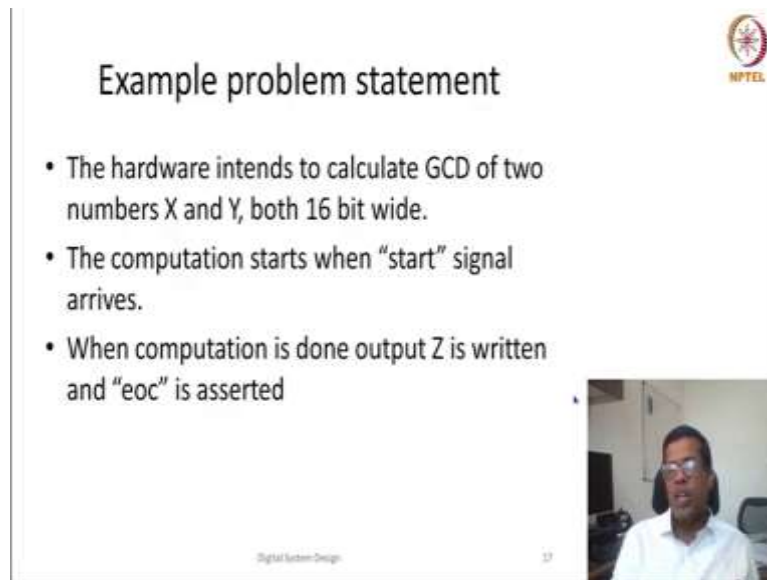



Digital System Design
Professor. Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture No. 62
GCD Computer at RTL Level

(Refer Slide Time: 00:14)






Example problem statement

- The hardware intends to calculate GCD of two numbers X and Y, both 16 bit wide.
- The computation starts when "start" signal arrives.
- When computation is done output Z is written and "eoc" is asserted

Digital System Design 17

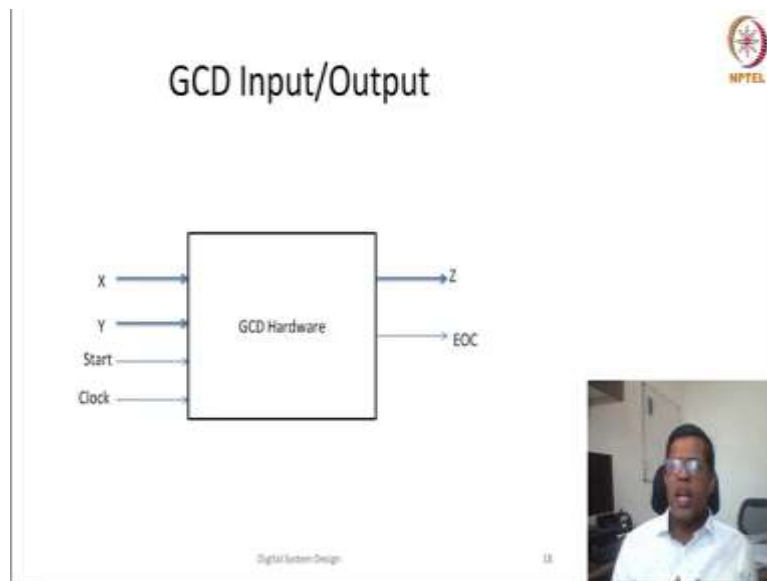


So, now the example what we are taking is, let us try to design an GCD Greatest Common Divisor. We would like to do this computation and we would like to model it in RTL way. So, GCD is usually is calculated by for two numbers X and Y, let us say and we consider that there could be 16 bit wide. So, it does not matter. Actually, it could be 32 bit wide or 16 bit wide so that would determine the data path, but otherwise, the design process is going to be seen.

So, along with that, let us also say that when 'start' signal is arriving, then only competition will start, and wherever competition is done, then there will be 'eoc' end of computation would be asserted or that means that would be 1. Otherwise, output will be Z, Z means high impedance.

So, when start would be there, then only competition would start and then X and Y would be taken as ainput and would help us calculating the GCD. And GCD would be returning Z. So, output is Z, it is not a impedance, but output a Z, but it would be read only when EOC is done. So, if there is an external circuit, external circuit will check if EOC signal is there, then only it will read the value of check. I am saying again, Z is not the high impedance, but it is the output, name with output variable.

