Embedded System-Design Verification and Test

Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar

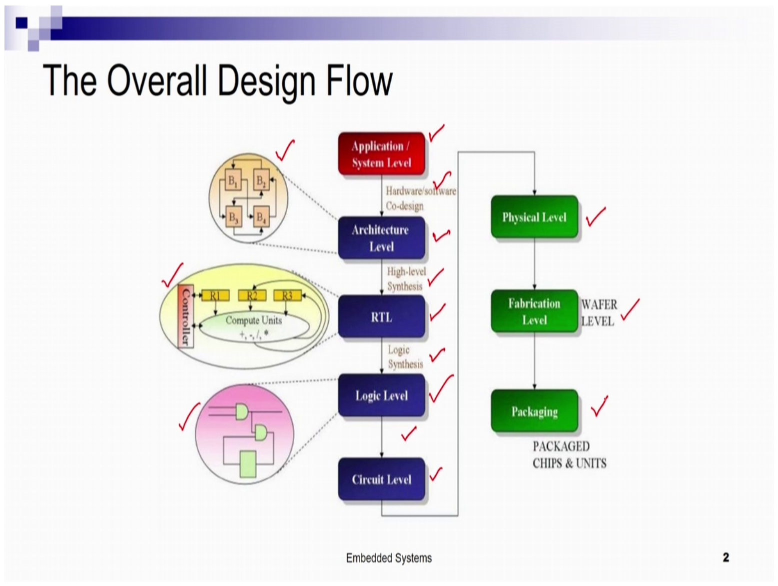
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture -04

Hardware Synthesis

Hello in this lecture we begin our study of Hardware Synthesis. Till now we have seen that at the beginning, we have an initial rough idea of the embedded system application that we want to design.

(Refer Slide Time: 00:52)



The Overall Design Flow

Embedded Systems 2

The first task is to specify that idea in English language as detailed as possible as accurately as possible, at specification that that allows us to tell what will be the inputs to the system, what will be the output from the system, how will the inputs will transform to the outputs. So, what will be the actual functionality that, we will all tell all these things we will tell in English. And then we have to also tell what are the performance requirements, what are the properties that this embedded system should satisfy, what are its performance requirements in terms of let us say the time within which the output should be produced, the energy that it is allowed to consume the size within which the system has to be built etcetera.

After we have this English language description of the system, we specify it formally using formal modeling tools, we saw formal computation modeling strategies such as the sequential program model and the finite state machine model in which the application can be modeled. Now, after this phase after this phase we have a task graph like structure of the entire application where each task, where each task node. So, it is a graph over task nodes where each task node is a program for example, the node corresponding it could be designed at various abstraction levels for example, we studied 8 CFSMs and the program state machines.

And let us say that each node could be a single program, or code in the program state machine each node. So, that node can for example, correspond to one node of the task graph at a certain level of granularity. So, if we see this figure first we have the application or system level and, then after the application of at the application of system or system level, we have modeled the entire application in terms of a task graph.

Now, this task graph on this task graph we have specifications in terms of its performance requirements as we told, the cost that it should take the cost that can be incurred for building this embedded system. And then others properties, based on that as we saw each task node in this task graph will either be built in hardware or software. So, this is called hardware software code design.

So, the entire embedded system application represented in terms of its task graph and, this and within this task graph each such task has to is will be built either in software and executed, as a C program let us say on a general purpose processor, or we will build a custom, or semi custom hardware corresponding to this task node. So, each task node will either be implemented in hardware or software.

Now, after we have decided which tasks will be implemented in software starts the hardware synthesis. Now, that we have decided that these stars will be implemented in hardware we have to design the hardware and implement it in the form of a chip. So, the today we will look at how a specification that is that, we want to implement as hardware will finally, be structurally realized in the form of a chip.

So, we see that from the application level we have this hardware software code design, where we partition the system in terms of what is to be implemented in software and hardware and, then for the hardware we first have the architecture level designer. In the architecture level design we have this task a task node, which we have to implement in hardware and this task node is represented in the form of a high level program, or our 8 CFSM let us say.

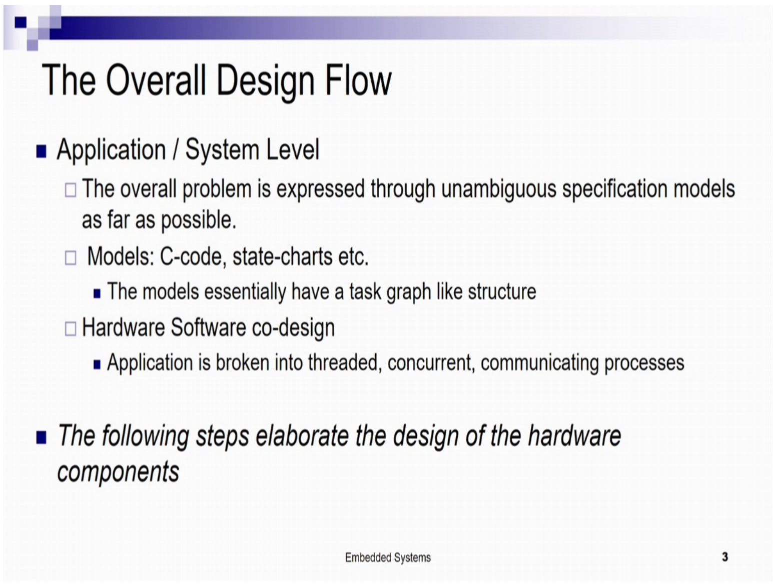
And, actually essentially what is it we have a set of concurrently communicating processes. So, each such process is a program it executes independently and concurrently with other processes and, has interfaces to talk with the other processes and with outside world. So, if you see this diagram here B 1, B 2, B 3, B 4 are 4 concurrent processes running concurrently with each other and, they can talk with each other through interfaces and they can also talk with the outside world.

Now, this set of programs this set of concurrently communicating processes, will go through a phase called high level synthesis, where it will be transformed in the form of an RTL, or register transfer level specification. And the structure of the (Refer Time: 06:26) the RTL will have a mathematical specification, in terms of what will be the register transfers at each time, we will talk in detail about that. And, it will have a truss structural representation as well in terms of registers functional units a controller that controls its etcetera, as shown in this elliptical enclosure ok. And after this RTL specification is obtained, we go through a process called logic synthesis, in which the RTL specification the register transfers are represented as Boolean equations. We also define the functional sorry, we also define FSMS corresponding to the controller and, then we realize this Boolean equations and FSM controller in the form of gates and flip-flop and that is that provides us the logic level design. So, this in encapsulation here, in this ellipse we see that we have represented the system that, we want to design in terms of gates and flip-flops.

Now, from this gates and flip-flops level design, we go through a process called technology mapping and we obtain a circuit level specification; from the circuit level specification, we this is a transistor level design. And from this we go to the physical level design, in which we obtain the geometrical structure of the circuit on the layout, or the layout of the circuit on the silicon wafer.

So, once the layout of the circuit so, how this transistor level design will be laid out on the silicon floor will be decided in the physical level design phase. And after this layout has been decided, we go for fabrication of this layout on the silicon chip and, we have a wafer level design and from that we have a finally, packaged chip. Going to a bit more within of this design flow we see that firstly, we have reiterating.

(Refer Slide Time: 08:49)



The Overall Design Flow

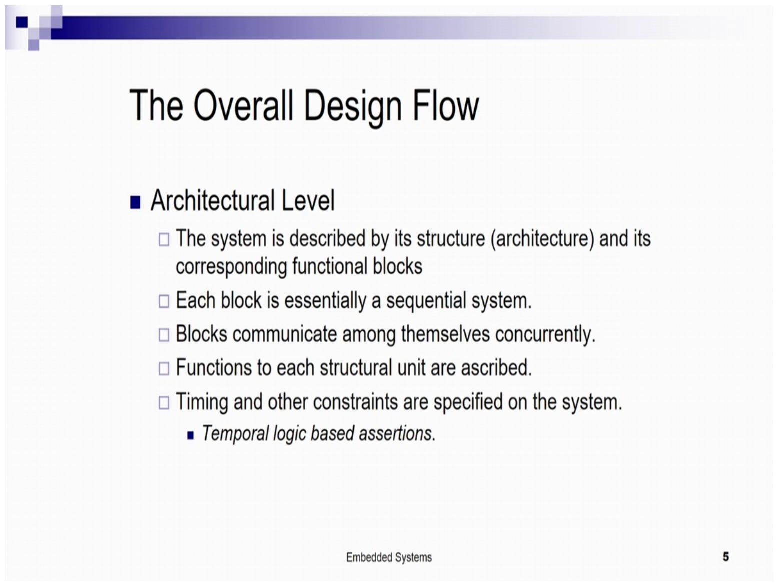
- Application / System Level
  - The overall problem is expressed through unambiguous specification models as far as possible.
  - Models: C-code, state-charts etc.
    - The models essentially have a task graph like structure
  - Hardware Software co-design
    - Application is broken into threaded, concurrent, communicating processes
- *The following steps elaborate the design of the hardware components*

Embedded Systems 3

Firstly, we have this application or system level design the overall problem is expressed through unambiguous specification models, as we just told. And the models includes a C-code which is a representation of sequential program model, sequential process model, and state charts which is a representation of the 8 CFSM, or hierarchical concurrent FSM models.

