**Optimization Techniques for Digital VLSI Design**

**Dr. Chandan Karfa**
**Dr. Santosh Biswas**
**Department of Computer Science & Engineering**
**Indian Institute of Technology, Guwahati**

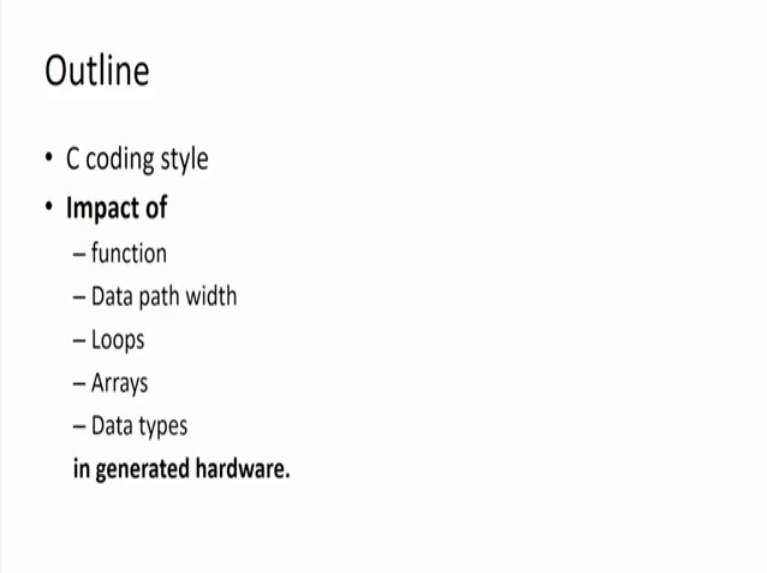
**Lecture – 04**
**Impact of Coding Style on HLS Results**

Welcome everyone. So, we have discussed in last two class about high level synthesis. So, what we have seen that we thus high level synthesis consists of scheduling allocation binding and data path and controller generation. And we have seen in the last class how to automate those steps and we have seen also those steps mostly in peak update we primarily heuristic this algorithm are widely used in high level synthesis industry. So, that part we understand that background algorithm steps and all.

So, now from the designer perspective when you try to write some C codes or the abstract code in MATLAB versus simuling codes in C ++ and try to use some high level industrial high level synthesis tool design the RTL; what are the things you should being aware of what are the things you should keep in your mind when you writing your C code. So, that is that is a something is the discussion topic today; so, today we are going to discuss coating style. So, this coating style is applicable to is basically irrespective of the language specific.

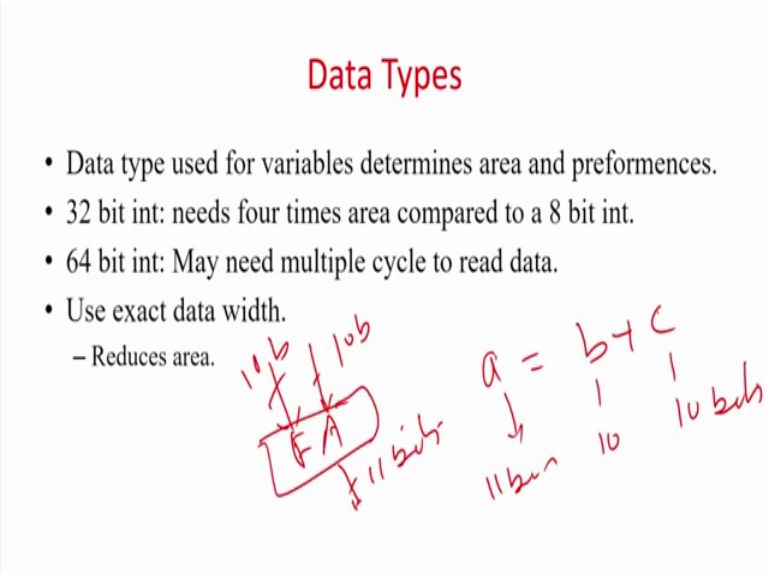
But we are tried to follow C because C is the most commonly used interface for any most of the industrial synthesis high level synthesis tool. So, but this is something you can this kind of logic that data types or say loops add a that are applicable to other language also. So, this style coding style that we are going to talk about is kind of most likely mostly it is language independent and it is applicable for any kind of any kind of language that you are going to use.

(Refer Slide Time: 02:11)




So, the outline of today's presentation is this coding style and primarily we are going to discuss about the impact of different kind of part like function, data types, data path with the loops array when they are they were how to use them how their impacting the generator hardware. So, what are the things you should be aware of or be sincere of or be careful about when you are writing your C codes. So, that the kind of hardware you are going to generate will be optimized one or the efficient one in terms of power or in terms of let us see. So, that is something we are going to discuss.

(Refer Slide Time: 02:45)



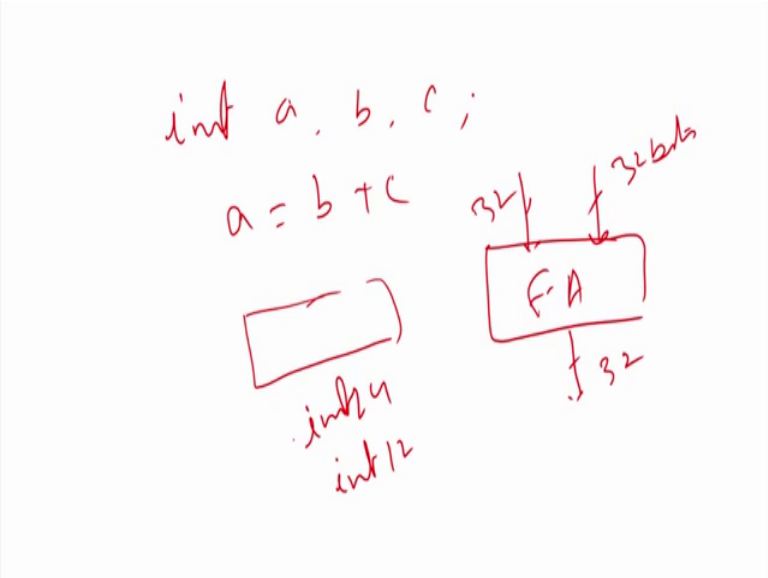

So, first we are going to discuss the data types. So, in C we try mostly use data types like integer long int long int or say float say those kind of data types right. So, when you are

writing C code we have to bother about exactly what kind of input we are going to give right. So, in we define something integer or may be mod values are maybe only.

So, integer most like is basically 32 bits, but some most of the time maybe you guess that your data may not go more than 10 bits or 12 bits, but we do not care those things in when you are writing C code because it does not matter. So, even if it is A 10 bits data particle using 32 bits integer its fine there is no problem, but if you just think about you are going to map it to the hardware. So, when you are going to map it to the hardware that instead of 10 bits if you are writing 0 to 30 bits also 32 C or 32 bits hardware your size of your design goes 3 times bigger than the design that have only 10 bits data path.

So, if you just think about you have say a equal to b plus c right; if you just design this in a and you will say this is all 10 bits right; so, these are all 10 bits. So, this will result it one full adder full adder which is of the inputs are of inputs are of 10 bits, the output is of 11 bits right, but if we just think about if we write in C this a is basically just define this a int a b and C right.

(Refer Slide Time: 04:29)




So, effectively when if I assume that my a b C all integer are 32 bits then the full adder that we are going to generate effectively is of 32 bits right. So, this is all 32 bits right. So, the problem is here this is a full adder. So, if we just write this a equal to b plus c. So, effectively it will generate a full adder of size 32 bits right.

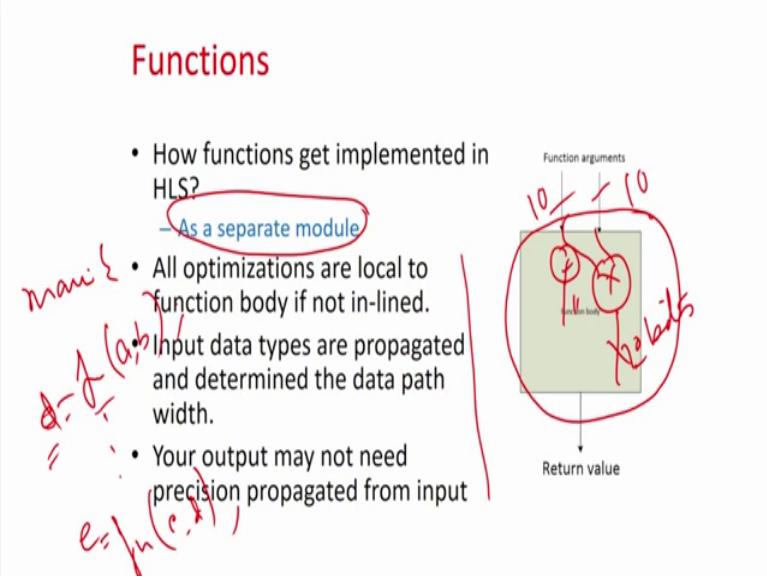
So, the problem is here is that if your data types or the data value we are going to use is very less not integer you should use a very customized integer type which is 10 bits or 12 bits. And you might ask the question how do I represent 10 bits or 12 bits in integer in C right, but most mostly in C we do not have option, but we are going to use this for any synthesis tool that has the option to customize your data width. So, for example, if you go for this xilinx (Refer Time: 05:35) they have this customized data type you can dip here int 12 or say int 24.

So, instead of 32 bit integer you can use 12 bits or say 24 bit integer. So, that is one aspect you should always be careful because when you are writing C code we never give concentration or give weightage on this aspect, but very simple thing just if you using without knowing this random integer, long integer your data path will be higher which is not required for your design most of the time those are basically useless.

So, the data types and very key factor for your design; so, whatever I just mentioned here that data types are used variable determines the area and performance. So, the area and performance will be used by the data types; so that has to be taken care carefully. So, this is something we always missed, but when you are writing some C code or C ++ code trying to map it to RTL through also high level synthesis.

So, you should be always careful about the exact database; you should be you should know that you are in your application area whatever the input with and specifically set your data path with according to your according to our requirement and that will actually reduce the area of your design. So, this is a very simple thing that has to be taken care ok; so, data type which one issue that we should taken care of.

(Refer Slide Time: 06:58)




The next thing you are going to discuss about function; so, we write lot of function. So, when you are writing code we try to make it the code modular; so, use lot of functions. So, you should know how the function will be get mapped into the hardware. So, that you should understand; so, when you write a function the function body will become a module in the RTL right. So, that become a module and whenever there is a call to the function that that particular model get instantiated. So, we why we write function? Because we try to reuse it in multiple places; so, in a hardware also whenever you declare a function; it is basically define as a separate module in the hardware and where ever you call that function we instantiate that module once right.

And now depends on the way that the synthesis tool work may be this two call is not that they are independent. So, you can actually utilize the sim module for both the call. So, it depends on the synthesis tool or most likely they are there is no dependency; they are running in parallel that part has to be there is a basically we have only one instance of the module in the hardware and that we are going to we would use for both the both the calls right. So, so that is something suppose this is your main right.

And you have say function you call this function for say value a b here and after sometime again you call the same function for say c d; so, there are lot of coding between. So, most likely that you have the one module that we instantiated for the function fn and this particular instantiation probably called and it has two input; the

argument will become the inputs and the return value become the output of that function and so, that return types.

