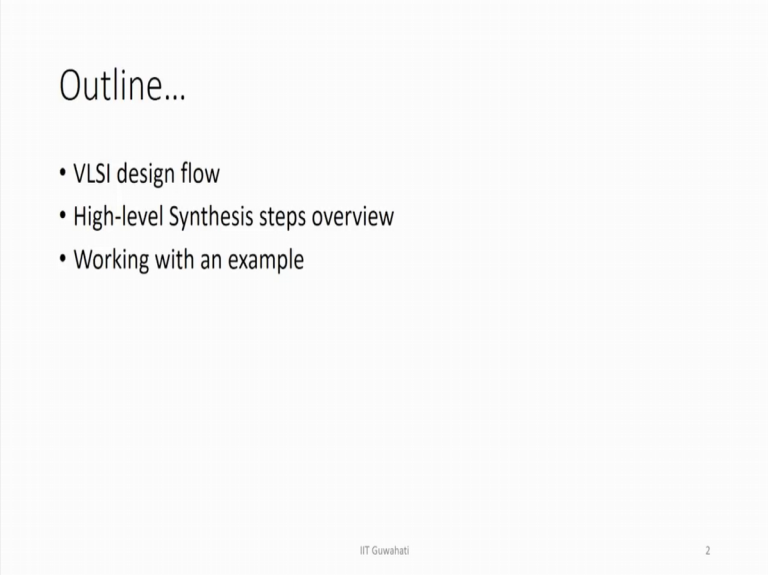
Optimization Techniques for Digital VLSI Design
Dr. Chandan Karfa
Dr. Santosh Biswas
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 02
High-level Synthesis (HLS) flow with an example

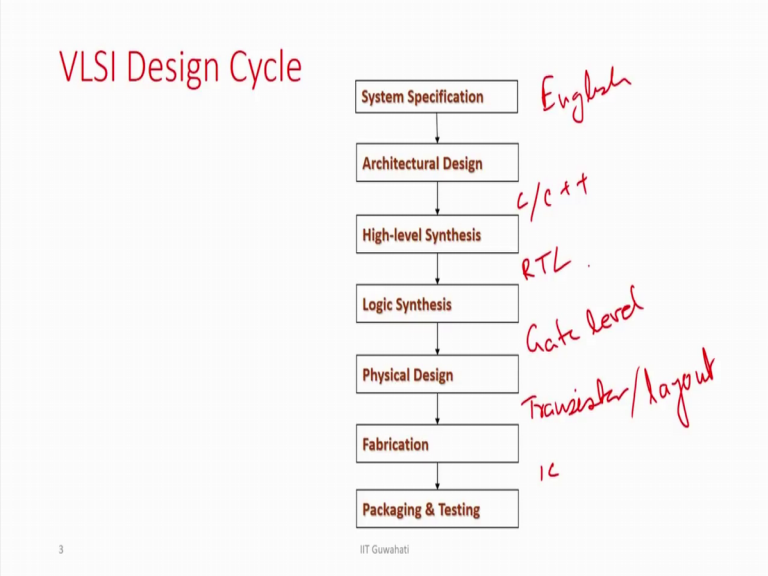
Hello everyone. So, today we are going to discuss on high level synthesis right. So, I mean as you saw in the last lecture is that we talked about this VLSI design flow, and you just saw the all the sub steps of VLSI design flow how we can take an high level behaviour and we can map that to I mean you know integrated circuits right. So, that is what we have seen.

(Refer Slide Time: 00:50)



So, in today's class we just recap that whatever we have discussed in the last class and then we just go through high level synthesis by primary with an example just to try to understand how what are the objective of each sub steps in today's class right.

(Refer Slide Time: 00:58)

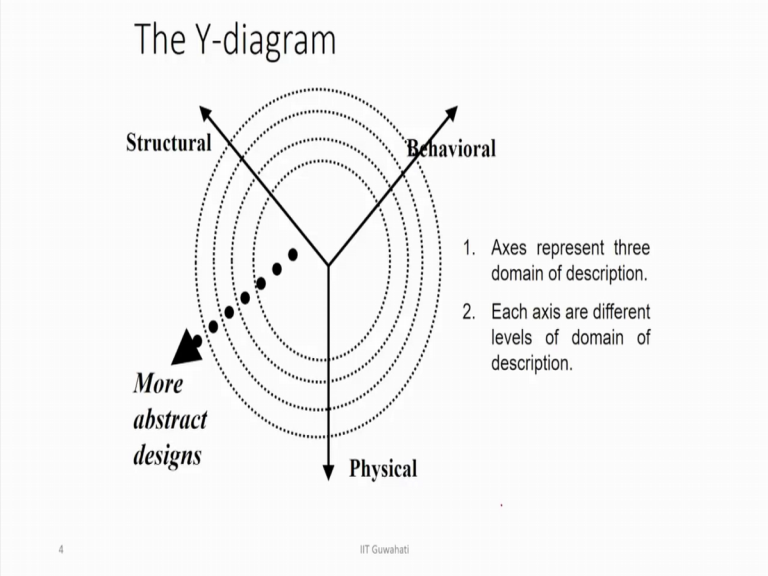


So, for if as you see in the last class is VLSI design flow is consists of several sub steps like high level synthesis, logic synthesis, physical synthesis then fabrication and packaging and testing right. So, and you can think about as actual things start from that English language, system specification then we generate that CC plus plus code from that manually primarily manually that part is not automatic.

And then we do high level synthesis to generate a register transfer level designs from a CC plus plus code, and then we do logic synthesis to get generate gate level designs, and then physical synthesis to generate transistor level design or the physical layout, and then you do the fabrication and packaging get the integrated circuits right and what is the objective of these cycles is, we try to design as abstract level as possible right.

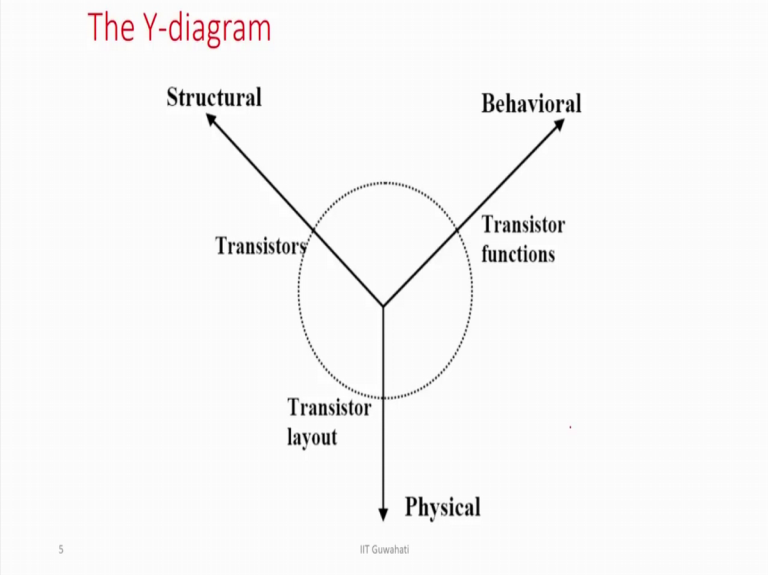
So, that we can design in quick time and we rely on the tools cad tool computer a design tools or EDA tools elect electronic design automation tools, to generate the next detailed level and that is the world objective so that, we can generate a big circuit in quick time. So, that is the overall objective of visual VLSI design flow, we try to design as abstract level as possible as quickly as possible and try to automate that steps, we use this synthesis tools to generate the next level of designs right. So, that is the overall objective.

(Refer Slide Time: 02:20)



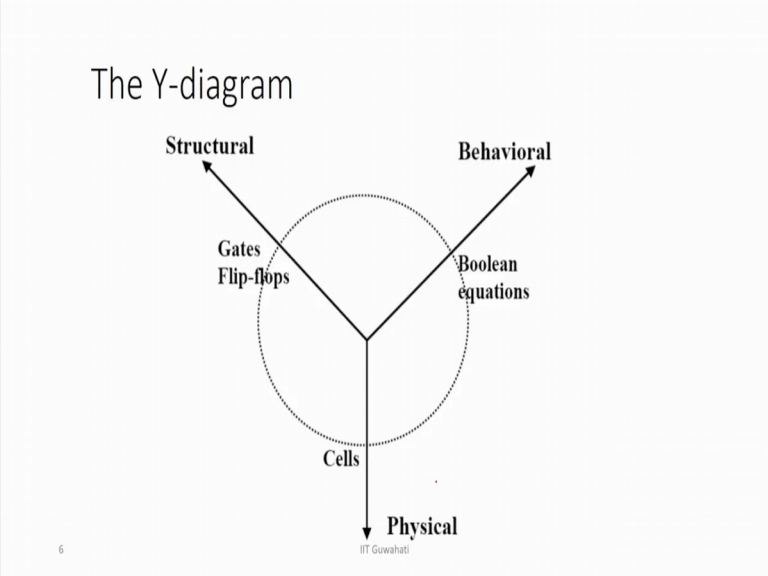
And this particular concept is nicely captured in this y diagram, which consists of three access right this structural accesses, behavioural accesses and physical accesses and.

(Refer Slide Time: 02:39)



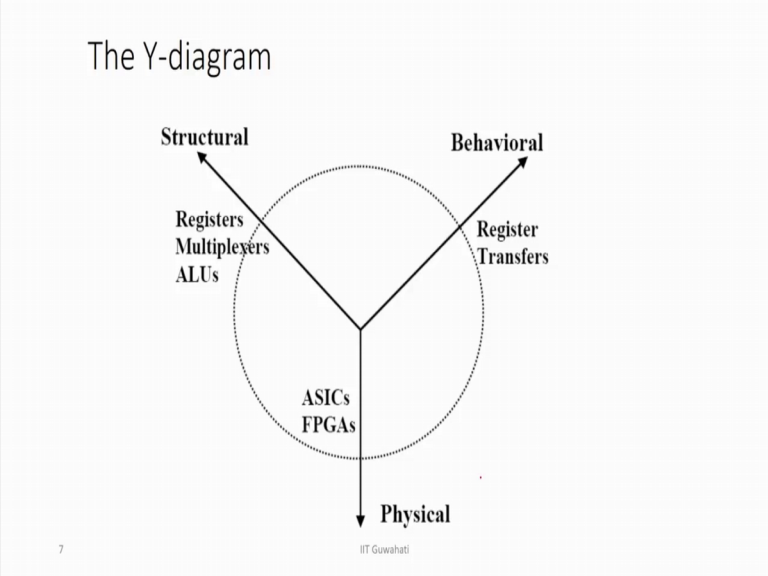
So, if we just think about if you just go from the centre to as for, there it is more abstract designed it; so, if you just think about this. So, in the code level in the structure you have only transistors right. And behavioural level you have the functions that represent the transistor functions right and the physically do have the layout transistor layout.

