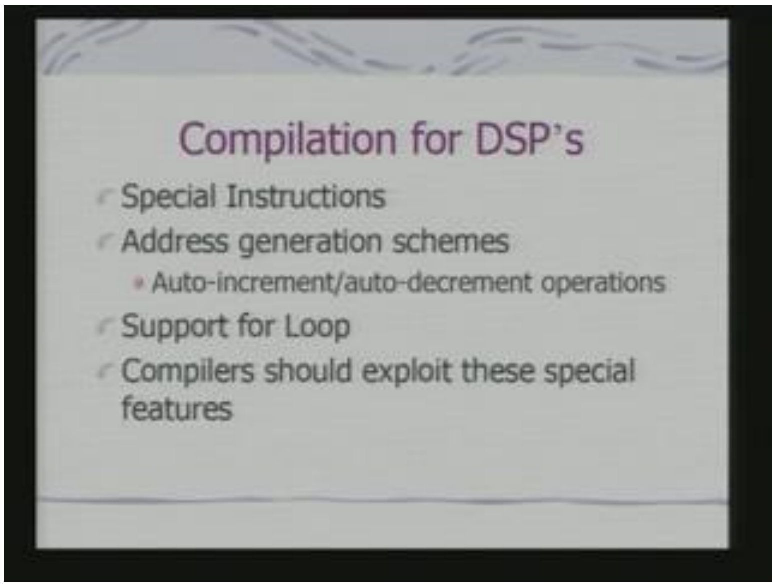
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture-35
Developing Embedded Systems

Today we shall discuss different techniques and methodologies to be followed for developing embedded systems. We have looked at the design process; we have also looked at some of the features that is required in compilers to generate efficient code for embedded systems. Today, we shall continue with the discussion in terms of architecture specific features of compilers and then the subsequent steps and methods to be followed for development of embedded systems.

So, in the last class we have discussed retarget able compilers we had also referred to the cases, where we need to have more architecture specific features for compilers particularly say for digital signal processors, VLIW processors as well as for multimedia data processing.

(Refer Slide Time: 02:11)



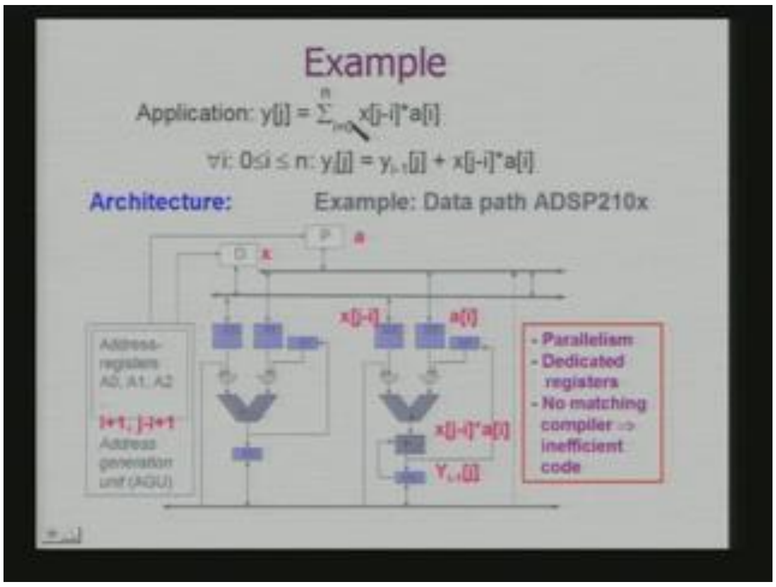
Compilation for DSP's

- Special Instructions
- Address generation schemes
  - Auto-increment/auto-decrement operations
- Support for Loop
- Compilers should exploit these special features

In today's class we shall see how compilers explore those features the first thing is compilation for DSP's. In fact, digital signal processors have got various special features which distinguish them from a general purpose micro controller, when a compiler is being designed to generate code for such digital signal processors, it has to take care of this specific and special features. There will be special instructions and these special instructions have to be selected for generation of the code.

The more common, the most common special instruction is your multiply and accumulate NLA instruction and use of NLA instructions have to be judiciously made. Another interesting feature is address generation schemes, many of these DSPs provide support for auto increment and auto decrement operations. In fact, your general purpose processor like arm also provides these features. In fact, arm in various ways do incorporate DSP features. Then there are supports for loop in fact, what is called a 0 overhead loop that support is there in the hardware and compiler therefore, should exploit the special features. We shall see how these special features can be real exploited.

(Refer Slide Time: 03:39)



Example

Application: $y[j] = \sum_{i=0}^{n} x[j-i]*a[i]$

$\forall i: 0 \leq i \leq n: y_i[j] = y_{i-1}[j] + x[j-i]*a[i]$

Architecture: Example: Data path ADSP210x

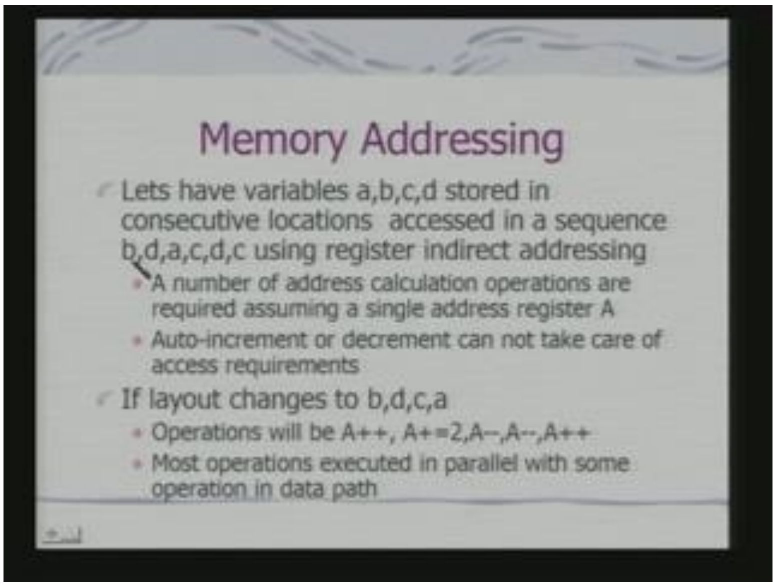
- Parallelism
- Dedicated registers
- No matching compiler => inefficient code

Let us take an example of architecture, this is DSP processor and typically say you will be working out these kinds of computations. If we considered this has a data stream what you are doing is your basically multiplying by a coefficient and getting the result summation and this is an incremental addition of this sums. So, if we can put it together. So, I am basically generating a stream of y values which is dependent on the previous y value as well as input stream. So, these are very typical computations which you encounter for a signal processing task. And the architecture typically supports various registers say for example, here what we have seen is that, I have got a i i have got x j minus i. And they will be computed and we shall get y i minus 1 which have to be used for the subsequent steps as well.

So, how to use these registers optimally that becomes the basic problem and you also have automatic address generation units. Now, you have to use automatic generation units. So that, you can generate addresses for this kind of sequential access. So, what we say that there is a parallelism in built into it there are dedicated registers. So, if I cannot match these requirements with that of the compilers basic code generation scheme then innovate, i can generate efficient code.

So, basic problem is that of match. In fact, we shall look at this problem in the contest of a simpler example because if you look at the problem units complete complexity it is really, complicated and there has been various sophisticated techniques developed for develop for generation of codes taking to account this consideration.

(Refer Slide Time: 05:50)



Memory Addressing

- Lets have variables a,b,c,d stored in consecutive locations accessed in a sequence b,d,a,c,d,c using register indirect addressing
  - A number of address calculation operations are required assuming a single address register A
  - Auto-increment or decrement can not take care of access requirements
- If layout changes to b,d,c,a
  - Operations will be A++, A+=2,A--,A--,A++
  - Most operations executed in parallel with some operation in data path

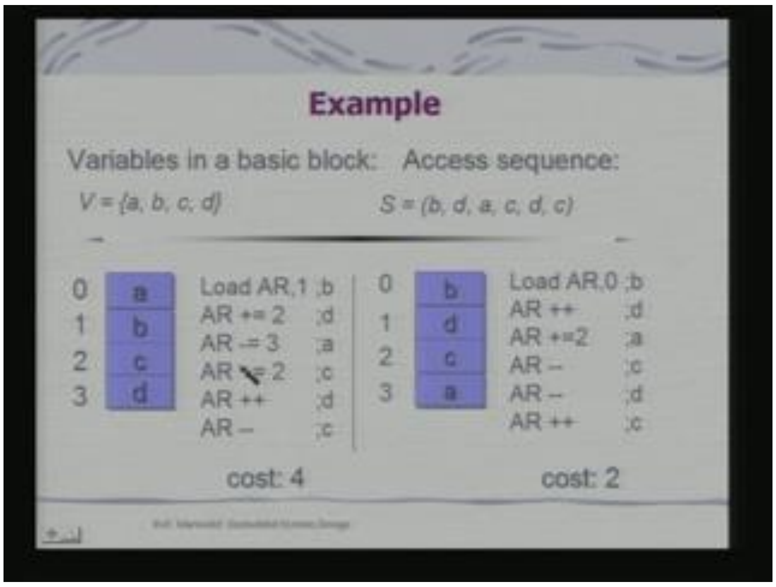
So, we shall consider a simpler problem that of addressing variables, let us say, we have got variables a, b, c, d stored in consecutive memory locations and which are accessed in a sequence b,c, a, c, d, c depending on the computation it is not a strictly sequential access, but we are accessing them b, d, a, c, d, c. So, I am not accessing them a, b, c, d, but in a different order although, this variables can be typically stored in the memory a, b, c, d and in maturity of the cases you will be accessing this variables using register indirect addressing. The question is there the optimal layout of these variables. So, that I can use auto increment or auto decrement option of the processor in an optimal fashion.

So, what we see is that a number of address calculation operations are required if we assume a single address register A. In fact, you can understand this is the kind of an abstraction. We are looking at the simplified version of the problem to understand its complexity and the way, in which it can be toggled. So, number of address calculation operations are required, if we assume a single address register and auto increment or auto decrement cannot take care of access requirements. Simply because after a it is not that your access in b, because b is placed after a you are first accessing b and then, you are accessing d and if they are placed even in consecutive locations what is interesting to notice that, I may load initially address of b into my address register indirect address register by the by incrementing it by 1 will not take b to d.

So, I have to do an actually an arithmetic operation on that address register and to access d so; that means, I have to execute a separate instructions. If I could have used auto increment; that means, I do not need an special instruction to do address calculation. If I do not use a special instruction for address calculation, it means; I consume less time, I consume less energy. So, in the layout now changes to b, d, c, a. So, what is interesting you can see is that, I can do A plus plus then, I increment A by 2 fine this is required and then I can decrement and again I can increment.

So, in fact, most of these address calculation operations are executed in parallel with some operation in the data path. Because auto increment and auto decrement features is part of your address generation unit.

(Refer Slide Time: 08:49)



Example

Variables in a basic block: $V = \{a, b, c, d\}$

Access sequence: $S = (b, d, a, c, d, c)$

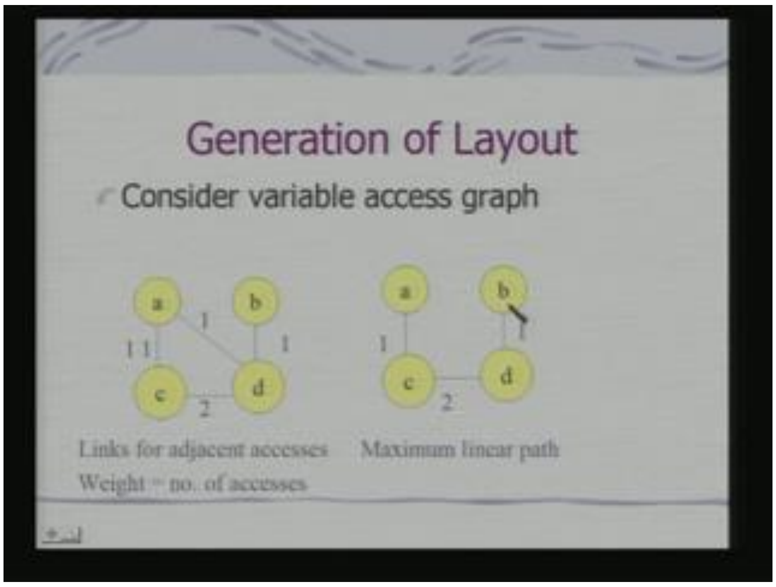
```
0  a    Load AR,1  ;b        0  b    Load AR,0  ;b
1  b    AR += 2    ;d        1  d    AR ++      ;d
2  c    AR -= 3    ;a        2  c    AR +=2     ;a
3  d    AR += 2    ;c        3  a    AR --      ;c
        AR ++      ;d                AR --      ;d
        AR --      ;c                AR ++      ;c

        cost: 4                      cost: 2
```

So, let us pictorial you look at what we are talking about, say this is my variables a, b, c, d. So, I have these variables a, b, c, d they are stored in consecutive locations this is an assumption. I have got a single address register AR through which I am accessing this variables see if i load first if i have to access b after load 1 if i am loading 1; that means, I have got my address of b here with the register AR. So, I can load the content of b now next, I need to access d. So, I have to increase AR by 2. Because then only I can go to d after d; I want to access a. So, I have to decrement AR by three again i have to increment AR by 2. So, all these operations are actually involved. So, these all consume time.

