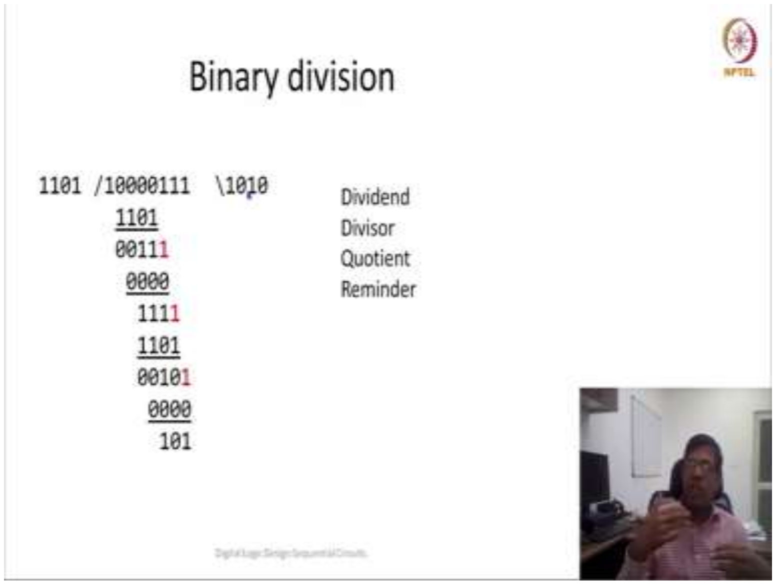
Digital System Design

Professor. Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture No. 59
Division Hardware Design.

(Refer Slide Time: 00:15)



Binary division

```
1101 /10000111  \1010
       1101
       00111
        0000
         1111
         1101
         00101
          0000
           101
```

Dividend
Divisor
Quotient
Reminder

So, division for starting with the division we have at least three terms, we have to understand. One is dividend divisor and quotient. So, dividend is something which needs to be divided and it will be divided with the help of divisor and whatever is the result that would be called quotient.

Now, how do we do division? The division would be done in a same way, the way we used to do a decimal division, the way we used to we have learned in our primary classes. So, what do we do? We say that let us say this is 4 bit division, so 4 bit divisor, so we will first start with the first 4 bits of the dividend, and we try to see whether we can subtract this number from this or not.

Because we cannot, so we will take the 5 bits so the from these 5 bits, we are trying to subtract our divisor, if we are able to subtract. If the answer is positive, then we are make one bit of our quotient as 1. So, we write 1 here. So, you see this 1 we are writing in the most significant place of the quotient. Now, after, then we do subtraction. So, after doing subtraction, we take 1 from the top, so 1 from one additional one from the dividend and then we try to see whether this remaining number whether it can be subtracted from the divisor or not.

So, we see that if we subtract then the output would be negative, so we say we cannot subtract. And because we cannot subtract the number will remain as is. And in the next cycle, we take one more bit from the dividend, and then we see whether we could now subtract or not. So, because now we are able to subtract, so we have written 1 here as a third bit.And after subtraction we get 10 and then this another 1 we take from the dividend, this become 101.

Now again, we will try to subtract whether we are able to subtract or not, because the output of the subtraction is going to be negative, so we do not subtract and we write 0 here in the in the quotient. So, the final subtraction would result whatever result is there that would be called a reminder. So, this is the process we were learned in our primary classes to do division. The same method we can use here also for doing binary division by using addition subtractions. So, what you will see here that couple of things to note.

So, one thing that whenever we are generating quotient it is generated as most significant bit then the then towards the less significant bits. The other thing that when we have started subtractionso we have subtracted our divisor from the most significant bits of dividend. So, what we can do is, we can consider that the we can also see it like this that the divisor is this and the rest of the bits are 0.

So, this is a 9 bit dividend this is a 4 bit divisor, so we have to consider here that rest of the 3 bits are all zeros, and then we are trying to subtract. And after that subtraction whatever is the result, here, we in case of a digital circuit, we will always have to subtract and then we have to see whether it is negative or not.

