


Digital System Design
Professor. Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture No. 56
Multi-cycle adder design

Hello all. In our previous couple of lectures, we have seen some of the applications of sequential design and the important most important one the one which you have studied in last couple of lectures were sequence detector. So, for designing a sequence detector the approach used to be that we were first designing a state machine and then state machine or a state graph and then state table and then implementing it using it.

But, there could be other interesting applications also. So, we will pick up the old topic which we have already done arithmetic design, we have already learned how to do addition, multiplication and various kind of adders and multipliers we have seen. So, in this lecture as well as the next lecture, we will focus again on these arithmetic units. Now, from the sequential design perspective.

When we are looking these designs from the sequential perspective, so, we will try to see that what kind of benefit can be there in these designs, this adder, multiplication or any arithmetic circuit for that matter, if they are designed using sequential circuits, then what kind of advantages we can get.


(Refer Slide Time: 1:46)



Multiple cycle adder design

- Add using one full adder
- Input: Two N bit numbers, output: one N bit number
- Accumulator: $A = A + B$ Use of registers?
Use of Shift registers?
- Questions:
 - Where to store N bit numbers?
 - How single adder will add N bits?

Digital Logic Design Sequential Circuits



And one more important point to note here that whatever techniques we are learning in these two lectures are generally applicable for any kind of a design. Combinational design can be converted into sequential design using these techniques.

So, the objective let us say the first objective is, we would like to design an adder which is which the addition is done over multiple cycles. Why would we like to do that addition in multiple cycles? So, you have seen that one cycle, one clock cycle depends on the latency of my adder. So, if latency of that adder is large, then we will not be able to. If the latency of that adder is larger, then the clock period will also be large.

So, the solution to that could be that, if we do the same addition over multiple cycles, then we can reduce the clock period. So, that is the overall objective. So, let us try to do it in a very simplistic manner. And first, the first objective is we would like to design any N bit adder using one full bit adder.

So, if you want to do one full bit adder one N bit adder using one full bit adder, so, the inputs are of course, you have N bit numbers two N bit numbers and output is one N bit number additionally carry C out is also an output. So, let us to make the problem simplest, even more simple. So, what we are trying to do is we are trying to design an accumulator where the output is also saved in one of the operands.

So, let us say two operands are A and B and the result is also stored in the A. So, when we are saying A and B are two operands so both of them are N bit numbers and result is also in N bit which is saved in A. So, since, we would like to design N bit addition, N bit addition, and on the other hand we are also seeing that we have to do this addition only using one full adder. So, one full bit adder will be adding only 3 bits at one time, one bit from A another bit from B and one carry in, and we will be producing only one bit of sum at one point of time.

So, we need to store, and we are we are saying that this addition would spread over multiple cycles, so that means we have to store these numbers A, B somewhere. So, these N bit numbers has to be stored. Where shall we store that numbers? So, we since we have already studied registers in flip flop. So, what we can do is we can use N flip flops and create a register and we store them in a register.

So, the other question, which comes to our mind that, yes, we can store these A and B in registers the other question is that now the adder is a single bit, and we would like to add these N bit numbers. So, most likely the thing which is coming to our mind is that we would

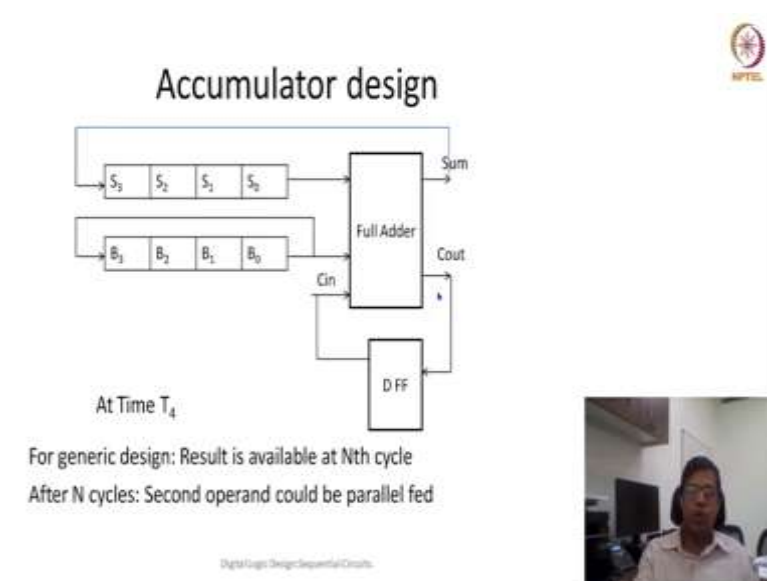
probably add bit by bit So, we can probably add bit number 0 then bit number 1 then bit number 2 then bit number 3, so on so forth.

