C-Based VLSI Design

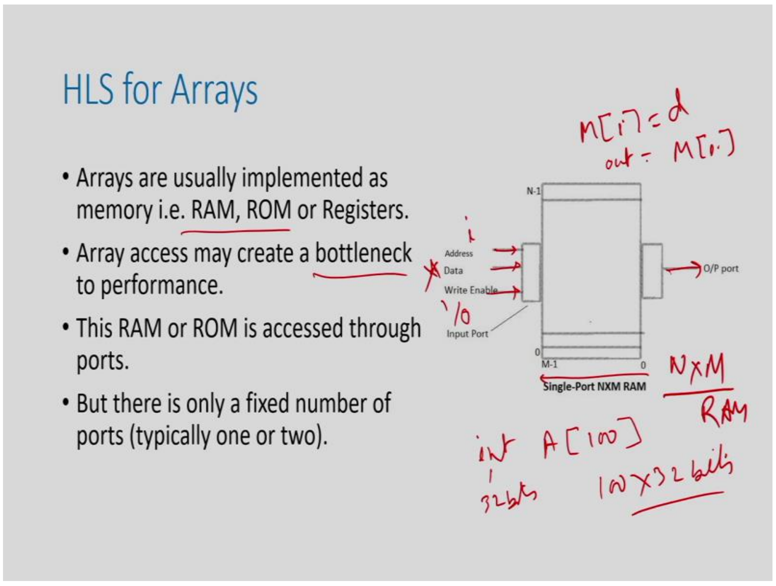
Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 07
Efficient Synthesis of C Code
Lecture - 23
High-Level Synthesis of Arrays

Welcome everyone to this course. In today's class, we are going to discuss of High-Level Synthesis of Arrays. So, as we know this high-level synthesis is something which convert an high-level behavior written in C or C plus plus into register transfer level design.

So, we should understand how the important construct of a high-level behavior like arrays get C mapped into hardware and what kind of bottle neck it might create during the efficient hardware generation ok. So, that would be the discussion topic today.

(Refer Slide Time: 01:17)



HLS for Arrays

- Arrays are usually implemented as memory i.e. RAM, ROM or Registers.
- Array access may create a bottleneck to performance.
- This RAM or ROM is accessed through ports.
- But there is only a fixed number of ports (typically one or two).

M[i] = d
out = M[i]
N-1
i
Address
Data
Write Enable
1/0
Input Port
O/P port
0
M-1
0
Single-Port NXM RAM
N×M
RAM
int A[100]
32bits
100×32 bits

So, as you know this array is a contiguous memory location, right. So, if you write a 100 integer a 100 so, it is basically 100 of integer in the memory, right. Now, if you think about how this array will going to map to hardware, so, obviously, you can map this array into registers. So, you need in that time 100. So, if I just write int A 100 and if I assuming int is 32 bits basically it is a 100 into 32 bits. So, this many register bits will be occupied, right.

And, since in a PGA kind of target the number of registers are limited or a fixed, so, mapping this array into register may not be good idea right because the then what is the solution? In the PGA board we have also construct like RAM and ROM, right. So, which is something RAM is used where you want to read and write both and ROM is only for Read Only Memory, right. So, if you want to have some data which you want to only read then we can actually store that things into ROM.

And, ROM and RAM is also the similar kind of thing is a contiguous memory location. So, it is a very one to one correspondence with the array right because the array is also is a continuous data and it is a continuous memory. We just map AI will be mapped to MI, right, as simple as that. So, that is the best target. So, we understand that mapping this arrays into RAM or ROM based on their access is the best way to do that, ok.

So, now, we will see what is the bottleneck, right, what is the problem here. So, now, the problem is in the RAM or ROM we have a fixed number of ports, right. So, the port is something read or write port; read port, write port or both read write port, but we are going to decide about discuss about the both read write port, ok. So, for a block RAM say I have an taken an example of say size N and the width is M. So, it is N into M size RAM and it has a single port, right.

So, each port has a multiple inputs. For each port we have a address line, we have a data line, we have a write enable line, these are the inputs ports and we have a output line course. So, this all four thing combined to a single port, right. So, these are the component of a port, ok and how read and write is going to happen into the memory? So, I am going to specify address i and say some data d and I will just put write enable equal to 1, right.

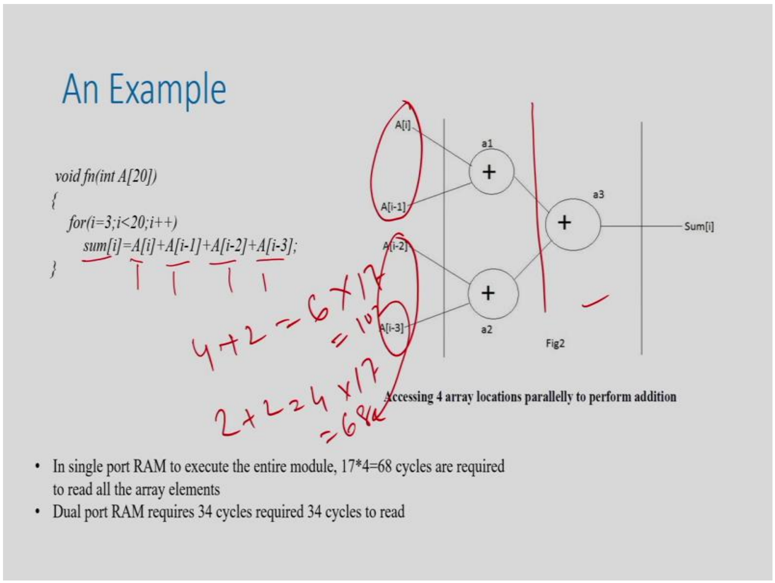
So, if I just put write enable equal to 1, then what will happen? So, in this particular RAM say this is memory and the ith location, data d will be stored, right and if I put 0 then it is basically read, right. So, then what will happen? Whatever the output so, out will be M of i that is and this data is relevant in that case.

So, this is how the RAM is get access in the hardware that it has ports, and you specify whether you want to read or write and you specify the address. If you want to read you just make the write enable equal to 0. So, that your the content of that address will be come into the output and if you want to write something there you make the write enable equal to 1 and you specify the data you want to write you give the address and in that particular location that corresponding data will be written, ok. So, this is the overall idea.

So, now, the main bottleneck comes here because this particular RAM has fixed number of ports ok, usually it is a single port or dual port ok RAM. So, that means, if it is a dual port; that means, at a time you can only write or read at most 2 times into the design because there are two port either you do two writes, two reads, one read, one write whatever it is, but you can (Refer Time: 05:00) to two operations on the RAM, right.

And, it is a single port; you can only have a one access to the RAM and in a clock ok and that is create a problem because once you write a C code we never bother about how many array location is going to use into compute an one operation, right, take an classic example.

(Refer Slide Time: 05:20)



An Example

```
void fn(int A[20])
{
  for(i=3;i<20;i++)
    sum[i]=A[i]+A[i-1]+A[i-2]+A[i-3];
}
```

A[i]
A[i-1]
A[i-2]
A[i-3]
a1
a2
a3
Sum[i]
Fig2
4+2 = 6×17
= 102
2+2 = 4×17
= 68

Accessing 4 array locations parallelly to perform addition

- In single port RAM to execute the entire module, 17*4=68 cycles are required to read all the array elements
- Dual port RAM requires 34 cycles required 34 cycles to read

So, here I have written a dummy code function which is actually doing a summation for sum i which is basically A i plus A i minus 1 plus A i minus 2 and A i minus 3. So, there are four array is going to add to sum i and the i is going from 3 to 20, right. So, this is what is happening. So, there are four array location is getting accessed, right.

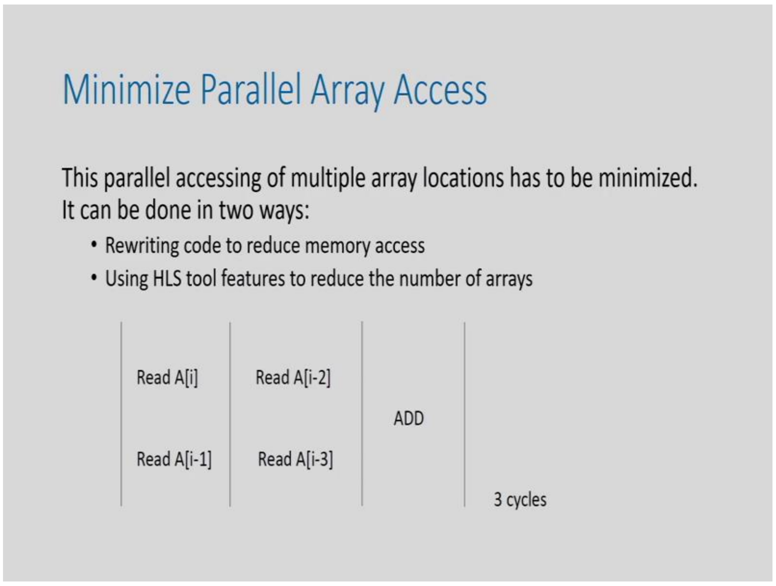
So, if I now think about I want to map this particular array A to A memory it was say block RAM M, then what is going to happen? There are four access to do one operations, right. So, in this once we write the C code we never think about that we just assume that this particular operation is going to happen in one clock because this four data is getting accessed. But, it is not the case in hardware.

So, now, think about a single port RAM. So, how many clocks you need to read this four data? You need four cycle, right and then if the operations are like this it is showing here there are two operations. So, if you assume that there is no operation chaining so, you need two clock to execute, right. So, 2 4 plus 2 so, that means, 6 clocks you need just to do one iteration and that is this loop is happening for say 17 iterations, right.

