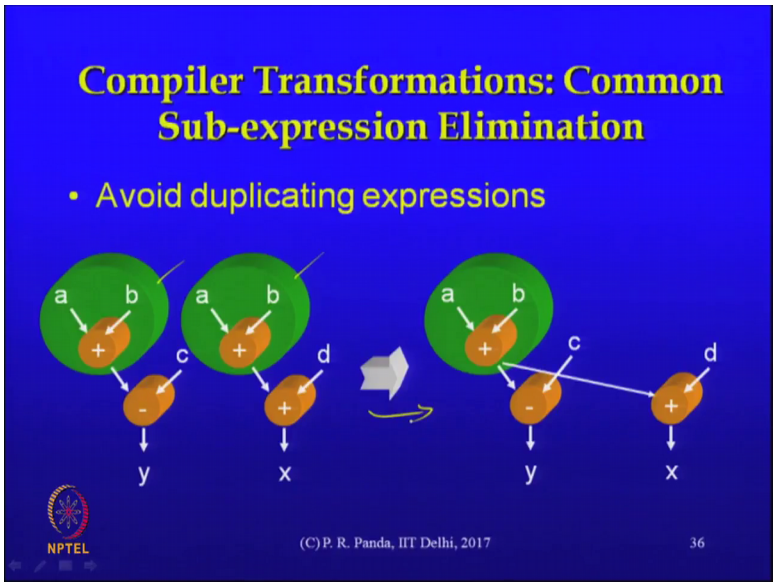
Synthesis of Digital Systems
Dr. Preethi Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 10
Memory Modelling & Compiler Transformations in High-level Synthesis: Common Sub-Expression Elimination & Loop Invariant Code Motion

Let us continue the discussion on the high-level transformations with a few other examples that are also borrowed from the compiler domain. Let us start off with common sub-expression elimination. This refers to a situation where you have multiple expressions that the user has put in different statements and through an analysis you realize that parts of the computation are common that there is a duplicate computation.

(Refer Slide Time: 00:59)



And you would like to of course, remove that duplication to the extent possible so that is the possibility. The example here is you have an expression a plus b that occurs in two different parts of what could be the same data flow graph, they are just two different sections of it. When you do that of course, you would like to avoid the duplication the recomputation as long as you can establish that a and b are the same, then the result of the tapeless base should be computed only once there is no need to compute it twice.

So, this transformation to that right graph should be intuitively obvious why that makes sense. Since this is extracted from code that we have written, we do need to first make

sure that a and b have not changed their values since the last time that expression was formed otherwise it is not a valid substitution, but this is something that makes intuitive sense, in fact, manually we often write expressions in this way. If we find that you have to write two expressions that are the same or they have something in common then we might manually restructure the code in such a way that you introduce a temporary variable and then express the other two both the computations in terms of that temporary computation which is performed only once. The idea here is that we would like to do it automatically right as part of the synthesis optimizations.

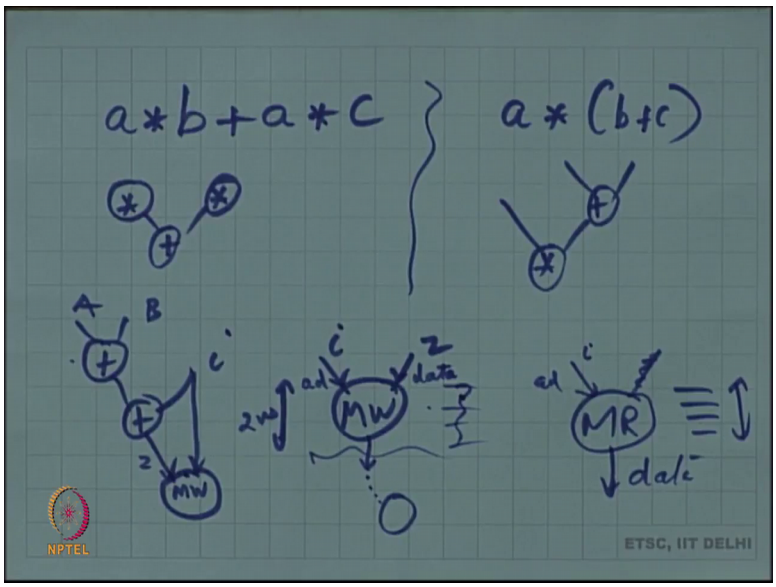
This is also an example of a typical transformation in compilers, but something that also applies in a straightforward way to synthesis, but even though it seems very obvious, the applicability seems very natural in fact it is a totally nontrivial optimization. Both from the point of view of how to do it, now graphically in a simple example it might be clear that there are these two sub graphs that are isomorphic means that there is a one-to-one correspondence between the nodes and the edges that is what they are what we have highlighted in green are the isomorphic sub graphs. They perform the same computation, structure of the graph is the same, variables involved are all the same, and therefore, we can say that there should be only one instance of that sub graph not multiple.

In the general case detecting that there are two sub graphs in a larger graph that are isomorphic is a difficult problem; it cannot be done exactly in a reasonable amount of time. So, that itself imposes some limitations on how aggressively you can look for common sub-expression illumination. Remember that these sub-expressions may form a small part of a much larger expression; and our objective in this optimization would be that no matter where it appears, if there is an opportunity we should detect it and exploit it.

So, this is applicable everywhere, but in this compiler context in the synthesis context and so on, but inherent difficulties are there with respect to how aggressively we can optimize this. In fact, within synthesis also it appears at multiple levels of abstraction, this is high-level synthesis, but even if you go down to logic level value where you are doing Boolean optimizations there also there is an opportunity for an optimization like this right. If we you have multiple Boolean expressions instead of arithmetic expressions these are bullion expressions and there too it makes sense to avoid any computation why not right.

So, we may come to that later, but wherever whatever the level of abstraction is such an optimization is hard to perform. Remember a plus b is the same as b plus a. And I would like to take advantage of that in this optimization why not if I can establish that two computations give the same result then too I should perform that computation only once. So, what it also brings up is that even the graph need not be isomorphic. Even if the graphs are not structurally identical, structurally equivalent then to the expression might be functionally equivalent, you can write the expressions in this way.