(Refer Slide Time: 02:52)



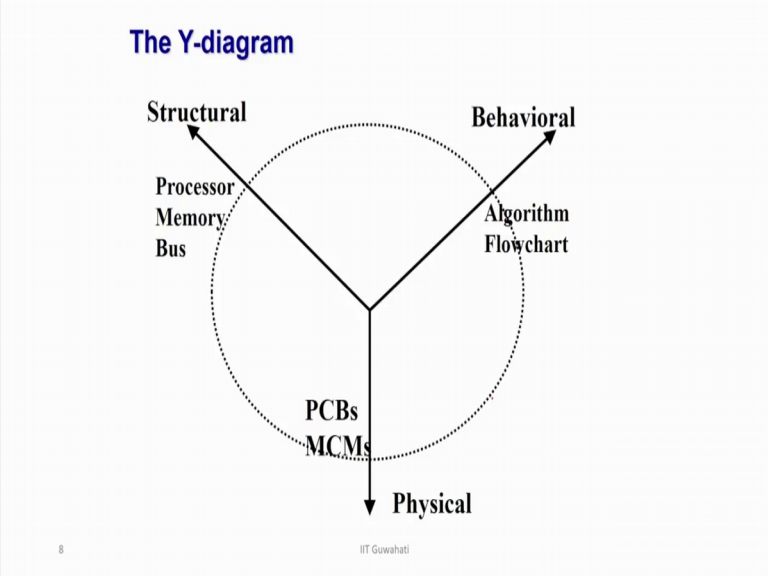
So, if you go one level off so, we will have gate level designs right. So, you have get flip flops in the structural level and the graph will get level circuits can be represented as in Boolean equations right and physically we have the cells, the standard cells of the cell libraries.

(Refer Slide Time: 03:07)



And if you go one level off in the structure level we have the registers, multiplexer, ALUs and physically registers transfer level designs or the ATL designs physically either you go for ASIC or FPGA right and if you go the more abstract level.

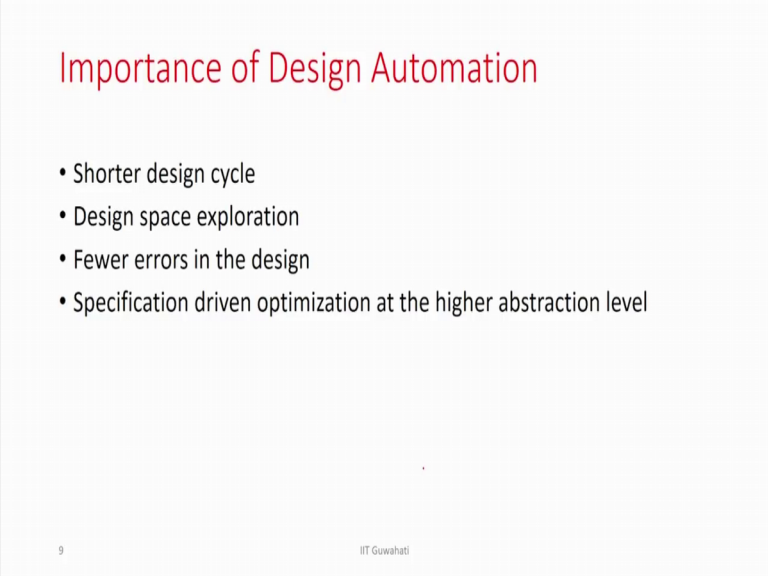
(Refer Slide Time: 03:19)



So, in structural you have processor memory bus may be a level, we algorithm flowchart in a CC plus plus code like this and physically you have this PCB or MCM.

So, this is how the whole VLSI flow stand stands and we try to start designing from the directly from the algorithmic level and that is the overall objective.

(Refer Slide Time: 03:42)



And what is the importance of these design automation steps? The first thing is that is the shorter cycle. As I mentioned if you start designing suppose you take an example of say sorting algorithm right and you just try to implement a bubble sort. So, you write that

bubble sort in c it is just hardly 4 or 5 levels of code right, but if you not think about how you want to design the bubble sort in RTL level, now you have to consider all these registers, multiplexers, adder, subtractor, comparator, clocks reset everything. So, the the bubble sort algorithm in RTL may be 100 lines of code right and if I think about and bubble sort in gate level, I asked you to design a bubble sort using gate level designs. Now, you have all this and gate or gate you have to realize they added using and get on gates or you have reused directly flip flops registers and all those things right.

Now, your design may be 400, 500 lines of code right and now if you think about I asked you to design a bubble sort in transistor level, which is may be 1000 lines or 2000 lines of code. So, that actually give you the idea right if you just write a bubble sort you will hardly 5 minutes, you can write a bubble sort algorithm and you can use a high level synthesis tool or the synthesis tool and this will generate a gate level design or RTL level designs or say transistor level design directly from using these tools.

So, that is basically give you the sorter design cycle. So, that is the one of the important aspect of this. So, the next is the design space exploration. Again if I imply abstruse implement a bubble sort, I mean more options in the hardware right. So, for example, if he try to execute the whole thing in 4 clock, you might need to adder or if I want to execute the whole things in say 6 clock, you might need only 1 adder or 2 adder right. So, then you are using relays resource or there is a trade off between time versus the resource. And there are a lot of possibilities right in some cases you it is to design everything in say 4 clocks instead of 6 clocks or in some time you need to execute everything in say in 4 clocks instead of 6 clocks.

So, based on your design parameters, you have a lot of options like that. So, basically you should already explore this design space the possibilities to get the best result out of it. So, that is also possible through design automation because it is automated technic you do not have to design everything from stech right and also the next possibility is a fewer errors in the design, because now you just think of have to write a gate level design consists of 4000 5000 lines of code and you have to handle all clocks all resets everything.

So, there is a high chance that you make some mistakes someone. So, you are very careful I mean of not making many mistake in the design, but if you just think about a

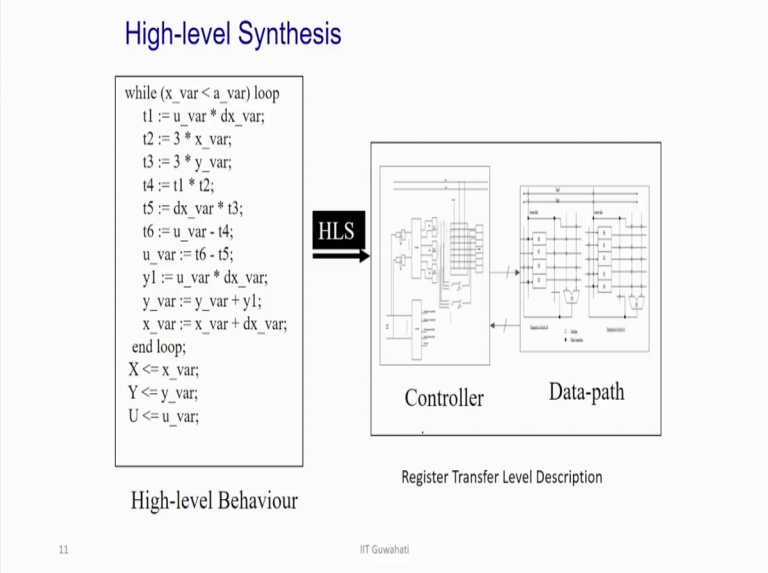
bubble sort is hardly 4 5 lines of code. So, you can easily write that easy layer and there is chances operator this minimum or less compared to if you just design that in our tailored you gate level.

So, if you use design automation the chances of errors is less and also there is another important aspect in specification driven optimization in the higher abstraction level. So, what is that? So, is basically it is if I ask you to RTL design. So, writing 4 variations of a say bubble sort is it will time consuming, but if I ask you. So, you design one bubbles are just to give minimum in it is a fastest RTL, which compute the whole thing a minimum possible minimum possible time right or I ask you to design another bubble sort is you useless least number of the list number of resources.

So, we have your abstraction goal or the design optimization goal maybe different; and based on your goal that RTL or the gate level design that you are going to get will be different right. So, again you can have a high level code you can always try to do these things using synthesis tool right, because you have high level it is take very less time to design an algorithm. So, again if you just use this automation tool design automation tool. So, the specification driven optimization easier right. So, that is another advantage. So, this is why we mean most all the Indian industry or these semi conductor is rely on this all synthesis tools, I mean provided by all these cad companies right. So, that is the overall objective

So, in today we what we are going to talk about is high levels synthesis is the first step. So, from that CC plus plus a high level behaviour from CC plus plus we are going to generate the RTL and how.

(Refer Slide Time: 08:11)



So, what we are going to do today instead of discussing all the sub steps algorithm behind that and what is the logic going on behind and complexities of this algorithm, we are not going to detail of that thing right that. If you just ask I mean as a layman if I ask you to write and ask you how you can convert a bubble sort algorithm to a RTL design, how you can do that?

What will be our steps or not is the common way to do that things right what are the steps we should follow? That we are going to discuss today and in next class we are going to discuss more on how we can automate those process, what are the algorithm going behind that right. So, that we are going to discuss in the two models class, but today we primarily discuss on it how manually or what were the steps that we should follow to convert a serial C code to a RTL design right.

So, as I mentioned high level synthesis is basically, I mean converting a high level behaviour which want to be written in CC plus plus to an RTL design. So, that I see you can see in the left hand side this code is I has a while loop and we have certain kind of operations in that and we are doing in a loop, and then we produce some output x y u and something like this right. So, this is something a behaviour and is kind of c. In the right hand side after high level synthesis we will get a register transfer level design which consists of a data path and a controller.

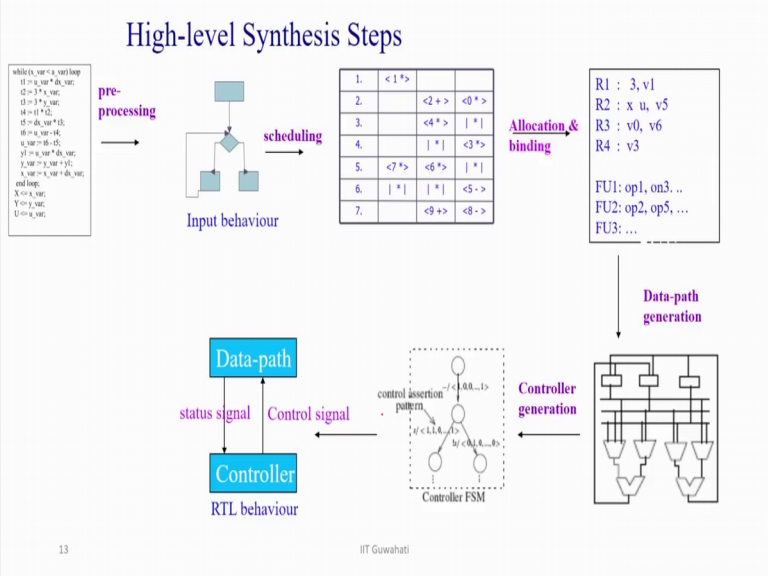
So, data path will execute these operations in the hardware right. So, data paths will is there to execute this operations and controller is there to control the execution operations in that data path. So, finally, our going to generate this kind of circuit from this, and we are going to discuss now how we can do this step by step right.

