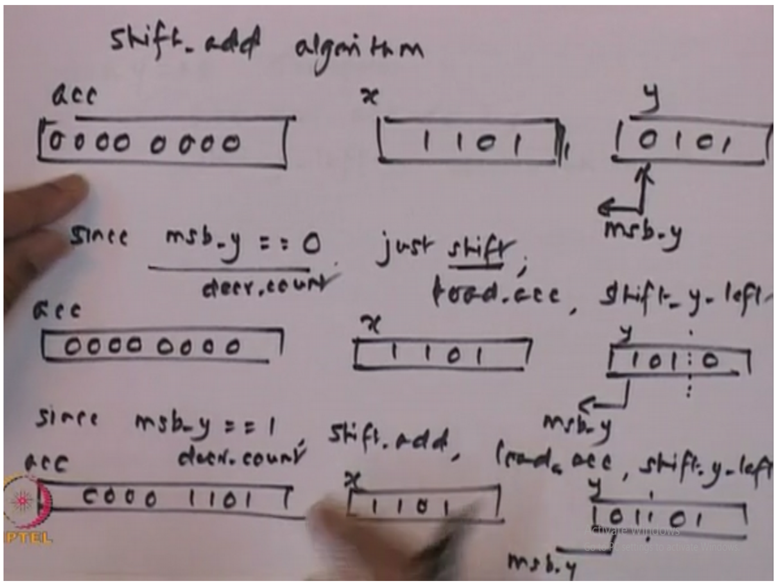
I will just, for a change will move to different example of again of FSM + datapath, very simple example where the concept of multiple state will again be evident but in a different context with a different requirement, okay?

(Refer Slide Time: 18:06)



So, this is the example of simple binary multiplications where let us ignore this, let us assume that numbers are positive, no-negative or positive numbers and we are going to multiply them using very simple the school grade multiplication system adapted for binary numbers. So, let us say the numbers are this is number 13 in binary this is number 5 the results should be 65. Say the way this computation is done is as follows.

(Refer Slide Time: 18:50)



The shift the so call shift add approach, shift and add algorithm which is very easy to hardware and that is what we are going to see it as FSM + datapath is as follows. So, we are going to maintain an accumulator where the multiplication will accumulate that accumulator will be initialized to it will be sufficiently big accumulator capable of storing the product, this is X and this is Y.

Input X and Y have been stored into registers Xin and Yin have been assumed to be stored in registers X and Y at the start of this computation, okay? Of course some of this registers, especially accumulators is going to be changing X will not be changing X is going to be repeatedly added, Y will be changing because we are going to look at the bits of Y one by one, okay?

So, assume that Y is going to be put in some kind of shift register so that which will be shifting left 1 bit per cycle at a time, okay? So, the algorithm is as follows. It looks at this particular MSB of Y this is the let me call this, this is MSB of Y. It happens to be 0. Because of that what we are going to do is simply shift since MSB Y is 0 just shift, okay? So, shift accumulators so the accumulator is going to be again shift and load accumulator, okay?

We also for the next clock cycle we will also pop out this particular MSB of Y, so we will say shift Y register to left, okay? So, these are the actions we will be taking in this clock cycles. So, at the end of the clock cycle Y would have been because of this control signal Y register would have been shifted left, so this MSB would have been popped out. New MSB would, MSB stands for most significant bit.

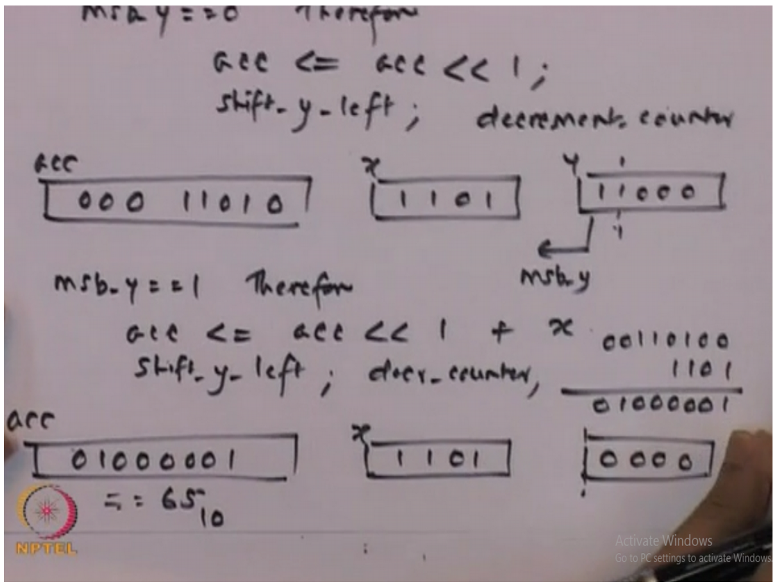
We will be looking at this particular bit in the next clock cycle. Accumulator would be loaded with the shifted version of the current accumulators, just shift and load accumulator. So, here there is no difference is going to be 0’s all 0s, okay. X is going to be as it is in the next clock cycle because of this shift Y left we are going to see 101 and 0. So, note that this 0 is the 0 was in this particular 0 telling 0 is not part of the original input,