(Refer Slide Time: 05:55)

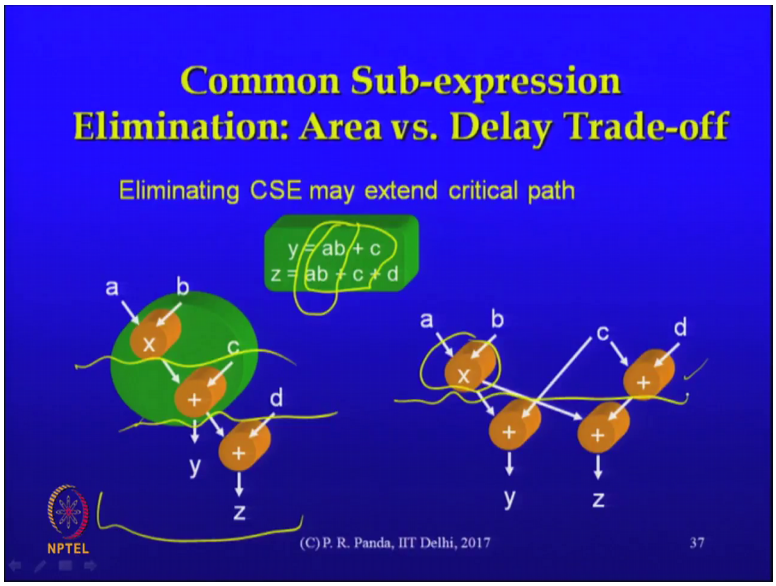


So, let us say I had a times b plus a times c this would result in some kind of a a graph right, but equivalently.

Student: a star.

Yeah if you say a star b plus c that results in a different graph that structurally these are different things. So, even if you had an algorithm for doing graph isomorphism sub graph isomorphism well that too would not be able to detect that these two instances are there and I need to perform that computation only once. So, that difficulty is there, but I would like to point out a an orthogonal effect here just because it is applicable, just because we have detected that two sub-expressions are common, it does not necessarily follow that we should actually extract the common sub-expression out and do their kind of replacement that is being suggested here.

(Refer Slide Time: 07:10)



We should be able to illustrate that through a simple example. Consider these two lines of code I have a b plus c is one computation; and a b plus c plus d is a different computation. If I aggressively perform common sub-expression elimination, then I get that graph like this and in fact it is a b plus c that is common across the two computations. So, I can say that I will perform a b plus c first and that is why and that result is also used in the computation of z that is one way of doing it.

Of course, as part of the common sub-expression formulation, it would be that I will try to maximize the size of that common sub-expression, the bigger the size of that common expression the more computations I am saving. So, it might seem that that about I should do, but consider that I do not take a b plus c as common sub-expression. And I just take a b as the common sub-expression. This is not as aggressive as I would have otherwise they are not as we did here, but it is just this much that is common that is used in y computation as before, but it is used in the z computation in a different way where I am actually performing that c plus d separately. So, these are two different ways of performing common sub-expression elimination; in fact, intuitively it seemed as though the first one actually is the more aggressive one, there should be the one that is preferred, but should we prefer that or not what is the situation that it depends upon.

Student: (Refer Time: 08:50) a b and c.

Yeah this is actually not so obvious whether you should perform it or not, because let us say we assume these are all single cycle operations. Then the way we have restructured this graph after common sub-expression elimination in the first case, when you do scheduling you would have to use three cycles to schedule this code. Whereas, the other alternative, you need only two cycles if each was a single cycle operation. So, how did this arise, it seems as though the lesson from here is that I should not blindly perform common sub-expression elimination in the most aggressive way.

So, an algorithm in synthesis that performs common sub-expression elimination somehow has to be aware of possibilities like these. And go ahead with the application of such an optimization only when it is clear that there is a benefit. When does this optimization take place, we already said that these are high-level transformations which means that we perform these first very early on in the synthesis flow much before any synthesis algorithm is run. Specifically, we have not done scheduling yet this is only a graph transformation, we read the parse tree into a data structure and we are performing all these manipulations on that data structure internal representation. We have not come to any synthesis strategy yet scheduling, register allocation all of those things have not yet been performed.

And yet this transformation has an effect on what happens later on, scheduling is going to come later. And when you do scheduling you find that actually what seemed like a good optimization, was not such a good optimization, but it is an interesting circular problem. And there is a phase ordering problem, ideally you would have liked to see effect of doing this and only if it is a good idea then you should be doing this transformation, but how do you find out that it is a good idea or not, you actually have to perform scheduling. Here if I were to perform scheduling I would know, but scheduling comes much later right, it would actually complicate this parse significantly because this may be may not be the only possibility we have maybe there are twenty possibilities of common sub-expression elimination.

And as the expressions become more complex you can easily see that there are there might be many possibilities of common expressions of various levels of complexity that are candidates, and all of them must not be performed. And the way to resolve it is to actually do scheduling, but you cannot afford to do scheduling on every one of those candidates. This is a standard design space exploration it shows up in many different

context it shows up in physical design versus high-level optimization right we have many candidates and all of them you do not know the effect of in fact, the effect might show up in physical design.

These kind of problems we must be alert to in a design flow like this. There could be many different ways of addressing it, but often to reduce the design space exploration time, we might rely on some kinds of estimations. What is the expected impact on the scheduling, we do not actually perform scheduling, but you say that based on whatever constraints are there, based on what information is available perhaps we can estimate the effect. There is no guarantee for that matter a number of these steps there is no guarantee. So, it is all right that if you cannot guarantee that this is the best possible solution, but it hopefully takes you in the right direction as far as the optimization is concerned.

This assignment that we are giving out we will pose a problem of this nature where it is a high-level optimization that you have to perform, but the effect shows up later and we do not want you to actually iteratively perform detailed synthesis on every one of the candidates. So, have some kind of a prediction mechanism that guides that high-level transformation, but this is a classical example of an optimization that makes sense in a compiler. Within a compiler the left one makes sense, performing the common sub-expression elimination aggressively makes sense because depends again on how you are going to evaluate it. But let us say it is the number of instructions that you are counting, then you have fewer instructions in the first one if you aggressively apply it and as long as assumption is there it is a single thread of execution single pipeline and so on then it is actually a good idea to do this.