(Refer Slide Time: 09:51)

So, high level synthesis consists of several sub steps pre processing, scheduling, allocation binding and data path and controller designing right. So, we are going to discuss all those things now right. So, what is pre processing? So, in pre processing what we does? We try to represent that C code in some intermediate from CDFG we calculate the data dependence inside that, we do some analysis, we apply certain compiler optimization on that to generate some optimized version of the intermediate code through which we can do then what we do?.

So, that is what is so, in this diagram. So, its starts from a high level C code, we do pre processing you get a control CDFG control and data flow graph and then what we do we do the scheduling.

(Refer Slide Time: 10:25)



So, in scheduling is what is that? So, we have multiple operations there right. So, we cannot exhibit everything in one clock. So, we have to assign time step to the operation, what we have to decide which operation is going to execute in which time step that is something scheduling right. So, that is what we are going to do next and then we try to map those variable to the registers in the hardware and we try to map the adder from function units, adder multiplier to the function units of this into the of the hardware so that is called allocation and binding.

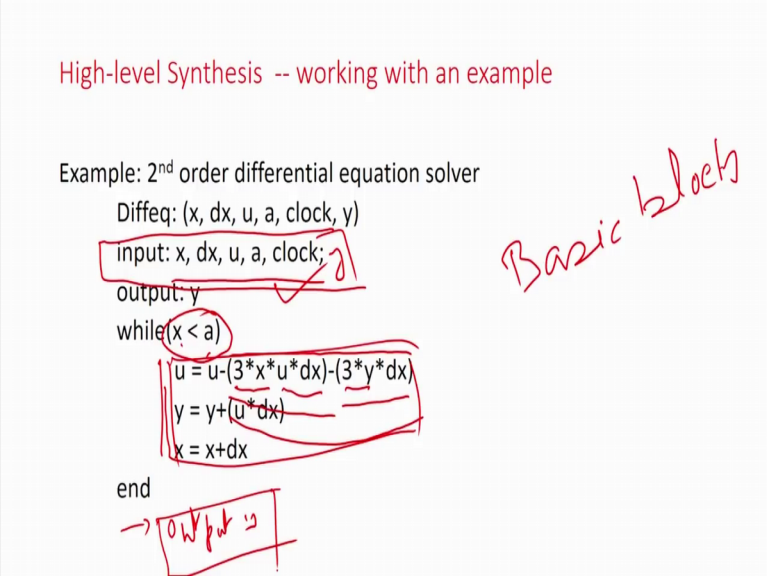
So, after allocation and binding we will we will get to know what are the variable will map to which registers, which of the operator is going to map to which function units right. And what we have done that we try to generate the data path. The data path is what? We have as I mentioned the data path consists of the function units, we have registers and the interconnection component like multiplexer, de multiplexer and the interconnection between them.

So, we try to generate such kind of data path out of this information. So, which is only execute this operations right. And then next once we decide about this we have to control the operations data flow in the data in the controller in the in the data path right, and that will be the control by the controller. We will discuss more on those in in subsequent in this class and finally, we are going to generate this, which is consists of a data path and a controller and this controller every signal generates some control signal to the data path

to execute certain operations and after the operation, you generate some data path gives some status signal to the controller based on that it generally decide the next state ok.

So, this is the overall flow. So, we will we will discuss those sub steps with an example.

(Refer Slide Time: 12:19)



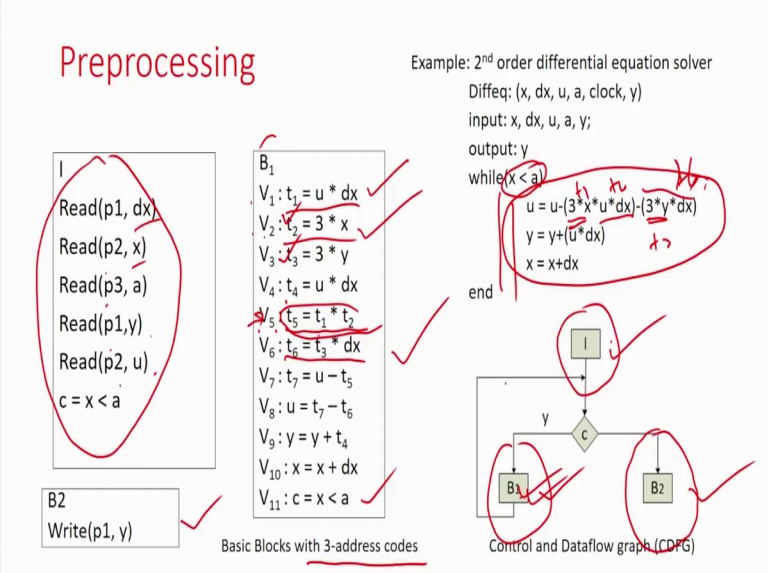
So, what we are going to take is that second order Differential equations equation solver, which is very common in is very there is widely use an example for any cad courses. We will take this example and we try to see how we can generate an equivalent hardware out of this behaviour right. If you look into this behaviour it has some 5 inputs x, d x, u a and y, y should be here and I mean I mean. So, y is also here and we have output is y and in there is a while loop; in the while loop we are going to execute when we do certain operations until x less than y and then finally, we here we are going to generate output right. So, this is the overall algorithm and we are going to discuss here how we can generate a hardware that is going to execute this behaviour right.

So, now will instead of going into detail of this step that we just learnt, what we are going to discuss what is the immediate step that we should do right. So, it will have look into this behaviour, first thing will you will come to your mind is that, there is a while loop right and there are big operations right. We cannot execute all these things in a single clock then what will happen? So, if you want to execute the whole operations in a single clock, the problem will happen is that you have to execute these three multiple it in a same cycle, then these two subtraction there are 5 operation in a changed. So, your

design will not have a good speed, your clock that we are going to achieve through this design will maybe hardly some pillars or something right. And also that there is a loop around this, again then this loop I do not know how many times it will execute it depends on this x and y value. So, again I do not know how many times I have to do this whole thing. So, how do you decide that I have how many clock I have to do these things? So, this is all these things are not possible to do and decide from this behaviour.

So, what we have to do first thing? We have to extract the what are the inputs of this behaviour. So, that we already know these are the and you have to read. So, you have a hardware, we have to we need to know we have to read these values right through some port hardware port. So, basically in hardware there is the input are coming through some port so; that means, we have to realize these inputs or the read this inputs through some port. So, that is what is this these operations right that we have to read suppose and we the number of port in the in the hardware is limited.

(Refer Slide Time: 14:44)



So, you have to decide how many port we are going to use for this design and say for these 5 reads, I have decided to reuse 3 ports say. So, then I am actually reading d x through port 1, x through port 2, a through port 3, y through port 1 and say y through port 2. So, that is what I have decided that I am going to read this 5 inputs through these three ports using through this logic right ok. And also next is we as I mentioned this operations big operations cannot be executed in one clock, then your clock speed will be

very less. We have to break this operation in small small operation for example, I can execute this is a sub operation; this is how a sub operation then will do this right.

So, similarly this is sub operation, then this is another operation then we do the whole thing right then we make it. So, this is already computed and now I am going to do this subtraction operation right. So, this is how I am going to split this big expression into small small part and that is called three address code right. So, what I did? The same example is here.

So, I break these operations, I mean the whole be a while loop operations are here I break them into three address code. So, what I did? I did this three into x I store into t 1 I do you into d x I sorry this u into d x I stored in t 1 I i take this three into x I store in t 2 and then I do this 3 into y here and I store into t 3 right. So, this is what I am doing here and then what I am doing. So, now, this is t 1, this is t 1, this is t 2, I am now I have to do this multiplication. So, I am doing this here right this t 1 into t 2 t 5. So, that is effectively this. So, this is become this t 5 is this and then this is basically t 3, I have to know multiplication d x that is what I am doing here t 3 into d x. So, this is become t 6. So, this is what I am doing here. So, I am breaking this big expression into three address operation.

So, similarly we just see even if I break this operation I am going to get this 11 operations out of this while loop right. I have to also execute this x less than x. So, that is also there. So, this is what I have done to break this big operations into small small three address code. So, that I can execute these operations in some functionality in the harder and we can achieve greater clock speed right. So, this is what is that call breaking these operations into big operations into smaller operations. Then we have to extract as I mentioned this is while loop and there is a basic block inside this, I mean we have to I do not know how many times this loop will be executed this body will be executed. So, I am going to represent the whole behaviour in terms of basic blocks ok.

So, I have one basic blocks here, which will do this reading operations I have one basic block here that will do this the loop body. So, basic block is were there is no branching statement they are all are sequential operations. And I have also have a basic block here just to output the value of y right. So, I have three basic block and that is what I have shown here that is the input basic block, I am moving all this reading operations, this is

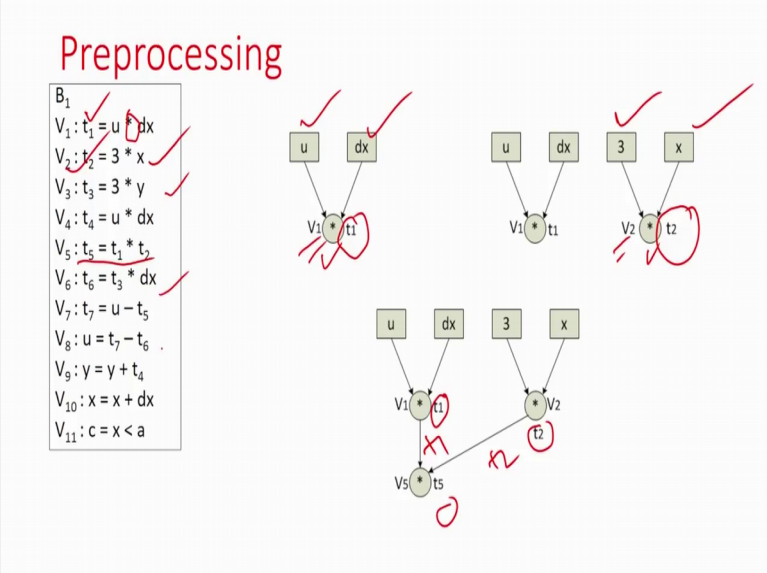
the basic block I am doing this body loop body and this is that output port this is b 2 right and this is that while loop and this is called control and data flow graph.

So, we extract and control and data flow graph from this behaviour right, which consists of set of basic blocks and their control flow. So, this is what I have done. So, we have done two things we break the big expression in the three address code, we extract a control or data flow graph out of it right and what next. So, this is so far we have done in the pre processing steps.