So, that means all these bits would be probably multiplexed to our full adder. So, each input of the full adder then would be multiplexed and they have to done sequentially. Because you know addition of second bit depends on addition of first bit and addition first bit depends on additional zeroth bit because there is a carry chain which is going on. So, because of the dependency you cannot do all those things in any order. So, it would be in a certain order that first with 0 then bit 1 then bit 2, and then Nth bit N minus 1 bit.

So, and all of these bits has to be supplied to our adder, single bit adder. So, what should we do? So, one option which comes to mind is possibly you have to use some sort of multiplexer, which would keep on choosing that which particular bit to be selected at any particular point of time.

Now, there is one more idea you see that because the order of bits is certain, where this this 1 bit full adder is going to first add bit number 0 then only it can add bit number 1 then bit number 2. So, can we use shift registers? So, the shift register will give us a certain advantage that at one point of time only 1 bit will be added and then if it is zeroeth bit then after that first bit will come automatically then second bit will come automatically after every clock cycle or when the bits are shifted. So, we will try to use shift registers. And if this whole picture is in our mind, because we are talking without visually looking at it.

(Refer Slide Time: 06:52)



So, let us try to see that how that accumulator. I am calling it accumulator rather than adder because we are saving the output again in the same register. So, let me consider that this probably looks like the basic starting point, the design that I have these two registers. So, to make the problem simple right now, I have taken N equal to 4. So, it is a 4 bit adder but this N could be generic, so this whatever we are learning here, it could be applicable to any N .

So, you see that these A and B are 4bit registers, actually 4 bit shift registers. So, my full adder is connected to least significant bit of both the registers. So, essentially it is you can also say this is serial in serial out register, which is shifting always in the right direction. This is serial in serial out register. So, basically, the out part is always LSB and the this is the serial in part so that is a shift register thing. So, now, these three bits would be added. This is bit 1 the operand 1, operand 2 and C in would be added and the output would be sum and carry.

Now, where should we store this sum? That is also a question. Because we were saying that this sum will actually replace this operand 1, operand 1 is A in this case, so the sum has to replace operand 1, and we otherwise have to use a new register here and then we have to replace that register. So, and also this is one problem that is sum where to be stored, storing the sum.

The second thing is that, when we would be adding the second bit let us say A_1 and B_1 , then whatever is the carry out that should be repelled to the carry in of the next bit. So, what we can do is, we can have a flop here, and we can have a flop here D flip flop and every clock cycle this C out will become C in. So, in the first cycle, the C in could be zero or we can have a D flip flop which is reset.

Reset means, reset to 0, so that means C in would be 0, but often the next cycle onwards the C out would be fed in, and so this will become C in. So, for example when we will see that when the bits would be shifted then the C out will become C in. And also we are connecting this sum to the serial in to the most significant bit. So, why that would be helpful? You will see that by end of the computation it would be able to replace this whole A register and we will have the sum is a sum of the 4 bits.

So, far so good, so we are able to, this is the minimalistic design and we can, it looks like that we will be able to add. So, let us try to see in the execution cycles that how it would be happening. So, now, one more feedback path would even more be helpful that let us say this

B is also feedback to this like there is also serial in serial out resistor B is also a serial in serial out register. So, what would be the benefit?

By end when the competition is finished, then we would be able to receive. So, this B0 otherwise if it is given here and we are shifting it, then B0 would be discarded. If B0 is discarded then it may be required in later point of time also. So, the better idea would be that let us save it, and let us save it in the same register by giving it as a shift rotation. So, in cycle 0 at time 0, this would be the configuration where A0 and B0 would be adding A0, B0 and C in. So, at time 0, C in would be the reset value of D flip flop and then we would be completing some and C out.