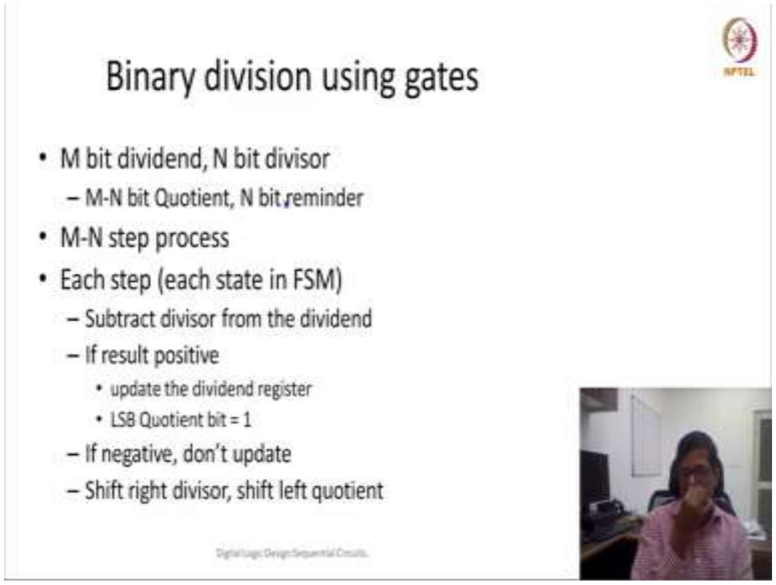
If it is positive, if my result is positive, so that means we are able to subtract that means my quotient will become 1. And because quotient will become 1, then we are able to subtract. If the result of subtraction is negative, then we have to restore, we have to say that whatever was the previous result of subtraction that will keep and we will take one more bit from there and we'll add 0 as a quotient also.

So, this process is again, there are couple of things to notice that a couple of more things you guys notice that this is a cyclic process. This is a repetitive process in each step. So, this in each step, what we are doing in each step we are trying to compare. We are trying to compare some N bits of my dividend some selected N bits starting from the most significant bits, we are comparing those N bits and then trying tosubtract our divisor, and if the result is negative.

So, this is one thing, which we are doing every time we are trying to subtract. And then if the result is negative, then we are making that particular bit of quotient as 0 if result is positive, we're making that bit as 1. The other thing we are doing is we are also shifting our quotient our divisor in the right direction. So, initially this was here, and then it was shifted 1 bit in the right direction here it was shifted 1 more bit in the right direction, here again it has shifted 1 bit one more bit in the right directions. So, every time my divisor is shifted in the right direction.

So, and if we are storing our result as a quotient, so, we can also say that first time my result was more significant bit than more significant with bit minus 1 then most significant minus 2. So, that way, we are also trying to for quotient we are always shifting in the left direction and then writing to the least significant bit.

(Refer Slide Time: 06:34)



Binary division using gates

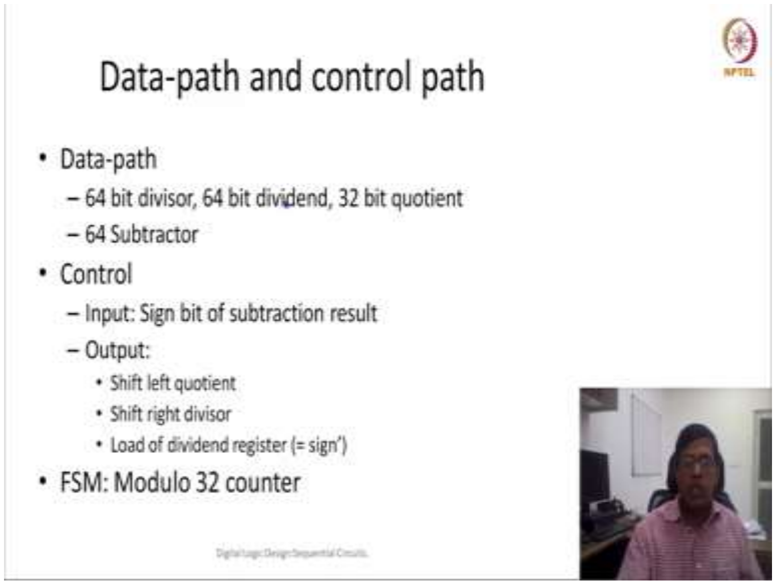
- M bit dividend, N bit divisor
  - M-N bit Quotient, N bit reminder
- M-N step process
- Each step (each state in FSM)
  - Subtract divisor from the dividend
  - If result positive
    - update the dividend register
    - LSB Quotient bit = 1
  - If negative, don't update
  - Shift right divisor, shift left quotient

Digital Logic Design Sequential Circuits

So, I can summarize all of these things like this that if I have M bit dividend and N bit divisor, the total number of bits in my quotient is going to be N minus N and there would be N bit reminder, and the overall processes total number of question bits M minus N. So, each step we can say that each step is one state of the Finite State Machine.

