Embedded Systems-Design Verification and Test

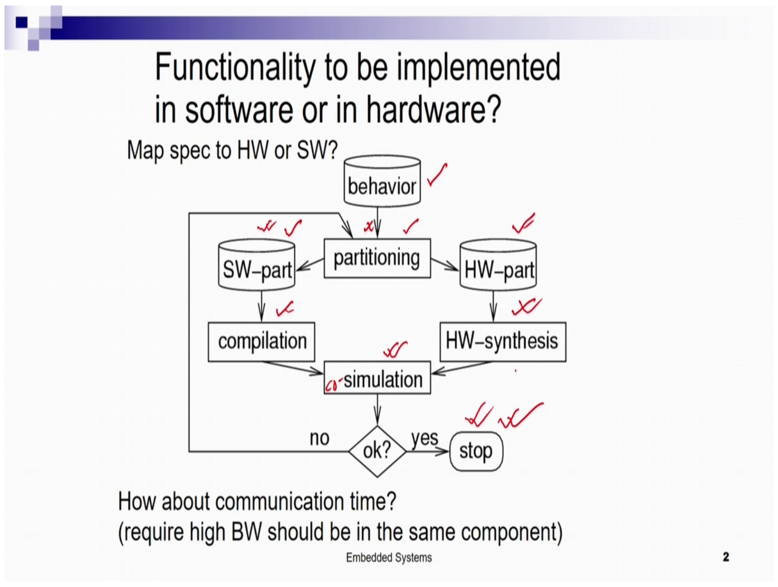
Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture - 05
Hardware / Software Partitioning

Hello. In the last few lectures we saw how embedded systems are modelled. We saw various forms of modelling, and we also saw a procedure by which modelling also allows automatic synthesis of the control software that we want to design. However, when the size of the embedded systems becomes very large and complex, the size of the models become very large. Such accurate formal models from which synthesis can be done is very difficult to obtain.

So, we have to resort to more non formal strategies not absolutely as formal as we took at last day. So, once a modelling is done, the next step that we embark on in embedded systems design is Hardware Software Partitioning.

(Refer Slide Time: 01:27)



Functionality to be implemented in software or in hardware?

Map spec to HW or SW?

How about communication time?
(require high BW should be in the same component)

Embedded Systems 2

So, firstly, we are given the design to behaviour; that means, we have the models in terms of say concurrent and hierarchical FSMS, HC, FSMS, and then some nodes of that HCFSM will be designed in software and some in hardware. By software we mean that,

that code will have a C code for example. We will write the C code corresponding to the node, and we will execute it on a general purpose processor. Now execution on a general purpose processor has several advantages.

For example, it allows a lot of flexibility, in terms of easy updates of the software for example, for the functionality. For example, if you have say a power window system software, or a suspension software in a car, a suspension system, a suspension control system is implemented as a software as a C code on a particular general purpose processor.

Now, if you want to update the suspension software, it is very easy to do, just change the C code and drive it on to the platform of the car through the network. And the update is done. So, therefore, it is just the updation of the C code that was required to change the suspension control system that we had. So, this was possible because we implemented the suspensions software. Suspension code or functionality in software and executed it on a general purpose processor.

Software execution software based execution to the C code also allows multiple executions to seamlessly co execute dynamically together without any hassle. So, therefore, software based execution has a lot of flexibility; however, hardware based implementation also has it is own advantages. So, what is a hardware implementation? So, that C code will then go through a set of hardware design steps, and we will design a circuit from a digital circuit and we will design or do a technology mapping and map the functionality on a silicon wafer, and we essentially get a single purpose processor for this functionality.

So, if we have a completely customized hardware implementation of a functionality, then what we have is we build essentially build a digital circuit that implements that functionality. So, it is only that functionality that is obtained, we do not have to go through the process of say, the instruction set of a software based execution essentially, what happens? You have the C code; you compile it to the machine code. And then you bring from the memory the instructions one after another, and the data one after another, execute instruction after instruction based on the micro instruction that the software platform or the software processor, the general purpose processor supports. And then send back the results to memory.

