Digital VLSI System Design

Prof. S. Srinivasan

Department of Electrical Engineering

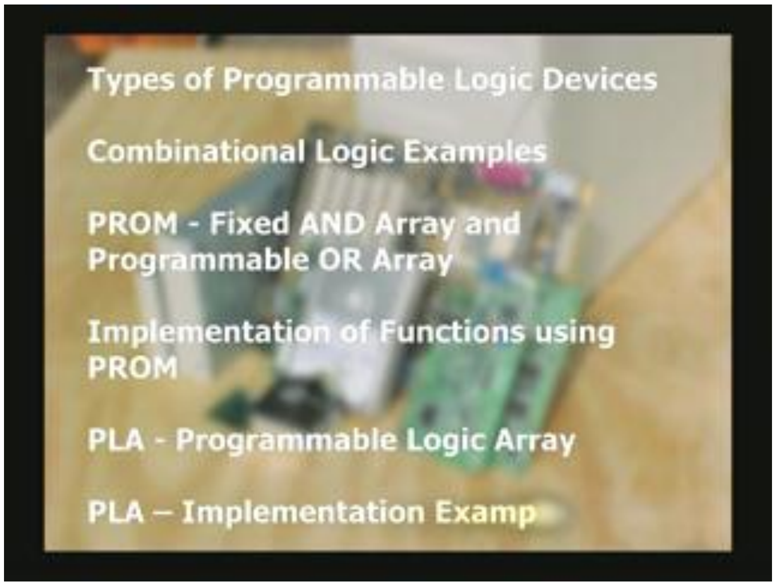
Indian Institute of Technology, Madras

Lecture - 4

Programmable Array Logic

Slide – Summary of contents covered in the previous lecture.

(Refer Slide Time: 01:12)



Types of Programmable Logic Devices

Combinational Logic Examples

PROM - Fixed AND Array and Programmable OR Array

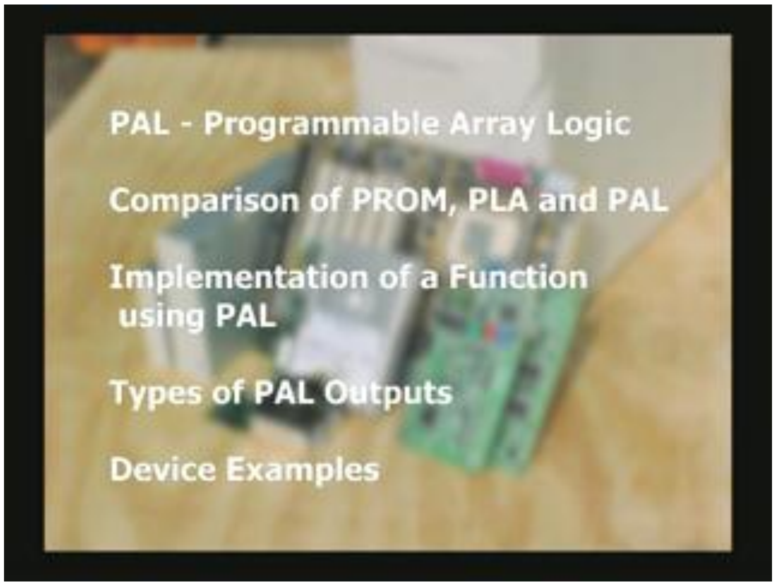
Implementation of Functions using PROM

PLA - Programmable Logic Array

PLA – Implementation Examp

Slide – Summary of contents covered in lecture 4.

(Refer Slide Time: 01:42)



PAL - Programmable Array Logic

Comparison of PROM, PLA and PAL

Implementation of a Function using PAL

Types of PAL Outputs

Device Examples

In the last lecture, we saw two programmable logic devices namely PROMs, Programmable Read Only Memories and PLAs, Programmable Logic Arrays. A minor variation of the Programmable Logic Array is called Programmable Array Logic. Today, we will see Programmable Array Logic also known as PAL. This is by far the most popular programmable device that is used for design in the industry. The difference between Programmable Array Logic and Programmable Logic Array is rather simple.

In the ROM, we derived all the min terms possible for a given set of input functions and combined those min term sin the OR gate array to get the functions required. Later, we decided it is the under utilization of the min terms, because for many functions the output uses only a very small number of min terms; instead of producing or generating all the min terms, we only generate a required number of product terms and combine this product terms to get the required functions, in so doing we try to identify as many common terms as possible.

The same concept of reducing the function to get a few product terms rather than all the min terms will be followed here, that means the approach is similar. So, we will program the input variables into a few product terms which are required for implementing the functions. But what is the difference between PAL and PLA? In the PLA we had the OR gates in which you can combine the product terms whatever way we want, while on the other hand, the OR gate connections are fixed in PAL.

You have no choose what product terms go into making a different function to make a specific output function. So what we have here is a slightly varied version of a Programmable Logic Array. This is the PAL structure, we have the inputs generated AND array as we did earlier by programming to get some product terms, p_n may be, and put them in a OR array.