And the models are essentially models essentially have task graph like structure, with the tasks with concurrent processes concurrent communicating processes within them. And that after that we do a hardware software code design and the application is broken into threaded concurrent communicating processes, and for the for each of the task nodes that, we want to build as hardware. Then the following steps elaborate the design of the hardware components.

(Refer Slide Time: 09:55)



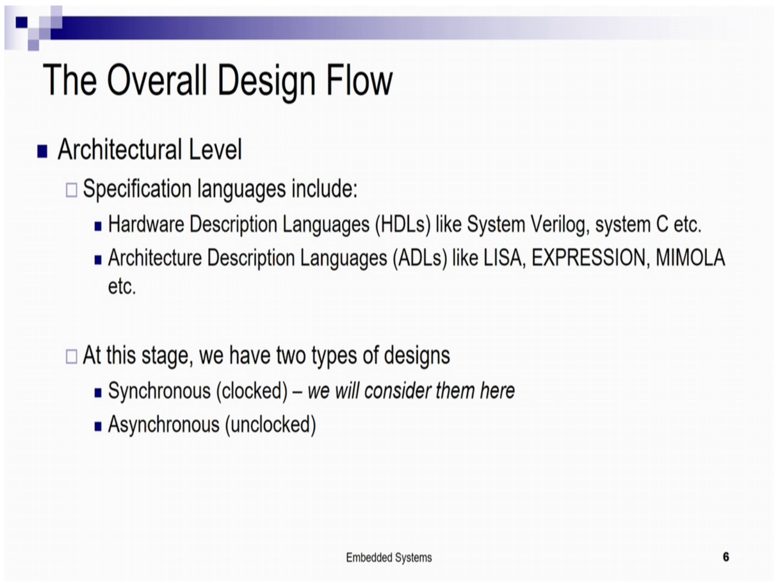
The Overall Design Flow

- Architectural Level
  - The system is described by its structure (architecture) and its corresponding functional blocks
  - Each block is essentially a sequential system.
  - Blocks communicate among themselves concurrently.
  - Functions to each structural unit are ascribed.
  - Timing and other constraints are specified on the system.
    - *Temporal logic based assertions.*

Embedded Systems 5

Now, the hardware design flow goes through a set of phases as I told, the first one of them was architectural level and the system is described by its structure and, its corresponding functional blocks. So, we have these concurrent communicating processes, each process is essentially conducts a function, carries out a function and these are the functional blocks. And, each block is essentially a sequential system, meaning that it will it will go time step after time step and conduct a series of operations. And the blocks communicate among themselves as we also said, the functions to each structural unit is ascribed, we will see how by designing the data path and control path for it and timing and other constraints are specified on the system.

(Refer Slide Time: 10:54)



The Overall Design Flow

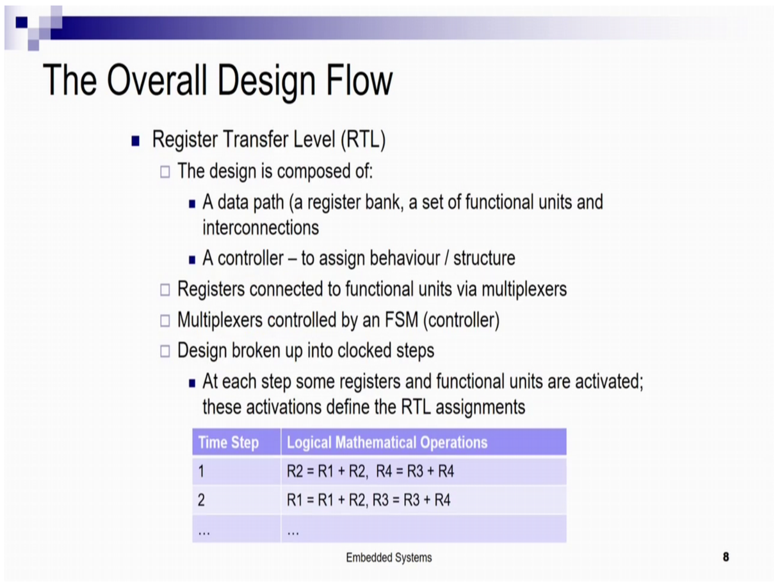
- Architectural Level
  - Specification languages include:
    - Hardware Description Languages (HDLs) like System Verilog, system C etc.
    - Architecture Description Languages (ADLs) like LISA, EXPRESSION, MIMOLA etc.
  - At this stage, we have two types of designs
    - Synchronous (clocked) – *we will consider them here*
    - Asynchronous (unclocked)

Embedded Systems 6

Now, the architectural level specification we said that we will have we will use models like C and 8 CFSMs as we told. The common languages that are used to model this in programs other than C are so, from that C specification for hardware it is most easy to use similar hardware description languages, which include which like system Verilog system C etcetera. From the C level specification it is easy to transform them into hardware description languages, or we could also have a bit at a higher level called architecture description languages like LISA, EXPRESSION, MIMOLA etcetera.

At this stage we can have two types of designs either synchronous, or clocked designs which we considered them here, or asynchronous, or unclocked designs which we will not consider in this course ok. So, from the architectural level we move to the RTL level.

(Refer Slide Time: 12:01)



The Overall Design Flow

- Register Transfer Level (RTL)
  - The design is composed of:
    - A data path (a register bank, a set of functional units and interconnections
    - A controller – to assign behaviour / structure
  - Registers connected to functional units via multiplexers
  - Multiplexers controlled by an FSM (controller)
  - Design broken up into clocked steps
    - At each step some registers and functional units are activated; these activations define the RTL assignments

Time Step	Logical Mathematical Operations
1	R2 = R1 + R2, R4 = R3 + R4
2	R1 = R1 + R2, R3 = R3 + R4
…	…

Embedded Systems 8

And at the register transfer level what do we have the design is composed of a data path and a controller. So, we said that essentially any hardware, that we designed has a logically structural view what is it is logically structural view composed of a data path, and a control path. And a controller the data path consists of register banks functional units and their interconnections.

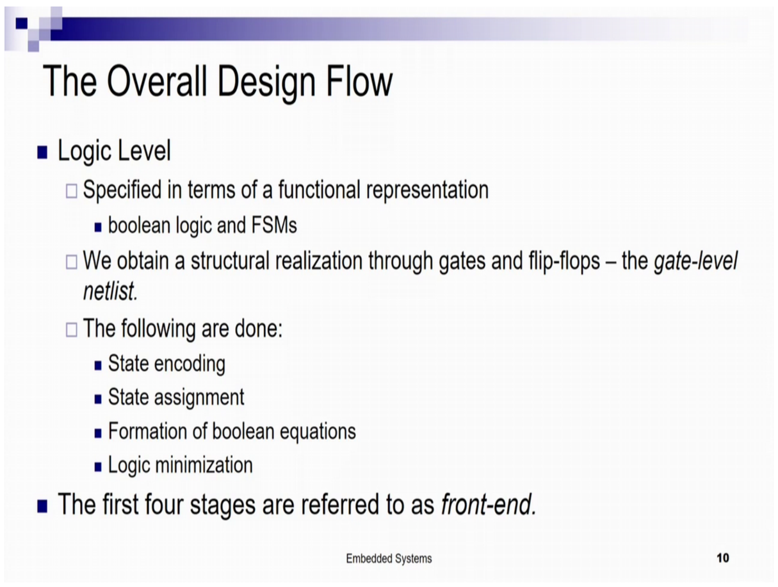
So, the interconnections will have wires MAXs D MAXs etcetera and the controller so, what will this data path do. So, data path will transform input data and it will allow certain data in certain ways to be to be transferred to functional units get, and will get transformed and then stored again back into registers. So, from inputs through the functional units back to registers this path is provided by, this data path that is why, because this path transforms data from input to output this is called a data path.