In a hardware implementation, on the other hand you have a de dedicated digital circuit only to perform that function. You do not have to go through you do not have to bring in machine; you do not have separate machine instructions for a software for a gen like a general purpose processor. Say, you just take the data, and the data is transformed through a set of digital circuits and the output is obtained. So, the execution time might be much more optimized as in a software with respect to a software implementation. So, the performance of a hardware implementation could be much more than a software based implementation.

So, if you require say very low latencies within which a function must be performed within a very short time, within say 15 milliseconds or some very small time within which a function must be performed. And that performance cannot be given by a software implementation, when we are executing interneuron on a general purpose processor. It might need to be executed on a hardware platform for the sake of this performance.

However, hardware implementation on the other hand is rigid it is not flexible. Once you build a digital circuit, you cannot update it, unless you throw away the chip or and replace it by another chip with the updated functionality, ok. So, therefore, hardware implementations although may be very optimized with respect to performance can be rigid with respect to implementation could be rigid with respect to a it is a co execution of multiple components together, could be rigid with respect to updation.

There could also be a trade-off between the 2 for example, say an FPGA based implementation, you want to do in an FPGS based implementation. Essentially, you have a c of gates, with interconnection between them. And you have to program those interconnections to provide a given logic. So, and it can also be reconfigured, or it can also be updated. Those interconnections can be reconfigured to get a different functionality; however, it is not as seamless as fast as the software based implementation. Seamless in the sense of co execution of multiple dynamic tasks together on the FPGA is not as easy as doing that in software.

And, the performance although; it is not that as and performance is also not as good as hardware, but it is better than a software implementation many a times; so therefore, FPGS could be a trade-off of both worlds. It is sort of flexible, it is not as rigid as hardware as it allows reconfiguration, but it is not as performance efficient as a completely customized hardware implementation as well.

So, therefore, what we are saying is that given a behaviour or model in terms of an HDFSM say we have to take each load of that model, and implemented it on software or hardware or FPGS or wherever so that our overall design constraints are met. And what would be the overall design constraints for the entire system? Could be the latency, the end to end latency of the functionality application that you that we want to do, that we want to have say the power consumption, the power consumption the area etcetera.

The cost the cost is again another very important thing. Software based implementation could be much more much less costly than a hardware based implementation. It is just in a software based implementation again you take the C code, and you take it and push it into a general purpose processor. For a hardware based implementation the same functionality has to be synthesized and then fabricated. And then you have to obtain a chip for it, and then you get that functionality as a hardware. So, it is both costly as well as time to market the advantages performance. Maybe size the size of the circuit implementing that functionality will also be much lower than that in for it is software based counterpart.

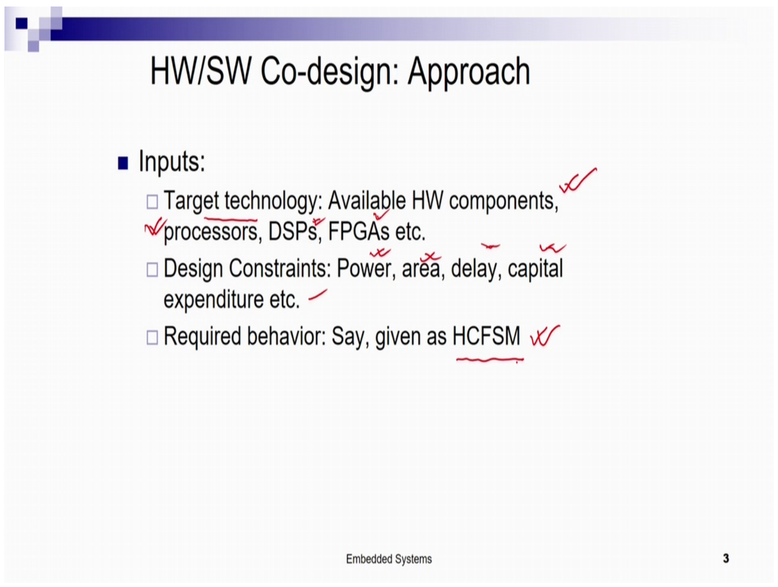
So, there are various trade-offs, and therefore, we want maybe we want to implement. Therefore, some parts of our embedded system in software and some parts in hardware. Maybe we want to implement. For example, if software based implementation is always much more flexible, and hardware based implementation is always much more performance efficient. But costly then we would want may want to implement as much as of the of the embedded system as possible in software, such that my constraints on performance are met, and such that my constraints on performance are met, because software implementation is both flexible and less costly as we just said.