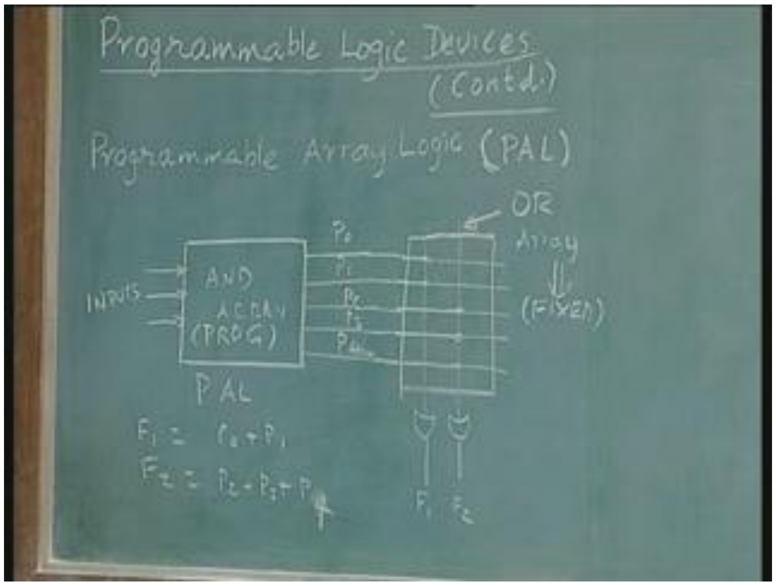
The OR array has output OR gates to which we can give the min terms and the product terms as input. The difference lies in the fact that the OR gate inputs are fixed, which means we can only have a specific product term as input to a particular OR gate. For example, if you call this F_1, F_2. F_1 we will say P_0 and P_1 alone will be connected to F_1. You have a choice of implementing P_0, P_1 using a programmable and structure from the inputs, so any combinational input structure to get P_0 P_1 is possible. But once you have done it P_0 and P_1 are fixed inputs to the OR gate. You cannot have any other input so this OR gate will have only two inputs and these two inputs are also fixed.

These inputs, P_0 and P_1, likewise may say F_2 will have P_3, P_4 and P_5. Let us take n is equal to 5 and n is equal to 4 in this case, that means we will call this P_2, P_3 and P_4. So P_1, P_2, P_3, P_4, which means my F_1 is P_0 or P_1, F_2 is P_3, P_4 (Refer Slide Time: 07:02). Now we have removed a certain amount of flexibility from the OR array. Earlier, the OR array was programmable; any combination, any number of the product terms that are generated could become inputs to the OR gates. In this case though the inputs, number of inputs as well as the products terms connected to inputs of the OR gates, are fixed. In this particular example F_1 can only take P_0 and P_1 as inputs. F_2 can only take P_2, P_3 and P_4 as inputs. That means, I am restricted in what type of functions I can implement.

F_1 and F_2 can only have the two product terms that is, the sum of product expression, 2 product terms allowed. In this case maximum of 3 product terms are allowed and these product terms themselves can be variable because I can programme it the way I like. This OR array is a fixed array, not a programmable array. Now that means, there is a slight difference in the philosophy of design. Earlier we tried to minimise the number of product terms, tried to minimise the function logic functions to as few product terms as possible but more importance was given to the commonality. We wanted to have as many common product terms as possible between different functions so that the same product functions to be generated can be put as the input to this etc (Refer Slide Time: 08:45). In this case, these are fixed; the common product terms are not going to help me. Even if P_2 and P_0 are same, I still have to generate it once for this and once for this. So common product terms are a specific advantage, but since I want to reduce the number of inputs to this OR gate to 2 and to this OR gate to 3. My simplification will now depend on how efficiently I can reduce it so go back to the original karnaugh map implementation and try to minimise the product terms, try to minimise the sum of product terms. Such that, they have as few product terms as possible, so that the OR gate are within the input limits of the OR gate.

To summarize them the three devices you have seen, three PLDs, one is PROM next is programmable logic array, third is programmable array logic; AND array is fixed to generate all the min terms; OR array is programmable you can combine any way you like.

(Refer Slide Time: 09:30)



In a programmable logic array, AND array is programmable. You can generate product terms by any combination of the input, by giving the appropriate inputs to the AND gates and OR gate is also programmable. You can combine the product terms so generated in any manner you like to form the output functions. The design concept here was to minimise the functions but keeping in mind have as many common terms as possible. PAL, on the other hand, will have programmable AND gate. That means, I can generate product terms based on the requirement of the function but the inputs to the OR gates are fixed and do not have OR gate will not have the programmable functionality. OR gates are fixed functionality, which means to that extent it gets limit restricted once generated the product terms, which product terms, how many product terms and which product terms go into each of the OR gates, is restricted. So we cannot change that but why do we go for this from here to here that means we are going one step backwards.