Now, you have to understand that. So, as I mention here this t 1 is you can done here and then this t 2, and this t 5 operation depends on t 1 and t 2 right. So, I cannot execute these operations unless and under this and this is over right unless I did these are executed, I cannot do this. So that means, there is a dependence if between these operation d 5 V 1 and V 2 right. So, these can only be executed once this V 1 and V 2 is over right.

So, that is called data dependency. So, once we are done with this control flow graph extraction and this splitting their operation in the three address code, we have to extract we have to we have to find out the data dependency and that among the operations of the behaviour right. So, we will if you just try to find the data dependency for this block let us try to do that. So, this is the same block I copied here and if I just take this right.

(Refer Slide Time: 19:39)

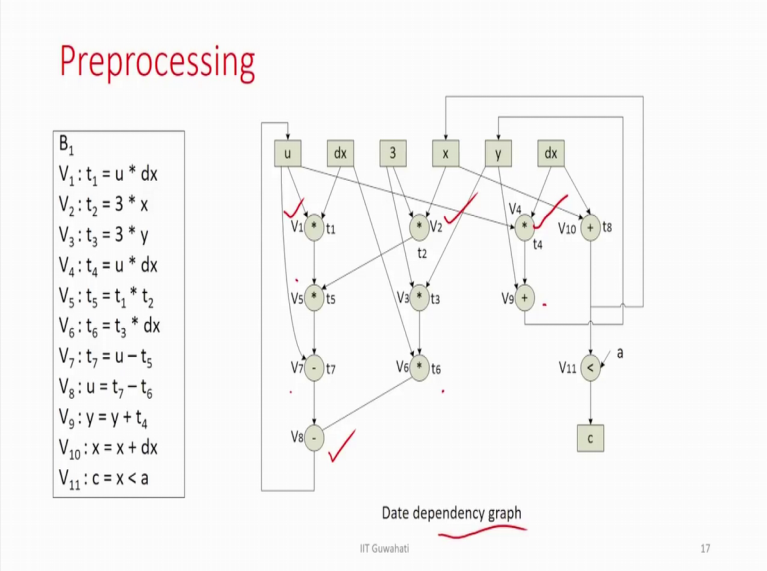


So, this is the data dimension from u d x, these are the input variable and this is that operation, which is a multiplayer. So, I do multiplication the our variable is t 1 and the operation in V 1 right. So, this is that data difference the if we try to represent this whole express this text or the body of this of this basic block using graph right by each input is some variables and the circular nodes are the operator and we try to find out the dependency between this right. So, that is what I just saw.

Similarly if I just do this for this t 2. So, t 2 is 3 and x and multiplication is happening and the output is 3 t 2 and the operation is V 2 right. So, this is I just do it for this and then since this is depends on this. So, if I just do this t 5. So, t 5 is now depend on this t 1 and t 2. So, t 2 is this I just right here.

So, now this since t 1 this operation V 5 depends on our t 1 and t 2 this will come next right. So, this is how the data dependency is expressed right.

(Refer Slide Time: 21:00)



So, this way I can if I just draw the for each operations I can construct a graph like this. So, this is called data dependency graph. So, we have to extract this data dependency from this basic block behaviour and what is signifies? It signifies the data dependency among the operations. So, what it signifies? I cannot execute V 8 unless V 7 or V 6 both are executed similarly I cannot execute V 5 unless V 1 and V 2 is executed and similarly for other operation right. So, I cannot do this unless this V 4 is executed. So, this is how

it actually give you the idea how the data dependency works and which operation will execute first and then which operation next right.

So, this is what is pre processing. In pre processing we do some more some more things which I will not covered here, is basically the compiler optimizations. So, once you write a code like this there might be this many are maybe on optimized code right we might have to might have better version of this code, which we have not written. So, then we can apply some certain compiler optimization technique like say loop transformations say constant propagation, dead code elimination, common sub expression elements and those kind of operations on this behaviour to generate some C code which is kind of a more optimized version of that.

So, that I am not going to discuss today, because our intention today is just to see how from a C code and our till can be generated those are kind of advance topic which will be covered in a very specific lecture. So, so far we have done that. So, we have extracted this data dependency graph, we can represent the whole behaviour as a CDFG each expression are converted into c 3 address code. So, this is this far we have done right. So, now, what is the next step? So, this part we understand.

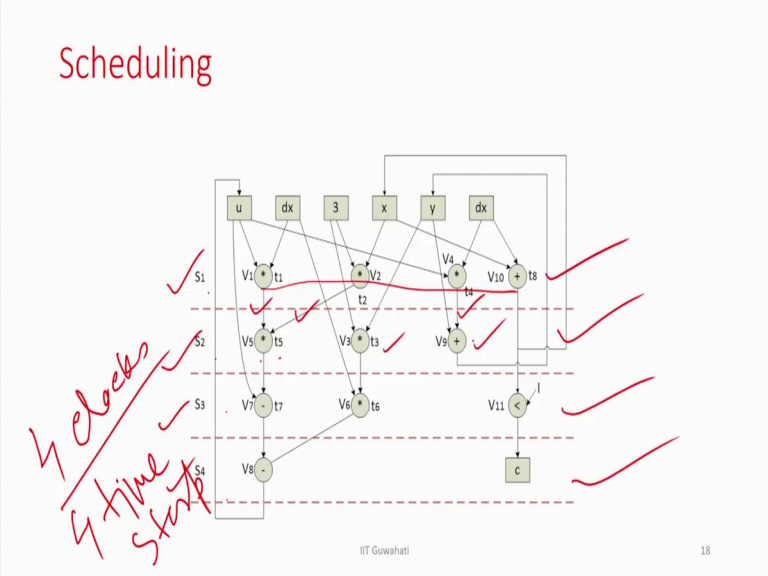
So, now, we have a intermediate representations, which is a basically CDFG for each operation is a three address operations and we also have the data dependence in correspondence. So, what is the next step? The next step is to decide where I am going which clock I am going to execute which operation right. So, that is our next step. So, what is that? For example, as I mentioned here. So, this whole loop body I mean cannot be executed in a single clock because of a lot of multiplications and other operations are there, we try to do it in multiple clocks right.

So, now that you have to decide. And here we remember we usually do this scheduling of this each basic block separately. So, we try to when we are going decide the number of time step for this B 1 I am not going to consider this I one or when I am going to decide the time number of time still required for B 2, I am not going to consider B 1. So, these are kind of we can say individual block, which you can we can decide the running run time or the number of times to be required to execute independently. But there are certain advanced techniques are there path ways scheduler which actually take both path, but

those are advanced technique we can we are not going to discuss on those in this in this course detail ok.

So, what we are going to consider? We go to consider each basic block and try to find out how many time number of clock we require to execute the operation inside this particular basic block right. So, that is what is the next step.

(Refer Slide Time: 24:21)



Because that is what we have to we need to know how many time step is required to execute certain the certain set of operations in hardware right.

So, now for this as I mentioned there are some dependency, I mean we cannot execute these unless these are this executed. So, based on that we can decide key I am going to use 4 clock time step. So, this is time step 1, this is time step 2, this is time step 3, and this is time step 4 executive this operations let us have decided that and now we see whether this data dependency is violated by this. So, here you can see that when I am going to execute all the input all these 4 operations, the inputs are available. So, I can exhibit them there is no data dependency on the other operations. So, this can be done in flow there is no data dependency by lesson right.

Now, if you just think about these set of operations once I am going to execute this, this is available, this is also available. So, there is no problem here similarly this depends on input so, but. So, I can do it here itself and this also this t 4 is available and we y is

available. So, I can do this. So, there is no dependency violation similarly for this and this. So, this is what say I have decided, that I use 4 cycles 4 clocks right are 4 times step right 4 time step to execute this 11 operation.

So, in this behaviour as i. So, there are 11 operations I am going to use only 4 cycle to execute all 4 origin operations right. So, this is what I have decided. So, this is what is called scheduling assigning time step to the operation. So, I am assigning time step 1 to these operations, I am assigning a time step 2 these operations, I am assigning time step 2 through these operations and time step 4 to this to this these two operations right.

So, this is what is called scheduling and this is our immediate next step. Because once we have a behaviour its untimed there is no time assigned this there is no information how many time how many clock cycles required to execute that, that we have to decide first key how many how many clock cycle I am going to use here to execute that particular set of operations. So, that is what is called scheduling right. So, this is done.

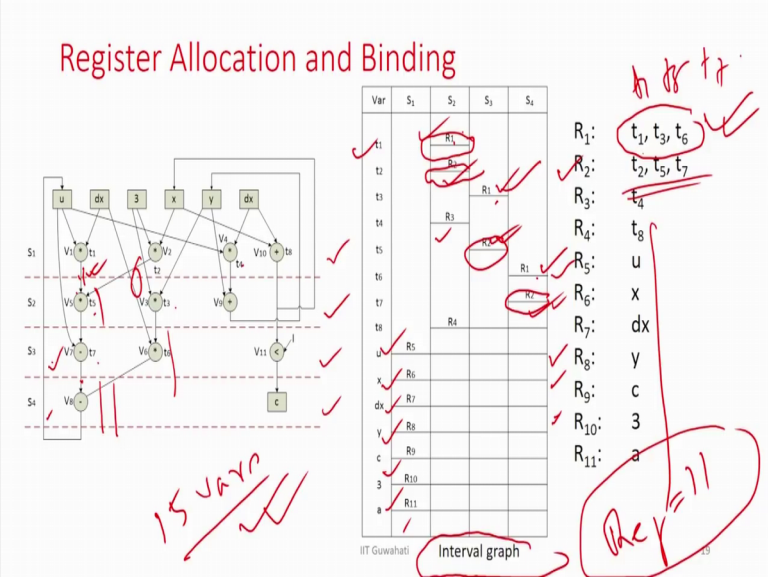
So, what is the next step? So, this is the schedule information and now we have to think about of the hardware. So, we have done with that key how many clock cycle is required or how many times step is required that is already done, and now we have to decide how many registered we are going to use or say how many function unit, we are going to use in the hardware right.

So, the obvious immediate solution is that I can store each variable in each register right that is um that that we can do. Similarly I can use this separate from function unit for each operator, that is the ores possible case rate, but the number of operator is huge and they are not rediced right. So, a that is not the purpose of hardware. I did not hardware I will try to use each operator each operator in I mean that will reduce them as much as possible. So, our objective would be tried to use minimum number of register to store all the variables of the behaviour, similarly try to use minimum number of functional units to execute all the operations of the behaviour right.

So, each substrate has some sub a sub optimization world. For example, in pre processing we try to generate some try to generate some data dependency over the length of the dependency graph is less right. So, that is goal and because of for that we use several compiler optimization technic to reduce that in scheduling what is the objective? The objective is try to generate some try to try to schedule all the persons in minimum

number of time step. The objective is try to reduce the number of time step as much as possible we might have some other goals that you are going to discuss later, but primarily tried to you execute all the operations in as much as less as possible clock possible right.