So, it is basically this number of iteration is going to happen to complete this loop which is something 102 right. So, 102 cycles are needed just to execute this, but it is a dual port. So, you can actually access two data. So, I will access this two in clock 1, this go into clock 2. So, I need 2 plus 2 which is 4 into 17, right which is something 68 cycles.

So, you can understand just having the port number of ports you can see the difference in the performance it will be much slower design, right. So, now, obviously, this is something is a bottleneck, right. So, what is the way out? So, once we write a memory I mean C code with array we can clearly understand that we need to minimize the parallel access, right.

(Refer Slide Time: 07:13)



Minimize Parallel Array Access

This parallel accessing of multiple array locations has to be minimized.
It can be done in two ways:

- Rewriting code to reduce memory access
- Using HLS tool features to reduce the number of arrays

Read A[i]
Read A[i-1]
Read A[i-2]
Read A[i-3]
ADD
3 cycles

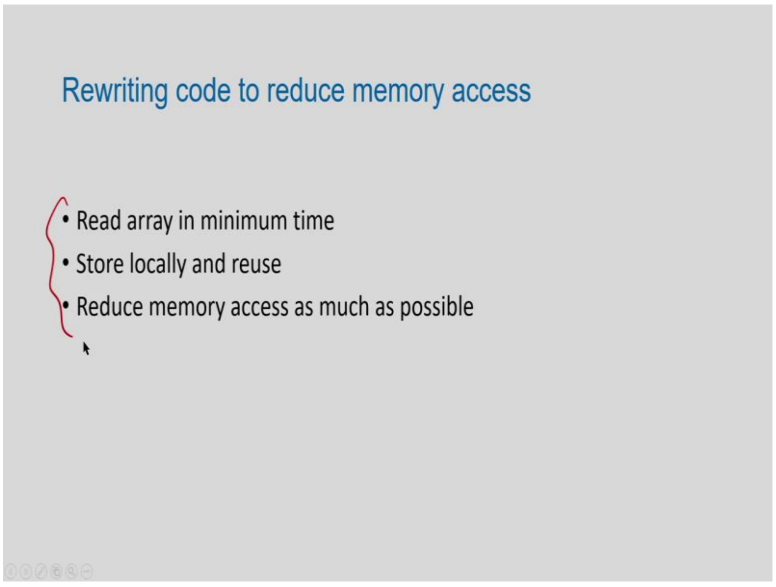
So, we somehow we have to write the code such a way that the array access is as minimize as possible, right. So, obviously, there are certain data you have to read you cannot just stop reading the array because you read the data right, but we can clearly understand this access unnecessary access we will create a bottleneck for performance in the hardware.

And, hence we need to minimize this parallel access to the array we will come into more detail and where how can you do that? Obviously, you can rewrite your code I will give an example on the previous code in the next slide, that how can you rewrite your code just to minimize the number of memory access or you use the high-level synthesis tool features to do the same, right. So, in this particular lecture we are going to discuss both.

So, given N exam program how I would just analyzing the array access can I reduce the access that is something we will see and the is the same thing can be done using the high-level synthesis tool features. So, both things we are going to see here and the whole class I am going to do in a example based, right. So, because in a big example, if I take a very big case study, it is very difficult to convince.

So, we will take a unit test case and we will take a very unique scenario and we will try to see how doing some kind of array manipulation helps in performance, ok. That is how I am going to continue this discussion.

(Refer Slide Time: 08:41)



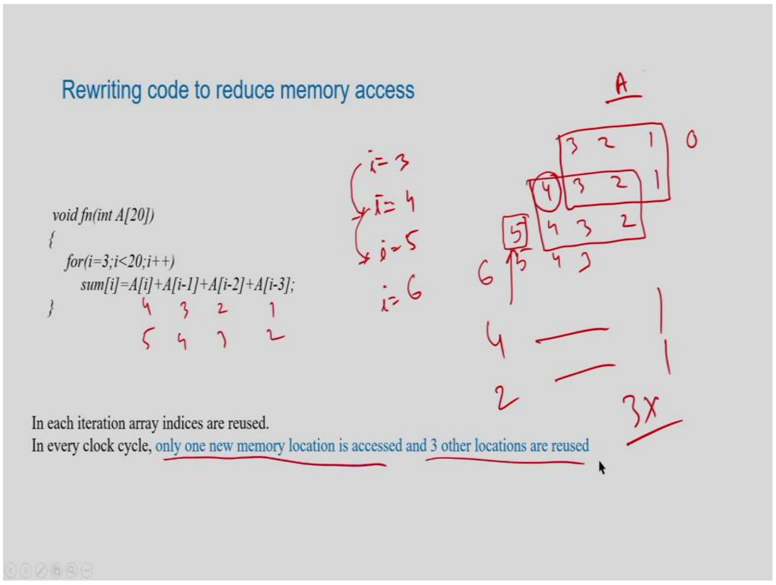
Rewriting code to reduce memory access

- Read array in minimum time
- Store locally and reuse
- Reduce memory access as much as possible

So, the key idea is this. So, many time we will notice that this array access is something redundant. We unnecessary access the same data multiple times instead of accessing the same data multiple times let us read it once and store in the hardware. What is the point of reading the same data multiple times in the array or the memory because it is a bottleneck performance bottleneck let us read it once and store it locally and reuse, right.

So, the primary the key point here, the whole discussion is you read the array minimum time, you store it locally and then you revise reuse, right. So, that will reduce the memory access as much as possible. So, that is the overall key objective, right.

(Refer Slide Time: 09:23)


Rewriting code to reduce memory access

```
void fn(int A[20])
{
   for(i=3;i<20;i++)
      sum[i]=A[i]+A[i-1]+A[i-2]+A[i-3];
}
```

In each iteration array indices are reused.
In every clock cycle, only one new memory location is accessed and 3 other locations are reused

So, now with this objective let us look into this code and let us see how can we rewrite this code just to reduce the memory access and hence improve the performance, ok. So, if you just see when your i equal to 3 what is happening I am going to access A3 A2 A1 and A 0 I am just writing the index only, ok. So, this is what I am accessing right. So, this is one i equal to 3.

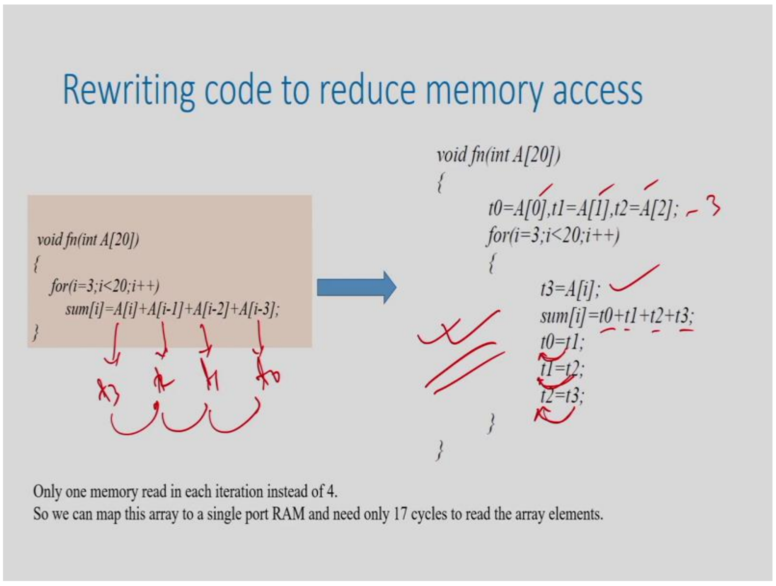
What about the i equal to 4? When i equal to 4, what I am writing? So, since this is 4, this is 3, this is 2, this is 1. So, what I am going to access? I am going to access 4 3 2 1; when i equal to 5, so, this is basically A 5, this is A 4, this is A 3, this is A 2. So, what I am actually accessing? 5 4 3 2. I hope you understand the pattern that I have try to figure it out here. So, in when i equal to 6 I am accessing the A 6, A 5, A 4 and A 3.

So, what is happening here, whenever I transit from the one iteration the next iteration only the value that I need is A 4 new value and the rest of the things is common which is already available that I have already read for my previous purposes, right. Once I move from i 4 to i 5 then next iteration then only data that is not available which is A 5, right and rest of the data is already available A 4 to A3 2 is already available.

So, no point reading A 4, A3 and A2 again when i equal to 5 rather you use the data is already written in a previous iterations and then we just only read this A 5 in every iteration. So, with this idea we can actually understand that we can only read only one data for iteration. So, earlier it was four data, now it is only one data. So, obviously, the cycle if it is a single port so, earlier it was a 4 cycle now it is a 1 cycle.

So, it is a 3x improvement in only the reading part, right. So, it was basically so, I do even if it is a dual port RAM. So, earlier I need 2 cycle again I am need only 1 cycle because I am reading only one data for this. So, this is something the basic idea that you only read one memory location in every iteration and use the other 3 location which is already read and you reuse them, ok.

(Refer Slide Time: 11:45)



Rewriting code to reduce memory access

```
void fn(int A[20])
{
  for(i=3;i<20;i++)
    sum[i]=A[i]+A[i-1]+A[i-2]+A[i-3];
}
```

```
void fn(int A[20])
{
    t0=A[0],t1=A[1],t2=A[2]; - 3
    for(i=3;i<20;i++)
    {
        t3=A[i];
        sum[i]=t0+t1+t2+t3;
        t0=t1;
        t1=t2;
        t2=t3;
    }
}
```

Only one memory read in each iteration instead of 4.
So we can map this array to a single port RAM and need only 17 cycles to read the array elements.