So, in each state we are subtracting divisor from the dividend and the same thing is written here. If the result is positive, then we are updating the dividend register, otherwise we are not and least significant bit of the quotient is 1 if the result is positive, and if it is negative, the least significant bit of the quotient is 0 and we do not update the dividend. And we are always shifting right the dividend and we are always shifting left the quotient.

(Refer Slide Time: 07:32)



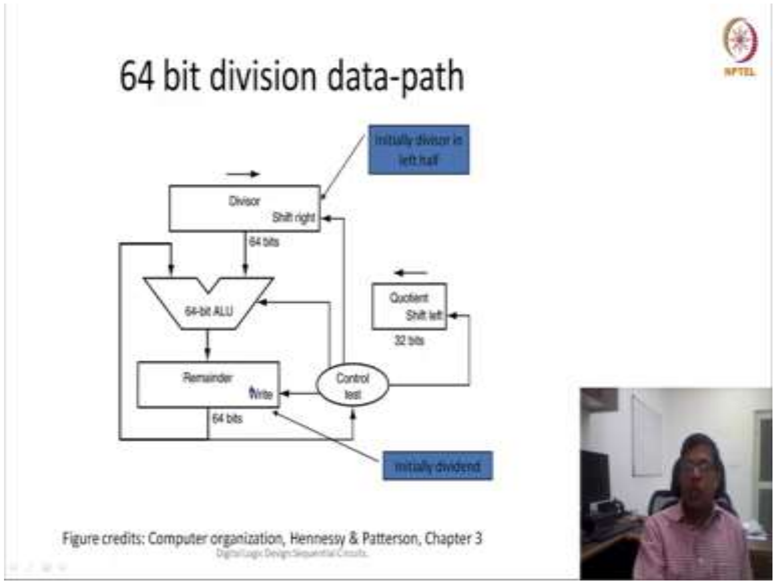
Data-path and control path

- Data-path
  - 64 bit divisor, 64 bit dividend, 32 bit quotient
  - 64 Subtractor
- Control
  - Input: Sign bit of subtraction result
  - Output:
    - Shift left quotient
    - Shift right divisor
    - Load of dividend register (= sign')
- FSM: Modulo 32 counter

So, let us see, if we see the overall process, we would let us say we are trying to have a 64 bit divisor. So, let us have 32 bit of divisor and 64 bit of dividend, and then we would get 32 bit of a quotient and 32 bit of the remainder. Now, here to keep 32 bit of divisor we are using 64 bit register because we have to do all the shift write operations. So, that is why 64 bit divisor is written over there, but actual divisor size is 32 bit only. So, we would also require a 64 bit subtractor here.

And if we try to see the control bit that that Finite State Machine which has 32 states it is taking a result of my operation subtraction operation as ainputif the result or basically sign bit. If a result is saying it is a negative, then we are loading into a dividend register or not otherwise, we are doing the shift left quotient and shift a divisor every time, and finite state machine like multiplayer is going to be a Modulo 32 counter. So, basically a 5 bit counter, which would start from 0 state to state number 31.

(Refer Slide Time: 09:08)



64 bit division data-path

Figure credits: Computer organization, Hennessy & Patterson, Chapter 3

Now, let us look at the data path diagram. So, this diagram is also taken from Hennessy Patterson Computer Organization book. So, what we are doing, we are keeping the divisor initially in the left half, upper half. So, in the upper half we have the divisor, and this particular register is initially the dividend all 64 bit of the dividends are stored here. Andnow when we are subtracting this 64 bit dividend here so what we are trying to do we are trying to divide or we are trying to subtract upper 32 bits of divisor and with the upper 32 bits of dividend.