But in the synthesis context, we may have more resources to play with. And if that second resource is there, there is a resource versus time trade off right area versus time trade off the second solution requires two adders, but you might be aware that such a resource is already available, the second adder is already available for other reasons. If that knowledge is there then the transformations can be guided in some way right.

Student: Sir, how delays increasing the right point?

Not increasing, it is decreasing.

Student: Decreasing.

Area may increase because you have to two adders.

Student: (Refer Time: 14:31).

In order to perform both of these operations in the same cycle, you need two adder. So, the resources you need for this solution is two adders and one multiplier that is what you need, but in the first one, one adder and one multiplier is enough.

Student: Sir, but then that compilation time also (Refer Time: 14:49) because we are doing that optimization.

The compilation time will increase the moment you perform any optimization or if you perform any aggressive optimization, the compilation time does increase.

Student: (Refer Time: 15:04).

In fact, we are implicitly taking that into account. When I say you estimate instead of actually performing the scheduling, what is the point, why are we suggesting that you do not perform the scheduling, it is because there is limited compilation time. If there are hundred possible choices, and we are trying to resolve between those and choose the best one that expected to be the best. Thus the default would be you just do not scheduling on all of them that is part of the compilation time. So, in going for a methodology like this where we are not explicitly enumerated all the choices and doing synthesis on all of them, we are trying to reduce the compilation time.

Having said that; it is always true that every one of these optimizations that we perform adds to the compilation time in some way or the other. Normally compilers or synthesis tools you are able to control in a little more fine grained way, the level of optimization that you want it to perform. If it is a high-level more optimization passes are run or each of those passes is run in more depth more detail and therefore you can expect higher compilation times, but hopefully also correspondingly better quality designs.

Student: Would be run time of subsequence depth like elaboration or the function unit by steps, scheduling steps also considered into account when we come up with an estimation algorithm, so that even is my compiler time increases by delta I am getting a two delta benefit or three delta benefit in my.

Yes while we would not have the time to discuss that in detail. The principle is what is being illustrated here which is that there are these number of passes we have to go through them sequentially, we do not want to combine all of them in the same algorithm just in the interest of engineering simplicity. And of course, the complexity of the algorithm needs to be taken into account because of that there is a sequentialization that is done; ideally to solve this problem well you should do scheduling also as part of the high-level transformation.

Student: (Refer Time: 17:28).

You could do that, but that complicates the algorithm so much. And why is the scheduling register allocation is also all of these are dependent on each other, no matter what ordering you choose sometimes there is a choice in the ordering, but each pass is going to affect the quality. Quality is one, but what you are pointing out is that the compilation time might also be influenced. In this example, it is not clear, but in a different example that we will come to that is the real concern where you are making a transformation locally it seems as though that transformation is. It can only help improve the quality, but it might increase the complexity of some other paths so much that you have to limit the extent to which you perform that transformation.

We look at the examples soon here it might not be that these common sub-expressions that we are extracting might not make that much difference to the scheduling. Here if I look at this transformation versus the scheduling, what is the impact on scheduling. There is a possible positive impact on the quality of the scheduling results. There is an impact on the scheduling time also the time taken for the scheduler to run that is dependent on the number of nodes and adjacent that you have in the graph you are transforming the graph. And therefore, there is an impact, but probably not so much of an impact in the general case there may be special cases where the transformations effect is so much that the graph size becomes multiplied by a large number such pathological cases might also be there. Often it might not arise here, but we will soon get to examples where we do need to worry about that.

Student: Excuse me sir.

Yeah.

Student: At the compiler time what we will scheduling comprise of will it be like just traversing all the graphs all possible graph.

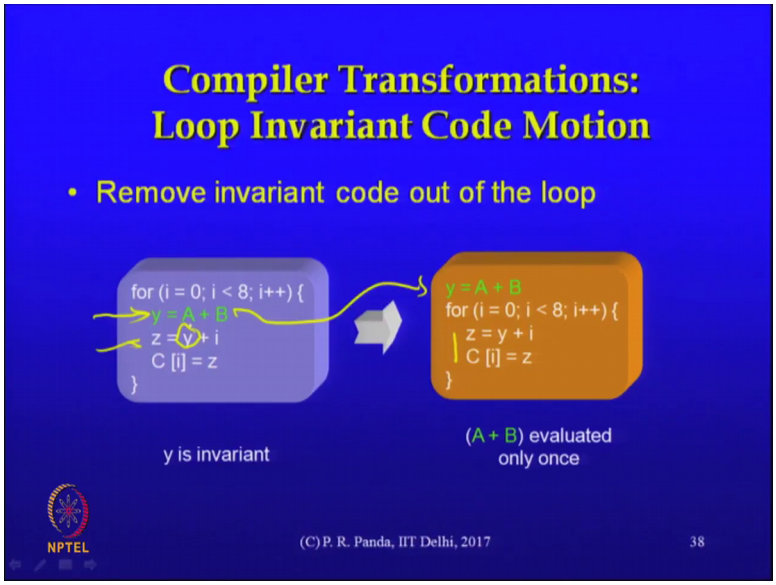
What will scheduling comprise of, why do not we talk about it when.

Student: Ok.

Discussion; so this remember the overall ordering what we are talking about here is a high-level transformation of the graph itself of the internal representation itself. We have not done scheduling the we are pointing out here the effect on scheduling which will happen later, but we have not done scheduling yet, but it will be done soon. So, why do not be when we discuss the scheduling we can talk about what is the input to the scheduling and go (Refer Time: 19:55).

Student: (Refer Time: 19:56).

(Refer Slide Time: 19:58)



A few other transformations are worth looking at loop invariant code motion refers to computations that are specified within loops, but the results of which are not changing across different loop iterations. So, because I have some computation here A plus B since A and B are not changing within the loop, I can move that computation out of the loop so that I perform it only once and the rest of the body is the same as before. What is the impact of a transformation like this, does it help in synthesis?

This is another example of a compiler transformation. It helps in compilation, why, because there is that instruction this translates to an instruction right. The add instruction if the loop is going to go through eight iterations then that add instruction would be executed only once as opposed to eight times and it is clear. In the synthesis context does it help?

Student: Yes

It helps, why does it help?

Student: The cycles (Refer Time: 21:06).