Now, when i change my layout to say b, d, c, a just look at this layout what is happening here, the basic issue here is that I am loading AR 0 and then if I increment by 1,I go to d for going to A, I need to increment by 2 no doubt about it. But subsequently it is all either decrement by a single value, to go to c decrement by again a single value to go to d and then, you incremented to again come to c. So, here all this operations AR decrementations as well as incrementation will take place in parallel using the address generation unit.

So, we can see these layouts are more efficient. In fact, a compilers problem is therefore, to exploit auto increment and auto decrement features of the instruction set is that of figuring out and optimal layout of this variables. The variables depending on their access pattern compiler should decide about its layout. Compiler cannot use, asserts a kind of an pre it determined fixed layout for storing this variables if it is doing. So, then it is paying for in terms of additional instructions.

(Refer Slide Time: 11:17)



Generation of Layout

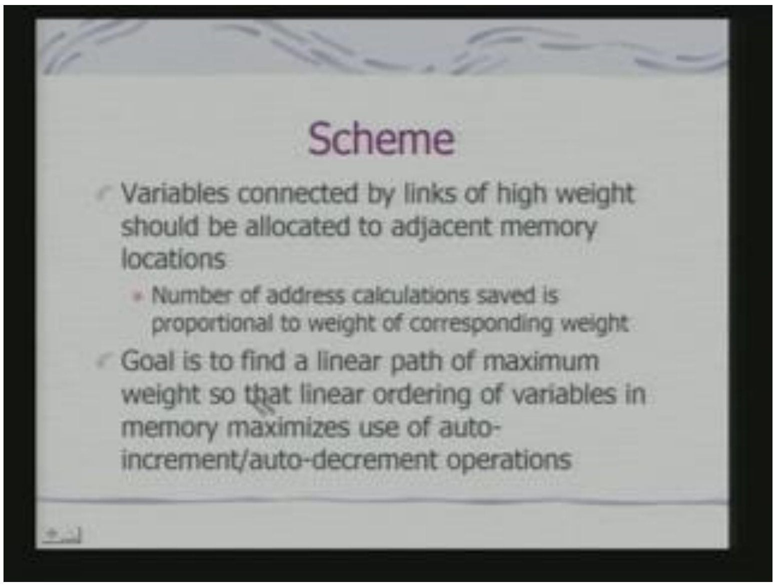
- Consider variable access graph

So, how to do that? So, you consider what is called a variable access graph there are various techniques we shall just considered 1 such technique a very simple technique which can be understood. So, we consider a very simple access graph and in that access graph, if you find that there are variables a, b, c, d and there are links between the variables indicating adjacent access now this is not eleven this is 1. So, a is connected to c by a link because a and c has got an adjacent access and there is only 1s adjacent access for the problem that we have considered. So, the link weight is 1; c and d has got 2 adjacent accesses. So, that is why the link weight is 2; a and d has got 1 adjacent access b and has got 1 adjacent access.

So, therefore, I have got a graph in the graph the links indicate whether there is adjacent access of these variables. And the weight of the link indicates the number of times this kind of adjacent access takes, this number of times this kind of adjacent access takes this. Now, form this graph I can extract out what we called maximum linear path. In fact, this is the maximum linear path. So, what does that mean; that means, if you look at these variables this is the bath along which the maximal accesses take place. Because, if we add of this 1 single path, this path gives you the kind of accesses that is there.

So, there may be more than 1 such linear path also, but here the interesting thing is what we have figured out here is that this gives you a linear path connecting all the variables, depending on their access pattern. And what it is suggesting that if I now, put a c, d, b; that is, c and d are consecutive that would maximally exploit the auto increment decrement operations because that weight is maximum. Similarly, a and c should be put together, because there is an adjacent access and there is no reason for putting a and b together, a consecutive locations because there is no adjacent access between a and b and d and b there is an adjacent access. So, there is a reason for trying to put them in a adjacent location. So, the basic problem is that of figuring out the maximum linear path. In fact, this is a graph theoretic problem.

(Refer Slide Time: 14:17)



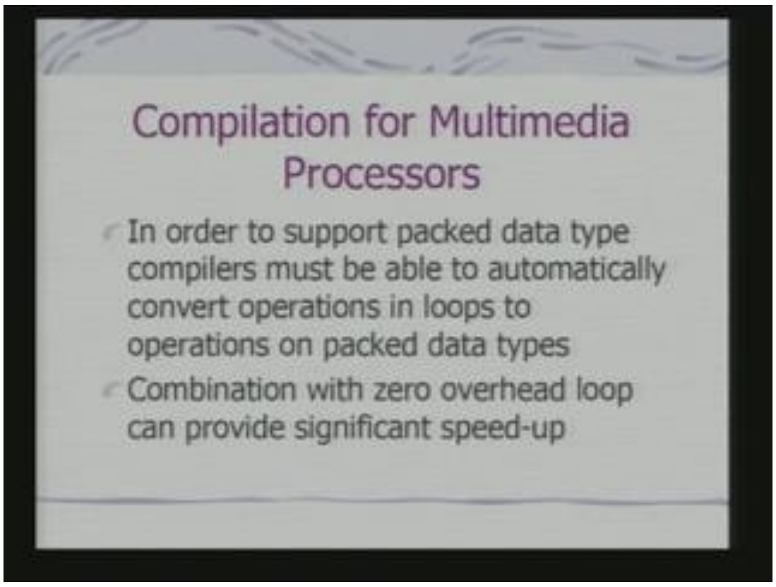
Scheme

- Variables connected by links of high weight should be allocated to adjacent memory locations
  - Number of address calculations saved is proportional to weight of corresponding weight
- Goal is to find a linear path of maximum weight so that linear ordering of variables in memory maximizes use of auto-increment/auto-decrement operations

So, what we say is that variables connected by links of high weight should be allocated to adjacent memory locations. I think intuitively the reason for it is clear because, that gives you the number of maximum adjacent locations and then only I can use auto increment and auto decrement features to the maximum possible extend. So, number of address calculations saved is proportional to the weight of corresponding weight. And the goal is therefore, to find a linear path of maximum weight. So, that linear ordering of variables, in memory maximizes use of auto increment or auto decrement operations.

So, in fact, again I have mapped the problem when optimization problem because I can have, a variable access graph and in the variable access graph. I have to find out the linear path of maximum weight and the linear path of maximum weight will decide on the layout of the variables. I am not going in to your exact algorithm to be used for solving this optimization problem, but you should understand that, if I can solve this optimization problem, I shall get the optimal layout of the variables. Obviously, with assumption here is what that, I am using 1 address register for register indirect addressing. If I talk about using more than 1 registers, talk about using automatic address generation unit the problem becomes more complex next we shall look at multimedia processors.

(Refer Slide Time: 15:53)



Compilation for Multimedia Processors

- In order to support packed data type compilers must be able to automatically convert operations in loops to operations on packed data types
- Combination with zero overhead loop can provide significant speed-up

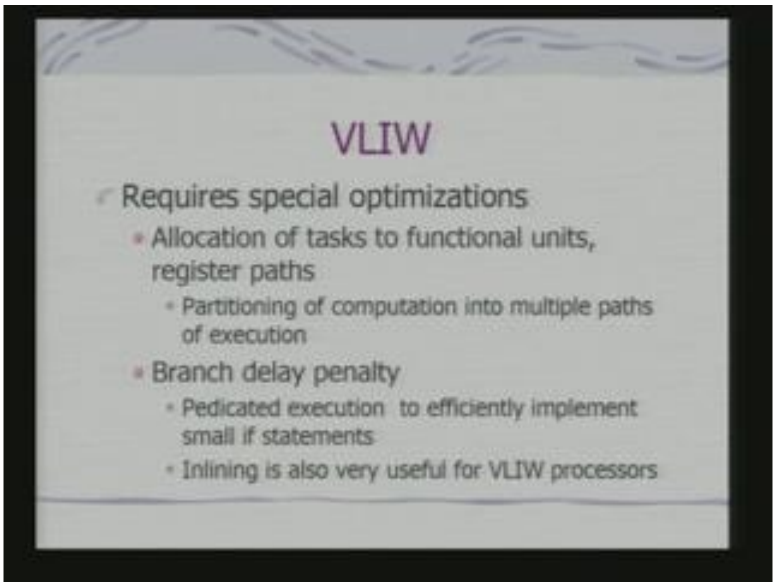
So in fact, majority of the multimedia processors use what are called today used packed data types. In fact, we have looked at packed data types in the context of Pentium processor also with mmx enhanced instructions where, I can pack multiple bytes or multiple integers together into a single word and use a single instruction to operate in parallel on the multiple data elements packed into 1 word. That means; if I packed 8 bytes together in 64 bit word and if you are using any of this multimedia instructions, I carry out, an arithmetic operation in parallel on those 8 bytes. So, I am having kind of a parallelism exploiting just the ALU feature for a 64 bit word.

What is the advantage here in this case, in this case the advantage is in many of this multimedia processing unit to use same operation repeatedly on a number of data items. So, I am actually having a kind of an in instruction set which is supporting what we called single instruction multiple data parallelism. Now, the compiler job is to figure out that parallelism because when, you write a processing operation in a high level code say you want to add a fixed value to each pixel in an image. So, an image is what a 2d array and if you are adding a single value to each of the pixel you will be writing a nested for loop the compiler needs to figure out form that nested for loop of the operations.

So, that it can be converted into a packed operation and the packed data type, this is the basic problem. So, what we say that in order to support packed data type, compilers must be able to automatically convert operations in loops to operations on packed data types. So, the operation that I am doing on 256 cross 256 pixels individually has to be converted into appropriate number of times in the loop. Because I can now, packed 8 pixels of 8 bits together into a single packed data type and used a single operation may be to increment each of those bytes. Now, combination of these with 0 overhead loop can provide a significant speed up.

So, that is the basic issue that compilers need to know that there is the packed data type it has the ability to identify the loops corresponding to which, the packed data types can be really used. It is not that it can be used for any for loop for that matter. So, it should analyze the code, have the ability to analyze the code and use the packed data types.

(Refer Slide Time: 19:06)



VLIW

- Requires special optimizations
  - Allocation of tasks to functional units, register paths
    - Partitioning of computation into multiple paths of execution
  - Branch delay penalty
    - Pedicated execution to efficiently implement small if statements
    - Inlining is also very useful for VLIW processors

When you look at VLIW processors, it requires some amount of special optimizations because there are multiple functional units and these multiple functional units can carry out independent operations. So, the compilers tasks are to allocate individual computational jobs to functional units as well as to the corresponding register paths. Because the registers if you remember the VLIW architecture registers are link to various functional units and there are different kind of register organizations, which is adopted by VLIW processors for connecting the registers to the functional units.

So, very interesting and important problem of the compiler to optimally decide and usage of this registers and the usage of this registers, so, that the functional units can be exploit. So, it is no point using a VLIW processor and carrying out only one computational job in an instruction, because in that case your functional units are remaining idle. So, the point therefore, comes back to is that you look at the computation as the whole, the problem is that of partitioning of computation into multiple paths of execution. And this paths of executions have to be mapped to, the set of registers and the corresponding functional units and these paths in a way, we have to identified such that the dependencies are minimum. If they have got dependencies then, I cannot schedule operations in parallel.