And, how the data path will transform the data is controlled by the controller; that means, the for example, the contents of which register; will go to which functional unit at a given time step is controlled by the controller. And then the output of the functional unit will go to which register is again controlled by the controller as we will see. So, this data path and controller essentially constitutes the register transfer level, level specification structural specification of the design.

So, registers are connected to functional units of higher multiplexers, the multiplexers are controlled by FSM. So, the select lines of the multiplexers will be controlled by FSM and therefore, the multiplexers will be able to choose will be able to choose which register contents will go to which functional unit at a given time. So, the design after at the RTL level the design is broken into clocked steps at each step some registers and functional units are activated and these activations define the RTL assignments for example, at time step 1. So, suppose we can have RTL specifications like this at time step one, perform the following register transfers. So, R 1 plus R 2 equals to R 2 equals to R 1 plus R 2 so, take the contents of a time step 1 take the contents of R 1 and R 2 pass, it to an adder and, then the output from the adder should be passed back to the register to R 2.

Similarly, also performed the same function also performed the function R 3 plus R 4. So, contents of R 3 and contents of R 4 should be transferred to the adder and the output of the adder should pass should be passed to R 4. So, this also should be done in time 1 and, then at time step 2 similarly other things other register transfers will be done. So, this is how the register transfer level specification is designed. So, after register transfer level design we go to the logic level.

(Refer Slide Time: 15:37)



The Overall Design Flow

- Logic Level
  - Specified in terms of a functional representation
    - boolean logic and FSMs
  - We obtain a structural realization through gates and flip-flops – the *gate-level netlist.*
  - The following are done:
    - State encoding
    - State assignment
    - Formation of boolean equations
    - Logic minimization
- The first four stages are referred to as *front-end.*

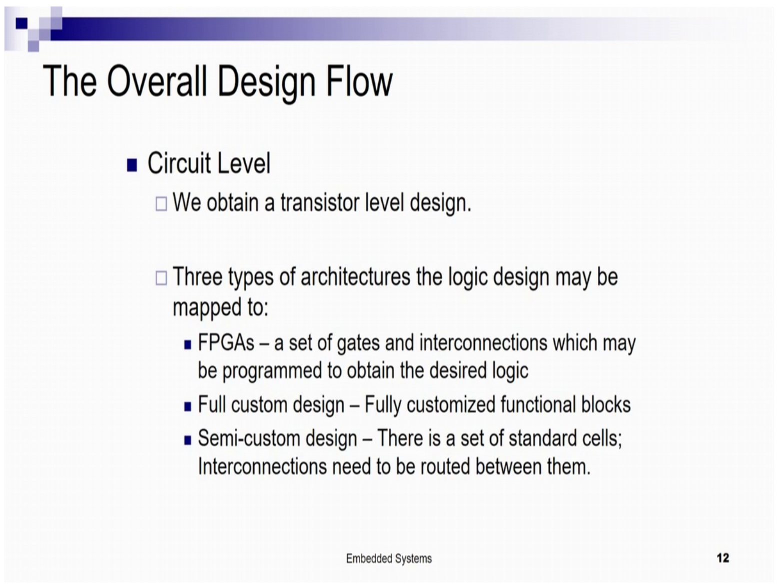
Embedded Systems 10

So, at the logic level what happens the application, or the m the hardware design is specified in terms of its functional representation. So, what is a functional representation as I told you from the RTL specification, we build the Boolean logic, or the Boolean equations and FSMs for the controllers. And then what do we obtained through the logic synthesis phase, we obtain a structural realization of this Boolean logic and FSMs in the forms of gates and flip-flops and this is called the gate level netlist.

So, what are the important functions that are done so, we do a state encoding we do state assignment for the FSM state encoding and state assignment. And then we form the Boolean equations. And after we have done the Boolean equations, we do logic minimization both two level and multi level logic minimization is performed to obtain optimized Boolean logic and, then this optimised Boolean logic is transferred to its gate level netlist through the synthesis phase.

The first these first 4 stages of the design is called the front end. So, architectural level the RTL level and the logic level. So, from the application level if we come so, them from the application level the architectural level the RTL level and the logic level, these are the front end of the design. And then comes the back end which is this, the circuit level the physical level the fabrication level and the packaging. So, these are the back end of the design.

(Refer Slide Time: 17:19)



The Overall Design Flow

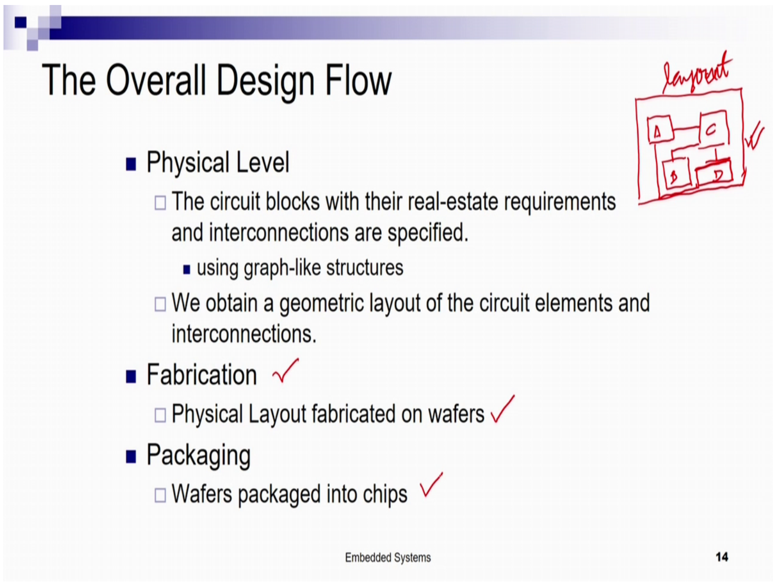
- Circuit Level
  - We obtain a transistor level design.
  - Three types of architectures the logic design may be mapped to:
    - FPGAs – a set of gates and interconnections which may be programmed to obtain the desired logic
    - Full custom design – Fully customized functional blocks
    - Semi-custom design – There is a set of standard cells; Interconnections need to be routed between them.

Embedded Systems 12

So, after the gate level netlist is obtained we told that we do a technologically technology technology mapping and, we obtain the corresponding transistor level design which is called the circuit design. And various types of architecture three types of mainly three types of architectures are possible for the logic design, meaning the logic design can be mapped to the circuit level in three types of architectures.

So, it can be mapped in terms of FPGAs. So, the logical representation of FPGAs is a set of gates and their interconnections which may be programmed to obtain the desired logic, we can have full custom design where we have fully customized functional blocks. We can also have semi custom design, where we have a set of standard cell cells and the interconnections among the standard cells needs to be routed to obtain the functionality that we want to have. After circuit level comes the physical level design.

(Refer Slide Time: 18:23)



The Overall Design Flow

- Physical Level
  - The circuit blocks with their real-estate requirements and interconnections are specified.
    - using graph-like structures
  - We obtain a geometric layout of the circuit elements and interconnections.
- Fabrication ✓
  - Physical Layout fabricated on wafers ✓
- Packaging
  - Wafers packaged into chips ✓

layout
A
C
B
D

Embedded Systems 14

At the physical level the circuit blocks with the real estate requirements and interconnections are specified. So, once we have the transistor level design we have a fairly good idea as to what will be the area taken by this design on the FPGA floor ok. So, then once we have an idea of the amount of space that it will take on the FPGA floor, we have its real estate requirements estimation.

