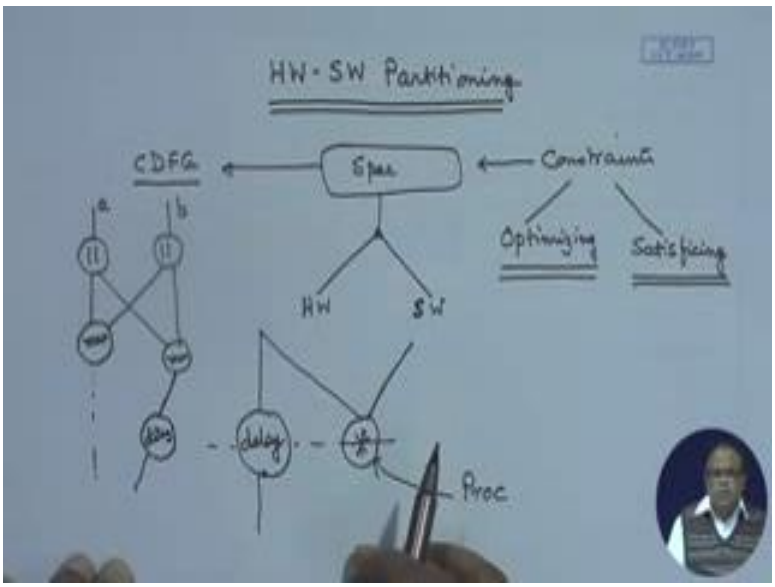
Embedded System Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 45
HW-SW Partitioning

(Refer Slide Time: 00:24)



Today, we will start discussing about a topic, which I was mentioning in the passing for a long time that is hardware-software partitioning. Now, in the last example that we have done we have seen that we have arrived at the design in an iterative way. So, what we did is we tried with different options started with only software option and then found that the constraints were not met. Then we gradually shifted towards hardware for some specific parts; and also did some software adjustments like instead of fixed point, instead of floating point going to fixed point and iteratively checking whether the constraints are made. Now, we can also look at it a little more systematically also one is alternatively, but that is very effective and mostly done, but there are some algorithms for doing that too.

So, before we go to hardware-software partitioning we know what it is that we start with the specification. And then during the design phase, I decide which parts will be in hardware and which parts will be in software. Now, along with the specification, I have got some constraints; and the constraints are of two types one is optimizing constraints

and the other I call it satisfying constraint. By that what I mean is that it is enough to be able to satisfy these constraints we did not optimize it.

Say for example, we are given a time constraint that the entire thing will have to be done within some stipulated time. Now, that we have to satisfy, we need not optimized all the time, but while doing that suppose the statement is that we will have to satisfy the timing constraint with minimal area possible, then we have to try to optimize on the area. So, that is why there are two types of constraints one is, now how do we make a decision as to which one will be done in hardware and which one will be done in software.

Obviously the advantage of implementations using hardware is that they will be faster and con side it is that they will be more expensive more power consuming maybe at times, they are not always, not always. Since, they will be faster the energy may be saved, the time to market will be higher etcetera. And whenever now this specification is coming to us in the form of a say control and data flow graph that we have seen already when we studied scheduling.

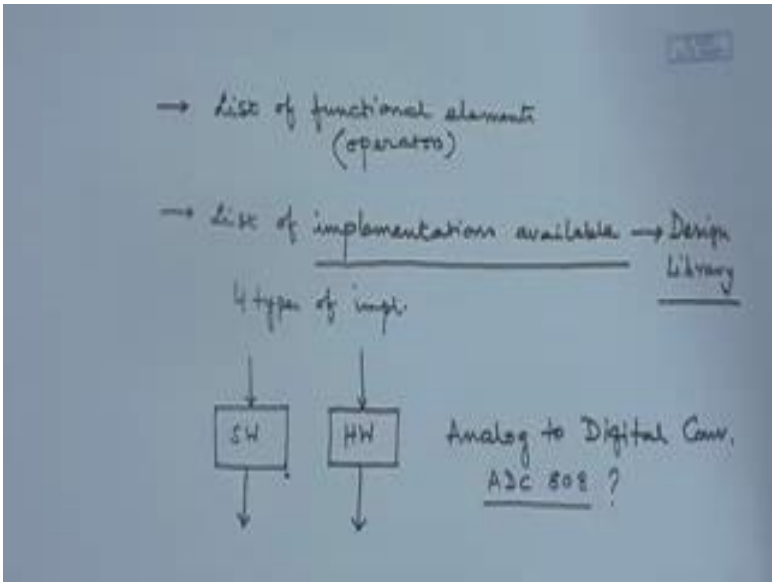
Now, when we are studying scheduling we are given a control and data flow graph and we decided on say for example, let me recollect the square root approximation algorithm. So, it was something like this that we had an absolute computation for a, we had an absolute computation for b, and then there was a max computation, there was a mean computation and so on and so forth. So, we proceeded in this way. So, in that way, there was a graph; and we had the option of selecting the absolute computation either by hardware or by software.

