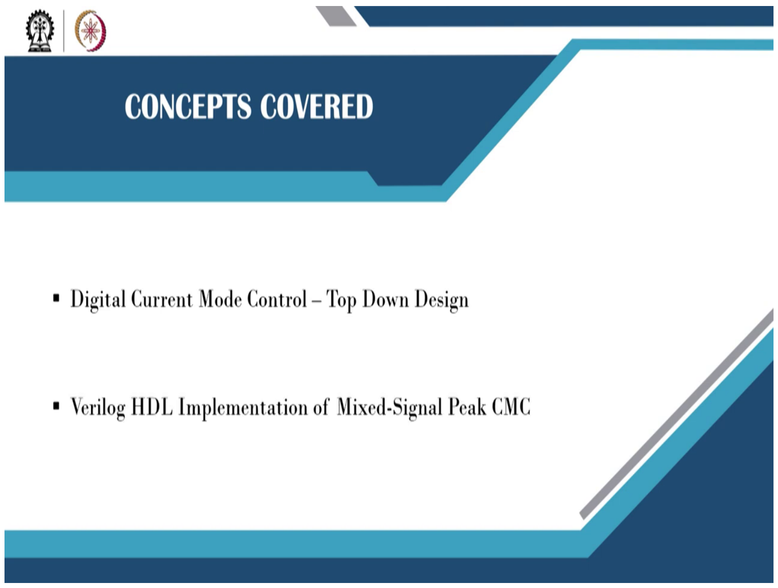
Digital Control in Switched Mode Power Converters and FPGA-based Prototyping

Prof. Santanu Kapat
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Module - 08
Digital Controller Implementation using Fixed-Point Arithmetic and Verilog HDL
Lecture - 76
Top Down Design Method and Verilog HDL Programming of Mixed-Signal CMC

Welcome back. In this lecture, we are going to continue the previous lecture and we want to show the Verilog Programming of Mixed Signal Current Mode Control Implementation.

(Refer Slide Time: 00:35)

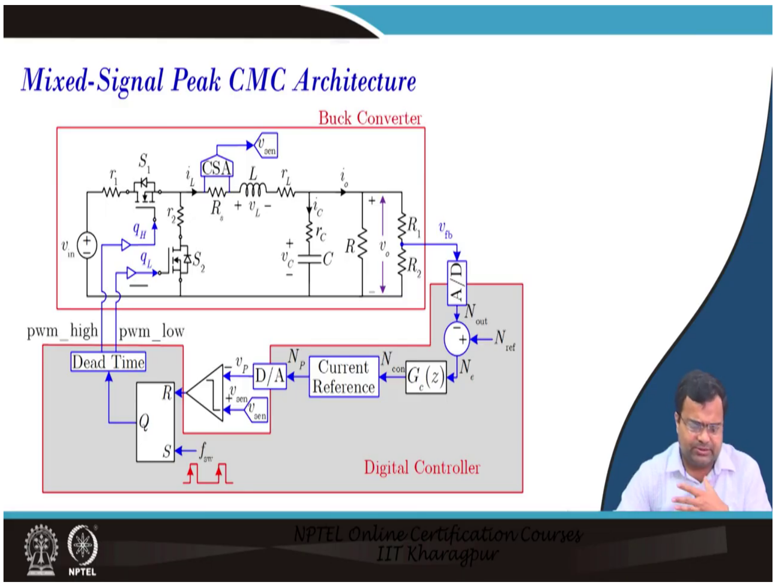


CONCEPTS COVERED

- Digital Current Mode Control – Top Down Design
- Verilog HDL Implementation of Mixed-Signal Peak CMC

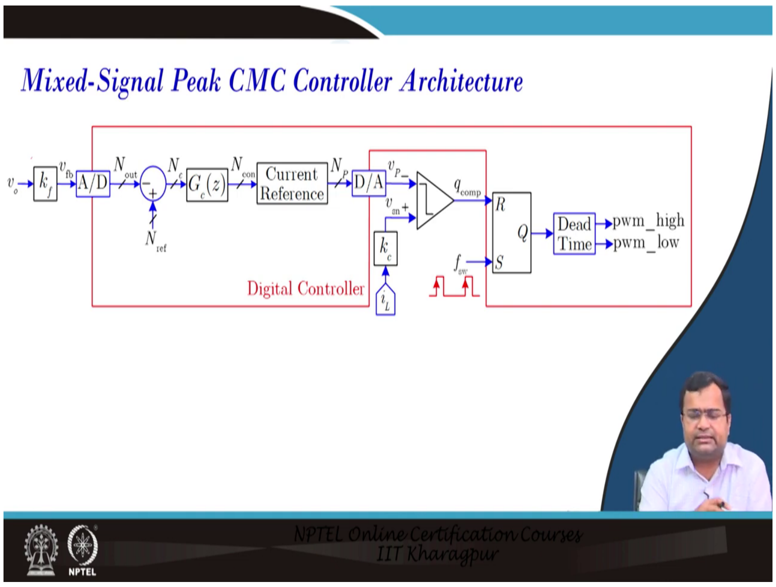
So, here we are going to talk about digital current mode control, top-down design, and Verilog implementation HDL implementation.

(Refer Slide Time: 00:41)


Mixed-Signal Peak CMC Architecture

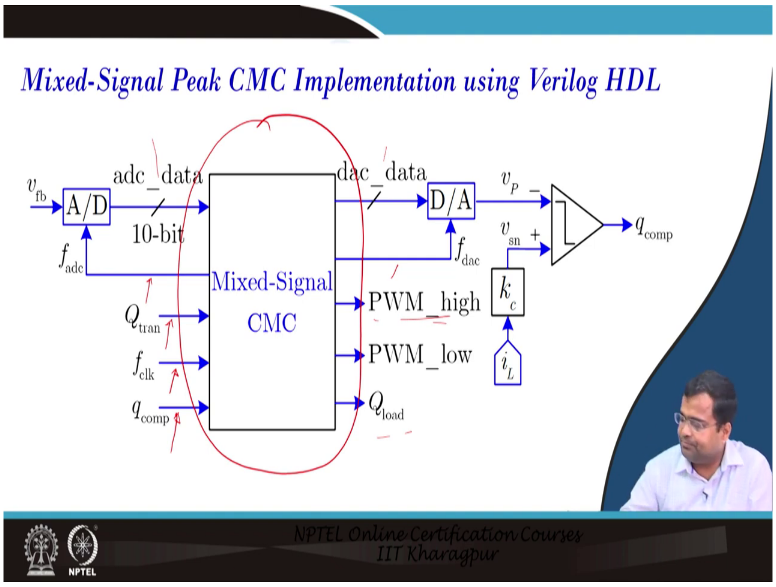
So, again this block we have discussed, what is the architecture of mixed-signal current mode control.

