

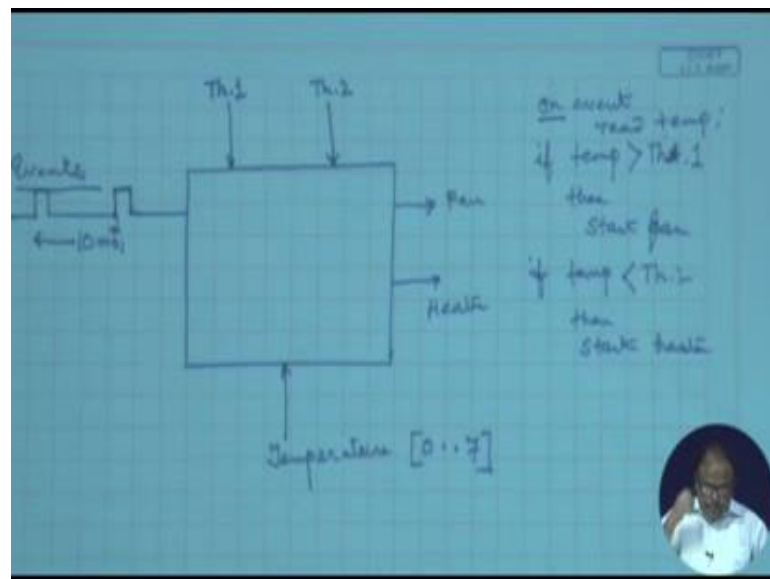
**Embedded Systems Design**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 05**  
**Optimization Issues**

Now I see, that is not been seen there; that was off.

So, in the last class, we have seen how a single purpose processor can be designed. And a single purpose processor is also known as an ASIC; application specific IC. Now the example that we have given in the last class was for computing the GCD of 2 numbers; 2 integers. Now that is an example of data dominated computation.

(Refer Slide Time: 01:31)



On the other hand if I give an example of another task that suppose I am to design a system that will accept timer signals, I mean some events, I would say some events which are coming at specific time and whenever say this event is coming every; let me put another one here, this event is coming every 10 millisecond, every 10 millisecond, this event is coming and whenever this event comes, this system should read a temperature; temperature which comes at as 8 bits that is being read through some port,

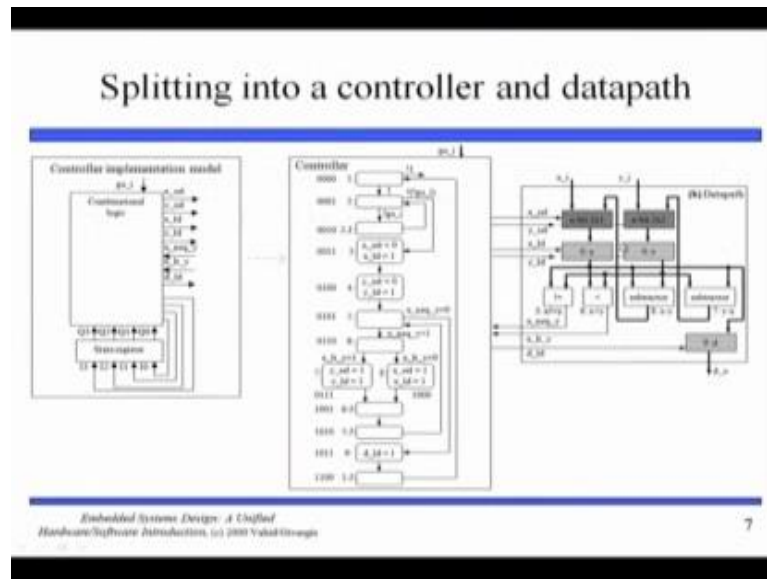
we will need some port of course, there is a design issue and through 2 thresholds will be fed, threshold 1; threshold 1 and threshold 2. If temperature is greater than threshold 1 then start coolant or start fan. So, some fan should be actuated for cooling it down. If temperature is less than threshold 2 then start heater, alright.

Now, this is a requirement. Just look at this and tell me if there is any gap, let me also check whether there is any gap because that is very important whenever you are giving the specification that is very important, whether there is any miscommunication or there is some gaps. Here events are coming every 10 millisecond. So, at every 10 millisecond, I have to sense the temperature alright that I have not written over here on even sense temperature let me write down, on event read temperature alright and then is being compared with this.

Now, this one you will have to design. So, I am, right now today I am leaving it is an assignment we will work it out at a later date, but for everybody, I would want that you try your hand in designing this very simple system. You will have to design its control path and data path. In continuation of what we are discussing in the last class, you will discover when you design it that here the data path will be much simpler than the one that was with GCD, where as the control path will not be very same, but not very complicated here also it is a very simple problem, but I want that you try it out.

