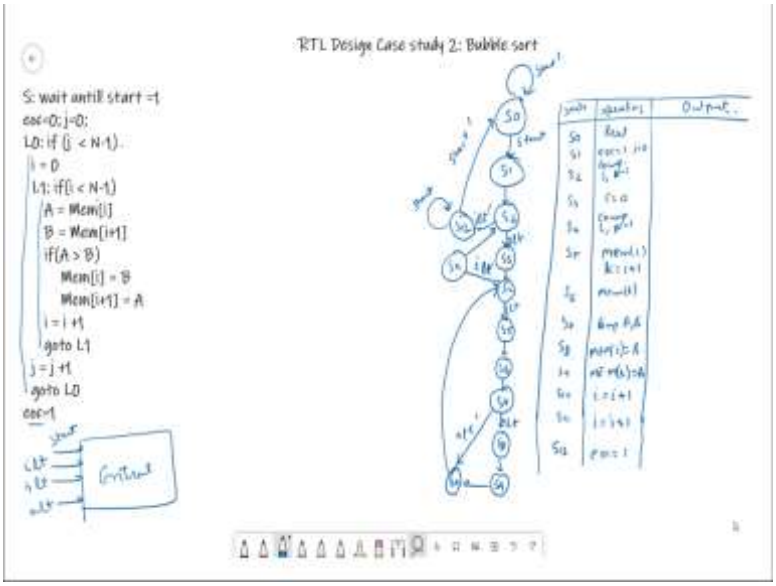
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology Ropar
Lecture 63
RTL Design – Bubble Sort

(Refer Slide Time: 0:14)



Hello everybody, today we are continue, we will continue with RTL design and we will take more sophisticated problem of, so, that we can understand how to divide operations and how to divide algorithms into smaller steps and create a data path and control path according to it. So, the problem we are taking is a bubble sort.

So, all of us know that bubble sort is a popular algorithm and why we are taken this algorithm, so, that we can understand that if we can do such kind of an algorithm then essentially anything, any algorithm can be designed in RTL or in hardware, we follow the if we follow similar steps. So, this is the idea of taking bubble sort. So, let us go ahead and see how does it work. So, this is the algorithm for bubble sort.

So, in the first line we are saying S wait until start equal to 1, so, until start equal to 1 will still remain on to this particular line and then we will say end of computation equal to 0 j equal to 0. The first loop if j is less than n minus 1 then we will continue like this i equal to 0 and the second loop L1 if i is less than n minus 1, then we will start the second loop that means, it will start from i equal to 0 and you will keep on comparing the adjacent element i and i plus 1. If i and i plus 1, if A is more than B then we will swap, swap here.

So, because it is a hardware, so, we have divided these operations into smaller operation that means, this is a read operation and B i plus 1 is again another read operation. So, if A is more than B the content of i memory content, memory content, ith address of memory i is more than i plus 1th content, so then we will swap it.

So, that means in mem i it will become B and mem i plus 1 it would be storing A. So, with all of this process now, because of that you are in, by the end of this loop, your highest memory location will have the least element. Then in the end of the loop we will increment with i plus 1 and then we will go back to loop L1 and the, so this is the notation which we are taking that all of this is L1, this is all L1 and this is all L0.

So, then we will go back to L0, once all of this computation is done, then we can say end of computation equal to 1. So, if we had to design this hardware over using our digital logic then the first step is basically we have used the conventional bubble sort algorithm and broken into even smaller pieces like all the memory axis are defined as separate operations.

So, further i plus 1 should have been divided into a different operation because these two operations probably cannot be performed at the same time. So, the other thing you may notice that we are running both the loops with 0 to n minus 1, again 0 to n minus 1.

So, there could have been optimization where we could have taken the second loop to lesser locations, but from the hardware design perspective, design perspective, it does not matter, total number of cycles can be reduced if we take the another optimization. So, as the second step, let us first make the state table which would help us to do different tasks.