So, this dependencies have to be figured out you have to apply operations to minimize the dependencies or to eliminate the dependencies then, identify the different execution paths and mapped the different execution paths on the registers and the functional units. So, that the potential of the processor can be completely exploited. In fact, related to this is what is called branch delay penalty. Why branch delay penalty is critical for VLIW processor, because if there is the branching then, what is the operation scheduled on different functional units because that may not be required at all. In that case of branch will cause or kind of a penalty or delay effectively or delay.

So, branch delay penalty for VLIW processors something which is very important, in fact it is true also for pipeline processors. They are also the instructions scheduling has to be search the pipeline should not be empty or has to be flush at a regular interval. So, 1 way to deal with this problem is, you know predicated execution to efficiently implemented small if statements. What is the predicated execution? It is nothing but, conditional execution depending on some predicate that is some condition you execute instructions. In fact, we have seen such instructions in arm processor as well.

So, if I use conditional instructions what happens, my pipeline need not be flushed effectively, I am getting are not inserted if that condition is not satisfied the corresponding operation is not really getting executed, but the pipeline need not be flushed.

Similarly, for VLIW processors the scheduling of instructions are search that the schedule need not be changed because of this problem. So, what we say that predicated execution to efficiently implement small if statements. So, I should used conditional instructions to implement small, if I should not make a conditional jump. The moment I am making a conditional jump instructions, which is to be executed has to be changed and that introduces overhead that introduces branch delay penalty which should be avoided.

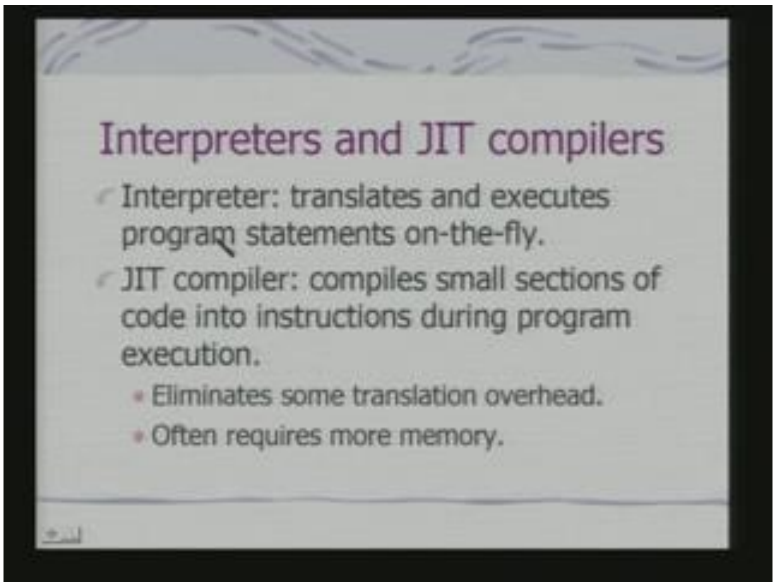
So, in fact in this context in lining is also useful what is in lining. In fact, whenever there is the function call there is similar penalty just like a branch delay penalty; whenever there is the function called there will be the similar penalty. Now, the question is if the function is small enough, you should not call that function. If the function is small enough you should not call that function you insert, the code of that function at an appropriate position where it has been called. Obviously, you can realize the moment I do in lining the memory requirement of the code increases, but the efficiency of its execution at the same time is better off.

In fact, in lining is an universal technique in various cases even if you are not using a VLIW processors compilers do in lining, whenever there is the small function calls it may decide for the purpose of optimizing the code to replies the function call by the actual code itself. Because you then, save the overhead of saving the registers pushing the current program counter and again when you are returning from the call you have the overhead of popping the registers as well as getting the address from the stacks. So, those overheads are completely avoided if I do in lining.

So, compilers as part of generic optimization strategy also use an in lining, but in the context of VLIW processor this becomes more important because you have now ever multiple functional units and multiple functional units idling a way because of a call for a small function is potentially more problematic, then wasting away few instruction cycles in a non VLIW processors. So, that is the precisely reason why in lining is extremely useful for VLIW processors.

So, this moral less covers our basic techniques that we had so, far talked about for the compilers for embedded systems and architecture specific features and you must have realize by now that, compilers also therefore, become a very important tool in the complete design process. Because without help of a good compiler you cannot generate efficient code, if you cannot generate efficient code the question of performance estimation in terms of your time as well as energy computations is no longer, very useful. Because if you are using an in efficient compiler generating inefficient code, doing a performance estimate and finding that, you cannot meet the requirements and therefore, go for a better processor. Then you are not actually exploiting the potential of the processor. And that is why compiler is a critical tool for doing an embedded system design.

(Refer Slide Time: 26: 38)



Interpreters and JIT compilers

- Interpreter: translates and executes program statements on-the-fly.
- JIT compiler: compiles small sections of code into instructions during program execution.
  - Eliminates some translation overhead.
  - Often requires more memory.

So, once you have done the design or working with it we should also know that, what are the variances of these compilers that are also today used in various embedded systems? In fact interpreters and just in time compilers are 2 variance of classical compilers, which is found very commonly today in embedded systems. This is particularly true in the context of java enabled embedded systems. If we have seen that today in order to provide a flexible programmability to the embedded systems, we have what we called java virtual machine built into an embedded system. So, what does the java virtual machine do?

So, a java virtual machine typically interprets what is called java bytes code and java byte code is obtained by compiling the java source code. What is really interpretation? Interpretation is translation and execution of program statements on the fly. In a very simplified fashion you can considered that there is the single statement that, is java byte code and that statement is being translated to the machine code of the target processor. In fact, java byte code will be in a language of java virtual machine; that means, it would use the instruction set of java virtual machine and you might be actually running your application and arm processor.

So, each instruction of java virtual machine needs to be now translated to a machine instruction of the target processor interpreter actually does that job. In fact java virtual machine in a way is that interpreter. So, what it does, it translates instructions 1 after another. Now, what is the advantage or drawback of this, advantage is very straight forward. I can download the java code from any source because, I do not need to have the java code translator to the machine instruction of my processor, I can get universal byte code load it onto my appliance and execute on my appliance. So, that is the advantage of the interpreted scheme.

What is the disadvantage? Disadvantage is the kind of optimization that I had talked about, for the compilers it is not possible to be applied there because you are not looking at a set of instructions, but considering instructions individually. So, interpreters really cannot implement the kind of optimizations that we have talked about, but optimizations are critically important if we are interested in running java on embedded systems. Because we have understood that from the examples and the issues that we have examine.

So, there is now a concept. In fact, in terms of architecture of the compiler you have the concept what is called just in time compiler. So, what is the just in time compiler does just in time compiler, in a put in a very simplified way is something in between a full compiler and an interpreter just in time compiler we look at not at the complete code. But may be at section of the code and it would now, compile that section of the code into the machine instructions of the target processor.

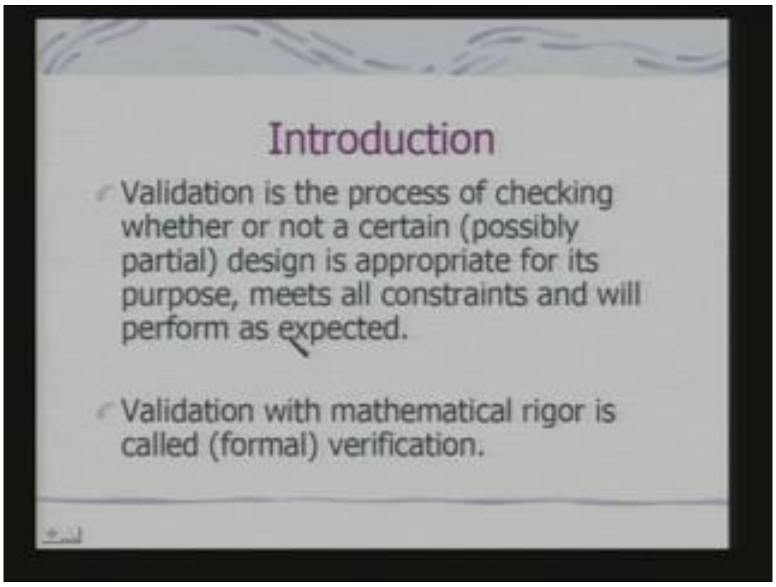
So, in this case what happens the compiler the just in time compiler has the option to carry out optimization with respect to a small sections of the code, may not be the complete section of the code, but atleast the small section of the code. And hence you can actually generate optimized code and not non optimized code which results, if I look at instructions individually for say a java byte code. And since it is looking at small part of the code and looking at a small part of the byte code, actually lots of translation overhead or in a way eliminate it.

So, you need not bothered to maintain a complex symbol table and those kind of data structures, as it is required for a complex on a complete compiler. But obviously, it would require more memory, why more memory because it has to now deal with a section of the code together and then convert that code through processing into the target machine code. In the earlier case, interpreter can just look at in a very ideal in a very abstracts scenario; the interpreter can look at a single instruction and convert upto the machine code. So, it did not have the memory to keep track of a section of the code putting it through a set of transformations for the possible optimized target to be generated.

So, in fact, just in time compiler compilation techniques are very useful particularly for implementation of optimized java code in embedded systems. So, once, we have got therefore, tools to do hardware software partitioning then hardware design. We have looked at hardware design in terms of processor choice, a specification of the computation also we have looked at software design translation to the architecture specific instructions. So, we have all the design tools with us and we can do the design.

Next question comes, that of design and product validation because we have started with what, we have started with requirements specification. The requirements could be functional as well as non-functional; once you have done the design we now, need to know whether at the end of the day, I am meeting my requirements specifications or not.

(Refer Slide Time: 33:10)

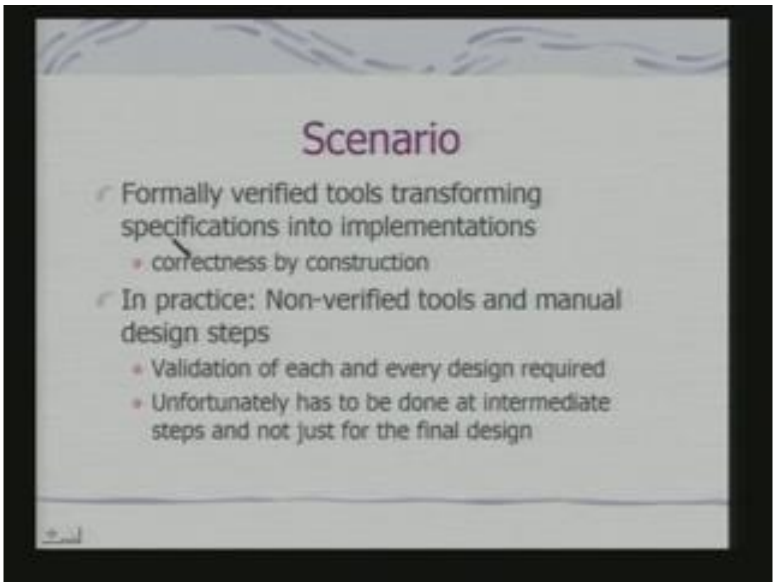


Introduction

- Validation is the process of checking whether or not a certain (possibly partial) design is appropriate for its purpose, meets all constraints and will perform as expected.
- Validation with mathematical rigor is called (formal) verification.

So, validation is the process of checking whether or not a certain design is the appropriate for its purpose, meets all constraints and will perform as expected. So, this is basically what is known as validation; now, validation can be done in a very formal way. So, validation with mathematical rigor is called formal validation.