and we are going to keep track of this with the help of a counter. So, I am not missed it out here but assume that there is a counter which is keeping track of how many iterations we are progressing we are progressed. In the next clock cycle, when the beginning of the clock cycle we have this situation, Y shifted accumulator has been loaded with the shifted version of that okay.

But now we have interesting situation, the MSB is 1 since MSB of Y is 1 we say shift and add okay. Load accumulator with the result of the adder and also pop out Y by shifting the register to left. So, now what we get is to the accumulator, we like know accumulator is loaded again with the shifted version + shift + add so that is 0’s+1101. So, this is going to be the new accumulator.

X is going to be as it is and in the beginning of the next clock cycle we will have this okay, the MSB will be 0 again. Okay, MSB of Y will be 0.

(Refer Slide Time: 23:36)



So, now since MSB is, since this therefore accumulators gone to be just loaded with shifted version of it, no addition we will take place. We will also be shifting Y to left and let me also something that missed out earlier. We will also be decrementing counter. The counter which is going to keep track of how many iterations we have progress. So, in the earliest steps also I should specify that decrement count, okay.

So, all this kind of actions have been, signals have been generated, appropriate additions are being done at appropriate times, shifting is being done, loading accumulator is being done, okay? So because of this, what we are going to get in the accumulator is shifted version of 1101, okay? X is not changing at all but Y is going to be now shifted further and for the last iteration you see that particular the last bit of Y as the MSB now.

Remember the Y was 0101, so this is we are in the last iteration. Now because MSB is high, MSBs of Y is 1, so therefore accumulator is going to be loaded with the shifted version + X, okay and then at the same time will instruct the shift register for Y to shift to left and instead of the counter to decrement. Now what is the result that is going to be at the end of the clock cycle in the accumulator is going to be this shifted by 1, 1 bit + this particular number 1101.

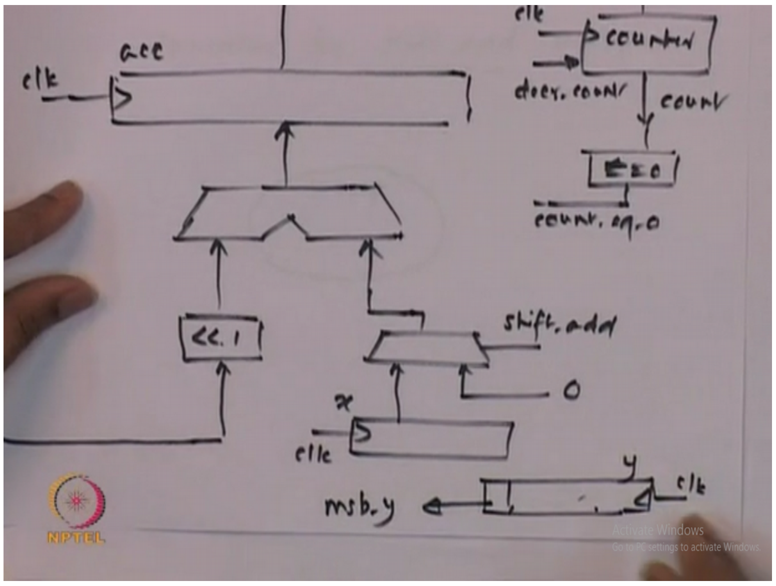
So let us calculate that so 110100 and this further 00 this is the 8 bit accumulator and 1101. So we get 10000010. So what we end up here is 01000001 and Y would be now all 0000, okay? Note that counter would have decremented to 0 now we know that we have done 4 iterations, okay? Which are the 4 iterations. Iteration number 1, iteration number 2, 3 and 4, okay?