(Refer Slide Time: 00:47)


Mixed-Signal Peak CMC Controller Architecture

And, we have also discussed in detail why these factors are coming for real implementation, why we need ADC and DAC. So, this part we have already discussed.

(Refer Slide Time: 00:57)

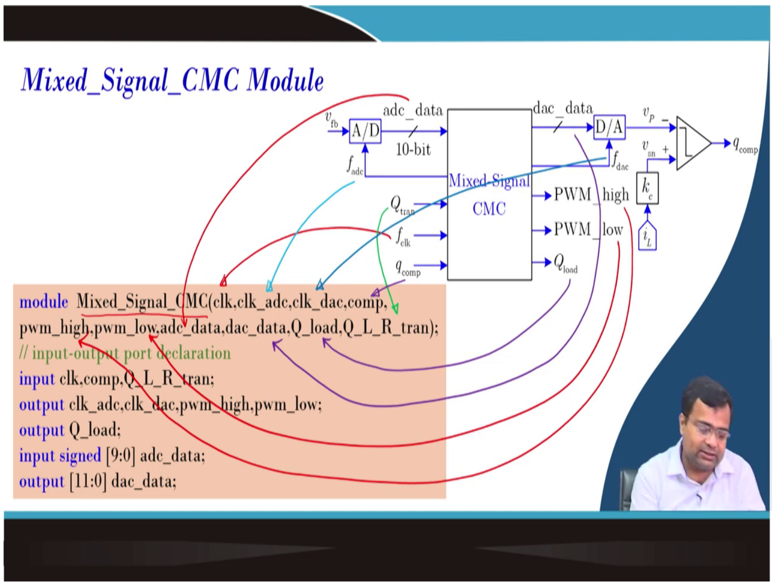


Mixed-Signal Peak CMC Implementation using Verilog HDL

And, our objective to make this module is the main module that will interface with the external world, where we need to take data from ADC, we need to send a clock to the ADC. Then, we need to accept the command from outside whether you want to make load transient reference transient.

We need to take the controller clock from outside, then we are taking the comparator output from the analog comparator. Then, we are sending the data out to the DAC, ok, then we are sending the clock to the DAC. Then, we are sending the gates, we are generating the gate signal for the high side and low side and this will go to the driver. And, we are also generating the load transient gate signal which will also turn on and off the switch load.

(Refer Slide Time: 01:42)


Mixed_Signal_CMC Module

```
module Mixed_Signal_CMC(clk,clk_adc,clk_dac,comp,
pwm_high,pwm_low,adc_data,dac_data,Q_load,Q_L_R_tran);
// input-output port declaration
input clk,comp,Q_L_R_tran;
output clk_adc,clk_dac,pwm_high,pwm_low;
output Q_load;
input signed [9:0] adc_data;
output [11:0] dac_data;
```

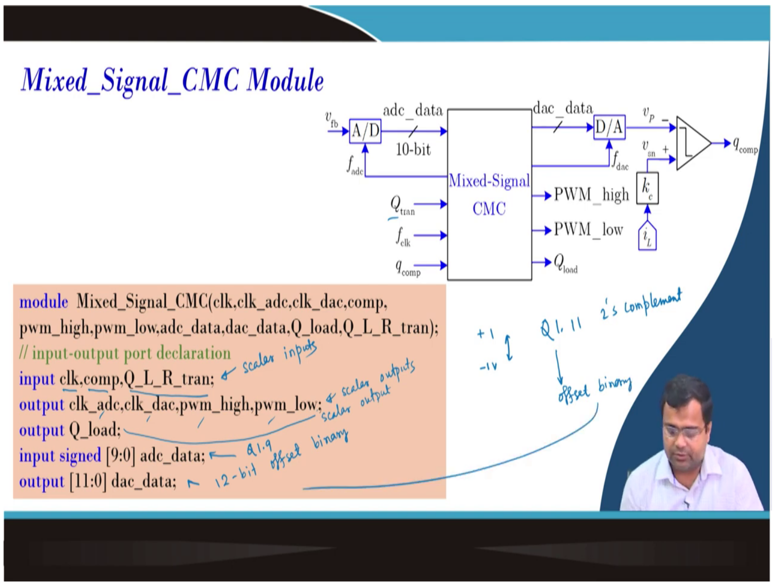
And, this is the main module that we have to design. So, we make the main module name Mixed Signal Current Mode control and you see, what is the input? So, this will be defined by this clock ok and then you can see so, I will use a different color for mapping. So, the data is coming sorry. So, next, we are dealing with the clock first. So, this one is the ADC clock. So, it is the clock for this ADC ok.

Then, if we use a transient Q tran so, this we are denoting as this ok, then Q comparator. So, the Q comparator we are using here. This is the comparator signal. So, the left side is over. Now, the Q load is going here. The DAC DAC is going here. Then, adc data is the data, then PWM is low. So, you can take this PWM low here, PWM high it is going to here PWM high. Then, what is left? The clock to the DAC. So, you can use another color.

So, the clock to the DAC; means, this particular signal. So, you have marked all the signals in the code; that means, these are the interface of this main module. And, we will show when you go to the live demo, this requires a definition, a clear definition of the User Constraint File, UCF file. So, that it will know the pin address on the actual board where these pins are located, what is the address.

So, it will send those signals or take the signal based on whether it is output or input accordingly. But, once it comes through the UCF file to the Verilog module, then after that what else is to be done? So, that is what we are going to discuss.