Now, there are several issues here regarding say for example, in say one of these operations are something like d a say somewhere in some hypothetical graph, we have some delay here, is it treatable, otherwise I am just showing it separately that in my graph somewhere I have got a delay. Now, this delay can be done by software I can just put a software loop, width loop or I can put a timer. So, I have got two options of doing this. So, depending on whether I want to do it using software or hardware, my cost will vary.

But suppose it was something like this that is here and the same step in the same control step to another path, I am going to do some multiplication. And suppose I decide that this multiplication will be done by the processor as I decide that this multiplication will be

done by the processor and since I am in the unique processor environment that is my assumption that I am doing one person. Then this delay cannot be done in our software therefore that is ruled out vice versa also that if I commit at per that this delay will be done in software then this multiplication cannot be done in hardware. Unless, I change the schedule and delay this delay from this control step to some other control step or delay this multiplication to some other control step. Therefore, my hardware software partitioning decision is also integrally related with the scheduling that we do.

(Refer Slide Time: 07:40)

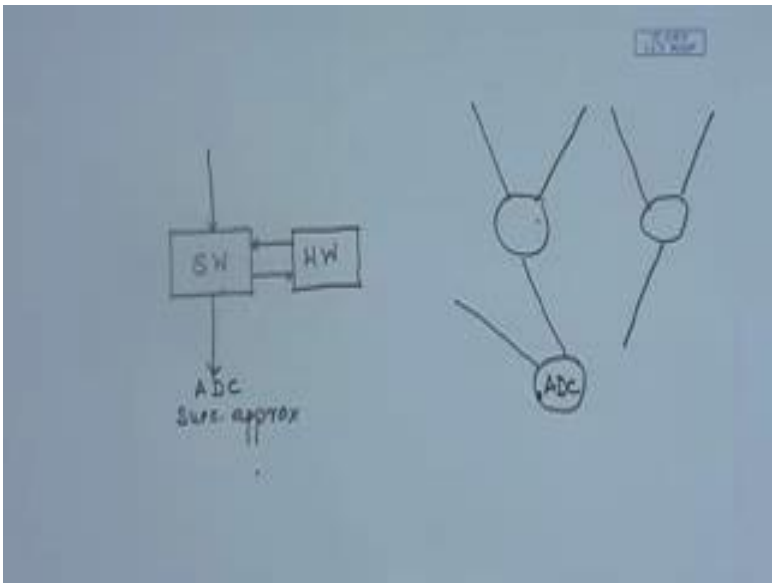


Now, when we do hardware software partitioning, we are we will have two things. One is the list of functional elements; list of functional elements means the operators, the operators themselves like max, min, multiplication, division, square root whatever that is one. Another one is list of implementations available. Now, what do I mean by available implementations. If you think of a laboratory scenario, whatever components are there in your cells, which you can use or in a integrate I mean high level scenario, advanced scenario all the different course that you have which are available, the way will cause definitions of whichever components that is those are my design library. So, this is actually giving me a design library.

Now, in the design library, I can have four types of implementations, I can have four types of implementation. What are the four types of implementations? One can be that it will be a pure software implementation. So, somewhere, it will take some input, we will

do some software computation, I will come out; and along with this implementations the costs and other parameters will be stored in the design library. For example delay, delay can be implemented using software. It can be purely a hardware implementation. For example, say I want to do the analog to digital conversion using a chip, I forgot the names of the chips, but it is something like very recollect ADC 801 or something like that 808, so there might be 808. So, so some chip is there and that chip is sufficient to do the analog to digital conversion. And this one this ADC hardware chip is doing it using a flash method.