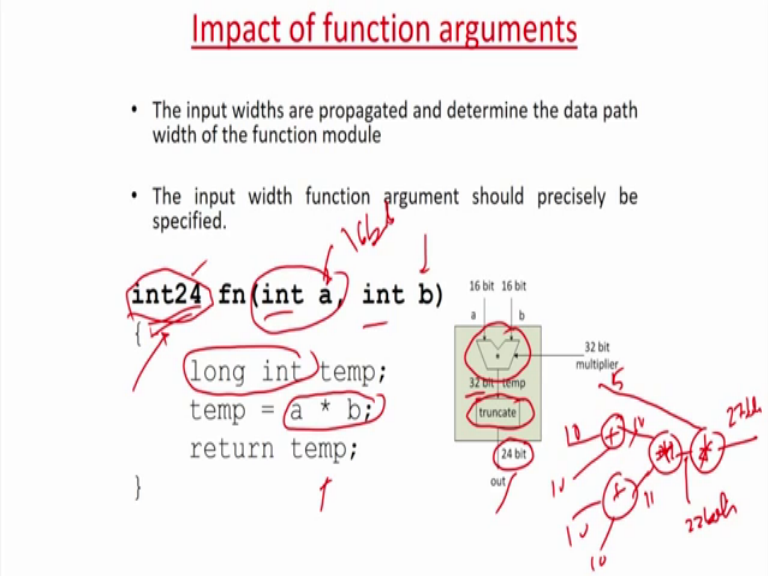
So, this is the module will be created in order and once you call this function first time with a b. So, a b will be passed to the parameter and then it will calculate this and it will return the value that will be stored in variable d. Again when you call this function c d and I am assuming here that is what the dependency. So, we are going to call the same function here with the value c and d it will compute whatever is going to compute here and then return the value that is stored in e, but if there are two calls there is a data dependency or it cannot be paralyzed right that probably it will create the two instance of the same module.

So, that is something in hardware, but a, but what the important point here is that you should understand that how the function get implemented and how it is going to use. So, the most of the time the synthesis to try to maximize the resource usage of those modules and a wherever is possible; so, that is something is going to happen for functions right. So, one more one more point has to be taken care carefully when you are writing a function again the data types. So, how to decide how the data type with data with within the function is determined right; so, that has to be also understood from the designer perspective.

So, the important point is here is that when you write a function the data path width is determined by the input width. So, if your data path say 10 bit; so, if you have an adder here. So, the output will become 11 bit because you are doing this if there is a multiplier in the function body then output will be 20 bits right. So, it gets depends on the 20 bits depends on the size of the input right.

So, your input data path data width is propagated through the behavior to determine the width of the individual module right.

(Refer Slide Time: 10:35)




So, if we just to give an example suppose I have a data path like this say suppose I have a adder here adder here, there is a multi multiplier here and there is an another multiplier here and suppose this is a circuit you are the function is going to decide. So, suppose this is 10, this is 10 and this is a 5 bits.

So, what will be the output width that will that how you decide. So, since this is addition is happening. So, all are 10 bits output will be 11 bits. So, this is 11 bits this is also 11 bits and then you are doing multiplication say there is an multiplication up to 11 bits. So, this output will be 22 bits right and now you are multiplying 22 bits data with the 5 bit data; so, output will be 27 bits right.

So, this is how the data path width is determined it is; so, input width will be propagated to the output and this will (Refer Time: 11:35) the intermediate all the resource intermediate may be the width right; so, that is very important. So, now, you have to be very careful about the input size of the function because whatever the value you are going to give it here that will be propagated and that will determine the size. As I mentioned in my this data type declaring this data types; so, the if you just give arbitrary value like long in int so, unnecessary your datapath will be bigger and most of the things is unused and your resources will be high which is unnecessary.

So, which could have could have optimized right. So, here is an example suppose you have this you have written this function or you passing argument a and b and you are

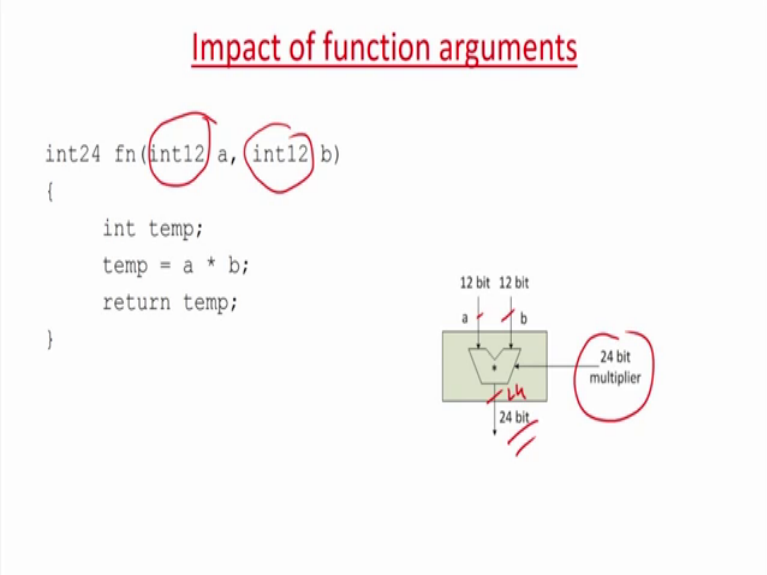
calculating just a star b and you returning that output right. And you define long int here and this is integer and this is integer and output is 24 bits; so, this is a customized version for xilinx.

So, I am using this notation, but for any other tool declaration time may be different, but they have the option to customize your data path way ok. So, this is what we are doing; so, now, what will happen here it will generate a module like this higher since your. So, I am assuming your integer is 16 bit. So, you can assume anything, but I am just assuming it 16 bit.

So, the multiplier will be of 32 bit width right because you are multiplying two 16 bit. So, your multiplier output will be 32 bit, but here your output is only required is 24 bit. So, basically that multiplier that you are having here 32 bit is waste right, you have to truncate 8 bits and you have to take the last 20 LSB 24 bits as the output not the all 32 bits.

So, actually you are unnecessary calculating extra thing which is not required and which is basically getting wasted right. So, instead of writing this is function if you avoid that your since your output required is only 24 bits. So, you no need to give a 16 bit input right you should a 12 bit input right.

(Refer Slide Time: 13:25)

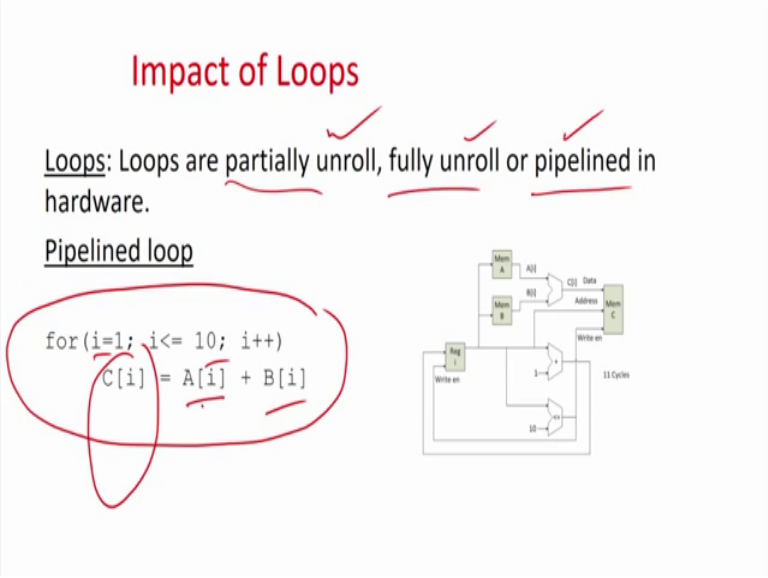


So, if you if you were just writing rewrite this function you should rewrite like this way your integer should be of instead of int it should be integer size of 12.

So, that its not generating extra hardware; so, if you just do this; so, this is 12 bits. So, this will be probably 24 bits and your output is 24 bit. So, the multiplier we generated is a 24 multiplier instead of a 32 bit multiplier; so, this is the advantage. So, this is something you should be always careful. So, just to summarize your function thing you have to remember two things first is that function is become a module and its get instantiated whenever it gets a call and try to the high level synthesis try to utilize those functions that module.

Try to reuse that particular module for multiple causes. So, that is one aspect and the second input is second important point around function is that that data path with within the function is determined by the input data path with. So, you to always specify the exact data path which is required for your design and it shall over just for like general inter long it if you are going to use then you are going to generate some extra hardware. So, you can actually optimize that things if you are little bit careful about that ok.

(Refer Slide Time: 14:42)

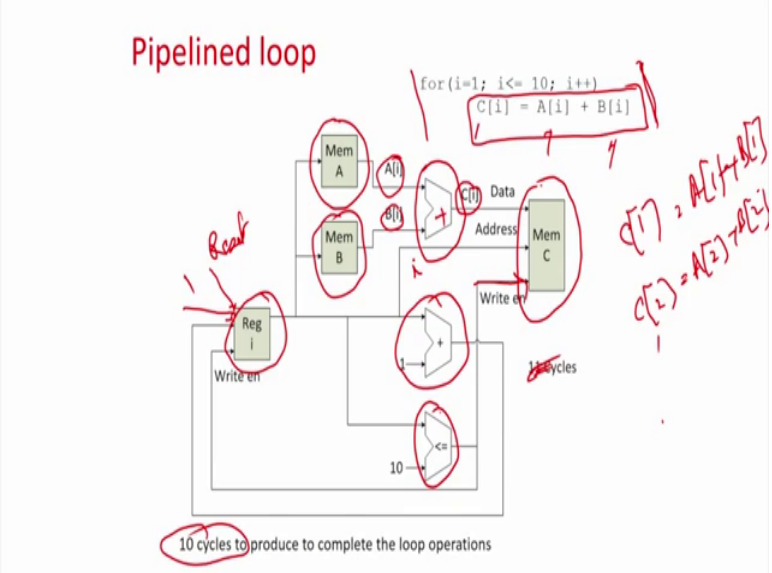



So, now I am I am going to move on to the next topic which is loop. So, whenever you write any code the most important two aspect is array and loop and they have a huge impact on other. So, you have to be very careful when you are writing something involving loops and arrays ok.

So, we will first discuss on loop and then we are going to talk about array how this array will be will be synthesized ok. So, for loop if you have a loop body how it is get implemented; either it is partially get unroll or it is fully unroll or pipelined during in hardware right. So, we will try to discuss that what is that implementation; so, when you write a loop in your program either it will be unroll fully unroll, partially unroll or pipelined based on the design choice you can actually give this choice when there is a control when you are synthesizing this some commercial synthesis tool they have some option to specify that.

So, whether you want to unroll a loop or partially unroll a loop or pipeline that loop right. So, we will just try to see if what kind of hardware it will be generated if we choose this options right. So, suppose I am going to take this simple loop where I am just calculating this C i equal to A i plus B i just adding two array A i B i from 1 to 10. So, this is the loop is going to iterate 10 times and it will calculate as array size C 1 to 10 right using the hardware value A and B ok.

(Refer Slide Time: 16:07)




So, now if you just think about the pipeline version of this implementation though the code is written here. So what is happening here; so, I am assuming this array A and B is used only here and they have only single usage right.

So, I can map those arrays to memory; so, I have inferred a memory for array A; I have inferred a memory for array B; I have inferred a memory for array C right. So, this is

inferred and so, there is a register for i which; so, which is has a initial value reset. So, during reset it will be set to one and then in each iteration that i goes. So, that is the address to this memory.