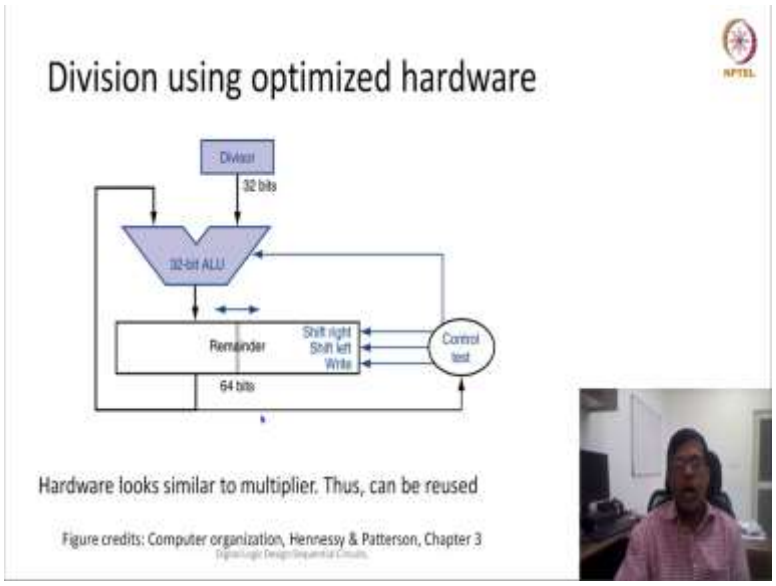
So, these are the upper 32 bits, and these are also upper 32 bits, although,it is 64 bit subtractor but essentially, because my divisor is now stored in Upper 32 bits, so only those will get subtracted. Now, if the subtraction is found to be negative, then I will check what is the sign bit, if sign bit is found to be 1, that means the equals negative, then I will make my quotient as 0. If sign bit was positive, that means I make my quotient as 1 and we will keep here as the least significant bit.

But because we are shifting the quotient every time in the left direction, so thisquotient, the first bit of the quotient will become more significant bit in in 32 cycles. So, every time divisor is shifted in the right direction by 1 bit every in every cycle in the next 8, so divisor will be shifted 1. And similarly, now, if the result of this subtraction is negative, we will not write to our dividend or reminder register, we will not write here. So, what does that mean? It means that the dividend would remain same.

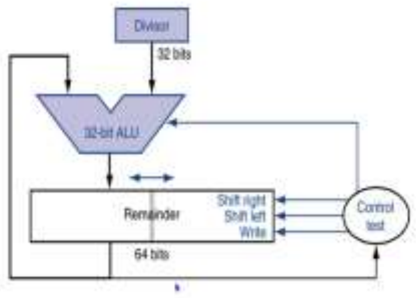
So, if you see this example, so, let us say if it is not written then we will consider it as 000. So, 111 remain 111, and then one more bit would be borrowed from dividend. So, writing not writing essentially means that dividend still remains same there is no change in the dividend. So, this is how we can we can do the 64 with division by a 32 bit divisor. And yeah, so, after 32 cycles we will have the results there in the lower half of my reminder register and the 32 bit of quotient will give me the correct results here.

Now, there are a couple of observations more observations here that one thing that this diagram look quite similar to multiplication diagram. So, that means, that the same optimization would be possible here also.

(Refer Slide Time: 12:42)



Division using optimized hardware



Hardware looks similar to multiplier. Thus, can be reused

Figure credits: Computer organization, Hennessy & Patterson, Chapter 3

So, as optimization, what we can do iswe can write we can use the quotient register. So, what we can do is initially, we can keep divisor as 32 bit and my 64 bit of dividend would be stored in this particular register. And every time my division would be there, every time I would do subtraction, this would be shifted in the left direction.

So, when we are shifting it in the left direction, so, the shift left would be taken every time, so when this shift left is taken. So, if the result is if the result of the subtraction is positive, this result is positive, then we will write 1 in the least significant bit of this register, and we will every time we will keep on shifting it in the left direction. So, first we will write 1 and then we will shift in the left direction.

So, this way this this division would keep on happening the suppression would keep on happening. And by the end of 32 cycles, my remainder will be there in the upper half of this register and lower half of this register would have quotient. So, like in multiplication, this register has been multiple has two uses. It is now used to store quotient as well as it is used to store a reminder.

Now, one more observation here that my divisor is again 32 bit, there is no shift operation required in the divisor and this ALU is right now performing the subtraction operation. So, other thing is also evident that this diagram looks exactly the same, as the diagram which was used for multiplication. Even the control circuit is same. So, in multiplication we were performing the shift right operation. Right now, for division we are performing shift left operation.