So, let me write here, my state table, let us say this is my S0 state. So, S0 state is my starting and initial state. If start signal is there, then we will go to S1 state. S1 state I will write here also I will create a table, so that we have certain meanings of these states also along with the, this is the, yeah, so this is my 0 state and this is what it mean S0 is a reset state. And then S1 means I will assign e o C equal to 1 here, and also we will assign j equal to 0.

So, from the S1 state, we will do will go here, so that means there is a S2 state. In S2 state, we are going to compare j and n minus 1. So, let us assume n minus 1 is available in one of the registers. So, n minus 1 and j would be compared. So, from S2 state, then there will be another state S3. So, if compare that means if it is less than then it will go to S3 and that means there has to be something if there is not less than.

So, if it is less than it will go to S3 and S3 essentially means that my i equal to 0. So, then, in S4 I will have, in S4 what I will do is I will in S4 I will again compare i with n minus 1. So, again, if the result of comparison is less than, then we will go to S5 state; in S5, what we are going to do, we are going to load memory i.

So, along with memory i, I can also compute i plus 1, let me have another register ij k, let us say k equal to i plus 1. So, if that is the case, then that means S5 is this and from S5 will move to S6; in S6 we will have memory of k. So, these are all sequential. So, S5 is there, S6 is there and then we will compare S7; in S7 we will compare A and B. So, this is S4 was also compare, this was also compare.

So, compare A and B from S7 then we will have S8 and S9 these would be there and then there would be, from there we will have S10. So, if in S7 we were comparing, if it was less than then it is going here, if it is not less than, then we are going directly to S8. So, S8 is actually mem i equal to B; S9 is mem k equal to A and S10 irrespective of that, we are always going to increment i equal to i plus 1.

So, and then we have this state. So, basically after S10. So, this is S10 from S10 we will move back to L1; L1 means where we were comparing i less than n minus 1, so basically there we were comparing i, so that means, it is S4. So, from S10 we are going to go to S4 great. So, the next thing is S10 to S4 this we have to still do, so that means, if it is less than we are going here and if it is not less than where shall we go?

If it is not less than in S4 it was not less than then we have to go to a new state where we are going to compute j equal to j plus 1. So, let us call it S11. So, in S11, we are saying j equal to j plus 1 and from there we are going back to S2. So, here after comparing if we are again comparing that whether this is S12, so in S12 we are so, basically in S2 state if j is not less than, so that means j is more than n minus 1, then we are going back to e o C equal to 1, so that is my last state e o C equal to 1.

So, from S12 if start is still there, then it will remain in S12 state and if start has become 0, then it will go back to S0 state. So, this is how the state diagram would look like and these are the various interpretation of different states. So, this is state and this is the basically interpretation and what data operations which are being performed.

So, after these operations are performed, now, the next step what we can do is the next step we can do is we can find the data path, what number of registers are required and how they

would be connected to each other. So, this state machine can essentially be implemented using our, the state diagram is already there, so this we can implement using state registers and you see the state machine is more or less complete because at S0 there are two signals start and start bar.

So, we know that if start is there, start if it is start bar, but then it will still remain in S0 state from S0 to S1 and from S1 it is always going to S2 state, from S2 state it depends on this LTE signal. So, let me call this as, there are 2 lt signal, so this is j lt. So, this is jlt signal and then it is always going to S3 state where we are going to make i equal to 0 and in S4 state this is i and this is ilt dash.