(Refer Slide Time: 28:29)



T and then we go to this register allocation binding and functional external binding, how you try to use minimum number of register to store or the (Refer Time: 28:37) we try to use minimum number of function you need to execute all the operations of the behaviour right.

Now, the question is how we can do that manually right. So, intuitively how we can do that. So, let us see this example. So, in this example we had we have seen this is we use for time step to execute we use 4 time step to execute this behaviour 4 time step, and now which I do the objective is here try to see are some register to store multiple variables. So, how many variables are here? We have this all these input variables and we have also this in temporary variable because we have introduced so many temporary variables, this t 1, t 2, t 3 and this. So, you have total let us see. So, 15 15 variables are there.

So, there are 15 variables in this behaviour in this particular code and then. So, what we have just to recap this. So, what you have done? We just do the scheduling for this a symbol of 1 right similarly we can do the scheduling for other block also right this basic

block of I and B 2, but since they are kind of a same. So, I am just going to discuss only basic block one. So, other is just it is just the same thing we can do in the other block as well right

So, now we have friction variables are objectivity use minimum number of register could store these variables and the question is how. Now if we just look into this behaviour, this schedule information the t 1 is generated here right at the end of time step 1 and it is going to use here. After this it is not going to use in S 3 and s 4 right there is no use of t 1 in s 3 and s 4.

So, there is no need to store t 1 in for the time step t 3 and s 4 right. So, we can actually use the register that is stored t 1 to store something else, because t 1 has only required in the further time step to where it is going to use here other time it is not required right. So, that is the overall idea. If some variable is not used in (Refer Time: 30:55) time so, you can delete the value in the hardware, because if the register is not stored that value is not available anywhere. So, if it is not required it is not required is it is not required to store as well as. So, we can do that. So, that is the idea. So, we try to find out the life time of each variable and that is represented in a graph called interval graph.

So, as I mentioned for t 1 it is just defined at the end of S 1 and it will be available from S 2 right. So, the life time of t 1 is only in S 2 right it is only using time step 1 after that it is never used. So, there is it is it is life it is basically it is not live variable right. So, it is I can replace the value with something else. Similarly for t 2 also the lifetime is s two, but t 3 t 3 is also defined here. So, it is for S 3 it is in time save S 3 this is t 3 and t 4 is defined here. So, life time is here and for t 5 5 is defined here.

So, this is for S 3 right. So, t 3 5 is defined in S 3 and for t 7 and t 6 they are defined at the end up stand of 3. So, they are available only lie here right. So, they are available t t 7, t 6 and t 7 here. And for other variables like this input variable, they are actually the since this is a loop and they are going to redefine there or there actually has to lie for whole time step because they are defining somewhere and they are going to reuse right. So, their lifetime is for the whole 4 time step right. So, this way I find out the lifetime all variables right.

Now, what we can do if. So, lifetime means where I have the use of the particular variable, and I have to store that data for that time at least other places, I may or may not, but that is there this must store that value right.

So, we try to use we try to use registers such a way that if there are two variables which lifetime is non overlapping, I can store them in a single registers. For example, here t 1 lifetime here, t 3 lifetime is this they are non overlapping and t 6 right. So, there lifetime here. So, they are this t 1, t 3 and t 6 they have a non overlapping lifetime I am going to club them into a single registers and that is not violet any problem and they are not going to problem in the hardware. Because they whenever the data is used that time that data is stored in the particular register.

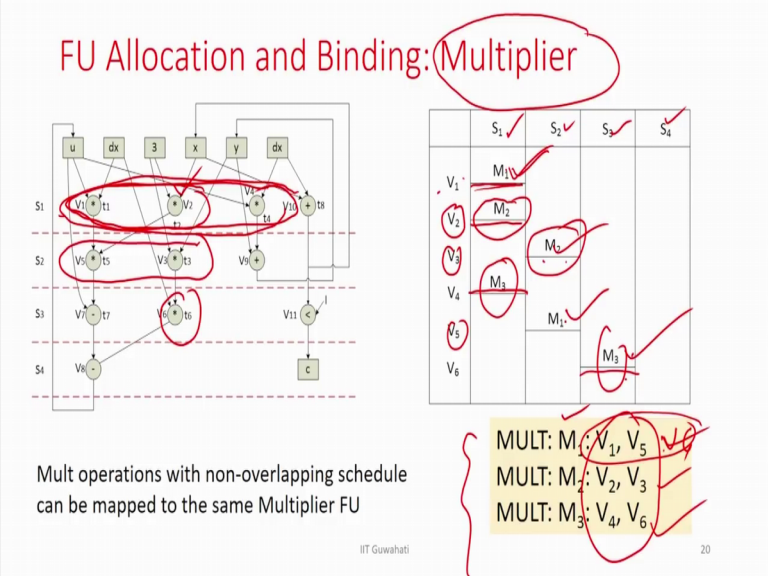
So, that is what is called registered sharing. So, I am going to share a single register to store this three variable. Similarly I can do this for R 2, R 5 t sorry t 2 and t 5 and t 7 in a register R 2. So, other than there is no other facility because they all are overlapping, on a. So, there they need a dedicated register all this very well integrated register. So, that is what I have done here. So, I need effectively 11 register here right.

So, I need 11 register here for 15 variables. So, I have some saving for instead (Refer Time: 34:07) here. So, if you can think of this is small example, but if you have big examples there are hundreds or thousands of variables, then maybe there is register saying maybe much more. So, that is our next goal that I try to map those register because finally, I have to execute everything in the hardware and there is no variable they are only have the registers. So, you have to map those variables into these registers.

Now, if you think about the adding actual design those has to be map into memory ram or ROM in the design. So, that is also another way of doing this. So, you have to has think of the size of the added and then you have to decide which particular is going to map into which ram or whether is it is going to map ram or not that is we also decide by the use pattern. If it is only read only mere array then it can we have two ROM right that if it is read and write both are happening then is going to map into ram.

So, that is also similar way, but I am as far as for the example concern I have only registers, I mean I have only variables. So, I can map them to registers of the hardware. So, this is also done. So, this is the step call register allocation and binding

(Refer Slide Time: 35:12)



So, the next step is similarly the function unit a functional unit allocation and binding and here in hardware you have to remember that the unit that is going to do multiply that is dedicated for multiplication operations right. Similarly the operation that is going to do addition that is dedicated for addition maybe addition and subtraction can be done by the same operator, but multiplier addition can be separate from same unit.

So, here I mean registered is a unique type of thing. So, all variables use map register, but here multiplier you want to map to the function unit it is multiplier functional unit and the addition is going to map into the unit function unit, which is type of adder right. So, we have to do this mapping for each type of operator. So, what I have done here is an example for our running example how to do it for multiplier right

So, again the concept is little bit same the way we do the register allocation binding. So, so if two operations are running in the same time step right for example, these two multiplication. So, I am going to do only for multiplier other operations are not there. So, these two are in the same clock. So, same multiplier cannot be used for them, right I have to use different multiplier for these two multiplier or I would say the three multiplier here. So, these three cannot be done in us using same multiplier. So, I need at least three multiplier to execute these three.

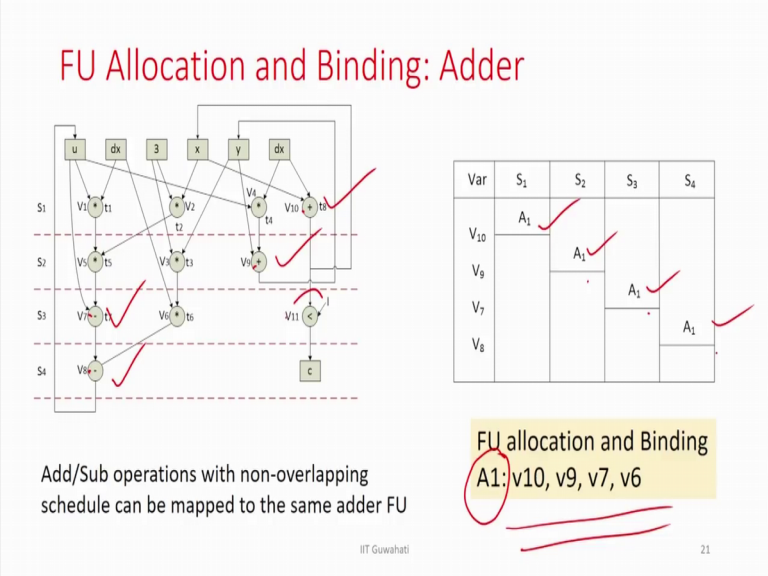
So, I am again I am going to represent their span or the used span for each variable in a interval diagram in interval graph where I have this time step S 1, S 2, S 3 and S 4 I am

going to write a bar in particular time step where this particular way operations is executed for example, V 1 executed here. So, I put a M 1 here I mean this bar here for V 1, V 2 is also executed in time step 1. So, I put V 2 for V 2 and then I put this for V 4 bar in S 1. So, these are the things in running of a time step 1 time step 2I have two multiplication operation V 5 and V 3 this is V 3 this is V 3 and this is V 5 and in time step three I have one multiplication, which is V 6 right. So, this is V 6 and in time step are there is no multiplication. So, this is the interval graph for the you are from the usage of the multiplier this right.

Now, again the same concept if that usage is overlapping, I cannot use a single multiplier to do that do this particular multiple operations in the same time. So, we need different multiply, but if they are used in different time step I can use the same multiplier right for example, here I can use the same multiplier to do this, this two even I can do this, but for some reason I also decided this I am going to do V 1 and V 5 in multiplayer one right this is for multiplier 1 I am going to do this V 1 and V 5 similarly I am going to do this V 2 and V 3 in multiplier 2; and for in multiplier 3 I am going to use this V 4 and V 6 right again as I mentioned for this since there are three multiplier in parallel I need at least three multiplied, but this mapping can be different I can I could have done V 1 and V 6 together.

Even in fact, in fact I can do V 1,V 3 and V 6 also in the same multiplier, but I just decide one of them right. So, these are all possibilities, but you can choose one of them. So, I have decided this. So, this is called from senate and of course, I m finding for multiplier.

