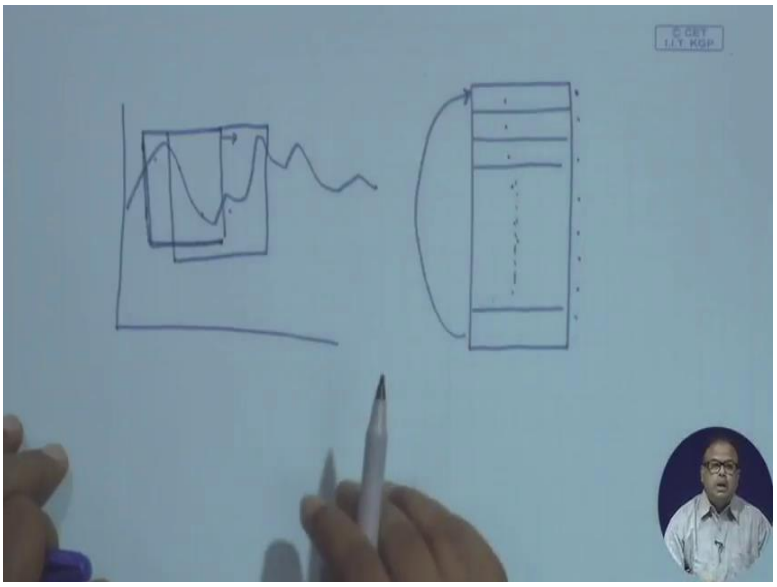
Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 25
DSP Application and Address Generation Unit

Now another way of reducing the memory is by a new addressing mode which is known as modular addressing.

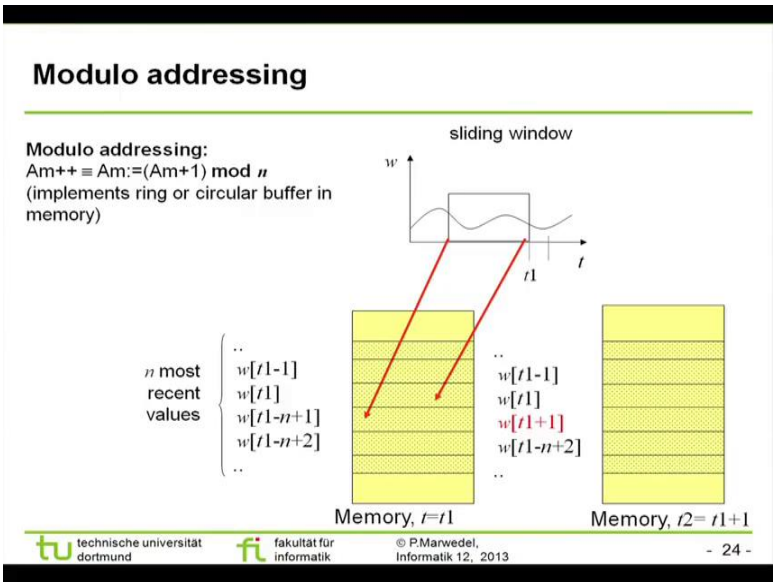
(Refer Slide Time: 00:35)



That means if I have got a most of the things are repeated. So, if I have got a limited space and limited address locations I will be loading the data in these locations one after another, and after that I will again rollback to this. Typically what happens is I have got to deal with some signal that is going on. Now I want to deal with that in DSP there is a technique called windowing I can fix up a particular window, and I can move this window. So, whenever I am taking this window in then the data that I take in this window will be coming here.

Next when I move that window gradually so here I move that window to the next part. Then some of these locations will be rotated back and the same location can be used that is known as modular addressing.

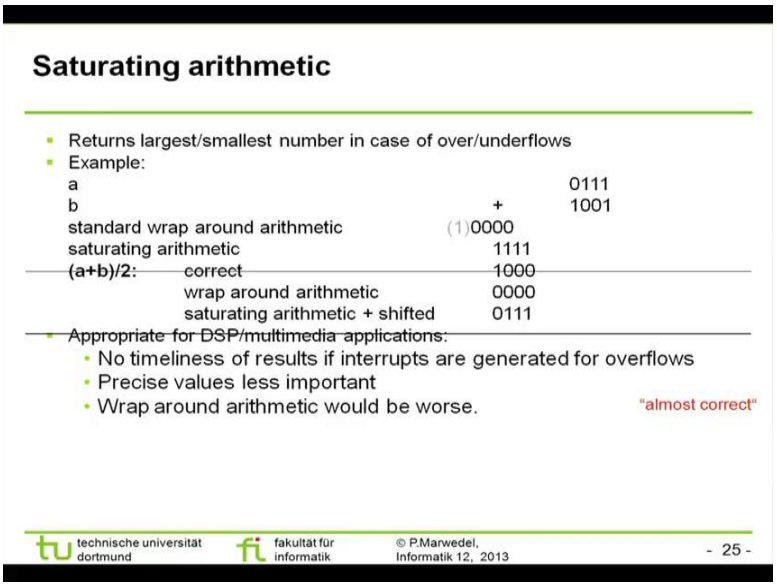
(Refer Slide Time: 01:52)