(Refer Slide Time: 04:06)



Mixed_Signal_CMC Module

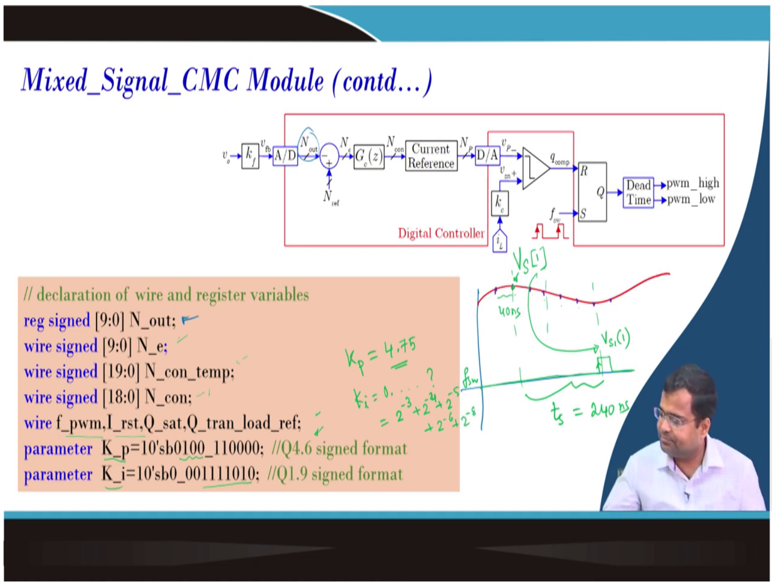
```
module Mixed_Signal_CMC(clk,clk_adc,clk_dac,comp,
pwm_high,pwm_low,adc_data,dac_data,Q_load,Q_L_R_tran);
// input-output port declaration
input clk,comp,Q_L_R_tran; ← scaler inputs
output clk_adc,clk_dac,pwm_high,pwm_low; ← scaler outputs
output Q_load;                            scalar output
input signed [9:0] adc_data; ← Q1.9
output [11:0] dac_data; ← 12-bit offset binary
```

So, in the main module, we have to define the clock to be input, comparator input, and Q_L_R. This is a transient, the Q transient. If it is 0, it is load transient, if it is 1, it is reference transient. And, this is a port declaration. These are all scalar input, these are all scalar inputs ok.

Then, output clock ADC, clock DAC, PWM high, and pwn low; these are all scalar output. Then, Q load you can take it here, it is also scalar output. So, we should have made a merge with them, then input is signed data it is the ADC. And, what is the bit size? 9, 0. So, it is a 10-bit and it is in Q 1.9 format. The DAC data is 12-bit data, it is a 12-bit offset binary.

When is the offset binary and the DAC data, the analog voltage of the day, we told that it can vary between minus 1 volt to plus 1 volt. So, this is the range we have. So, before offset data I mean if you take 2's complement; that means, it will be Q 1.11. And, if you complement so, this is like a 2's complement, then you have to simply complement the MSB. Then, you have to convert it into offset binary. So, this is the data ok.

(Refer Slide Time: 05:51)



Mixed_Signal_CMC Module (contd…)

```
// declaration of  wire and register variables
reg signed [9:0] N_out;
wire signed [9:0] N_e;
wire signed [19:0] N_con_temp;
wire signed [18:0] N_con;
wire f_pwm,I_rst,Q_sat,Q_tran_load_ref;
parameter  K_p=10'sb0100_110000; //Q4.6 signed format
parameter  K_i=10'sb0_001111010; //Q1.9 signed format
```

Next, we have to declare. So, N out is a register where we are taking the data, the output to the; that means, this is the actual output. Because, ADC is sending data which is the ADC data, but that data is coming at the rate of 25 megahertz clock. But, we want to synchronize the output data with the switching frequency clock which is 200 kilohertz. So that means, we will take only the edge of the switching frequency clock.

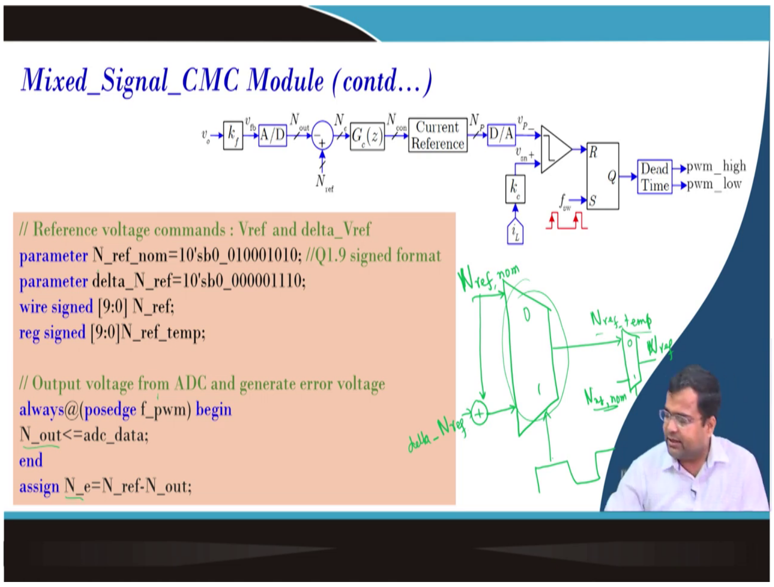
And, if you say suppose if this is you know if you draw the output voltage waveform. So, if this output voltage waveform looks like this. So, samples are captured let us say at a very fast rate ok, but these are available. And, we are talking about this clock maybe this is your switching frequency clock. So, we are talking about this point, but remember there is a delay; that means, better we should keep it here.

So, let us say we keep it somewhere here. So, this edge; means, there is a 6-cycle delay; 1 2 3 4 5 6. So that means, this data is what was sampled here; that means, I will say it is my output voltage sample 1. So, this data will be available here and we call it a V s 1 data will be available at this point. Because there is a delay, the sampling delay t s is nothing, but 240 nanosecond because we are using a 40 nanosecond this clock time period. So, this is 40 nanosecond and it has a 6-cycle latency.