Now, how do you obtain this hardware software partitioning? You first get the behaviour, and then that behaviour we do a partitioning of that behaviour into the software part and the hardware part. For the software part we compile say the C program, and then for the hardware part we do the hardware synthesis and estimate. And then we do a simulation to estimate what does it meet the required performance. If yes, we are satisfied with the hardware software partitioning we stop.

If you are not satisfied after simulation we go back and do a repartitioning, and see other we put other components essentially we do a design space exploration, into various types of; so, a component can be first we will try in software. Then we will try in hardware, different components with different cost variations, right. We will take another hardware software partition again implement, again compile it. Again to a hardware synthesis may be a high level synthesis we do to get an estimation of what will be the hardware cost, area etcetera and performance. And then do a co-simulation of the software and hardware and then see if it is ok. Once we have once all our objectives are met, and the constraints are satisfied we stop, ok.

So, how does this hardware software code design approach proceed?

(Refer Slide Time: 11:59)



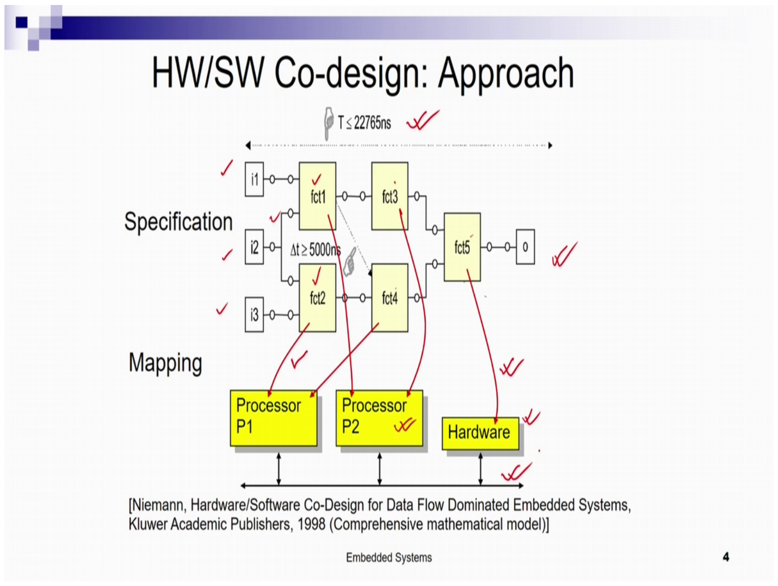
HW/SW Co-design: Approach

- Inputs:
  - Target technology: Available HW components, processors, DSPs, FPGAs etc.
  - Design Constraints: Power, area, delay, capital expenditure etc.
  - Required behavior: Say, given as HCFSM

Embedded Systems 3

Firstly, we have to look at the inputs. So, what are the inputs? What is the target technology available to me? What are the available hardware components? Do I have is it possible for me to design completely customized hardware or we have to make go for a standard cell implementation, or we can have FPGAs, etcetera? What kind of general purpose processors do we have? ARM, Intel what kind of general purpose processors do we have? DSPs, digital signal processor, FPGAs, etcetera. So, what are the different types of components that I have at my disposal. That has to be known, what are the design constraints that I have on power, area, delay, capital, expenditure, etcetera. And we also have the required behaviour on which we will do the partitioning in the form of say an HCFSM, ok.

(Refer Slide Time: 11:50)