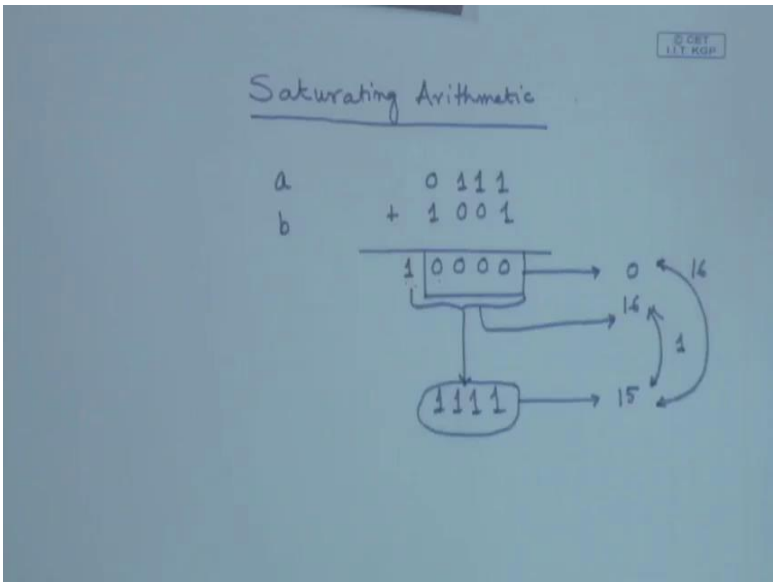
So, that is explained in this slide. So, I have got some address I increment that address Am plus plus say is Am is just the address plus 1 mod n; if n be my size of this array. The example is I am just taking the n most recent values. So, some w t 1 some particular time say this 1 earlier than that minus 1 next couple of them. So, here is my sliding window, and on that sliding window as I slide a particular as this one is slided; I go back. As I slide this window then once again I was here and this point was loaded here, this point was loaded here. As I slide this window it goes on that side.

Now, this thing will be this n minus 2 that was there. Once again let us see this animation. I was here and the first this one was here and the earlier one, now this one will be the earlier one, but as I move it then this one will be the earliest one. So, this one was somewhere here and that will go down and will be coming here. In that way I can modulate over this that is another way of doing it.

(Refer Slide Time: 03:41)



(Refer Slide Time: 03:54)



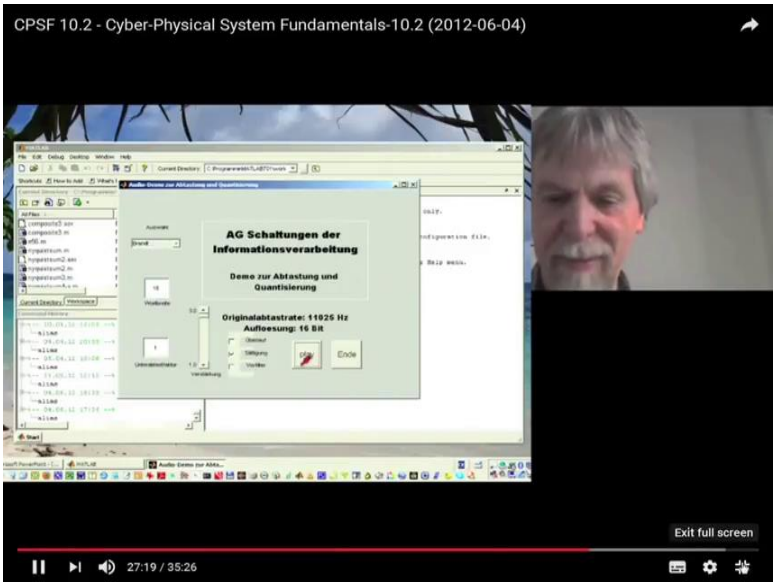
Another very important thing that we do is; let me just come to this straight on paper this is known as saturating arithmetic. It means that if I have some say value which is 0 1 1 1 and another value b which is being added to that is 1 0 0 1. If I add them the result will be 1 is coming as a carry and all these zeroes. Now if I restrict myself to 4 bits only the result that I will be getting will be 0, whereas actual result would be what, 15, not 15 16 actual result would be if I had taken this whole thing it will be 16.