So, we are taking the N out output digital number, but which is captured 240 nanosecond before ok. Then, N e is the output error that we are defining. This is a temporary control variable. So, because it is require resizing, there is a control variable. So, the wire these are the all wire definition I reset, Q Q saturation, and Q transient type that we want to select. This is the parameter proportional integral gain. So, proportional is in Q4.6 format, and integral is in 1.9 formats because proportional we want to vary largely.

So, what is the value that we have taken here? So, here it is like the first peak so; that means, 4; that means, your K p is what is K p? It will be 4 because the integer bit is 4 points the first bit is 0.5. So, it is 4.75. What is K i? You can accordingly set it because it is all a fractional number. So, it will be 0. what? That you can get it from here; that means, it will be 2 to the power minus 3 minus 3 plus 2 to the power minus 2 plus sorry minus 4, 2 to the power minus 5 plus 2 to the power minus 6 plus 2 to the power minus 8. So, if you sum up this will be K.

(Refer Slide Time: 09:18)


Mixed_Signal_CMC Module (contd…)

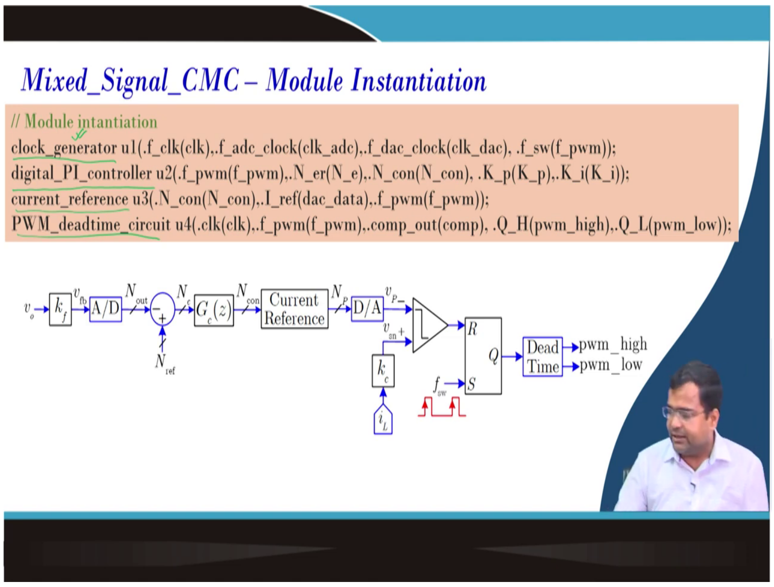
```
// Reference voltage commands : Vref and delta_Vref
parameter N_ref_nom=10'sb0_010001010; //Q1.9 signed format
parameter delta_N_ref=10'sb0_000001110;
wire signed [9:0] N_ref;
reg signed [9:0]N_ref_temp;

// Output voltage from ADC and generate error voltage
always@(posedge f_pwm) begin
N_out<=adc_data;
end
assign N_e=N_ref-N_out;
```

In the reference comment so, we have this reference right. So, N_reference; means, we are setting a sign format and this will you know N_ref. So, this is for voltage reference, not the current reference I am sorry, for the reference voltage. So, we are making a load transient a reference transient, I have discussed in the earlier digital voltage mode controller the same thing.

So, we are making muxing this. In one case, we are taking N_ref nominal, in the other case we are taking so, N_ref I would say N_ref you know sorry. This will be the sum of this plus delta N_ref some value. So, then we are getting N_ref_temp based on what you select because it depends on whether you want this or this is for the reference transient. And, this whether actual reference will be again there is a mux, whether it will be N_ref nominal.

So, this is your actual N_ref; that means, we may or may not want reference transient. So, if we do not want, you simply take the nominal value and if you want, then you can make this customized block by setting this clock can be varied. So, you can make a reference transient. So, all these are possible and that we have discussed. So, this output is captured at the edge of the PWM signal ok and now we are making the error voltage N_ref minus N out.

(Refer Slide Time: 11:20)



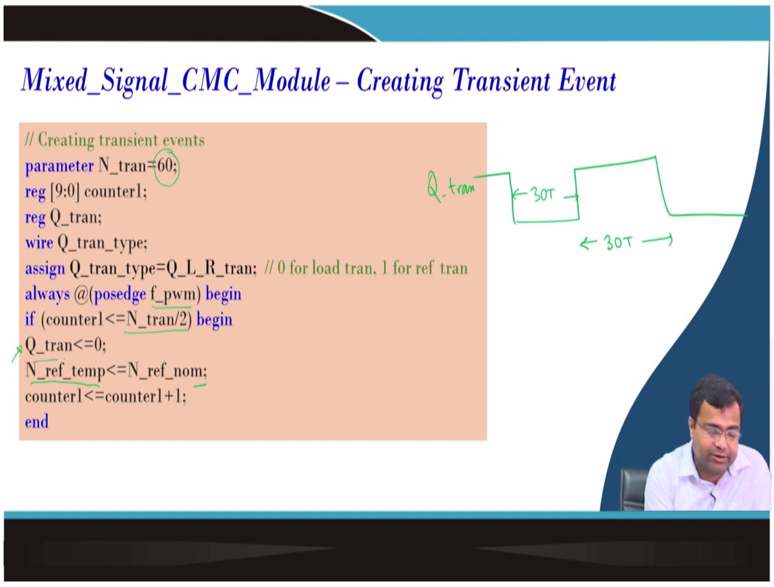
Mixed_Signal_CMC – Module Instantiation

```
// Module intantiation
clock_generator u1(.f_clk(clk),.f_adc_clock(clk_adc),.f_dac_clock(clk_dac), .f_sw(f_pwm));
digital_PI_controller u2(.f_pwm(f_pwm),.N_er(N_e),.N_con(N_con), .K_p(K_p),.K_i(K_i));
current_reference u3(.N_con(N_con),.I_ref(dac_data),.f_pwm(f_pwm));
PWM_deadtime_circuit u4(.clk(clk),.f_pwm(f_pwm),.comp_out(comp), .Q_H(pwm_high),.Q_L(pwm_low));
```

Then, we are instantiating one block is the clock generator. This we have discussed in the previous in k, in the context of digital you know voltage mode control, the same block. It is just doing the clock division, it is generating a switching frequency clock, ADC clock, and DAC clock.