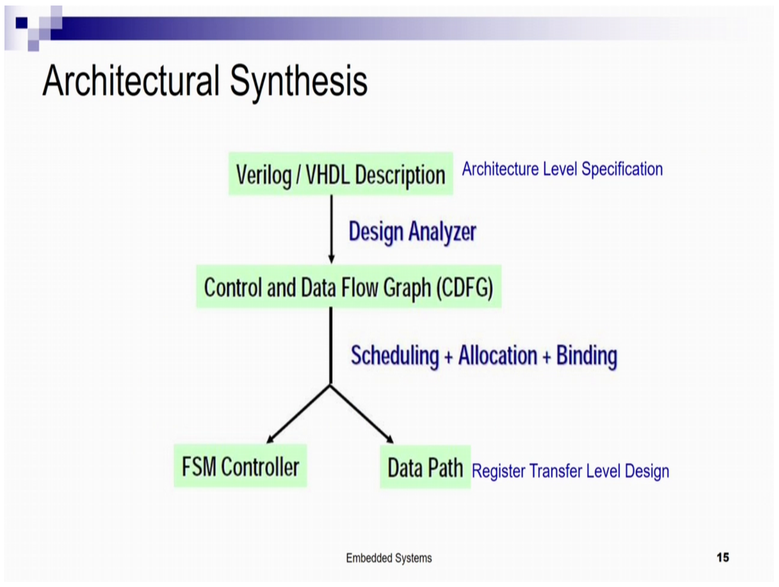
And we when we have the real for the different types of so, the circuit will have different say gate blocks and each blocks of gates for different connected functions. And each such connected function will have its real estate requirements known, once we do the technology mapping and do the circuit level design. And then and then once the real-estate requirements are of these blocks are known we also know at in addition to the blocks the interconnections among these blocks.

So, the blocks and their interconnections together specify the functionality right. So, once the realistic requirements of the blocks themselves, which are for the circuit blocks themselves and their interconnections are known, we represent them again as graph like structures called necklace. We are and the output what do we obtain as output from the physical level design, we obtain a geometric layout of the circuit elements and their interconnections ok.

So, we will have so, after the gate level design is done at the and after the gate level design is done, we obtain the circuit level we obtain each block of circuit and for each block of circuit, we have obtained the real estate requirements. And these so, these will be the circuit blocks A B C and D let us say, these are the circuit blocks these are the circuit block. And the interconnections among these circuit blocks are also known. So, once this is known we can have a layout like this.

So, we will have a layout like this. Now, this layout is the g this is what is the geometric layout and is the output of the physical level design, such as geometric layout is the output of the physical level design. Now, once we have a layout like this layout can be fabricated, this layout can be fabricated on the silicon wafers. And once we have the fabricated silicon wafers it must be packaged into chips at the packaging level.

(Refer Slide Time: 21:23)



Architectural Synthesis

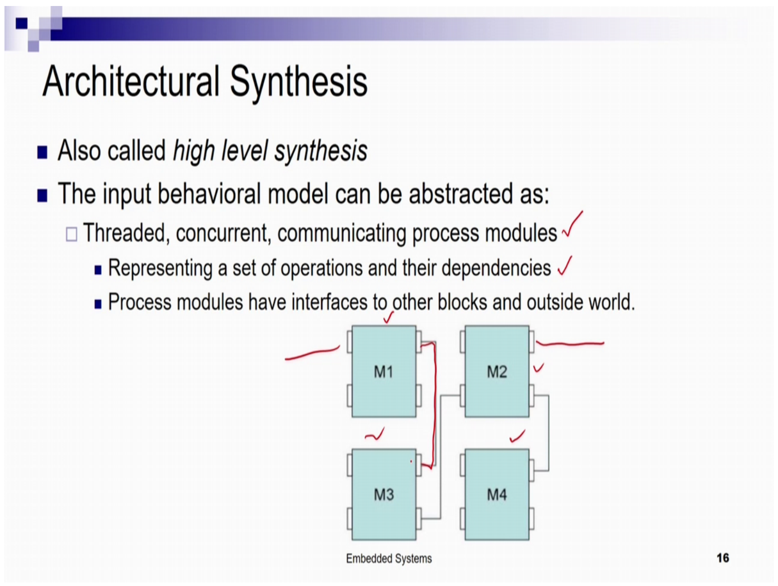
Verilog / VHDL Description
Architecture Level Specification
Design Analyzer
Control and Data Flow Graph (CDFG)
Scheduling + Allocation + Binding
FSM Controller
Data Path
Register Transfer Level Design

Embedded Systems 15

Now, after we had this overall hardware synthesis steps after, we have a overview of the overall hardware synthesis steps, we will take a deeper look into architectural synthesis that is the step that takes the design from the architectural level to the RTL level ok. So, we said that at the beginning of the architectural synthesis step, we will have a set of concurrently communicating processes. And these concurrently communicating to each such concurrently communicating process will be represented in terms of a sequential program model and the tool, or language that it is commonly used for hardware, hardware design in this sequential program model is Verilog VHDL etcetera. So, this so the programs for each process each communicating process will be specified.

Now, once this program the Verilog VHDL description is known, we have a pro another program called the design analyser, which transforms this specification into a control. And control and data flow graph representation, or CDFG representation which we will look at what it is the this control and CDFG represent control and data flow graph representation is then transformed into the FSM controller and data path, at the RTL level through a set of processes called scheduling, allocation and binding ok, which we will also look today. What are they yes architectural synthesis is also called high level synthesis.

(Refer Slide Time: 23:17)



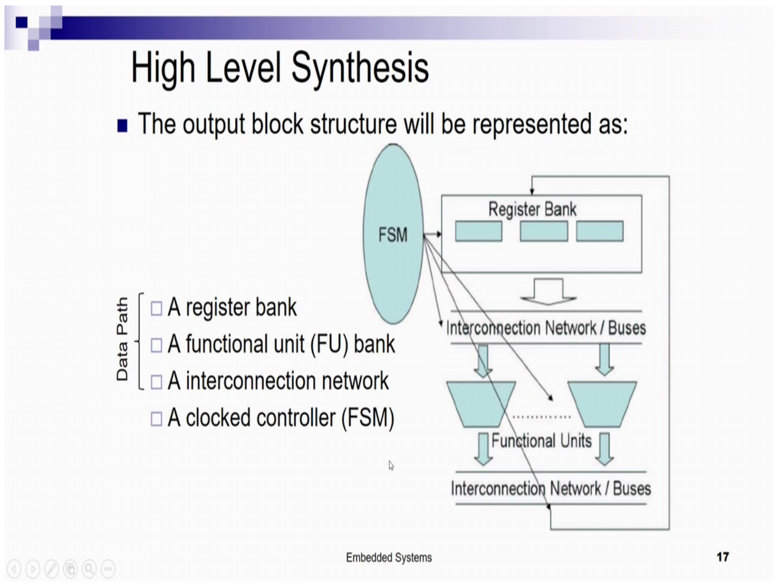
Architectural Synthesis

- Also called *high level synthesis*
- The input behavioral model can be abstracted as:
  - Threaded, concurrent, communicating process modules ✓
    - Representing a set of operations and their dependencies ✓
    - Process modules have interfaces to other blocks and outside world.

Embedded Systems 16

And the input behavioural model can be abstracted as we said threaded concurrent communicating process modules and, these are the models M 1, M 2, M 3, M 4 are 4 concurrent process modules. And they represent the set of operations and their dependencies, process modules have interfaces to other blocks and the outside world. So, these may be connected to the outside world these are connected to other blocks.

(Refer Slide Time: 23:47)



High Level Synthesis

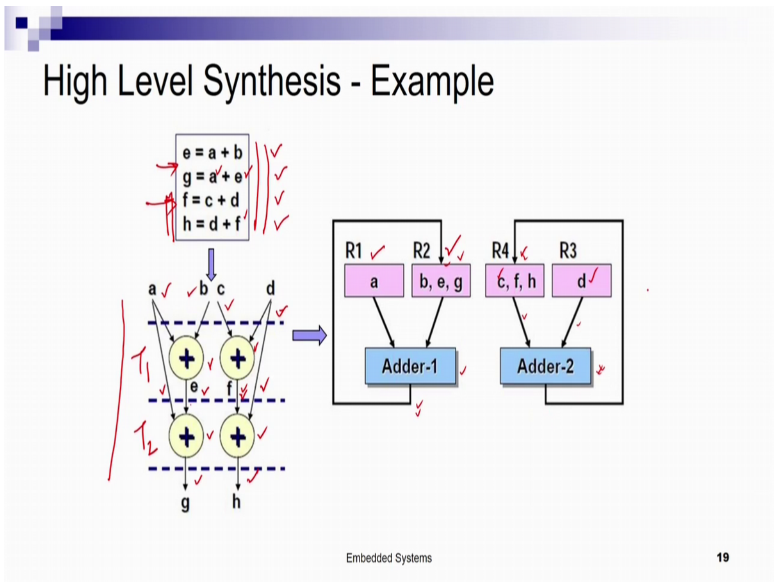
- The output block structure will be represented as:
  - Data Path
    - A register bank
    - A functional unit (FU) bank
    - A interconnection network
  - A clocked controller (FSM)

FSM
Register Bank
Interconnection Network / Buses
Functional Units
Interconnection Network / Buses

Embedded Systems 17

Now, what is the output so, the input was a specification like that and, what do we get as output at the RTL level the output is a block structure, which is represented as we have a set of registered banks containing data. And the data in the register bank are transformed to a set of functional units, through an interconnection network of buses and the output of the functional units are further transferred back to the register banks ok. So, we have a data path consisting of register bank functional unit bank and interconnection network and, we have an FSM controlling the all the all the data transfers within the data path this is the output.

