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 - 72
Top Down Design Methodology in Digital Voltage Mode Control - II

Welcome. In this lecture, we are going to talk about Top Down Design Methodology in Digital Voltage Mode Control. This is the continuation of the previous lecture.

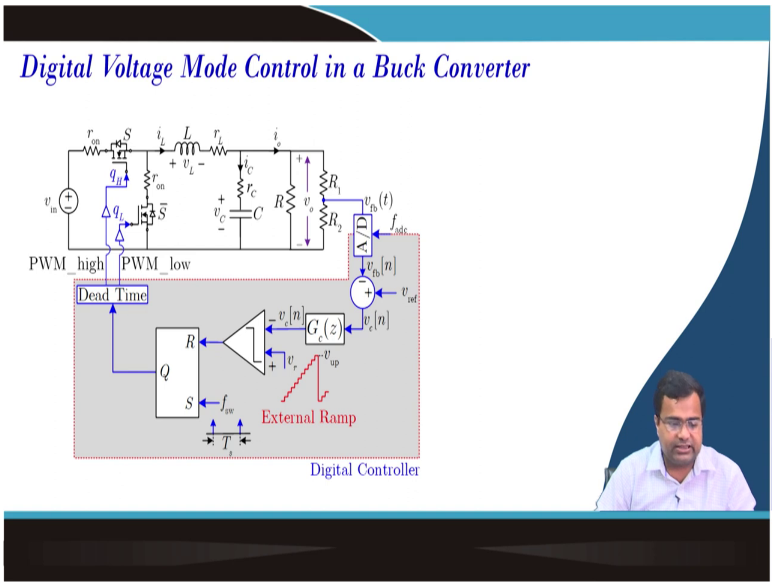
(Refer Slide Time: 00:33)

Concepts Covered

- Top Down Design Method for Digital Voltage Mode Control
- Verilog HDL Coding for Overall Digital Voltage Mode Control

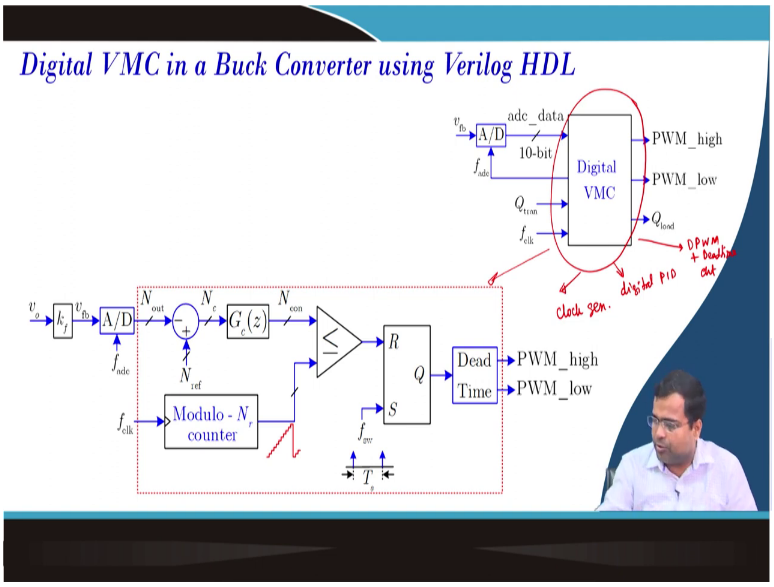
In this lecture, we are going to talk about Top Down Design method and then Verilog HDL coding for the overall Digital Voltage Mode Control.

(Refer Slide Time: 00:42)



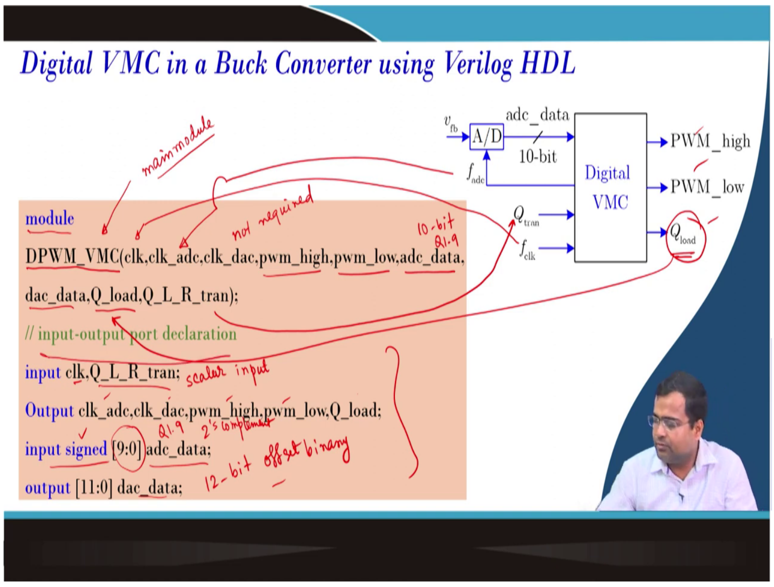
So, here, again we will recall our digital voltage mode control implementation of the architecture that we have discussed.