ADC clock and DAC clock are using the same switching frequency, the same frequency of 25 megahertz; digital PI controller which I will be discussing in the next lecture, a little bit detail; current reference, and the PWM_deadtime. So, PWM_deadtime is an easy block. We have discussed deadtime.

(Refer Slide Time: 11:55)



Mixed_Signal_CMC_Module – Creating Transient Event

```
// Creating transient events
parameter N_tran=60;
reg [9:0] counter1;
reg Q_tran;
wire Q_tran_type;
assign Q_tran_type=Q_L_R_tran; // 0 for load tran, 1 for ref tran
always @(posedge f_pwm) begin
if (counter1<=N_tran/2) begin
Q_tran<=0;
N_ref_temp<=N_ref_nom;
counter1<=counter1+1;
end
```

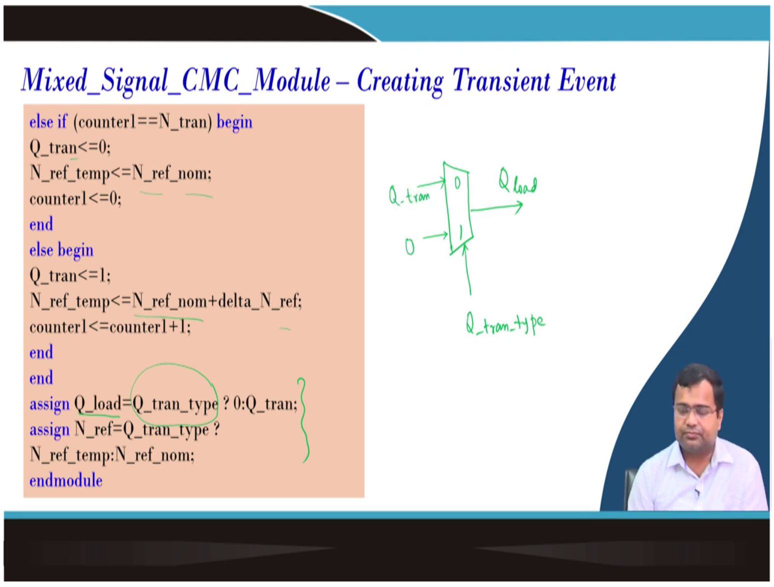
So, now the creating the transient event. So, in the main module, we are continuing. Suppose, we want after like a 30 switching cycle, we make a transient. So, this is your 30T and this is also your 30T. So, I make 30T high, and 30T low. So, this can be used for load transient, this can be used for reference transient.

Instead of 60, you can take 100 whatever you want this is possible. So, this you are making periodically; that means, it is taking at the switching frequency clock, at every edge of the switching frequency clock for the first 30 cycles, it will be either low you can see up to this point. Then, if crosses half value, it goes high. So, it is a transient ok. Next so, next is that inside that block we are making Q tran; that means, this Q tran 50 psi.

So, this is like you can say Q tran. Now, we can use this Q tran either for load transient or reference transient. This is just a flag signal. We are also making the N_ref_temp, we told that in the previous example that N_ref_temp can take either this nominal or ref. So, it is not the I will say muxing, but basically, it is a register which can either be loaded with this or this loaded with this the register ok.

In this case, we are using a register variable because if you use mux, then it will look like instantaneous blocks. But, it is not the instantaneous block, it is a register variable that is defined here. Next, it can take either the nominal value for 50 percent of the time when the Q tran is 0.

(Refer Slide Time: 13:50)



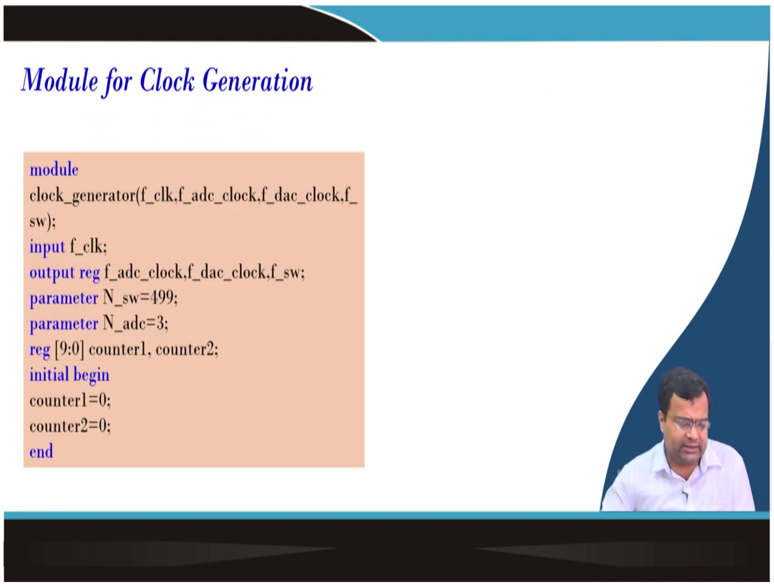
Mixed_Signal_CMC_Module – Creating Transient Event

```
else if (counter1==N_tran) begin
Q_tran<=0;
N_ref_temp<=N_ref_nom;
counter1<=0;
end
else begin
Q_tran<=1;
N_ref_temp<=N_ref_nom+delta_N_ref;
counter1<=counter1+1;
end
end
assign Q_load=Q_tran_type ? 0:Q_tran;
assign N_ref=Q_tran_type ?
N_ref_temp:N_ref_nom;
endmodule
```

Then, when it reaches the upper limit the Q tran is 0 nominal value. But, the rest of the time when the rest of the 50 percent Q tran is 1 and it takes nominal plus delta value ok. Now, for load transient, if the transient step is 0 so, it is a muxing; that means, we have discussed this transient event in the context of voltage mode.