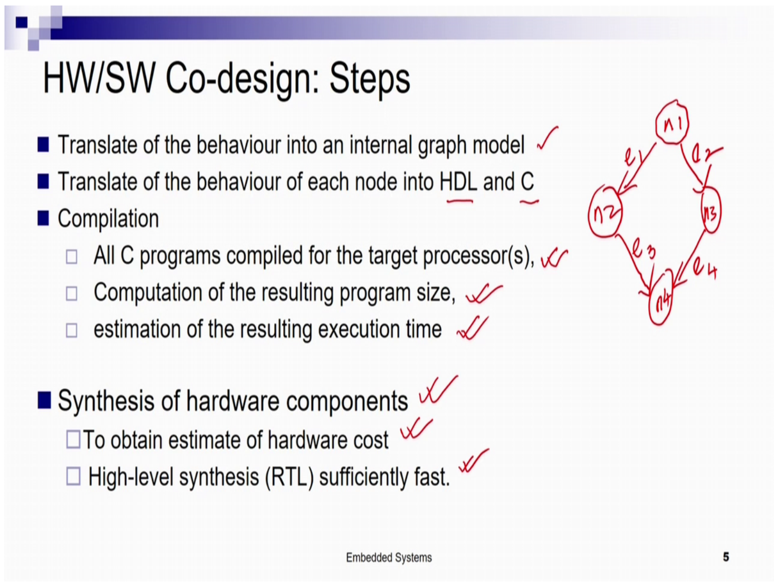


[Niemann, Hardware/Software Co-Design for Data Flow Dominated Embedded Systems, Kluwer Academic Publishers, 1998 (Comprehensive mathematical model)]

Taking a small example of a data flow dominated graph. This one is a data flow graph, where let us say you have 3 inputs i1, i2 and i3. This is the specification. And you have these functionalities fct1, fc2, 2, 3, 4 and 5. And these are the components, and we say that for this functionality the end to end latency bound that should be met is 2 2 7 6 5 nanoseconds; and also between the end of functionality one and the beginning of functionality 4. Let us say there must be a time gap bit of 5000 nanoseconds. Given these constraints, let us say we will have to decide what to do.

For example, here possibly we have decided that we will implement functionality 1 and functionality 3 on processor 2. So, it will co execute functionality one and functionality 3 will co execute as a software code on processor 2. Functionality 2, on the other hand will execute on processor 1, and functionality for 5 is possibly extremely performance hungry, and we need very efficient performance for it and we will implement it in hardware. And all these processors talk between themselves over buses, ok, this is an example.

(Refer Slide Time: 13:08)



HW/SW Co-design: Steps

- Translate of the behaviour into an internal graph model ✓
- Translate of the behaviour of each node into HDL and C
- Compilation
  - All C programs compiled for the target processor(s), ✓
  - Computation of the resulting program size, ✓
  - estimation of the resulting execution time ✓
- Synthesis of hardware components ✓
  - To obtain estimate of hardware cost ✓
  - High-level synthesis (RTL) sufficiently fast. ✓

Embedded Systems 5

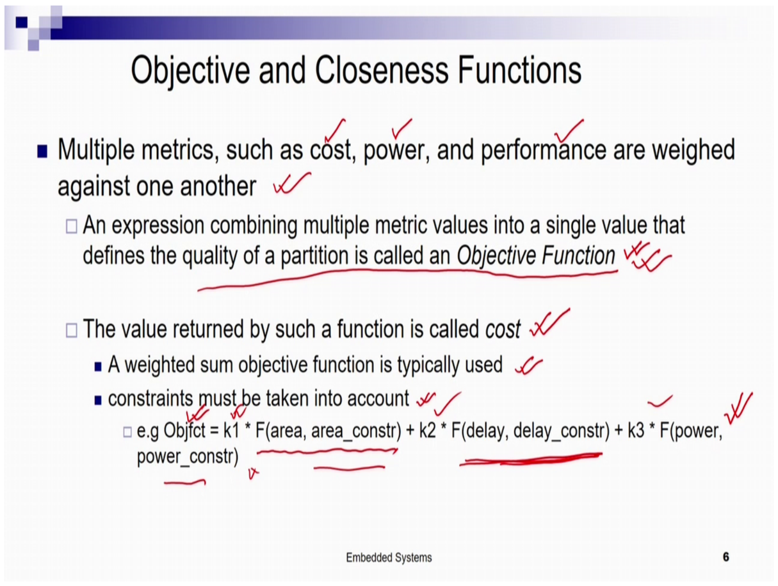
Therefore, what are the steps do we have? The overall steps, first we translate the behaviour into an internal graph model, could be a task track task graph model, which says that for example, these are my nodes, these are my nodes, say and these are the edges with the frequencies, ok. So, the edges will tell me how many times say suppose these are my end nodes representing sequential component programs. These are my sequential component programs, and these are my edges say. And these represent say how many times does n 3 needs to call n 4, if these are modules.