At the end of this 4 iterations we should have, we expect to have the result in accumulator about this result is 64+1 this is 65 in decimal, okay? So it is exactly what we expected to be 13*5 65 okay. Because this is 13 and Y was 0101 that is 5. So we have this correct simulation of this particular shift at add based algorithm. So its datapath is going to be quite simple, so it is quite instructed to look at this as another example of FSM + datapath.

So what is the datapath? Yeah, so what you expect? We definitely require an adder, okay? We require a shift register for what? Shift register for Y. We require a multiplexer; I mean in most cases we require there is a role for multiplexes there is a role for ALU. We will say exactly, we require a counter for keeping track of the count for updating the count. What else do we require in this algorithm?

Accumulator, okay a register, several registers. one shift register for Y and registers for appropriately big register for accumulator and for X. X is in fact constant, so we might able to optimize it a little bit. So let us start like placing these components and wiring them up.

(Refer Slide Time: 29:08)



So here is a big accumulator. This is a clock resistor; I call it accumulator. This accumulator is going to be taking at times just a shifted version of itself or the result of the ALU addition, okay? So when you test the shifted version of itself it is like will regard it as if it is the adder is adding 0 to it, okay. So in the case of shift and add will be passing this accumulators content through 1-bit shifter and putting it at this adder,

and other input to the adder is going to be either X which is stored in this register X, the clock register. Either X or it could be 0. So this will depend on whether we say the controller says shift and add. If the controller says only shift that means shift add is 0 then 0 will be sent to this adder and then it will have the effect of loading updating the accumulator with its left shifted version, okay? Note that this is not a shift register.

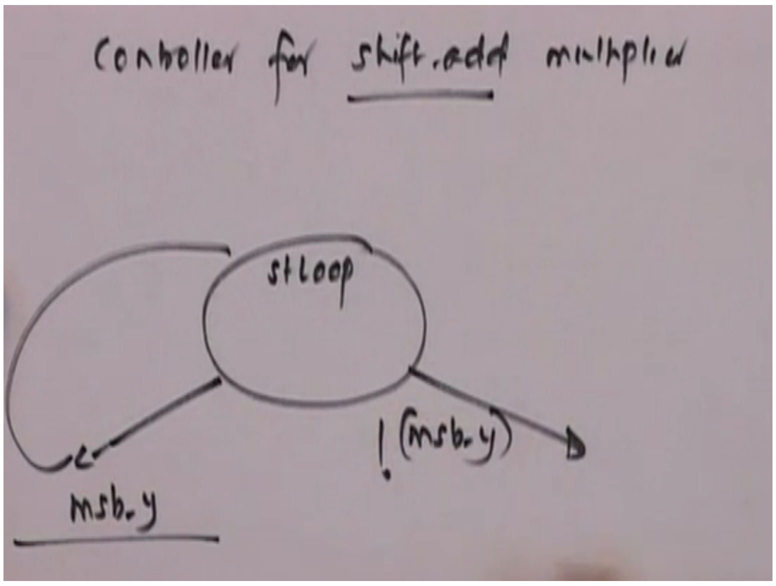
This is a simple shifter, combinational shifter, okay? So in the inventory of datapath very typically we require shifters, combinational. In general, the balance shifter is very useful component, very efficient. In general, 1 uses balance shifter. A very important typically component of useful datapaths. Okay, here we have a simple combinational shift which is only shifting by 1 bit.

We also require a counter again asynchronous clock system whose output is being used like and which is being instructed which is being told which can be instructed to decrement and so on. And will be interested in seeing whether this count has reached 0 or not. Whether this count is 0 or not 0 checker will use here, count=0, okay? What else do be require in this algorithm and the datapath shift and add, we will come back to it, okay?

This looks like this is more or less what we require, we require a controller now. So controller for shift add base multiplier. So now again we see that we are very typically going to have one state in which the system will stay for long enough time knows what to do depending on the status signal generated by the datapath itself. The controller will use those status signal generated by datapath, and like decide on the control signals to be input to drive the datapath again.

So looking at a status signal which to control and generates the control signals to the datapath and that is state is denoted by typical state loop.