So, at the end of T0 and when the next active post, active cycle, active edge of the clock cycle will come so this sum would be given as a input to this shift register, and B0 will be given as input to this shift register and C out will be given as input to D flip flop. So, what would happen at time 1 basically after the active edge of the clock cycle, so S0 will come here, the first some bit, which was the addition sum. And A3 would move to this bit and A2 would move to this and least significant bit will become A1 and B1.

So, similarly, because of this parallel in serial in serial out, so, basically B0 came to the become the most significant bit. Now, during this time period T1, A1 and B1 would be added and C in is actually whatever was the C out generated during the first zeroth clock cycle. So, and then after by the end of this clock cycle we will have S0 shifted to right, S1 would be the most significant bit and similarly B1 will become the most significant bit. And so A2 and B2 would be the least significant bits. C out and C in would behave in the same way.

So, similarly after T3 and T4. So, after T4, you will have all the 4 bits here as the sum. There is no K, because A, the overall design was A equal to A plus B. So, now this becomes the new A and we have again received B, as it was the original point. So at time equal T4, so you have to be, you to notice that this T4 is the beginning of T4. So, we have done computation in T0, T1, T2 and T3. And by the end of T3 or in the beginning of T4 we have this sum which is completely ready and our register B or the other second operand is again in the original position where we started.

So, this way using a single full adder we would be able to design any N bit adder in nth number of cycles. And the additional cost you see is having this register. So, two registers one flip flop, and this way you will see these registers are anyway would be required whenever we are doing addition or multiplication or any other operation because we would

be doing everything within a clock period, and we will be saving the operands in some registers.

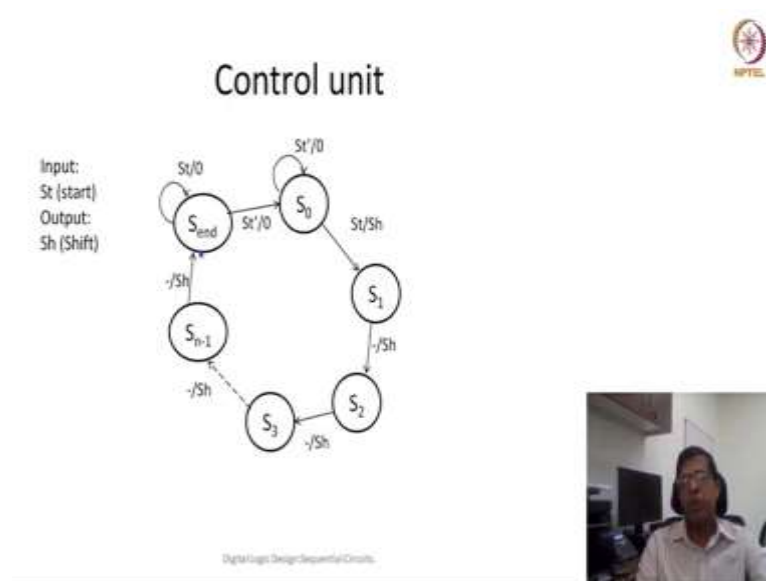
So, that means this essentially is not an additional cost only one D flip flop and one full adder is able to design a complete 32 bit or any N bit adder, which is quite a significant thing. So, we are able to reduce the area to a very minimal number by using this kind of a sequential, we call this kind of adder as a serial adder. Accumulator is not important because we can have another register as a sum here, which could be shifted always in the right direction and by the end of N cycles, we can have sum as another register.

So, we did accumulation for simplification here, but we could have sum as a different register also so that was not a constraint at all here. So, you can see the overall benefit. Area has reduced to a significant amount, but the latency has increased. The overall latency of this design is the latency of one full adder plus the latency of my D flip flop. And latency of D flip flop would also involve a set of time of D flip flops. So, that means this this would can reduce can increase the clock period to very significant amount.

Now, so, this is a very generic design and the result would be always available at the nth cycle. Now, let us see that so one more point here that what if we would like to have a new operand here, new operand in A or B. So, that means these registers are not only serial in, serial out, but they also should be universal register, so that they could be parallel fed. So at that time, whenever you want to add a new number instead of B there is a C, so that C could be a parallelly given as an input to this register and similarly, some of the register ABC could be given as a register as an input to this register, so all the 4 values could be changed at the same time, and then after that, we can start.