This is one of the assignments and will come back to this maybe after couple of weeks to redesign this.

(Refer Slide Time: 06:40)




Now, today we will discuss about some optimization issues, I will initially go back to the earlier design that we had done we had designed this data path and control path for the GCD, we had 2 registers to hold the values, does the control lines, the 2 multiplexers and 4 operators and ultimately we got the GCD and there were 10 states, now when we carry out the optimization of these; that means, they are there is a scope of managing or reducing some redundancies that are already there.

(Refer Slide Time: 07:28)

## Optimizing single-purpose processors

- Optimization is the task of making design metric values the best possible
- Optimization opportunities
  - original program
  - FSM
  - datapath
  - FSM

Embedded System Design: A Unified Hardware/Software Introduction, © 2006 Vahid Givargis




Now, the redundancies can be there, we can do the optimization in the original program itself, we can opt try to optimize the FSM we can optimize the data path or we can also optimize the controller the complexity of the controller each of them, we can do.

(Refer Slide Time: 07:55)

## Optimizing the original program

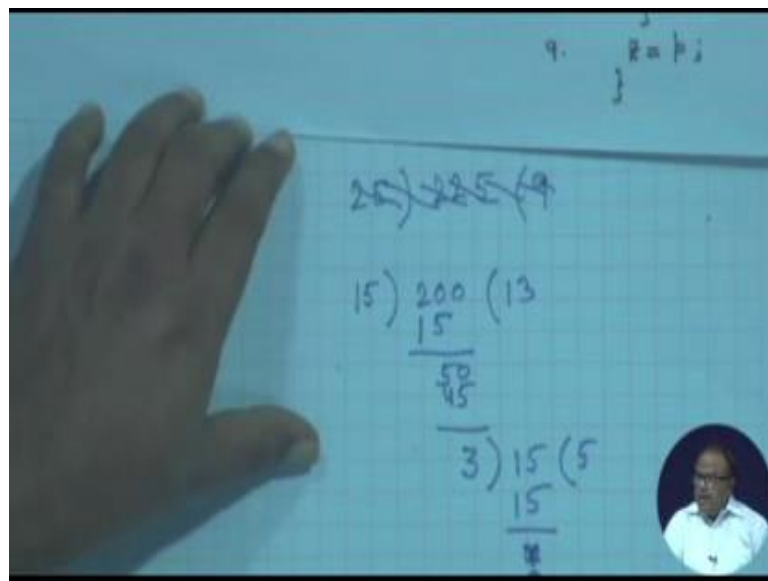
- Analyze program attributes and look for areas of possible improvement
  - number of computations
  - size of variable
  - time and space complexity
  - operations used
    - multiplication and division very expensive

Embedded System Design: A Unified Hardware/Software Introduction, © 2006 Vahid Givargis



Now, optimizing the original program just if you think of a little bit looking at this program that is being given for computation of GCD can you see any scope of computing less or making it more efficient, if you work out this part you will see that I am repeatedly comparing and subtracting I am doing that repeatedly where as the typical GCD that.

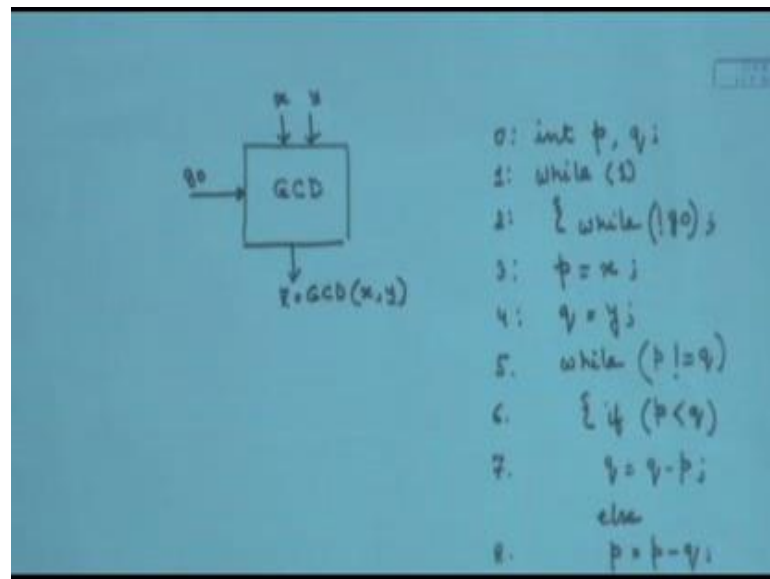
(Refer Slide Time: 08:57)



We compute say 225 and maybe 25, I am trying to divide this 9 times. So, then this becomes straight way coming as the GCD. So, I am not save again, let us say 15 and 200, if I do, what do I do? I repeatedly divide. So, and I take the remainder and then divide and go on dividing till my remainder becomes 0.