(Refer Slide Time: 11:01)



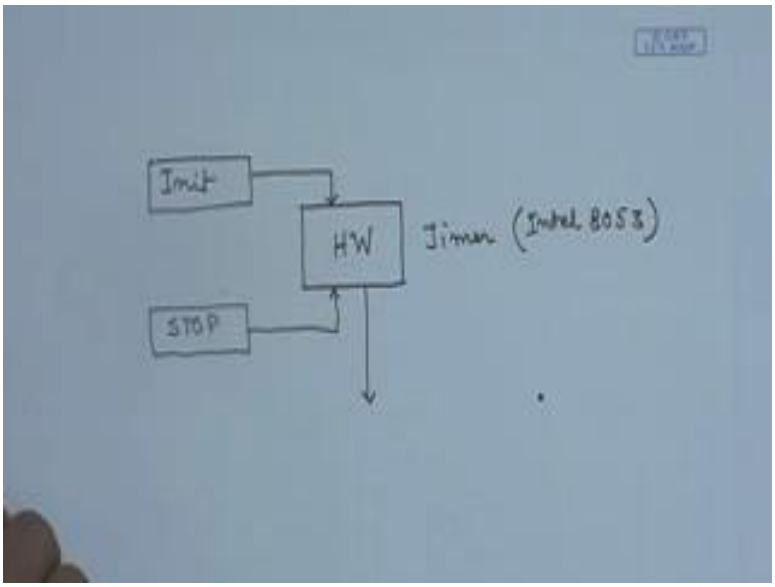
Besides I can have an implementation where I am doing a software implementation, but for that, I need some hardware support. For example, if I want to do ADC using successive approximation method. So, this one can be done using successive approximation, but for that ADC with successive approximation, but for that I need the support of the hardware I need a comparative hardware comparator and I also need DAC digital to analog converter.

So, as soon as I decide that this ADC part I will be doing by software. So, what did you have, you had in your task graph, there are a number of nodes coming in and here somewhere you have come to a point where you need to do ADC. And this ADC now is a decision point you want to weigh your options and want to see if I can do this ADC

using software maybe you are very well comfortably placed with regard to time constraint. So, you can opt for a low cost solution. So, you try to do this using this.

But as soon as you do that you will need some other hardware to be incorporated, therefore the implementation list that was there I was talking of the list of implementations available that the list of implementations available must take note of this facts, the implementation of ADC using software method is successive approximation additional hardware required of this. So, when you compute the cost you have to take into account that.

(Refer Slide Time: 13:34)

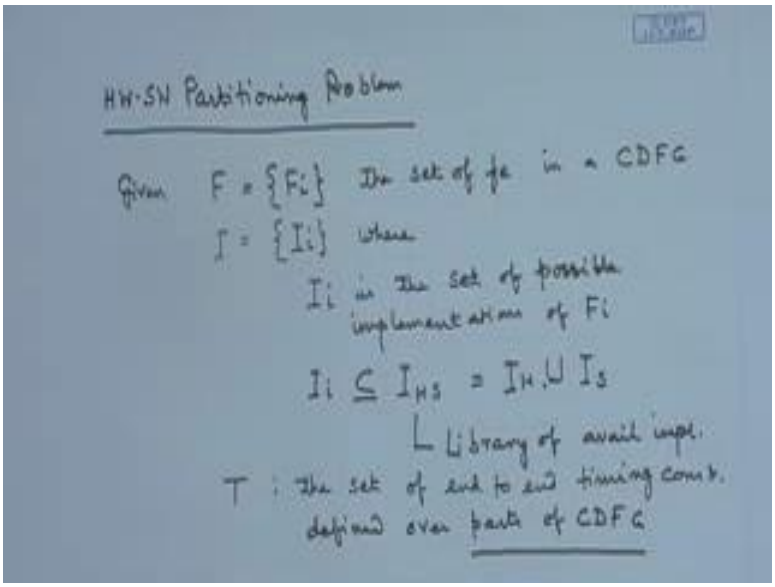


The other option that can be is suppose I want to implement something with hardware. Now, I want to this hardware is basically I want to implement a timer and for the timer say I select for example, Intel 8053, I hope that is the timer is my memory does betray me. So, it is a timer all right. And the timer being a programmable timer will require some software support. What sort of software support, there should be some initialization, this timer has to be initialized; and maybe this timer will have to be it is a counter say, it is acting as a counter. So, after some point maybe the software will give it a stop signal, and then this will give me the data. So, these are coming from the software.