Now, in saturating arithmetic what we do is we take the earliest most accurate one, so this one in saturating arithmetic will be 1 1 1 1, because there is an overflow just before that it would be this value. So, if I take this value then it would be 15. So, from the actual my error is only 1, whereas if I had not done the saturation arithmetic then my value would be the error would be 15 or 16. So, that is something that is adopted in order to restrict on the data size, but by applying this thing I will just try to demonstrate to you the effect of this saturating arithmetic versus non saturating arithmetic it means if I succeed in doing that.

Student: does not mean that two values there be only 1 (Refer Time: 06:29).

No obviously, I come to the closest value when I saturate my 4 bits with that how much maximum can I come to 15. So, I am coming to that had it been other 8 bits or something it would be different.

(Refer Slide Time: 06:50)



So, here there is I have borrowed it from our lecture where this has been demonstrated by; I hope the sound will come, let me see if the sound is on and here let me put the sound to max

That you were saying small audio example (Refer Time: 07:19) general suggestion (Refer Time: 07:32) on the matter.

Demonstration has been given with respect to historic speech. First it is given with non saturated.

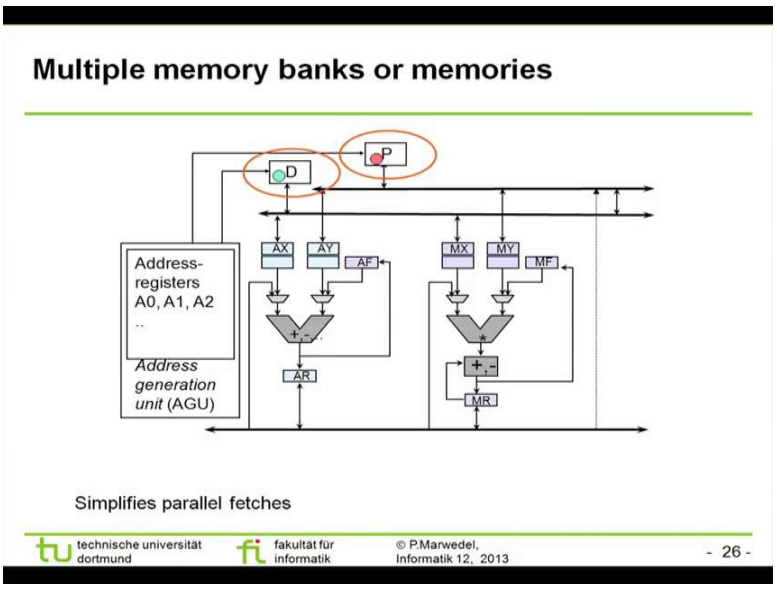
So, I am using a speech sample which is in German, because that was one that we had access to was our violating any license constraints. And now I am going to play that little speech example it is in German, but I mean the content of the speech does not really matter in this context. And I am going to play this little sample first of all using this standard (Refer Time: 08:46) arithmetic where you will hear that there is a lot of distortion

First non-saturating will define [FL].

I think that was sufficiently poor. So, now, we are trying the saturating arithmetic. There will still be some distortion, but I think you will hear the difference [FL]. Much clearer is it.

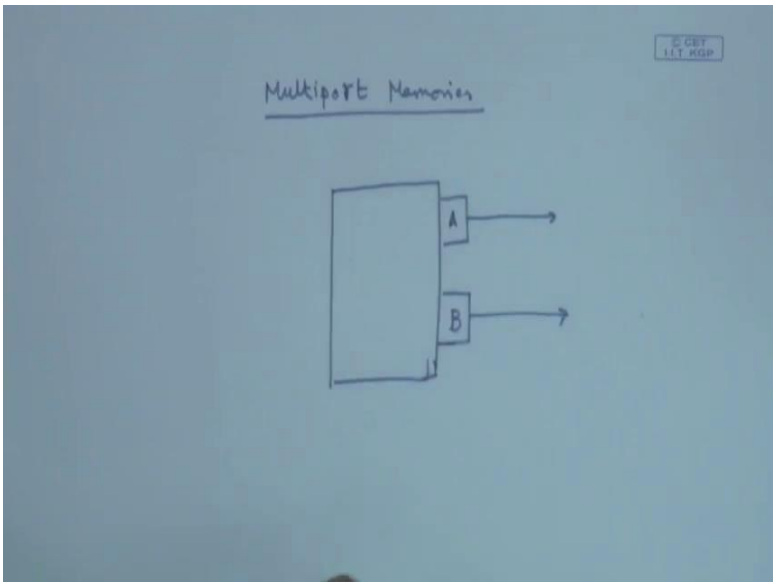
As you could see you could see how clear it was. Now they have to do something over that. So that was the scenario of say just by saturating arithmetic how the quality of the bit patterns can be improved. So, we will conclude our discussion on such enhancement of efficiency with the last point.