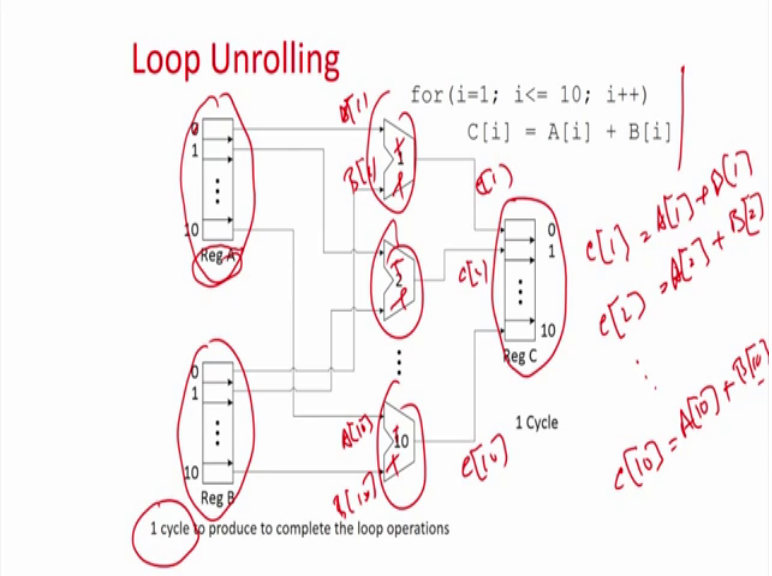
So, A i will be read and here B i will be read and then I will do just A i plus B i which is basically C i and that is getting this is the data for this memory and the address is i again this is i. So, this is how the whole thing will be generated and we will have this implemented this i plus plus. And this data will be return back to i and also I have the component less than equal to I will just check whether the current i less than equal to 10 or not.

So, this will work as the right enable to this array right. So, this C i will be updated until your i is less than equal to 10 right and this is also the right enable to this register ok. So, in abstract level this is the data path will be generated; if you choose to pipeline this array with this loop to pipeline this loop; this is the kind of data path it will be generated ok.

You can see here that I have only one instance of the loop body; so, this is the loop body right. So, the loop body I have only one addition operations and in the hardware I have actually have a one instance of that particular loop body; the operator in the loop body and I have actually pipelining this. So, in first iteration; so, first cycle A 1 equal to sorry C 1 equal to A 1 plus B 1 will be calculated right.

In the second iteration C 2 equal to C 2 equal to A 2 plus B 2 will be calculated right and so, on. So, I need 10 cycle to complete this operation right. So, in the pipeline mode I have going to use only one instance of the loop body whatever the resource required to implement the one iteration of the loop body that kind of hardware will be used and it will be pipelined. So, that the number of iterations is required this is the number of iteration the loop iterates and after those number of iteration the whole data will be available right. So, this is the pipeline implementation of the loop ok.

(Refer Slide Time: 18:16)




The next is unrolling; so, what is that? So, I am going to unroll this. So, I am instead of writing a loop is effectively basically there are tens operations right. So, the 10 operation like C 1 equal to A 1 plus B 1, C 2 equal to A 2 plus B 2 so, on then C; C 10 equal to A 10 plus B 10 right. So, there is effectively there is no loop here right; so, I have only 10 operations.

So, that is what the unrolling means. So, you unroll the loop you eliminate the loop right. So, I have only 10 operations; so, I am going to infer 10 adder this all adder. So, 1 to 10 and since and this is my register storing A, this is the register storing B and this is the register storing C and then this is A 0 sorry A this is. So, A 1, A 1 this is A 1, this is B 1 and so, on you can understand I guess.

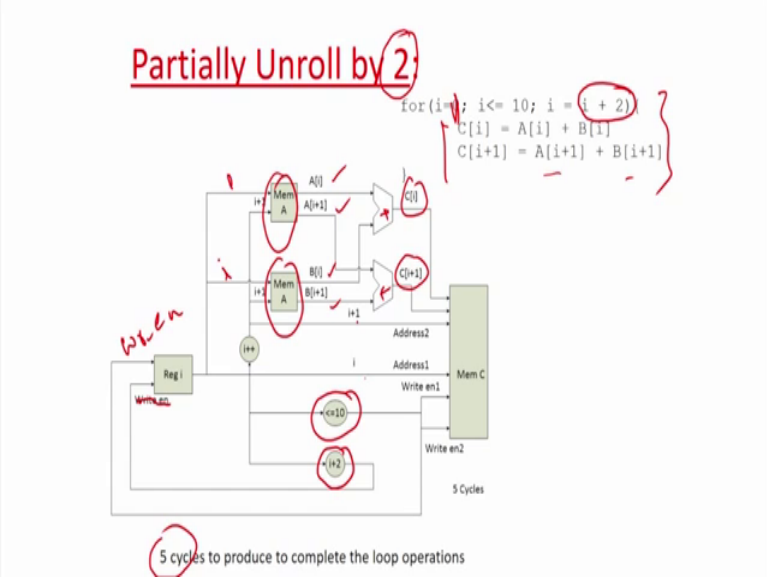
So, this is B 10 this is A 10 right. So, I am actually multiply I am reading array A from 1 to 10 array B from 1 to 10 I am just summing A 1, B 1 and storing it to C 1; this is C 2 and this is C 10 right. So, this is how this whole operation works; so, all this 10 operations is running in parallel and I need only one clock cycle to execute the whole thing right. And there are two things to be note here the first thing is that this is not a memory it is no more because in when you are going to map it something in memory in memory I do not have access to 10 10 access at a same time.

So, in memory you can have maximum one or two access because the memory has a port and you have to access the memory through that port read port and write port. So,

whenever you have more the more accesses you cannot map this register to this array to memory; it will be map it to register and you are reading all this 10 data parallel, where you need 10 instance of the 10 instance of the loop body right. So, these are all adder and you are doing everything in parallel right this is what is loop unrolling ok.

So, this is something you understand you need it very fast latency is only one clock, but it need hardware is number of iteration of the loop right and this is the tradeoff between this two is the partially unroll.

(Refer Slide Time: 21:20)




So, partially unroll means what? I am not going to unroll the loop all 10 times instead I am just partially unroll for example, I gave this for the same example I just unroll by 2. So, the effective code will become this; so, now, I am going to this is 1 to 10. So, this C 1 equal to 1 plus B 1 and this C i plus 1 equal to A i plus B i. So, I unroll one two times and my instead of i plus plus now my loop will be implement by i plus 2 right.

So, that; that means, the loop originally it was iterating 10 times here the loop will be iterating 5 times. And in each iteration I am doing iteration of two of the original code right. So, initially the loop is iterating 10 times; now I am iterating 5 times and in each iteration I am doing two set of operation right iteration of I mean operations of the two iterations in the same loop same iterations here right.

So here what will be the data path will be generated I; I will have generate two again this is kind of pipeline now. So, you after partially unroll you are going to pipeline it right. So, I have two because I now in the iteration I have two operations. So, I am going to infer two adder and the from the register this is my memory A and this is memory B; I am going to read two data right i and i plus 1, this is i and this is i plus 1.

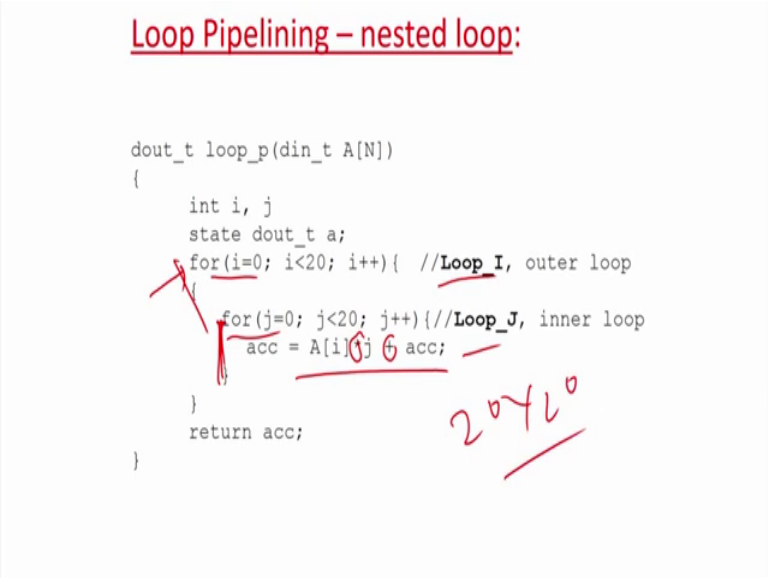
So, I am going to read A i and A i i plus 1 from B; B i and B i plus 1 and then here I am going to do A i plus B i here A i plus 1 into B i plus 1. So, C i will be generated here C i plus 1 and I am going to write two data into the memory C right. And the right address is also I, this is i and this is i plus 1 there are two address and this I less than equal to 10 is the enable right enable signal for the for the port and this now i is incremented by i plus 2 and that is also writing back right.

So, this is also writing back to this to the actual data path right. So, this is how; so, this is the right enable. So, this is how we are going to generate; so, this is kind of a tradeoff between fully unroll versus fully pipeline. Because now I need 5 cycle to complete because the loop is iterating from 5 cycle; I am doing operation of two iteration at the same. So, this is kind of a tradeoff between the actual unrolling versus fully unrolling versus pipeline and which is your good choice it depends on your requirement.

If your design is very want to have execute very fast then you probably unroll is better because if you do not your resource does not matter. But if you are bother about the resource, you are going to use as less resource as possible then you should put properly totally pipeline. So, that you can generate the minimum number of resource and if you want to make a middle of this I mean probably that is 10 cycle is to for too much and the generating 10 adder is too much then you probably go by I mean intermediate trade of by partially unroll and then pipelined right.

So, this is the design choice based on your requirement you have decide, but you have to understand that all the synthesis tool has this kind of facility. So, you have to choose the right option for your design based on your requirement. So, this is what we have discuss for single iteration loop right.

(Refer Slide Time: 24:42)

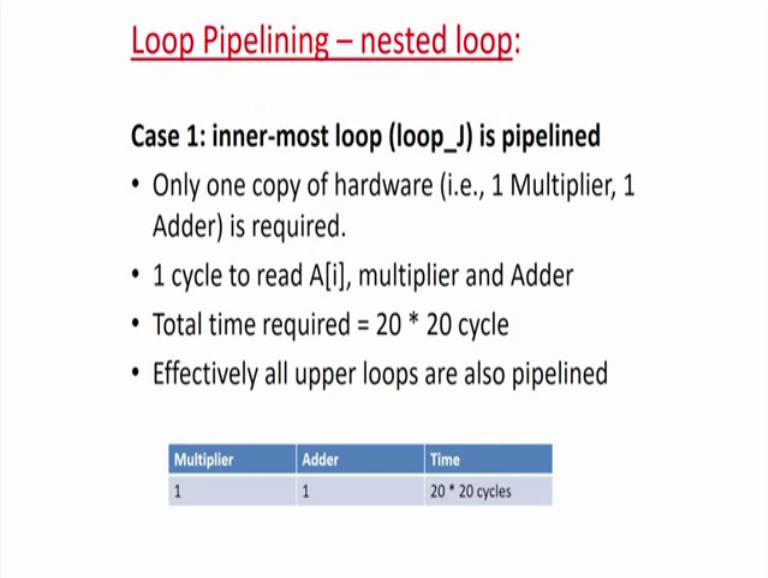



So, now if you just go for the nested loop what kind of design choice you should make right. Because this is common kind of coding style we have nested loop. So, here I took an example of two nested right for i for i and for j. So, loop I and loop J right.