This is the historic evolution PROM as first there and then the PLA was found and programmability was there. The reason for going for PLA was the program fixed AND gates generating all the min terms was an inefficient way of using the product terms, as well as consume lot of silicon. So they thought of programming this so that will have programmable options, so this becomes more efficient implementation-wise. Then we are going backwards by one step because in this case these two degrees of freedom is difficult to optimize. You will have a programmable AND structure and a programmable OR structure in order to get most efficient implementation and see most of these results automated, you should realise this. We use a simple example in the class so you understand it but in practice the systems are like huge, with a large number of inputs, a large number of outputs. When you design a tool to optimize becomes more efficiently optimized, when the number of parameters to be optimized is small.

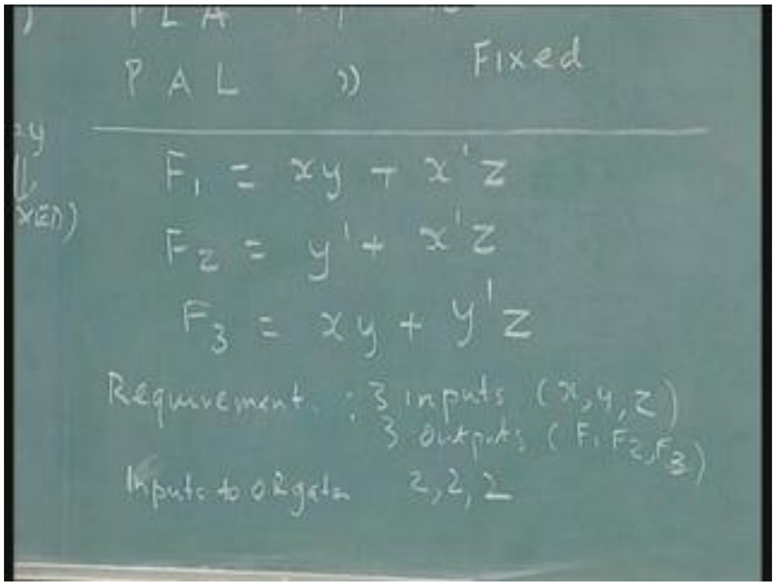
So, that is the main reason we can do is at a lower expense; in terms of the utility, in terms of utilization and also in terms of programming efforts and because of this we go to PAL. Even though we are losing a degree of freedom, it becomes easier to program manually as well as automatically, which means when something is easy to program, it is easier to optimize it so you can get better, efficient structures out of this. That is the philosophy for going with this.

Now let us try to implement the same function we implemented previously using PLA on a PAL. So if you remember, those functions were F_1, F_2, F_3. Did we call them F_0, F_1, F_2, or $F_{0,}$ $F_{1,}$ $F_{2,}$ F_3? It was F_1 xy, x-bar z. F_2 y-bar x-bar z, F_3 x y, y-bar z; what we have to do is you have to have a requirement.

We are only talking about programmable fixed AND gates OR gates structure, requirement of course, there should be 3 inputs because there are x, y, z. There must be 3 outputs so this is common to both the devices PAL and PLA. But what has to be specified further in this case, which is not done in the case of programmable logic array, is how many inputs to each of the OR gates, so will have to say here that we have output AND gates to OR gates that is, inputs to OR gates. Input OR gate has to be specified by you, supposing somebody give you, of course this trivial function.

Suppose, there are only 2 inputs required supposing you need a 5 input OR gates and give you a device with 3 inputs only with OR gates then you are going to implement this function; so this also has to be specified in your requirement. Input OR gates in this case are 2, 2, 2; I want to 3 OR gates with least 2 inputs each one of them that is what it means, so this is an extra requirement that you will need to give in your specifications also.

(Refer Slide Time: 15:40)



PAL)) Fixed

$F_1 = xy + x'z$

$F_2 = y' + x'z$

$F_3 = xy + y'z$

Requirement : 3 inputs (x, y, z)
: 3 outputs $(F_1 F_2, F_3)$

Inputs to OR gate 2, 2, 2

How do you design the system? Now, this system is very easy. I am going to have my x, x-bar, y, y-bar, z, z-bar, as I said earlier. I generate the product terms, I call this product term P_0 and P_1. This first function is x, y, x or z, x-bar, z, x, y, x-bar, z. So, these two should go into my first OR gate; OR array which is the fixed array, F_1 will have these two functions as inputs (Refer Slide Time: 17:30). Then you go for the second function. The second function is y-bar is P_2, x-bar, z, x-bar, z, P_3. These two go as inputs to my second OR gate. Now the third one is x y. This is what

I mean; this x y is generated already for F_1; we are generating and again for F_2 .We have not used that what we generated; likewise, what we have generated is x-bar z for F_1.