(Refer Slide Time: 11:14)



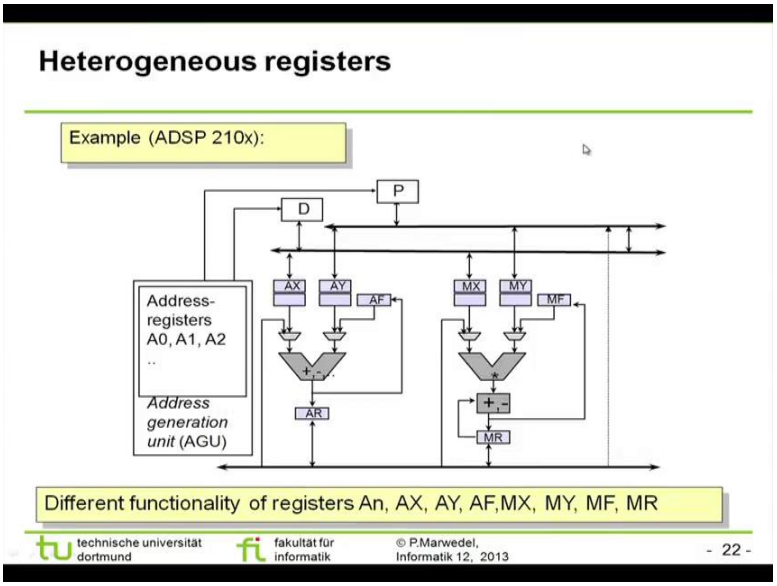
There is happened here with another thing that is as you can see in typical DSP processors we have got multiple memory banks or multiple memories.

(Refer Slide Time: 11:26)



Now, there is one concept of multiport memories that is where a particular memory block can have multiple read write ports. Say suppose there are two read write ports in that case two programs or two processes can simultaneously access the memory. With of course, there are number of issues that if one is writing another is reading from there, there will be some conflicts and races and all those; so that apart, that is one way of doing it. And the other thing is if you can have multiple memory banks where we have got data on both of these, and in the same cycle therefore if we have got multiple memory banks we can fetch more than one byte of data at the same time.

(Refer Slide Time: 12:47)

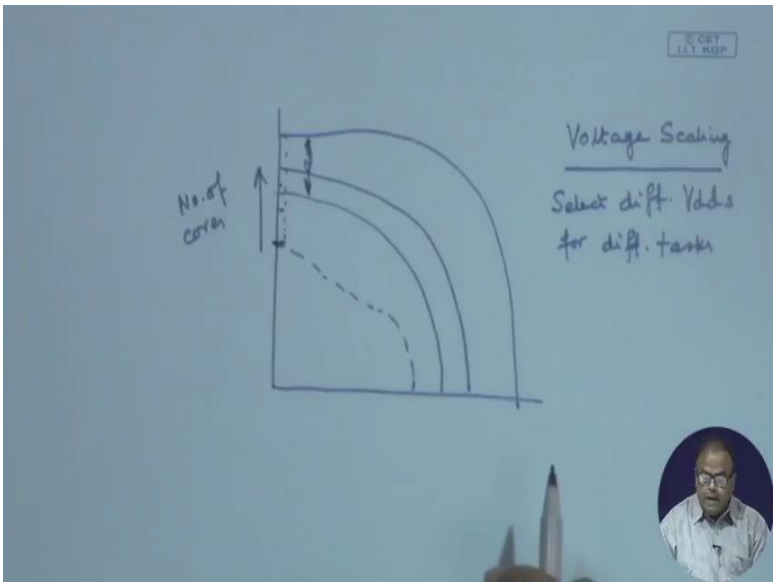


So obviously, that is another advantage of the DSP processors where we can utilize as we did in our earlier example that we worked on where we did this MX and MY were being fetched the two values of w value and a value were being fetched simultaneously. That is possible because we have got multiple memory bank or memories. So, that is another advantage another measure that adds to our efficiencies.