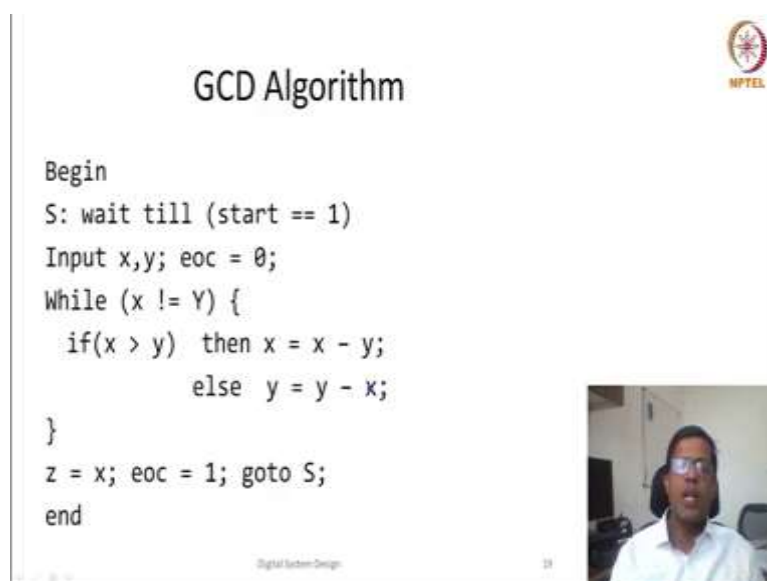
(Refer Slide Time: 01:52)



So, the overall circuit, we can draw it like this, that X and Y are the input both are 16 bit wide. And there is another signal which is start, which is 1 bit and there is a clock which is taking, which is deciding to internal operations of the GCD hardware. Z is the output and end of the computation is another output. So, Z is also 16 bit and end of computation is 1 bit signal. So, what we have done.

So, from the computation, from the problem statement, the first thing we have done is we have tried to identify very clearly what are the inputs, what are the outputs and what are their bit width. So, this is quite an important task whenever we get such an assignment.

(Refer Slide Time: 02:38)



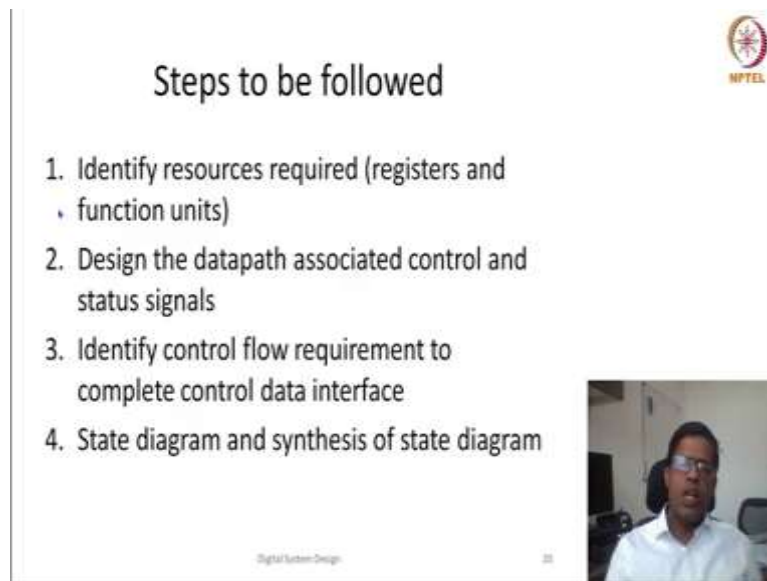
So, after this, let us try to write some kind of a high level language or high level statements for doing this particular task. So, you see that this is done at an algorithmic level. So, the code looks very similar to C, C plus plus. So here, it is not at register transfer level, but it is done at the algorithmic level. So, by doing things at algorithmic level, we are at least clear what the process would be then we can go ahead and decide things into that which all could be put into different states.

So, algorithm we can say that it would start S is a level. So, S says that until start is 1 we are still waiting. So, that means we are still at the S level until start equal to 1. If start is 1, then only we will go to the next line otherwise, we will keep on moving on to this particular line. So, when S start will become 1, then we will take input X and Y . So, basically X and Y would be read maybe in some registers and end of computation has to be assigned to 0 at the time.

So, and after that, this is the algorithm which we are considering when X is not equal to Y , if X is more than Y then we are saying X equal to X minus Y and 1 so that means Y is greater. So, we are saying Y equal to Y minus X . There are multiple GCD algorithms which are popular. So, the fastest algorithm for calculating greatest common divisor is by dividing, but here instead of division we are using subtraction because subtraction is a faster operation than division. So, we are assuming, so that is why we are taking this simpler operation rather than so it can still be done using division but.