(Refer Slide Time: 33:34)



Scenario

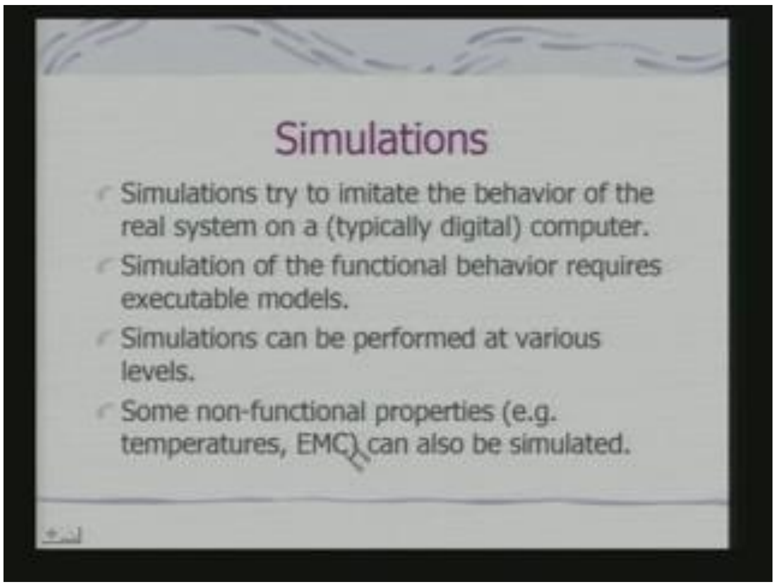
- Formally verified tools transforming specifications into implementations
  - correctness by construction
- In practice: Non-verified tools and manual design steps
  - Validation of each and every design required
  - Unfortunately has to be done at intermediate steps and not just for the final design

But formally verify tools what they do, they transforms specifications into implementations and in that case we have got correctness by construction. That means, if you are using tools which are formally verified; that means, tools are guaranteed to generate designs which are to be correct. Then you get a correct design through construction provided you have provide you have given the tool correct specifications.

If your specification itself is wrong you cannot get an implementation we should be correct you will get an implementation which is correct with respect to your specifications. But in practice in maturity of the cases you usually do not use this kind of formally verified tools and you go through manual design steps. You might be using tools at various steps, but the overall design; obviously, involves manual interventions and manual decisions.

So, in that case validation of each and every design is required and unfortunately it has to be done at intermediate steps and not just for the final design. Because if you are waiting till the final design then the man and the money spends for doing that; so, how it is done?

(Refer Slide Time: 34:52)



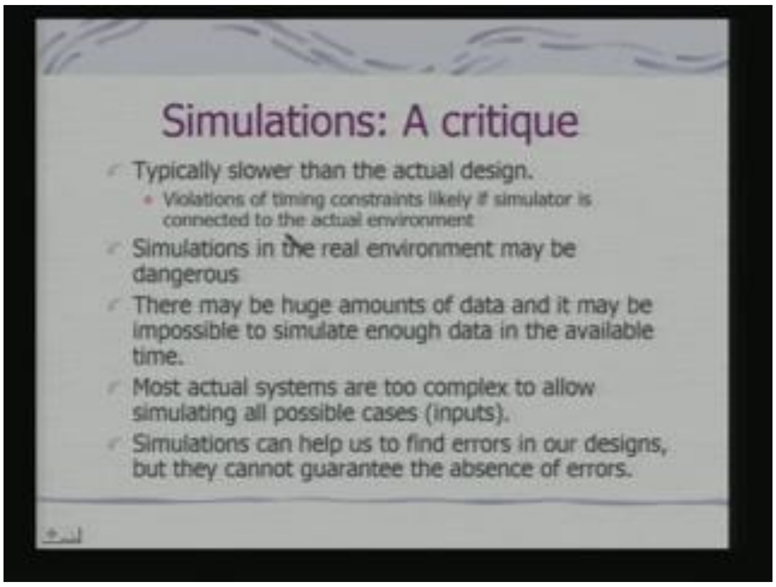
Simulations

- Simulations try to imitate the behavior of the real system on a (typically digital) computer.
- Simulation of the functional behavior requires executable models.
- Simulations can be performed at various levels.
- Some non-functional properties (e.g. temperatures, EMC) can also be simulated.

So, a very simple way is simulations because simulations try to imitate imitate behavior of the real system on a digital computer. Simulation of the functional behavior request, what we called executable models, because I need to know, how you execute the whole system. And simulations can be performed at various levels. In fact, we are talked about behavioral simulation just to see whether the behavior is being met. At a second level I can do a clock level simulation to see whether the timing constraints can be actually met. And you can do this therefore, simulation at various levels and you can do hardware and software co simulation as well.

So, that means; we can design the architecture mapped architecture to the simulated test pad have the code and run the code while simulating the architecture on the simulation test pad. In fact, some non functional properties exams temperature EMC can also be simulated. So, EMC is basically electromagnetic conditions which are there prevailing in the environment in which your embedded systems is expected to be placed.

(Refer Slide Time: 36: 14)



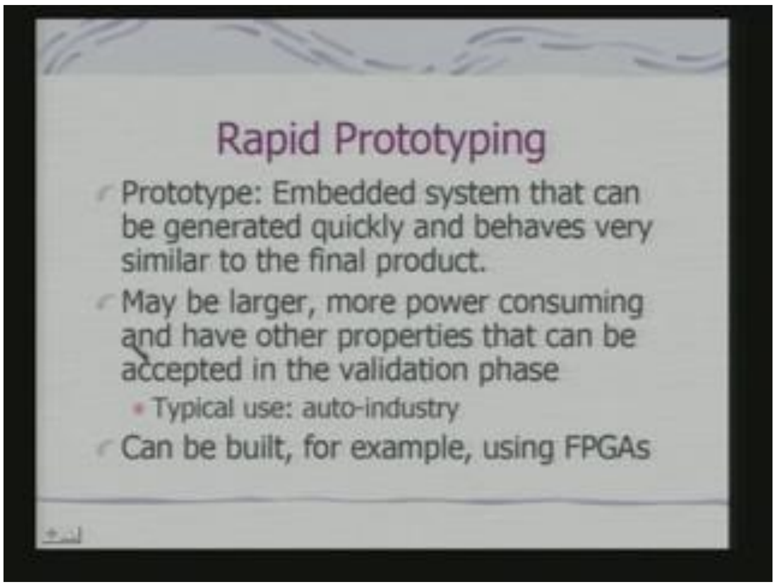
Simulations: A critique

- Typically slower than the actual design.
  - Violations of timing constraints likely if simulator is connected to the actual environment
- Simulations in the real environment may be dangerous
- There may be huge amounts of data and it may be impossible to simulate enough data in the available time.
- Most actual systems are too complex to allow simulating all possible cases (inputs).
- Simulations can help us to find errors in our designs, but they cannot guarantee the absence of errors.

So, a simulation is typically slower than the actual design and violation of timing constraints likely if simulator is connected to the actual environment. Because simulations obviously, means you are doing some kind of approximation number one, number two is you are also doing to monitor the performance some kind of management of the environment that code which goes into management of the environment. That is, housekeeping tasks that itself can consume time and. So, it is typically slower than actual design. So, next thing is simulations in the real environment can be dangerous obviously you will not like to simulate a safety critical system with the real object.

So, you will not like simulations to produce catastrophic results and there may be huge amounts of data and it may be impossible to simulate enough data in the available time. See if I am actually designing a compression engine then, I have to actually get a volume of data to verify whether the compression engine is working correctly or not. And most actual systems are too complex to allow simulating all possible cases, because then the time requirement for doing simulation becomes high. So, simulations are done typically at an intermediate stage and simulations can help us to find errors in our designs, but they cannot guarantee absence of errors. This is the very basic observation, simulations can help us to find errors in designs, but they cannot really guarantee the complete absence of errors. In fact, we need to develop prototypes after we have gone through the simulations in order to have some assessments about, the quality and the nature of the product

(Refer Slide Time: 38:04)



Rapid Prototyping

- Prototype: Embedded system that can be generated quickly and behaves very similar to the final product.
- May be larger, more power consuming and have other properties that can be accepted in the validation phase
  - Typical use: auto-industry
- Can be built, for example, using FPGAs

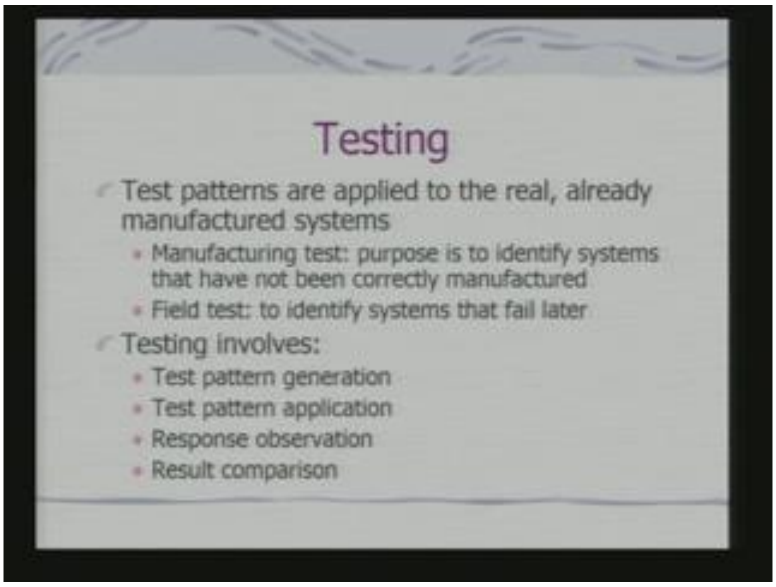
So, what we come to is Rapid prototyping? So, embedded system a prototype is what a prototype is an embedded system that can be generated quickly and behaves very similar to the final product may be larger. So, when you are talking about a prototype we may relax some of the performance constraints some of the non functional constraints in order to do the to do the rapid prototyping. So, that means, we can have may be physical size wise, a larger system we may have a system which is more power consuming and have other properties that can be accepted in the validation phase.

In fact, typically what happens for an auto industry the practiced way to validate design of an embedded system targeted for auto industries to built prototypes? The prototypes may not exactly fit into all your engineering aspects of the design that we say for example, if you are trying to design a navigation aid which is likely to going to your dash code, you might not really design a product meeting the measurements weight as well as power consumption requirements.

You might build a system which may not have these constraints, but verify whether it is providing the navigation. If it is providing the navigation the next part of the engineering aspects of the design is, reducing it size try to optimize for the power etcetera. And many cases, these prototypes are built using FPGAs because FPGAs can be very easily synthesized starting from say VHDL or system C descriptions of the embedded appliances. So, from there we can built FPGAs we can even execute our code on this prototypes, FPGA based prototypes and verify the overall design.

Once that overall design is verified; now, these FPGAs can be converted to an ASIC or SOC as the case may be. In fact, you can have even FPGAs with processors; we have looked at such platforms if you really have such platforms, then what we can do. We can use such platforms also for design validation may not be final design, but also for design validation.

(Refer Slide Time: 40:57)



Testing

- Test patterns are applied to the real, already manufactured systems
  - Manufacturing test: purpose is to identify systems that have not been correctly manufactured
  - Field test: to identify systems that fail later
- Testing involves:
  - Test pattern generation
  - Test pattern application
  - Response observation
  - Result comparison

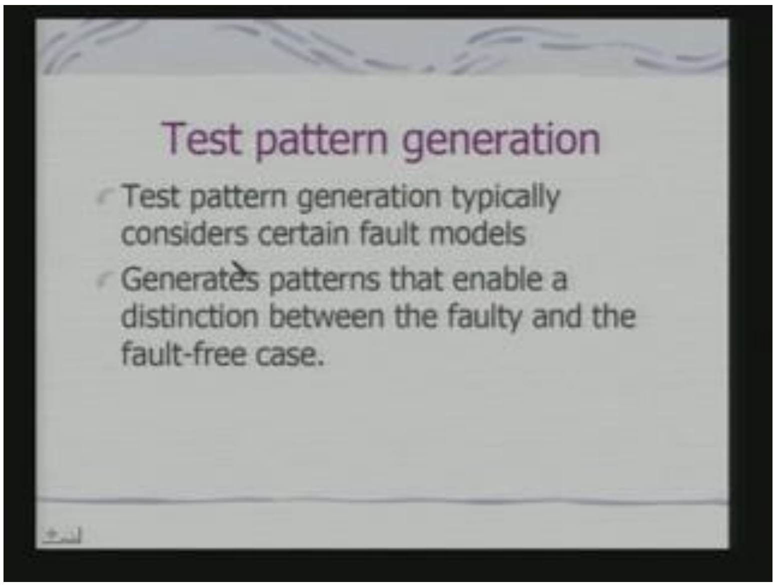
So, next thing is testing. So, in fact a validation process really involves testing. So, test patterns are applied to real already manufactured systems. In fact, when we build the prototype, we are also we have to applied the test patterns to check whether it is meeting our requirements or not. But, typically test refers what we called manufacturing test and field test; that means, if we assume that, we have gone through a design validation process we have finally, design the product and we are in the process of getting the product out, then products have also to be tested.