So, if you have nested loop whether how do you choose pipeline, how do you choose unroll, which loop has to be unroll which is better the choice which loop has to be unroll pipelined or should we do both unroll should we do both pipelined? So, these things the choice will come right because nested when your nested can be more I just took an two example nested depth of 2.

So, I am going to discuss now if you have a nested loop how to pipeline and what are the choice you should make for your design ok. So, I took this example of nested left 2 and as I mentioned you have different choice now.

(Refer Slide Time: 25:37)



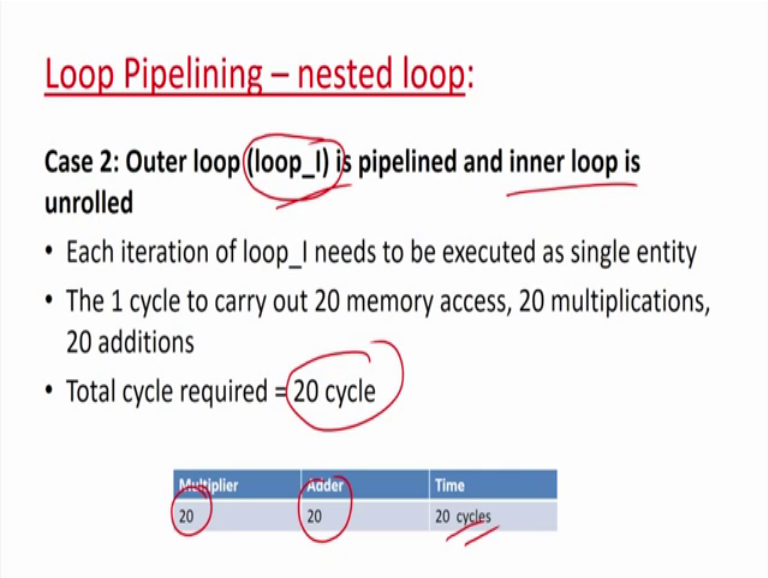

So, the first choice you can made that the inner most of that loop J is pipeline. So, you have to pipeline this loop and if you try to do this automatically the outer loop will also have to pipeline because what I am doing I am just getting one instance of this. So, I need one multiplier and one adder to do this operation.

So, I am going to do this and this is same as this implementation this pipeline implementation I have only one copy this is that pipeline implementation that I have only one copy. So, this way this the inside loop will be sorry this inside loop is a pipeline. So, I have created one instance of this particular loop body and then since this is going to take 20 cycle to compute. So, this will also automatically it will take 20 more 20 into 20; so, 20 into 20 cycles right.

So, the point is here that if you pipeline the inner most loop, the outer loops automatically get pipelined and I need only one instances of the resources I need only one multiplier one adder for this design, but I need 20 into 20 because this loop iterates 20 times this loop iterate 20 times. So, I need 20 to 20 number of cycles right.

But this is something if you choose the case 1 that you try to pipeline the inner most one the loop J. The second option is here is you try to pipeline the outer most loop the loop I right. So, for my example the outer most loop is loop I and inner most loop is the loop J.

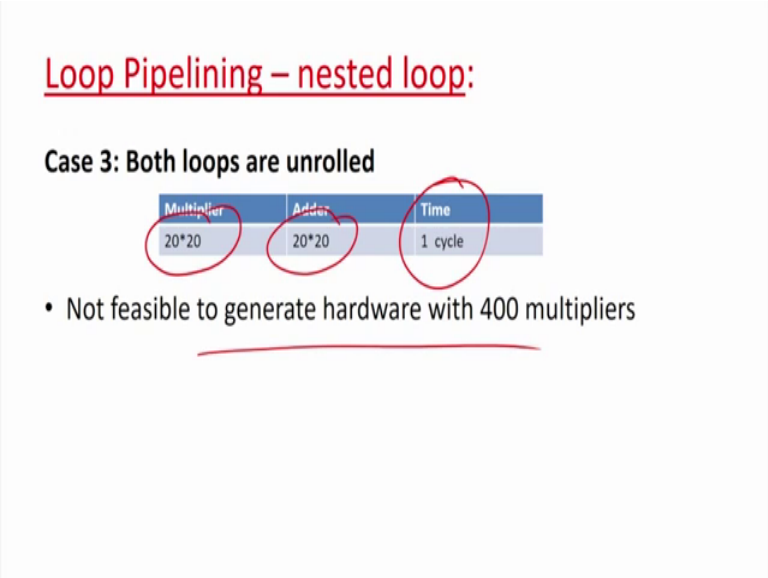
(Refer Slide Time: 27:05)




So, if you try to pipeline the outer loop what will happen automatically the outer loop will get unroll right. So, you are going to create since this loop has 10 iteration it is getting unroll. So, you are going to create 20 instance of the loop right; so, I explain for the other cases.

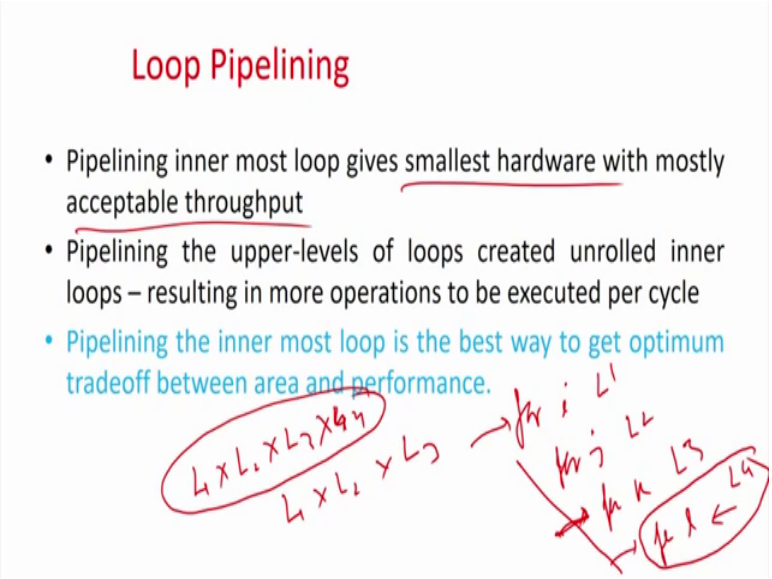
So, what will happen? Here we have to generate 20 instance of the body. So, that is I need 20 multiplier and 20 adder right. So, I need 20 multiplier and 20 adder if you try to pipeline the outer loop and automatically the inner loop is unrolled and I need 20 cycle right I need 20 cycle because the outer loop only get pipelined. So, this is iterating for 20 times and this is unroll; so, this will take one one cycle all the operations will be executed here and this will take 20 cycles. So, I need 20 cycle, but I need 20 multiplier and 20 adder.

(Refer Slide Time: 27:57)




And the third option is that unroll both right. So, where there may basically one iteration you are computing all this operations right. So, I need this 20 into 20; that means, 400 multiplier and 400 adder and I need only one cycle right, but this is feasible not feasible right. So, that that design with 400 multiplied is quite unlikely there; so, this is not at all a choice.

(Refer Slide Time: 28:20)




So, to summarize that when you have a nested loop pipelining you can understand that it is better to smart choice is that you try to pipeline the inner most one as possible. So, if

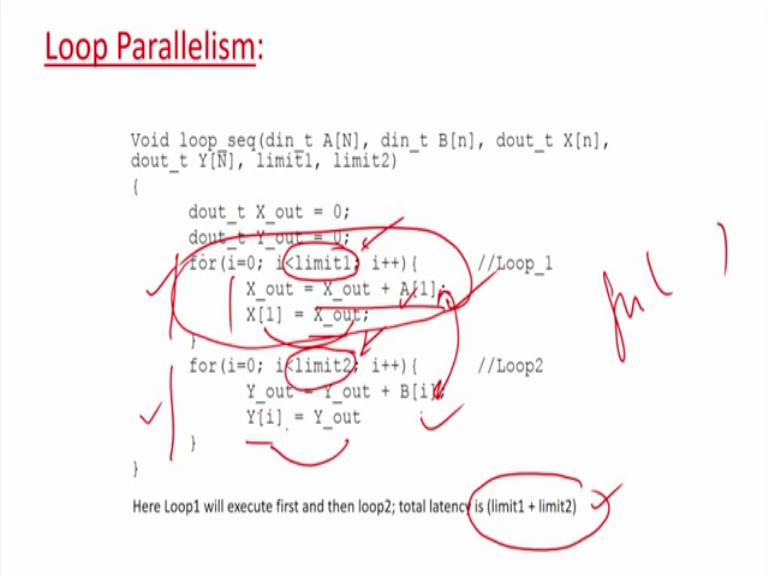
you have a say nested depth of 4; you may be decided that the inner most one you unroll, but you try pipeline the at least second level not the third level or forth level based on your requirement.

But it is better to pipeline the inner most one as much as possible then you will you generate the smallest hardware with acceptable throughput right. So, this is something you have to choose. So, if you have loop depth of say 4 its better to say this is say for I then for j then for k and then for l if not possible. So, if you just pipeline this one all 4 will be pipelined right. So, you need. So, if this iteration L 1 time, this is L l 2, times this is L 3 times and this is L 4 times; your number of time step will be required L 1 into L 2 into L 3 into L 4 right L 4.

But if you pipeline this one and this is you unroll then you need basically L 1 into L 2 into l 3 times right because this is pipeline this is unroll so, but it will generate 24 sorry l 4 times of hardware for this. So, now, it depends of the idea is that it should not pipeline that outer most one as inner most of based on your requirement, you should always try to pipeline the inner most one right I mean if your case if your this number of clock cycle is not acceptable; you probably go for pipelining this, but if you are going to pipeline this one all this 3 will be unroll all this 3 and it will create a huge hardware right.

So, since the nested loop you should always think of pipelining the inner level iterations as much as possible. So, this is what we have discuss about this loop and how to choose to synthesize the loop whether you should pipeline whether you should unroll or whether you should partially unroll right. So, this is what we have discuss about loop.

(Refer Slide Time: 30:32)




Now, I am go to move for the another topic which is called loop parallelism. So, that is another topic; so, so far we have talked about a single lop right. Now if you have say multiple loops and in what happens for example, in this behavior say there are two loops right; this is loop 1 this is loop 2 right. So, if you just write this kind of code what will happen? This will be iterated first and then this will be iterated right; so, this is how the sequential code works.

But in hardware and then the number of; so, if you just write this kind of R T L that will be generated that will also follow this one and what will happen here? The number of time stamp is required since this loop iterates for limit 1 times, this is limit 2 times. So, the number of latency require limit 1 plus limit 2. So, this number of clock cycle is required ok.