So, we are not spending time. So, here we can say there is a mux ok so, this is your Q load. So, 0 and 1, and this is your Q tran type. So, if what it takes; that means, it can take either if it is 0, it takes Q tran or it takes 0 OKs. So, this we have discussed.

(Refer Slide Time: 14:46)

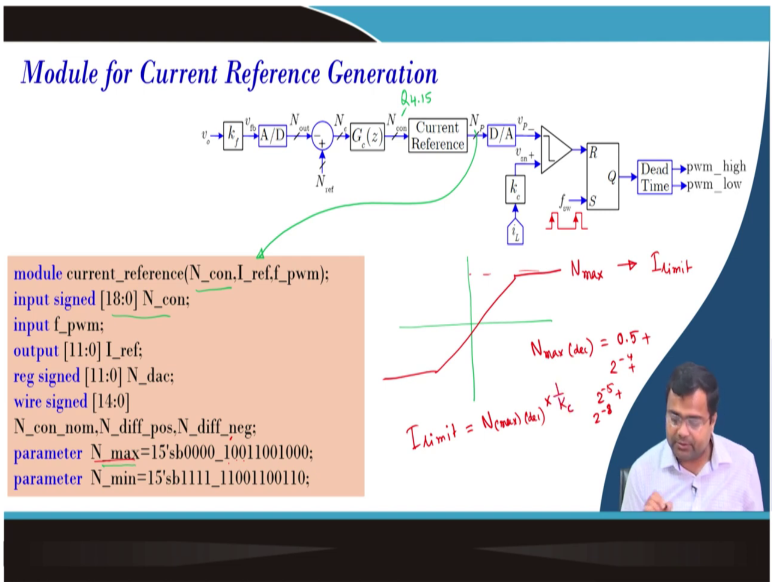


Module for Clock Generation

```
module
clock_generator(f_clk,f_adc_clock,f_dac_clock,f_
sw);
input f_clk;
output reg f_adc_clock,f_dac_clock,f_sw;
parameter N_sw=499;
parameter N_adc=3;
reg [9:0] counter1, counter2;
initial begin
counter1=0;
counter2=0;
end
```

Then, the clock generation I think these things we have discussed in the previous lecture. So, we are not spending time on clock generation. These are already discussed. The clock generation block and the digital PI controller will discuss in the next lecture.

(Refer Slide Time: 15:01)



Module for Current Reference Generation

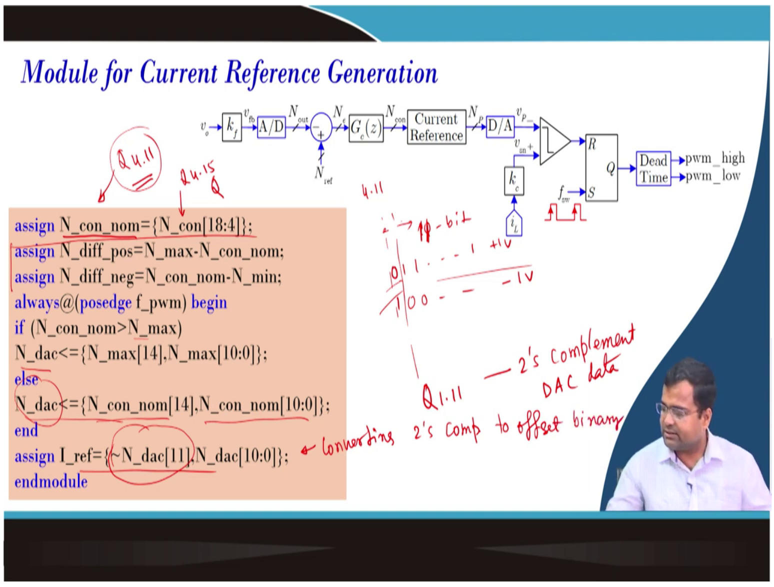
```
module current_reference(N_con,I_ref,f_pwm);
input signed [18:0] N_con;
input f_pwm;
output [11:0] I_ref;
reg signed [11:0] N_dac;
wire signed [14:0]
N_con_nom,N_diff_pos,N_diff_neg;
parameter  N_max=15'sb0000_10011001000;
parameter  N_min=15'sb1111_11001100110;
```

Then, the current reference is the important part. So, the current reference block module, accepts the control output which is N_con, there may have a so, this N_con has; what is the format? It is Q 4.15 format that so, it is a 19 bit, are a 19 bit and N p which is going to the that is in this case it is I_ref and it is there is a PWM clock. So, what does it do?

It first takes that you know the nominal value. So, we are setting up maximum; that means, here we are setting a current limit; that means, we are setting a limit. So, the current will go and there is a limit, limiter circuit. And, this N_max, this N_max corresponds to some I limit. And, what is the value here? You can see N_max in decimal, it will be 0.5 because of this digit plus 2 to the power minus 2 3 4 minus 4 plus 2 to the power minus 5 plus 2 to the power minus 7 8.

So, 2 to the power minus 1 minus 2 minus 3 minus 4 minus 5 6 7 8 yeah. So, this value and whatever you get when you multiply this number. So, what is your I limit? Sorry, what will be your I limit? Because there is a scaling factor, the I limit is equal to whatever N_max you get in decimal, you multiply with 1 by K c.

(Refer Slide Time: 17:07)



Module for Current Reference Generation

```
assign N_con_nom={N_con[18:4]};
assign N_diff_pos=N_max-N_con_nom;
assign N_diff_neg=N_con_nom-N_min;
always@(posedge f_pwm) begin
if (N_con_nom>N_max)
N_dac<={N_max[14],N_max[10:0]};
else
N_dac<={N_con_nom[14],N_con_nom[10:0]};
end
assign I_ref={~N_dac[11],N_dac[10:0]};
endmodule
```

So, now the module's current difference generation so, we have this controller normalization; that means, if we go back we have this N_con which is 19-bit ok. And, we are discarding 4 bit; that means, we are taking earlier it was Q; that means, this is how much? Q 4.15. And, this will be what? Q 4 is there and we are discarding 4 bit; that means, 11. So, it is a 15 and then we are you know we are simply because this we are not using this. If N_con nominal is greater than N_max, then we will take N_dac data to be the maximum value. So, it is a current limit.