So, there are two kinds of test that products typically undergo; 1 is called manufacturing test, the other is field test. The purpose of the manufacturing test is to identify systems that have not been correctly manufactured and field test to identify systems that fail later it may be. So, that we may not applies strictly speaking manufacturing test, to each and every appliance of the each and every product that, I am manufacturing. In fact, there are typical what are called statistical quality control, standards and paradigms which are followed for the purpose of sampling out the products and testing them out.

So, if from a batch of products we find that the number of products turning out to be failed or fault prone or failure prone is it such number is greater than of threshold as per SQC norms then we might need to reject the complete batch of products. If we find such faulty cases are less than threshold we may accept the entire batch. So, the testing involves test pattern generation, test pattern application response observation and results comparison.

So, I need to apply a test patterns needs to generate the test pattern then apply the test pattern. See what kind of observation the system is providing and figure out whether results that you are getting needs to a specification or not.

(Refer Slide Time: 43:04)

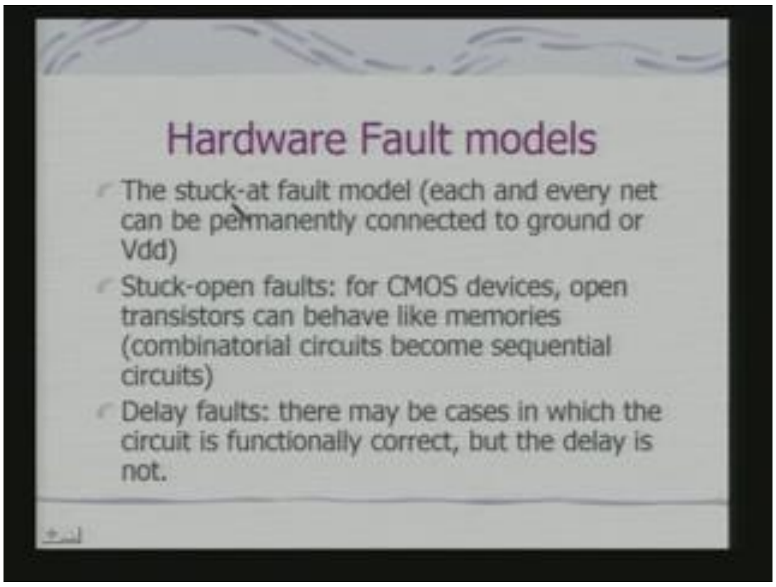


Test pattern generation

- Test pattern generation typically considers certain fault models
- Generates patterns that enable a distinction between the faulty and the fault-free case.

Test pattern generation typically considers certain fault models; because you have to have the fault models without, which you cannot really generate the test patterns. So, what does that mean? It generates patterns that enable or distinction between the faulty and a fault free case.

(Refer Slide Time: 43:22)



Hardware Fault models

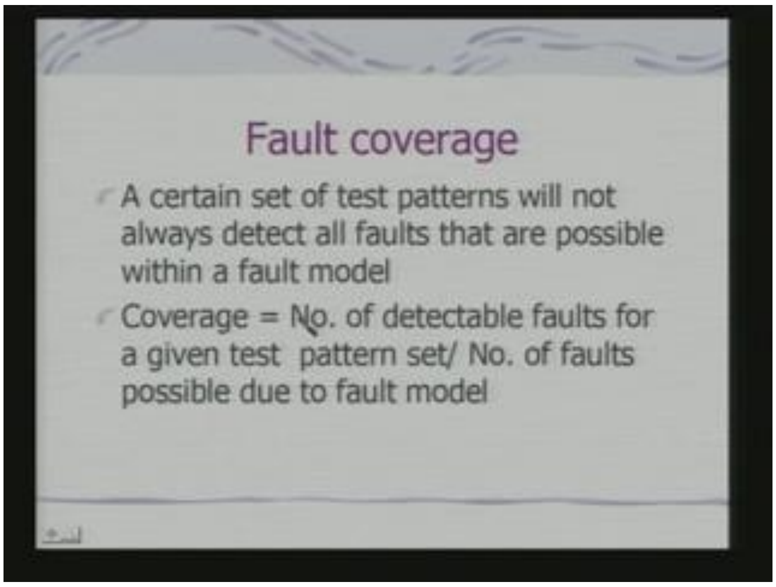
- The stuck-at fault model (each and every net can be permanently connected to ground or Vdd)
- Stuck-open faults: for CMOS devices, open transistors can behave like memories (combinatorial circuits become sequential circuits)
- Delay faults: there may be cases in which the circuit is functionally correct, but the delay is not.

So, what are different kinds of fault models typically we are looking at hardware fault models. Hardware fault models can be stuck at fault model each and every net can be permanently connected to the ground or vdd that means, each and every line can be connected to vdd or ground. So, you can test out if I look at stuck at fault model you can always figure out an input data combination which would give you an incorrect output if a particular say data path in processor is stuck at 0 or stuck at 1.

Similar thing would be true if any data line or an address line for a memory chip is stuck at 0 or stuck at 1. The other thing is stuck open faults that open transistors can behave like memories CMOS devices. So, in fact stuck open means there are disconnections. So, you need to know, what kind of behavior is expected of the device and accordingly you have to decide on the test data. So, that you can figure out, whether such an error has occurred or not also there are delays faults there may be cases in which, the circuit is functionally correct, but delay is not.

Because delay in a circuit can be see, if I look at a very sequential circuit it would depend on the number of gates that is there is a path. So, propagation delay would give you the actual switching delay. Now, you might have got a design for propagation delay search that you are getting a switching delay which exceeds the constraints, but functionally the circuit is behaving the way you would like 2. So, you got 2 have a test strategies testing strategy; so, that this delay faults can be identify related to this is the concept of what is called fault coverage.

(Refer Slide Time: 45:18)



Fault coverage

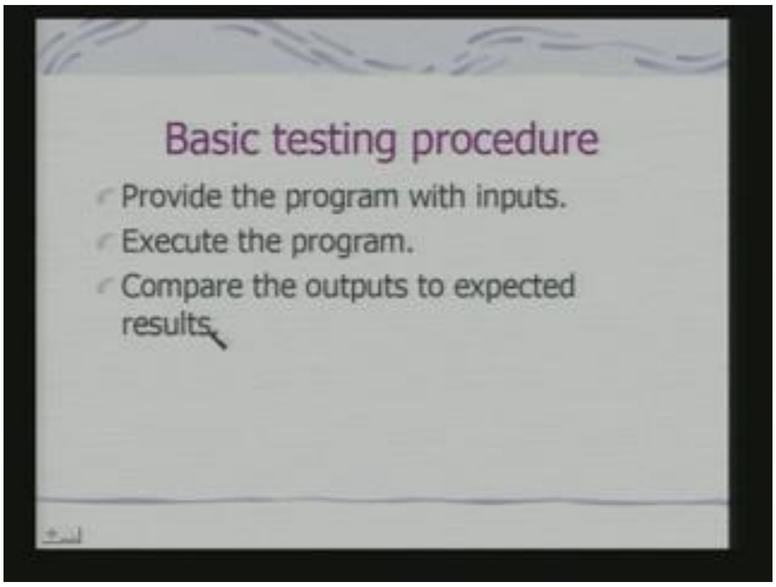
- A certain set of test patterns will not always detect all faults that are possible within a fault model
- Coverage = No. of detectable faults for a given test pattern set/ No. of faults possible due to fault model

A certain set of test patterns will not always detect all faults that are possible within a fault model. Because that is not really always possible then the number of test pattern that you need to use can become very large. So, we have to select a set of test patterns which will detect the different faults has suggested for the corresponding fault model. Now, to actually evaluate the test we use a parameter which is called fault coverage. Fault coverage is the number of detectable faults for a given test pattern divided by the number of faults possible due to fault model.

So, coverage if is 1, if we have tested with all possible test patterns then in that case, I can cover all possible models all possible faults that can with their because of a particular fault model. But for the purpose of efficiency we would like to use a coverage which may be as close to 1, but may not be really 1. And our objective would be select, a set of test patterns the set of test patterns may be small enough and give high coverage.

Now, similar testing has to be carried out for the software as well because software is the driving force beyond the hardware, we can check the hardware circuitry, but when and how to use the hardware circuitry would be determine by the software running in an embedded system. So, testing of the software for embedded systems is also critically important.

(Refer Slide Time: 46:58)

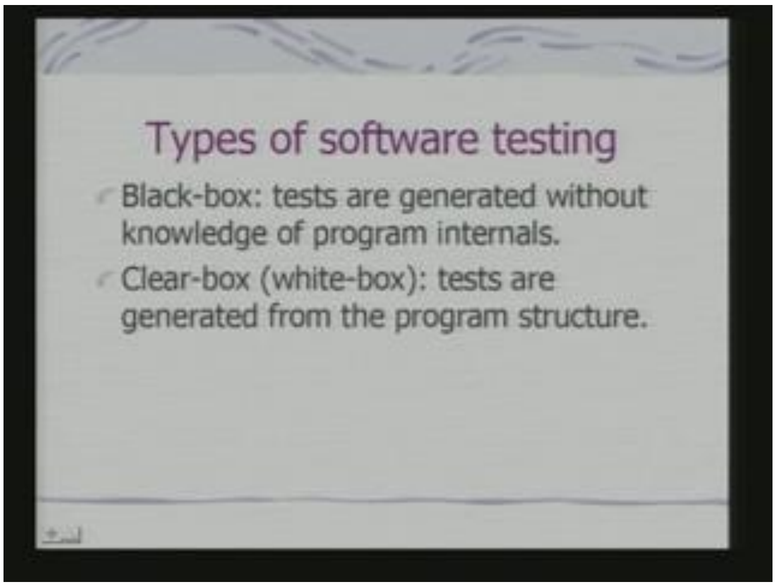


Basic testing procedure

- Provide the program with inputs.
- Execute the program.
- Compare the outputs to expected results.

So, basic testing procedure provides the program with inputs execute the program and compare the outputs to expected results that is, the standard is same as that of the hardware.

(Refer Slide Time: 47:07)



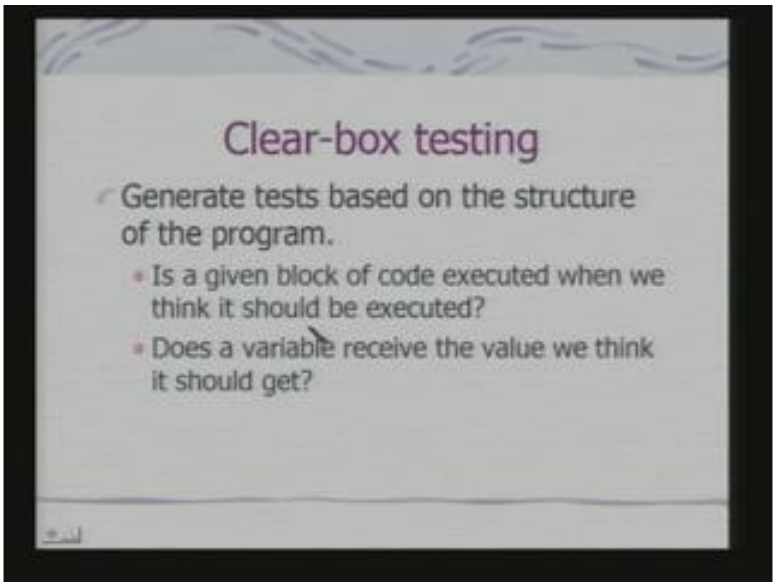
Types of software testing

- Black-box: tests are generated without knowledge of program internals.
- Clear-box (white-box): tests are generated from the program structure.

What are the types of software testing you really have? Black box testing and clear box testing and these are very. In fact, standard software engineering practices have to be followed here as well. In a black box test tests are generated without knowledge of program internals. In fact, these tests are typically generated from the specification; that means; once you know the functional specification given the functional specification you should design the black box tests.

Because if you try to design the test by knowing how it has been implemented then, you may miss out certain aspects of the specification all together. The other thing is clear box testing tests are generated from the program structure. In fact, typically you would be doing a clear box testing first and then going to a kind of black box testing.