So, we have a task graph like representation like this. So, we translate the behaviour into an internal graph model, and then translate that behaviour into an HDL and C compilation. HDL is a hardware description language like Verilog and VHDL and C compilation. So, what we will do? We will have both for each functionality.

For example, we have to have the option of either implementing it in software or implementing it in hardware. So, we have to estimate, what is the cost of implementing a particular functionality, a particular node in hardware and software. So, to estimate the hardware cost of an implementation we have to implement that node in HDL say Verilog or VHDL. To estimate the cost that this node will have for a software implementation, we have to implement that particular functionality as a C code. Then we compile all C programs on the target processors available. We compute the resultant program size estimate the resulting execution time etcetera. So, various cost parameters we obtain, and for the same functionalities we also synthesize the hardware components. We obtain estimate of the hardware cost and we do a high level synthesis, and then which is sufficiently fast we look at how high level synthesis procedure happen, how the high level synthesis works, how do we get an architectural hardware architectural architecture corresponding to a function in the next subsequent lectures.

So, we have to do a high level synthesis to obtain a fairly accurate estimate of the hardware costs.

(Refer Slide Time: 15:37)



Objective and Closeness Functions

- Multiple metrics, such as cost, power, and performance are weighed against one another
  - An expression combining multiple metric values into a single value that defines the quality of a partition is called an *Objective Function*
  - The value returned by such a function is called *cost*
    - A weighted sum objective function is typically used
    - constraints must be taken into account
      - e.g Objfct = k1 * F(area, area_constr) + k2 * F(delay, delay_constr) + k3 * F(power, power_constr)

Embedded Systems 6

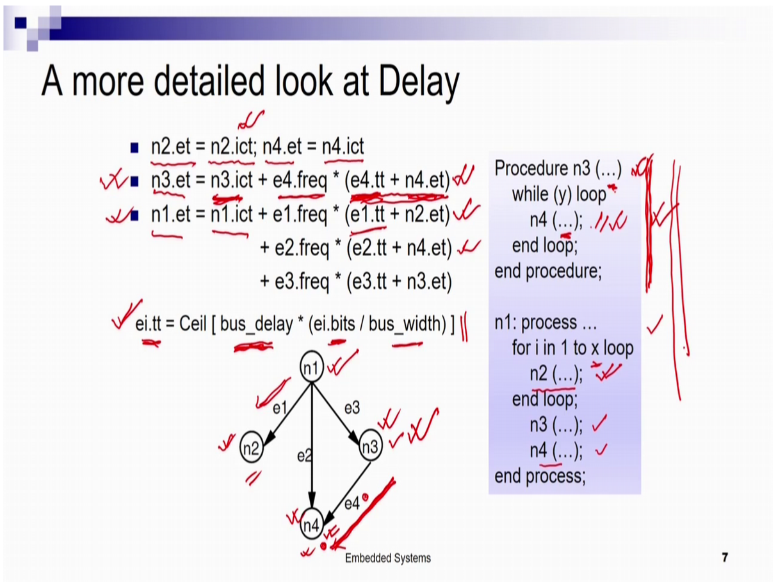
So for each component, we now have the software cost as well as the hardware implementation cost. And then we will have various objective and closeness functions depending on what we want to achieve for the entire functionality. We will have multiple matrices, as we just talked about for example, cost power performance. And they has to be weighted against one another into a single objective function. An expression combining multiple metric values into a single value that defines the quality of a partition is called the objective function.

So, we will have a partition of into hardware and software. And then how do we partition into hardware and software? So, therefore, some of my modules in my task graph will be implemented in software, and some of my modules will be implemented in hardware, and this is essentially a partition. For this partition, we will have to find out how good that partition is. That will be determined by an objective function. And that objective function will provide me a single value weighing different multiple matrices on which my partition depends for example, cost power performance etcetera. So, the value returned by such a function will be called the cost.

The weighted sum objective function is used. For example, all constraints have to be taken into account how for example, the objective function could be let us say k 1, which is a weight k 1 into area, plus k 2 into delay, plus k 3 into power. Suppose what is the function f area comma area constraint tells me? So, it will be returned me of value which with respect to the area taken by the system against an area constraint that I have. F delay comma delay constraint. Similarly will tell me what it will return your value with respect to the overall latency of let us say, this embedded system with respect to a delay constraint that I have, likewise.