(Refer Slide Time: 24:31)



High Level Synthesis - Example

Embedded Systems 19

So, now let us take an example let us say that we have a set of statements, we let us say our VHDL Verilog statements some high level, high level language set of statements are here e equals to a plus b g equals to a plus c, f equals to c plus d and h equals to d plus f, these have to be implemented in terms of a hardware circuit. So, we have to do a hardware synthesis corresponding to this corresponding to this small example high level code ok.

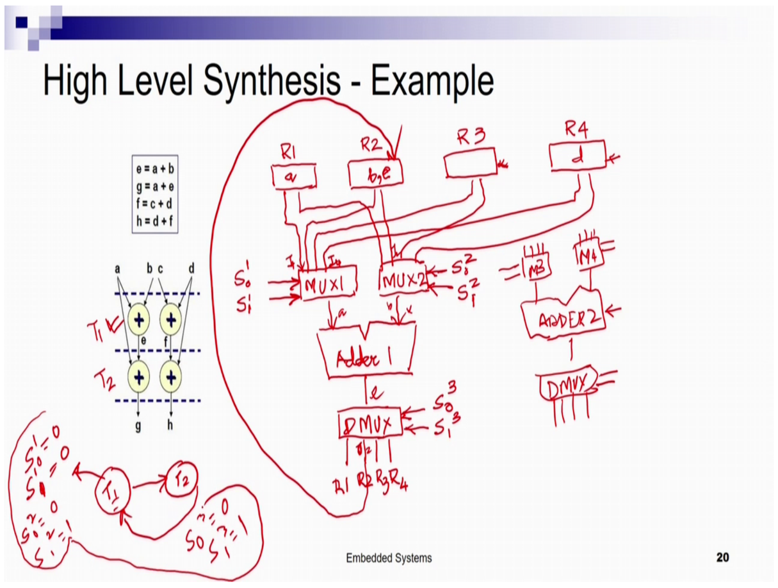
Now, first this code is transformed to a schedule like this, this schedule has 2 time steps T 1 and T 2, how this transformation is done we will see and it transfers. And, it transforms it in this way so firstly, if we take the first statement we see that a and b are passed through an adder and the output generated is e at time step 1 and at time step 2 because e is generated at the end of time step 1. So, the at time step 2 what we have is that, this e and a are then added this e and this a are then added, in this adder and the output g is produced at time step 2.

Similarly, c and d statement three here, c and d are passed through the adder and this output f is produced. And at the next time step f and d so, f and d are added in this adder and the output h is produced ok. So, this is the schedule of operations that we have and, what is the kind of arc RTL level representation in terms of the data path that we realize. So, in we stored in R 1 a and we store b in R 2 and then we pass it to adder 1, and the output of adder 1 goes to goes back to R 2 and e is produced. And after e is produced e and a is again added in adder one and the output g is produced which is stored in R 2 ok.

Similarly, for the second adder here for this for the second set of statements 3 and 4 for statements three and 4 these two statements, what we have is the following we have c in R 4 and d in R 3. And, these two are added in added to and the result is passed back to R 4, which is f f and d are then added again f and d are then added in a register in adder two at time step 2. And the result is stored back in h.

Now, we will look at this example in a bit more detail to understand what are the sequence of control signals that are generated so, that the required timing can be ascribed to it, and the register transfers correctly do happen, for the example that we have here.

(Refer Slide Time: 28:10)



High Level Synthesis - Example

So, let us say we have the same set of 4 statements and, we have this schedule T 1 and T 2 this is what we have. And, how is this implemented. Firstly, we have four registers we have four registers R 1, R 2, R 3, R 4. And initially in R this one is R 1 R 2 R 3 and R 4. So, initially we have a in R 1 say b in R 2 and b in R 2 and let us say c in R 3 and d in R 4 ok.

Now, we said that and let us say the functional unit we have two functional units 1 of them is adder 1, this is say adder 1, adder 1, we said that the input to the adders are the are connected through MUXs this one is MUX 1 and let us say this one is MUX 2, this one is MUX 2. So, this one is 1 input to the adder this one is the other input to the adder. And, we also said that the MUXs the registers will be connected to the functional units via MUXs. So, R 1 will be connected to this one R 2 will be connected R 3 will be connected and R 4 will also be connected ok.

Similarly, for the MUX 2 also the same thing is going to happen ok. Now, MUX 1 is let us say controlled by two select inputs S 0 1 and S 1 1. And MUX 2 is also connected through two inputs S 0 2 2 select inputs and S 1 2 ok. So, these are the two select inputs corresponding to MUX 2. And say that R 1 this input, this one is input I 1 of MUX 1 and this one is input I 4 of MUX 4. So, I 1, I 2, I 3, I 4, I 1 is selected when the select inputs are 0 0. So, R 1 will be selected when the select input is 0 0, R 2 will be selected when the select input is 0 1 4 MUX 1 likewise.

Similarly, for MUX 2 so what we want to say is here is that MUX 1 will allow the contents of R 1 to pass to adder 1, if the select inputs are 0 0 corresponding to MUX 1. Similarly we also have a D MUX, we have a D MUX at the output and, that will also be controlled through select inputs say S 0 3 and S 1 3 and, it will have 4 outputs going to say R 1 R 2 R 3 and R 4 ok.

Now to implement this operation this operation e equals to a plus b let us say that what we want is what we want is that the output e should be should be stored in register R 2, why because the input this data in b this data b is no longer required after time step 1; however, the data input a is required again at time step 2. So, there is no problem, if b is overwritten by e at the end of time step 1.

So, now our controller the specification of our controller would be actually a counter, which has two states which has two steps and this one is time step 1 and this 1 is time step 2. So, our FSM is accounted with two steps and the outputs will be if when it is at time step 1, it will produce the outputs what will be the outputs the outputs will be for time step 1 the outputs will be S 0 1 equals to 0 S z z S 1 sorry S 1 1 equals to 0 and for MUX 2. So, why is as 0 one equals to 0 and S 1 1 equals to because at time step 1, we want the input in R 1 to come here we want a to come here at time step 1.

So, therefore, this will be selected for R 2 sorry for the other input of adder 1, we want the value of R 2 which is b to come into the other input of adder 1 and, that will be done how that will be done by having S S 0 2 equals to 0 and S S S 1 2 equals to 1, why because this corresponds to the second input I 2 of MUX 2 and I 2 of MUX 2 will be selected when the select inputs are 0 1.