(Refer Slide Time: 47:57)

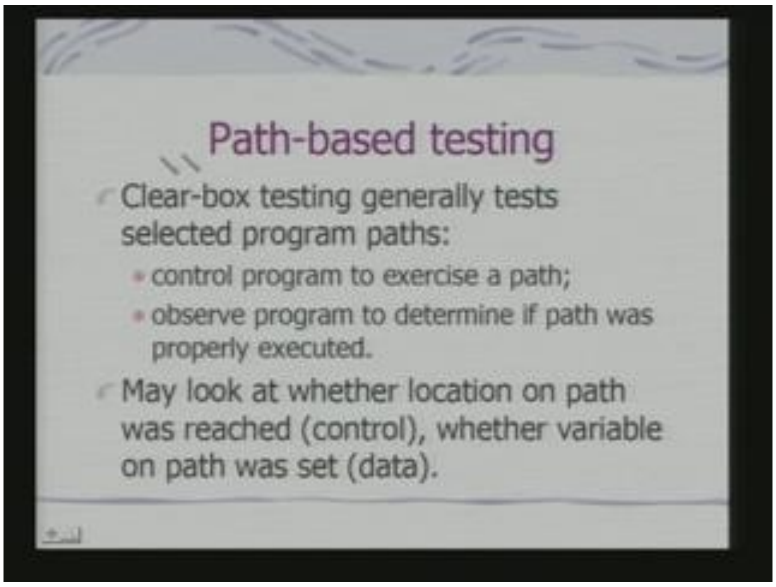


Clear-box testing

- Generate tests based on the structure of the program.
  - Is a given block of code executed when we think it should be executed?
  - Does a variable receive the value we think it should get?

What is the clear box testing? It generates tests based on the structure of the program. So, the question is is a given block of code executed when we think it should be executed or; that means, is my control path, does the variable receive the value we think it should get. That means the computation assignment process is correct or not.

(Refer Slide Time: 48:21)

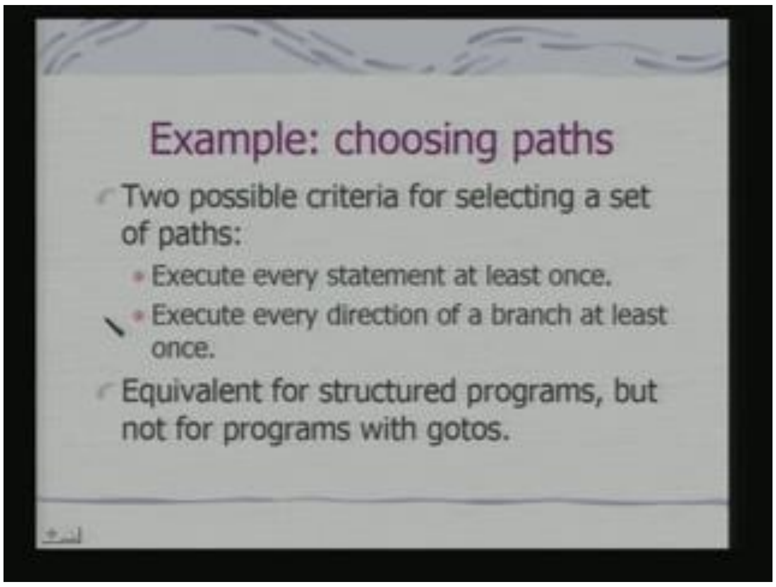


Path-based testing

- Clear-box testing generally tests selected program paths:
  - control program to exercise a path;
  - observe program to determine if path was properly executed.
- May look at whether location on path was reached (control), whether variable on path was set (data).

So, one aspect or one way of doing clear box testing is what is called path based testing. In fact, clear box testing; obviously, cannot test all possible paths in the program. It would select and test few of the possible paths. So, use a control program to exercise a path you observe, the program to determine if path was properly executed or not. And may look at whether location on path was reached. So, this is the control aspect of it and whether variable on data path was set this is the data aspect it. So, typically in a clear box testing you do what is known as path based testing.

(Refer Slide Time: 49:04)



Example: choosing paths

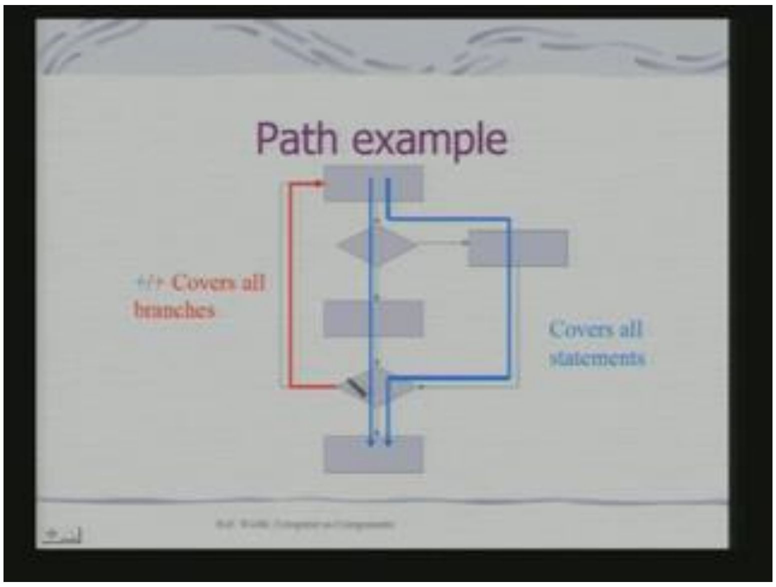
- Two possible criteria for selecting a set of paths:
  - Execute every statement at least once.
  - Execute every direction of a branch at least once.
- Equivalent for structured programs, but not for programs with gotos.

So, your problem is that of choosing the paths fine; in hardware it was the problem of choosing the test patterns such that you have the maximum fault coverage. In case of software, if you are doing a clear box path the problem is that of choosing paths and once you choose the path. Then only the problem of test pattern will coming because test pattern or test input will force the software to go through these paths.

So, there can be two possible criteria of selecting paths; execute every statement at least once and execute every direction of a branch at least once not for programs with gotos many cases for embedded systems you really write programs with gotos. A very simple example: we have seen in the last class that, when you are using scratch pad memory. I have to put go to for a set of code which is putting scratch pad memory and come back.

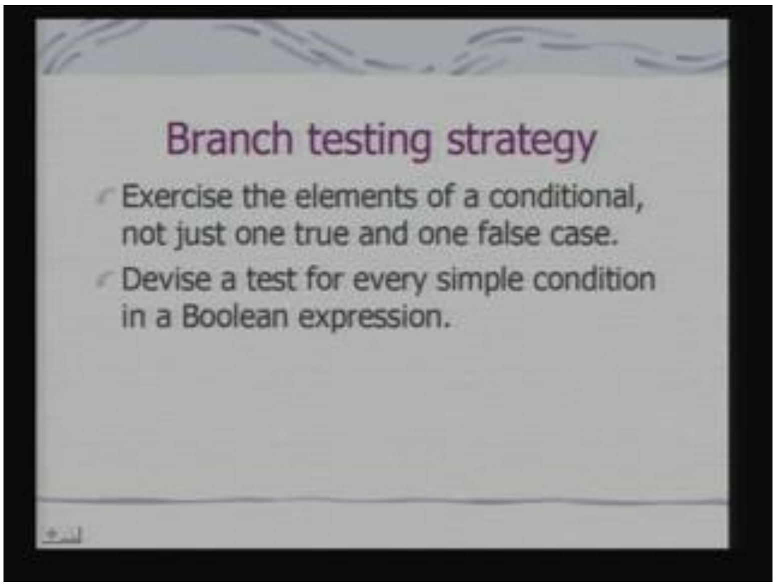
So, I need to have test to check whether those go to they are also taking place correctly or not. So, execute every statement at least once and execute every direction of the branch at least once.

(Refer Slide Time: 50:25)



So, take an example this is a simple flow graph. In fact, this is this is a representation of the program as the CDFG. So, if I take a path. So, if I take these two paths it covers all statements it covers all statements, but it does not cover this branch. So, I need to check this as well condition as well to test all branches corresponding to this CDFG.

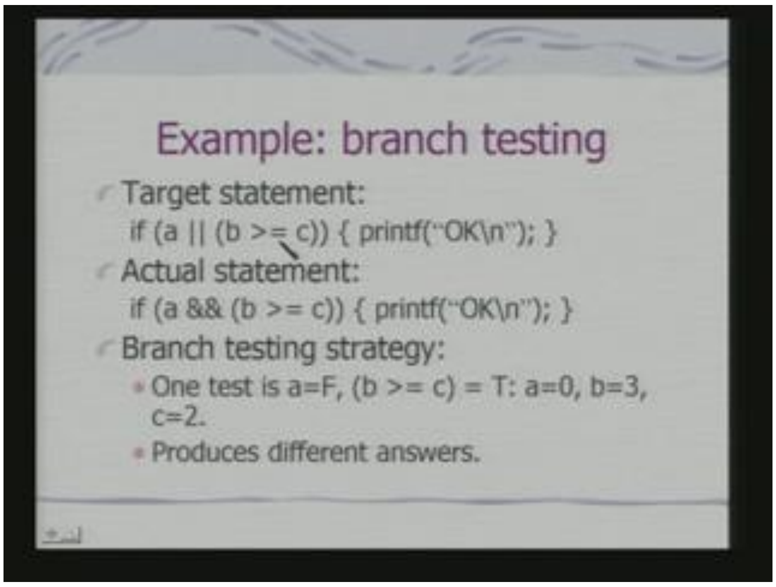
(Refer Slide Time: 50:54)



Branch testing strategy

- Exercise the elements of a conditional, not just one true and one false case.
- Devise a test for every simple condition in a Boolean expression.

(Refer Slide Time: 51:07)



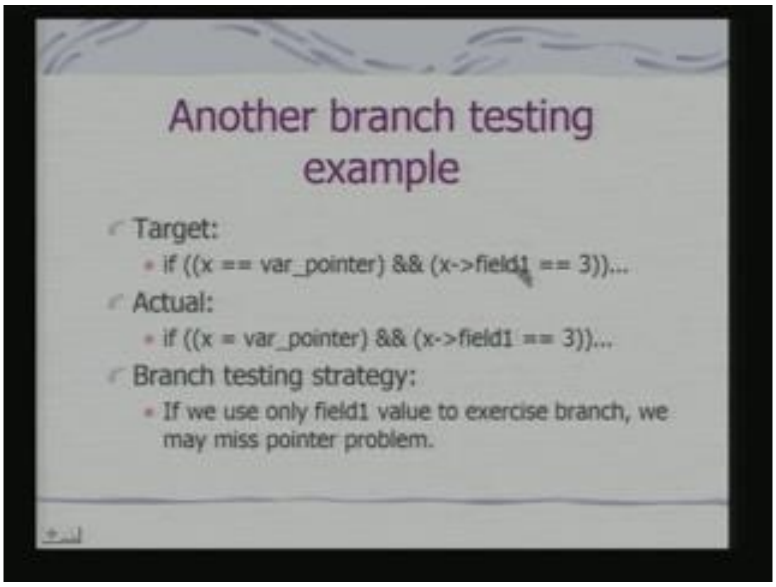
Example: branch testing

- Target statement:
  `if (a || (b >= c)) { printf("OK\n"); }`
- Actual statement:
  `if (a && (b >= c)) { printf("OK\n"); }`
- Branch testing strategy:
  - One test is a=F, (b >= c) = T: a=0, b=3, c=2.
  - Produces different answers.

Now, for testing the branch we have to exercise elements of a conditional statement, but not just 1 and 1 false case. Because there are various combinations can really come up. So, takes an example let us say I have target statement is if a or b is greater than C this is the condition, I would like to execute this is the branch statement. And the very common mistake is this very common mistake is this.

Now, if we use a test and this test is important because you have to select the test data such that these two can be distinguished. Because you do not know, where in the code such an error has corrupt, in and you need to figure out exactly the data combination. So, that if there are such errors branch test error it should be possible to identify. Because there may be combination of inputs where, for both the cases you get the same results then it is not possible to figure out that this is an error which is there and this is the very common error for of the branch statements.