So, then X and Y , X is not equal X is not greater than Y . So, that means at some point of time, X would be to Y . So, when X would be equal to Y that means, we have finally calculated the GCD. So, at that time either X or Y would be equal to GCD. So, at that time we can assign Z equal to X or we can assign Z equal to Y . So, X because X and Y both are same so it does not matter. At the time, we can also say that end of competition is 1 and then we can go and wait on to S again. So, after writing the algorithm, what shall we do?

(Refer Slide Time: 05:34)



The slide is titled "Steps to be followed" and features the NPTEL logo in the top right corner. It contains a numbered list of four steps:

1. Identify resources required (registers and function units)
2. Design the datapath associated control and status signals
3. Identify control flow requirement to complete control data interface
4. State diagram and synthesis of state diagram

In the bottom right corner of the slide, there is a small video inset showing a man with glasses speaking. At the bottom of the slide, the text "Digital System Design" and the number "22" are visible.

So, there are 4 steps which we can follow. So, first step we can say is that we will identify what all resources are required. So, resources means hardware here, and it essentially means what all registers are required, what all memory units are required, or what all function units are required. And in those function units, whether this is addition, multiplication, subtraction, or shift operations or decoders, multiplexers, all those resources we will try to identify.

And then, we will make an overall data path. Overall data path means that here we see that what all operations, what all resources would be required. So, we see that we would require some sort of a comparison operation, and we would require a subtraction operation. And we need to read to the register, we need to write to the register. So that is the overall operations. So, how many registers we may require that is usually we cannot identify at this moment, but here things are simpler. So, we can see that 1, 2, probably 2 registers from where we can read and 1 register, where we can write.

So, these 3 registers maybe sufficient and operations function unit means we require comparison, we require subtraction. And then we also have to design a data path. So, this may not be concrete at this moment, but slowly it will get concrete after we have designed the state machine that which all possible paths are there from the input to output. So, let us see in a moment.

And after that, so, if we have designed the data path, then we can also see that what could be the control signals requirement. So, control signals requirement means that which, so, you will that also we will see in next slides. So, along with that, we will also identify that which all operations would be done in which particular clock cycle.

So, that sequencing of operations would help us in drawing the state diagram, and then that state diagram could be synthesized. We can determine how many flip flops we require and the next state writing these next state equations, then we can also find the combinational units which are required there.

(Refer Slide Time: 8:07)

1. Resource required

- 3 registers to store x, y and z
- One comparator
- One subtractor
- Multiplexers

NPTEL

Digital System Design 21

So, for this example, we have seen that there would be 3 registers X, Y and Z where there would be 1 comparator, 1 subtractor and there will be some multiplexers which would be required.

(Refer Slide Time: 08:19)

2. Data path

start, X, Y, EOC, EOC, Z

R1, R2, R3

Comparator, Subtractor

NPTEL

Digital System Design 22

So, after keeping them, so we can say that, we said X and Y are the input start is the also an another input. EOC is the output and Z is the output. So, we can see that there would be 3

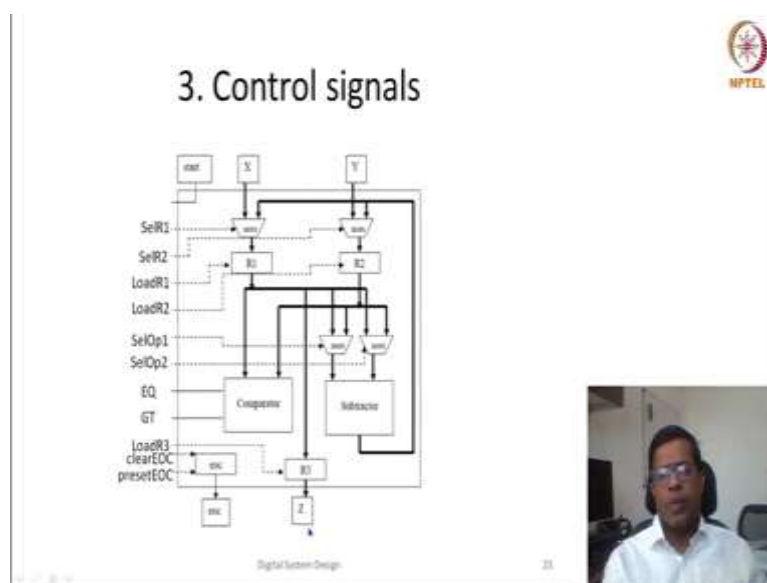
registers at least which are required one is R1, another is R, which would be able to read X and Y and let us consider that another additional R3 which would be finally writing the result Z. So, then you will question that what kind of, why we have put in so many multiplexers and etcetera. So, you will see that sometime we are this, if you see here, this X is getting updated from X minus Y and this Y is also getting updated from X minus Y.