Therefore, or say for example, a port or an interrupt controller that is the hardware and interrupt controller is the hardware, but you need a programming to do that. So, therefore, there will be some software over it or software cost for that hardware itself.

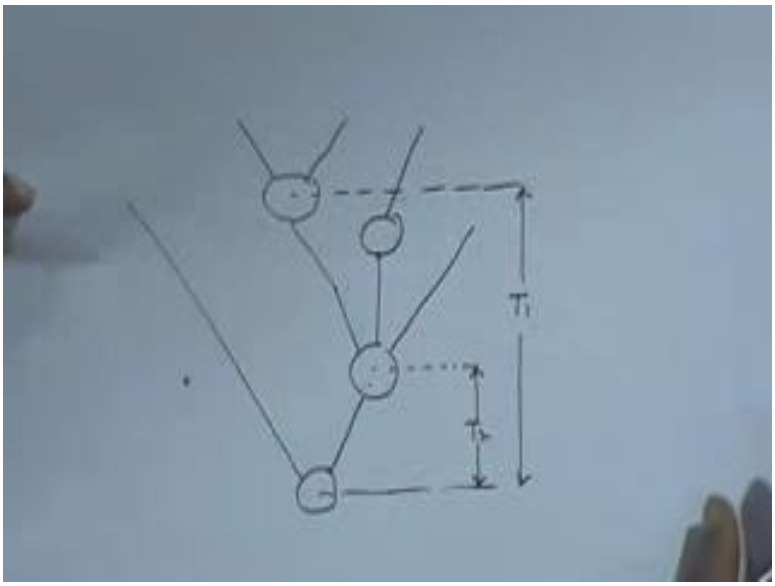
So, there are cases where it is not just hardware or just software, but often it is a combination. So, we will have again what I said we have a design library. Now, how can we define the hardware software partitioning, let us look at a formal way of looking at it.

(Refer Slide Time: 15:46)



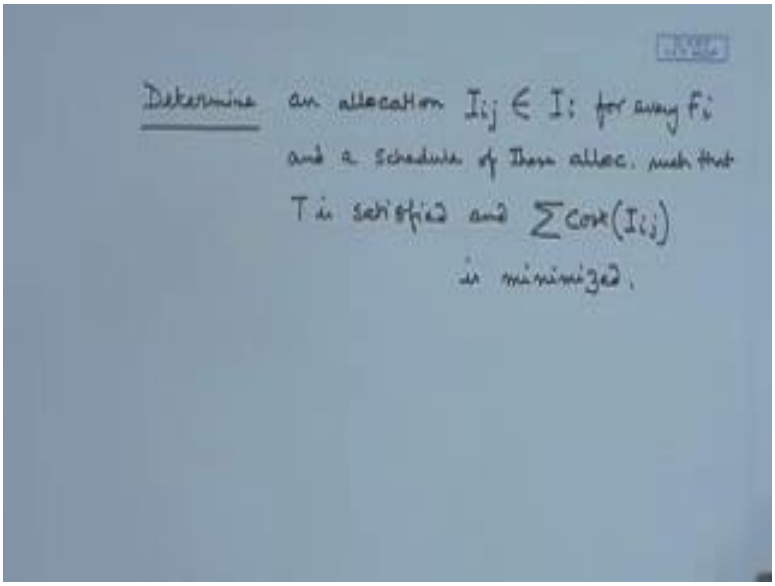
We can define the hardware-software partitioning problem as follows. Given say F, F is the set of functional elements f e means functional element in a CDFG, we know what is the controller into the program. And I is a set of I sub i, where I sub i is the set of possible implementations of a particular function of F i. And also I i the sub set of I HS which is the hardware implementations union the software implementations. Now, whenever I take a hardware implementation, I have to look at whether this hardware implementation requires some additional software and vice versa for the software. I H S is the library of available implementations. Now, given this definition is the definition clear? I H S is what, I H S is the set of all hardware implementation and I S is the set of all software implementations. And there is T which is the set of end-to-end timing constraints defines the word parts of CDFG.