I can continuously check for the remainder to be 0, if not I will take the remainder and repeat the division with that will save me from repeated subtraction and repeated comparison of p and q.

(Refer Slide Time: 10:19)



(Refer Slide Time: 10:26)

### Optimizing the original program (cont')

<pre>original program 0: int x, y; 1: while (1) { 2:   while (y &lt;= 0) 3:     x = x, 1; 4:   y = y, 1; 5:   while (x != y) { 6:     if (x &lt; y) 7:       y = y - x; 8:     else 9:       x = x - y; 10:  } 11:  d, 0 = x; }</pre>	<p>replace the subtraction operation(s) with modulo operation in order to speed up program</p>	<pre>optimized program 0: int x, y, r; 1: while (1) { 2:   while (y &lt;= 0) 3:     // x must be the larger number 4:     if (x, 1 &gt;= y, 0) { 5:       x = x, 1; 6:     } 7:   else { 8:     y = y, 1; 9:   } 10:  while (y != 0) { 11:    r = x % y; 12:    x = y; 13:    y = r; 14:  } 15:  d, 0 = x; }</pre>
---	--	--

GCD(42, 8) - 9 iterations to complete the loop  
x and y values evaluated as follows: (42, 8), (43, 8),  
(26, 8), (18, 8), (10, 8), (2, 8), (2, 8), (2, 8), (2, 2), (2, 2).

GCD(42, 8) - 3 iterations to complete the loop  
x and y values evaluated as follows: (42, 8), (9, 2),  
(2, 0).

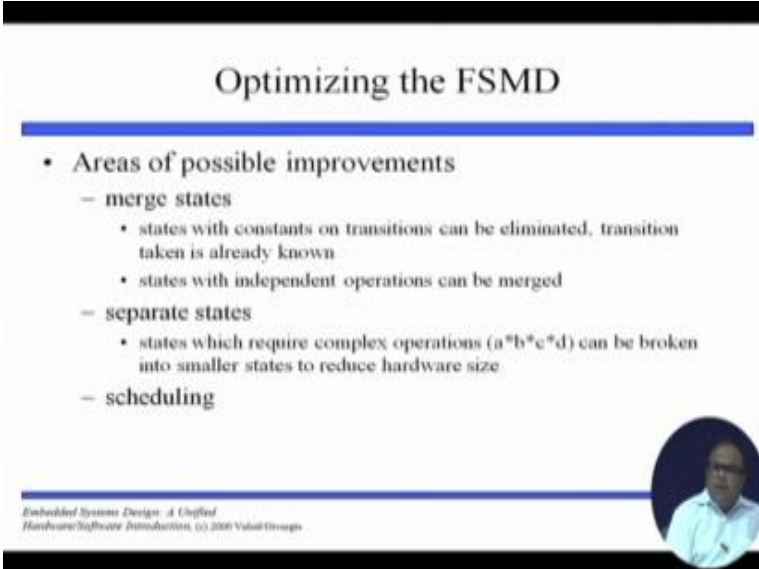
Embedded System Design: A Unified  
Hardware/Software Introduction, © 2009 Vahid Inanoglu

If I try to do that you see this part compared to, what I was drawing on this green paper and the one that has been given in the slides of Vahid and Givargis, my p and q are his x and y. So, please bear with me to make that replacement. So, here you see in the original 1, this was the loop while x is not equal to y, if x is less than y, I subtract, else I do the other subtraction and this thing I do.

Now if I replace it, here while y, y is the remainder or the while y is not equal to 0 say like this, what is the numbers that I have just now taken 15 and 200. So, 15 is the y, while y is not equal to 0, I am taking the modulus r is the modulus and modulus means the remainder. So, say 200 divided by modulus integer division 15. So, that is leaving me with the remainder 3 then I am just looking doing x y is now being assigned to x and r. So, they have been changed y and r, there are being just swapped here and I am repeating this.

Now look at compare this, how many operations I had to do here? there was a comparison operation there was a checking operation that checking operation is no nevertheless, there are 2 subtractions and here there are assignments 2 subtractions and 2 assignments, here I can just do simple 2 assignments and one operation of modulus, thereby the entire program has been made more efficient that will save in the number of clock cycles also the number of that I need here, I need to say if I take 42 and 8, if you work it out, it will take 9 iterations to complete the loop 42 divided by 8 that is being done is not being done by division is being subtracted 42 minus 8, 34 and in that way it goes on. So, it takes 9 iterations where as if I do it in this algorithm with these operations then I can get it done within 3 iterations. So, this is a very nice example of showing, how I can play with the algorithm itself to go for more computationally efficient algorithm.