(Refer Slide Time: 52:05)

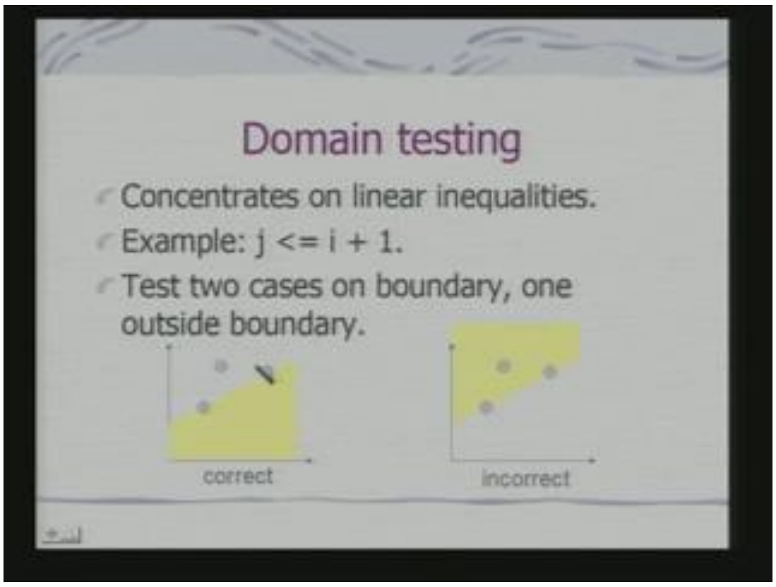


Another branch testing example

- Target:
  - `if ((x == var_pointer) && (x->field1 == 3))...`
- Actual:
  - `if ((x = var_pointer) && (x->field1 == 3))...`
- Branch testing strategy:
  - If we use only field1 value to exercise branch, we may miss pointer problem.

Another example if this is the pointed related error, this is an assignment and this is equality, see if you have this assignment the entire thing will go. And here also, if we simply check that is what, I said that if you only the field value then both the cases will give me identical results and these whole error will remain un detect. So, these errors are very critical errors they can be they are really problematic. So, your branch testing becomes that is why important then, other thing is domain testing when you really use inequalities.

(Refer Slide Time: 52:43)



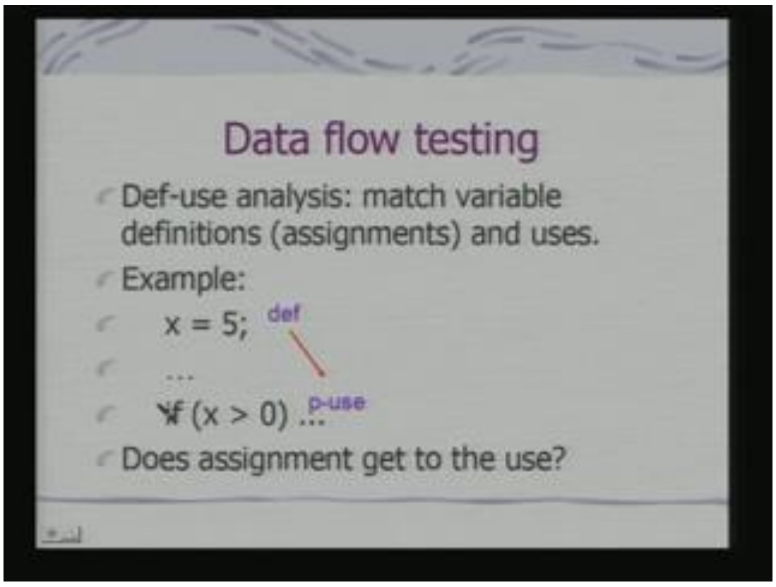
Domain testing

- Concentrates on linear inequalities.
- Example: j <= i + 1.
- Test two cases on boundary, one outside boundary.

correct
incorrect

So, typically what we say test two cases on boundary at 1 on outside the boundary because, if I have an example that less than equal to i plus 1 then, I need to do the test on the boundary because there can be various values of GNI and this is really defining a partition. So, the question is by putting in values of GNI i should check whether i am really at the boundary or i am outside the boundary or inside the boundary. So, says that you should test 2 cases on boundary and 1 outside the boundary.

(Refer Slide Time: 53:21)

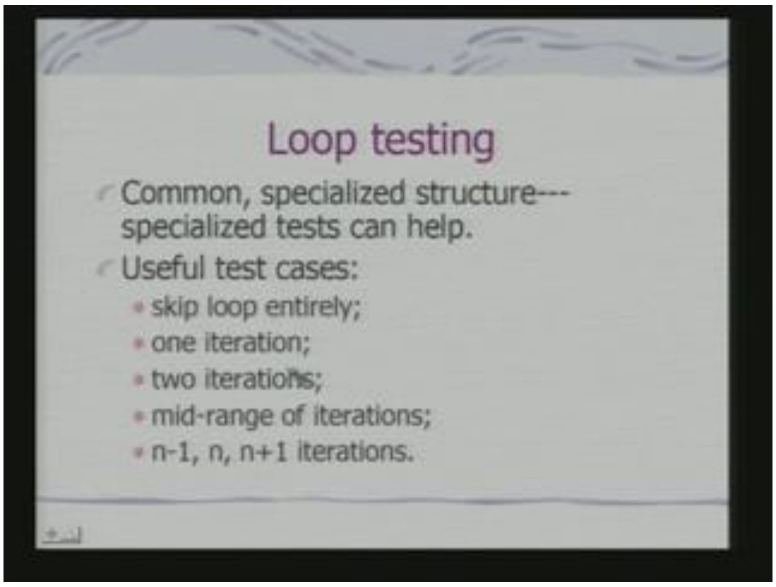


Data flow testing

- Def-use analysis: match variable definitions (assignments) and uses.
- Example:
- x = 5; def
- ...
- if (x > 0) ... p-use
- Does assignment get to the use?

This is domain testing, I am just giving you the basic flavor of domain testing domain testing means that, whenever searching equalities are there corresponding to the equalities you have to make a choice of your test patterns. So, if you look at the branch testing domain testing, we started we went in to that from the perspective of finding of paths to be covered and this testing schemes actually provide you with the exact nature of test patterns which is to be used.

(Refer Slide Time: 53:58)

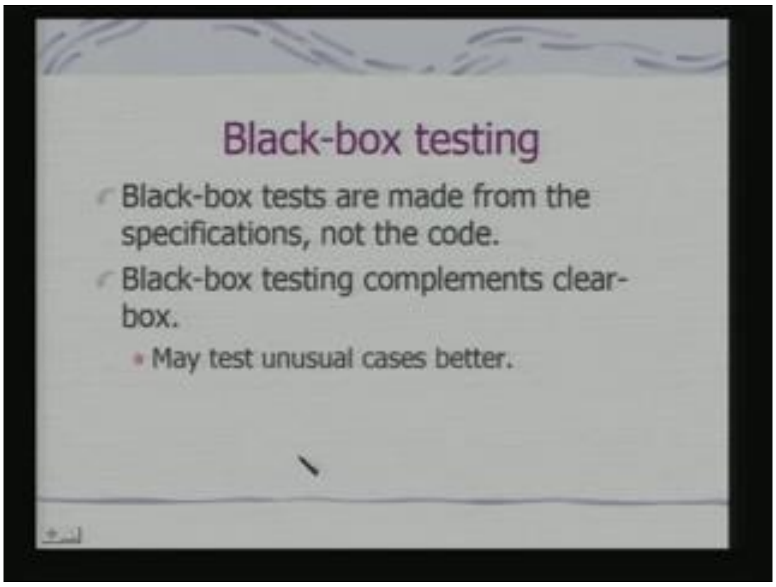


Loop testing

- Common, specialized structure--- specialized tests can help.
- Useful test cases:
  - skip loop entirely;
  - one iteration;
  - two iterations;
  - mid-range of iterations;
  - n-1, n, n+1 iterations.

So, data flow testing is this say if I am defining this variable am I using the variable correctly then, there are loop testing the loop is various conditions can happen with the loop you would like to skip the loop entirely. You have, would like to go through one iteration multiple iterations look at mid range properties and look at boundary n minus 1, if n is the boundary for loop n minus 1, n and n plus 1 iterations is it really the loop is going through n plus one; that means, there is an error if n is the boundary.

(Refer Slide Time: 54:27)

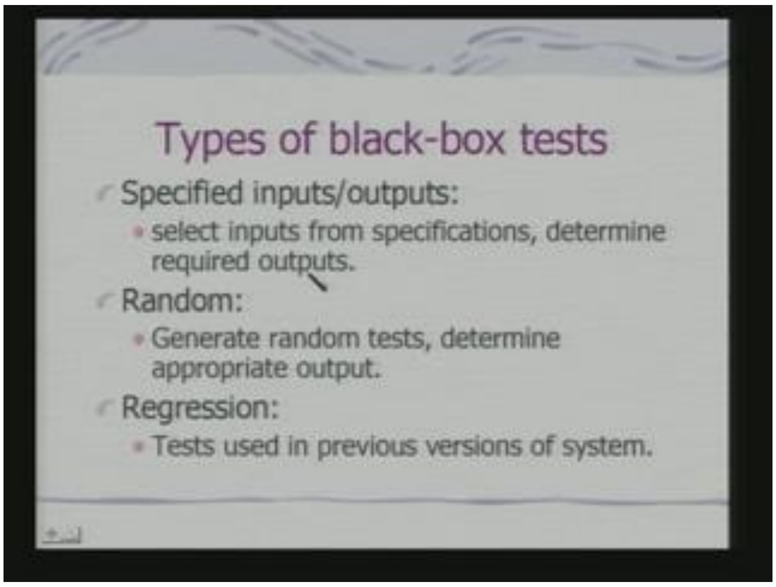


Black-box testing

- Black-box tests are made from the specifications, not the code.
- Black-box testing complements clear-box.
  - May test unusual cases better.

The black box testing I said that, make from specifications and not from the code. And complements clear box and typically should test un usual cases better ideally a person who is designing a black-box testing for an appliance should not know, how it has been implemented. You should only know it is complete specification and. In fact, black box testing can also on; actually gaps in the specification there can be missing points in the specifications. So, black box testing can an such gaps as well. So, there are various kinds of black-box testing specified input outputs.

(Refer Slide Time: 55:06)

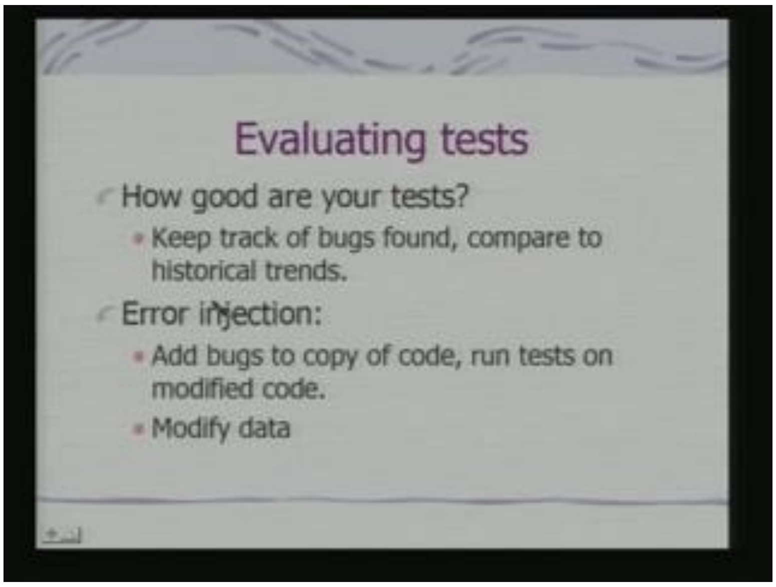


Types of black-box tests

- Specified inputs/outputs:
  - select inputs from specifications, determine required outputs.
- Random:
  - Generate random tests, determine appropriate output.
- Regression:
  - Tests used in previous versions of system.

So, select inputs from specifications and determine the required output from specification and see whether it matches. There can be random generate random tests and determine appropriate output. And there can be test used in previous versions of the system, if I generating an inversion it should be consistent previous versions.