So, with this idea I can rewrite this code into this, right. So, which is very simple that. So, I assume that A 0, A1 and A2 because in the first iteration I need this A 0, A 1, A 2. So, that is I will read before start of the loop, right. So, at this we need 3 cycle, right. So, I will just store them into t 0, t 1 and t 2 and in the loop what I am going to do? I am only going to read A i because that is the only new thing I need A 3, right.

So, and then I am going to add this t 0, t 1, t 2, t 3 this is nothing but A 3, A 2, A1 and A 0, right and then what I am going to do what I am going to do is I am going to move this because there are three temporary variable so, initially it was to hold 2 1 and 0 now it has to hold 3 2 1 1. So, what I am going to do? I am going to move this one to this location 2 to this location and 3 to this location, right. So, this is what I am doing.

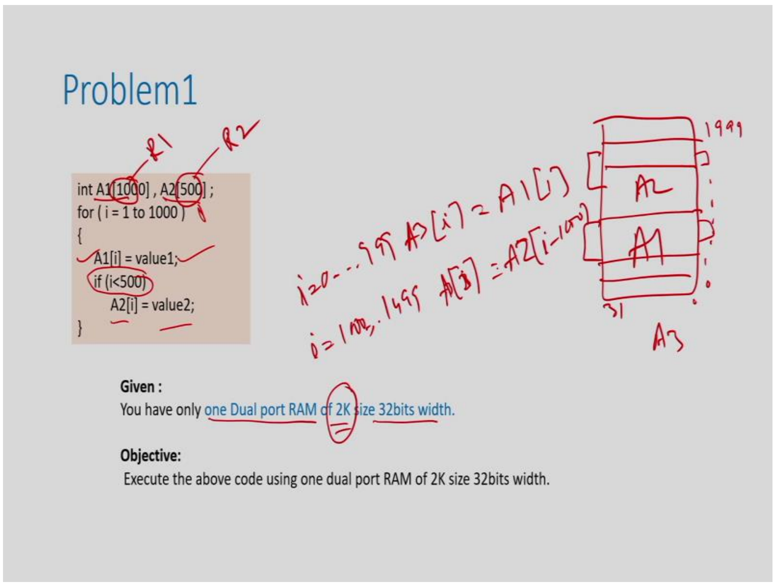
So, I am moving this t 0 to t 1, t 2 t 1 to t 2 and t 2 to t 3, right. So, that means, so, that it is in other way. So, t 1 to t 0, t 2 to t 1 and t 3 to t 2, right. So, not that means, every iteration I have t 0, t 1 and t 2 which is actually holding the value of. So, this is my t 2, this is my t 1 and this is my t 0.

So, every iteration t 0 is already holding the value A i minus 3, t 1 is holding the value A i minus 2 and t 2 is actually holding the value A i minus 1. I am only reading this into t 3, right and then I am just moving the whole thing by 1; as a shifting the things so that I am going to t 3 can read the new data right.

So, this is how if I do I can actually I just show that from 4 to 1 this is the 3x performance improvement I am going to get for this example, ok. So, the key I key (Refer Time: 13:33) of this or discussion is that some time just looking into the array access you see whether is there any duplicate reading is happening or the same data which is already read I am going to read it again or not multiple reading. So, the multiple reading can be avoided, right.

So, the multiple reading of the same data will be avoided and what we can do is we just store them locally and reuse them and which is actually both the code is doing the same thing. So, people will say what is the difference they both they worrying the same thing, but this will this much much better in terms of the hardware because this is actually where the number of array access is less. So, the overall performance will be greatly improved if I just write this version of the code, ok.

(Refer Slide Time: 14:13)



Problem1

```
int A1[1000] , A2[500] ;
for ( i = 1 to 1000 )
{
   A1[i] = value1;
   if (i<500)
      A2[i] = value2;
}
```

Given :
You have only one Dual port RAM of 2K size 32bits width.

Objective:
Execute the above code using one dual port RAM of 2K size 32bits width.

R1
R2
i=0...999 A3[i] = A1[i]
i=1000..1999 A3[i] = A2[i-1000]
1999
A2
A1
31
A3

Now, I am introduce another problem, ok. So, this is one we where we actually reducing the array reuse. Now, I am going to talk about the mapping how this array gets map and where what is the problem it might happen and how can I actually make this mapping better, ok. So, I am going to take three such problems I am going to introduce a new concept, ok.

So, the first example that I have taken here is I have taken a very hypothetical C code let us say it has 2 array A1 and A2 which is an integer array which is of 1000 locations, it has 500 locations, ok and then in a loop what I am doing is I am just updating A1 with a value 1 A i and A2 with a value 2 because it has 5 locations I have a condition, right.

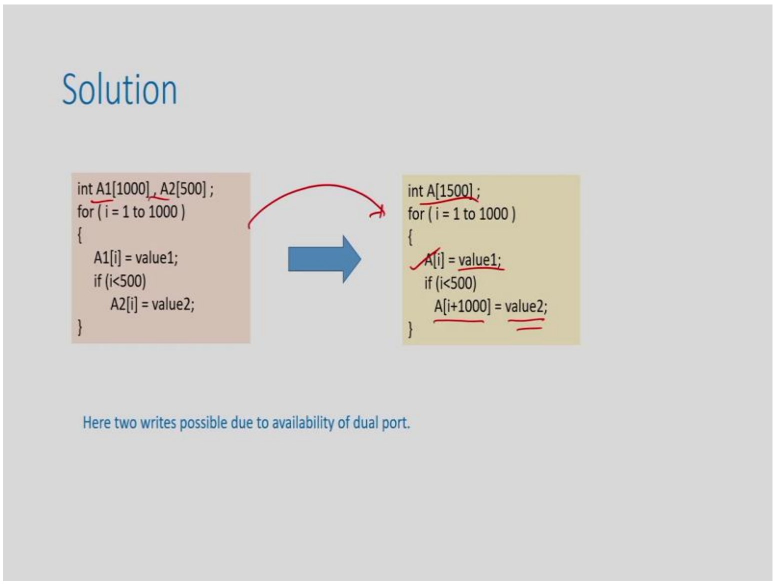
So, basically in this loop I am going to write the A1 all the locations of A1 and all the locations of A2 ok. So, and I am assuming the integer is 32 bits throughout the discussion, ok. So, now, say during the mapping it turns out to be that we have only one dual port RAM is available, ok.

So, that means, I have a RAM which has two ports I can read 2 data or read or write 2 datas from that RAM which is of size 2K right 2000 locations and each of them is 32 bit width right. So, I have a RAM which is one data is 32 bit width. So, this is 0, this is 31 and it has location 0, 1, to 1999, right. So, there are 2000 locations and it has two ports right. So, I can read 2 data ok.

So, with this particular RAM I want to execute this particular behavior because I have two arrays in general I need 2 RAMs, right. This will be map to RAM 1, this is map to RAM 2, but I do not have 2 RAM available with me. I have only one RAM available with me right. So, how can I execute this particular program using a single RAM, right that is the question that I ask you.

So, I will request you just think of, pause the video for a minute and then you move to the solution next slide.

(Refer Slide Time: 16:19)



Solution

```
int A1[1000] , A2[500] ;
for ( i = 1 to 1000 )
{
   A1[i] = value1;
   if (i<500)
      A2[i] = value2;
}
```

```
int A[1500] ;
for ( i = 1 to 1000 )
{
   A[i] = value1;
   if (i<500)
      A[i+1000] = value2;
}
```

Here two writes possible due to availability of dual port.

So, the solution here is that you have 2 array, but what you understood here that this array of size is 1000 and this is 500 and my RAM is much bigger this is 2000 locations. So, if I map this array 1 into RAM 1, it has 1000 unused locations, right. Once you map that it is unused and if I map this array 2 into RAM 2 it has 1500 location is unused.

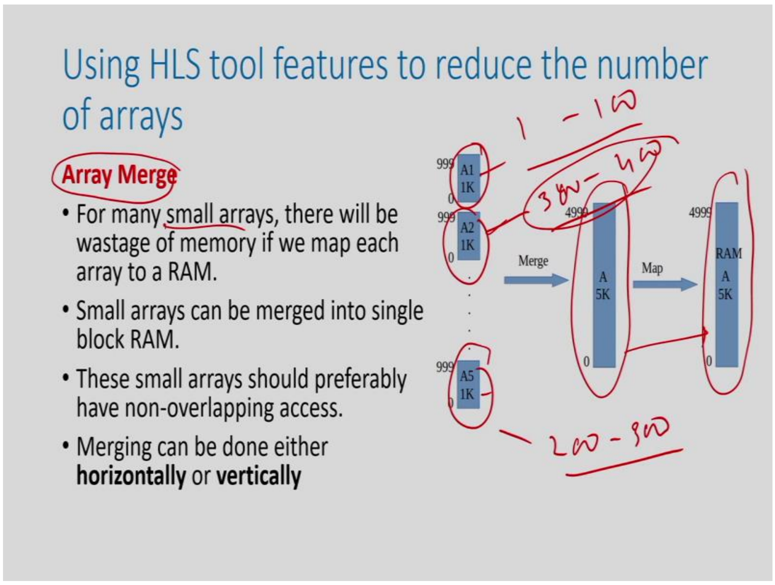
What about can we map both of them into a single RAM right? So, basically the idea is that map both A1 and A2 into single RAM. So, if I do it like this say that first 1000 location I just put A1 and then next 500 location I put A2 and so whenever. So, basically A2 and this is my say A3 array, so, A3 i is equal to array i A1 i, for i 0 to 999, right.