(Refer Slide Time: 13:33)



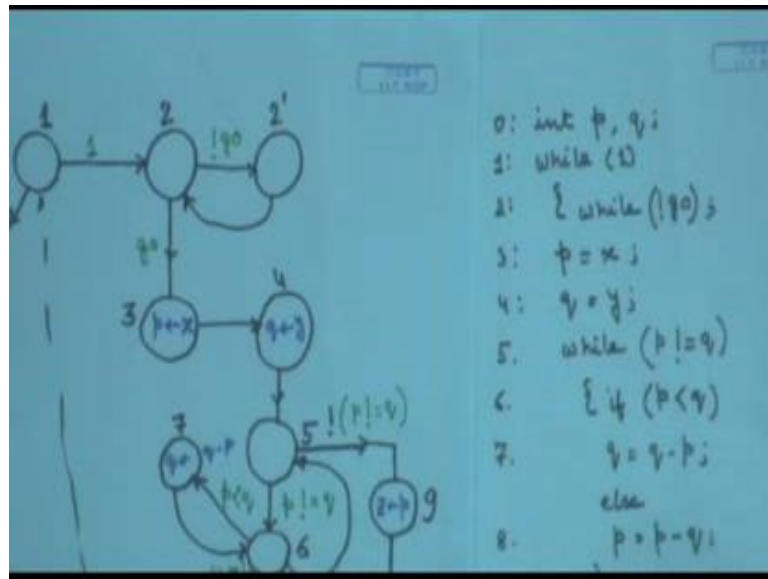
## Optimizing the FSMD

- Areas of possible improvements
  - merge states
    - states with constants on transitions can be eliminated, transition taken is already known
    - states with independent operations can be merged
  - separate states
    - states which require complex operations ( $a*b*c*d$ ) can be broken into smaller states to reduce hardware size
  - scheduling

Embedded System Design: A Unified Hardware/Software Introduction, © 2009 Vijal G. V.



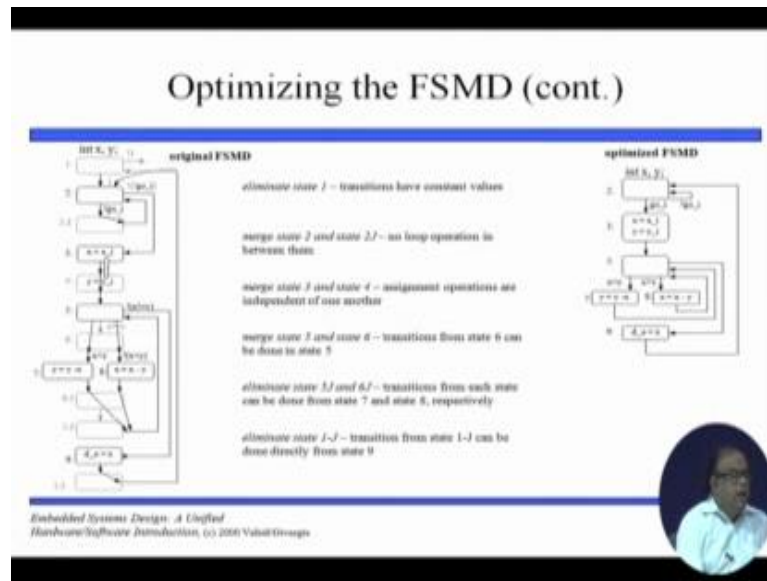
(Refer Slide Time: 13:47)



The next stage is we can optimize the FSMD itself, there are because you see when I had drawn this FSMD; this is same as we will see just drawn in a different way, I did not bother about minimizing the states whether anything useful is being done in between in the in the states or in between the states I did not bother about that I simply translate it this algorithm into this diagram step by step now it is time that I can have a people look at this and see if I can reduce this there is a second step.



(Refer Slide Time: 14:35)



We can say here now this state is actually not required right now I can simply start mark these 1 and 2 or eliminate this one and start with 2 and in 2 and 2 j, we have here what is being done 2 is on not go I am going here and again coming back. So, that could be very well done with self loop. So, 2 and 2 j can be merged to give 1 state 2. So, on go I come here and on not go I remain in the loop next 3 and 4 these were 2 distinct steps in my algorithm and accordingly I had kept them as computational states as computational states, now I can straight away merge this 2, I can merge this 2 and in 1 state I can do 2 computations why because these 2 computations are completely independent there is no way dependent on each other I am taking from one source and loading in one register taking from another source and loading in another register. So, I can merge these 2 and I get 1 state.

Next 5 and 6, similarly the assign here the assignments operator independent, now this transition from 6, I can very well take it from 5, there is no need of x not equal to y, I come to this and then take a transition, I can also merge these 2 and so my scenario becomes this, but you see here 7 and 8 cannot be merged, there is no question of merging, they are these are 2 alternate paths I can take any one of them, alright I can take any one of them depending on the 2 different conditions. So, these 2 are kept as it is this is for x less than y and this is for x not less that y and then I look.