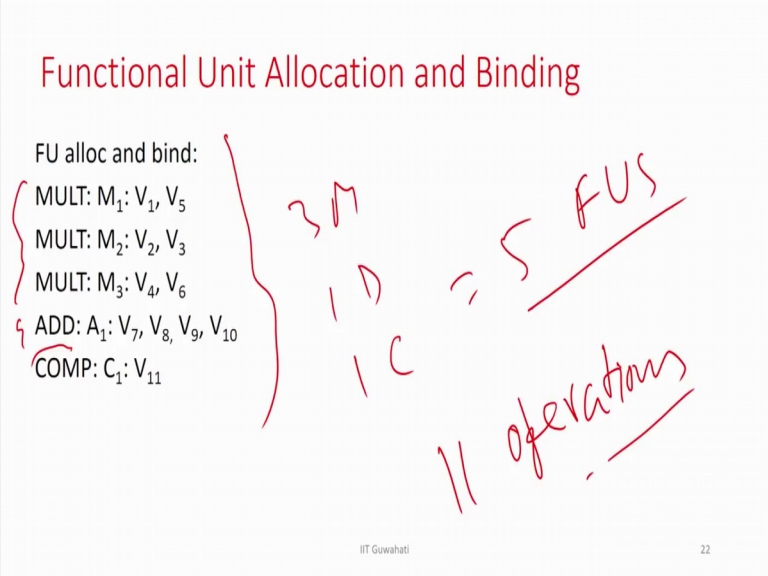
(Refer Slide Time: 38:51)



Similarly, I can do the same thing for adder, and I have assumed that addition and subtraction is done by a single function unit called adder and you can see there are 4 adders here this is one, this is one, this is one substracter these are all adder or sub stracter type. So, there are four, but they are all in different different clock step right in time step. So, these are all in.

So, this is V 10, this is V 9 this is V 7 and this is V 8. So, I have done this in different different clock step since they are known about let me again use a single adder or substracterto execute all these 4 operations. So, I need only one substracter added to execute this all 4 additional subtraction in the of the behaviour in the hardware. So, at the end what I have received. So, I you I got three multiplayer one adder and similarly for comparator I need one because I have one comparator operator and basically operation it is comparator.

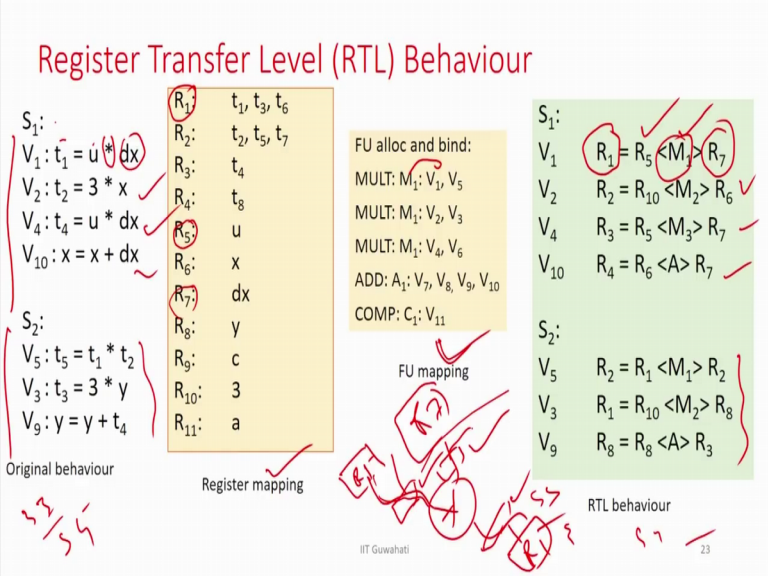
(Refer Slide Time: 39:43)



So, I got one comparator. So, I need three multiplier 1 adder three multiplayer one adder and one comparator total 5 to execute all this 11 operation. So, that is 11 operations are there and I have 5 functional unit. So, that is what we got from this sharing information. So, this is what we have done.

So, far, you understand the concept right. So, we first decide the time step I am going to do a which operation. based on that scheduling information or that assigning time step operation information we decide how can share some register to store multiple variables th V is (Refer Time: 40:28) sharing other some and then we again decide how to do multiple operation using a same function unit and that is what is called function unit allocation and binding and after that what we see is this

(Refer Slide Time: 40:41)



So, we have the register mapping information I have the considering information and in S 1 I have I have done this for this 4 operation right. So, if you just see here I have done this V 1, V 2, V 4 and V 10 in a S 2 I have done V 5, V 3 and V nine similarly in S 3 I have done this three operation in s for this. So, I just call do it for two time step I i can do for S 3 and s 4 as well right this is s S 3 and S 4 as well, but. So, I have 4 operations here and in S 3 I have three operation S 3 S 2 I have three operation in S 3 also I have three operation in s 4 I have one operation right.

So, and now I have this register nothing information, I have this function you will nothing information and now I can actually represent this high level behaviour using register transfer level behaviour what is that? So, for example, in time step 1 or S 1 I have operation relative one equal to u d x, but u u is nothing, but hardware u is nothing, but R 5 right. So, I can replace this u by R 5 similar t 1 is nothing, but in hardware is R 1. So, I can replace this t 1 by R 1 right similarly this multiplied is nothing, but M 1 because V 1 is happening here. So, I can replace this multiplied by M 1 multiplied one right and this similar d 6 is nothing, but R 7. So, I replace is d x by R 7.

So, now these operations t 1 equal to u star d x is nothing, but in hardware some register transfer level operations between R 5 and R s7. So, I am reading R register 5, I am reading register 7 and I am doing a multiplications and that result is stored in register R one. So, this is what is an R register of transfer level design operations in the hardware.

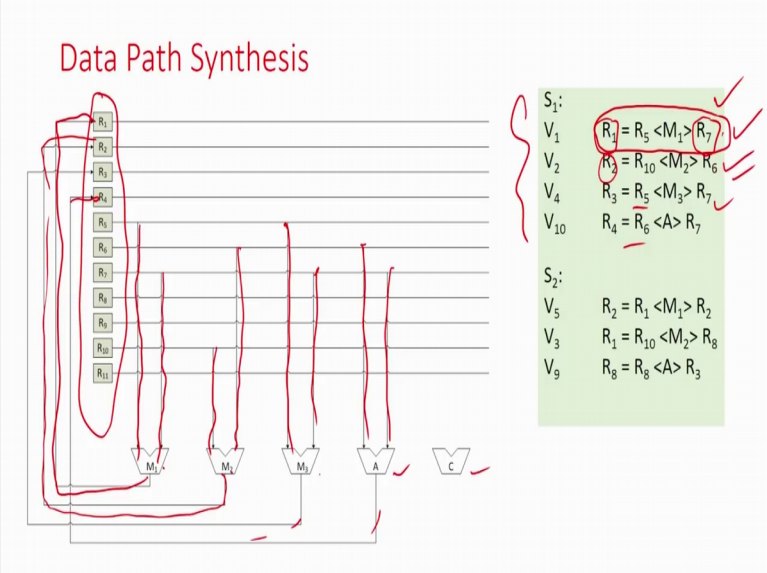
Similarly I can do this V 2 I can replace this variable with their corresponding register name and this operator is the corresponding multiplayer, I will get this operations similarly for V 4 I will get this, for V 10 I am going to get this, for S 2 also this operations I am going to get this, for S 3 also I am going to similar kind of thing and s 4 for also times s 4 also I get a operations like this.

So, then you can understand the basic clock, I have 11 operations first I partition them into 4 times step and then I find the register information's based on nothing information, I found the function in your nothing in formations and then I can actually represent that high level behaviour is using the register transfer level behaviour.

So, I have now I do not have I do not have any variable I do not have any operator, I have only registers and the operations right. So, this is called a register transfer level behaviour. So, now, I actually map those high level V b R at least one basic block into is equivalent register transfer level behaviour right. So, this is clear right.

So, now what is the next step?.

(Refer Slide Time: 43:40)



The next step is very natural Is I have to generate the data path, because I just make here these are the operations I am going to executed these are the operations I am going to execute in the harder, but I have to make the connections right. Because here I just make this essence, but in hardware what is going to happen? This has to be make the

connection proper connection here so, that this operation can happen right. So, if you just think about I have this is say R 5, this is say R 7 and say this is the multiplier. So, I make I have to make this connection to this is R 1 then only this will happen in the hardware right. So, this is are this kind of connections I have to make and I have to make this connection because I am using this multiplier, this may not be a direct connection I have to make a ma add a max here right.

So, because multiple input might come here right because I am sharing similarly here also this might go to multiple places. So, that is called the data path generations and the objective is to minimize this interconnection cost. Use minimum number of multiplexer demultiplexer just to make the connections right. So, that is the objective of this data path generation and that is our next step data path synthesis. So, what I am going to do here, I am going to do it for one to operations in the data path connections and then you will understand the how the how the things works right.

So, as you have decide I have 11 registers. So, this is my final hardware is I am going to generate. So, I have 11 registers here, I have 11 registers here, I have three multipliers I have multiplier 1, multiplier 2, multiplier 3, I one adder, I have on comparative. So, this is what is the components that is already decided here. Now adder net the connection for this operations right. So, let me do it for these operations first right. So, I have a connection to multiplier from R 5. So, from R 5 to multiply 1 I make this connection right. So, this connection is made and then this is the right R I is operant for this multiplier.

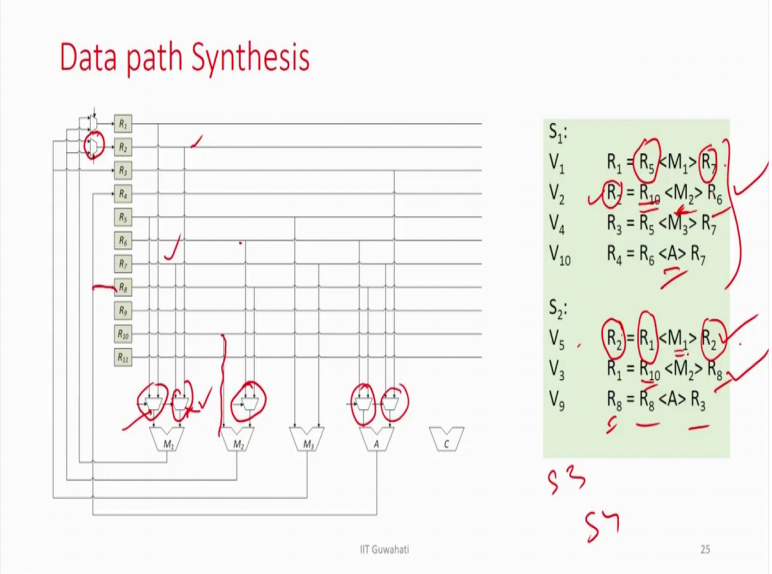
So, from R 7 I make this connection right and the result is stored into R 1. So, this is going to R 1 we can see this is going to R 1. So, just to do these operations I make this connection similarly if I just do it for R 2 for this V 2 operation V 2 and I have to make the connection from R 10 to M 2, R 10 to M 2 and then R 6 to this multiplier to R 6 to multiplier 2 I will make this connection and the result you are going to store in R two. So, this is going to this is going to R 2 right.

So, this is done similarly I have to make the connection for this R 3. So, for V 5. So, from R 5 to this R 5 to this R 7 to this and the result did not to R 3 you can understand this going to R 3 similarly for this x R 6 for this adder this coming from R 6 and R 7 and

output is going to R 4 you can. So, this is going to R 4 right you understand this. So, for the time step 1 I make the connection steps there is no problem right.

Now, if you just go into this time step 2 what will happen? Let us again we are going to consider this.