So, we will have a single objective function that will tell me the goodness of a certain hardware software partition at any given time.

(Refer Slide Time: 17:46)



A more detailed look at Delay

- n2.et = n2.ict; n4.et = n4.ict
- n3.et = n3.ict + e4.freq * (e4.tt + n4.et)
- n1.et = n1.ict + e1.freq * (e1.tt + n2.et)
  + e2.freq * (e2.tt + n4.et)
  + e3.freq * (e3.tt + n3.et)

ei.tt = Ceil [bus_delay * (ei.bits / bus_width)]

```
Procedure n3 (...)
  while (y) loop
    n4 (...);
  end loop;
end procedure;

n1: process ...
  for i in 1 to x loop
    n2 (...);
  end loop;
    n3 (...);
    n4 (...);
end process;
```

n1
n2
n3
n4
e1
e2
e3
e4

Embedded Systems 7

Now, to look a bit more deeply as to how such parameters of performance power etcetera could be calculated, we look at deeply at oh at one important parameter the execution time. One of the important objectives of a system could be to minimize execution time. Of course, within a given cost, as I said hardware based implementation could have a certain could be better in terms of performance. That means, lowered in terms of overall execution time, but they could be much higher in terms of the cost it that is incurred, but estimation of the execution time is required, ok.

So, we now means one of the important goodness parameters of a partition would be that what is the execution time of the overall application when such a hardware software partition of the modules are obtained, is obtained. For example, let us say we have a procedure like this. In this procedure I have a process n 1, that n 1 calls within a loop, another procedure n 2 x times. And then after the loop it calls n 3 and n 4. Procedure n 3 which is called by n 1 again goes like this. It calls within a loop until something is true and the loop n 4 and then it ends, ok. So, therefore, this is the overall thing we want to implement.

Now if you want to estimate the execution time we will have something like this. N 2 we see has does not call anybody; n 2 does not call anybody. So, execution time of n 2 is given by the internal computation time ICT of n 2, and nothing else. Similarly, n 4 does not call anybody. It is a monolithic module program, sequential program. So, the execution time of n 4 is only the internal computation time of n 4. For n 3 we have this procedure, we have this procedure.

So, what is the computation the total execution time of n 3? It is given by the internal computation time of n 3 plus what n 3 calls n 4, n 3 calls n 4 over the edge e 4. So, let us say the number of times n 3 calls n 4 is given by e 4 dot freq, ok. And let us say the transmission time for one such data transfer of the data from n 3 to n 4 is e 4 dot t t. And the execution time of n 4 is already we obtained is n 4 dot et which is equals to n for dot ict.

Why is this? The overall execution time of n 3, because the overall execution time of n 3 is equal to the internal computation time of n 3, plus the time that is spent executing n 4, plus and the time that is spent sending data input data from n 3 to n 4. So, n 3 does something, and then it passes something to at the arguments of n 4. So, n 3 passes arguments to n 4 for passing these arguments these are data, this is data, and this when it passes it essentially it is a data transfer from n 3 to n 4. And each time n 4 exists n 4 executes this data is passed from n 3 to and 4, ok.

Now, what is the transmission time for the data? What is the approximate transmission time for the data to pass from n 3 2 n 4? It is given by this. Transmission time of any edge is given by the bus delay for this edge. So, I will implement this on a bus. So, given by the bus delay into number of bits divided by bus width. For example, let us say I need to send 64 bits and my bus width is 16. So, number of transfers that I will require is 4, 64 by 16. So, bus width is 16 I have 64 bits to transfer.

So, I will require 4 transfers. And in etcetera, each such transfer I require a delay which is equal to the bus delay. And this is the total transmission time for one data transfer from n n 3 to n 4. So, once I call n 4 I will have to incur a delay of ei in e 4 dot tt. And each time I call I execute n 4. So, each time I call n 4 I incur the execution time of n 4 plus the time to transmit data from n 3 to n 4.