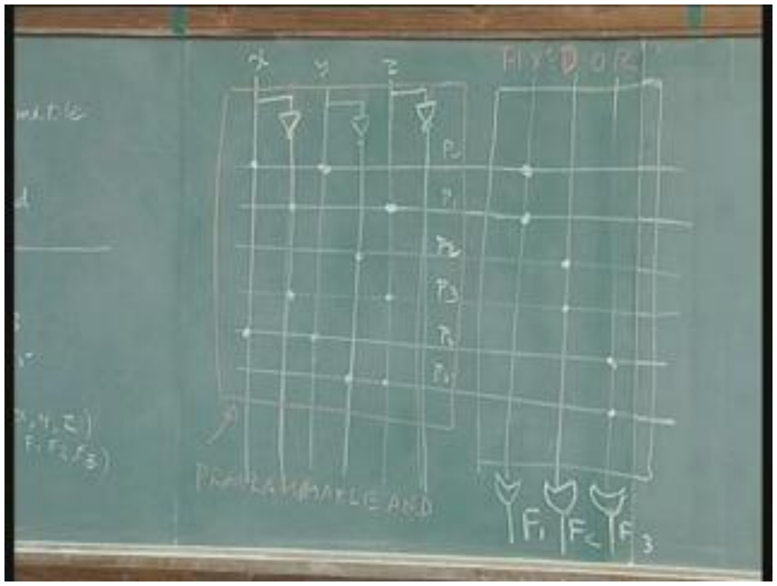
We have to generate again for F_2, see it is what it is. This is x y and y-bar, P_4 and P_5 and that will be the inputs of my third OR gate or third function, $F_{2.}$ So this is the programmable AND programmable and this is fixed OR. To identify this as $P_{0,}$ P_1, P_2, P_3, P_4 and P_5. Each term has been generated, so here the simplification will be to reduce the number of product terms for each output.

In the programmable logic array approach the technique would be to make as many common terms as possible. Here we have to reduce the total number of product terms into each of the input so that the OR gate can be used, but as I said in terms of, even though it is less flexible in terms of design effort and also the number of inputs, in the earlier case each product terms I should able to connect each OR gate.

Suppose I had a very huge system with lot of inputs and several product terms generated, and each OR gate should have the capability, would take as input any of those so the hardware, if you look at it that way the OR gate size will be very large in the case of programmable logic array. Here we have only 2 inputs so it is a 2 input OR gate.

Supposing there were five terms I should have had a 5 input OR gate out of which I have to use 1 or 2 or 3; that way there is cost saving also. Cost indirectly means silicon, so it is realistic having a little bit but the flexibility is removed to a certain extent but, programming is very easy now, both manually as well as automatically. These are reasons why the program array logic approach has become very popular and becomes almost an industry standard. Now having said all that, this is programmable array logic, as I said, which are very popular today. They are used in many situations.

(Refer Slide Time: 22:18)



People have worked further on it. There are variations. What we have seen is a simple AND OR implementation and followed by this is the sum of the products as they call it.

Now, there are different types of outputs possible. Suppose we have types of programmable array logic outputs, one is OR gate, what we saw right now; we have many AND gates, we need to have a single OR gate. This is the combinational output with AND gates input feeding into our product terms feeding to OR gate. I can also have a NOR gate output; the input AND gates can feed into OR gate, which is inverted, which means NOR output. So, this is an active high combinational logic active low combinational logic output.

I can also have an exclusive OR type of output or exclusive NOR by making these inputs feed into an exclusive OR gate; EXOR out EXNOR out. With EXNOR I have put an inverter; I will put this bubble with a color, so if the bubble comes it becomes EXNOR (Refer Slide Time: 24:55). Very interestingly, we can also have an output called Registered Output. What is a registered output? All of you know flip-flops. Flip-flops store values and you can read these values along the clock edge.

Suppose you have a combinational logic whose values have to be preserved till the next clock cycle. What do you do? You feed the combinational logic output into a flip-flop; store it in to the flip flop and when the next time clock edge occurs, you read it off. Any circuit, even though you have been discussing combinational logic, no system could be designed completely with combinational logic, and especially a system which is having some utility, very few circuits, are purely combinational.

Most of the circuits, digital systems we see are sequential circuit, also use combinational logic, and blocks of sequential circuits such as flip-flops, counters etc. It will be nice to have in a single device, a combinational logic and a register that is what is called register (Refer Slide Time: 26:30). The AND gate which feeds into an OR gate instead of directly becoming the output can be stored in a flip-flop and the output of this flip-flop can be made available as the output. This is a registered output. The interesting thing is that I can also take this output back as an input to the PAL (Programmable Array Logic) like this x, x-bar, y, y-bar, z, z-bar (Refer Slide Time: 27:14). If this value let us say Q, is some value let us say P, what are the output variables P is a registered output variable. Suppose I want to use this P value into by design, what is done by the typical combinational sequential logic design?