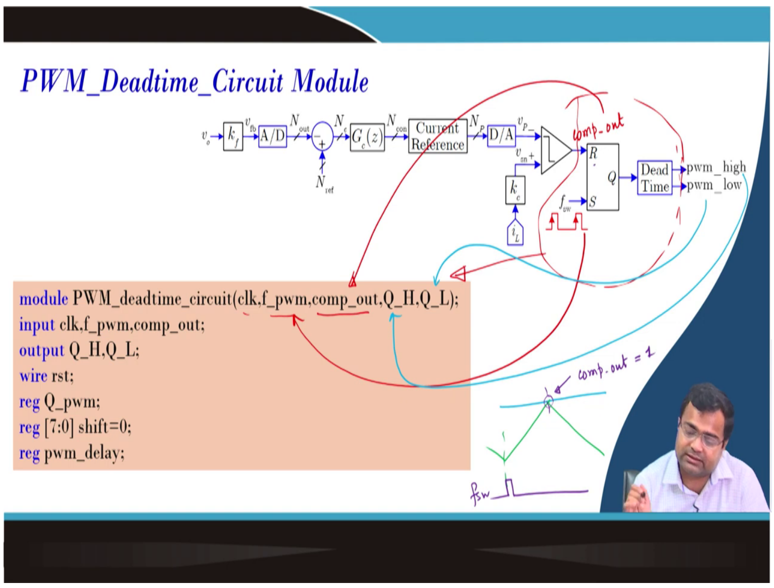
If else it will take that nominal value and it is a sign bit. So, where are you doing here? So, because this total is 15 bits and for 2's complement, this DAC can take in 2's complement from 0 all 1 to maximum is 1 all 0, sorry 1 all 0. So, this is the minus 1 volt, this is plus 1 volt. So, it is you know in the Q format, this is the fractional number.

So, after these all controllers, we initially thought of 4.11, but we are simply limiting to 1 bit only; that means, even the controller value may be large. We are limiting because it is defined by the maximum and then we are resizing the data to fit into the DAC. So, DAC will accept only 12-bit data. So, we are discarding those MSB, because we are limiting the value only to 1 volt. So, that is why it does not require more than 1 integer bit. So, we are discarding the other 3 bits.

And we just check whether it is higher than your maximum current difference or not. If it is higher, it will set the maximum value otherwise if it is lower, it will take because the then other 3 bits anyway will be 0. So, you can take the controller bit, and the sign bit, and then the other 3 bits are discarded. The rest of the bit will be left side; that means, there are 10 bits on this side so, 11 bits sorry.

Because it is a 12-bit DAC, we are talking about 12-bit DAC so, the DAC in Q s format should be 1.11 in 2's complement that DAC data. But, we have to send the DAC data to offset binary, that is why we are this data whatever we are getting, we have to complement this MSB, that is this process is converting 2's complement to offset binary ok, that is it.

(Refer Slide Time: 20:23)



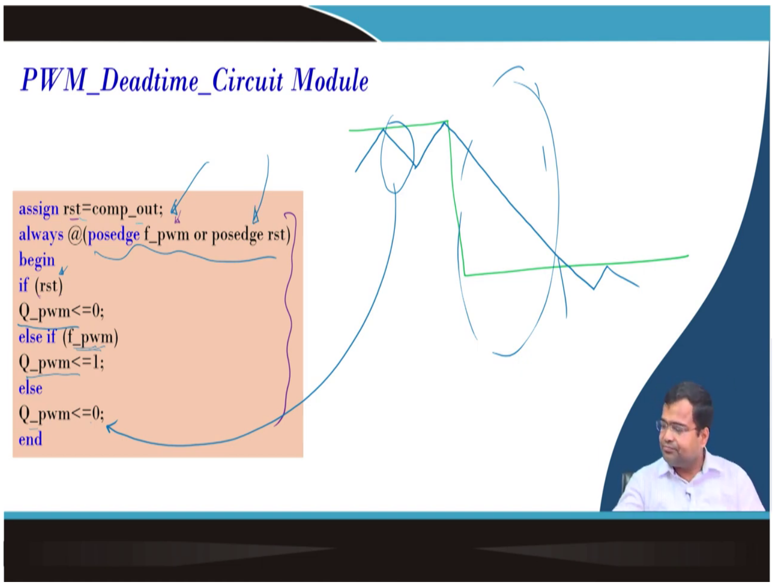
PWM_Deadtime_Circuit Module

```
module PWM_deadtime_circuit(clk,f_pwm,comp_out,Q_H,Q_L);
input clk,f_pwm,comp_out;
output Q_H,Q_L;
wire rst;
reg Q_pwm;
reg [7:0] shift=0;
reg pwm_delay;
```

So, then PWM_deadtime we have discussed, the clock f_pwm with the switching frequency clock, comparator output that is important because the, if you look at this, this is your comp output right, comp output. I mean virtually it is the internal variable, the actual interface is a comp only, and there it is generating Q_L. And so, we want to generate this. So, these blocks deal with only this signal.

These are the signal and we generate these signals. So, this is the block we are talking about. So, it takes so, this signal is nothing, but this signal, then comparator output will be this only and the other two signal that we are talking about is one is a high signal which is here, another is the low signal which is here ok. So, it is just you know again we need a shifting block.

(Refer Slide Time: 21:19)



PWM_Deadtime_Circuit Module

```
assign rst=comp_out;
always @(posedge f_pwm or posedge rst)
begin
if (rst)
Q_pwm<=0;
else if (f_pwm)
Q_pwm<=1;
else
Q_pwm<=0;
end
```

So, first, we are not using so, reset comparator output. How the comparator output is set? You see v p is the current difference, and now V sense is the voltage. So, V sense is the voltage, you can see. So, at the edge of the switching clock, this is your switching clock fsw. So, the switch will be turned on and when this condition is met, then this results in comp output comp out at this point it will be at this instant, particularly it will be what? Because v sn is tried to exceed v p, it will be 1.