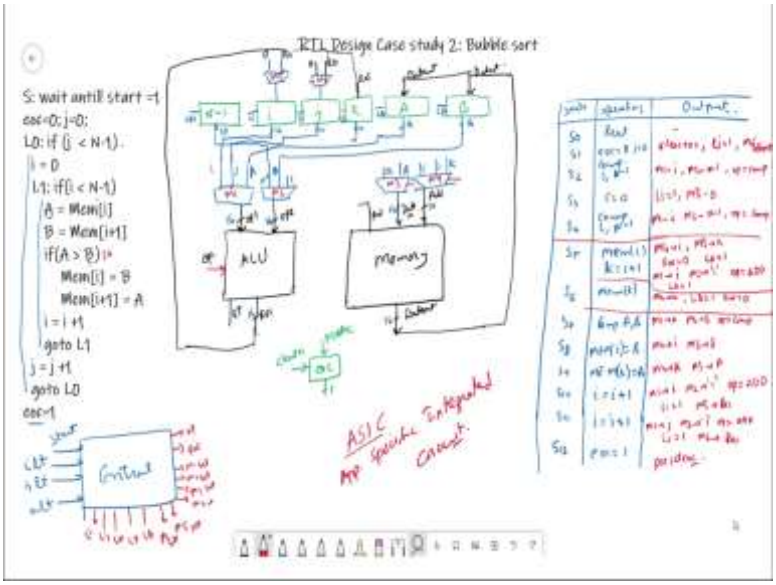
So, basically these are two different signals ilt dash and if it is ilt then it is going to S5 state, sequentially S6 then S7, S7 is again comparing A with B; so let me call it alt, this is alt dash. So, from there sequentially it is going to S8 then S9 and then S10. From S10, yeah, so from S10 it is going sequentially to S4 So, if A less than is not true, then from S7 directly it can go to S10.

So, from S8 also if it is i less than, ilt is not true, so that means ilt dash is true then it will go to S11, where we are going to do j equal to j plus 1 and then we will sequentially go back to S2 state and so this way our state machine is complete and from this state machine what we understood is that the information which is required for this state machine; input to this state machine is let us say this is my control signals.

So, for control 1 of the signal is start another is ilt, another is jlt and the third is alt. So, these are the three signals which, these are the four input signals which are required for the control and what kind of output we it will generate, so that we will write here. So, this whole process now tell us what is the synthesis process of the control part. Now look now let us look at the data path part.

So, for data path part what I am going to do is I am going to rub this because this is no more required, we just need to understand that what operations is being performed in each of the state and that is mentioned in this table. So, let us first erase this state table and let us focus on the data path.

(Refer Slide Time: 14:49)



So, now, let us see that what data path, for the data path we need to know what kind of operations are required and what number of registers are required. So, if we see carefully in this algorithm as well as these operations, the number of registers required would be let us put them in green. So, number of registers we would require is let us say we have this n minus 1 which is available as a register. So, we can, otherwise we can have operation here itself, which where we can compute n equal to n minus 1.

So, for simplicity, we assume that n minus 1 is available in a register. So, then we have, we require i, we require j also to be stored in a register and we are not defining another new register k which is actually i plus 1. So, these are the four registers additionally we also require A and B, so that operation can be done. So, n minus 1 i j k A and B these are the registers which are required.

So, what else is required what operations are required, the operations are required, we require, we need to have a memory. So, let us say we put memory in a black, so we require a memory unit. So, in memory unit there would be three signals; one would be address, this is a input and we require data in and there would be a data out and let us say there is a signal which we will say whether it is RW. So, this is my memory. Now, if RW is 0 then let us assume that it is read operation, if RW is 1 then it is right operation.

So, this is one the other we would require some sort of addition subtraction. So, basically we require addition in most of the cases and addition is required and then we require comparison. So, let us say we have one unit which is doing both addition as well as comparison. So, let me call it ALU. So, which is doing op1 and op2 it requires. So, these op1 and op2 are required,

then there would be a result which is also of the same direction and now there is a another signal which is lt which is generated because of comparison.

Now, why we have combined both comparison and addition here in ALU, because we know that addition as well as comparison both can be one using an adder and so in case of comparison we can use two's complement and then do subtraction and after subtraction the same bit can tell us whether it is less than or whether it is not less than. So, this was about all the operations which are required. Now, what are the size of these operands that also need to be fixed.

So, what is the memory required? What is the size of memory, what is the width, what is the, for that we need to assume that what is the size of A and B or what are the elements? So, let us say that all the data in is 16 bit, this is also 16 bit and how many memory locations we are sorting out, let us say we are sorting out some 1000 locations. So, in that case my address is going to be our 10 bit So, this operand 1 is also 16 bit, this is also 16 bit, my result is also 16 bit.