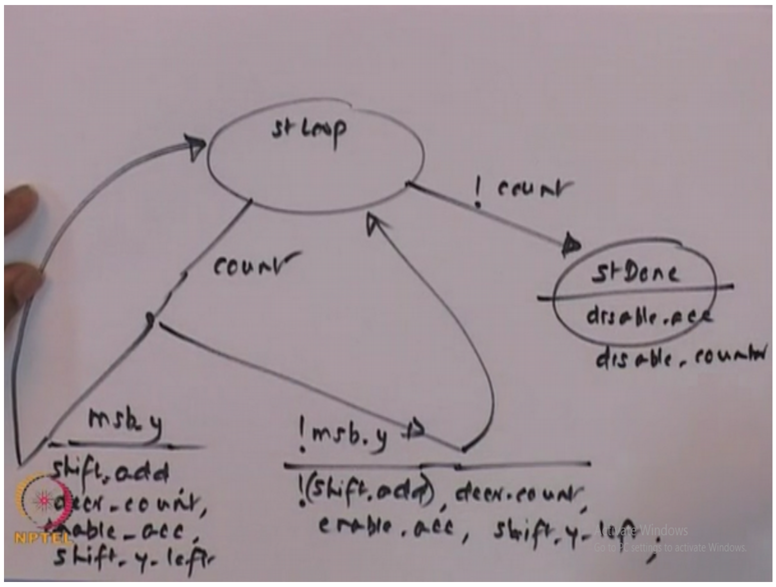
(Refer Slide Time: 33:28)



Which is where majority of the work is going to happen. Most of the cycle the system would stay in this state and what are the actions? The outputs would depend on what the inputs are. So most important input to the controller is just you can see there is a control signal shift and add. To decide whether to shift and add and anything else it will a lot will depend whether the count is 0 or not as well as whether the yes, we missed one thing right?

MS device the shift register for Y, okay and this is the MSB. As in the resistor, okay. So the MSB of Y the most significant by a bit of the shift register is going to be of crucial importance and based on that actions will be decided. If MSB of Y is non 0 and if MSB of Y is 0, these are the 2 possibilities. If MSB of Y is non 0 then we are going to go back in this yeah actually we missed the important the count whether count is = 0 or not even has higher priority. So this will not work.

(Refer Slide Time: 35:33)



So, quickly like from this state we check whether count is 0 or not, if count has dropped down to 0 we know that this computation is finished, okay? Let see what to do whether to go back in this state or not. This is important this is when the computation continues. If count is not 0 then it would depend on whether MSB of Y is non 0 or whether MSB of Y is 0, okay? If MSB of Y is non 0 then we know that, not only we had shift but also add X.

So we asserted control signal shift add to the datapath, okay? We should also be decrementing the count, right? We should also be loading the accumulator with the result of the adder, so enable accumulator to load itself and 1 more control signal will be shift Y to left. So all these control signals are asserted so that the multiplexer lets the X go through to the adder, So that X is added with the shifted version of the accumulator and result and count is decremented. The result of the adder is loaded into the accumulator and Y is shifted left. So, all this with the help of this control signals at the end of this current clock cycle appropriate datapath operations will happen, okay, and then we go back to this state, okay? On the other hand, if MSB of Y is 0 then we only we do not shift,

we do not add so which is like so it does not mean that we do not shift simply means that we do not add. So this signal is de-asserted because we have decided to use the signal with the name shift add as the input to this multiplexer. If shift add is 1 then we let X go through if shift add is 0 then we let 0 be added that means it has the effect of only shifting, okay. So shift add is de-asserted when MSB is 0.

But we still have to decrement count enabled the accumulator for loading and shift the Y, shift register left by 1 bit and we go back to this state. Now let us come to this particular situation when count was found to be 0, so should we what are the control signals to be generators control or some output signals to be generated and this state we go to next. So, if like in the single state GCD controller.

If we attempted to go back to this state, then what would happen is that the computation would have been finished but nothing could stop from this like shifting of accumulator and all that to happen because the system will keep reacting to, might react to like the date about the value of Y is MSB Y is 0 or not might change the contents of accumulators so which is what we do not desire.

So if we want the system to freeze, the freeze the results in the accumulator. Fortunately, that was not a concern in the single cycle because once Y become 0 then even if swapping and even if that like subtract operation happens continues to happen X does not get changed because X-Y its will happens to be the same as the old X. In this case this like, this accumulator contents might get shifted, corrupted and so on.