Now, these things that were there 6 j and 5 j were there, they can also be they can the transitions here what I was doing from 7, I am coming to 6 from 8, I was also coming to 6 j day and accordingly I was doing the transition, but that that is how it was written in the algorithm, but I can straight way merge them and come to this point therefore, this becomes a reduced state diagram. So, how many states did I have 9 states or 10 states here I have got one 2 3 4 5 6 states 6 states how many flip flop should I need 3, 3, 3, 3, bits, how many bits do I need? 3 bits and earlier I needed 4 bits, why? Where did I need those bits? For encoding the states, now here since the numbers of states are less the number of bits required is also since there are 6 states I can do that with 3 bits.

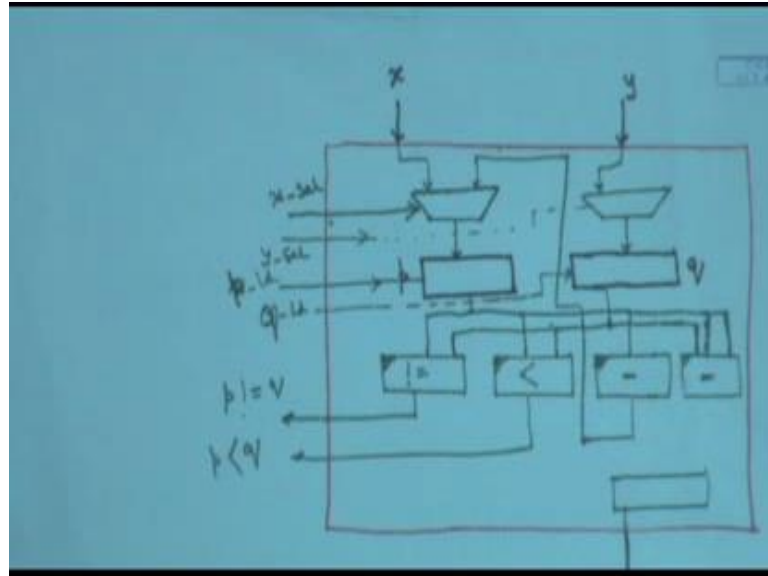
So, I need I will be needing 3 flip flops, thereby I am optimizing not yet coming to the complexity of the control path, but purely on the number of flip flops. So, that is the second part of optimization that we can do.

(Refer Slide Time: 18:43)

The slide is titled "Optimizing the datapath" and contains two main bullet points. The first is "Sharing of functional units" with two sub-points: "one-to-one mapping, as done previously, is not necessary" and "if same operation occurs in different states, they can share a single functional unit". The second is "Multi-functional units" with one sub-point: "ALUs support a variety of operations, it can be shared among operations occurring in different states". At the bottom left, there is a small text: "Embedded Systems Design: A Unified Hardware/Software Introduction, © 2009 Vahid Inoué". At the bottom right, there is a circular portrait of a man in a white shirt.

Optimizing the data path that was a little evident in the last class itself, what we did is one-to-one mapping that we did; you can come to any one of those diagrams.

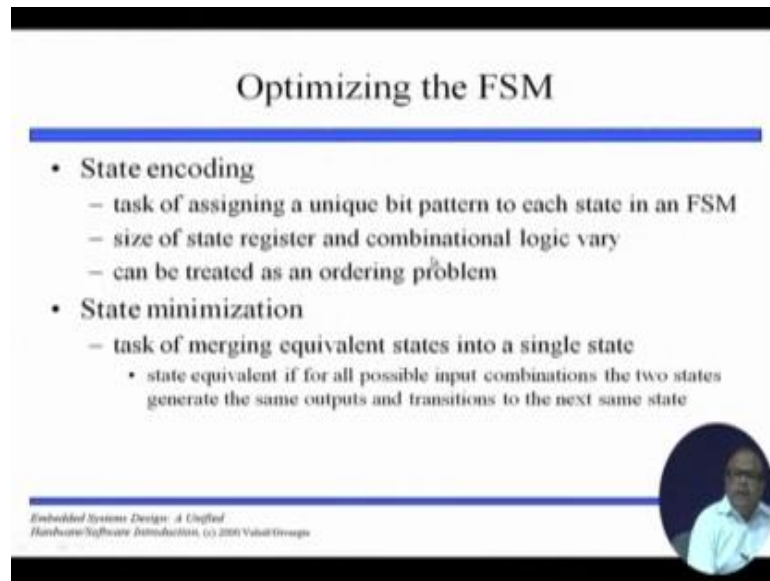
(Refer Slide Time: 19:09)



Say for example, this diagram where I needed a p and q, I had put in 2 subtractors and everything and there was one to one mapping one operation with this subtractor, one operation with this subtractor, why I can very well reuse the subtractor because why can I reuse the subtractor? Both of them are not being used in the same time therefore, I can use the same functional unit right also I had used comparative there was one not equal to there was there was 2 subtractors if I had, but now if I look at this algorithm lets here. So, except for this which I can merge in one state this is purely loading this is no arithmetic operation this assignment is purely the task of loading the multi controlling the multiplexer at loading the registers. So, that is not an issue.