This is given by e 4 dot tt plus n 4 dot et. And how many times do I have to in does n 3 have to incur this cost which is, that is equals to the number of times the freq of e 4. So, e 4 dot freq into e 4 dot tt plus n 4 dot et, ok. Plus, in addition to this it has it is own internal computation time ict. So, the overall computation time it is execution time of n 3 is given by n 3 dot ict plus e 4 dot freq into e 4 dot tt plus n 4 dot et.

Now, what is the overall computation time of n 1? It is again given by this one. So, we see that n 4 executes itself, and it calls n 2 it calls n 4 and it calls n 3. So, then like we just calculated for n 3, what will be it is total execution time? N 1 dot et equals to n 1 dot ict, plus e1 dot freq into e1 dot tt transmission time plus n 2 dot et for this one, plus e 2 dot freq into e 2 dot tt, plus n 4 dot et, for this one, and e 3 dot freq into e 3 dot tt plus n 3 dot et for this one. So, this is the total time of n 1 dot et.

Now, you it is we should appreciate that, let us say in n 1 is implemented in hardware, and n 2 is implemented in software. N 1 is implemented in hardware and n 2 is implemented in software. So now, this happens you have them in separate chips. And therefore, in separate chips and therefore, what happens is that, the transmission times could be very huge. So, it could be so that you design one chip, and design all the hardware functionalities into the same chip.

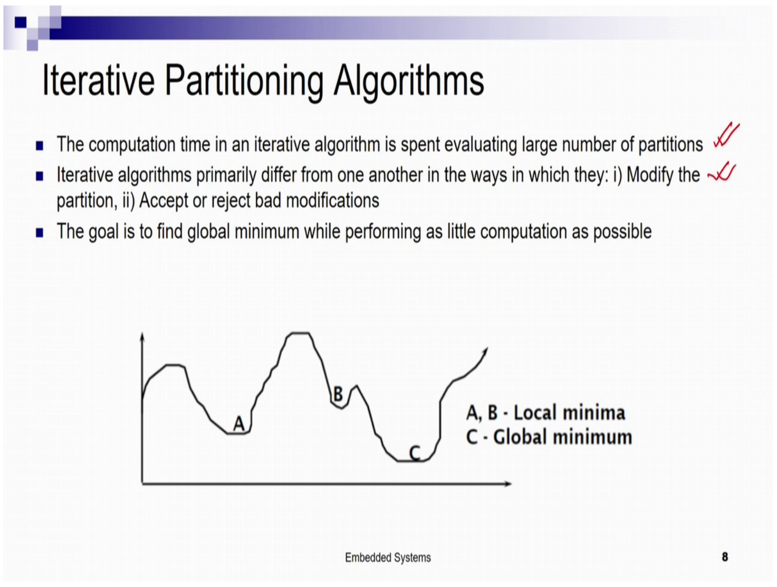
So, therefore, a hardware to hardware trunk communication cost a transmission cost could be very low. Software to software transmission cost could also be low, because number of programs executing on the same general purpose processor let us say- however, hardware to software transmission cost could be high, because these are on 2 separate processors, ok. And we also have different amounts of ict, ict that is the internal computation time would also be different for hardware implementation and software implementation.

Now, let us say if always software based implementation is lower software plays implementation has lower performance than hardware based implementation, it could be that we want to implement everything in hardware. Because, the hardware based implementation will give you the lowest execution time. If software based implementation we are supposing assuming that software based implementation will have higher execution times always than hardware based implementation. If that is so, then we should implement everything in hardware. However, cost of the hardware implementation as we told is much higher than that of soft.

So, estimation of this execution time is not everything. As we saw in the previous slide, we have to have a weighted function the overall cost is a weighted function of multiple matrices delay or execution time or latency is just one of them. We need to considered other things not everything is considered in this objective function. For example, one important objective function could be the expenditure that we need to bear for this design; and also the time to market for example. So, all these have to be borne in mind before saying that this is my before, finally, determining the cost of a certain partition.

However, this slide tells you how you can estimate one important metric the execution time of a given partition. Now there are different types of algorithms that are used for hardware software partitioning. One important class of algorithms is iterative partitioning algorithms.