And, for i 1000 to 1499 A3 is nothing but sorry A i A3 i is nothing, but A2 i minus 1000, right. So, basically that I am actually mapping this A2 to the 1000 to 1499 those indices, right. So, if I can do that then actually I am able to use the same RAM, right? So, which will actually solve my purpose, right, because it has lot of locations. So, I can just map them into one array, right. So, this is the solution.

So, what we can do? I can rewrite this code like this instead of two arrays I am going to use the same array which is I just given examine in the previous and for each equation I am going to write A i equal to value 1 and since this array 2 is mapped to from the 1000 to 1500. So, for i I am going to add i plus 1000 or I am going to showed the value 2, right.

So, this way I can actually able to execute this particular program into using a single block RAM, right and you can actually do these things manually because you understand that you from the hardware specification that you have a very big RAMs no point having 2 array which I can do using single array and which will help me to map the whole thing into a single block RAM, ok.

(Refer Slide Time: 18:41)



Using HLS tool features to reduce the number of arrays

Array Merge

- For many small arrays, there will be wastage of memory if we map each array to a RAM.
- Small arrays can be merged into single block RAM.
- These small arrays should preferably have non-overlapping access.
- Merging can be done either **horizontally** or **vertically**

1 - 100
300 - 400
999
A1 1K
0
999
A2 1K
0
999
A5 1K
0
Merge
4999
A 5K
0
Map
4999
RAM A 5K
0
200 - 300

So, this is what is called the array merge, right. So, the idea is that whenever there are many small arrays if I just map this particular small arrays into individual block RAM, most of the locations of the block RAM will be unused. So, rather you merge this into a single array and then this single array can be mapped to a single block RAM. So, which will reduce the number of block RAM uses.

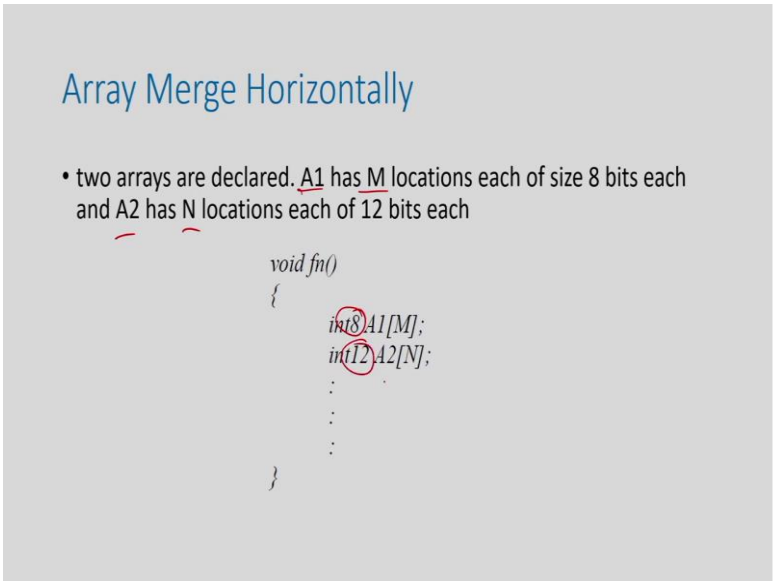
So, there is one bottleneck here obviously, is that if you just map all the small array into a single block RAM and you want to access all of them at the same time, then again the problem that we discussed earlier it will come, right. So, we try to identify the arrays which has access at the non overlapping time; that means, say this array one I am going to access in say timestamp 1 to 100 in this time. This I am going to access in say 200 to 300 this time, right.

And, this is going to access a say basically from 4 300 to 400 this during that time. So, that means, these arrays are actually getting accessed in different different time and hence they can be merge, but if the both array are accessing in the same loop then it will reduce the number of access and if some cases that might stop you to map the whole thing into a single block RAM, right.

So, for example here you can see here there are two access. So, if I map them into single array now we have a two access. So, if it is a dual port it is a dual port there is no problem because it is a dual ports you can access 2 data, but if you map another array A3 into the same array A then what we going to happen? Now we have three accesses and since the dual port if the performance will be affected by that.

So, we need to keep in mind that we try to identify the small arrays which are have non overlapping access or they are accessing in different different time let them merge into one and then that merge array can be mapped to block RAM ok. So, this is what is called merge horizontally, right. So, this is what is called merge horizontally.

(Refer Slide Time: 20:41)



Array Merge Horizontally

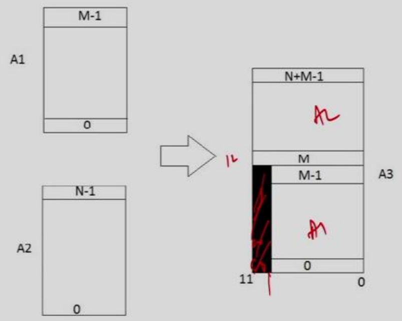
- two arrays are declared. A1 has M locations each of size 8 bits each and A2 has N locations each of 12 bits each

```
void fn()
{
        int8 A1[M];
        int12 A2[N];
        :
        :
        :
}
```

And, if you do it little bit formally. So, suppose you have an array A1 which has m locations and A2 have N locations and it is say 8 bit and this is 12 bit ok, this two arrays.

(Refer Slide Time: 20:52)

Array Merge Horizontally

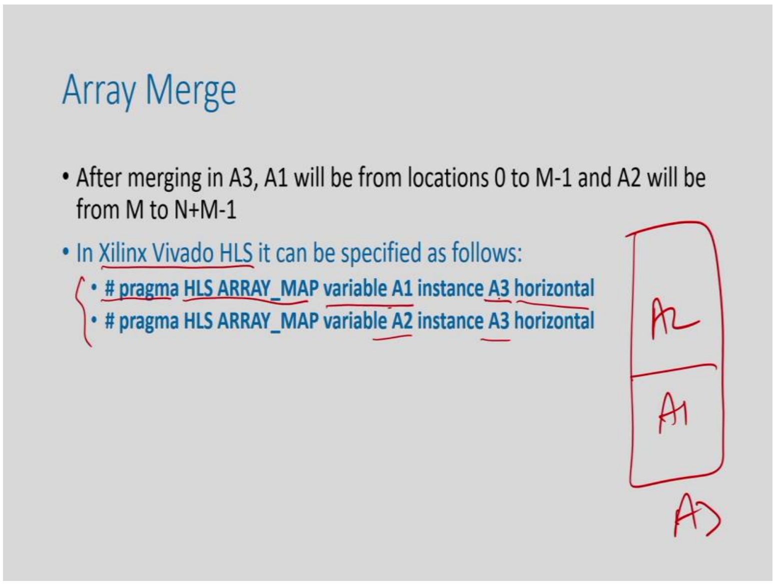


A1 and A2 are merged horizontally into A3

And, then how the mapping is happening? So, I am going to map this A1 here which is has 0 to M minus 1 location and M 2 N plus M minus 1 is my A2, right. So, that is where I am going to map this A2 and since this is basically say 12 bits and this is 8 bits. So, this part of the things is basically 0 padded, right. So, these are all 0 which is basically has no meaning because this is 8 bit.

So, these two array may not be of same data may not be sub same width, right. So, in that case the rest of the part will be automatically 0 padded, right. So, you do not have to bother about that because you are not accessing that; because when this map happened the tool will make sure that this particular location it has zeros, ok. So, this is what array merge horizontally.

(Refer Slide Time: 21:36)



Array Merge

- After merging in A3, A1 will be from locations 0 to M-1 and A2 will be from M to N+M-1
- In Xilinx Vivado HLS it can be specified as follows:
  - # pragma HLS ARRAY_MAP variable A1 instance A3 horizontal
  - # pragma HLS ARRAY_MAP variable A2 instance A3 horizontal

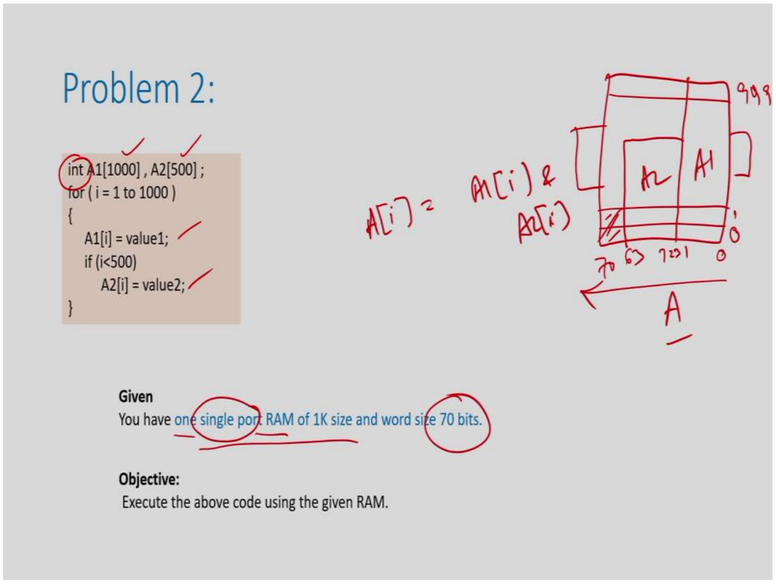
And, now you the tool usually the high-level synthesis tool have features to do that, you do not have to do yourself also, right. So, specifically because the tool that I am going to use in this particular course is Vivado HLS of Xilinx it has a pragma. So, whenever you want to do something you have to instruct the tool right you do something and that is given by the pragma, right.