So, let us say if R1 represent X and R2 represents Y. So, that means that this R1 should come from X as well as from the output of subtractor. Similarly, this R2 which would be selected either from the Y or from the output of the subtraction. So, similarly, we see that the subtraction could be done some time X minus Y sometimes Y minus X.

So that means, we there should be a path for the first input could be either R1 or R2 so, there has to be a multiplexer here. Similarly, the second input can also be either R1 or R2 so another multiplexer would be required. However, for comparison both R1 and R2 could be fixed that this is the R1 corresponding to X this is the R2 corresponding to Y. So, we are always comparing X and Y. So, there is no multiplexer required here.

So, this is how we are putting different multiplexers, and this whole is called data path. Now, if you see these data paths carefully so which means that the, if this is the register R1 we see the worst case logic would be R1 is going to this multiplexer and then going to subtractor and after that getting to another multiplexer and then getting written to a register. So, that means, the worst case path is going to be two multiplexers delay of two multiplexers plus delay of a subtractor.

(Refer Slide Time: 11:02)

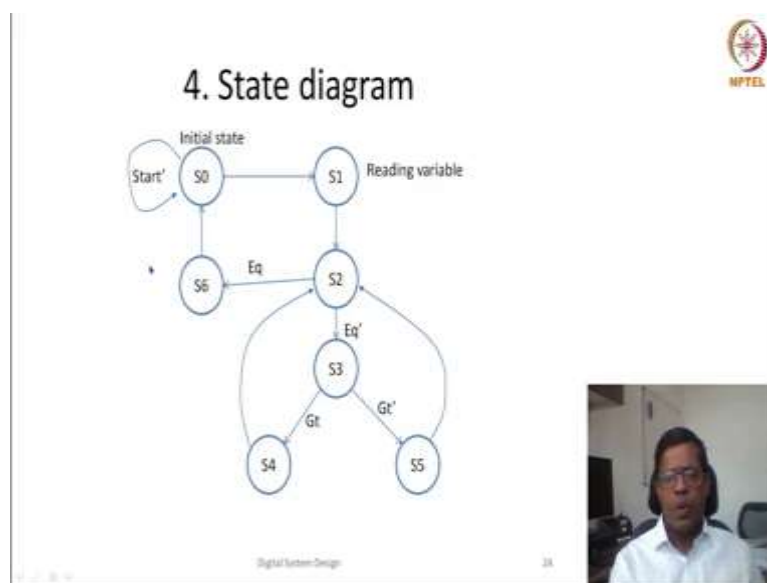


So, now, along with this data path, there would be certain control path signals also, which would be given by state machine. Now, somebody will tell us that when X would be selected and when the output or the subtractor would be selected. So, that means, let us consider that select R1 is a signal which is required to be generated for this mux and similarly, select R2 is a signal which is generated for this particular mux, so which is deciding whether Y is selected or whether the output of subtracted is selected. So, whatever is the output, whatever is the control signal they select R1 and select R2 based on that the values would be returned to R1 and R2.

Now, there will be another signal here load R1 and load R2. If load R1 is there then only this value would be returned to R1 otherwise R1 will retain its previous value. Similarly, if load R2 is there, then only R2 would retain, otherwise R2 would retain our its previous value. Similarly, here it will decide whether this mux selecting R1 or R2. This mux is also selecting either R1 or R2. So, Select Op 1. Op 1 here is operant 1 and operant 2. So, operant 1 is being selected here operant 2 is being selected here.

So, comparator would generate let us say two outputs one is equal another is greater than. So this equivalent greater would be given as input to my state machine. So, EOC, let us say there are two signals one is clear EOC and another is pre-set EOC so, that means this is a simple register which is asynchronous we can say which is if clear EOC is there and it is marked to 0 if it is preset it is marked 1. And similarly, if load R3 is there, then only a registered R3 would be written otherwise it would not be written. So, now, let us see the state machine also.