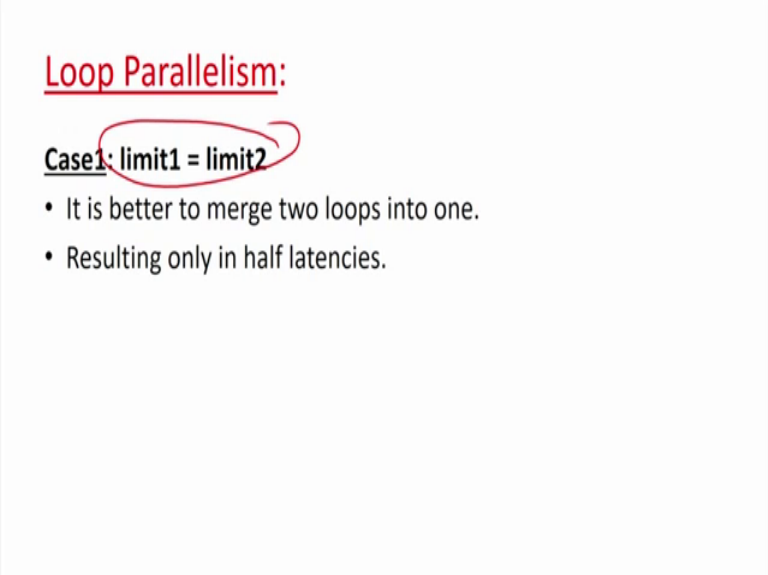
But in hardware you should always think about your hardware adds in parallel basic mistake we always make is that we always do not differentiate how the C works versus how the hardware works. So, in C everything is sequential if you write say 20 lines of code the line 1 will be executed then line 2 line 3 and so, on.

But in hardware if you have say 10 modules all tens run in parallel. So, you it may be that module 1 is doing line 1, module 2 is line you can do like this way [FL] I execute module 1 for line 1, module 2 for line 2 and so, on, but in hardware all 10 are available all the clock.

So, when you are writing your C code you should always try to write such way that you try to make parallelize as much as possible you try to utilize the parallelism inside the hardware as much as possible. So, that is something is the point for this loop parallelisms.

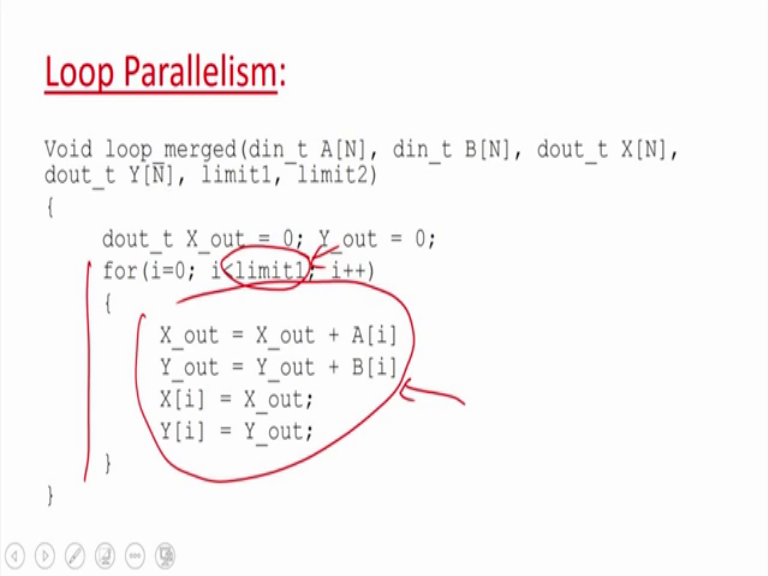
So, what I am trying to talk about is that there are two loops if you just write this way the generated code hardware will take limit 1 plus limit 2 times because this will be executed first and then this will be executed right. Now what will be the alternative the alternative is here is that we try to parallelize this loop as much as possible ok. So, the first possibility is that if these limits are same right; the possibility 1 is if your limits are same you can probably merge this loop.

(Refer Slide Time: 32:38)




Because in the loop you are doing the same operation right you are just doing addition same operations you are doing. So, if you have this scenario you should probably merge this two loop right.

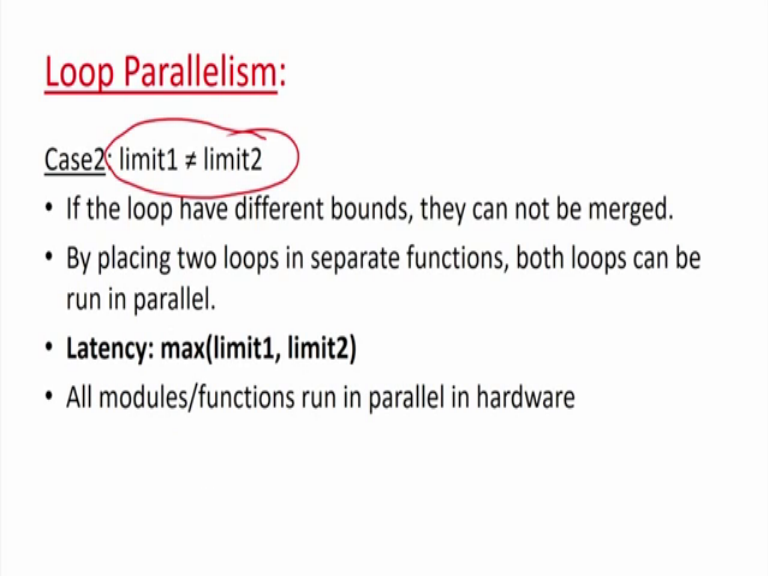
(Refer Slide Time: 32:51)




And the kind of code will be generated this right. So, I just I now I have only one loop and this is limit by limit 1 because limit 1 is equal to limit 2 and all the operations of loop 1 and loop 2 is merged to this and I just generated one iteration. So, the number of cycle is required here is limit 1 instead of limit 1 plus limit 2 I need now only limit 1 number of time step. So, this is very corner case because if the two loop body are same then only you can do that.

But if your loop body are not same there; so, in the next scenario I am going to consider is that your loop body is same, but the limit is different ok.

(Refer Slide Time: 33:33)

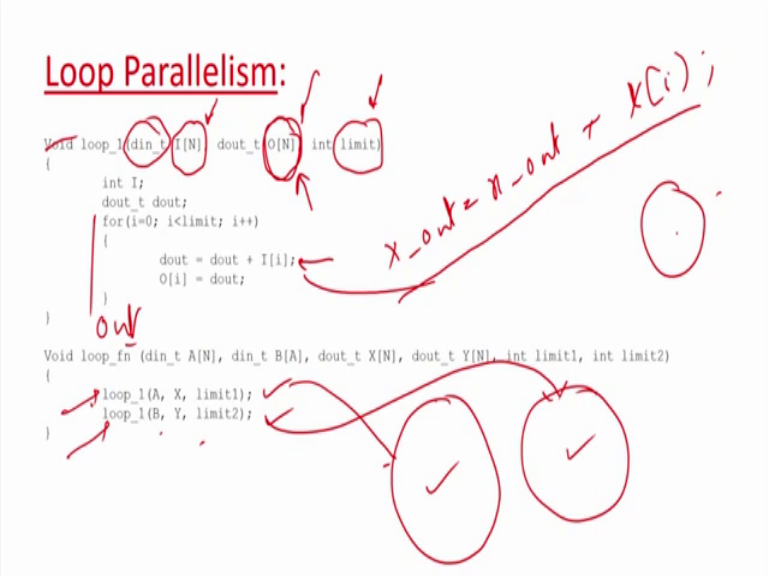



So, the case two is your limit are not same you have different limits, but your loop operations are same ok. So, then what you should do actually. So, you should write function for this loop instead of writing the whole thing in your main code; you should write a function that which generic way to do that operation right.

So, for example, here since the loop operations are mostly same here I am just doing X out equal to X out plus A i here Y out equal to Y out plus B i. So, this operation is same only the variable name has change right similarly here Y out to Y 1 and X out to X 1 only the variable name is same, but the type of operation is exactly the same.

So, what I am going to do? I can write a simpler function right which will do this body and that parameter to that particular function will be this A and this output will be Y right. So, written type will be X or Y and input will be A.

(Refer Slide Time: 34:26)




So, if you just write this; so, this is what I have written here is that I write a function where my input is say input I and I have output is O and I have a limit. And the loop I am just doing the same operation the operation type is same d out plus. So, actually it was written X; see here X out plus.

So, what I am doing here; so,; so, X d out equal to; so, actually it was written like this right X out plus X i. So, I just replace this iteration by this right because I now make it parameterize. So, I have the input is I; I have input is limit and output is this right. So, here I can see that I have always using the data types d into t data types input types. So, it is not integer; so, it can be 10 bits 5 bits two bits based on your requirement and the this is the same loop I am just calculating the same thing, but in the parameterized one right and my output will be it is not void should be written types would be this O ok.

Or you can actually passes by argument. So, both are possible this are the language specific; so, I am doing this and now I have a just two call here you can see I just call loop 1 with A X and limit 1 then I call this loop 1 again with B Y and limit 2. So, if you just do this; so, in the first call it will do the operations of this loop 1 it will do the operations of this loop 1 and in the second call I am going to I call the same function again with this different parameter and it is effectively going to do the operations of the second loop right.

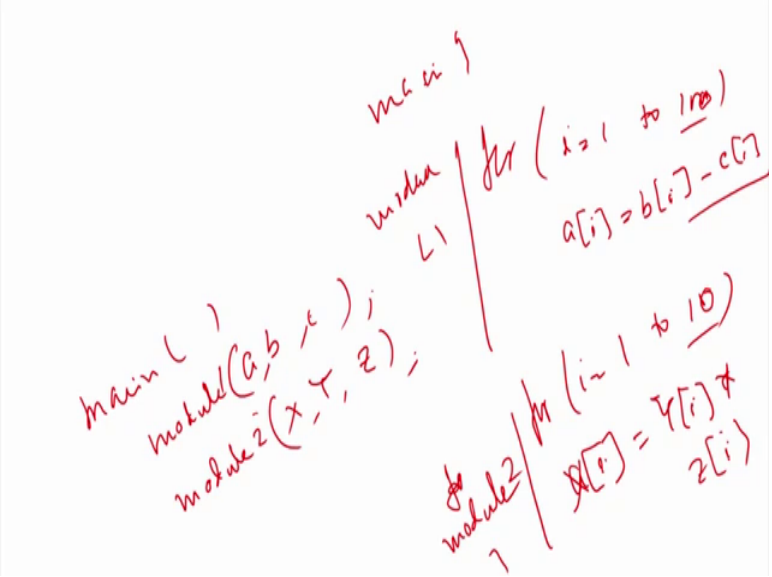
So, this is something is going to happen, but the advantage of this is big. So, now, I have declare this 2. So, I have only this one way declare this as a function; so, it will create a one function body right. So, that or the one module for this and now you can actually it will there are two options you try to run them in parallel then it will create two instance right.

You can it will create because this two are independent thing this two loop body are independent. So, you can create two module and one module will call for this and one module call this right. So, it depends on your choice; so, you can create two instance or two loop body or you can actually create one loop body only. And then you use the same loop body or the module just to do this one and then you can re use the same body to do the same operation; so, both are possible.

So, based on your requirement you can if you need only you want to minimize your resource you can probably make them sequential so, that only one variable will be created and that will be reuse for both the calls. This is case 2 where your limit is different, but the loop body is perform similar kind of operations.

But if your loop body is also different then you cannot reuse always it will create multiple body, but it is better to call I mean write them in a function so, that you can and you can call them whenever is required like this right. So, this is something at least if their loop body are different; you are creating two different instance of the loop body, but still you are actually make it parallel right.