(Refer Slide Time: 55:26)

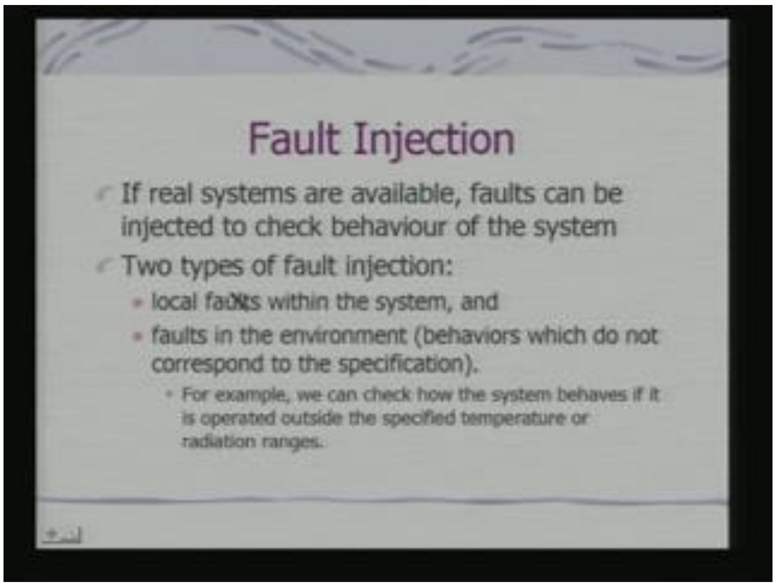


Evaluating tests

- How good are your tests?
  - Keep track of bugs found, compare to historical trends.
- Error injection:
  - Add bugs to copy of code, run tests on modified code.
  - Modify data

How do you evaluate tests? You evaluate tests, by keeping track of the bugs, but it is not always possible to know whether you have taken care of all possible conditions. So, what do you inject box.

(Refer Slide Time: 55:41)

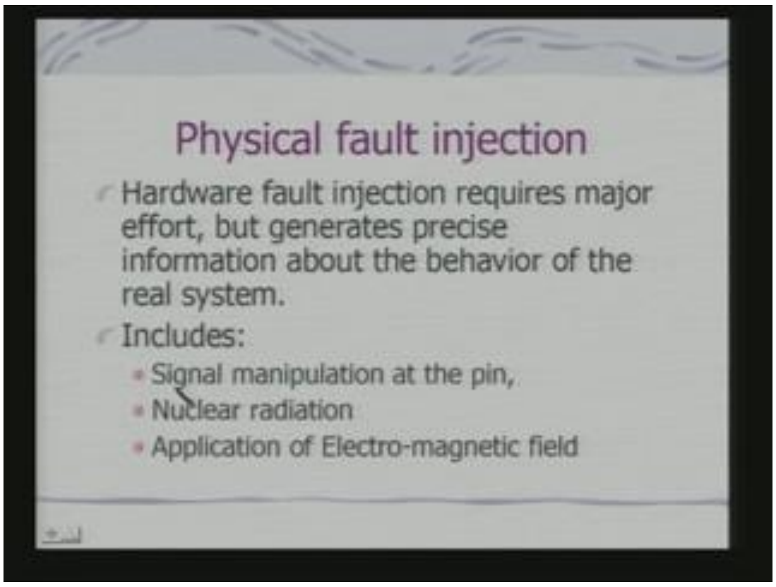


Fault Injection

- If real systems are available, faults can be injected to check behaviour of the system
- Two types of fault injection:
  - local faults within the system, and
  - faults in the environment (behaviors which do not correspond to the specification).
    - For example, we can check how the system behaves if it is operated outside the specified temperature or radiation ranges.

So, the basic idea is the injection of the box and two types of fault injections is there, local faults within the systems and faults in the environment. We can check when we are talking about faults in the environment, we can check how the system behaves if is operated outside the specified temperature or radiation changes.

(Refer Slide Time: 56:04)

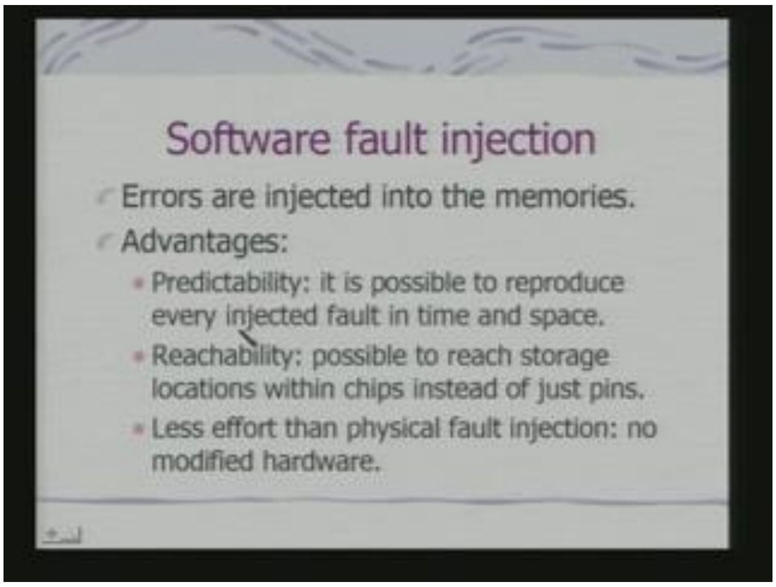


Physical fault injection

- Hardware fault injection requires major effort, but generates precise information about the behavior of the real system.
- Includes:
  - Signal manipulation at the pin,
  - Nuclear radiation
  - Application of Electro-magnetic field

So, typical hardware fault injection would mean signal manipulation at the pin. So, that means, what I am doing, I have designed the system. Now, I am trying to inject fault into the system you can manipulate signals at the pin you can apply electromagnetic field on nuclear radiation and see, these are all hardware kind of fault injection then see how the system behaves, then for fault software fault injections you can actually modify the code introduce path or modify the data. So, advantage is predictability it is possible to reproduce every injected fault in time and space because you know what the fault you are introducing.

(Refer Slide Time: 56:29)

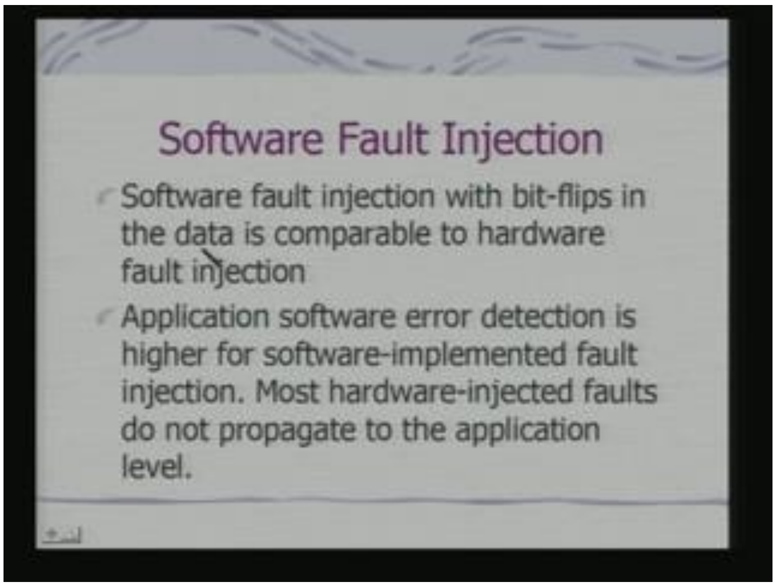


Software fault injection

- Errors are injected into the memories.
- Advantages:
  - Predictability: it is possible to reproduce every injected fault in time and space.
  - Reachability: possible to reach storage locations within chips instead of just pins.
  - Less effort than physical fault injection: no modified hardware.

So, you can reach storage locations within chips instead of just pins because this is not possible, if I am using only hardware manipulating the signals, I am not really the software code and it is less effort than physical fault injection.

(Refer Slide Time: 57:02)

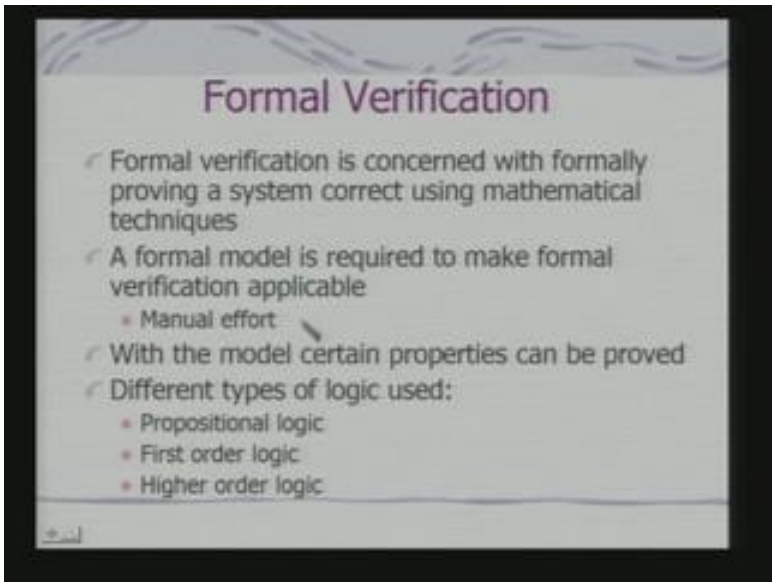


Software Fault Injection

- Software fault injection with bit-flips in the data is comparable to hardware fault injection
- Application software error detection is higher for software-implemented fault injection. Most hardware-injected faults do not propagate to the application level.

So, software fault injection with bit flips in the data is comparable to hardware fault injection. So, if you see typically like stocked faults would give you what bit flips. So, I can flip the data in the memory and see whether the system is working correctly or not what the behavior is whether it can be taking care of. And at an application level the software error detection is higher. So, one way of testing is you what we have talked about is, we have defined the testing schemes, we have defined the test patterns and to see whether your system gives the desirable behavior. You inject fault into the system because you can really expect the faults of the box to come up on your own always.

(Refer Slide Time: 58:03)



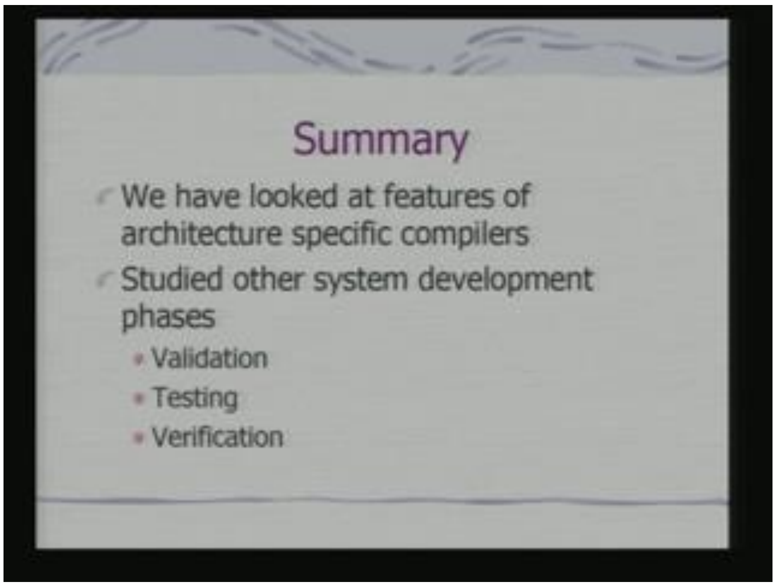
Formal Verification

- Formal verification is concerned with formally proving a system correct using mathematical techniques
- A formal model is required to make formal verification applicable
  - Manual effort
- With the model certain properties can be proved
- Different types of logic used:
  - Propositional logic
  - First order logic
  - Higher order logic

So, see that if you inject the fault whether that fault is getting detected by your testing process or not because you exactly know the faults that have been introduced. And then you can be sure that your testing process is correct and you would implementation is possible desirable. There also formal ways of doing it the formal verification is concerned with a absolutely regardless mathematical techniques. You can use formal logic, may be propositional logic first order logic or higher order logic to verify the system.

In this case what happens, would build the formal mathematical model of the system that is a manual effort and with respect to the mathematical model try to prove certain properties. If such properties are proven then you can say that your system is guaranteed to show that kind of a behavior. In fact, I have talked about the formal construction tools when we talked about validation.

(Refer Slide Time: 59:00)



Summary

- We have looked at features of architecture specific compilers
- Studied other system development phases
  - Validation
  - Testing
  - Verification

So, a construction tools would; obviously, implies that they are incorporating this kind of formal verifications schemes as well. So, what we have looked at today a different features of architecture specific compilers and studied the different system development phases, validation, testing and verification.