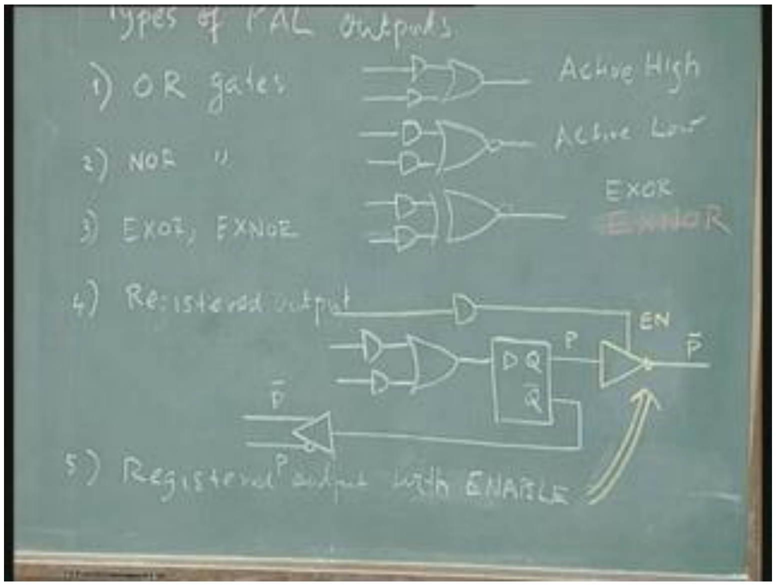
In a sequential logic design you have the combinational logic and previously stored values from the flip-flops; together they determine the next output. So this P will be required again to determine the next output that can be put here as a Q-bar. Now I can have P; this will be Q-bar so this is P-bar, so this is P-bar and P; so P, P-bar also is available (Refer Slide Time: 28:03). Just as A-bar, x, x-bar, y, y-bar, z, z-bar, P, P-bar is also available; I can combine this in order to produce my combinational terms. This is called registered output. This is a very popular type of output in PAL because we have the option of designing a system which has combinational and sequential components. As you said, any digital system which is of practical utility or a reasonable application will have both combinational and sequential part.

Combinational logic will be taken care by the array of the AND OR sequential part will be taken care by the registers which are available at the output of this so in a single output of these combinational logic, so in a single chip a device, I can produce the whole system instead of having to separately connect the combination logic to flip-flops. Now I can do further on this. I can modify this by adding a buffer which we can Enable it; so registered output with buffer for Enable. This output P, can go through a gate or an Enabler (or I will put here itself in order to understand it), (Refer Slide Time:30:10).

I can put a buffer which can be enabled by another term of inputs; combinational logic. This is called enabled signal, so if this enabled signal is high this buffer will enable the output P will be available here. Usually, this is a buffer with an inverter so this is P bar. If you do not want that you can disable this; I have a choice of using this output or not by enabling and this enable function by enabling this buffer. This enable function is not a fixed function, it can be a function which is again a function of the input variables because this enable function also can be generated using this same combinational logic array here.

x, y, z, P, all of them can form part in determining the enable function so this option of this enable is the fifth output pass. Output combination pass will output variation possible. Since it is I, use the same drawing, I will put this arrow to indicate the registered output enable because of the extra logic that I have drawn here using the coloured chalk that is the thing (Refer Slide Time: 31:32) and of course, this flip-flop requires a clock whenever you have the output of this OR gate to be clocked into the flip-flop, to be stored in this flip-flop you need to clock it and it is available as input in the next clock cycle so the clock input is there. But I am not showing it here because it only just adds to the clutter of the drawing here but you should all realise that no flip-flop can work without a clock. Later on when you look at the system level we will show the clock.

(Refer Slide Time: 32:05)



The last type of input that is available is called Programmable I/O input output, programmable input and output. This is a very interesting and important type of output function. What happens in this is, we have an option of using a given pin of the device - the programmable logic device, as either an input or output. Let us look at this drawing of this AND gate with OR gate its combinational high simplest possible output structure. This is an output combinational high output. Suppose I have too many such outputs, I do not need some of them, what I can do? Use this and this pin goes out to the external.

Suppose I have too many such functions that I do not have a use for but I am restricted by the number of inputs available to me in my device; I would like to have extra pins for input and I cannot create them. What I can do, is to use this pin as an input instead of an output. That is what you mean by programmable input output. That means, a pin or a connection can be programmed either as an input or output. The way it is done is very simple. You have again an enable buffer here, same way I showed earlier (Refer Slide Time: 34:06). This buffer can be enabled by a combinational function using the combinational AND array and if this is enabled this function let us call this F, gets through this as F-bar.

If you disable, on the other hand, if you so make the function such that this is disabled, there is nothing here even though the function F is here. Output of a gate which is disabled is a tri-state output, high impedance output. High impedance output or a tri state output means practically there is no electrical connectivity between this point and this point. This point is isolated from this point when you disable this buffer; that pin can now be used as an input (Refer Slide Time: 35:12).