(Refer Slide Time: 19:08)



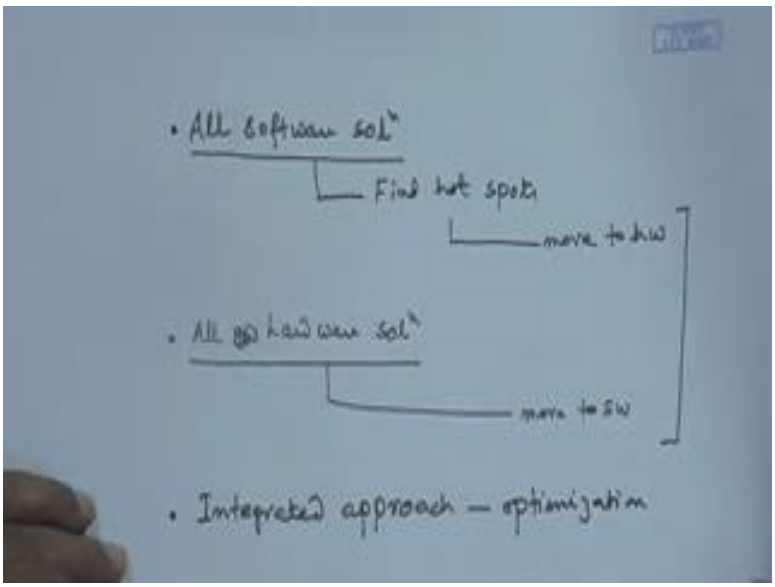
Now when I write parts of CDFG, what do I mean I mean that there may be cases that I have got a CDFG flow graph coming to b something like that and well this goes to something else and this one comes here and something else comes here and I go like this. Now, it can be that the timing constraint from this operation to this operation is T 1. I can also specify that after this is done the completion of this must be done in T 2, I can have multiple timing constraints attached that is why I am saying that it is part on the part of CDFG. So, given this let us look at once again even the set of functional elements and the set of implementations, our task hardware-software partitioning task is to determine.

(Refer Slide Time: 20:16)



And allocation I i j belonging to I i for every F i and our schedule of these allocations such that T is satisfied and cost of I i j is minimized, so that is my hardware-software partitioning problem. So, we have seen what is I i j, I i j is a set of implementation. So, you have to find an allocation that a particular F i is being given an allocation j and that is among the all the available implementation and that can be in hardware or that can be in software. So, I have to do a, I have to satisfy the time and I have to optimize the cost, so that is the hardware-software partitioning problem. Now, in computer science, you have come across such problems often, different varieties of optimization problem. Now, historically if we look at the hardware-software partitioning problem has been addressed in a simpler way.

(Refer Slide Time: 22:47)



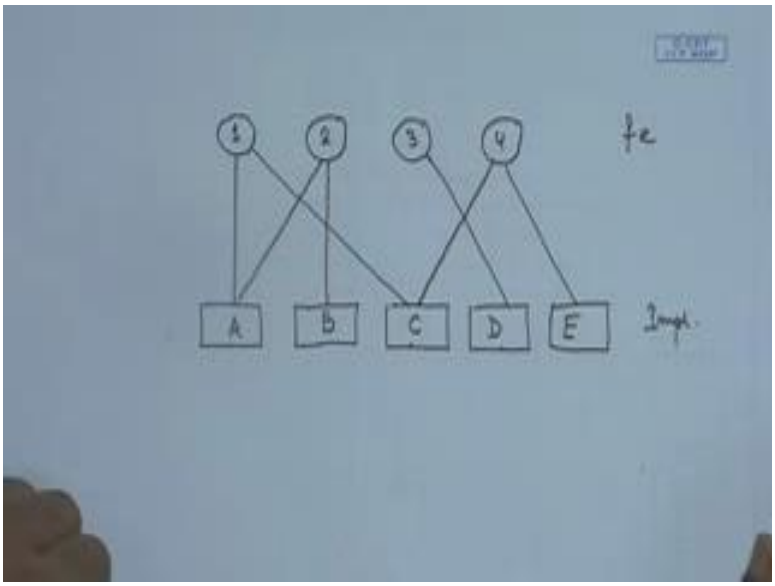
The way we have seen in the digital camera example first what was done is you try for all software solution. So, and then you find whether the constraints are satisfied; if not then look at the hotspots, find hotspots and moves them to hardware. Now, obviously here also you have to have a decision-making, so that is an identity way of doing it the way we did for digital camera example. The other option is obviously, a dual of this you try for all sorry all hardware solution. In that case, what will happen you will be able to meet the timing, but maybe you the cost is we are not actually taking the time to market so much in concentration in this discussion, but the cost will be high the power of all those things can be violated. Then we try to see which are the places, where I can soften the speed and then move to software in parts. So, these are different ways of doing it.