(Refer Slide Time: 46:54)



So, now this is already done and for that this is the connection I register right now if you just think about this operation. So, now, the input of multiplier 1 is coming from R 1 right, but earlier it is coming from R 5. So, now, I need a multiplexer just to decide between R 5 and R 1 right. So, this is what I have done here.

So, in the previous diagram I have only direct connection from R 5 to this, but once I consider this connection, now I have two more option or 5 or R 1. So, I have to use a multiplexer and I need a control signal here just to decide between R 5 and R 1. So, 0 means R 5, 1 means this R 1 right. So, this is what I have to add this multiplexer; similarly for the right and right hand R reaches this right operator I have earlier R 7 now I have R 2.

So, I need again a multiplexer that will choose between R 7 and R 2. So, I need this multiplexer and this contraceptive 0 means this R 7, 0 means this one is this right and the store result is going to R 2, but in earliest times there R 2 data is coming for m M 2, now it is coming from M 1. So, I need m for at the input of R 2, I need a multiplexer that will

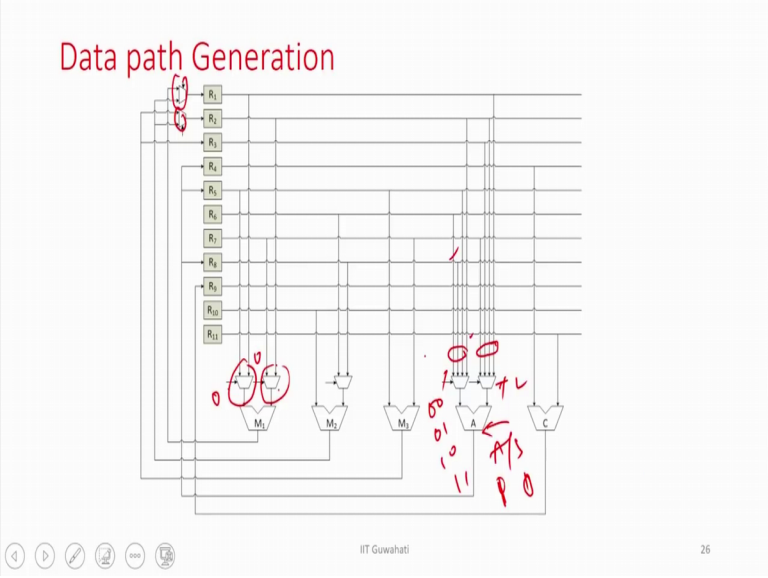
dis choose between this R 2 sorry this M 1 and M 2 because I in first kind of this R 2 input is coming from M 2 multiplied 2 now it is coming from multiplier 1. So, I need a multiplier 1 that is choose the input from either from multiplier 1 or multiplier 2 at the input of R 2 ok. Similarly if I just consider this one I have to again the (Refer Time: 48:38) problem will come for M 2, I have to add this multiplexer here, but you can see here there is interesting thing that V 2 stores R 10 right

Now, also it is coming from R 10. So, I do not need a multiply this you know both the cycle this is data is coming directly I have I do not have to use a multiplier here. So, that is kind of saving. Because if we just add a multiplexer is useless in first clock is going to use select R 10 second clock also is going to select R 10, effectively that multiplexer is easier done.

So, I am not going to use it right, but for the right hand side earlier is living from R 6 not going from R 8. So, I have this is R 6 and this is R 8. I need a multiplexer here and the control signal 0 means this R 6, 1 means R 8. And for multiplier three there is no multiplier 3 here and for adder now it is coming from R 8 and R 3. So, I have to make add multiplexer similarly and the result in storing into R 8. So, there is a different way.

Now,. So, then this is no multiplexer is required because it is updating a new register now. So, for these two time step, I make the connection. So, if I just add for S 3 and S 4 this way the final diagram will get this right.

(Refer Slide Time: 49:39)

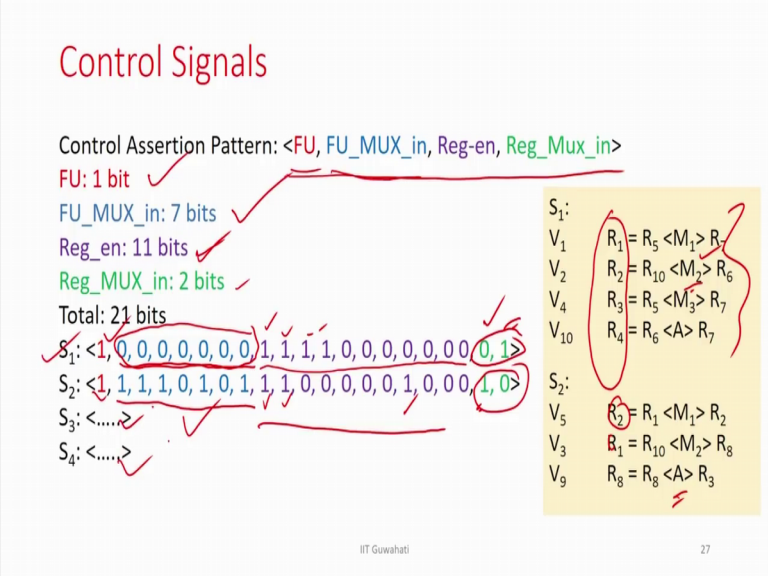


So, I can see that the multiplex are the input of this multiplier 1 I have only two inputs and this also two inputs, this needs two inputs, but for the adder when there are the 4 inputs possible input and there are two bits as control signal is required 0 0 0 1,1 0 1 1 0 0 means it select this, 0 1 means did you say the second one, 1 0 means second third one and if it is 1 1 then it is the fourth one right and so on. Similarly here also I needed two bits control signal to select between this, and I have added multiplexer only for two register other is the multiplexer is not required because it is only one data is coming to this.

So, this is the overall data path that is going to generated from this information's right from this information's from this register transfer operations, this is the data path is required right. So, this is the final data path that is generated out of this information right. So, data path is ready; I have 11 registers three multiplier 1 adder one comparator I have 1 2 3 4 5 6 7 multiplexers and the inter connections. So, this is the data path that is generated. So, we are almost at the end of this (Refer Time: 50:57) So, only left its control signal and controller generation.

So, what is what is the need of controller and what is that that you should understand right. As I mentioned here this multiplexer is not known it has to be controlled right, the data flow from the input to the output to the multiplier has to be controlled by these control signals and that has to be generated and for different control step or the time step the operation that are going to execute in this in this I mean the selection that is going to happen here is different right.

(Refer Slide Time: 51:27)



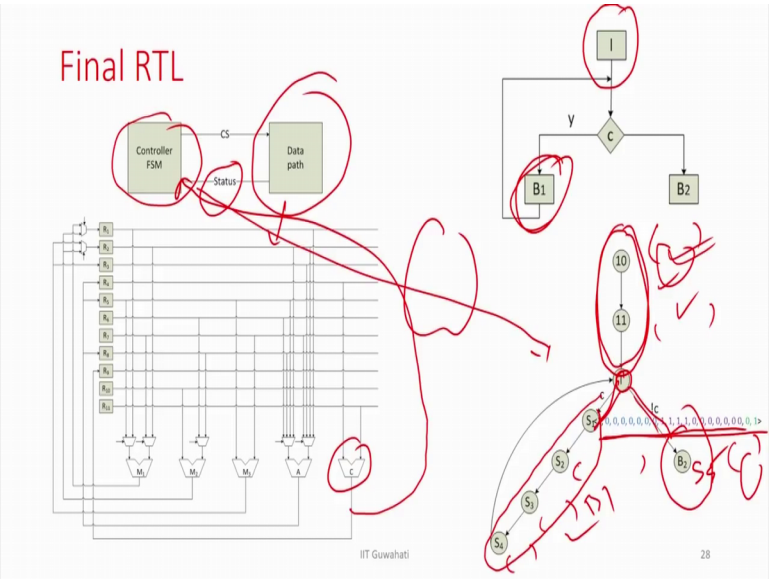
So, what a how many control signals are required here? I need control signal for FU for this are all multiplied I do not need, but here I need 1 because it is going to do add or sub right. So, 0 means add 1 means sub or other way this is 1 means add and 0 means subscription. So, I need a control signal to tell this adder this function you need to do perform and perform either addition or subtraction right. So, that will be decided by this. So, I need one bit for the function unit for other I do not need and then I need the control signal for FU marks in, the multiplexer that are the input of the f use function units right how many bits are there? One bit for this, one bit for this, one bit for this, two bit for this, two bit for this. So, 4 2 2 4 and 3 7.

So, I need 7 bits of control signal to control the data flow through this multiplexers right and then how many register the next important thing is that in this register enable. So,. So, this registers unless you control the right to this is going to write in every clock and that will. So, that will the garbage data will be retain. So, we have to control the writing on the registers and we need the right enable signal right.

So, once the right enable signals is there, then only that register be updated otherwise you hold the old value and that is very important otherwise what will happen? Even if you are not doing any operations if you do not control the control the writing on that registers what is happen? It will just store some it will update is the some garbage value and you have no clue what is happening there. So, that is very very important. So, you

need register enable signal right enable signal for our registers and I have 11 registers. So, I need 11 bits for that and I have 2 marks here and 2 bits. So, I need to 2 bits register marks in 2 bits right.

(Refer Slide Time: 05:11)



So, this is what the number of control signals required. So, total number of control signal is required is 1 7 8 plus 11 plus 2 is total 21 bits say, 21 bits control signal is required and that is order is this right and I use the colour coding to decide that. And now we have to decide in time step 1 what will be the control signal right or the control assertion pattern for time step 1. So, as I am doing addition in the time step 1 right I am doing a addition I the function unit I decide 1 right and the next 7 bits represent the mux input right. As I told you that 0 means it will select this 0 means it will this.

So, I am going to give 0 here, 0 here to perform these operations this and then I have to give 0 here to select this and to perform this operation, these operation because I am now in time step I want to do execute this operation not this operation right. So, based on that I am giving this 0, here also I have to give 0 0 and 0 0 to select the first one that is that.

So, the control signal to the multiplexer is all zeros right and then I am writing for register R 1 R 2 R 3 and R 4 in the time step 1. So, this right and enable signal will be 0. So, these are the next signals at the right enables right. So, this all the 11 bits are right enables right this is R 1 this is R 2 this is R 3 this is R 4. So, this is the right enable signal

for this R registers and all other register will not be updated. So, they will be 0 right and then I need here multi signal for this right.

So, now here I am updating register 1 from multiplexer 1. So, this is 0 and I am updating from multiplexer 2, I am sorry multiplex 2. So, this is one right. So, this is one right. So, this is 0 1 for this. So, this is 0 because it is updating from multiplier 1 this is updating from multiplied 2. So, this is 0 and this is 1 right. So, this is the control signal.

