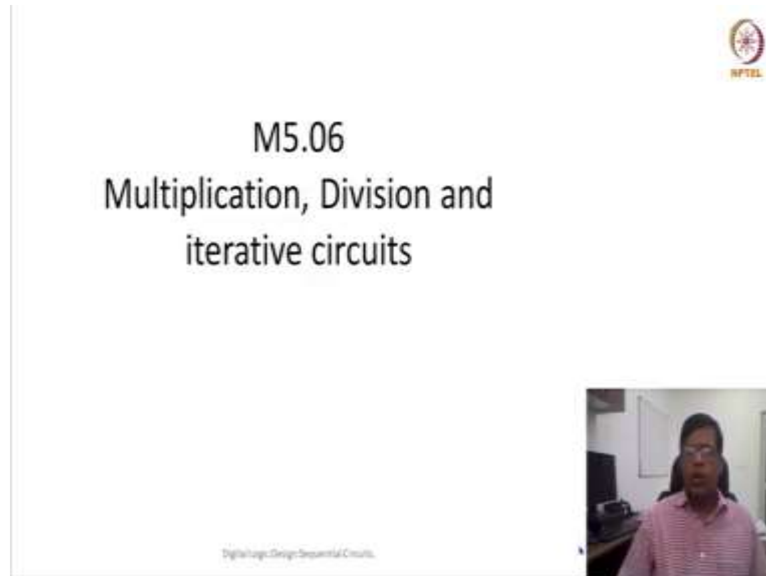
Digital System Design
Professor. Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture No. 58
Multiplication design

(Refer Slide Time: 00:15)



Hello all. Today, we are going to discuss few more applications of finite state machines.In our previous lecture, we have seen that how addition can be performed using finite state machine, which will reduce the area andthe delay would be very similar to a ripple carry adder. And also we have seen another concept in our previous lecture of pipelining where, where throughput can be increased withsequential circuits.

Today, we are trying to see the similar applications of finite state machines in other arithmetic circuits like multiplication, division and then we will generalize it as a iterated circuits.

(Refer Slide Time: 01:05)

Somultiplication, we have seen multiple times during this whole course, in module 1 and module 3. And now, we are again retaking this concept of multiplication, and this slide is again is taken from module 3 slide. So, we have seen that multiplication is a two-step process. In step 1, multiplicand is multiplied with each bit of multiplier and a partial product regenerated and then we are adding the partial products.

So, this is the example. For example, this is themultiplicand this is multiplier. Each time we are multiplying one bit of the multiplier with multiplicand and generating a partial product and then we are adding all these partial products. So, in this lecture we are trying to design that what would be the hardware.

So, let us look at this multiplication more carefully. So, what you will see that every time we are adding. First time when we are adding then we are adding the multiplicand, we are putting the multiplicand here. And what is the condition of putting multiplicand here? When least significant bit of the multiplier is 1; if it is 1, then we will put as such whatever is the multiplicand as the result.

Now, when next time we are adding, then we are looking at the instead of zeroth bit we are looking at the first bit. So, if it is 0, then this whole multiplicand is shifted left and then addition is performed. And for the second bit it is again shifted left and then addition is performed. And for the third bit again, it is shifted one more times, then addition is performed.

So, in other way so every time this is supposed to be an 8 bit addition and after adding one multiplicand one set of multiplicand then after one partial product, then another person product would be added. And finally, if we consider our partial result, as a 8 bit result then every time we are adding the shifted multiplicand, shifted multiplicand.

So, this also we have seen that if my multiplicand is N bit multiplier is M bit the result is going to be N plus M bit. Now, this hardware course we have seen last time, so, we are trying to revisit in case of sequential multipliers. So, in our previous case, we have seen the multiplication when we did in module, did this multiplication in Module 3, we have, we did this multiplication either using N number of adders or we have also done usingcarry save adders where we are trying to add all these partial works at once.

(Refer Slide Time: 04:12)

So, now let us look at the other method where if we have a single adder and then we would like to multiply. So, if that is so and let us consider N bit multiplicand, and M bit multiplier. So, if multiplier is M bit then the total number of steps are going to be M steps.Each step we can add to one clock cycle or each step we can map to one state of the FSM, Finite State Machine.

Now what would happen in each state? First step or first state, it would assume the result is 0 and it will add the multiplicand to the result. Result is initially result was initialized to 0 so we are adding multiplicand to the result. If multiplier zeroth bit of the multiplier is 1. And in all forthcoming steps every time whatwe are essentially doing we are shifting our multiplicand by left essentially we are multiplying it by 2 and we are right shifting the multiplier.