Now we have got many such other measures, but I just tried to present some of the representative measures and regularly novel features are being brought in so that we can have more and more efficiency in terms of run time. So just summarizing; let us summarize for a second in this I understand that it has become a little more loaded to all of you. So, let me just quickly summarize what are the things we have done in this; regarding power.

First thing is that we know about the power equation and that is alpha times the load capacitance Vdd square and the frequency. On the other hand the frequency or the inverse of that the delay is proportional to Vdd; sorry inversely proportional to Vdd. As we increase the Vdd we can go for higher and higher frequencies. Now as I go for higher and higher frequencies there are some problems also.

(Refer Slide Time: 14:42)



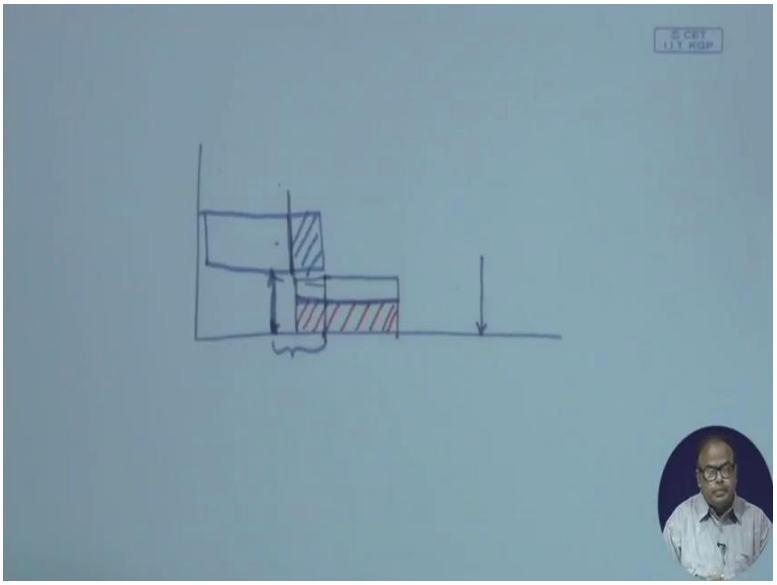
Now, a very important thing that we had looked into that we had the power area the power envelope that we had was something like this that we are restricted by some area curve like this, where we can go on increasing the number of cores to gain efficiency.

But we are also restricted by some power line here beyond which; we cannot go beyond this line in power.

Therefore although I put in more and more cores here these cores will not be used. And sometimes we do not want to access that, but then we are losing although we have put in some cores here. So, how can I increase this, how can I utilize this area? The one way is that we reduce the voltage. Now if I reduce the voltage then obviously I incur some delay, because frequency is coming down. So, there is always a tradeoff so I can still access this part. But there is a limit I cannot access the whole thing because of bandwidth limitation, I cannot put in more and more cores.

So, how can we schedule this? How can we schedule the different voltages, for any getting different performances? We can do voltage scaling. And voltage scaling means I can select different Vdd's; select different Vdd's or supply voltages for different tasks. Now that necessitates that you should have a number of discrete voltage levels available to us.

(Refer Slide Time: 16:52)