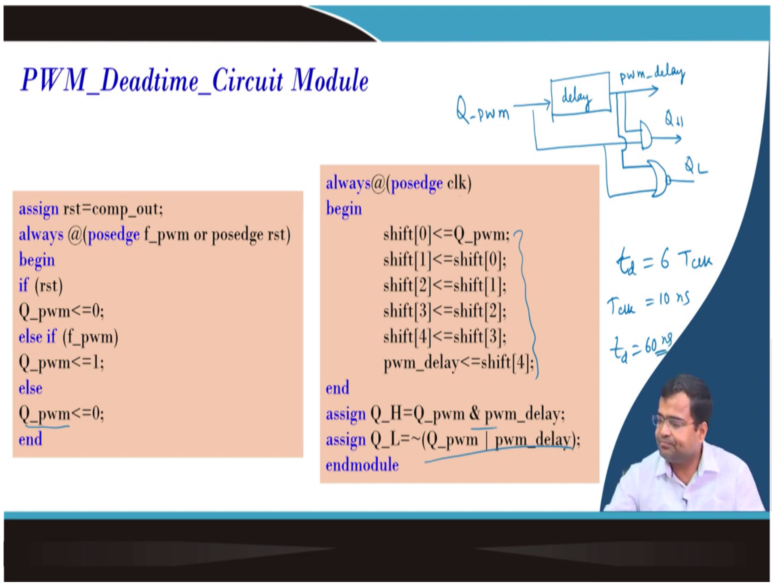
So that means, this comp output only will become 1, when the inductor current tries to cross the peak current and that is used to reset. So, we are setting this r; that means, we are using the reset command as a comp output. When the comp output will be high, it will reset. Now, this is the latch operation, the trailing edge modulation. How does it do? So, it will take the pauses of the switching clock or the pauses of the reset clock whichever comes.

So, whenever the switching pauses come, it will check whether is reset is high or not; that means, whether the comparator output is high or not. It may be the case what happens if the inductor current is already higher. Suppose, I am talking about so, the scenario; that means, you know we have this reference current; here due to load step-down transient, it goes down. So, what will happen is the switch will be high like this, and suddenly it shows that it is going low.

So, it will go down and down, then goes high. So, during this cycle this particular duration, your comparator output would be high. As a result, Q would be always 0. So, the switch is continuously off. But, if this signal is not high then at the edge of f_pwm if it is high; that means, when it goes high, the Q s pwm will be 1. And, if the reset is high then it will be 0, otherwise, Q_pwm is 0 because in other conditions than that it should be 0.

As if it has crossed the limit and you have to turn it off. So, this condition during this condition is denoted by this particular case. When there is no comp high edge, there is no PWM high edge. In that case, it should be 0 because this is an edge trigger circuit. This circuit will only be enabled when, the pauses of this fw come or the pauses of the reset come, otherwise this block will not be executed ok.

(Refer Slide Time: 24:09)



PWM_Deadtime_Circuit Module

```
assign rst=comp_out;
always @(posedge f_pwm or posedge rst)
begin
if (rst)
Q_pwm<=0;
else if (f_pwm)
Q_pwm<=1;
else
Q_pwm<=0;
end
```

```
always@(posedge clk)
begin
        shift[0]<=Q_pwm;
        shift[1]<=shift[0];
        shift[2]<=shift[1];
        shift[3]<=shift[2];
        shift[4]<=shift[3];
        pwm_delay<=shift[4];
end
assign Q_H=Q_pwm & pwm_delay;
assign Q_L=~(Q_pwm | pwm_delay);
endmodule
```

$t_d = 6\ T_{clk}$

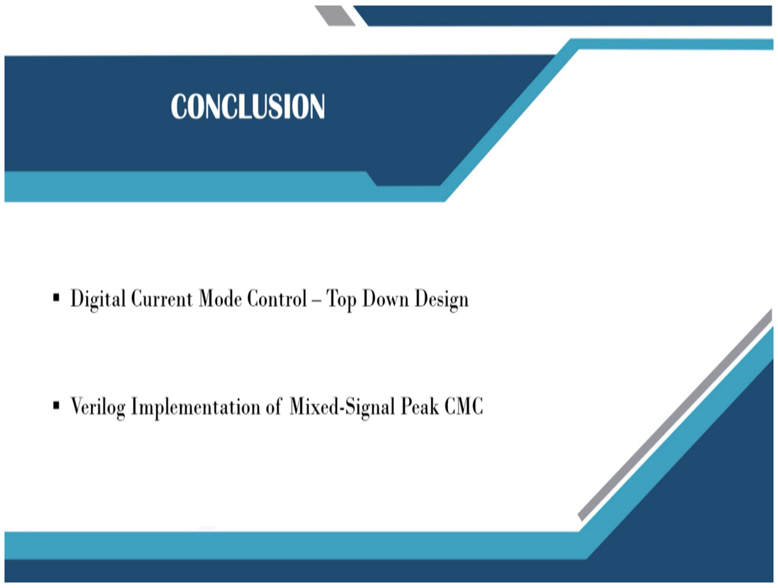
$T_{clk} = 10\ ns$

$t_d = 60\ ns$

And, this block is used to generate the deadtime; that means, the Q_pwm which is coming out this is the; that means, Q_pwm it is coming out, we are delaying this signal. There is a delay and we have discussed in lecture numbers 69 and 60, 70 that this is our delay. And using these two signals; means, if we take their AND operation, then you will get Q H and this is what exactly.

And, if you take their NOR operation, then it will be Q L and this is here. And, the delay we are using from Q 0 to Q 5, eventually, because there will be one additional delay of the clock. So, the delay will be tau d, which turns out to be 6 point T clock, where the T clock time period is 10 nanoseconds. So, you will get t d of 60 nanoseconds ok.

(Refer Slide Time: 25:24)



CONCLUSION

- Digital Current Mode Control – Top Down Design
- Verilog Implementation of Mixed-Signal Peak CMC

So, in summary, we have discussed digital current mode control – top-down design. We have discussed Verilog implementation of mixed-signal current mode control. In the next lecture, we are going to talk about digital current mode control implementation using Verilog, that is it for today.

Thank you very much.