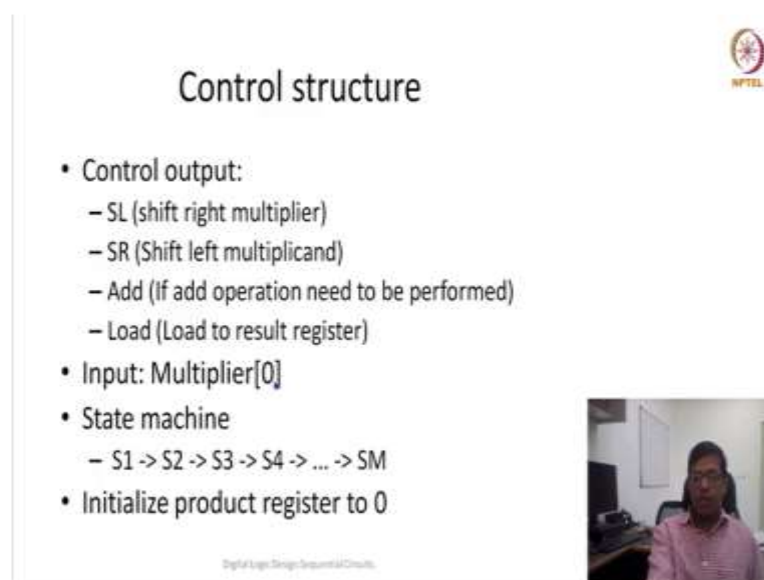
So, right shifting the multiplied means that this multiplied 0 bit will go away and the first bit will become zeroeth bit. So, this way in any Ith step we are looking at Ith bit of the multiplier. And by shifting multiplicand right, we are every time we are shifting it by a right nice we are we are doing this operation. So, we are shifting all the this multiplicand in the right direction. So, these are the two operations, which we are always doing. We are always shifting the multiplicand by 1 or basically shifting left by 1 bit and we are also shifting right we are shifting multiplier in right direction again by 1 bit. So, these two steps we are always doing.

The other step is, we have to add the result of multiplication multiplier, ifmultiplicand and the previous result if my multiplier zeroth bit of multiplier is 1. If zeroth bit of multiplier is not 1 then we need to add or in other words even if we add we will not show it as a result. So, and after this M steps, we would be done because we would have done only already these M partial additions. So, often M steps we have we are done.

Now, if we want to implement this structure, so, we can see that, we can design this as aM statefinite state machine with M stateswhere each of the state.M state will have likeyour first state will initialize the results and after that every time the input to this this Finite State Machine is going to be multiplayer zeroth bit of multiplayer. And there are a couple of operations which we need to perform in each of the state.

For example, shift left of the multiplicand shift right of the multiplier and we have to see whether we need to add or not, and also to see whether we need to write the result in our result position or not.

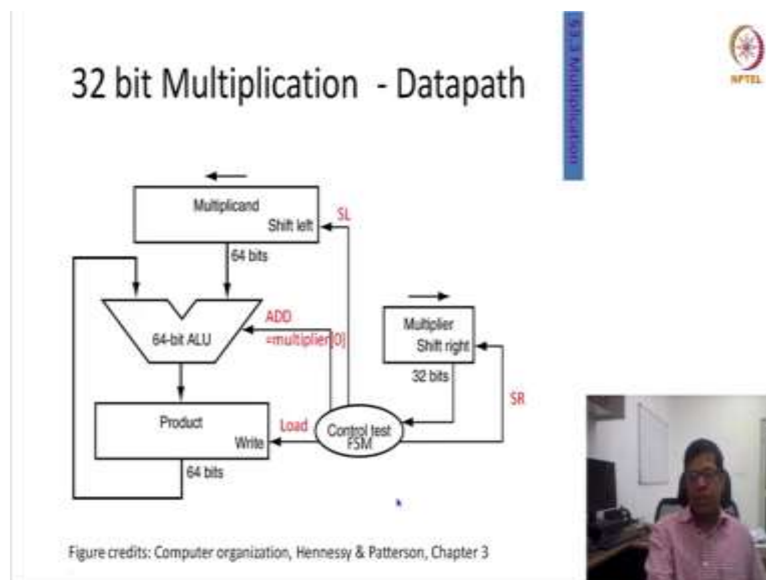
(Refer Slide Time: 07:38)



So, if we see, these are the control, these are the outputswhich we have to generate every time whether we need to shift right the multiplier whether we were to shift left our multiplicand and whether we need to add this operation or not. So, whether we need to this do addition or not and whether result register need to be returned or not.And the input is definitely multiplier zeroth bit of multiplier.