So, what about the registers? So, these registers i can be 10 bit, j can be 10 bit, k can also be 10 bit and A and B are going to be a 16 bit each. And because it is 1000 location, so n minus 1 is also going to be 10 bit; so these are the registers. What are the other signals required? So, for each register there is going to be a load signal. So, I will say load n, load i, this is load j, there will be load k, load A and load B.

Now, these are the various data path elements. Now, let us look at the connections and how things would happen. Now, for ALU if we see the ALU operation we would require i plus 1 is required here j plus 1, k plus 1 et cetera, so for this op2 one of the operand has to be 1. So, one of the operand has to be 1 and the other operand can actually, so can come from either it could be B.

So, other operand could be B, if we can find it out from here, so we are doing either we are using this ALU for comparison or addition. So, for addition purpose k equal to i plus 1, j equal to j plus 1 i equal to i plus 1, so in that manner it is only the second operand is either 1 or the second operand could be, when we are comparing A and B it could be B. So, that is the only operand and for the first operand there is a wide possibility, for first opened it could be n minus 1, sorry yeah.

So, here we were comparing with n minus 1 also. So, that means n minus 1 has to be a operand, second operand. So, this could be a second operand in this case. So, i could be here, j could also be an operand here, k could also be as the first operand k is not there, so we are always but A could be an operand. So, it could be n minus 1, it could be i, it could be j and it could be A also.

So, it could be i n minus 1 i and j. So, n minus 1 is not there in this, the first operand, so that also can be removed actually it is i. So, we will correct it again. So, basically this is i, this is j and this is A. Now similarly, for read write signal this will come from the control. Now, data in data, data in could only be either A or it could be B, there are only two possibilities. For the address, the possibilities are only three one is i, another is j, another is k. These are the only three possibilities.

And we would also require some select signals. So, we would require some select signals here, we would require some select signals here. So, to make it easier for us, so, let us call this as mask1, mask2, mask3 and mask4. So, these are the signals here. Now, let us again see that something would be required, n minus 1 does not require any information basically, it is once initialized then it will always remain contain its value.

So, i would be written by the result of this ALU, so either sometime, this result need to be written here. And same way in j also sometimes a result need to be written here. But initially there could be j equal to 0, so there has to be a mask, here as well. So, let us put a mask, so that things are easy. So, we will say that there is a possibility of mask here, there is a mask here also, one of the mask is selecting 0 and other is selecting from the result. So, similarly, this is selecting from 0, this is selecting from a result.

For k, it is only the result from where it can come, it is only the result and now for A, A could come either as a data out or as a A is always as a A data out, so that means this is always data out and this is always data out. Good. So far so good, so basically this is how these connections would be made. So, that means this data out value will go and will reach here as well as here and this result of comparison will go and who would be connected here, here as well as here. And like these M4 masks, let us have these masks also labeled as M5, M6.

Now, this is the complete data path. So, in this data path, we know what all registers are there what all function units, here we have 1 memory as a function unit 1 ALU where we have to tell whether this is, we have to tell whether this is what operation. So, this operation we have

to tell that whether you are going, we are going to perform comparison operation, where we are going to perform and increment operation.

Now, let us if we see all of these things now, let us see that what all, so yes, just a minute, let us complete this for a moment. So, the input here could be n minus 1, this could be A and this is B actually, the comparison could be B and with n minus 1 and sometime it could be with 1 also, 1 is required for this purpose. Good.

So, now after data path is complete, our state machine was already done. Now, we have to see what kind of result this control has to produce. So, this control in every state what it has to do is, it has to control only signals, for example, when this value should be loaded in register i li has to be enabled, lj has to be determined from this control, lk, lA, lB and similarly, all these five masks M1, M2, M3, M4, M5, M6, the six multiplexers has to be controlled by this control unit and control unit will also generate this op signal, it will also generate this RW signal.