We accounting cycles, and we have to argue that it leads to fewer cycles. So, question is does it lead to fewer cycles right.

Student: Depending on if I am limited values sources if I have sorry.

Student: As many resources I can use in the.

Yeah.

Student: And then can paralyze all of them, I can still if with same code.

Actually these two statements are independent of each other, well actually they are not independent of each other, there is a dependence between these two statements. But that part is independent the y part of it is independent of the loop iterations, but the i is changing though over the different iterations. If I did not take this into account what would my graph B for the left loop body. So, let me exclude the loop, let me just have the graph for the loop body right, so that is the data flow graph.

So, I have a plus operation and this is my A and B, another add that is I and that is my z. What happens to that last statement, this is a graph. So, what do I write for that c i equals z what kind of statement is it, it is assigning to an array in the programming language context. So, I have to create a node and edge for it that node represents what?

Student: Operation.

Write operation into the array which in terms of hardware would mean what kind of thing. I have to have a piece of hardware in mind for every statement, every expression I have right in the.

Student: Sir.

So, what hardware is this?

Student: Sir, mux (Refer Time: 22:46).

Student: Register push.

First of all what are the inputs to that operation, what are the outputs that is the way to analyze what is the?

Student: Input sir one and output is the (Refer Time: 23:02).

Student: Input is i and z.

Student: (Refer Time: 23:05).

This is essentially a memory write operation that array C I did not show the declaration here, but you can assume that there are some elements in an array, and we are indexing that array that might usually translate to some storage in a memory. And this is an indexing into that storage and storing value there that is what it is it is a memory write operation. So, to that memory write operation what would the inputs be and what would the outputs be?

Student: Address.

Address should be there let us just as further the case of simplicity assume that somehow i is the address because that is what the index is. So, this applies if I have a separate piece of memory just for that array c. If I had many arrays then there is a some other computation that mapped to the same memory module then I have to generate that address in some way that also should become part of my computation, my data flow graph should have those operation address computation operations also. But otherwise let me say i is the address and what is the data?

Student: (Refer Time: 24:15).

So, those so somehow I need an address and a data as the inputs to a generic memory write operation. What about the output?

Student: That.

Output is.

Student: (Refer Time: 24:35).

The modified version of that array I remember so, but what do I put here? The outputs are the kind of things that one would get connected to some other operation that is what we do output means here does I introduce an edge and that edge represents dependency here. So, what would that be for the memory write operation? Remember for an equivalent add operation my inputs were the two operands and the output is the sum sometimes the carry also is another output from depending on whether you are using it or not. But that is how I constructed my other nodes that I am just now worried about how to construct the node to represent a memory write faithfully.

Student: Sir, should give the location of the place have written the data.

Student: Or like status if it was successful or not memory write.

Well, we could make the assumption that the memory write takes a variable amount of time I do not know how much time it takes. If that is the protocol then you can say that you will need some status signal saying that it is done without which you cannot proceed with for example, using elements of that memory because if the operation itself might not be complete. But I mean that is an interesting variation but to simplify things let us assume that I know what is the maximum time required for a memory read or a memory write operation that puts it in the same class as an add operation. In fact, even an add takes a variable amount of time depending on the value of the operand so but when we say an adder delay is 5 nanosecond what do you mean.

Student: Maximum.

It is the max delay irrespective of what the operands are it might cause the carry to propagate all the way from the LSB to the MSB that is the time that we take into account when we say the adder delay is 5 nano second, it is the worst case delay. So, similarly to

simplify the memory read operation, which is not a bad idea actually, it is not actually it is not as varying as a simple SRAM access is not complicated. There is not that much of variation write time or the read time.

So, let us assume that the delay is fixed time it requires to if this is 2 nanosecond then that is fixed that just to simplify. Just because that is what we have assumed scheduler cannot work remember if your operations have a variable amount of delay or at least it cannot work in the simple mode that we have seen so far. Because you have an operation if you do not know whether it takes one cycle, two cycle, three cycles and so on, how do you actually split that in two clock cycles, because you would also like to share you all other operations here this is being done statically. So, in fact, you will be forced to not schedule anything in parallel and you wait for the handshaking signal to come from the memory before you can do anything else. It can be done, it is done this way if you have a more complex memory like a DRAM or something like that, but for the simpler memories let us just treat it as just another operation just so that it can fit into the simple scheduling formulation we have now.

Let us introduce the memory write node here. So, at least the inputs are clear, z is an input, i is that other input so that is the rest of my graph output actually is not clear from a memory write operation what output you take. In fact, there need not be any output right. As opposed to a memory read operation where what are the inputs, what are the outputs?

Student: Address.

The address as before is an input.

Student: Then an output.

Then well actually you need only an address nothing else. And the output is the data. Here it is clear. You produce the address and after a fixed number of cycles that is known because the memory has been characterized earlier for its worst-case delay, I have the data. This kind of a node you can see how to incorporate in a clean way inside the data flow graph.

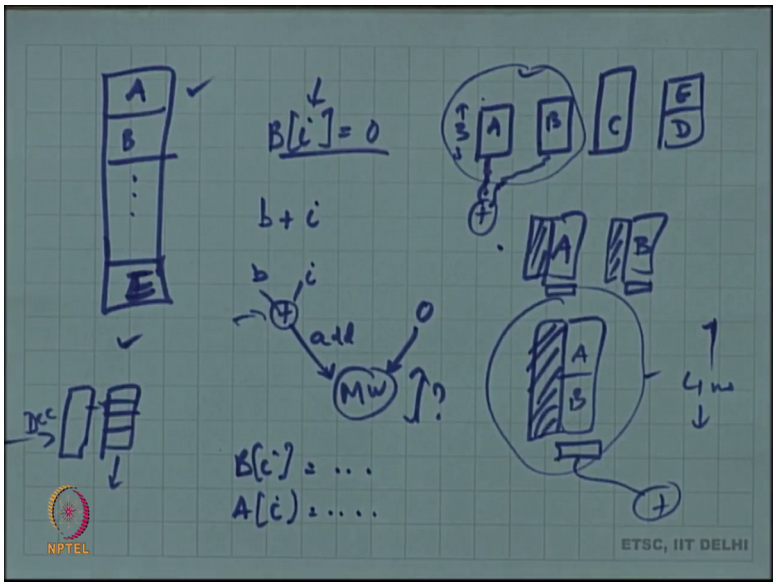
Student: if my memory has been structured prior to this decisions, then I do not need an option tell how many to fetch, but otherwise the word size or byte whatever I am trying to delay.