And it is going to be a M state machines and there the states are simply going from one to another S1 to S2, S2 to S3, S3 to S4. So, essentially it is a counter, it is a Modulo M counter. And if it is a Modulo M counter and for example, if number of bits is 32 then 5 bit counter would be sufficient.

(Refer Slide Time: 08:34)



And, so let us look at the data path. So, what is data path? Data path is essentiallywhen we are doing multiplication there are going to be two major portions. One is this finite state machine, which is our control and which would be determining when to do these operations. And the other part is the flow of operations, flow of data. And becausethis flow of data would also require some sort of operations like addition here and shift register here, so all of these things will form the data path.

So, usually data path would have certain combinational logic like addition and multiplication. And also our data path also involves the registers, different kinds of registers. And now to control what operations need to be performed, and when to perform these operations and what would be thecontrol bits of multiplexers, all of these things would be decided by a Finite State Machine, and each state may generate certain output bits and based on that, the operation will work.

So, here we will see that initially, this multiplicand would be stored in the right half. So, basically this portion multiplicand is we have to remember that multiplicand is 32 bit and in a 32 bit multiplication we will have a multiplicand which is 32 bit multiplier, which is also 32 bit and the product we are assuming is a 64 bit.

Now, since 32 bit multiplicand is there initially it would be in the lower half. Lower half means you can say 0 to 31 bits. And thenmultiplier is also stored in the 32 bit register. So, every time and there is also a product register which is of 64 bit. Now, initially at a time equal to 0 or in the starting of my zeroth state in the I will initially this product to 0.

So, because product is, this sort of 0 would be sent as one of the input to my addition circuitry and this 32 bit multiplier would be given as the other input. Now, based on what is the zeroth bit of this multiplier. Based on the zeroth bit of this multiplier we will add or based on again, this zeroth bit of multiplier will load.

So, basically if this addition has been performed then only, we will write that into this product register otherwise we will not write. So, after this first state what we will do will shift the multiplier by right. So, when we are shifting the multiplier by right what we are doing is we are dropping the zeroth bit and first bit will become zeroth bit.

And similarly, this multiplicand would be shifted in the left direction. So, initially let us say it was 32 bit now to will become 33 bits. So, these 33 bits would be added to the product register, which is also initially because of 32 bit sum, so it would be 32. Now, 32 bits would be added to 33 bits it will become 34 bits.

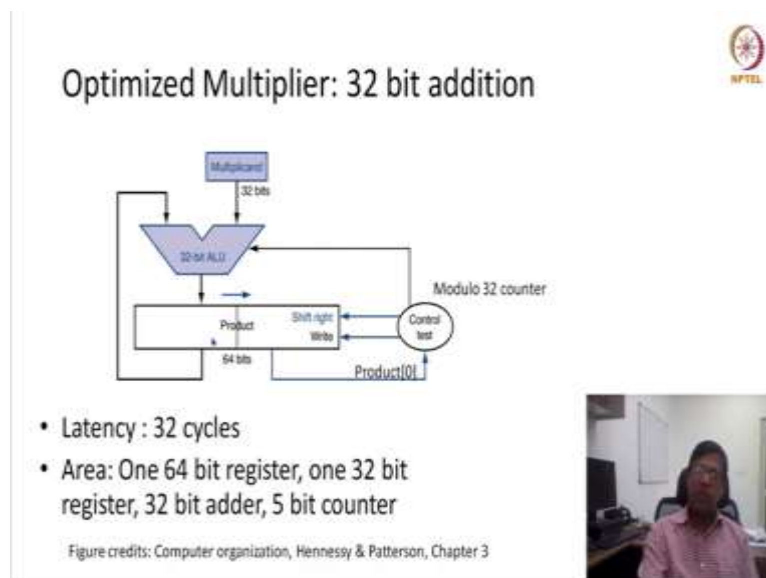
So, similarly, in the next cycle or in the next state this multiplicand would be further shifted. So, instead of 33 it will become 34 bits. So, similarly this will keep on happening for 32 cycles and by the end we will have a result, which is a product of 64 output of 64 bits and would be done.

So, what we have to notice here that one this load would be a 1 only if a multiplier zeroeth bit is 1 or the other thiswe are calling it ALU Arithmetic Logic Unit because let us we are assuming we can assume that for sake of simplicity it can perform addition subtraction and other things also. So, but even for non-generic case it could be a 64 bit edition also. So, this SL and SR will always be 1 during the old the 32 states. After 32 states SL as well as SR will become 0 so that no further operation would happen.