Now, when any of these signals would be generated, let us see here. So, now, let us write the control part. So, when reset equal to 0 then there is no there is no output. So, that means all of these signals are supposed to be 0. So, when all these load signals are 0, which means that whatever is there in the register it will remain there. So, there is one more thing which got missed out here that there has to be another register which is e o C which is 1 bit and there has to be preset, preset e o C and also clear e o C. So, these two signals also would be given as input to this e o C register.

So, in reset state all of this ln, li, lj, lk, lA, lB, all of these are going to be 0 and whatever is the value of op, whatever is the value of RW, it does not matter and whatever are the values of all the masks control that also is not important. Because whatever we select no operation is going to perform, there is no register, so there is no change in the in the state of our computation.

So, in S1 state we have to clear the e o C, we have to say j equal to 0. So, that means, we have to, then what we have to do is we have to say clear e o C has to be generated along with that we have to say j n load of j, lj equal to 1 and M6 we will select 0. So, M6 we will select 0, so that is that signal has to be there. Now, with that this operation should be sufficient which will make our load equal to, so by the end of this S1 state our j register will contain 0 and by the end of this state, S1 state e o C would be equal to 0.

For the next state, S2 state we need to compare. So, because we need to compare j and n minus 1. So, that means my M1 will select, M1 will select j and M2 will select n minus 1 and operation would be equal to compare these are the three things and also as a result, so whenever this has to be written, so this value will not be written anywhere, because this is just a comparison, there is no result that will get written.

So, because of that, this is the S2 state, so in the S2 state we are comparing j with n minus 1 both the select signals are there, we are able to compare j with n minus 1 and the output lt would be used for our next state whether we are going to there in either in S3 state or we are there in the S11 state. So, either S3 state or end of computation state. Good.

So, for S3 again we have to make a load of i equal to 1 and also M5 need to select 0. So, load i has to be enable and M5 equal to 0, so also we have to note that whenever S3 or any other states is coming, so that means whatever signal which we have given in the previous state they will all automatically will become 0. So, all other, so that means only li equal to should be equal to 1 and all other ln, lA, lB, lj, lk, all of them has to 0.

So, then the next state is S4, in S4 what we are going to do is we are going to compare i with n minus 1. So, we are going to compare, first of all we have to say M1 need to select i and M2 need to select n minus 1 and operation has to be compared, there is no result need to be written. So, none of the l signals would be 1. So, after that in case of S5 state, we are using memory i.

So basically, that means first of all address need to be selected. So, that means M4 will select i, M3 will select, M3 is not selecting anything, because it is not a right operation. So, M3 does not matter, M3 is don't care. And then RW is equal to because it is a read operation, so RW is 0 and whatever is the value generated, so this data out has to be written to A, so that means lA should be equal to 1. So, whatever is the data out generated that should be written to A, so that means lA equal to 1, along with that, we are also performing k equal to i plus 1.

So, that means, this M1 register, M1 multiplexer, M1 multiplexer is selecting i M2 multiplexer is selecting 1 and the operation is going to be add and also this operation would be written to k, so that means lk is equal to 1. So, this is all about S5 state, so, this is all S5 state. Now, for the S6 state in S6 state, S6 state is B would be equal to mem k. So, that means, my address would be M4 will select k and M3 will not select anything, because it is a read operation there is no writing part being done. And now, l of B would be equal to 1, RW is equal to 0.

So, these are the operations here and so we can say this is a about S6 this the data which need to be generated. From S6 to then we will go to S7 state, in S7 state we are doing comparison, this is our S7. So, in S7 we are comparing A and B, so that means, my M1 will select A, M2 will select B and operation is going to be comparison operation, there is no writing happening, so none of the l signals would be 1 and based on the lt then my control will decide whether we are going to be there in S8 state or S10 state.

So, if we are there in S8 state that means we are going to write something. So, if we are there in a state S8 state, then B is written to i so that means my address is i, so M4 will select i and my data in so that means M3 will select B, there is no data out, so none of the l signals will get generated or they will not be 1.