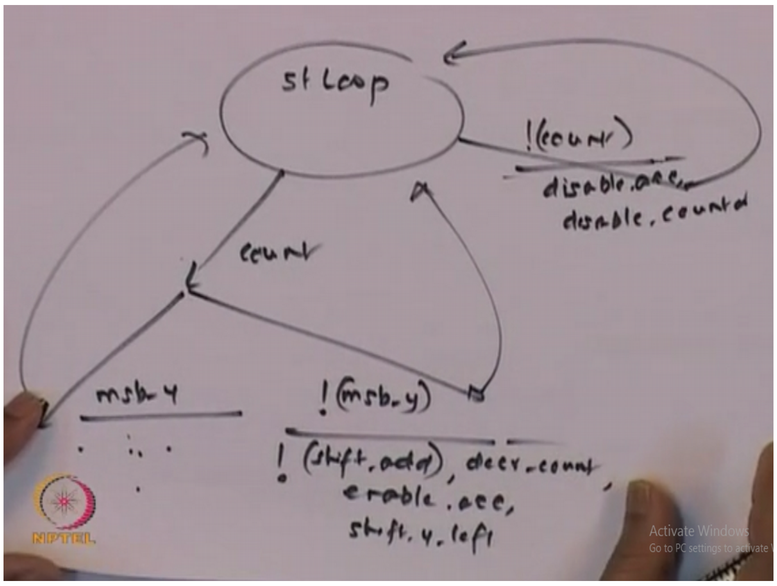
So, that is why we want the system to freeze when count becomes 0. So, we will go to us, will make the controller go to a state called state done, okay and stay there until the next trigger. So here I am just like just avoiding the role of the ideal state and all that. You can enhance this particular FSM with some more states for appropriate protocol of start triggering the computation of like indicating readiness of the result and so on.

But essential the main part, core of the FSM is this. Here I am just illustrating that I do not want to use the same state even when count become 0 because the count becomes 0 like go to the same state, yeah I mean it is a simpler more typical approach of having a special state for indicating the end of computation. So, in this state all the signals would be set to in such a value that the system would freeze, okay?

Accumulator would not change Y resistor will not, unnecessarily keep shifting and so on. So, quite typically I said we would like to have a state in which when the control is in that state we know that system is the computation is done. So, when count becomes 0 we move to that state in which we can disable the accumulator from further change like loading something into itself.

We can also disable the counter so that it will remain at 0 so that even if we have where to decide to go back here the count would have remained at 0 and so on, okay?

(Refer Slide Time: 42:09)

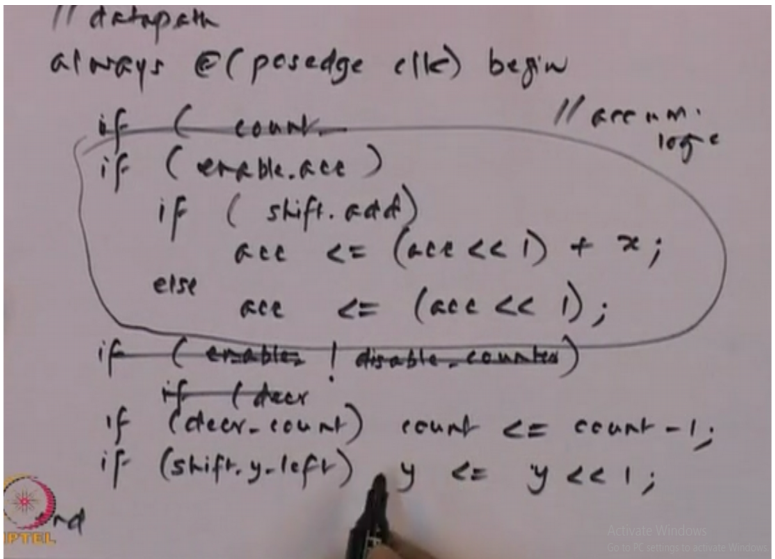


Or as the matter of test you might prefer a variation of this instead of this might feel that this could suffice as it is if count is not 0 it will depend on MSB Y and lot of this signals are specified here if MSB 0 then just shift do not add decrement counter, enable accumulator and so on so forth what else shift Y left and similarly here, we go to this, we go to this. If count where 0 then we might prefer to use a single state machine, of course it will have some inadequacies but it is not wrong or it like it can, we can still make it work by like generating the control signals as follows. We disabled the accumulator, we disabled the counter. Once we disable the counter the count is going to stay at 0 and the system will keep moving in will keep doing this transition and would keep, continue to keep the accumulator disabled continued to keep that counter disable.