(Refer Slide Time: 37:46)

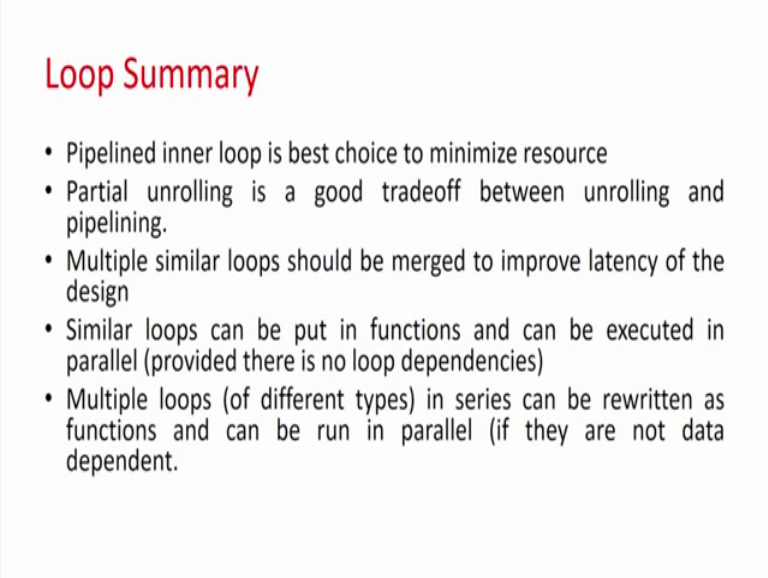



So, what I am trying to make is this is like suppose you have a loop 1. So, where you are doing say I equal to 1 to 100, you are doing say a i equal to b i minus c i right and this is loop 1 and this is in loop 2; if you are doing same thing i equal to say 1 to 10, but you are doing say x equal to Y i into Z i. So, I am doing a different operation now ok.

So, I am doing Y i into Z i; so, this is your main. So, if you have your loop body iteration level this limits are different this is 10 and this is 100 and also the operation is different. So, you cannot create a single module to do this, but still what you can do you can create a module 1 for this right, you can write a function 1 for this is function module 2; for this function the way I just write this kind of loop and you can call this two function module 1 and module 2 from main right.

So, you can just do this from main you can just call module 1; module 1 with a, b and c and module 2 with X, Y and Z right. Now since these two are not dependent they are independent they will run in parallel; so, that is the advantage. So, you always try to try to merge; so, parallelize is the loop that is because loop will taking lot of time.

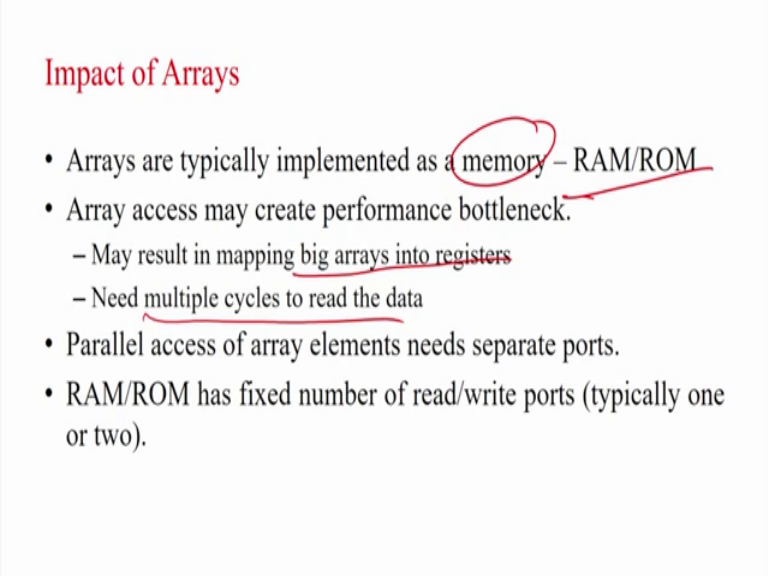
(Refer Slide Time: 39:35)




So, summary is this you try to parallelize this loop as much possible you try to merge two loop if possible or you can actually if their limits are same and the operations are same. Or if it is not limit is different then you can probably create a same function body and you call it two times.

So, that either it can be reused or may be parallel body will be created or if the limits are also different and the function body is different still you write a module; so, that this two module can run in parallel in hardware right. So, this is something you should always think about when you have multiple loops in the in your behavior ok; so, this is on loops.

(Refer Slide Time: 40:10)



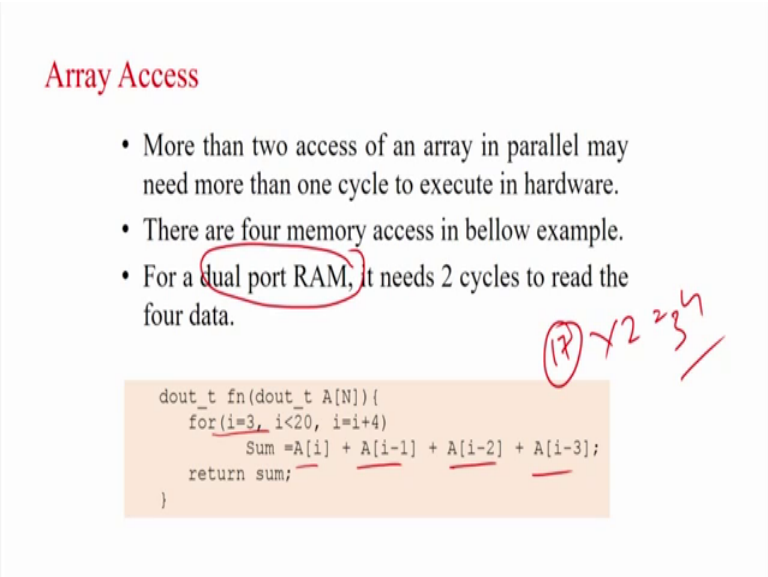

Now, I am going to talk about array ok; so, this array is another important aspect in program. So, you use lot of arrays and this is the most critical thing which actually make bottleneck to your performance of your hardware ok. So, usually this array is getting implement as memory; that means, RAM or RAM RAM or ROM. Then the problem is here is that this RAM or ROM in the hardware, they are going to access through port ok.

So; that means, and the this hardware typically in for synthesis. So, we have only single port or dual port; that means, you can maximum one a limit or maximum two a limit depends on your RAM choice, but in c you do not care you can read say 10 elements at a time from array, but if you have a say multiple access to array limit then it is a problem to infer a memory out of it right. So, either it will if because of there are multiple access to that array this will be map to a big registers or you need multiple cycle to read that data right. So, what I am trying to say is that there are say 4 read from array and this 4 read cannot be done.

So, in C you can do it there is no problem you just read for elements, but in hardware since you have maximum (Refer Time: 41:25) I assume that you map it this into say dual port RAM. So, you maximum read two elements; so, you need two cycle to read 4 four elements or if your performance is critical you want to access all the element at the same time then you cannot map that particular array to that memory RAM, you have to map that particular array to registers in register you can have parallel access.

But the problem is that for a big array it is not good choice to map them to register because that is your all the register will be utilize and then it will create a problem for other purpose you need register you may not use it right; so, your all register will be used. So, that is something we should always think about right.

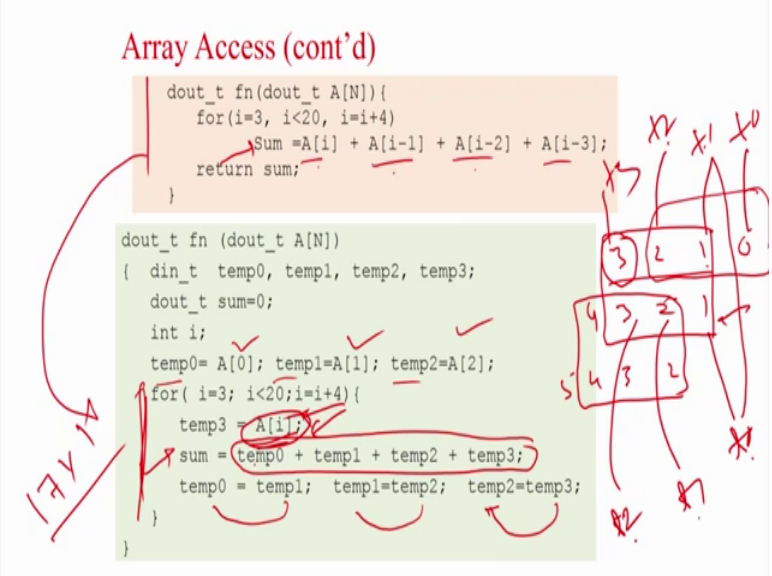
(Refer Slide Time: 42:10)




There is an example here; so, as I mentioned there is a loop here where I am just adding 4 I am reading 4 array limit and just sum them right. So, you can there are 4 access and if you try to map this array a to some RAM. And suppose that particular RAM is dual port RAM, then you need two cycle to read this 4 data right. So, every time you have to for every iteration you have to wait for two cycle to read all the data then you do the summation. And this loop iterates for 20 minus 3 means 17 times; so, 17 into two 34 cycle will be required right.

But because it you if you if you map this array A into this some RAM ok.

(Refer Slide Time: 43:00)




But you can see here that this particular array indexes are reused right if you just see here you can see that. So, in the code for the initial array; so, this is A 3 A 2. So, this is 3 2 1 0 loop percent right; so, in the first iteration I am reading A 3, A 2, A 1 and A 0 right.

In the second iteration when i equal to 4 I am reading actually 4, A 4, A 3, A 2 and A 1 right. So, 4 3 2 1 you can see this 3 elements are common I am reading them 2 times unnecessary in the for the first operations; iteration 1; I am reading them. Again I am reading them for iteration 2; similarly for iteration 3 it is 5 4 3 2 again this 4 3 2 is going to reuse every time.

So, this is an unnecessary read; so, in C there is no problem because it is just reading 4 elements from array what is the big deal, but it is a big deal for hardware; it is creating a bottleneck where you are unnecessary reading the same element multiple times which you could have avoided right.

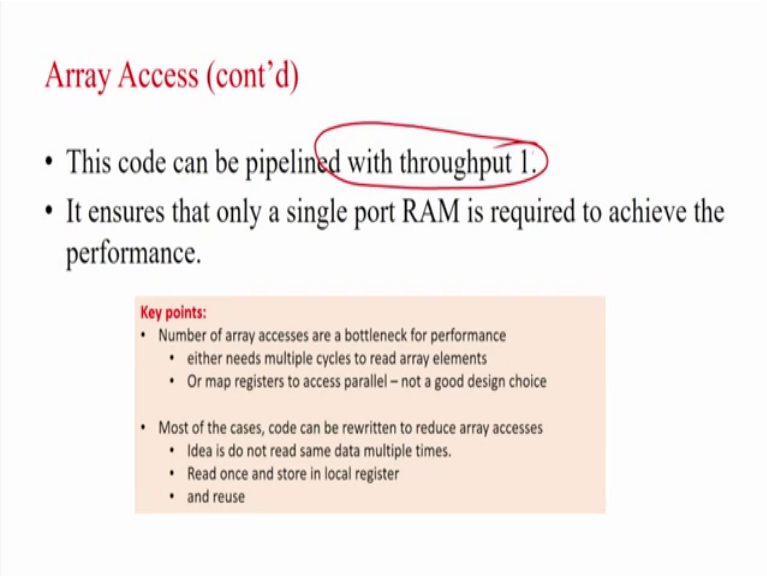
So, now you should write yours this code this way. So, that what I am doing here initially before the starting of the leap I just read this two initially I just read this A 2, A 1 and A 0; I just read A 0, A 1 and A 2 into 3 temporary variable temp 0, temp 1 and temp 2 and in the loop what I am doing? I am just reading this temp 3 sorry A 3; I am just reading A 3 and this is already available this 2 1 or 0 is available here. So, the sum is I am doing this where I am using this temp 0, temp 1 and temp 3 is already read they are stored there.