Student: (Refer Time: 24:40).

That is another factor, the question is whether flexibility should play a role, yes, that is coming as a non functional requirement at the very beginning and that can serve as an additional constraint. That I want to have the option of adding some features, therefore, at priori when I was saying that I am given the set of functional elements the constraints can also dictate that this, this, this parts task will have to be done in software or this, this, this things will have to be done in hardware. So, if that is given then that is already determined you have to play with the other things. So, in that case in our allocation in our definition this allocation some of the allocations are already done. So, I am left with another set for doing the a location that is a very good issue that especially in the mobile phone and all those we always look at the scope of adding features, so that we can be competitive in the market.

Now, so these are the two things. And the other way is some integrated approach. Now, for optimization, now there are several approaches of carrying out optimization. Now, this one the hardware software partitioning problem that we have defined the way we have defined maps very well to a well-known computer science problem, which is the consistent labeling problem. For those of you who have done that it may be familiar to you this will also become a consistent labeling problem; and the standard methods of consistent labeling may be attempted to be applied, I will show some difference with that consistent labeling problem. But before showing the difference you just keep in mind these definitions that F is the set of functional elements, I is the set of implementation.

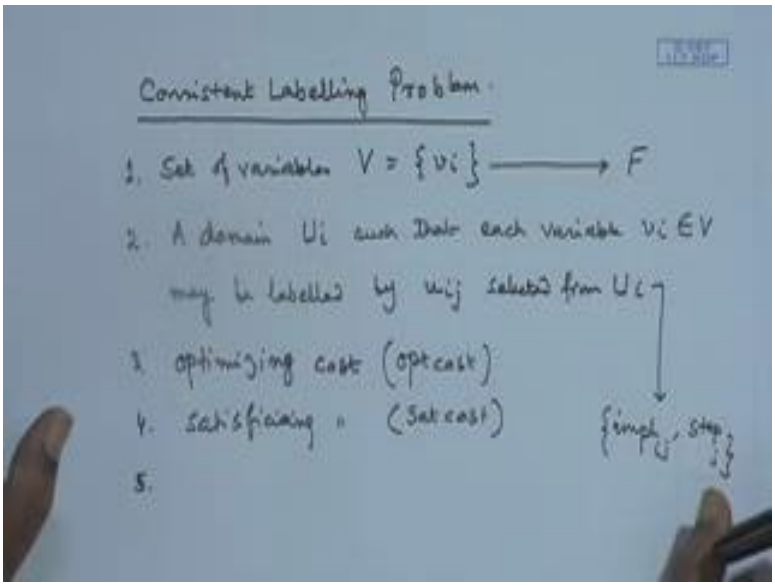
(Refer Slide Time: 27:12)



Just to give you a notion, I can try to explain it in the form of a bipartite graph also that I have got some functional elements which are being shown as circular nodes. And there are some implementations which are shown as rectangular blocks. Now, we have got the implementation list. So, these are the implementations, and these are the functional elements. Now, the implementation list may say that ok this functional element one can be done can be achieved with A. Similarly, C can also achieve one whether their hardware or software that part I am not coming to right now. 2 can be achieved by A or by B; 3 can be achieved by D only say; and 4 can be achieved by C and E.

Suppose, and I have to find the and along with that there are some constraints also given the cost constraint for each coloring or each mapping, I have to put I have to incur some costs and the cost constraints are there, each of these functional elements have got time and area cost. So, the task is to find the consistent labeling of the functional elements with the available implementations such that all the constraints are met that is consistent labeling.

(Refer Slide Time: 29:02)



So, let us define it the CLP, where in CLP we have got a set of variables V, a set of variables, a set of vertices. And 2 - a domain that is a set of values domain is a set of values U i such that each variable V i belonging to V may be labeled by U i j selected from U i. There I will select from the domain some value for mapping it to this variable. Two types of now when I do this along with that I add there will be some optimizing cost, we can which I can say opt cost and satisfying cost, I call it set cost and also there can be a cumulative cost like the time and all those. So, now if I have this and this, I can straight way find the mapping between the CLP problem and the hardware software partitioning problem.