And, since this is a array mapping the pragma name is HLS array map right it does means how do you are going to map the array to RAM during high-level synthesis, ok and then how do how to we have to specify? You have to specify the array. So, array is given by variable. So, A1 I want to map into A3 and it is horizontally, right. So, basically you have to specify for each array so, this is I want to map A1 to A3, A2 to A3, right. So, basically this is what exactly is A1 and A2 and this is my A3.

Exactly the example that I have shown here. So, if I just write this two line of code in your program it will actually automatically do this job for you right or you can actually do yourself by rewriting the code the way I have shown here. So, both are same thing.

So, writing the code yourself has more control and here if you ask the tool it may or may not do based on the situation, right or it might give you the warning that it is not possible to do it. So, both can be used ok.

(Refer Slide Time: 23:01)



Problem 2:

```
int A1[1000] , A2[500] ;
for ( i = 1 to 1000 )
{
    A1[i] = value1;
    if (i<500)
        A2[i] = value2;
}
```

Given

You have one single port RAM of 1K size and word size 70 bits.

Objective:

Execute the above code using the given RAM.

A[i] = A1[i] & A2[i]
999
A2
A1
0
70
63
A

So, I will now introduce another problem, ok. So, again you take the same example that I have taken from my previous problem 1, that I have 2 arrays A1 and A2 I am going to access both in a loop in a single iteration of the loop, ok. Earlier I told you there is a dual port RAM of size 2K.

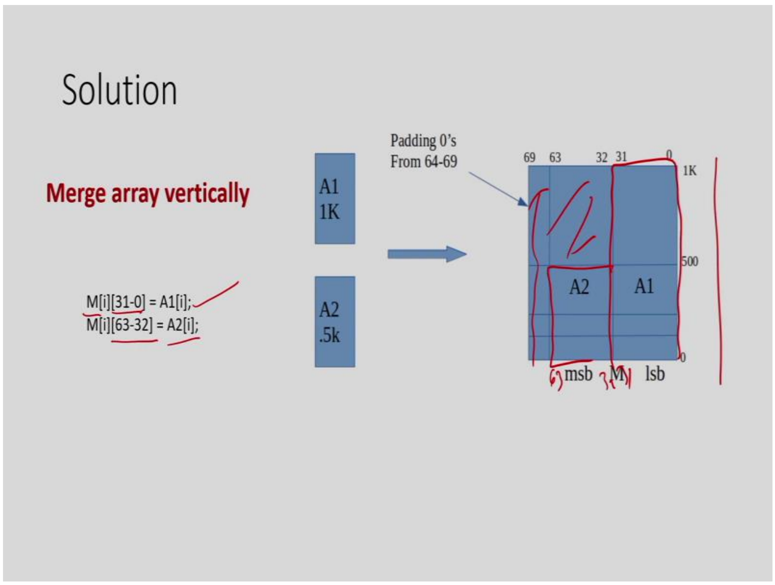
So, since this is a 1000 location is 500 so, both can be merged horizontally into the same array because it has enough locations, right? It has 2000 locations and I need 1500 locations. So, I can map them. And, it is a it was a dual port and hence even if I map both of them into the same RAM I can access 2 data in the same iteration because I need to access both of them. So, everything was solved with my merge horizontally things.

Here I am going to introduce another problem. So, I have a same behavior but I have now a single RAM, ok. First of all it is A1 RAM and it is a single port. It does not have two ports, so that means, even if you merge this into horizontally you have two access would not be satisfied because it has a single port, ok. So, you have a single RAM it has a single port, right. So, it has a single port not all like earlier there are dual ports and it has size 1K right.

So, this is 0, this is 1 to 999. So, it has 999 locations, but the width is 0 to 70 earlier it was 32 bits. Now, it is 70 bits ok. So, the that merge horizontally will not work primary of two reason. The first is it has only 1000 locations and if you try to merge them horizontally you need 1500 locations.

So, merging horizontal is not possible and also the another bottleneck is that it is a single port and you need to access two times in a single iteration. So, that is also not possible if you just merge horizontally. So, can you still execute this particular program using the given block RAM? Ok, that was the question. Again, you think yourself, pause it here and then move to the next slide for discussion ok.

(Refer Slide Time: 25:09)



Solution

Merge array vertically

```
M[i][31-0] = A1[i];
M[i][63-32] = A2[i];
```

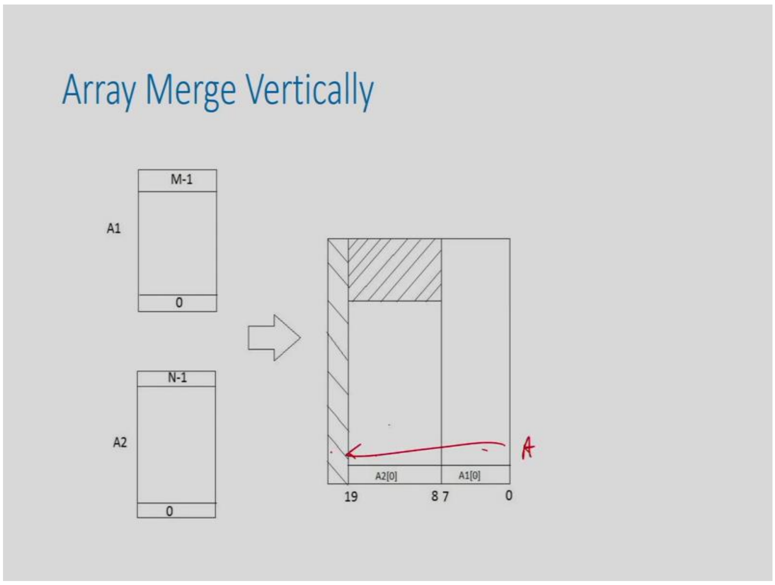
So, the solution here we have to be understood that we have 1000 location. So, hence you cannot merge horizontally obviously, but the things given here is that this 70 bits. So, I have lot of width in this direction, right. So, I need the integer which is 32 bits, but I have 70 bits. So, if I just map A1 to this here so, if say I map A1 here so, it will use 0 to 31. So, 32 to 70 that will be unused, right so, that is also unused bits.

What about mapping A2 here. So, this was my A1 and this is A2. So, I am actually mapping both A1 and A2 vertically, ok. What does it mean? So, I am now accessing this is my say array A, A i which is will give you both A1 i and A2 i, right. So, accessing just one location of this array will give the data both A1 and A2 i. So, just because and this is 32 to 63. So, this part is again 0 padded, right. So, that is what is called vertically.

If you have the RAM is lot of width in the data width of the array RAM is more then I can actually it is instead of merging horizontally I can actually map the array vertically as well, right. So, that is what is exactly discussed here that A1 is stored 0 to 31. And, then A2 is store from 32 to 63 and this is my because A1 of size 1000 and this is 500. So, this is what and this these are all part is 0 padded, ok.

And, how I am going to access? So, whenever I am going to access the memory location i 0 to 31 those bits is nothing but A i and 32 to 63 bits are nothing, but A2 ok. So, this is what is called merge array vertically.

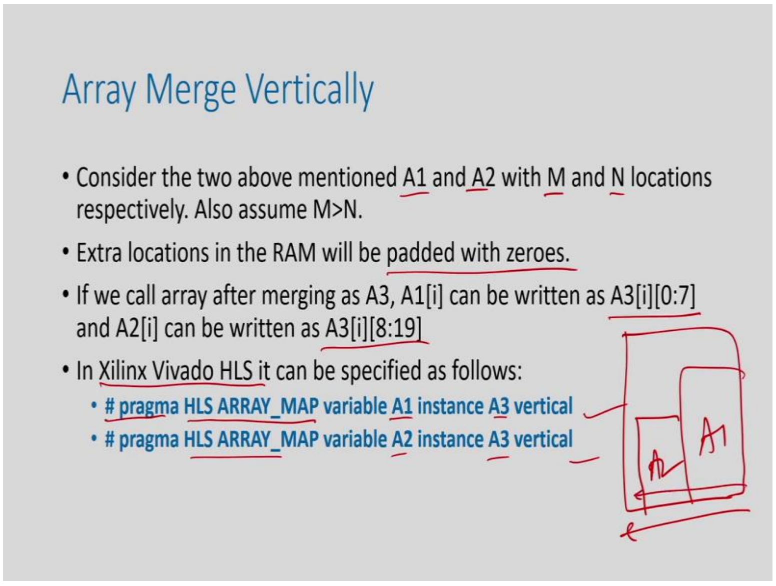
(Refer Slide Time: 27:14)



Array Merge Vertically

So, when there is no enough room to merge horizontally, you look for is there any space vertically and then you can merge vertically. Again, this diagram in the same thing that you have there are two array you merge vertically and one location specify two array locations ok.