Here none of these operations are being done concurrently. So, I can very well I could have used a one ALU for that now I have to see whether the cost of the ALU is more than having them separately a simple solution is of course, the subtracted could be reused or I can see if I can use the ALU also, but you see can you if you just think of think a little bit as soon as I use a ALU with this here I am using separate things right and accordingly separate control signals will come to each of them activate it and if it be an ALU there will be different control signals coming so that the control circuit will also be changed. So, the optimization criteria are multi functional units or sharing of the units.


(Refer Slide Time: 21:24)



**Optimizing the FSM**

- **State encoding**
  - task of assigning a unique bit pattern to each state in an FSM
  - size of state register and combinational logic vary
  - can be treated as an ordering problem
- **State minimization**
  - task of merging equivalent states into a single state
    - state equivalent if for all possible input combinations the two states generate the same outputs and transitions to the next same state

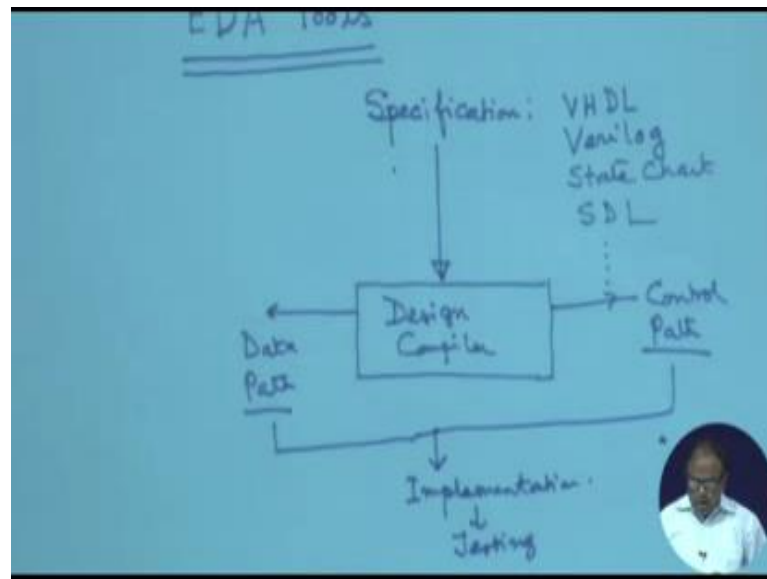
Embedded System Design: A Unified Hardware/Software Introduction, © 2009/VeriBit Group



And the other thing is the optimizing the FSM, the state encoding, we can assign a unique bit pattern to each state size of the state registered and combinational logic vary we can minimize the states multiple states into a single state that is possible. So, in these different ways we can carry out the optimization.

Summarizing, what we have done in this design state encoding you have to do, but there are ways and means of encoding by which you can reduce the number of reduce the complexity. So, because here the way you assign the unique bit pattern to each state that is very important because accordingly you will have to use the flip flops also alright now. So, what we have done in this exercise is we wanted to design a GCD this is a simple example we started with an algorithm fine and we translated that algorithm into the control data flow graph or FSMD which I got here not here where is that we got an FSMD here right and based on that looking at that we developed the data path and then we first developed the data path and then we proceeded to design the control path.

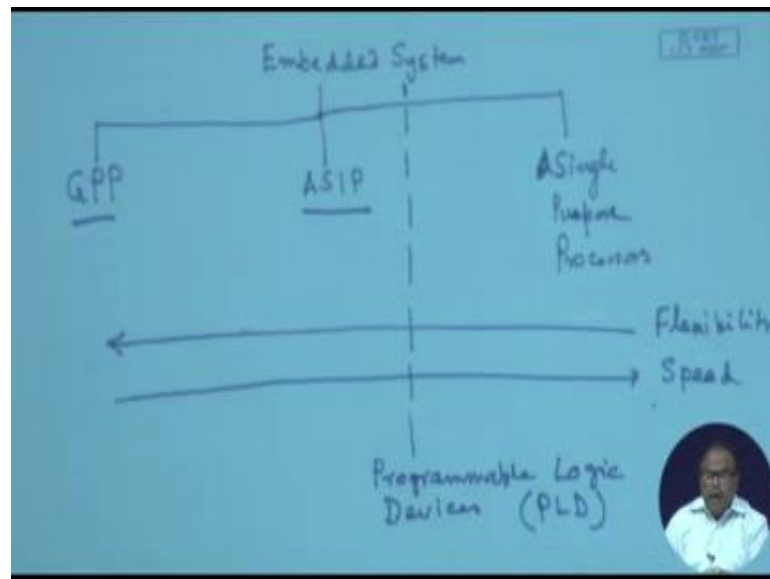
(Refer Slide Time: 23:22)