In which case we know that system would not escape into this count will not get modified to something, count will not get updated to non 0 and then based on MSB some accumulator loading or Y shifting will happen. So, with the help of this kind of control signal values we will ensure that system will loop harmlessly in this state with everything frozen effectively. Or if you prefer we might do it that,

this particular we have special state just the way we had a special state ideal to indicate the beginning of computation waiting for some trigger. We might we should also have this, this is more prefer we have doing it, okay. Clearly, if you are trying to make it purely combinational there should be some protocol problem not really a problem but something not so clean, okay?

(Refer Slide Time: 44:42)



```
// datapath
always @(posedge clk) begin
  ~~if ( count~~                              // accum.
                                                 logic
  if ( enable.acc )
     if ( shift.add)
         acc  <=  (acc << 1) + x;
  else
         acc  <=  (acc << 1);
  ~~if ( enable~~ ! ~~disable_counter~~)
     ~~if ( decr~~
  if (decr_count)  count <= count - 1;
  if (shift.y.left)  y  <=  y << 1;
```

So let us describe the datapath in Verilog. So, this will also consider it understanding of how Verilog semantics capture the datapath as well as the controller. So the datapath, datapath is synchronous system, right? So what is happening if let us make is of single cycle controller if count is not 0 or other like, okay so the logic for updation of. So, after having seen the datapath and this simple finite state machine base controller for shift add base multiplication.

We will just capture this in Verilog just to consolidate our understanding of how the semantics of Verilog and how the synthesizer will map it into the circuit that we have drawn. So, let us first look at it this is the controller and let us look at the datapath. You see the datapath, what are the data storage components on the datapath there is an accumulator.

There is a resistance inside the counter, which is being updated there are flip-flops in the counter which are basically memory synchronous memory elements. These also a register X which is fortunately not getting updated maybe if we have a start state ideal, I mean that is the time X will be loaded from the inputs and there is a shift register for Y okay, whose most significant bit is being named MSB Y, okay?

So, we have to see what kinds of actions are taking place on Y accumulator and a counter. This will capture this in Verilog. So, this is how it would look like, always at this is the description of datapath in Verilog always at positive edge of the clock that is a triggering edge of the clock, if the control signal says for enabling accumulators says yes, that is there must be, we assume that there is a control signals controlling whether the accumulator can be loaded or not, we call it enable accumulator.

If enable accumulator has been asserted by the controller then we would look at the output of the controller which is the control signal shift add, if shift add is asserted is true then accumulator is to be loaded with shifted version of itself +X. So that means we have to compute the addition of the shifted version of accumulator, and X and load it into accumulator. That is this particular thing.

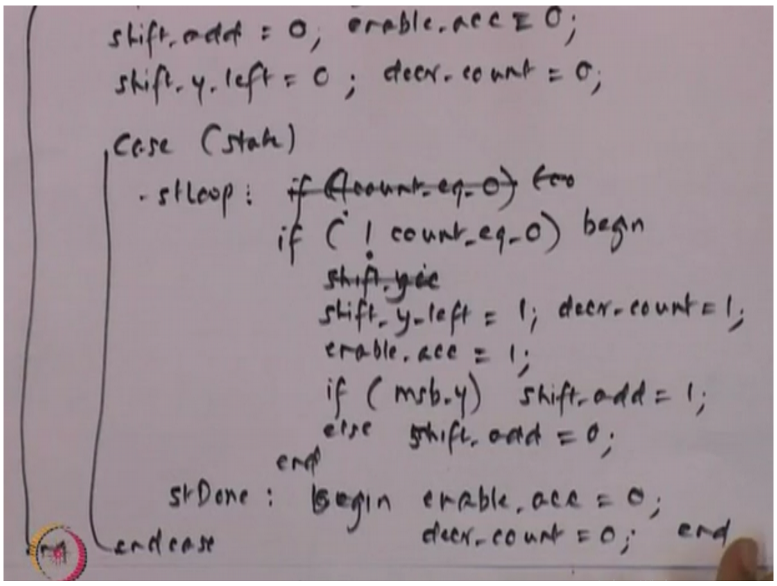
If this accumulator is to be loaded in this current clock cycle, then it would depend on what the value of shift add is if this shift add is 1 then X is to be added with the shifted version of accumulator and loaded back into accumulators. That is this particular statement, okay. If enable accumulator if shift add then accumulator is loaded this way, else accumulator is loaded only with the shifted version of itself.