So, you understand this is the first clock controller will generate this signals to this data path control right so that it is effectively execute this only these 4 operation are nothing else right. So, that is what is going to happen. Similarly let us discuss that S 2. So, again I am going to do addition. So, I need 1 here and the rest this 7 bits are for the multiplexers and now I am going to choose the second input.

So, I need 1 here, I need 1 here and I need 1 here, I need 0 1 here because I am going to select the second one 0 1here. So, the counter pattern is this 1 1 1 for this 3 3 multiplexer, and this 0 1 here, 0 in here. So, this is the control signal for the multiplexers, I am updating R 1 R 2 R eights. So, this is R 1 this is R 2 and this R 8 R eights are z0. So, this is that right enable signal for the registers and I am updating here R 2 and R 1. So, I need 1 and 0. So, this is the control assertion pattern that has to be generated for the control step 2.

So, similarly I can generate the control signal for S 3 and S 4 when you understand that right. So, this is I am done now. So, what I am getting here. So, this is the final data path. So, the final R T L. So, I am done with everything. So, I have generated describe the data path this is my final data path, this is my final data path, I have the controller FSM which is nothing, but this how I am going to get this. So, I have just talked about this 4 time step and this is nothing, but B 1.

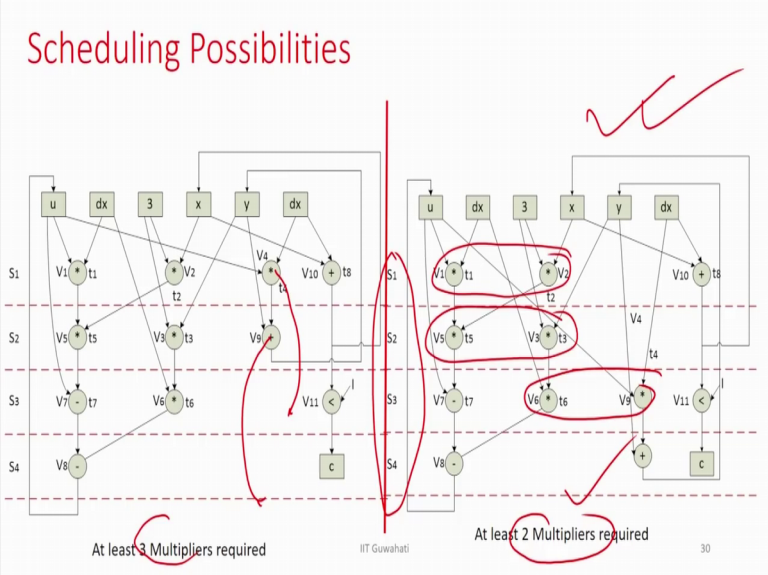
So, I just replace the B 1 here by these time steps and similarly I am going to schedule this I 1 and for the example that took it you need two clocks time step. So, you need to two steps similarly for B 2 I have only one operation. So, I am going to replace that by that particular time step right. So, this is say S 5 and this is I 1 and I 0 ok. So,. So, this is my controller efficient and in what it does? In controller every time step it generate the controller assertion pattern right.

So, I will just talked about the controller assertion pattern of S 1 and S 2. So, therefore S 1 this is the control signal I just talked about for S 1 also have some assertion pattern, S 3 also have some assertion pattern, S 4 also some assertion pattern, here also some under assertion pattern, here also and here also. The assertion such that is going to execute on is that may read that because in this block I am doing this reading from portrait. So, that will execute the read operations from ports here this is for the basic block B 1, the all the operation that over discussed right.

So, finally, we got this whole RTL right, which is consist of this and this I mean this is the part of only this we have some are certain other data path for this and certain data path for this. So, this is the overall structure and what is the status? So, I am doing this comparator this will go to directly if it in this controller. So, that I am going to decide whether I have to from this state I have to go this or this right. So, this is decided by the status signal.

So, this you understand this is the final RTL that you generate which is going to execute that differential equation solver right that I have shown in the earlier the start of the presentation. So, that is the C code and this is the equivalent RTL and what we have discussed here is basically, how we can generate hardware out of this what is the intuitive next step, next step, next step to generate and hardware out of it. I do not discuss any automation or the algorithm so far, I just so this is the logical way to generate and hardware or additional transfer level design out of a C code or a high level behaviour right.

(Refer Slide Time: 58:38)



So, now when I do not talk about other aspect, I will just briefly give you some idea for example, I decide about this scheduling rate and it has also 4 time step. But if I just think about another version where I just do, because these operations here depends on input I could have done it here also right. So, if I just do it this is another schedule right you understand this set. So, now, this also depends on only input. So, I can put this to here. So, I just put it here t 3.

So, I can do this 5 operations in time step 1, that still I need for proper number of time step. But here I need 4 multiplayer at least 4 multiplayer right I need 4 multiplayer here I need 3 as we already discussed. So, this is better that because I need only 3 multiplayer the same time, I am going to executing this in 4 time step I need three multiplayer I need (Refer Time: 59:23) I always take this it. So, there is a difference, the possible there other possibilities are there I choose this for some reason.

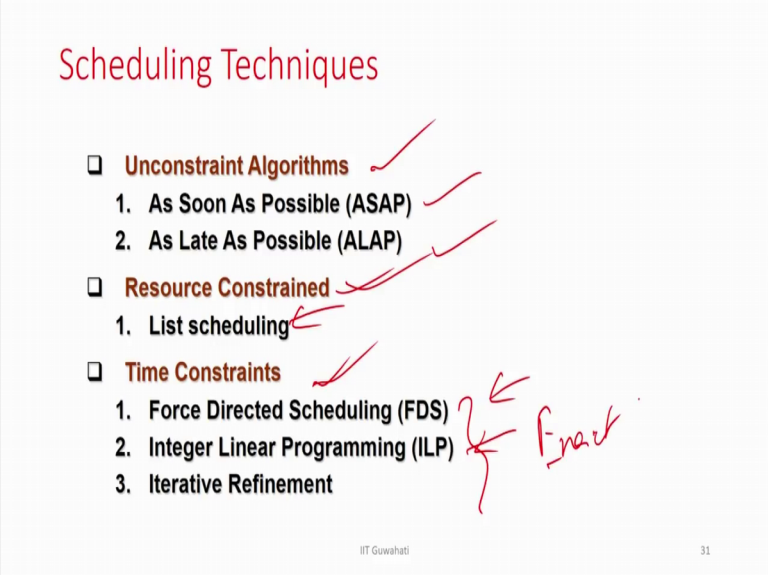
Similarly, you can do one more things here. So, since this I need 4 times step after that this part idle right I can move this to this time step and this to this time. So, then also this is fine there is no problem right. So, this is that schedule. So, the advantage here is that still I have 4 time step, I am executing the whole thing in 4 time step, but the advantage is now see I have to maximum parallel multiplayer, there two multiple parallel two multiplayer here.

So, I need maximum 2 multipliers, 3 multiple instead of three multiplier. So, this is better instead of this. So, what I am trying to emphasis here is that, (Refer Time: 60:05)

scheduling there are multiple options and you never know which one is better. I just show three possibilities one is generating for multiplier one is generating three multiplier, one is generating two multiplier, but all are using 4 time step.

So, that is what I call the design phase expression and based on your objective or based on your target, you might choose this one or you might choose this one or you might choose the other one. So, that is something is the choice or the design of expression or the design targets. So, scheduling technique is something is how to automate this whole thing and check which is the best one right. So, that is what is called scheduling and we are going to discuss all these techniques in the next class.

(Refer Slide Time: 60:45)

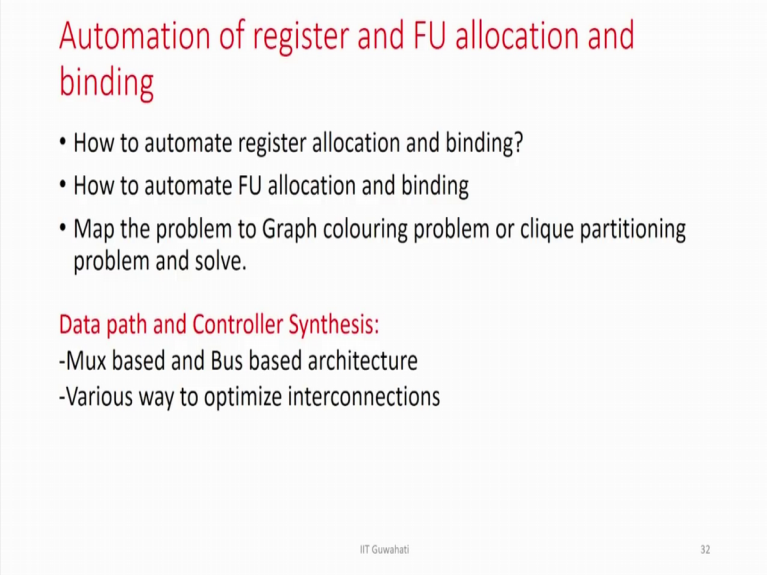


So, there are different kind of scheduling ASAP ALAP or resource constraint, time constraint. So, we are going to discuss those algorithms and how you can automate that process in next class.

So, that is what is that there. So, initially what I just so, I just give you some schedule without thinking how we have done that, but now I have to automate that process rate and during automation I can add, it can be unconstrained, it can be resource constant, it can be times constant and it can have various kind of algorithm, some of them are basically exact solution, some of them are heuristic, these are all heuristic this is exact. So, we have different operation options right and we are going to discuss various

scheduling techniques in tomorrow's class similarly we just going to this register allocation and functional allocation binding we have various possibility.

(Refer Slide Time: 61:29)



I show you in that particular example here, that instead of this mapping I just show you here that instead of this mapping I can a different mapping also right. I could have do this and this in one multiplayer right. So, I can do V 1 V 5 and V 6 here also right.

So, we cannot have the other options which in is better I know I do not know right. So, maybe if you do this the interconnection cost you less we I have I have to use less number of multiplexers or if you do this may be the best options. So, there are multiple options even using the same number of multiplayer. If I told you can use 4 multiplier still you can do that; you can use 6 multiplier and you can do all the operations differently that is also you can do that, but which one going to have the very less number of resource and the interconnections. Interconnection also depend on this kind of mapping similarly register transfer also there are this mapping also can be different.

So, you have instead of t 1 to t 3, 6 I could have stored t 1 t 5 and t 7 that also I could have done instead of this I can store t 1, t 5 and t 7 that will I could have done. So, there are other possibilities are there my point here is just to emphasis that point that the choice is not unique you can have multiple choice and you never know and all are interdependent that if you do one kind of scheduling the kind of register sharing we are

going to generate is different or if you do kind of register sharing or function units sharing the kind of interconnections that going to be generated in different.

So, we are going to talk about those techniques I mean and how these things are interrelated this more detail in tomorrow's class. So, in tomorrow's class we are going to discuss more on this various scheduling techniques, how we can automate this f u allocation and binding techniques when the steps, and how this data path is depend on all these steps that we are going to discuss in the next class in more detail.

Thank you.