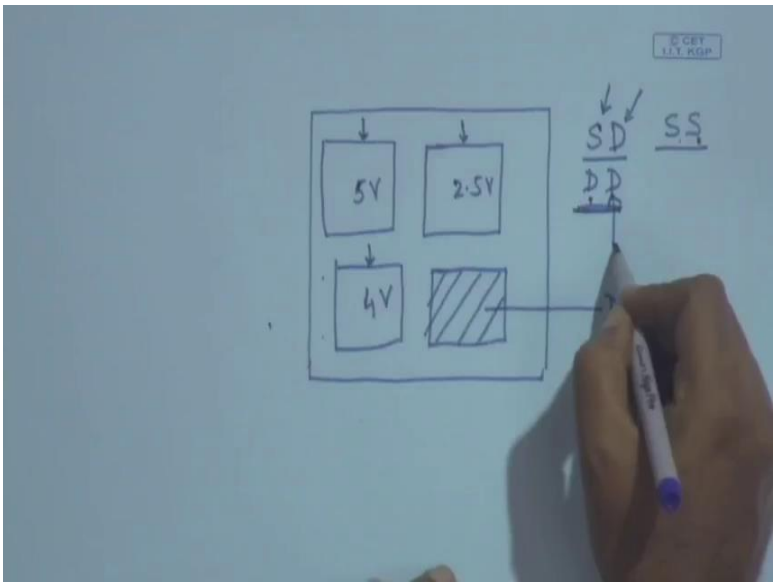


And when we have got some real time constraints, like we have to some jobs are arriving at some point of time and I am sorry the jobs are arriving here and there is a deadline here, so I have to finish it. Suppose the job can be started at this point of time. Now I can give it the maximum voltage and finish it very quickly, but that is not necessary because I still have got some space to play with. Or if the earlier job it came here why did not I

schedule it here, I did not schedule it here because there might be some earlier job which was running till here.

Now if this earlier job finishes early, suppose it finishes here then I get this much of to be the free time and so I can start my job here. And consequently since I am getting more slack time I can reduce the voltage. And ultimately I can make the whole thing run at a lower voltage and yet meet the deadline. So, that is voltage scaling.

(Refer Slide Time: 18:14)



The other thing that was there was dynamic power management we have got a complete circuit. Now we have also seen that by parallelism we can reduce the voltage. I can afford to reduce the voltage and consequently the energy if I have got multiple operations being done at different points. Also I can have different blocks inside the core and all of them may not be operating at the same point of time. So, some part may be that suppose I have designed one embedded chip. Say for example, for telephone for mobile phone.

Now you want to in a particular version, you may like to include some features or may not like to include some features. If you do not include features then this part may not be required. So, you never put the power on on, this you must have the selectivity that only the parts which are required to be powered on I can power on and this will be a dark silicon in that place.

Also in the earlier example here where we are talking of although I can put so many numbers of cores because of power constraints I may not access some parts of the circuitry silicon. So, that is also the dark silicon. Dark silicon is the silicon which I am not being able to power on or utilize may be intentionally. And again out of these by voltage scheduling we can put in different voltages at different. So, here I can put in 5, volt this I can put in 2.5 volt, this I can put four volt I can make them operate at different voltages.

So, all these techniques are available for managing the power. We have also seen an integer linear programming model of scheduling. And for this dynamic voltage there are two different algorithms that we came across: one was the SD algorithm and there was the DD algorithm. What is the difference between them? The SD algorithm was it is a static scheduling; that means I know when the jobs will be scheduled, but I play with the voltage and I can change the voltage and make it dynamic.

Compared to the static both being static the scheduling is static and the power is also static, I have got not much play to reduce that. On the other hand the dynamic one here by schedule is not known apriori. So, jobs can come at any point of time and as the jobs are coming in I have to adapt and accordingly my dynamic scheduling here will also be adaptive with respect to the arrival of the jobs. These are two algorithms that we have seen.

Next we move to the task of enhancing the efficiency of the code. There we have seen today the code efficiency can be done by compressing the instructions as is done in ARM. And also we have seen a very interesting approach which is the dictionary based approach using which we can reduce the amount of memory that we have to address all the time. Next we move to the separate class of processors that is the DSP processors. The DSP processors have been optimized with respect to some very common DSP applications that is the multiply accumulate, where we have got specific hardware which does this multiplication and addition in the same cycle. And along with that we have got the zero-overload I mean loop; the loop overhead; zero-loop overhead.

Hardware or by prefixing instructions we can do that. So, that the loop index is also checked while the previous loop body is executed. While the loop body is executing it is checking whether I should go for one shot more or not; that is one. And being a separate

there are mostly the arrays are addressed over there. And typically I will discuss that again further when I discuss about the operating system, that for embedded system design we do not prefer dynamic data structures like linked lists. We can have, since we know the application apriori we can decide on the size of the linear array that we will be using. And therefore, those arrays have got a particular size and there will be a certain number of arrays in a DSP processor application.

So, accordingly we have got specific address registers. And there is a special hardware which allows this address registers to be (Refer Time: 23:58) to be updated along with the computation of the code. Besides we have seen new type of addressing that is the modular addressing and the saturating mode of arithmetic, and of course the multiple memory features that is also there in the DSP.

So, all these help us in making it efficient and I am sure if you work on this you can also invent or innovate some other architectural measures or for specific applications. It would be interesting that you take some embedded applications and you yourself play around with that and see how you can make it more efficient; that would be a nice exercise for all of you to do.