I can totally altogether give some other input so in this case becomes a input pin. Earlier it was an output pin and if you disable this gate this become input pin and I give a total different function to the F. I can give another function, let us call P; this P becomes an input to be the device or your AND gate array P and P-bar. This particular pin can be used as an output pin. If there is a requirement for this type of combinational logic output by enabling this gate and there by this buffer by combining the proper input combination to the gate, so that this buffer is enabled output F or F-bar or F-bar F whatever it is, is available. In case you do not need it there is a redundant number of outputs, you do not now need this output, I can disable this by disabling them by proper combination of the AND gate, I can disable this buffer and take the output of this. This output is completely isolated from the rest of the logic, so this pin is available to use as an input pin. Any other input variable which you could not accommodate earlier in your system could be put here. In this case I called P and P becomes an extra input – additional input I put to the system.

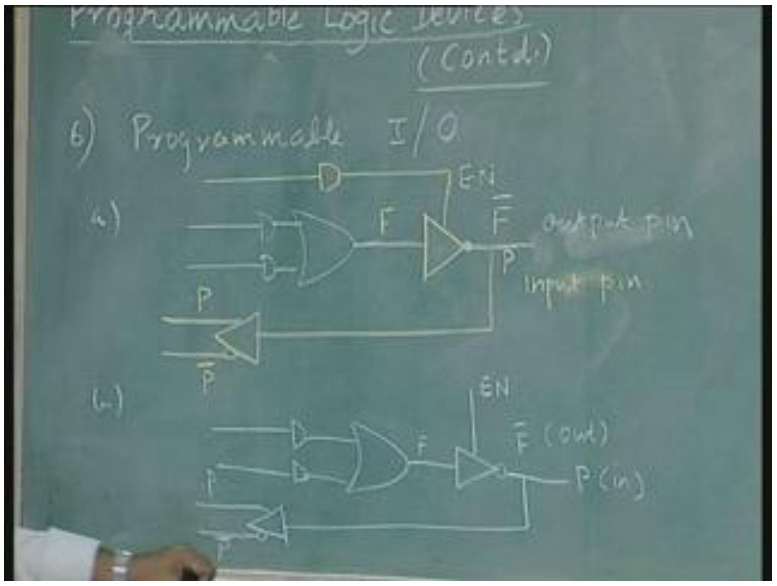
This is called programmable I/O and this enable need not be always through the logic functions because sometimes I know I want only 5 inputs, 5 outputs. There are total of 8 outputs available there are 3 redundant outputs I know for sure in the beginning so I am going to disable these 3. There is no point trying to make a combinational logic to disable every time. This can also be simplified to another variation where this can be an extra pin in your drawing; where it again enable this by this if enable is 1, this output is connected to this; the enable is zero, this is connected as input (Refer Slide Time: 38:05).

What I am saying is this enable function need not come from the combinational logic within the gate structure and array gate structure. It can come directly from a input that you give enable input to the chip. That been done, several of this redundant output pins can be connected together. Enable inputs, are all of them, will be available to the designer as inputs. This is the feature which is very popular because for your given device size, you may want to design a system with more number of inputs and less number of outputs, or more number of outputs and less number of inputs to match a wide variation of inputs and outputs using a given device; we use this option. If this option were not available what you have to do is to go for a bigger device which can accommodate more inputs.

Suppose I have 10 inputs, I have device which only 8 inputs available and I will go for a bigger device and 10 input device may not be available. You may have to go for a 12 input device or a 16 input device and on the other hand we have 4 I/O pins in which 2 of them only using output and 2 extra pins are available to me by making this I/O option. These 2 extra pins can be used as an input, so I will have 10 inputs possible without ever having to go for a bigger device. This is the great advantage in this type of output variation - output feature.

Now having said all that, we look at two commercially popular devices just to show to you how these things are done in practice. So I will take a very simple device called 16R4, which is a very popular device.

(Refer Slide Time: 40:02)



I am not giving any manufacturer's name in this because several manufacturers make similar devices. Without mentioning any manufacturing name, I can give the part number as they call it, like 741 OP-AMP you say, all of you know it is an operational amplifier but 741s can be made by different manufacturers. Similarly, 16R4 is a generic programmable logic device for PAL which is manufactured by many vendors. I want to take this as an example to just show you how the chip is organized.

R means, is a registered output 4. There are 4 registered outputs, 16 for inputs, usually the first number is input the number of inputs second number is type of output and third number is the number of outputs. What is done in this case? Let us draw this as I am not going to draw this structure; I am not going to draw this as a block 16R4 is a 20 pin device. Any device needs a ground and power supply, Vcc. This is numbered from here, 1, 10, 11 to 20 (Refer Slide Time: 42:08). Pin 10 is ground, pin 20 is Vcc. This is how it is organized: 1 is the clock, and you will see the explanation as we go along. The clock pin is required as I said in the previous case, in the registered output case. Remember that a flip-flop requires a clock and I did not show the clock at that time because of this clutter at a drawing, but no flip-flop can work without a clock so I need to provide a clock which is common to all the flip-flops. How many flip-flops are here? In this case 4 because it is a 16 R4 registers; so all the 4 flip-flops require a clock. This is the common clock for all the 4 flip-flops.