(Refer Slide Time: 13:08)



So, in the state machine, we are starting with the initial state S0. So, from the initial state if start is not there or basically start is 0 it is still rotating to state number S0 itself. So, if start is there, then S0 is moving to S1 state. In S1 states we are reading the registers, reading the variables and then in S2 state we are comparing. So, here the comparison would be done. And after comparison there would be, if it is not equal, then we are again comparing. We are saying that whether it is greater or whether it is not greater.


If it is greater, then we will again go back here and during this S4, we will do the operation of subtraction, X minus Y. Here if it S5 is there, then we are subtracting Y minus X, and then going back to S2. So, we are segregating here going from S2 to S3. Otherwise you may ask that both of them are comparison operations. So, why not to perform S2 as well as S3 in the same state. So, that would have been possible?

So, that would have been possible, but for the sake of simplicity right now, we have we are putting S2 and S3 as a different states, although, because the operation in both of them is performed as comparison so which is done by same operator. So, there could be 4 outputs here. So, whether it if is greater than it could have been gone here. So, in that case, we can we could have merged S2 and S3 also. So, but if it is equal to then we are going to S6 state. In S6, we are finally writing, we are saying that results would be written. And after writing the results, we can go back to S0 state.

(Refer Slide Time: 15:15)

4. State diagram

State	Operations	Control
S0		Reset all
S1	Copy X to R1 Copy Y to R2 Assign eoc = 0	LoadR1, LoadR2, SelR1, SelR2
S2	Compare R1 and R2	
S3	Compare R1 and R2	
S4	$R2 = R2 - R1$	SelOp1=R2, selOp2=R1
S5	$R1 = R1 - R2$	SelOp1=R1, SelOp2=R2
S6	$R3 = R1$	PreSetEOC



So, these operations are being written here that S0 no operation is done in S1, X, B is being copied to R1. So that means, load R1 would select X, and will write it to R1. And along with that, so, basically both of these loadR1 would be set to 1 and Select R1 is also set to 0 such

that X is selected. Similarly, select R2 would select the would select Y and will also unable load R2 so that R2 would get written and this Y would be written to R2.

Also EOC would be assigned to 0, so that means that clear, EOC would be would be asserted during this S1 state. All of these are the output signals of S1 state. So, during the S1 state all of these signals need to be generated and controlled. So, S2, there is no control operation, this comparing R1 and R2. So, as I said S2 and S3 although could we merge so because we are performing the same operation again and again.

So, in S4, we are doing this operation R2 equal to R2 minus R1. And so, to do this operation, what we are doing is we have to select operation 1 as select OP1 as R2 and we have to select Op2 as R1. While in S5 state, we have to generate select Op1 as R1 and select Op2 as R2. So, this has to be selected, so that this operation can be performed. So, in other words, what you will see that the structure is always there, but because my state machine is saying that I am in S4 state and because of that R2 is selected and R1 is selected so automatically this operation would happen.

And when this operation would happen, this will also decide, yeah, so which will also say that this R1 will get updated. So, there is also one more thing which has to be written here thatso along with that, we also have to select that R2 has to be returned load R2 has to be there and in this S5 state load R1 has to be there. So, that finally when this value is computed, R2 will get updated in this case R1 will get updated. And which also means that we have to select this R1 register, we have to select this R2 register also according to this R2 and R1.

So, these are the signals which are asserted according to in which state we are and finally in S6 state R3, R1 would be assigned to R3 and preset EOC would be asserted. So, this is the final design and using this final design. So, you see that all these structures are always there, but because in a particular cycle that particular, if we not write in S5 state R1 value this R1 value will not get updated, R1 value will remain the same all the time.

So, this is how the same structure is able to compute different things in different clock cycles. And because different things are computed in different clock cycles so we can perform different kinds of overall application using the same structure by just generating different state machine and these signals could be generated according to the state machine. So, this gives us at least some idea that how an RTL level design is done, and how the states are being identified.

So, one step which we have quickly bypassed here is that we decide what all operations need to be performed in single clock cycle. So, here because my state machine was simple, and the algorithm was simple, that only one of the operations were performed at single time. So, let us take some more sophisticated algorithm in our next lecture, and then we can see that how the steps could be done in in that in that exercise.

(Refer Slide Time: 19:58)



So, with this, I would like to close today. And I can summarize that today, we have seen what is an RTL level design, what is the significance. Why do we do it, and what would be the implication of clock period and what all information we will finally get if it is RTL level design. And we have also seen it with one example that how RTL level design can be done. So, with this I will close. And thank you very much.