How many bits to fetch yeah, there is an assumption when I say memory what kind of memory it is, memory is characterized by at least some elementary operations one is the number of words and the width of each word. This is not obvious as actually the what this is pointing out is another set of decisions need to be taken for example, is c the only content of that memory or are there other contents that determines the size of the memory. In fact, that determines the delay of the memory.

Student: Yes.

Delay of the memory is a function of the number of words in the memory. So, before we proceed to scheduling and other such things, we do need to finalize the mapping of the variables on my arrays in the data to the memories. This is a different problem by itself that also needs to be solved as part of the synthesis strategy which is you may have ten arrays in your design. Now, how are they placed into your memories?

(Refer Slide Time: 30:23)



One architecture could be that you just have one memory module this is array A, array B and so on this is my array J. So, I have all of these arrays all of them are mapped to the same physical memory. What is the implication? Now, when you say B of i this

translates to a calculation of the address, because now i is not enough for me to say what the location is. So, you may have some base address plus i may be a multiplication if necessary depending on how you are addressing it.

So, some computation is first performed, so that is the address computation b is the base address for that array capital B. It is a constant because we might know that it is a constant the sizes might be known, but we still have to actually schedule an address computation then that becomes the address to the memory write, and the data is zero in this case. So, such a computation has to be part of the graph that is being scheduled, it is an operation and that operation needs to be explicitly scheduled as part of our scheduling strategy, but this is not the only way to organize our hardware.

Remember unlike in a processor where the memory architecture decision has already been taken by the processor designer, and effectively what is available to us is one large address space and we have to just decide the location and of all my data elements within that address space. Here we have the flexibility to choose our memory architecture in the way that is suitable for our particular application that is being synthesized. What is an alternative architecture, instead of storing all the arrays in the same memory module what could I do?

Student: (Refer Time: 32:34), sir we can break it in smaller chunks depending on what kind of control logic I need where how much I want.

Yeah there are implications, but clearly this is not the only solution. So, other extreme could be, one extreme is you store all the arrays in one memory, yeah you just map all the arrays to different memories. There is nothing that is preventing you A is a separate physical memory, we are generating the hardware remember. We have the flexibility to generate whatever kind of hardware we want our optimization functions are area and timing and so on, but depends on whatever we are optimizing what is the advantage and what is the disadvantage, let us see of doing things this way. These are variable sizes are not the same, so that might translate to memories of different sizes. And maybe the D and E some of those arrays could be actually mapped into the same physical module, but in general there is nothing that is restricting us to store all of the data in the same memory module.

And other extreme would be every array becomes an independent memory, but the general solution could be that you do some sharing and where it is necessary you can actually keep them out as separate modules. What would guide such a decision? This is not necessarily part of the loop invariant code motion discussion obviously, this is a different discussion, but in general this falls within the area versus delayed trade off that we are talking about in what way. So, let us say these are five arrays and here we had five. So, these are two possible memory architectures for just storage related decisions the rest of the designs not changing, but I have the option of storing A and B in different physical modules or the same module. What would the trade of be, what is the advantage of doing so and what is the disadvantage?

Student: Sir, control logic need may need to be replicated for each physical (Refer Time: 34:40).

Something would replicate the memory is not merely the bunch of locations, there is in addition to all these locations that consist of registers or capacitors or however you are implementing an individual cell in the memory. There is a decoding logic when the address comes, there is some decoding logic that ultimately says that that is the element that was addressed, and then I will read it out. If you have separate memory modules, so for A you would have a decoding logic for B, you would also have a decoding logic. Of course, if you combine A and B into one memory module, you also have decode in fact, you have a bigger decoder then those, but in general there is some overhead where decoder is one then the other part is what is called as sense amplifier if you are familiar with the way memories are designed.

So, just keeping this discussion simple, there is some circuit that when the data is read tries to as fast as possible detect whether it was a 0 or a 1 that there would be two instances off here. Here there may be only one instance so width is the same in all of these cases. So, the point is that when you split it up in this way you have a little bit of an area penalty when you are splitting them up. There are a number of things that come into the play routing overhead is the other thing that comes into the picture.

Student: Independent.

Because you read something from here or read something from there, ultimately maybe they are going to the same adder. So, you need to have these routes right both from that

memory and from that other memory. Instead, in this case there would be only route from one so that is a little bit of intangible thing right now we are not able to see the impact. Impact is there how much impact it is hard to say, but in general you can assume that if everything is compacted together into one memory from an area efficiency point of view, it might be better. So, there would be an overhead by splitting.

Student: (Refer Time: 37:01).

So, that is one thing. Other thing is what about time now I have to do a similar analysis with respect to (Refer Time: 37:10).

Student: Sir, behavior level may be timing you have to be same, but at physical level it is non-trivial to (Refer Time: 37:16).

Well, for the behavioral synthesis timing is a parameter for the memory write operation, how many cycles is an input right. For every operation, the library element we are assuming has a delay associated with it. So, here the memory write would have an associated delay which would be better in this case or in this case.

Student: Separated one will be better.