(Refer Slide Time: 00:47)



Then, we are about to design the whole digital control module that we have discussed. And we have discussed that this is a main module and it consists of 3 submodules. One is the clock generation, the clock generation, the other is the digital PID controller and the third one will be DPWM plus your dead time circuit, ok. So, now, let us start. So, this is the overall block that will look like this, the internal architecture of this block.

(Refer Slide Time: 01:29)



Digital VMC in a Buck Converter using Verilog HDL

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

So, first, we will start with the main module. So, again we start with the Verilog module. And now the code will continue. So, first, this is the name of the module which is the main module.

So, I will say this is the main module. And when we go in the next week or the subsequent lecture, that is the step-by-step methodology of implementation and hardware demonstration, we will talk about this require a user constraint file, and how to do it because we want to show using our actual Xilinx code we want to show a live demo regarding that Verilog implementation in the subsequent lecture.

But this is the main module. Today in this particular week we are talking about how can we use write Verilog code for this implementation. But while we will be demonstrating the live demo of digital voltage mode control, then we will say what other files are needed. So, there is a clock which we represent, so this clock is nothing, but this clock, ok. Then, we also need something like an ADC clock, this is here. We are giving a DAC clock because in our setup we have DAC.

So, DAC clock you may or may not provide or you may not use DAC data, because for voltage mode control we do not need D to A converter. But when we go to the current mode we will use that. So, I am just keeping you know this particular line in mind. So, this is not used. I would say this is not used. So, I can drop this. We are not using this, not use here. We have just used it because it is where it exists, but we are not utilizing it. I will say not required.

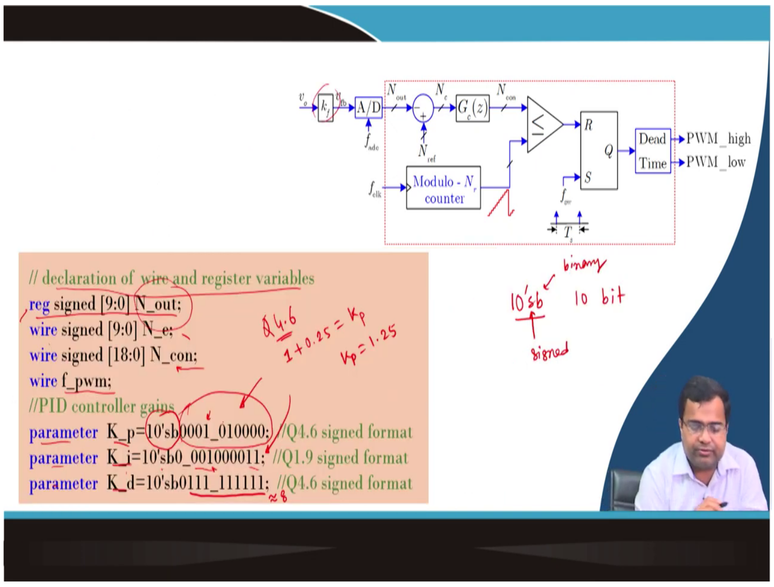
PWM high is the high side gate signal going to the driver. PWM low is the low side gate signal going to the driver. ADC data is the data coming from ADC and this is a 10 bi10-bit Q 1 dot 9 format. We have discussed this. DAC data we are sending 0, just so you know we may or may not say because we do not need DAC we will not use it in this voltage mode control.

Q load is going out. This is the Q load, the Q load which is here, ok. And Q L R tran which is here. That means, it is input on what kind of transient you want, do you want a reference transient or load transient, we will discuss this. Now, we have to declare in the Verilog module which is the input, which is the output. Even among input, so the clock is the input, Q L R is the input, but these are the scalar input, these are the scalar input, sorry, scalar input. These are the scalar input. There is vector input also, a 10-bit vector and it is a sign. So, we have to define the ADC data.