Now, after we did that we went for optimization, now there are EDA tools there are tools electronic design automation tools which are also known as EDA tools which allow you to describe the behavior of a circuit in or behavior that you want to do using some standard languages specification in through some standard languages we learn some languages, but very common are VHDL, Verilog, etcetera, there are other languages will look at them as state chart, SDL so and so forth, we look at those.

Now this specification is fed to a design compiler, the specification is validated and then you can call it a design compiler that leads to the data path and the control path and optimizations and that. So, for complicated circuits we will not be able to handle it handle all the apps or optimizations manually, but after that we after the comp the optimization are done then this thing is coming is brought in to the actual implementation followed by testing there is the more or less the flow of the tools the EDA tools.

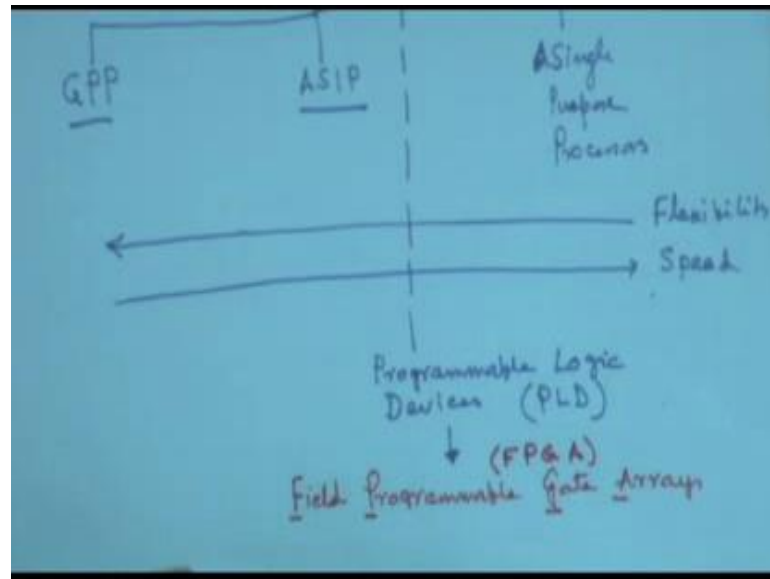
(Refer Slide Time: 25:40)



Now, summarizing whatever we have discussed up to now that is for embedded system design we can utilize general purpose processors we can utilize ASIPs or we can design single purpose processors, single purpose processors. Now if we look at it from the flexibility side then as we move in this direction, we are gaining more and more flexibility, if we, but; obviously, if I just put another axis of energy cost I mean the time to market time to design wave which way does it increase it will increase in this side right now as regard speed which side should be faster this side should be usually faster because here we are getting everything on dedicated systems, here there is software components involved which are doing which are being interpreted by the processor and that is a relatively slow than this.

However the energy cost or the time to market will increase if I want to make single purpose processor for everything. So, in between these 2 in between this range there has been another segment which is known as programmable devices programmable logic devices or PLDs as they are called.

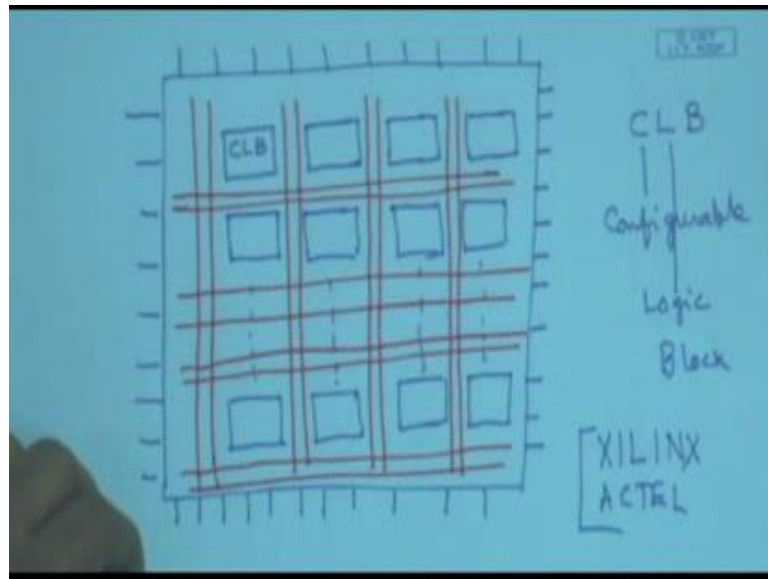
(Refer Slide Time: 28:14)



And ultimately in embedded system market a variant of this PLDs a much more advanced version of them has come in the form of field programmable gate arrays right field programmable gate arrays in short they are known as FPGAs this FPGAs can be or hardware, but their interlinks can be programmed alright interconnects can be programmed and that can be done using some software and once that is that interconnect programming is pumped into the FBGA board then that particular FPGA can carry out any logic function of your intention.