(Refer Slide Time: 27:49)



Iterative Partitioning Algorithms

- The computation time in an iterative algorithm is spent evaluating large number of partitions
- Iterative algorithms primarily differ from one another in the ways in which they: i) Modify the partition, ii) Accept or reject bad modifications
- The goal is to find global minimum while performing as little computation as possible

Embedded Systems 8

Now, many algorithms a are very efficient in terms of the performance and some are less, and what we have done for this course is that, we have tried to take a few heuristic algorithms and we looked at 2 we will have a look at 2 heuristic algorithms here. We will not look at the optimal algorithms here we will only have an overview. One important advantage of the heuristic algorithms we will see is, that it almost always gives us good results, may not be the best result, but is very quick. Now why it is quickness essential, because this hardware software partitioning is essentially a design space exploration process; meaning that, we will have lot of options for hardware and software, we have to test different alternatives, different nodes in software and hardware before; Finally, saying that yes this is the best partition. So, we have to quickly be able to evaluate the cost of the partition and go and proceed and get an answer. And we have to go on doing that. So, therefore, the there has to be a compiler that tells me how good a partition it and give us give us quickly a partition so that we can do the next steps.

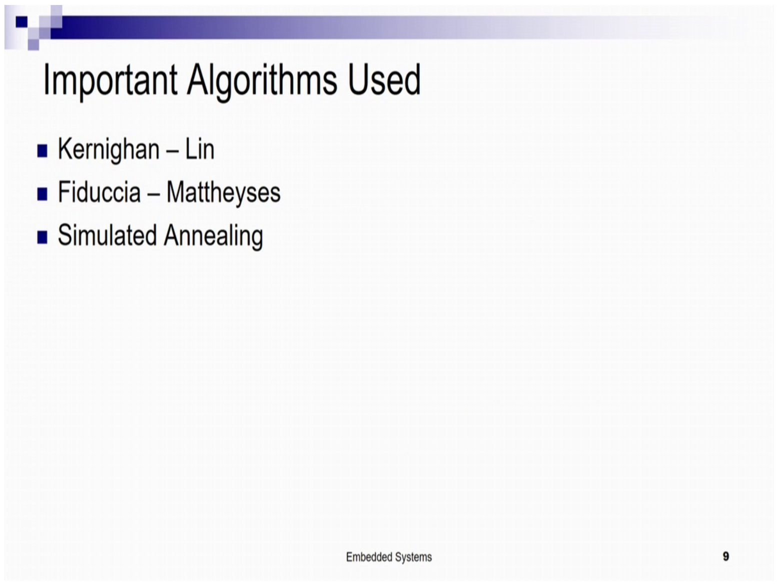
Now, if the optimal algorithms will take a long time to decide as to how to do this. We look at other places will look at optimal algorithms here we will look at 2 heuristic algorithms in this class. So, we will look at 2 iterative partitioning algorithms. How do they proceed? The computation time in an iterative algorithm is spent evaluating a large number of partitions. So, the main the place at for which the computation time is incurred is in an iterative algorithm is spent in evaluating a large number. So, we will have a large number of partitions and we have to see which partition is good among them. The iterative algorithm is primarily differ from one another in way in the ways in which they modify partition.

So, suppose how do they part up update partitions. How? This is a certain partition in terms of a certain modules in software and a certain modules in hardware, and then it will take one step and modify that partition. What is the methodology they take in modifying the partition? So, some of them are in this updation process you it may shift a few models from software to hardware and a few models from hardware to software say.

So, therefore, what is the mechanism that they are using to modify the partition? And do they accept bad modifications? Or how do they accept except bad modifications or reject bad modifications? Why is acceptance of bad modifications at all necessary? Because locally bad modifications at a certain time can later result in very good partitions; so a certain transfer from software to hardware certain modules, it may give you a certain negative impact on the partition cost, but due to that due to that transfer from software to hardware; let us say, other subsequent transfers could result in huge improvements in cost. So, it is sometimes essential to also accept bad modifications.

Now, the overall goal of the iterative partitioning algorithms is to find a global minimum while performing as little computation as possible.

(Refer Slide Time: 31:24)



Important Algorithms Used

- Kernighan – Lin
- Fiduccia – Mattheyses
- Simulated Annealing

Embedded Systems 9

And as we said, there are a few important iterative partitioning algorithm that we will see. We will see the first 2 among them, will not see simulated annealing, but the important ones are Kernighan-Lin, Fiduccia-Mattheyses and Simulated Annealing.