So, sum is required; so, I need only one read from memory of array A right. And then you have to do this shifting because in initially for the next iteration this temp 0; this is your temp 0, this is temp 1 and this is temp 2. Now for the next iteration this will become temp 1. So, this is your temp sorry temp 0; so, this is temp 1, this is temp 2.

Now, this become temp 1 this become temp 2 and this become temp 3 right. So, you have to just shift the things you have to reassign the value. So, this is what I am doing this temp 1 become temp 0, temp 2 become temp 1. So, see temp 1 become temp 0 sorry this is I make a mistake here. So, this is temp 0 this is temp 1 this is temp 2 right.

So, now temp 1 become temp 0, temp 2 become temp 1 and that a temp 3 become temp 2 right; this is your temp 3. So, this rewriting is happening; so, that for the next iteration I have the right value of the temp 1 temp 2 and temp 0. So, this is how if you just rewrite this code you can actually execute the whole loop with only one read per cycle per loop. And you can understand this where actually I just what I did? I just reduce this unnecessary memory access.

(Refer Slide Time: 46:08)




And if you do this you can actually do this using pipelined with throughput one; that means, you can actually read you can map this particular array to a single port RAM and you can actually execute this loop under 17 times.

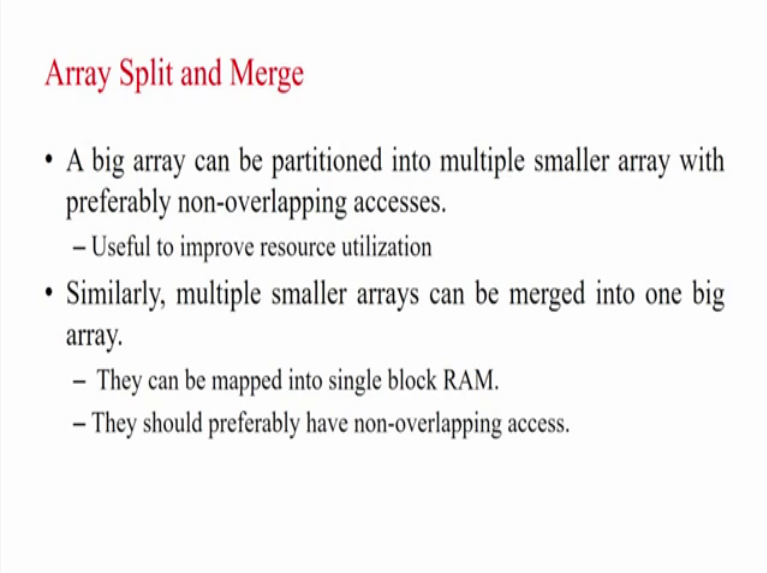
Because every iteration I have only one read. So, I need one cycle; so, instead of 34 cycles here I can do the loop with now 17 cycle right. So, this is the some very simple example that actually tell you the logic behind inferring RAM, ROM from array right. So, the key points here is this; so, the number of array access is the big bottleneck in performance right.

So, the if you a have multiple access either you need multiple cycle to read them in your hardware or you have to map that particular array to register to make all the read parallel; that means, you want to [FL] you can read all the elements in one cycle or. So, the which is not a good design choice right.

So, most of the cases you have a option to rewrite your code. So, that you can reduce the multiple array access and you stored the read you read the data you store in once and store in local register and reuse that is the basic idea. So, instead of accessing multiple times; you access each element one times, you store it locally and then you reuse whenever is required whatever I have done here right. So, I do not read this element this A 3, A 2 and A 1 multiple times instead what I am doing? I am just try to reuse I read it here for the iteration 1, I use them in iteration 2, I read this 3 in for iteration 1 I am going to use them for iteration 3 and so, on.

So, this is the core idea behind array access. So, this is a very critical thing you have to always try to make sure that your array is not over access. So, that inferring RAM or ROM from array is getting hampered because of your access multiple access. So, you should always think about writing this kind of code instead of this kind of code ok.

(Refer Slide Time: 48:11)




So, this is something we have discussed there is another option with your array split and merge. So, if you have a very big array you can probably split it into multiple all right.

(Refer Slide Time: 48:19)

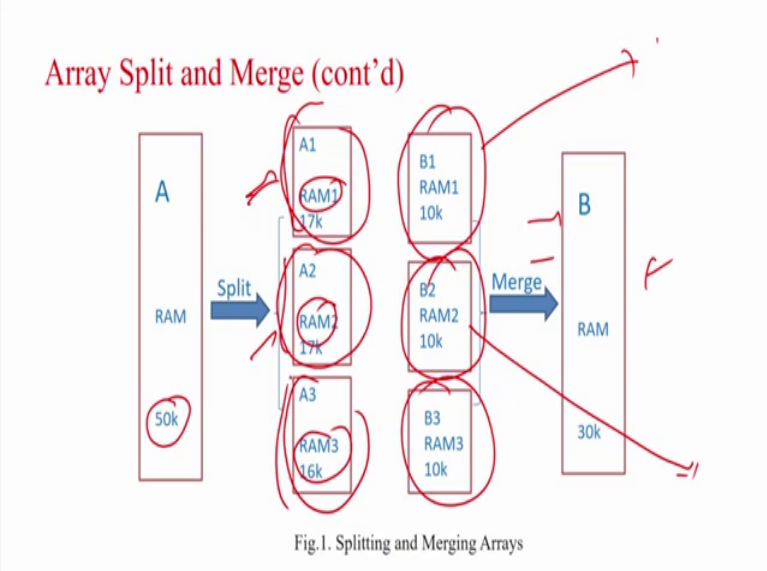


Fig.1. Splitting and Merging Arrays

So suppose you have a very big array if you try to the problem is that if you have a big array there is a high chance that I mean there is a multiple access and its not getting inferred RAM right.

So, it will map into register which is a disaster right which is a 50 k size of array. So, its better to split into small array; so, that you can actually distribute your access in local to

this. So, all the local access is map to this all the accesses that is map; so, that can be map to this. So, that you can partition it properly and you can infer RAM out of this block, this block and this block where if you just try to map the big array probably this cannot be map to a single RAM because of multiple access.

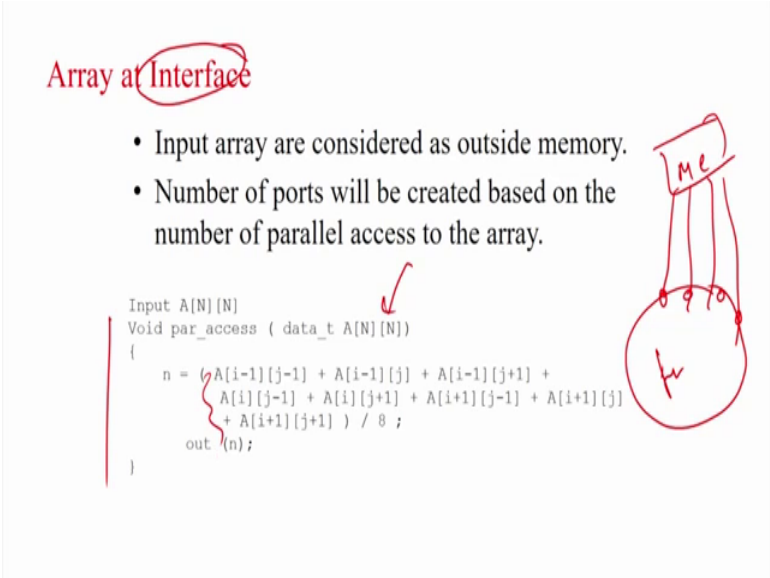
So, if you split it according to their access; so, that non overlapping access is club in together. So, then you can infer probably multiple access again this is very design specific it is not always possible, but you always try to keep in mind that very big array there are multiple access and it might create a problem in accessing RAM rom. So, you can split that array into multiple small block where all this where the access to this are non overlapping; non overlapping in the sense they are not access in parallel ok so that you can actually map them into different block.

Similarly, the other way if you have very small array inferring RAM for them is waste of resource right; because your array this your RAM size is say 18 k, 20 k in hardware and suppose your array only 1 k or 2 k. So, that time if your if you did not violate the performance in the sense that this is say access very early in the loop and this is access very late of the of a main code; then probably they are access on non overlapping right probably even if you map them.

So, this is their schedule in different time step their access can be can be done through same port. So, then you can probably its better idea to merge this small array into a bigger one and use that array in your code. So, that instead of inferring 3 RAMs here you can infer 1 RAM still you can actually execute all the read and write operations there through the single port or dual port of the big RAM.

Again you understand this is also very design specific its depend on your code; sometime this is not possible, sometime this is possible, but always you should always think about this aspect of writing code that should I merge most probably (Refer Time: 50:41) big one I am going to use it or I should be array I want to split that big array into small one. So, that I can infer the inferring RAM is I mean easy right or feasible ok. So, this the another aspect you should think about.

(Refer Slide Time: 50:53)




The second thing is that the third thing is that that array has interface whenever you write a function and it has array input. This function is create a module right and it will assume that there is a since there is a array input it is something external memory. And this is your function and it will create some port here right; so, that memory access happen through this port ok. So, this is how this will be mapped, but if you have very large number of access right.

(Refer Slide Time: 51:26)

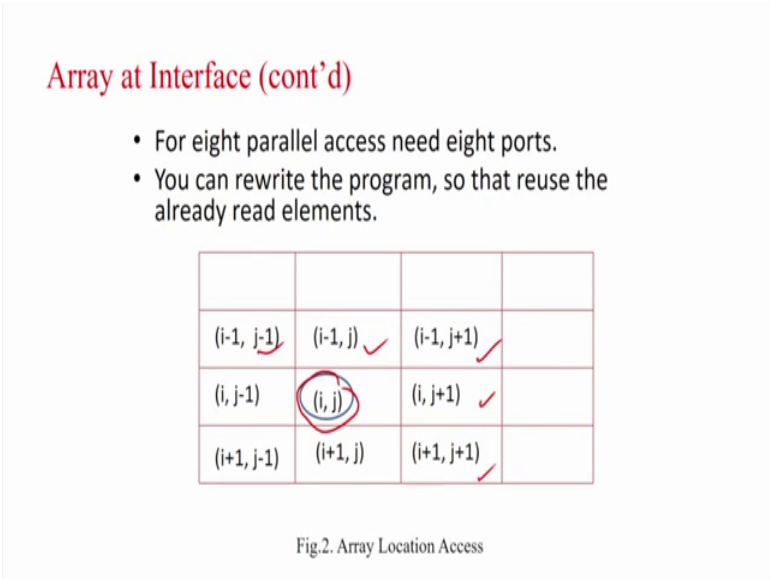



Fig.2. Array Location Access

So, I have here 8 accesses this is nothing, but it would take this ideal location I am going to read all this never this 8 location; this is what I have written here 8 location and I just make average out of it right. So, I am just doing something and the problem is here there are 8 access, it will create 8 port right the problem again is that this number of port is something you should always keep minimum because in your hardware; the number of port is always fixed. And you cannot overuse this as problem then may be your code may not be map to some specific target ok.