So, if shift add is 0 then 0 is routed to the adder nothing happens just the shifted version of the accumulator is routed back into the accumulators. So, effectively means accumulator has been shifted by a single bit, okay? So, that is the logic for how accumulator is updated. So, that is the logic in this particular block.

Okay, now about the counter of course this is not complete Verilog code but main part of is this if decrement counter signal is asserted by the controller then the counter will decrement. Otherwise counter will stay at its own value it is like it will have the effect of disabling the counter. Similarly, the shift resistor is to be modified if is to be updated if the control signal shift Y left is asserted then shift resistor is shifted left by 1 bit, otherwise it stays.

So, this shift Y left is a signal say send to this, this particular shift resistor. Shift add signal is generated by the controller and is controlling this multiplexer enable accumulator signal is generated by the controller is controlling the loading of accumulator and decrement count another output of controller is controlling the counters. So, counter is also in part of datapath and it is been control by the FSM, okay?

(Refer Slide Time: 50:23)



```
shift_add = 0; enable_acc = 0;
shift_y_left = 0 ; decr_count = 0;
case (state)
  stLoop : ~~if (count_eq_0) ~~ ~~[illegible]~~
           if ( ! count_eq_0) begin
              ~~shift_y_e~~
              shift_y_left = 1; decr_count = 1;
              enable_acc = 1;
              if ( msb_y)  shift_add = 1;
              else  shift_add = 0;
           end
  stDone :  begin  enable_acc = 0;
                   decr_count = 0;  end
endcase
```

Then these remains to understand how the controller would be generating the control signals which are those, those are the signals shift add, enable accumulator, shift Y left and decrement count, okay. So, it is the combination logic for describing the output of the controller. Initial these are not it is initial, these are the default values which are going to be changed appropriately in the appropriate conditions.

So, if the current state is state loop then if the count is not equal to 0 that means that is the main part of the loop. Then we will be shifting arranging to shift Y, we will be decrementing the counter at the end of this clock cycle. We will also be enabling the accumulator for loading at the end of this clock cycle. So all these signals are to be asserted and but about shift add it would depend on whether MSB is Y, MSB Y is non 0, if MSB Y is 1,

then we will be like performing addition of X to the shifted version of the accumulator that is controlled by the signal which is the input to the multiplexer else the shift add will be set to 0 which would mean that 0 will be send to the adder and which will simply have the effect of only shifting the accumulator by 1 bit, okay?

In the state done we forces enable accumulator signals to be de-asserted, which means that accumulator will not be loaded whatever might be happening in the adder at the end of the, at the output of the adder or somewhere else. Similarly, counter will not be frozen by setting decrement count to 0. So, counter will not be decrementing, we hope that counter will also not be incrementing.

We will have, we just assume that we have a specialist counter which only decrements or freezes so and so forth. So, this is an always block describing the combinational logic for output, okay? Output logic of FSM which is combinational which depends on current state and the status inputs from the datapath like whether count is 0 or not whether MSB of Y 0 or not, right.

And we know that these signals have been generated whether count is equal to 0 or not is generated by the taking the output of the counter and having a 0 checker. MSB of Y is taken directly from the most significant flip-flop of shift resistor of this, okay? Yes, so this completes our description, I mean this can be easily wrapped up in a module and appropriate declarations of signals.

As an exercise you might want to upgrade this controller. Yes, you might want to upgrade this controller like in the case of GCD to have an ideal state, to have a start trigger, and to be able to load X new values of X input and Y input into X and Y, okay. So, this is assuming that at like on reset itself we have something in X resistor and something in Y resistor. That is good enough only for testing but as a subsystem you will need like, appropriate input triggers, so like typically you can make use of on state, left state, idle or state start.

Then this is the main state and this is the ending state, okay. And from here again you will go back to that ideal state waiting for start trigger and so on. So, this can be like enhanced into more like better protocol for using a multiplier system, okay that just left as an exercise we have tried to illustrate the main points and I hope that should suffice, okay.