(Refer Slide Time: 27:24)



Array Merge Vertically

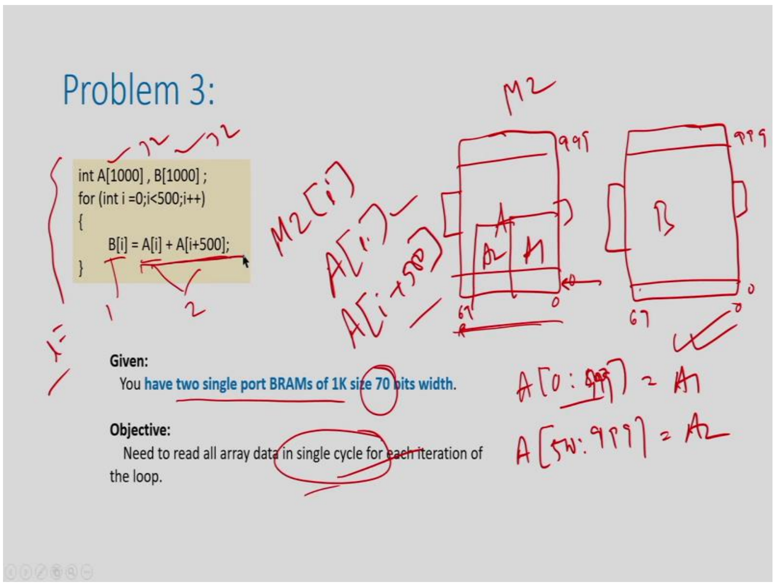
- Consider the two above mentioned A1 and A2 with M and N locations respectively. Also assume M>N.
- Extra locations in the RAM will be padded with zeroes.
- If we call array after merging as A3, A1[i] can be written as A3[i][0:7] and A2[i] can be written as A3[i][8:19]
- In Xilinx Vivado HLS it can be specified as follows:
  - # pragma HLS ARRAY_MAP variable A1 instance A3 vertical
  - # pragma HLS ARRAY_MAP variable A2 instance A3 vertical

So, again in the high-level synthesis so, the same thing if you just try to define the things formally if A1 of size M, A2 of size N and then what I am going to do? I am going to put a 0 to M a 0 to M that is my first location and then M plus 1 to M plus N minus 1 is the second location of the array of A i, ok.

And, then and the whatever the things are not used it is basically zero padded ok; and I can access by just doing this right by just doing this I can access the same location. Incidentally the tool like Vivado HLS has the facility to do the things automatically as well. If you just specify again the pragma that I want to instruct the tool to do something I want to instruct the tool to this array mapping of variable A1 to A3 vertically, array mapping of variable A2 to A3 vertically.

So, this is where exactly the thing is going to do is it will put A1 here and A2 here and if the width is not enough it will give a warning that it is not possible to merge because I do not have any RAM which has enough width to map vertically, right. Otherwise it will just do this exactly the things that I have discussed in the previous slide, ok.

(Refer Slide Time: 28:43)



Problem 3:

```
int A[1000] , B[1000] ;
for (int i =0;i<500;i++)
{
    B[i] = A[i] + A[i+500];
}
```

Given:

You have two single port BRAMs of 1K size 70 bits width.

Objective:

Need to read all array data in single cycle for each iteration of the loop.

Now, I will move on to the problem 3. So, another scenario I am going to introduce, ok and I am going to see what could be the solution, ok. So, let us take another new example where you have array A and B, both of the 1000 size 1K size and I am actually accessing B i equal to A i plus A 2. So, what is important here? I am accessing a B one time and A two times in an iteration, right.

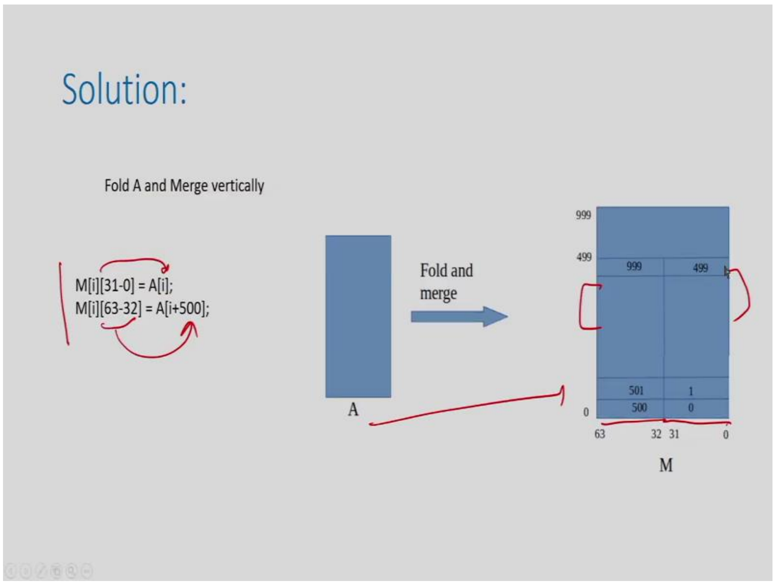
For i for each i I am going to access A two times B one times, ok and what is given to me? I have given two single port block RAM ok. So, I have given two block RAM of size 1K. So, that means, A and B can fit in there ok I have 1K size, but the width is 70 bits, ok and this is 32 bits, right. So, these are all 32 bits, this is 70 bits, ok. So, this is what is given to me that this is 0 to 69, this is 0 to 69 and the location is basically 0 to 999 and the locations are from 0 to 999, ok.

So, obviously, you can think what is the problem I just map A here B here, right. Obviously, because the size is matching width matching. So, there are lot of unused bits for each width, but the requirement is that I want to read all the data of A1 iteration in one cycle, ok. So, if the requirement is this obviously, if there is no problem in mapping here. So, there are two arrays, two block RAM, size is same both has single port.

So, it has a single port, this also has a single port it does not matter because. So, for B it does not matter because we have only one access, but for A that single port matters because it has two access in one iteration, but I want to access both of them in single cycle, right. So, that was the problem that for B there is no problem. It is perfectly mapping to A block RAM; it has one access single port. So, the one cycle requirement of reading is done.

But, for A it has since the block RAM has single port and this particular array has two access in a iteration. So, I cannot just read both A i and A i plus 500 in single cycle using the single port, ok. So, can I still map this program into this two array available to solve this to meet my requirement? Ok. So, again you think on this and pause here and you see the solution next.

(Refer Slide Time: 31:24)



Solution:

Fold A and Merge vertically

```
M[i][31-0] = A[i];
M[i][63-32] = A[i+500];
```

So, the solution here is yes, it is possible because I have lot of width available in A right it is basically the merge vertically thing, but since A is already of size fix what I am going to do? I am going to break a 0 to 500 this location into A1 and so, this is 499 to be precise and this is 500 to 999 this location into A2 right.

So, basically I am folding the A into two arrays, ok. Once you have this two array, I can merge vertically because this is 32 bits, this is 70 bits, right. So, that the way I hold it so that whenever I access this memory this is say my memory 2.

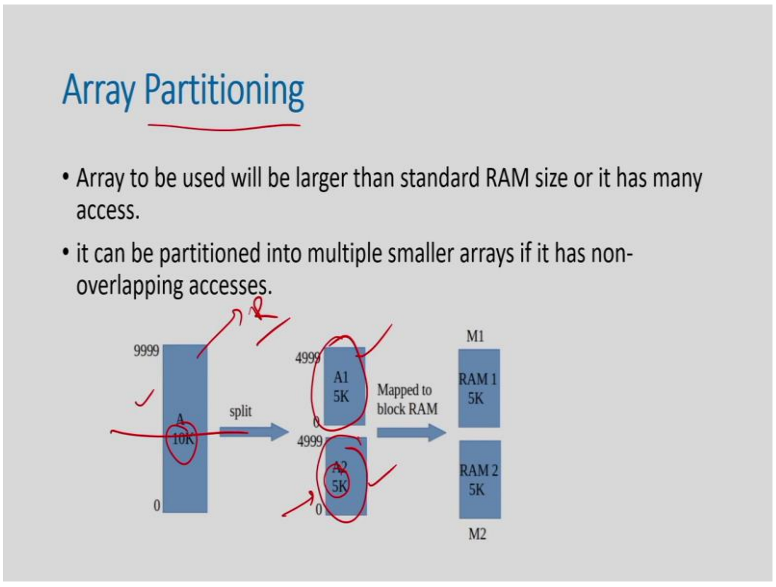
So, which is memory 2 i I am actually accessing A1 i basically I am accessing A i and A i 500, right because the single access of this memory. Because I am mapping this A1 here and A2 here whenever I am going to access a single location I am accessing both i and i 500 and that is what is my requirement ok.

So, this is exactly the solution that I talked about that I fold this A into 2 and I will store only in for the 500 locations and the first 0 to 31 is my A and this is A plus 500, right. So, i plus 500 ok and so, whenever I access M i which is basically 0 to 31 is A1 and 32 to 63 is my A i plus 500. So, I can actually do the things. So, this kind of specification is not possible by high-level synthesis tool like Vivado.

So, if you see that you are actual taking lot of time you want to improve the performance of this kind of array accesses and still you want to map them into a single block RAM you have to rewrite your code in this manner so that once you rewrite this code the tool will map this particular A into a block RAM and see although it has a single port still can execute the things in single cycle ok all the access in single cycle, ok.

So, I have already discussed three scenarios where we actually have to merge the arrays because the to utilize the array better or the memory better, right. So, either you merge horizontally or you merge vertically, but both will give you the lot of performance benefit, ok.

(Refer Slide Time: 33:53)



Array Partitioning

- Array to be used will be larger than standard RAM size or it has many access.
- it can be partitioned into multiple smaller arrays if it has non-overlapping accesses.