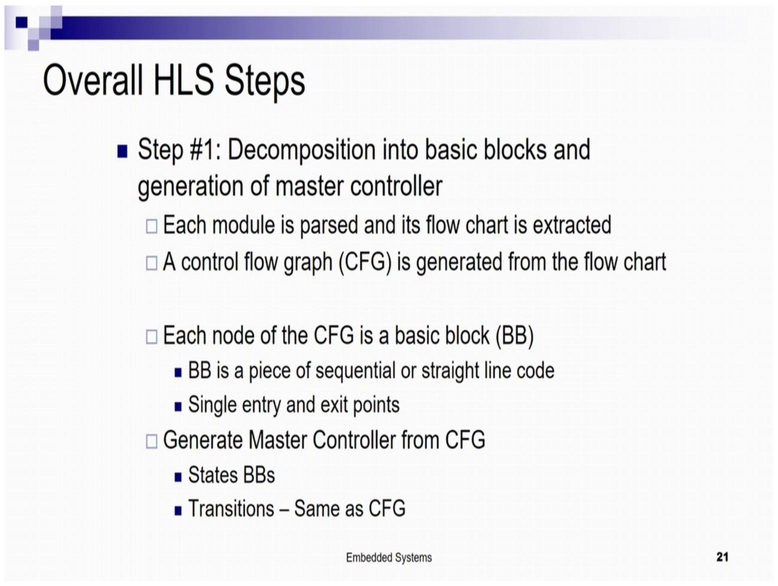
So, at time step 1 R 2 the contents of R 2 can go to adder 1 if select inputs are 0 1. And after that we have said that let us say this one is the output 2 of the D MUX O 2 of D MUX and we want the output 2 of D MUX, because it goes to register two we want the output 2 2 of D MUX 2 go to meaning that this to be selected, we want the second output of the D MUX to be selected. And therefore, at T 1 the select inputs S 0 3 equals to 0 and S 1 3 equals to 1 should be selected, because when the D MUX select inputs are 0 1, the second output of the D MUX will be selected and the output will be correctly passed to register R 2.

Similarly, same thing will happen for the other adder. So, we have I have not drawn the full there will be another adder adder 2 and, for that it will have again the MUX inputs MUX 3 and MUX 4. And it will similarly have inputs from the 4 registers and, it will appropriately allow correct inputs to pass. And it will have another D MUX and each of them will have select inputs each of them will have select inputs and, also the functional units can have activation inputs and registers can have load inputs as well.

So, these inputs can also be controlled by the FSM so, the FSM essentially will select all the correct control signals which are the select lines load inputs active activation inputs, or functional units etcetera and it will correctly activate correct data transfer from correct registers through the correct functional units, through the through the correct D MUX outputs back to the registers. So, this will be controlled by the FSM controller. So, these will be the this will be the set of control signals that will be output at state one of the controller this is how the design will be realized and controlled.

Now, when we have seen the when we have seen the architectural synthesis overview, we will see the steps in a bit more detail. So, at the end of the as we said at the beginning of the architectural synthesis step, what we have are concurrent communicating processes and each process is a high level language program let us say. And then this high level language program is transformed by the design analyzer into CDFG, or control and data flow graph.

(Refer Slide Time: 37:54)



Overall HLS Steps

- Step #1: Decomposition into basic blocks and generation of master controller
  - Each module is parsed and its flow chart is extracted
  - A control flow graph (CFG) is generated from the flow chart
  - Each node of the CFG is a basic block (BB)
    - BB is a piece of sequential or straight line code
    - Single entry and exit points
  - Generate Master Controller from CFG
    - States BBs
    - Transitions – Same as CFG

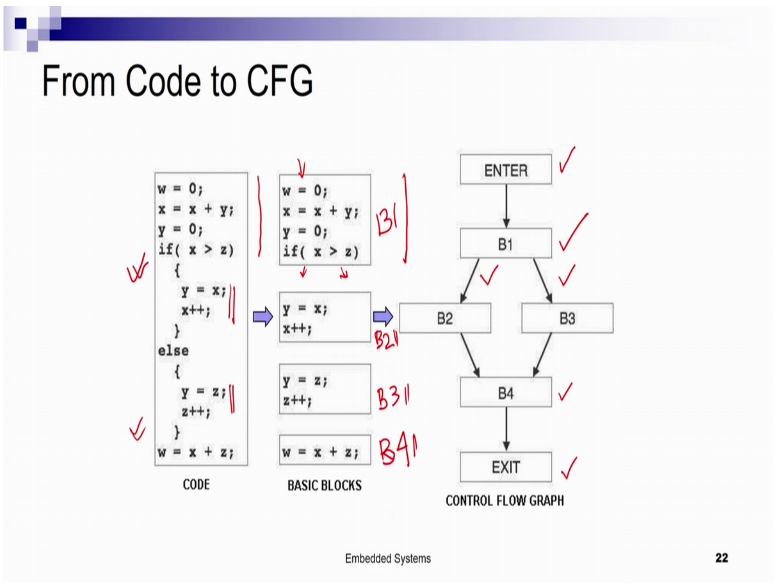
Embedded Systems 21

So, first we look at what a control flow graph is so, first step in high level synthesis is the decomposition into basic blocks and generation of the master controller. Now, what are these so, each module is parsed and it is flowchart is extracted. So, the program that we have of each common concurrent communicating process so, each process is a program and for that process we obtain the flowchart and from that flowchart we obtain the control flow graph.

So, what is this control flow graph? Each node of the control flow graph is a basic block. So, what is a basic block? A basic block is a piece of sequential or straight line code with a single entry point and single exit point. So, we look a bit more deeply into what a basic block is in the next slide and, then once we have disintegrated the entire functionality in terms of its basic blocks and interconnections between these basic blocks realizing a control flow graph, we have a master controller we can generate a master controller for this program.

So, what will be the master controller of this program the master controller can is design from the C CFG and the states of the master controller are basic blocks themselves and the transitions are same again as the CFG.

(Refer Slide Time: 39:22)



From Code to CFG

```
w = 0;
x = x + y;
y = 0;
if( x > z)
  {
    y = x;
    x++;
  }
else
  {
    y = z;
    z++;
  }
w = x + z;
```