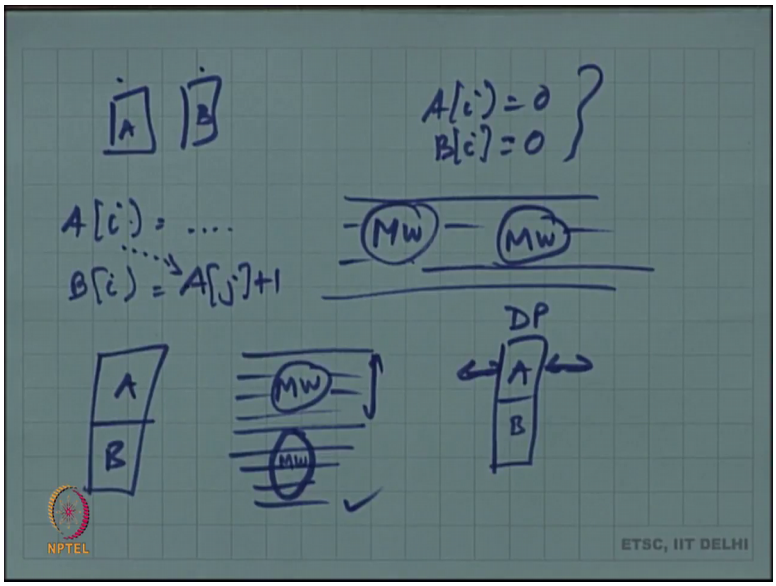
The smaller memory would have a lower access time that is of course assumed; timing wise it might actually be better. But from a scheduling point of view can you see which one might be better I have two arrays A, B; and the choice is between assigning them to different physical memories or the same physical memories, so this choice versus this choice. And I have operations B i equal to something, I have A i equal to something else these operations translate to the DFG nodes we talked about. And they need to be schedule which is better from a scheduling point of view one thing is of course, that if I split them up then the individual delays might be different let us say this is 3, and this is 4 nanosecond, time to access it as it is 4 here, and here it is 3.

Student: (Refer Time: 38:41) one cycle.

Yes, there is a parallelism related advantage when we split things into different modules like this. Let us assume that these are single port memories.

Student: That is what I was (Refer Time: 38:57).

(Refer Slide Time: 38:52)



Yeah single port memory means that at any point in time you can access only one element that. So, then when I have my A i equal to 0, B i equals 0, ultimately there is a two different memory write operation right. These memory write operations could be scheduled in parallel. If it is three cycles fine, this is three takes three cycles, but the nodes corresponding to the two physical memories to different physical memories would not be conflicting in a scheduled, you know they are independent modules, they can be activated simultaneously the accesses can proceed simultaneously, so that is till three cycles.

But more important is that the two accesses could be scheduled in parallel. As long as there is no dependency data related dependency between those two accesses which is here. What is an example of a data related dependency that will prevent you from doing simultaneous access even though A and B are in different modules. I have A i equal to something and let us say I have B i.

Student: Equals to (Refer Time: 40:17).

Right equal to some a j let us say plus one some such. So, there might be a dependence between as of now I do not know I unless disambiguate and prove that i and j can never be the same that is a different argument. But you see that this kind of a dependency may sequentialize the two memory accesses even if the B and A are different.

Student: Even if i and j they are not same, since I have a single port memory I cannot schedule them.

Right this would be multiple accesses to the same memory which would not be allowed.

Student: (Refer Time: 40:53).

So, there are some tradeoffs there. We introduced it in the context of how to implement that C i equal to z statement. But this is one thing as opposed to if I had a single module in which A and B were both mapped, what would the schedule be? I have two sequentialize them because it is a single port memory, and I have a memory right even though there is no dependence between the two statements, I have to sequentialize them because the two memory write statements are competing for the same resource. The two memory write nodes are competing for the same resource. If one is on, then the other one you have to wait. Just like you have two different addition operations competing for the same adder, if you have scheduled one addition then the other one has to wait, memory operations can also be thought of as similar right.

So, this was my four cycles. And the other one would be another four that would be the second memory write operation if my cycle time was 1 nanosecond. So, there could be fundamental differences you see that the this implementation is much faster than this on one side it illustrates a very interesting architectural decision that you have to make and then how it influences the schedule. We did not get to multi port memories, but that would be another orthogonal effect to worry about. Alternatively instead of two single port memories it could be that I have a dual port memory. A dual port memory means that I assign both of these to the same module, but two simultaneous accesses are permitted to that memory it designed it in a way of.

Student: Dual port and we have three (Refer Time: 42:58) A, B and c (Refer Time: 43:00)

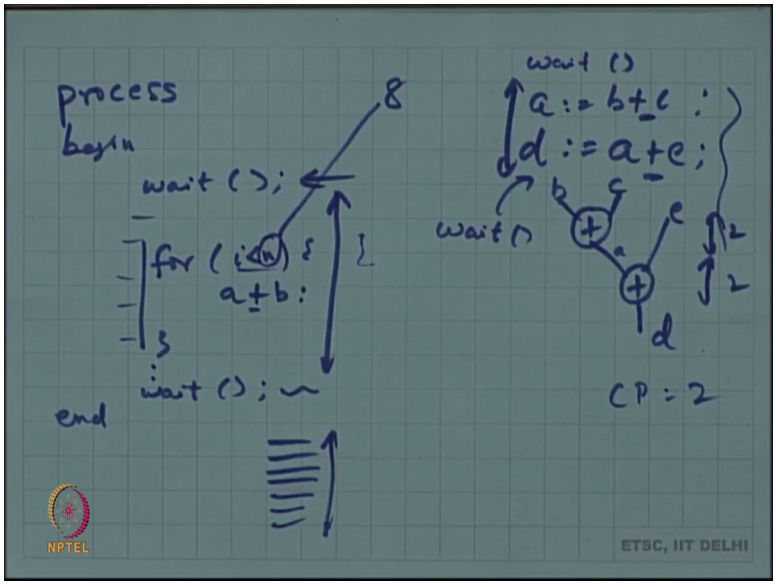
If it is dual port, it means that you have a max of two simultaneous accesses. And if you have more than three that are independent and they are schedulable operation you cannot schedule all three together. So, there too there is a sequential relation that is necessary. I have to also do a similar cost calculation what is the impact. Simultaneously, so if there are up to two accesses we could actually do in parallel, so I could do better than this in

terms of the schedule in terms of the performance. Area of a dual port memory would be larger than the area of a single port memory of the same size. The fact that you permit dual port multiple accesses to that same bit cell of the memory means that you have to redesign that bit cell to have more than one all the way. So, every memory element is larger in a dual port or a multiport memory that was a digression.

Let us get back to first of all whether that loop invariant code motion helped us or not. Just by itself it might not be obvious just like several other transformations, you have to look at the resources sometimes that does take you a little bit further into synthesis steps to figure out whether it is actually helping. But it could belong to that class of optimizations where it is not necessarily improving, but it does not make it any worse. If that kind of an argument you can make then it may be safe to make optimization.

Student: From the language perspective, these for loops will be there in the processes and in process all these are executed once like we have a wait or a synchronization point.