9999
A
10K
0
split
4999
A1
5K
0
4999
A2
5K
0
Mapped to block RAM
M1
RAM 1
5K
RAM 2
5K
M2

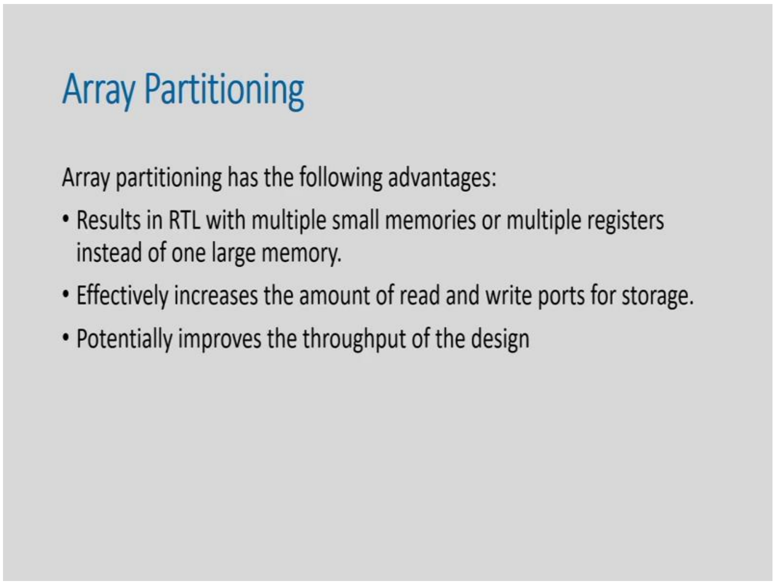
Now, I will just talk about the reverse side that sometime you probably have to partition the array as well, ok. So, what is the scenario where you have to partition the array is, basically if you have a very large array and your RAM size is smaller, right. For example, you have say array of 10K size and your block RAM is only 5K. So, you cannot map this 10K into a block RAM because it has a lot of double size ok.

So, if it is the case what the tool usually do, it put all this data into register and you can understand that it will create 10000 into 32 number of bits of registers which is huge, it might occupy all registers of the PGA board ok. So, whenever you have a very big array you have to be very careful about mapping them into block RAM block mapping them into block RAM is kind of must for very large arrays, ok.

And, sometime if the array size is too big that cannot be fit into a single RAM then the best idea is just partition into smaller arrays so that each small array can be mapped to different block RAM, right. So, that is the idea and how you should partition? The partition should be such a way that the access is distributed, right. So, if the locations here and the locations here should be access in parallel, but the locations of here should not be access in parallel, then they have to access in the same clock and that will create a bottleneck, right.

So, when you try to partition the array it is not that you split it here. There are many way you can split it I am going to discuss it here, but the splitting should be in such a way that the accesses of different components would be in parallel, the access of the content of the same array should be sequential so that the bottleneck of the port should be avoided, ok. So, that is what the idea, ok.

(Refer Slide Time: 35:36)

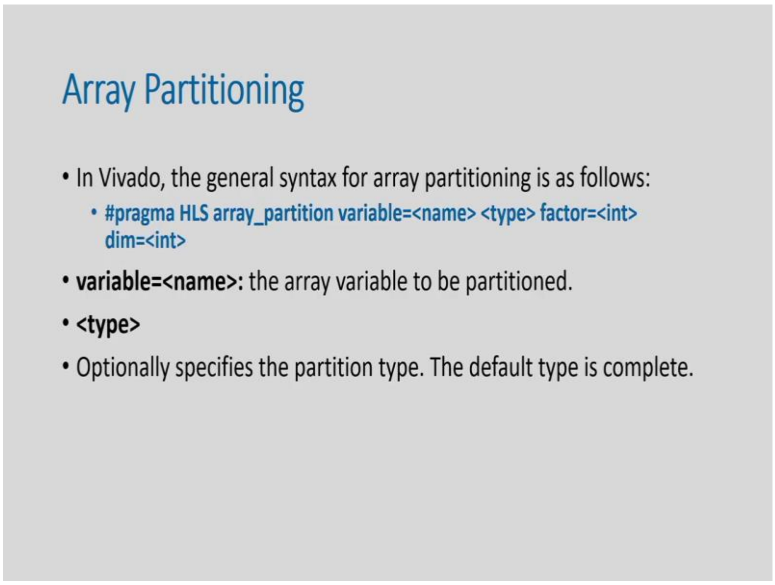


Array Partitioning

Array partitioning has the following advantages:

- Results in RTL with multiple small memories or multiple registers instead of one large memory.
- Effectively increases the amount of read and write ports for storage.
- Potentially improves the throughput of the design

(Refer Slide Time: 35:37)



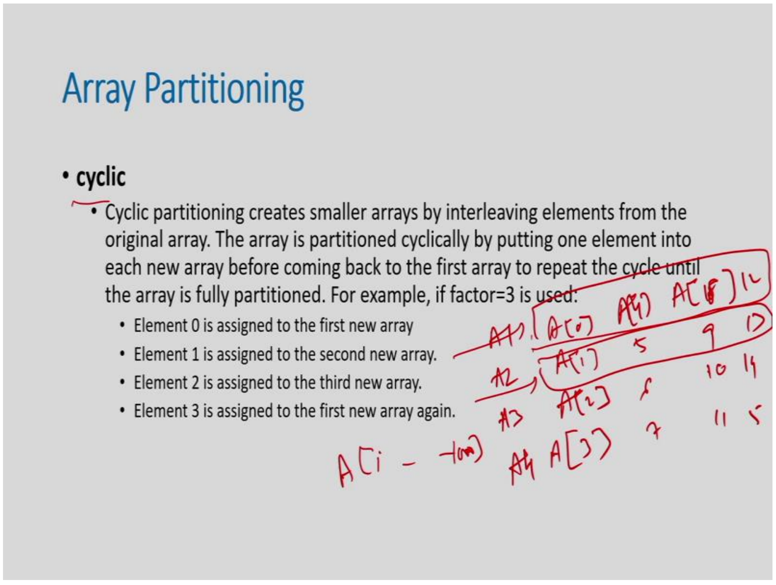
Array Partitioning

- In Vivado, the general syntax for array partitioning is as follows:
  - **#pragma HLS array_partition variable=<name> <type> factor=<int> dim=<int>**
- **variable=<name>:** the array variable to be partitioned.
- **<type>**
- Optionally specifies the partition type. The default type is complete.

So, again you can do this using the array partitioning you can do it manually or you can do using the pragma HLS partition HLS array partition. You mention the array name, you do the what type of partition you want to do I am going to discuss those, what is the factor, how many small part you want to make a break the array and which dimension you want to break right. So, I am going to discuss them ok.

So, the array variable name is the array name. So, you mention the array which was the big array you want to partition, ok. So, now, based on the access pattern of the array you have to decide whether what is your partition type, ok.

(Refer Slide Time: 36:14)



Array Partitioning

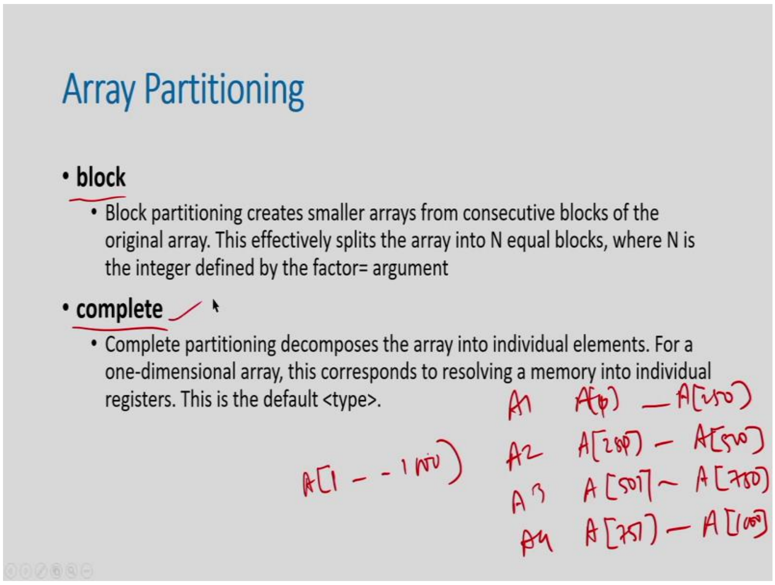
- cyclic
  - Cyclic partitioning creates smaller arrays by interleaving elements from the original array. The array is partitioned cyclically by putting one element into each new array before coming back to the first array to repeat the cycle until the array is fully partitioned. For example, if factor=3 is used:
    - Element 0 is assigned to the first new array
    - Element 1 is assigned to the second new array.
    - Element 2 is assigned to the third new array.
    - Element 3 is assigned to the first new array again.

So, there are three parts here partition type – one is cyclic, right. So, if the array is say A1 to say 1000. So, in 1000 so, in cyclic and say suppose you want to break into 4 component.

So, what I am going to do I am going to put a 0 in the block 1 right. So, you want to partition into 4, right. So, I just put in A1 I will put A0 and then say in A2 I am going to put A1 because this is a cyclic manner, right. It is not something in the A3 component I am going to put A2 and A4 component I am going to put A3, right and then cyclic way I am going to do A4 here, A8 here right, 8 here, 5 6 7 here, 8 9 10 11 here, 12 here, 13 14 15. So, this is how I am going to cyclic way I will just break it.

So, if the access is such that you are actually accessing A0 and A1 at the same time A4 and A5 in the same time and so on. Probably it is better idea that you put A0, A4 into one block and A1, A5, A 14 those things into another block and so on. So, that even if you want to access A0 and one A1 in the same time this can be accessed from two different location of block RAM so that they can be parallelized, ok. So, this is when the cyclic is important.