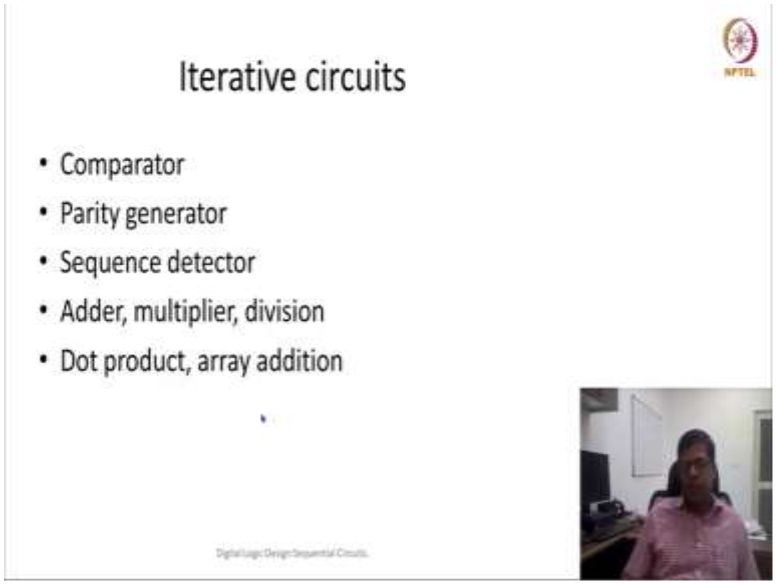
And the condition for right is different. Here condition for right is that if the subtraction is negative or not then we are going to write otherwise, we were not writing. However, in case of addition the right condition was depending on the last bit of the multiplier, if (multi) last bit of the multiplier zeroth bit of the multiplier is1then we were writing otherwise we were not writing.

So, but because the circuit of both these multipliers and adders are same, so, what we can do is the same hardware can be used for multiplication, as well as the addition depending on the control. If I am able to use a arthritic logic unit, which can do addition as well as subtraction, then rest of the hardware, rest of the control logic is more or less overlapping. So, the same hardware can be used for (addition) for multiplication as well as for division. So, that is the anotherbeautiful observation we will drive from these observations.

So, the state machine it would be same as the multiplier it is a 32 state,32 state machine or 32 states state machine, which means a 5 bit counter and which is because states are always looking for measurable S0 to S1, S1 to S2, etcetera.

So, the other thing, which we can also conclude from these discussions that we have been trying to divide our circuit into two parts, one is data flow, another is control flow. This control flow is trying to control the data paths circuit data path is at what control I need to provide to ALU for in eachstate and when should I write, when should I read, when should a left shift or right shift my register so, all of those things would be decided by in which state I am, and what operations I am doing. So, all those things could be separated as a control path. And wherever we are operating on the data that we can keep as a data path.

(Refer Slide Time: 17:16)



Iterative circuits

- Comparator
- Parity generator
- Sequence detector
- Adder, multiplier, division
- Dot product, array addition

Digital Logic Design Sequential Circuits

Another conclusion from today's discussion as well as the previous lecture is that we can categorize all of these circuits, as I iterative circuits. You remember, when we were discussing the, when we were discussing the combinational circuits Module 3, then we have designed compared to using one cell which was repeated again and again.

Here also it is the same thing in comparator the same cell is again and again repeated again and again, but what we are doing, we can do the same task using the one cell of the comparator where results are being stored in the register. So, the same comparator which was used as a iterative circuit in combinational design required 32 cells here, this if one cell is used here, and that can perform the same operation 32 times using sequential circuits.

One thing to be noticed, one thing to be to be kept in mind whenever we are designing this. So, when iterative circuits were designed using combinational circuit, we specifically designed what is the boundary condition. Here, those boundary condition would still be decided, but those boundary condition would be would be generated using state machines. So, let us say boundary condition here for a 32 bit comparator would come whenever I am in state number 32 or whenever I am state number 1.

So, state number 1 would be my first cell and state number 32 would be my end cells, so the last cell. So, in those conditions, there could be different inputs and different output or different operations, but otherwisethe same cell would keep on repeating. So, since all the examples which you had done using the iterative circuits in combinational circuits are equally applicable here also, but the difference would be that here, all those things could be done in multiple states or each cell can be performed its computation in one of the cycle and the rest of the cycles can keep on performing the other.

So basically, other cells would represent the other cycles, clock cycles or other clock steps or other control steps of the same circuit. SoAll the examples would you have done that in that lecture competitor, parity generator, sequence detectors. Adder, multiplier, divisor we have done today. So, all of them are also iterative circuits. So, basically if we are able to do one particular state, then the same state is getting repeated again and again.

The other examples, which we have mentioned was dot product or array addition. So, all those things are also iterative circuits because you are doing one other state and after that, in forthcoming states, you keep on performing the same operations. And the control circuit for all these items circuits are simple, so it is going to be always Modulo N counters, because they are always going from state number 1, to 2 to 2 to 3, 3 to 4 so there is no branching or there is no other arrows, which are there in the state machines.

So with this, I would like to close today. And this is also almost the conclusion of our finite state machinesdiscussion. Now we will start in the next lecture more generic design of a sequential circuit. So, till then, goodbye. Thank you very much.