This is one step, what typically we do is if I have got an idea of a design we may like to do it on an FPGA quickly and see whether my functionality is met only then and sometimes that FPGA solution works fine alright it will be a little slower maybe then the single this thing, but is much more it is not as flexible as these, but there are certain advantages of this which I am coming to again alright not as flexible as the purpose processors, but it is much easier to program I can reprogram them if I like depending on the type of FPGAs.

(Refer Slide Time: 30:41)

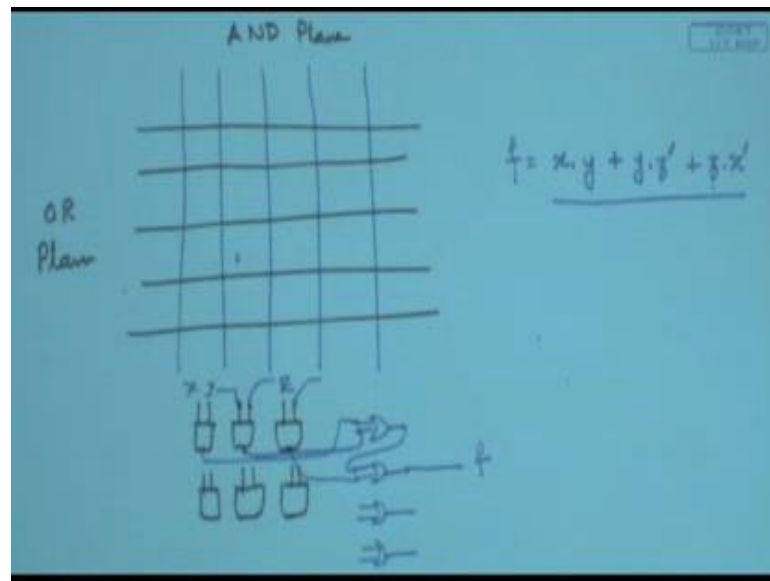


An FPGA or FPGA is essentially a scenario like this will have an FPGA board on which there will be several io pins on all sides with which we can connect to the external world and there are a number of CLBs, this is CLB what is CLB? CLB stands for configurable logic block these are configurable logic blocks there will be an array of these typically the present day FPGAs give you around 3 hundred thousand such CLBs on the typical chip on a typically FPGA chip the FPGAs the best known FPGA manufacturers are XILINX, did I spell it correct or ACTEL and there are other also others also I should not say, but these are very popular.

Now, there are number of combinational logic blocks and these are surrounded by a number a of interconnection wires number of huge number of interconnection wires now these interconnections I can very well program alright now this. So, I can just give you an idea of how this is what do I mean by this programmability let us go back from FPGAs to their earlier predecessors PLAs.



(Refer Slide Time: 37:17)



I mean this where imagine you have got an array of and gates there are some interconnections this and plane with the number of and gates and there is an or plane there is an or plane and here is the and plane. So, here at this level green level are the colors visible yes suppose there are number of and gates and on this side there are number of or gates recall that any logic function we can write as say  $x y$  or we can write as a combination of, or as a main terms right I can write in this way 3 input say I am talking about 3 input function  $x y$  and  $z$  suppose this.

Now, this is nothing, but a combination of ands and ors therefore, it is possible for me to implement this functions if  $xyz$  are available here then I can connect if I can connect say this and gate and here I put in  $x$  and  $y$  and I connect this and connect this to this or gate I take  $y$  here, this is  $x$ , there is  $y$  and I take this  $y$  again here, connect this  $y$  here and I take  $z$ . So, here I am available either I am getting the compliments of the variables and the variables themselves  $z$  and  $z$  prime are both available to me. So, I can connect  $z$  prime here and I take this and output and put it at the input of other input of this circuit therefore, I am getting this function.

Now, I can similarly do this and then again connect the output of this or gate to this or gate. So, I can similarly have  $z$  coming here,  $z$  is coming here,  $x$  prime coming here, I

can take this and gate and this one can come to this or gate and this or gate can be connected to this therefore, from here I am getting the function  $f$  the point that I wanted to make here is just by making these connections I can connect the and gates and or gates at my will, and therefore I can implement any particular Boolean function that serves the basis of the idea of all programmable logic devices.

In the next class, I will discuss about how we really actually do the interconnection. The basis of that and we will deal more on FPGA's.