CODE

```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

B1

```
y = x;
x++;
```

B2

```
y = z;
z++;
```

B3

```
w = x + z;
```

B4

BASIC BLOCKS

Embedded Systems 22

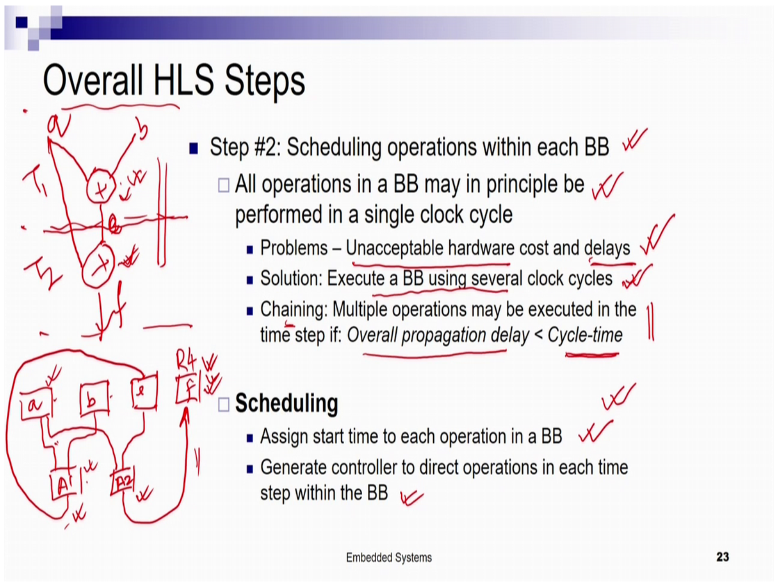
So, now we see how a particular high level code of a Verilog out of VHDL code need, we take an example of C here a C code can be can be disintegrated. So, we will take the high level code and transform it into HCFG. So, for CFG as we said first we have to find out the basic blocks. So, this program over here has 4 basic blocks B 1 B 2 B 3 and B 4 and B 4, and how are these basic blocks obtained see that the first 4 lines of this high level code form the first basic block why, because this is a piece of straight line code which are single entry point and single exit points.

Once the program control enters into the first statement w equals to 0, there is no other way that it can exit other than the last after the last statement. So, once w 0 is a is executed once w is control enters into w 0, we know that up to the f statement it is always going to be ex executed. So, if w 0 is executed then x equals to x plus y must be executed y equals to 0 must be executed and, if x greater than z this condition check must be executed.

Now, after that the control breaks so, we do not have a straight line piece of code anymore and, we have two exit points correspond from out of this basic block why, when the if condition is true and the other when the if condition is false. Now, we have said when the, if condition is true these two statements are executed, which is again a straight line piece of code and this forms basic block 2. And the other basic block is y equals to 0 z plus this is again a straight line piece of code becomes basic block 3 and finally, after the join we have another statement which is which forms the last basic block, because this one contains only one statement with a S this is the only straight line piece of code that we have for this basic block.

Now, what is the structure of this control flow graphs we enter the control flow graph, we first go into basic block 1 after executing basic block 1, we either go into basic block 2 or then our basic block 3. And then we have basic block 4 after the execution of basic block for we exit. So, this is the control flow graph representation of the high level program that we have here. So, in the first step in architectural synthesis, the in the input process input program is transformed. So, each concurrent communicating process is transformed to its corresponding control flow graph. After the control and data flow graph is obtained we schedule operations within each basic block. So, from the control flow graph we now go within each basic block.

(Refer Slide Time: 42:30)


Overall HLS Steps

- Step #2: Scheduling operations within each BB
  - All operations in a BB may in principle be performed in a single clock cycle
    - Problems – Unacceptable hardware cost and delays
    - Solution: Execute a BB using several clock cycles
    - Chaining: Multiple operations may be executed in the time step if: *Overall propagation delay* < *Cycle-time*
  - **Scheduling**
    - Assign start time to each operation in a BB
    - Generate controller to direct operations in each time step within the BB

Embedded Systems 23

And we say that all operations in a basic block may in principle be performed in a single clock cycle, what do we mean by this statement. Let us consider the example that we had just seen a while back. So, we had this a b e f so, we had this a b e f. And we said that we had done it previously in two clock cycles T 1 and T 2. So, could we have and so, that required two time cycle two time cycles, but we are saying that all operations in a basic block may in principle be performed in a single clock cycle.

Could we have done this in the same clock cycle, with the resource constraints that we had over there, we could not have done it why, because we were having only 1 adder and we had only 2 registers; however, suppose if we had 4 registers R 1 R 2 R 3 R 4 and we could say something like this. And if we have 2 adders adder, 1 and adder 2, then we could do we could say something like this we could say that transfer a this one and, then you this output let it go into e this output may in parallel may go into e, and the output e comes here and then the output goes here.

Now, if we have we do not have register we do not have let us say resource constraints like this we have 4 register possibly, we could have done it with 3 registers as well, but we needed at least 2 adders and we have 2 adders. Then what is the advantage do we have so, a and b a a and b passes their outputs no matter, when we start the time step 1 and everything is done in time step 1, how at the beginning of time step 1 a and b is transformed quickly into A 1 and the output is produced and e is e the e goes into register

1 and, then the output of this output from this register goes into A 2 and finally, the output of A 2 goes into this register ok.

Now, what are what is the constraint here constraint are the delays of these registers and functional units and the MUXs and D MUXs that we have, now if the delays if the summation of these if we have a clock cycle which is greater than the summation of the delays of these registers MUXs functional units etcetera. Then we basically after the start of the time cycle, we are we are waiting for a sufficiently long time and we will see that the output has correctly come into let us say if this is register output the output has correctly come into register R 4. If we have waited for a sufficiently long time such for a long time that a time duration, which is more than the cumulative delays of the registers and the functional units and MUXs and D MUXs which come in the data path processing ok.

So, we are limited by the delays of the of the of the resources, but we do not have any constraints on time, because we do not have any resource constraints. However, if we have this sort of a design in which we are reusing this adder, we cannot do it in a single time step why, because we do not know we do not know we need to have a crisp time at which we will have this e value output and this output will be stored in a register by the end of time step 1. So, that this output we will have a stable output at the beginning of time step 2 and we will produce this f ok.

So, otherwise the we could have a corrupted result, because the output e will be produced and this output will go back to the same functional unit and it may interfere with a and b which were which were the previous contents, we will have we could have several problems in there. However, we if we have a sufficient amount of resources, we have no problem in implementing this whole thing in the same clock step ok.

So, therefore, if we can do all operations ideally in a basic block within the same clock step, but the problems are unacceptable hardware cost and delays. So, unacceptable hardware cost means we saw that, we need a lot of hardware to implement it in the same clock step and delays, we saw that we need more delays we have to wait for thus for a sufficiently long time for all the delays of the resources to be to be to be taken care ok. So, the problems are unacceptable hardware cost and delays.

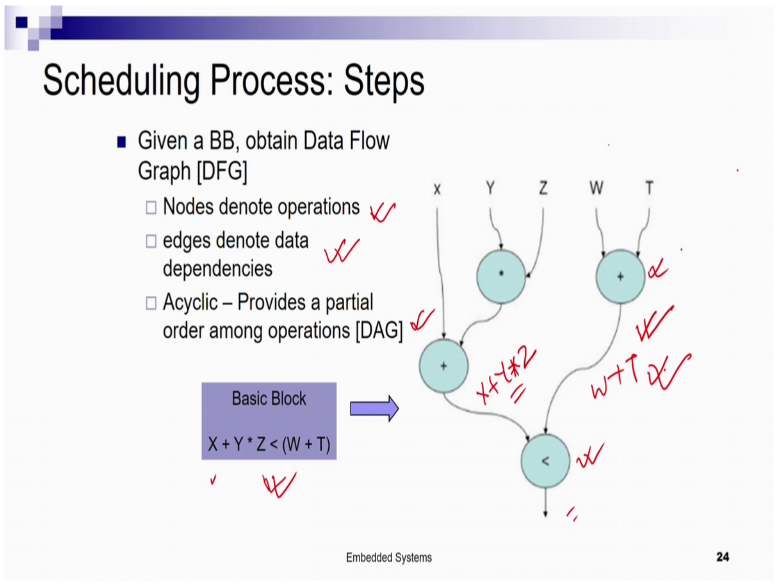
So, what is the solution to this, the solution to this is to execute you execute a basic block in several clock cycles. So, the operations of the basic block has to be executed in several clock cycles this is the solution and therefore, we need to schedule operations in the various clock cycles within the basic block. The other side of this problem is that let us say a clock, we have a clock and this clock will not be exciting only a single basic block, it will possibly be provided the same clock will be provided to multiple basic blocks.

So, we will have a fixed clock size clock step size for all basic blocks ok, if we have that then if a particular basic block has nothing much to do it may have to wait ok, or let us say if we have allocated only one operation per clock cycle. Then it may have to wait, because these operations are very small, it takes a very small time and, it has to wait after needless to wait before the next operation can be started.