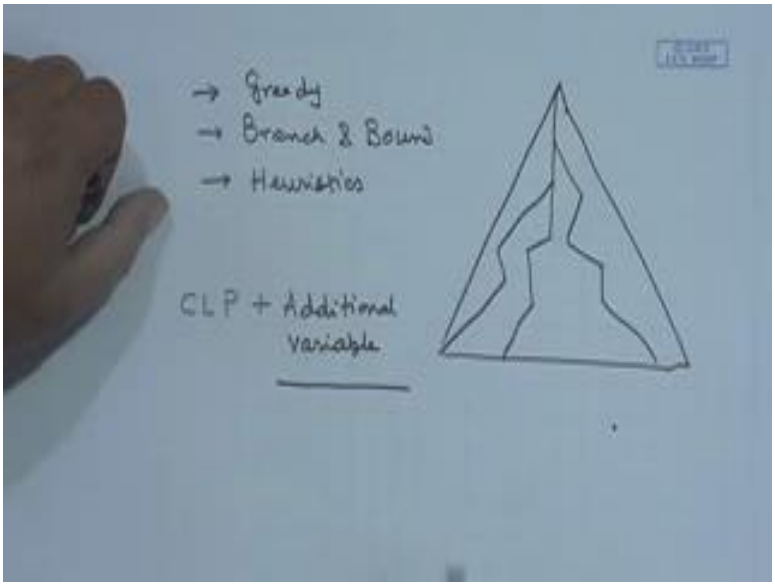
What is the mapping, what is the mapping, the mapping is this v it is just mapping to my set F the set of functional elements, the U i here is mapping to the set say what, i j which will have a particular implementation and a step right implementation i j and step j. What did I denote it with this here the ultimately we are determining the i j that is coming as U i j. The other costs we have already seen therefore, we can apply the methods of consistent labeling problem to solve the hardware-software partitioning problem, but there is a major difference from the consistent labeling problem and the hardware-software partitioning problem.

What is that, in the consistent labeling problem the set of variables are static, the set of variables are known a priori, but in our case in hardware-software partitioning, they are

not known at priori, they are known at priori. But sometimes what happens as soon as you select a particular implementation then just as we had done a few moments earlier, whenever I select an element that will be implemented using software. And for that we need some hardware then I have to add that hardware to my list of variables because for that hardware I will be needing some other implementation.

So, the set of variables are not constant. Now, I am not going into the details of the algorithms, but for so one thing is that as we have done in the digital camera example you can start with all software solution and move to hardware. Or you can do start with hardware and move to software or you can try to apply the consistent labeling problems type of solution. And the solutions are actually at done through different means for any of these, you can do it with dynamic programming, you can do with branch and bound actually and three heuristic search.

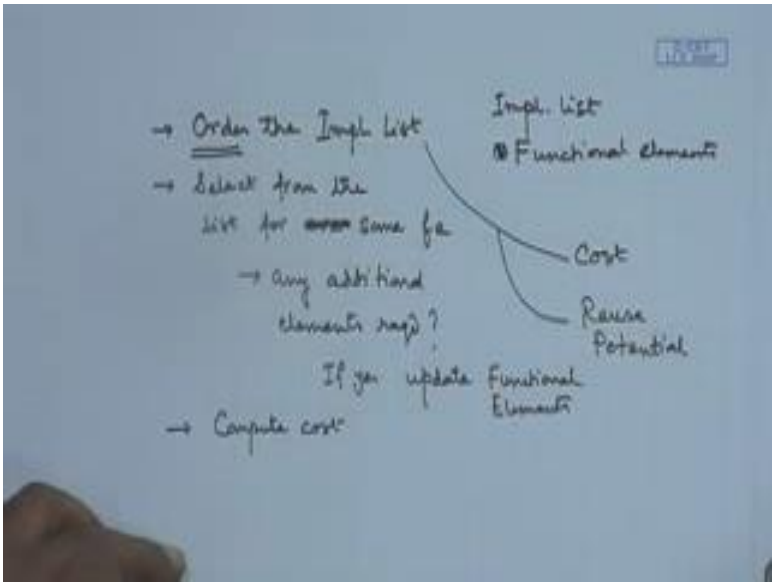
(Refer Slide Time: 34:26)



So, you can do brunch and bound heuristic search or greedy why am I leaving out greedy let us start with that first, which often. Say for example, the once that I was talking about that put every solution in hardware and then gradually moved to software that is also greedy, move everything to software that is also greedy. So, either you can do and as you get this. So, what will happen is you will have a search space with all possible paths that can be traversed in this search space, you understand that this is a free sort of thing where you can follow different solution parts right as you go on doing the allocation.