(Refer Slide Time: 44:52)



In a VHDL process, if this will loop were to occur would be that you have a process begin end. Somewhere you have this loop that is an interesting point we overlooked. So, there is no synchronization within that loop itself.

Student: Yeah.

But maybe later on there is a wait, and maybe earlier there was a wait.

Student: But (Refer Time: 45:18).

I have used system c syntax for this wait because for loop was written in a c syntax way, but that wait is similar to the VHDL wait for rising edge your question is.

Student: So, before after first wait and then the next wait, all will be like executed once.

Right.

Student: And assignment will be done only at the second wait. So, this moving this y is equal to a plus b would not make a difference in the cycles.

There is a difference between the way the simulation works, and the way the synthesis will work like I pointed out earlier the two may not be perfectly equivalent. Simulation wise how is this treated. I was here in some let us say end up some delta, and all of this is happening in fact in how much time?

Student: Same.

zero times.

Student: Zero time.

Within the same delta.

Student: Delta.

Right that is what the implication is here. But do you think there is a hope of implementing this complicated loop in zero time?

Student: No.

What about one cycle? Actually it is a little tricky we should look at how to implement a for loop in general if you have a bunch of statements of course, you cannot implement in zero time, because every statement takes some amount of time. Let us say it is a simple combinational statement a equals b plus c, d equals a plus e and so on. So, these operations are there. Can I do all of this in the same clock cycle?

Student: Yes, in the same combinational.

I could if I had enough time. There is a dependency here right, it is that kind of b c that is a and that is e and that is that is my graph. Is this graph schedulable, the whole graph schedulable in one clock cycle?

Student: Yes.

Yes, it could be if the clock cycle is appropriately long then we could do it. So, I could have the very complex set of statements and all of that could still be schedulable. Our scheduling algorithm should be such that it is able to pack all of these as long as it fits into a clock cycle list. From the graph itself, I do not know; I need that clock cycle information before I decide how to schedule it and how many cycles it translates to.

What if I had a loop that is what our example here is. And in the loop, I have some bunch of statements here. Even without the loop let us say I had a wait here right, and then I had a wait right. And it was the case that these delays were two this delay was and in fact, the clock period was also two then what happens. You expect a mismatch between the simulation and the synthesis. Why because the simulation everything was simple and you did not give any delay here, these are just variables anyway the assignment happens in zero time. Everything happens in zero time, and even if there was a clock to that process everything certainly finished in one clock cycle. The simulations view is that in fact, any complicated computation you have between two successive waits in fact all of it is done in one clock cycle, but you can see that there is no way the synthesis tool can generate a circuit that obeys the simulations few point.

You do not since the delay here through this combinational logic, there is a combinational logic; it could be done in the clock period if the clock period was wait enough, but this clock period is not wait enough. So, what do we do the flexibility we give to a behavioral synthesis tool is that that ordering should be respected, but what was zero time here or one clock cycle here in the simulation. The tool should have the freedom to expand that into multiple clock cycles, just to produce a realizable design. So, what was one clock cycle might actually expand to a multiple clock cycles.

Now, whether that is accepted or not, it does depend on the scenario or usually it is, but sometimes it could be that the design requirement are such that input change and exactly five cycles later you need the output to change that is an external imposition. You could think of designs where handshaking is taking place, it is part of a bigger design in which

the result is expected after five cycles. And you do not have the freedom to do this. Then if it is not possible you would come back and say there is an error in the scheduling because I could not respect your constraints.

Scheduler should be able to come back and say do the evaluation of whatever its possibility whatever its choices are with respect to resources that are there. But sometimes it might be that the scheduler is not able to give you design that respects all the constraints.

On the one hand there are resource constraints, on the other hand there are dependencies which impose constraints on the scheduler, and you also impose a time deadline related constraint all the constraints might not be simultaneously met. So, that we should be prepared for in general that there may be mismatches timing related mismatches between simulation and synthesis which might be its just that if we are doing validation, we check the result. Here we do not check it in the intermediate stages the synthesis result because we do not know the synthesis tool might have expanded a clock cycle into multiple clock cycles. Because the original specification remember was a functional specification to the synthesis tool, it is the functionality that is specified.

The timing details are not known; even to the designer it might not be known it is. The tool that may come back and say that I could meet your timing or I could not meet your timing or this is the best that I could do considering the constraints that you have placed. So, that is the second thing which is you may have a complex computation between two wait statements and that may take some time. But let me get back to the general question which is I have a loop. How do I handle a loop?

Student: We can consider that as a flat code with each block repeated.

Yes, I need to the thing is that in order to do scheduling I need to know the delays of all the deterministically know the delays of the operations, without that scheduling cannot be done. So, there could be one uncertainty in the for loop which is here I have a plus b and so on, these are all deterministic, there are known operation delays are known, but what is the one thing that might not be known when you have a loop?

Student: Number of times.

Number of times you go through the loop might not be known right. So, this I just said a loop. Here this loop that we have on the slide is deterministic I know exactly there would be eight iterations. But I do not know in general I may have i less than n where n is dynamically computed I might not know there is that leads to some difficulties for scheduling. Because how many cycles do I expand this two, if I do not know n, then this is a little tricky right. So, what do we do? There could be two solutions one is like he said we just expand out everything assume that this is actually statically known some number 8, it could be because the users code is written in that way it could also be that you perform some other optimizations such as.

Student: Constant propagation.