So, to as a solution to this people suggested chaining, in which multiple operations may be executed in a clock step if overall propagation delay is less than cycle time. So, what does chaining allows as we saw that multiple operations can easily be incorporated, but the problems are we need sufficient hardware and sufficient delay ok. So, therefore, when can multiple operations be executed in a single time step through chaining, when we have the propagation delay of all the resources in that in those set of operations to be less than the cycle time, if we satisfy this and we have enough resources then we can do chaining.

So, now we come back to what are the steps in scheduling. So, what is scheduling is the assignment of start times to each operation in a basic block. And then after assignment of the start time to each basic block, we generate the controller to direct the operations at each time step within the basic block that is all so, we have seen.

(Refer Slide Time: 50:46)



Scheduling Process: Steps

- Given a BB, obtain Data Flow Graph [DFG]
  - Nodes denote operations
  - edges denote data dependencies
  - Acyclic – Provides a partial order among operations [DAG]

Basic Block

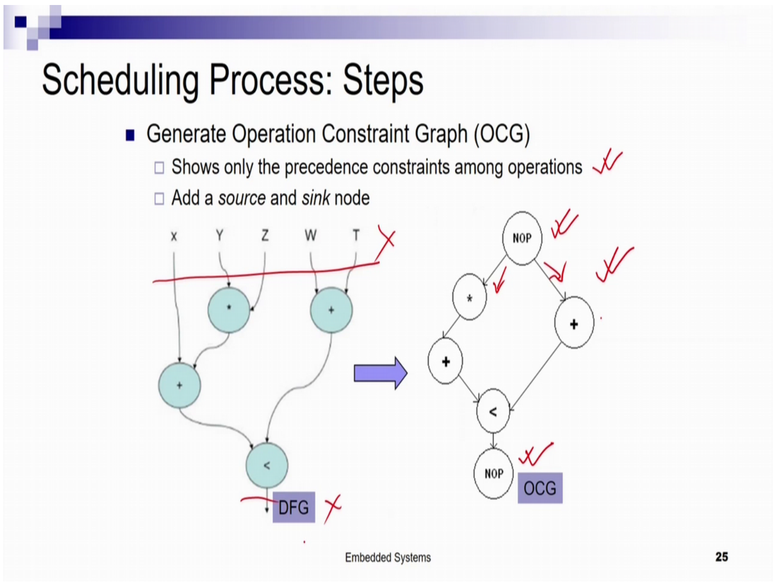
X + Y * Z < (W + T)

X
Y
Z
W
T
*
+
+
<
X+Y*Z
W+T

Now, we will look at the scheduling steps in a bit more detail. So, we had the control flow graph in the control flow graph, we had basic blocks with the basic block was a set of statements one or more statements in the, of the program and, those statements are converted into a data flow graph like this. In the car in the data flow graph nodes denote operations and edges denote data dependencies, they are as this graph is acyclic and provides a partial order among operations.

For example, if we have a basic block such as X plus Y into Z less than W plus T. So, what we have this one performs X plus this one performs X plus Y star Z see the dependencies have been adequately taken care, this one performs W plus T and, then this one performs the comparison operation and the output is produced here. So, this is the data flow graph representation of this basic block, with its dependencies taken care and the nodes representing the operations.

(Refer Slide Time: 52:00)

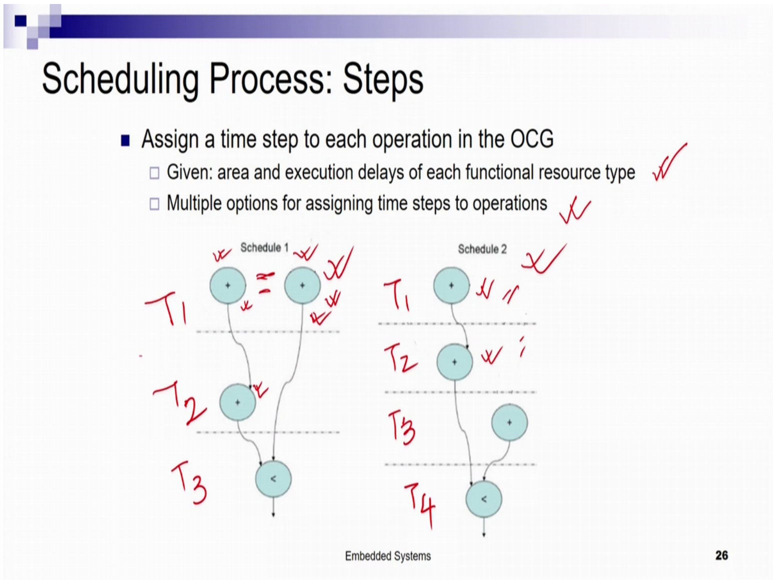


Scheduling Process: Steps

- Generate Operation Constraint Graph (OCG)
  - Shows only the precedence constraints among operations
  - Add a *source* and *sink* node

After the data flow graph is obtained the data flow graph is converted to its operation constraints graph, the operation constraint graph is very similar to the data flow graph, and it shows precedence constraints among operations and with additional source and sink node produced. So, the inputs and the outputs are removed it only shows the precedence constraints precedence constraints among the operations the inputs and the outputs are removed and, we add a source node and a sink node.

(Refer Slide Time: 52:37)



Scheduling Process: Steps

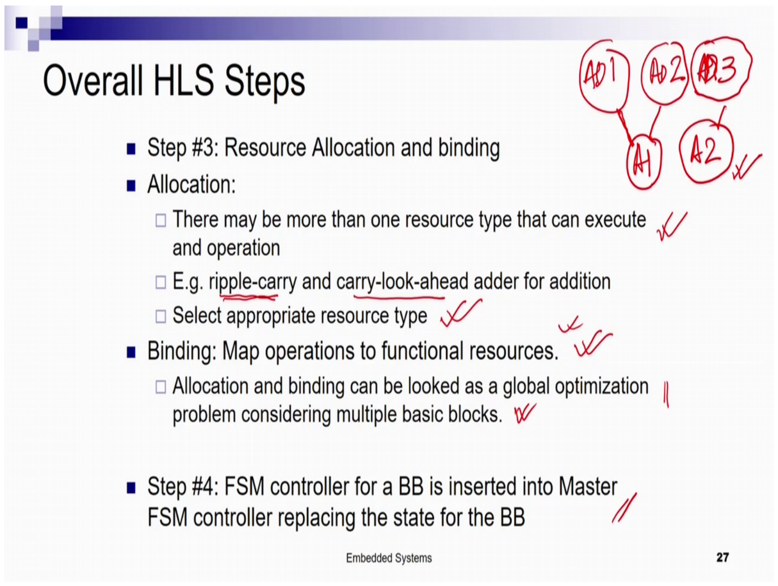
- Assign a time step to each operation in the OCG
  - Given: area and execution delays of each functional resource type
  - Multiple options for assigning time steps to operations

After the operation constraints graph is obtained, we generate schedule we schedule the operation constraints graph essentially ok. So, a given area and execution delays of each resource type, we generate the schedule multiple of options are there for assigning time step to operations. For example here is the same data flow graph same data flow graph, but two schedules. So, this schedule takes this schedule takes three time steps this schedule takes say 4 time steps ok, but we see that here we require 2 adder units in parallel.

So, I need to adder units to do it. However, here I need only 1 adder unit because, no adder unit more than 1 adder unit has not been used at the same time step. These two this adder unit can be reused with sale this adder unit or this adder unit; however, these 2 adder units I must have two physical added units for this addition and this addition operation ok. This ante for when this addition is performed I can reuse one of the adders, we used in time step 1, because it is in a different time step.

However, in the in schedule two we have all the addition operations in this think time step so, only one adder suffices. Therefore, different schedules will have different amounts of times. So, schedule one will have lower delay than scheduled 2 so, performance will be better; however, schedule 1 will require more resource and, hence will consume more area will have more cost etc than schedule 2.

(Refer Slide Time: 54:31)



Overall HLS Steps

- Step #3: Resource Allocation and binding
- Allocation:
  - There may be more than one resource type that can execute and operation
  - E.g. ripple-carry and carry-look-ahead adder for addition
  - Select appropriate resource type
- Binding: Map operations to functional resources.
  - Allocation and binding can be looked as a global optimization problem considering multiple basic blocks.

- Step #4: FSM controller for a BB is inserted into Master FSM controller replacing the state for the BB

Embedded Systems 27

Now, after scheduling comes resource allocation and binding, what is resource allocation there may be more than one resource type that can execute an operation. Now, we have said that after we have scheduled which operation to do in ad with time step, now this operation has to be implemented on an actual hardware resource. Now, we can have multiple operations for which for a resource for example, for the addition operation we can use the carry look ahead adder, or a ripple carry adder we can have a ripple carry adder or carry look ahead adder. But ripple carry adder has higher say delay than carry look ahead adder right, but the hardware again the hardware cause resource cores are also different. So, therefore, we have to select appropriate resource type.

Once the resource types are selected we need to do binding; that means, the resources operations have to be bound to the functional resources. So, the actual are the functions the operations have now to be actually bound to specific hardware resources. So, we have done an allocation; that means, which resource type to use let us say we have two ripple carry adders, now rr rr ripple carry adder say A 1 and A 2 are two ripple carry adders. And we have operations now say hmm AD 1 AD 2 and AD 3. So, we have operations AD 1 AD 2 and AD AD 3. So, these are the three addition operations and we have 2 adders ripple carry adder A 1 and A 2. So, binding would be something like this we want AD 1 and AD 2 to be to be implemented on only A 1 and let us say AD 3 will be implemented using A 3 so, it maps operations to the hardware resources ok.

These that we will use to ripple carry adder say A 1 AD 2 was selected in the allocation phase and, this mapping of operations to the actual hardware units was done in the binding phase. So, map operations through functional resources allocation and binding can be looked at a global optimization problem considering multiple basic blocks why, because then the optimization; or the minimization that we can obtain can be much better, if we have multiple resources. Because, the same hardware resource can be used by some operations in some other basic block as well.

So, therefore, this is actually a global optimization problem. Finally, after all these steps are done the FSM controller for a basic block is inserted into the master FSM controller, replacing the state of the basic block. So, we had the master FSM controller in the master FSM controller the states, were basic blocks and the connections, were at the edges of the control flow graph again.

Now, this in the master FSM controller this basic block noad will be replaced by the FSM controller that we have generated for this basic block. So, for this particular basic block to excite different control signals at different time steps and to appropriately conduct the data transfer operations, in this basic block we have generated an FSM controller. This FSM controller is essentially the controller for this basic block and, then it will be replaced corresponding to the overall node of the master controller ok. So, these are the steps that we have in high level synthesis.

With this we come to the end of this lecture.