So, idea is again is that you when you have this kind of function array has the input, you try to reuse the array access.

(Refer Slide Time: 52:11)

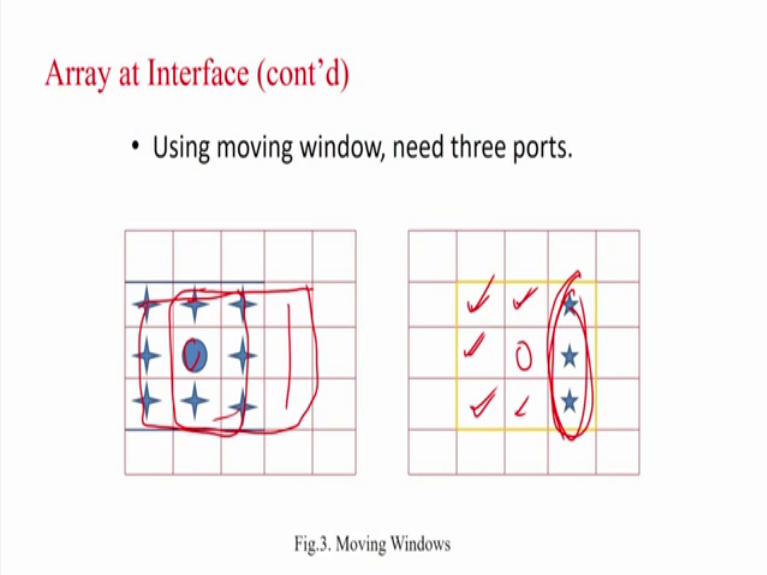


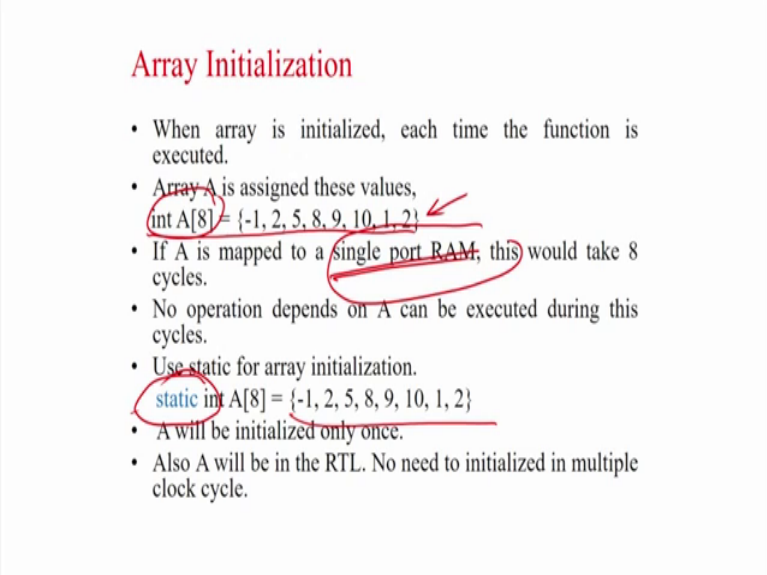
Fig.3. Moving Windows

Again the same example is happening here like my previous example you can see here for this location I am going to use this all 8 location right. So, you can see here; so, this is my. So, for the second iteration you are going to access this; this iteration these are common right this 5 location are common only this 3 is the new one right. So, this location this location, this location, this location is already read.

So, you should rewrite your code this code such a way that you can actually reuse this 5 location; this 5 data you can reuse when you are going to do the average for this node right. So, that you do not have to create 8 port instead of 3 port is sufficient because every iteration you are going to access 3 new data. So, instead of 8 you can only read 3; so, that is something you should always be careful. So, the summary of this is like when

you have array at interface you should always try to minimize the access. So, that it can create less number of port, ok.

(Refer Slide Time: 53:06)




So, I will move to the next topic the array initialization. So, you always initialize some array in our design right. So, you just define A 8 equal to this and if you do it inside a function or say in your main every time you call that function is getting initialized. The problem is there since suppose this is map to a single port RAM what will happen? You need 8 cycle because you are writing 8 data to the RAM.

So, it will it will take 8 cycle to read I mean initialize this array and it is a bottleneck right if you call 100 time this function; it will take 800 times 800 times or cycles to only the initialization of that array; because this is something its suppose this is map it to single port RAM.

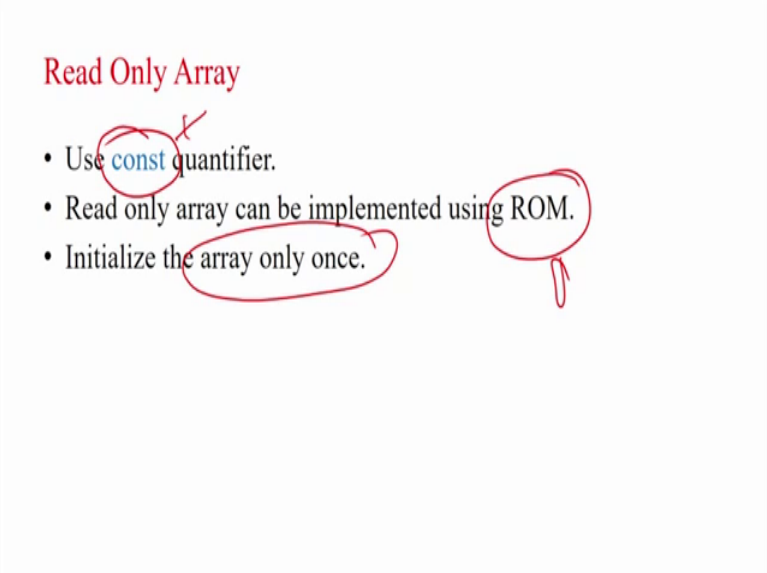
But what you can use suppose your initialization purpose is only once and you are not going to reuse that initial data more than after the first call, you can probably use static right. If you use static then it will be initialized only once and then you are going to reuse. So, that is the basic idea that is the same concept of c. So, instead of writing only it is 8s this initialization you can actually do static.

If you do static then it will be initialize only once in hardware and it will be use. So, it will reduce the number of size not each call it will be initialized. So, it will save lot of

time if your design is not going to use depend on your initial data the initial data. Most of the time what happened that initial data required for only the first iteration for the first call after that whenever you call the whatever you computed that is used.

So, it is better to use static instead of normal initialization because that is the simple thing, but that will save lot of latency of your design ok; so, this is array initialization another issue.

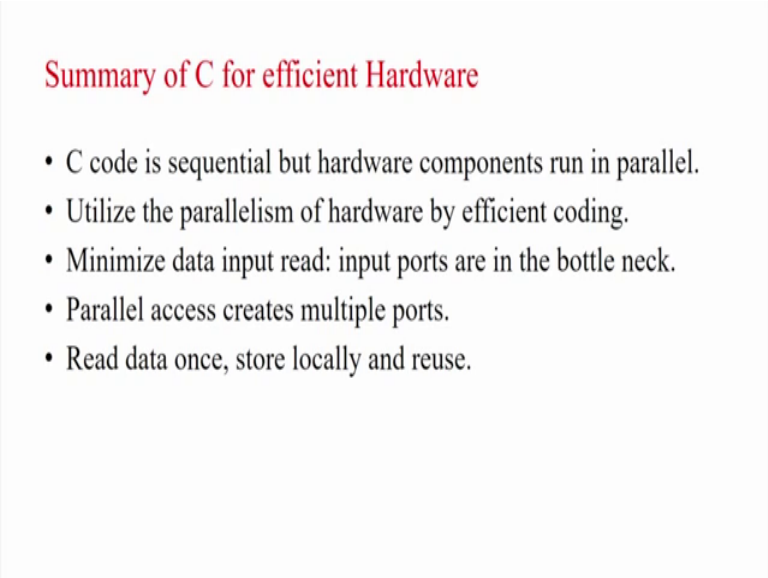
(Refer Slide Time: 54:44)




So, the last topic is like this read only array. So, if you have some array in your design these are going to read only I mean only read, there is no write on that particular array. Then it is better to infer a ROM from that right RAM is something where you can read and write. So, that can be use for the array that is something is going to both read and write.

But some array is only going to be read then it is better to infer a RAM and for that you should use const quantifier if you use a const quantifier there is a constant array. So, the synthesis tool will understand there a constant array this is not going to be updated; it will be initialized only once. So, this will be it will automatically infer ROM out of it and it is going to be; so, it is not going to map into RAM. So, this is also use useful features that you should always aware of using const quantifier for array which is only read only helps to infer ROM out of it ok.

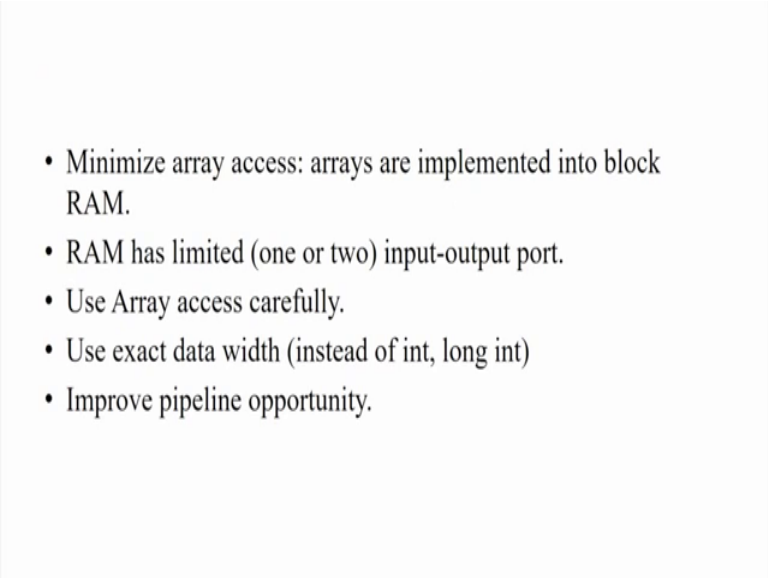
(Refer Slide Time: 55:46)




So summary: so this is what I all I have today is like summary is that you the first thing you should always keep in remember that your C code is sequential right, your code is sequential, but hardware is runs in parallel. So, you try to write your code so, that you can utilize the features of this parallelism of hardware.

So, you can always think of writing C keeping in mind that I am going to have a hardware which is parallel. So, that is a very basic thing we always missed; so, that is a key point that you always keep remember. And the second thing is that your data type should be very efficient. Efficient in the sense will be exact it should not overuse or over I mean overuse the actual data types, you should always give specify the exact data types because that will determine the data path width; so, which is save lot of area; so, that is another point.

(Refer Slide Time: 56:45)




The third point is that that regarding this array you should always minimize the array access because that will create problem or bottleneck for implementing this array into memory like RAM and ROM. So, you should always idea is that you always access carefully you try to reuse the array access you should not read the same element multiple times during this you try to save them locally. And then you try to use them right and for loop you try to use pipeline as much as possible, because if you do not pipeline if you unroll everything it will create huge hardware which is not efficient and this is not sometime feasible at all right.

So, you should always try to create opportunity so, that your loop can be pipelined because that will give you the minimum number of resource usage and kind of acceptable speed ok. So, this something the key points that you should always be aware of when you are writing C code or say high level code for your target high level synthesis tool.

Thank you.