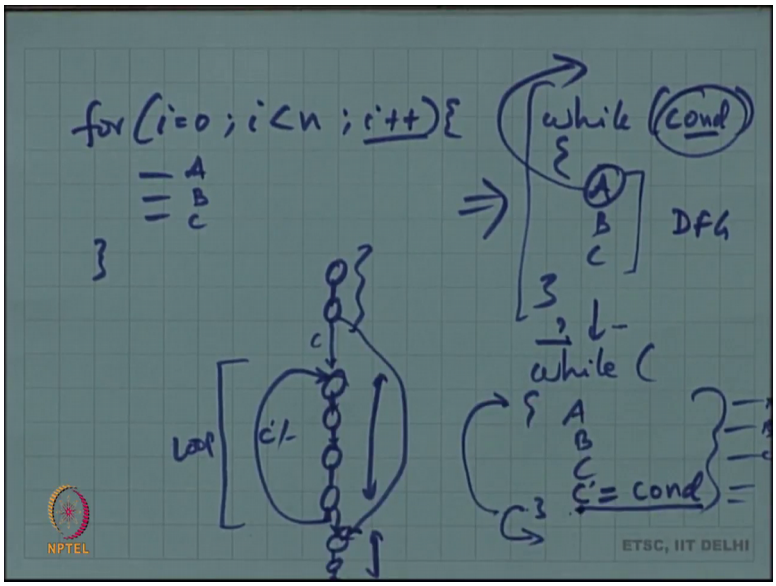
Constant propagation where you had i less than n, but n even though it appears as n here in fact, the value of n is known considering what else you have done with n before you got here. Either way if at compile time we are able to figure out how many iterations and resolve any such uncertainties then there is a one simple solution we could actually expand out the loop into multiple loop bodies. I will soon get to what is the synthesis related consequence of that, but in principle there are eight iterations, you just copy the loop body eight times it is possible at least you have a solution there. It has some very interesting implications, but it is possible.

So, the scheduler now instead of seeing loop with an uncertain number of iterations that it is not able to do anything about is just seeing a bunch of statements larger bunch of statements than the original, but that is the bunch that is going to be scheduled that is one solution. So, this solution works if you are actually able to statically determine the number of iterations. What if you cannot can the synthesis still work if you are actually not able to determine the value of n.

Student: (Refer Time: 54:39).

Student: needs some (Refer Time: 54:40).

(Refer Slide Time: 54:42)



Right. So, I have for i equals 0, i less than n, but this time I have no way of determining n that is possible right, maybe n came from outside it was an input port right. So, there is no assumption you can make plus plus, there are some. Assume these are all variable. So, that the signal related complication is not there of course, we know how to handle that case where I have a mix of variables and signals. Let us say these statements are A, B, C. Can we think of a way to handle this in our schedule in our synthesis, first of all what about a representation how do I even represent this we learned how to represent if statement.

Student: Yes.

We learned how to represent generalize if statement if it is a switch or something then also we know how to do. If a loop how do we represent?

Student: So, if at the end or beginning, so (Refer Time: 55:42) while loop convert it into a.

Yeah these are all equivalent syntactically one can be transformed into the other. So, let us assume it is a for loop or a while loop does not matter it could even be while condition and a loop body. And you know how to transform a loop like that into a loop like this the while loop is the more general one. For example, what happens to this if you translate that into a while loop?

Student: (Refer Time: 56:09).

It has to be put into the body where in the body will you place it you may place it at the end.

Student: End.

Yeah so, but the rules are straightforward for doing that translation. So, we worry about one kind of loop how do you synthesize that loop. Before doing that let us look at how do I represent that loop. So, that I submit the appropriate set of statements appropriate graph to the scheduling. You can see that this part of it is easy I do the same thing whatever I was doing earlier, it is a bunch of statements, there is no loop inside there. And those statements are translated into the graphs in an obvious way, but there is a loop around and closing loop how do I handle.

Student: That control as (Refer Time: 56:52) it should go back if that condition is met.

Right.

Student: The control as end if the control and if that condition is ordered then the controller should you know move.

Yeah. So, I should generate the standard data flow graphs that I normally do for the body; only thing is this condition has to be incorporated. Now, in a simple way, let us first translate this loop like this you just say while. So, I have A, B, C right; after C, want to evaluate that condition right. So, let us put that condition evaluation also here actually that condition evaluation comes in two places one is even before you enter that condition has to be checked.

Student: Yes.

So, that you decide whether you come here or not. But other is let us say you evaluate here, whatever that condition is at the end of this, so that condition evaluation is also part of our DFG at the end. So, and I schedule this part right as usual and at the end of this schedule I have this value whether C is true or false. If it is true then essentially externally I need to go back to that same schedule whatever I had generated; if it is false then I have to somehow get out from there to the following statements. This is a patch

work that the FSM side of things can easily handle this translates to a loop in the FSM right. So, you have this set of statements A, B, C, so they translate to sequential FSM structure each of these cycles some bunch of statements are scheduled right and bunch of operations are scheduled like we have seen earlier. But the last one at the end of which that value that it is a C prime value is known that is the condition that may if it is true lead to right that is the condition remember these were unconditional transitions.

Student: Yes sir.

But at the end of the loop body, you have a conditional transition that takes you back to the first statement, but c can be false also if it is true then you go back there. If it is false then you just proceed to this corresponds to the loop body; after the loop body I had some other basic block that has been scheduled like this and translates to a bunch of cycles like that. And I would adjust the next state accordingly that is how you would do it. In the form of representation, you could actually therefore, think of a hierarchical CDFG structure, CDFG is anyway a hierarchical structure where the loop itself forms a separate level of the hierarchy right.

So, the loop body is scheduled separately and each basic block could be actually scheduled separately. But this packing up has to be done because we understand that this is a separate hierarchy. This is one schedule, this is a different schedule, but they have to be put together in the construction of the overall finite state machine in a way that the conditionals work the way we would like to.

The condition check here actually got duplicated in two places one is the loop body, but this is done even before you enter here right let us say before this you had another bunch of statement so corresponding to some states in the FSM. And here also actually you need to check whether that condition, should I come to the first statement of the loop that is also a conditional thing right, you do not always come. And therefore, that condition needs to be included not just here, but also in that basic block that is preceding the while loop, it also becomes part of that computation and at the end of that you are checking whether that value is prime.

Student: Sir, we can just check the condition once at the first statement of loop.

You could do it in different ways; it is all right. You could also do that here and exit from the loop here to that next statement. It could be done other way of doing it is you actually duplicate here, but if it is true, you come here; if it is false you would know how there are some different ways of making that adjustment. So, that could be the most general way of translating a loop of synthesizing a loop. Considering that this is what it might translate to. Now, get back to this question of loop invariant code motion. If I remove some code from the loop right and move it outside, the loop body, might it help from a performance point of view?

Student: Yes.

It might help in the general case, because it might reduce the length of the schedule the loop body schedule might actually be faster. And therefore, the number of states that you are generating here in the FSM that might also be smaller and therefore overall there could be a difference.