Now there are 8 pins available here, which are used for inputs I_1 to I_8 so pin numbers 2 to 9 are used for 8 inputs I_1 to $I_{8.}$ I also said there is an output enable whenever you want the input output to be used; input output combination, input output choice of certain unused outputs. I said one output enable can come either from a combinational logic block or from a simple pin out. This pin you will provide the enable for the buffer. This case it is a simple pin enabled buffers so this output enable buffer enables all the buffers (Refer Slide Time: 44:15) which have programmed I/O feature.

All the programmed I/O types are enabled or disabled by this, which is why we need 1 pin which is similar to clock which caters all the 4 flip-flops. This will cater to all the programmed I/O pins so that means there are 8 more left and we still have to wonder how we are going to accommodate 16 inputs. Out of these 4 are registered outputs (Refer Slide Time: 44:54), 4 registered outputs are pin number 3 that is registered output should be this is 11, 12, 13 this will be 14, 15, 16 and 17. Pin 14 to 17 are 4 registered outputs that are serviced by the same clock and how many pins are now left? 4 pins. These 4 pins are programmed I/O pins; there are 4 programmed I/O pins 4 registered output pins. 4 registered you said, how do you get the 16 number? These are 8 here, 4 here and 4 here (Refer Slide Time: 46:15). All these 4 can be used as inputs all of them can be used as a inputs also so the 16 is a sort of the trick that the manufacturers play on you. You cannot have the 16 inputs and 4 outputs at the same time.

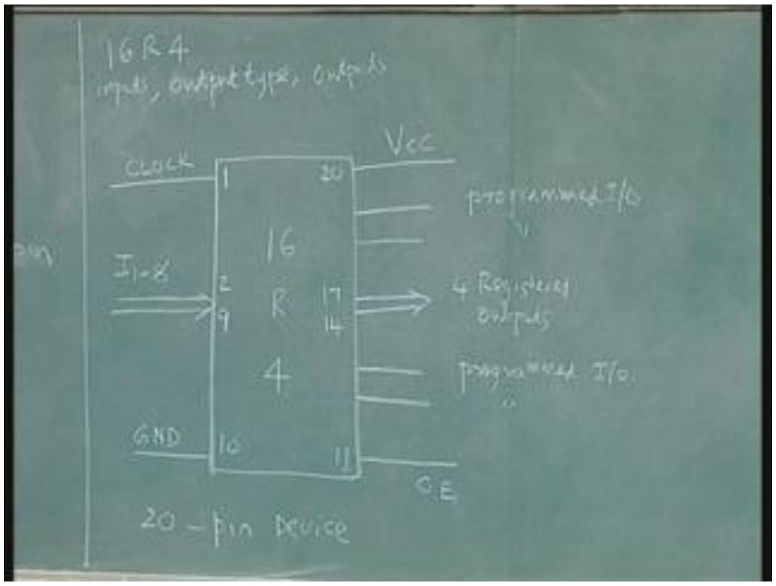
16 is the total number which can be used as input. If you use all of them, 16 as inputs, no pin will be available as a output because we need 4 pins; 1 for ground, 1 for clock, 1 for VCC, 1 for output enable. 4 pins are already gone. Totally 16 are remaining. Out of 16, I can use all of them as inputs, where do you get the output? From nowhere but if we use only pure inputs and not use programmable features registered features. These registers are inputs, remember that. Each registered output, also cater as input. These 4 outputs are registered outputs and also act as inputs (Refer Slide Time: 47:11) because they are fed back. Look at the drawing there (Refer Slide

Time: 47:21). Any registered output also acts as an input. So 4 inputs extra, these 4 outputs are counted, 4 outputs and 4 inputs which is reasonable, nothing wrong about that. What is the trick? It is that the other 4 can be used only either as input or outputs. If I use them as outputs I do not have them as inputs, if I use the input they are not available as outputs. That is why he is not totally wrong because R4 means 4 registered outputs. Even though there are 4 registered outputs, all the 4 can be used as inputs; plus another 4 inputs are possible using the output enable which caters to all these 4; this is how you get 16R4. This example of how devices sort of organize and are very versatile, you see.

I have registers, I have programmable features so I can increase the number of inputs and I have a clock, I have output enabled function and what else do you need? A moderate size design of a sequential circuit which given by the combinational logic can be designed using this and implemented using a single chip. That is the beauty of the programmable array logic. All these features are developed programmable array logic even though the programmable logic array can also be done with all these features because the historical, as I said from PROM, they gave came to programmable logic array and then they realized it is too difficult. They moved on to programmable array logic quickly and all the developments and further work happened around programmable array logic and that became the industry's popular device.

Once something becomes a popular in the industry, they keep on adding features so that they want to sell more and more of them to the industry, which is why these are common. Another other device I will quickly introduce to you, I am not going to spend to lot of time on it, is 22V10.