(Refer Slide Time: 37:44)



Array Partitioning

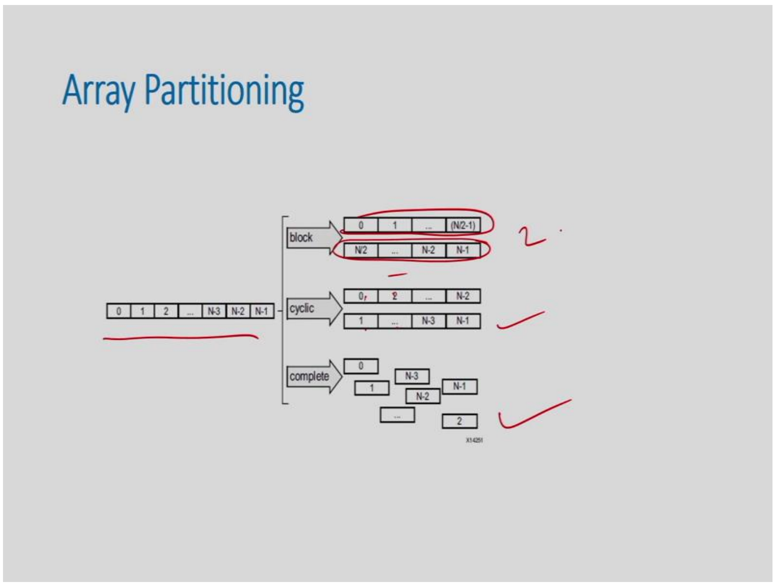
- **block**
  - Block partitioning creates smaller arrays from consecutive blocks of the original array. This effectively splits the array into N equal blocks, where N is the integer defined by the factor= argument
- **complete**
  - Complete partitioning decomposes the array into individual elements. For a one-dimensional array, this corresponds to resolving a memory into individual registers. This is the default <type>.

And, the block is something is obviously, you understand that suppose I want to block partition the A again the same example say A is 1 to 1000 there are 1000 location. So, in A1 I am going to put A 0 to say A1 to A 250 say ok and say I want to partition into 4 right. So, in partition 4 I want to put A 250 to 500. You understand the point, right.

So, the first part you want to put in array 1, A3 I am going to put A 501 to A 750 and A 4 I am going to put A 751 to 1000, right. So, this is how I can actually split. So, I just give the first part into block 1 second part into second say block 2 and so on. This is block and if you mention complete, it will split into registers, right. So, in some cases say there is the other way you are accessing even if you partition it does not solve your purpose.

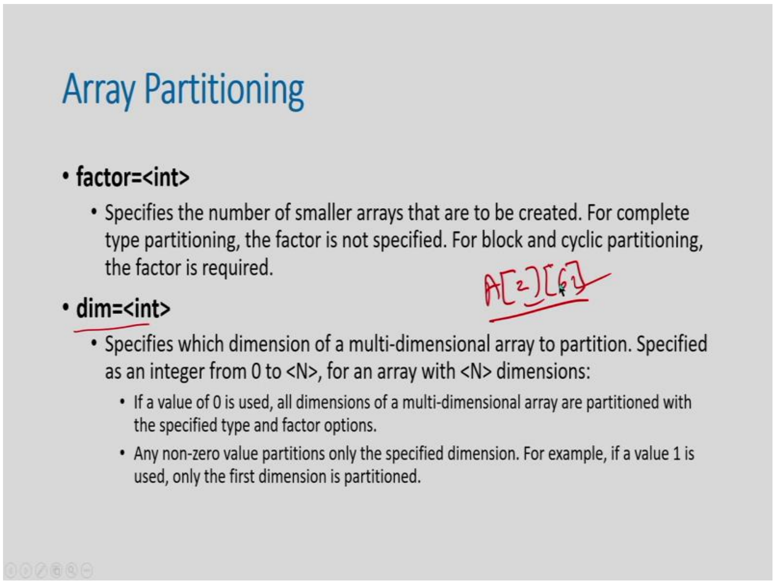
So, then you can actually split the things into individual registers, ok and it is also useful the complete it is a very small array, no point actually mapping them into block RAM because a block RAM size is minimum is say 500 and this array is A 10, A 12. No point holding a 500 location for 10 datas, right rather this split it. So, that time you again to partitioning and you mention it is a complete partitioning; that means, this array will be distributed into registers. So, that is also where this array partitioning will help.

(Refer Slide Time: 39:18)


Array Partitioning

So, the exactly diagram that I try to show. So, if you have these locations. So, block will put 0 to N minus here N minus 2 here and so on and cyclic I am just 0 1 2 3 and so on, right. I am showing two partition here and complete into individual registers, ok. So, what is the factor? So, how many component you want to create. So, here your factor is 2. The example I have taken the factor was 4. So, how many small partition going to create that will be given by the factor ok.

(Refer Slide Time: 39:47)

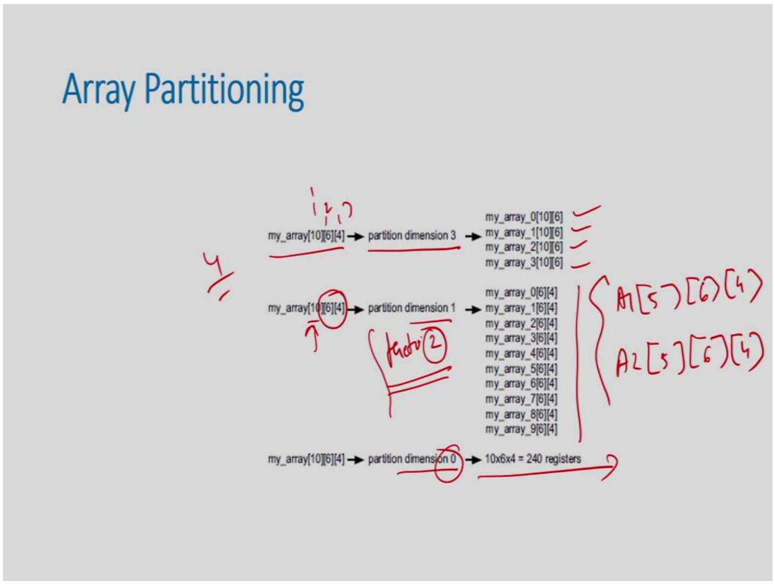

Array Partitioning

- **factor=<int>**
  - Specifies the number of smaller arrays that are to be created. For complete type partitioning, the factor is not specified. For block and cyclic partitioning, the factor is required.
- **dim=<int>**
  - Specifies which dimension of a multi-dimensional array to partition. Specified as an integer from 0 to <N>, for an array with <N> dimensions:
    - If a value of 0 is used, all dimensions of a multi-dimensional array are partitioned with the specified type and factor options.
    - Any non-zero value partitions only the specified dimension. For example, if a value 1 is used, only the first dimension is partitioned.

A[2][6]

And, the dimension is also important; whenever there are a multi dimension array, right so, there are say A 2 A 6. So, it is a two dimension array. So, which dimension you want to partition? You want to partition, this dimension or this dimension. So, that something you have to decide and that is given by the dimension ok.

(Refer Slide Time: 40:07)



Array Partitioning

my_array[10][6][4] → partition dimension 3 → my_array_0[10][6]
my_array_1[10][6]
my_array_2[10][6]
my_array_3[10][6]

my_array[10][6][4] → partition dimension 1 → my_array_0[6][4]
my_array_1[6][4]
my_array_2[6][4]
my_array_3[6][4]
my_array_4[6][4]
my_array_5[6][4]
my_array_6[6][4]
my_array_7[6][4]
my_array_8[6][4]
my_array_9[6][4]

my_array[10][6][4] → partition dimension 0 → 10x6x4 = 240 registers

So, if you say I into partition the dimension 3 for this array or so, the dimension 3 means it will. So, this is my dimension 3, this is 2, this is 1, right and then what will happen it will create and say I want to do a 4 factor is 4. So, then what is going to happen? So, I am going to create 4 such array, right.

So, the third dimension will create each partition, ok and similarly, if I put try to put the dimension 1 you do not have to mention the factor because if you just mention I want to break the dimension 1, then what it will create 10 such partition right each of them of size 6 into 4, right. So, this is how I am going to partition and if you just makes the partition dimension 0 it will make into register, right. So, it will partition into individual array.

So, one important thing here is that if you just mention I want to partition the dimension 1 and I want to make a factor equal to not 10 rather say it is 2, right. So, that means, what is going to happen this dimension 1 will be splitted into 2 arrays. So, that time I will create 2 array A1 which will be of size 5 6 and 4 and another dimension which is basically again of size 5 6 and 4, right.

So, mentioning both if you do not mention factor it will split the whole dimension into individually or if you mention the factor plus dimension, then it will split that particular dimension based on the factor. So, since here I mentioned the factor equal to 2 and the dimension size is 10, so, it will split into 2 part, right. So, this is how it will create the things ok.

So, I hope you understand and again this is something based on the application. So, suppose you have written a code and you are seeing that because of this array it is you want to map this array to a particular block RAM and it is getting splitted into register and as a result you are saying that your area is not sufficient to map, right and you identify that because of a block big RAM big array is getting mapped to register the problem is happening.

So, what you can do is, you just partition the based on that particular array just by you running one, you analyzing the fact you understand that and you actually partition the things. And, then after partitioning probably those small arrays will be mapped to block RAM and hence your performance will be achieved or you are able to map the whole thing into a target architecture, right.

So, these are all advance level of things, but once you are actually becoming master of high-level synthesis you need to apply this kind of techniques to achieve your target, ok. So, just to summarize before conclusion that in this discussion we have seen that how the array is getting mapped to hardware is basically mapped to block RAMs or ROMs and we have seen that since the array that block RAM has a limited access.

So, it is better to reduce the parallel access as well as possible and whenever you have the content which is getting multiple time read we are accessing the same data multiple times instead of reading it memory multiple times it is better to read it once stored locally and reuse, right that was the idea.

And, for bigger arrays you probably have to partition them into smaller arrays to fit into target block RAM or if you have on smaller arrays it is better to merge them into to other so that is the utilization of the block RAM will be much better ok.

So, that is these are the things tricks you have to taken care during efficient synthesis of array into hardware, ok.

Thank you.