Similarly, load will also become 0 so that my final product is there, it is going to be there that is it. So, so, one acknowledgement that these figures are out basically this whole explanation is taken from a book of computer organization by Hennessy Patterson, if somebody would like to see. So, otherwise book of Roth also explains almost the same thing. So, that explanation is is equally valid. So, I find this as more simpler way to explain. So, I prefer to use Hennessy Patterson method of multiplication for explanation.

So, now, one more thing that let us say can we make this thing a little better a little faster or not faster, but it more optimized. So, what you will observe here that in the initial half in the initial, in the first cycle you will see that the right half, left half of this multiplicand is empty it is 0. And similarly, this initially the for the zeroth cycle this initial 32 bits are all empty, so, they are not used. So, can we somehow utilize this and can we do something better?

(Refer Slide Time: 14:48)



So, if we try to see the optimized operation, so what can be done is that we can take a 64 bit product register or the output register. So, in the 64 bit how we keep our multiplier in the in lower half and the upper half we initialize to 0, so that is our product. So, we initialize the upper half to 0 and lower how we keep as a multiplier. Now multiplicand is always 32 bit. So, now we can simplify the whole process, because every time whenever we are shifting in the right direction 1 bit get we get a space of 1 bit here. And if you observe doing the partial additions of a multiplier, you will see essentially 32 bits are being generated, that is it.

So, every time we are going to add 32 bits zeroth bit, for example, after first addition will never be altered. So, when it is shifted right, then it can take a space in this multiplier domain. And similarly after one shift right, this zeroth bit will anyway will go away. So, this

particular method of addition, will ensure that there is always a space there is this both multiplier and our product can be stored in the same register and both the operations are also same, and this also reduces the size of my adder. So, my addition would be only 32 bit instead of 64 bit.

So, this not only reduce theone additional register, which we were using for multiplayer and it it also reduces the delay because instead of 64 bit edition, we are now doing only 32 bit addition. Rest of the design is a same. So, the rest of the design is same means my control FSM is still same, we are always doing the right operation, there is no left operation required in the multiplicand because we are always doing the keeping the same 32 bit of the multiplicand. This add would again depend on the zeroth bit of this product register.

So, this way it could be more optimized, and this product 0 would help us in deciding whether to add or whether to write to this product register or not, and rest everything is is same. So, whenever we are doing addition, we are only taking the upper half or basically upper 32 bits of thisand would be added to the multiplicand, if product zeroth bit of the product is 1. So, and control is similar to our previous example, it is 32 Modulo 32 counter.

Now, as a conclusion, what we can see that this particular addition will consume 32 cycles to perform one multiplication, if multiplication is a 32 bit multiplication. So, 32 cycles would be required and 1 cycle would be equal to the delay of one 32 bit edition plus setup time and propagation time for register. And total area requirement would be we can say that the 64 bit register, this 32 bit register and 32 these two are the register requirements.

Now, these register requirements we can say would anyway would be there because whenever we are multiplying two numbers we will be storing their results in a register, and the input we are also expecting from some register. So, we can say this this requirement could anyway would be there whenever we are doing multiplication, but for the sake of writing for sake of completeness, we have written that these two registers are the requirement. In addition we will have 32 bit one 32 bit header andcontrol is simple, we only require 32 Modulo 32 counter over 5 bit counter.

So, this, what would be the advantage over the other multiplayer. So this multiplayer is definitely very, very less in terms of area. And the hardware is also or the connections are also quite simple. The state machine is straightforward 32 Modulo 32 counter and where every time we are going to shift right so there is no multiplexers involved, there is no other logic involved.

We are going to write only if product 0 is 1, then only this edition would be returned. So, that way this this multiplier is quite simplistic, although, it takes 32 cycles. So, could be utilized whenlatency is not an issue. One more thing to point out here that this particular multiplication cannot be pipelined because this addition is used almost in every cycle. Because it is using every cycle this product register is using every cycle, multiplicand is used in every cycle, so there is no other optimization other than this.

So, pipeline optimization can also be (possib) can only be possible whenever resources are free and when you have sum. For example, here we have at least utilize these resources product register has been multiplexed it is doing both the operation. It is keeping the multiplier, as well as keeping the result so, this way we are trying to have some operation. So, with this, this motivation in mind let us look at the other design also, how a division would be done.