(Refer Slide Time: 48:24)

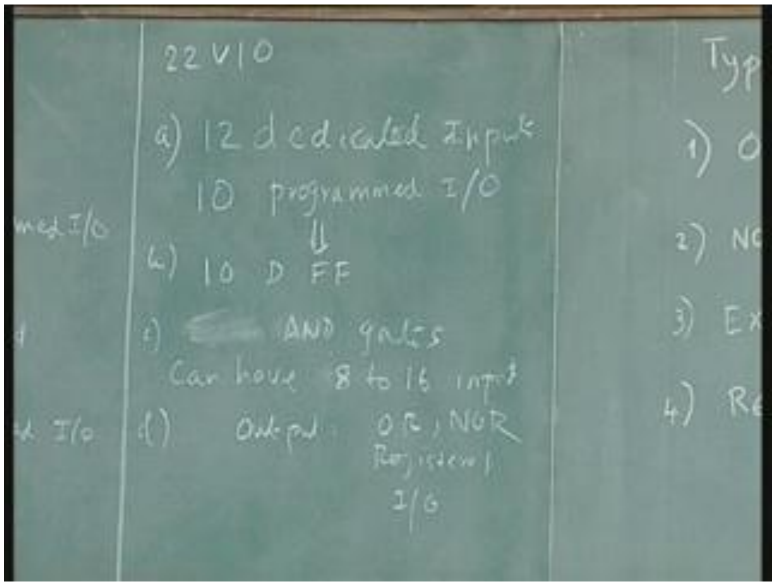


Again, it is a generic device can be made by different manufactures. 22 stands for number of inputs, 10 stands for outputs, so some sort of gimmick has been done; not to get all those but I am just going to write only the major features; 12dedicated inputs that means they cannot used as outputs. 10 programmed I/O that is how you get 22-10. Other programmed 12 are dedicated input-wise each of these programmed I/O was also connected to 10 D flip-flops; this is again a common feature.

The given pin can be used as a programmed I/O pin or flip-flop pin by a proper logic inside. So 10 D flip-flop and each of them is given by a OR gate combination. Then each of the flip-flop, and AND gates can have 8 to 16 inputs; output can be registered, all output combinations or output registered output I/O output.

I am not going to talk further about this; you can look it up in any standard text book or any manual which list the programmable. There are lots of manuals on programmable logic device today. You can even go to web and search for this. All the major manufactures have websites in which they give the parts manufactured by them. Look up AND 22V10 and find out these features further but what I want to say quickly is these outputs are very nicely done. We have a combination. There is a little block at each output. This block decides whether the output is the combinational output with combination OR or NOR. I can have an OR or NOR output or I can have a registered output. There is a multiplexer which is an OR output, NOR output registered output. Again register can be either Q or Q-bar, or if I totally do not want any of this I can go for input options. All of these are possible; so this is an extra nice logic of the output of each of the pins. These 10 pins which are used as a programmable I/O as well as 10D flip-flops. All these 10 pins have any way from 8 to 16 gates. Each one is the fixed number of gates plus they have this feature so it is very, very interesting and useful device and we may later on have a chance to look at this as an example. We will stop here today.

(Refer Slide Time: 52:35)



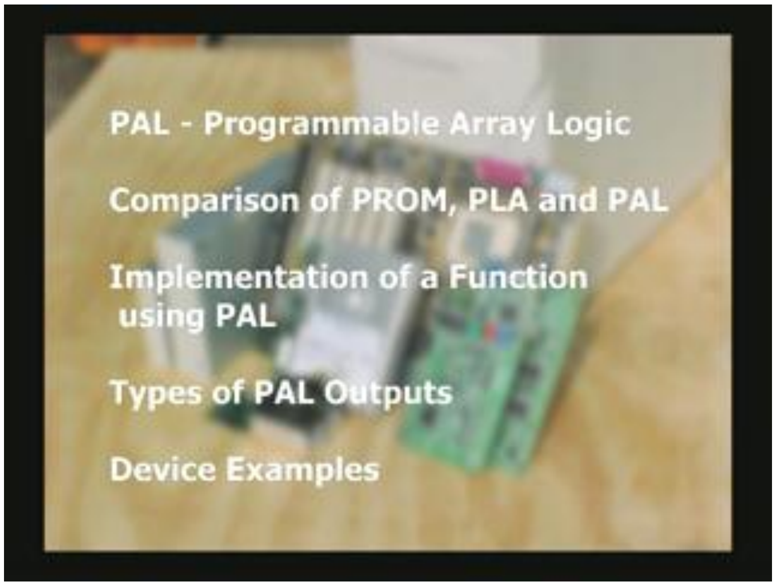
22 V10

a) 12 dedicated Inputs
10 programmed I/O
⇓
b) 10 D FF
c) AND gates
Can have 8 to 16 input
d) Output OR, NOR
Registered
I/O

Typ
1) O
2) NO
3) Ex
4) Re

Summary of Lecture 4

(Refer Slide Time: 53:02)



PAL - Programmable Array Logic

Comparison of PROM, PLA and PAL

Implementation of a Function using PAL

Types of PAL Outputs

Device Examples