Now, this looks like the way we have discussed it was mostly the combinational part, or specifically, we call it a data path part where we are doing the data computation. But since it is a sequential design it depends on multiple cycles. My whole design or my whole calculation is done in N number of cycles. What would be the control steps or what would be the finite state machine for this particular design?

(Refer Slide Time: 17:08)



So, let us look at the finite state machine. So, initially, so finite state machine, first of all, let us, let us very clearly specify what are my inputs and outputs. My input is, let us say start St and my output is a Sh Shift. So, this is the input and output of a control system, the control the controlling part of my, this is the input and output for my FSM Finite State Machine.

Now, let us have a little more specification of the input output also. So, when start is 0, nothing should happen when start is 1, then it would kick in the addition. And after first cycle of start, it does not matter what is the value of start, then because then addition has started it will only finish when N cycles has elapsed. And shift would mean that went this A and B registers to be shifted, when they are not to be shifted.

So, let us start with a reset state S_0 . In the reset state my flip flop, which flip flop which was saving the which was saving my C out or carry, that should be given as a reset. And along with that, I would be always waiting for a start. So, if start is 0, if started 0, then it would still remain in the state. And 0 here, 0 as output means that shift is 0. So, that means no shifting would happen, if there is no start. So, that means my A and B whatever operands are there in the register, they will remain there, they would not be shifted.

Now as soon as start would be 1, then I would shift and we will go to a next state S_1 and then after that, after every clock cycle it does not matter whether what is the value may start whether it is 0 or 1 I will be always giving Sh or basically Shift as 1. So, that would keep on happening till n th N minus 1 cycle, so that my result is almost there.

So, after $N - 1$ cycle, so that means I have rotated it so this is the, so after $N - 1$ states we have computed the $N - 1$ bits. So, here bit number 0 was calculated here bit number 1 was calculated here, bit number 2 was calculated here, bit number $N - 1$ was calculated, 0 $N - 2$, so that means one more bit need to be calculated. So, that means this is my bit $N - 1$ or the last bit where it will be computed. So, here I would say that this is the last step. So, where all the computation has been done 0 to $N - 1$ all the bits has been computed. So, what should be done here?

Here, one thing is sure, that when I reach the state S end or the last state then there should not be any shifting unless even if there is this the start is 1. So, if even if start is 1 there is no shifting which is happening. So, you see that we started here in my S_0 initial reset set when start was 0 then after that start was 1 then we started.

After that start was do not care, we do not care whether it is 0 or 1, but somehow if it is 1 here and we reached this 0 state that means we will automatically will go to S_1 state and all others. So, that is why we are saying that if start is 1 and we have finished the computation then we will keep on remain, we will remain in this particular state.

Our start is 0 then we will go to the reset state and we will say that we are still in the we are not shifting anywhere. So, one more important thing to notice here the state machine is written in a slightly different format. So, far what we are doing in whenever we are writing the state machine we were writing here 0 or 1, 0 or 1 now instead of 0 or 1 we are writing the variable name directly. What is that? And why it is so?

So, this is again, a more visually correct way or visually different way of representing the state machine where we need not to bother that whether it is 0 or 1 we focus on the variable name, and by focusing on variable name, it helps us to recognize that whether we are doing shifting for example here or not shifting. So, what we do also associate that whenever we are seeing S_h , then it means shift and whenever we are seeing S_h dash that means no shifting. So, this way, this is another way of representing a state machine where instead of writing 0 and 1 we directly write the variable names.

So, do not get surprised when we get such kind of a state machine representation where instead of 0 and 1 instead of direct binary values we are writing the variable name. So, these variable writing variable name has another advantage when number of input and number of outputs are little more, more than 1.

So, another thing that if I would like to implement this state machine, then you will see that we can implement this easily using an N bit modular N counter. So, because modular N plus 1 counter actually, so this particular state machine can be implemented using modular N plus 1 counter. So, from 0 to 1 we are always shifting. So, because we are always shifting so we can implement this part using counter. So, this is the, this is addition circuit using single bit.