Similarly, for the S9 state now we are saying M4 would be selecting k and M3 would be selecting A. So, the next is S10 state, in case of S10 state we are going to perform an add operation. So, M1 multiplexer will select i signal and M2 multiplexer will select 1 and operation is going to be add. And we are also going to write this value in i register, so that means li will become 1.

So, the value would be updated, li is 1, so in the next clock cycle because li was 1, so whatever is the value here it will get picked. So, because i need to be selected we also have to say M5 need to select i not the 0. So, similarly, for S11 state which is also similar to this, so M1 will select j, M2 will select 1 and operation is going to be add l of j will become 1 and M6 will select j. Here we are saying we will select Res, this is also going to select Res.

So, then all of these states are done and S12 state will say preset e o C would be there, so this e o C would be 1. So, now here if we see what all control signals outputs were there, they were li, they were lj, lk, lA, lB and along with that it was also selecting op, it is also selecting RW which was also selecting M1, M2, M3, M4 select, M5 select and M6 select, so all of these signals were the output of my control.

Now, if we, so this is almost the conclusion of this exercise that yes, if you want to design any of the algorithm in hardware using digital logic, so the process is more or less same and standardize that we can break the algorithm into smaller operations and after the smaller operations, so in our previous lecture we have first created the data path and then created the control path, today we did first the control path, then the data path, so anything can be done.

So, if we are doing control path first then we can also exercise, we can also see that what all operations can be done in parallel, for example here the operations the only in S5 state we utilized both memory as well as ALU at the same time, but in general there is a good possibility that there could be number of operations which can be done in parallel to make it faster.

So, after identifying all these states, then we have created the data path, first we have identified how many registers were required, then we identified how many resources were required, what kind of selection signals were there. So, for doing the selection signals, there is one approach; first approach we can identify that we can maybe connect all the multiplexers to all the possible signals, then we can prone it off, prone it off means we can remove, we see that what is required, what is important, so only those signals could be connected to multiplexers.

What kind of encoding to be used? It is up to us, so for example, as we have discussed earlier, so if we one hot encoding, so sometime this can be useful here in selecting whether we would like to select i or j or A or n minus 1 or B or 1, so all of these selections can be done based one hot encoding or we can use the compressed coding also. So, with this we can conclude that if we would like to design any algorithm using hardware, then we can do it.

One question which may be asked at this moment that, why somebody would like to do it? The question is more important in today's context that if whenever we do this algorithm, this algorithm would be written in some high level language C, C plus plus, Java, Python. And then we would have compiler and it would be running on some processor, processor is also an hardware.

If we are doing, if we are running it in a standard processor, why somebody would like to design this kind of a hardware which is specific for that application which cannot run any other application other than this. By the way, because this hardware can run only one application that is why it is also called application specific circuit, we called it ASIC, Application Specific Integrated Circuit.

So because it is application specific, so this integrated circuit, this circuit can perform only one application. So, if a hardware which can perform only one application, what is the use, what is the utility of such a hardware? Because processor is also a hardware which is capable of running any kind of application, we just need to compile a new application and then we can turn that binary code onto processor, it can run it usually fast.

So, the reason we would like to design ASIC or some application specific hardware, because many a times it would be faster than running it on a standard processor. Because it is a custom design, the wires which are here are not generic, they are specific, so there is a good possibility that the number of wires would be less and also the number of operations which can run in parallel can also be more, so the overall performance could be high.

So basically, overall execution time could be less in this specific hardware and the processors are called general purpose hardware. So, in general purpose hardware you can run anything and you can just intuitively think that if we can run anything, that means the hardware has to be general enough that it can run anything.

So, there would be redundancy in the hardware, sometimes those instructions, those function units would be there which may not be required. There could be registers which may not be required. So, that is the only difference, but overall the structure of your processor would also be very, very similar to this, that it would be reading from the register and it would be doing operation and then again writing back to the registers. So, with this I would like to close, thank you very much.