Now, this you can do with a greedy method, you can do with branch and bound or you can apply suitable heuristics for doing that that is one way of doing. But while doing this, we have got the CLP issue plus we have got the additional variables scenario, you have to add the variables and accordingly. So, what will happen is the variable list that you have, you will have to continually update at every step of allocation. So, what will the algorithm being look will look like. So, I will just say in brief what the algorithm may possibly look like.

(Refer Slide Time: 36:19)



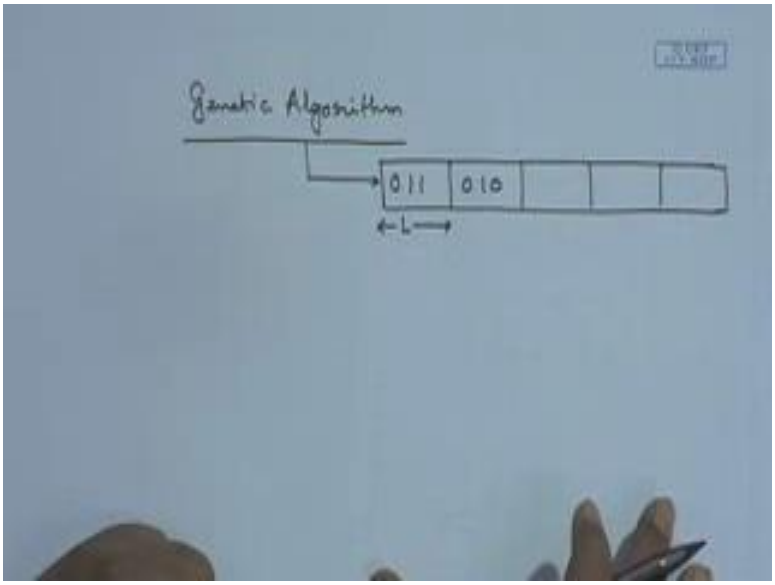
It will be you will have some implementation list, and you have you have got the functional elements. So, maybe heuristically, you will have to first order the implementation list may be based on their cost maybe they are reuse potential. As you do it this is something I forgot to mention, we look at the cost and also there is another issue of looking at the reuse potential. For example, if I select an ALU, it will be expensive maybe, but it can be reused for so many functional elements to cover so many functional elements, so it is cost should be compensated for by its reuse potential.

So, accordingly, we order the implementation list and then select from the list on the list for every F i for some f e some functional element and see what first we check any additional elements required? If so, then we will have to add to the implementation list right; if yes update functional sorry update functional elements, because we will need more functional elements to do. And then compute cost you go on computing the cost till

now and check whether that cost has violated the ultimate cause. Now and this will be done absolutely heuristically.

And you will have to see, so usually what a problem that occurs in this sort of solution for any CLP solution or for many of the I such solutions is that the search space is very large. So, you have to do some look ahead, and do some pruning just like branch and bound, you see that if I make this selection, if I follow this particular part then I am already exceeding my cost. So, there is no point proceeding in that path in any further and so I will select the other parts. In that way different algorithms can be applied, once we formulate the hardware software partitioning problem in a suitable way as we have done today.

(Refer Slide Time: 39:57)



Another option I will just mention it and another one that I will be just mentioning is it genetic algorithm, it has also been applied very successfully for hardware software partitioning. What is done in that case is that whereas you know genetic algorithm will have different chromosome. So, I will have a list of chromosome and each chromosome will consist of a number of genes. So, whatever this is my list of all possible functional elements, and I will have for each possible functional element, I will have a gene code of particular length L say all right, and I associate some particular implementation with this. Some particular implementation, say for example, 001 or 010 in that way we form the entire big stream.

Now, this is one solution there will be another solution in this way. So, there will be a set of population. And then we take, we goes to the crossover functions by which we take one such chromosome another such chromosome, and I have a crossover and get a new solution. For each of them we try to find out the fitness function, we will try to see if we can look at it a little more detail.