The output we have is clock ADC, clock DAC, PWM high, PWM low, and Q load. There are so many out output, output, output. We are sending an ADC DAC clock that may or may not be needed. But the output of the ADC that we are giving out, sending out data for the DAC here we are not using it, but we can give send some random data unless you know as long as it is not if the part of the loop does not matter. But it is a vector, 12-bit data.

And remember, it is a 12-bit offset binary that we will be using, offset binary. And this is 2's complement, ok. And this also has like a plus minus 1 volt, but in offset binary and this is also Q 1 dot 9 format, ok.

(Refer Slide Time: 05:44)



```
// declaration of wire and register variables
reg signed [9:0] N_out;
wire signed [9:0] N_e;
wire signed [18:0] N_con;
wire f_pwm;
//PID controller gains
parameter  K_p=10'sb0001_010000; //Q4.6 signed format
parameter  K_i=10'sb0_001000011; //Q1.9 signed format
parameter  K_d=10'sb0111_111111; //Q4.6 signed format
```

Next, this is the continuation, the code is continuing. So, we have declared the ports, all the ports. That means this part is called a port declaration. Now, we are going to the declaration of wire and register because internally we are using some register or some wire, and we have already discussed this. So, some of the register N out, because N out is a register, because this we are using for capturing the output data from the available ADC input. Because we are taking data from ADC and I told you that ADC data is coming at the rate of 25 megahertz rate; that means.

But we need to get the output sample where we want it. Though it is a sample somewhat earlier, we want to get the data synced, we want to load our register at some instant in time. Then we need to define the error voltage which is why N e wire and N con which is the controller output a 119-bit sign number. Then, we are also defining f PWM, that is, this is denoted as a local variable for frequency, switching frequency. It is not going out or coming in, but this is used internally using as a switching frequency.

So, all the gate signals will be generated in synchronization with this PWM as well as the controller computation will happen at this, with this clock, synchronize with the clock. We are using 3 K p, K i, K d. So, K p like a proportional control, K i and K d are integral and derivative controller gain in discrete domain remember. And this we are defining as a parameter which is a signed binary, 10-bit sign binary. So, when we say 10-bit signed binary. So, it is a 10-bit, s indicates a signed number, and b indicates binary, ok.

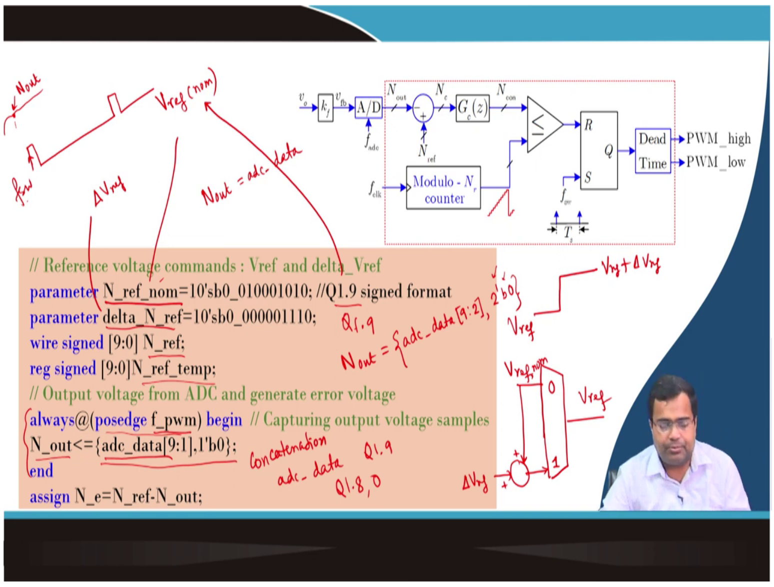
And we are giving some value, what is the format you define? Q 4 dot 6. Then, what is this value? This value will correspond to the 2 to the power 0 bit. So, it will be 1 plus the first digit is 0.5, this is a 0.25, so 0.25. So, you are using K p is equal to, and K p is equal to 1.25. This is in the actual number. And we will discuss how to select K p, then we have a step-down factor.

So, we will take the design case study in the subsequent week. But here whatever value will come from the design we can simply plug in. Integral gain, again we are taking integral gain and this is in the Q 1 dot 9 we have discussed. That means Q 1 dot 9 means this will be 0.125. So, correspondingly you can get the value, you can convert this data.

The derivative gain, we are taking, is the first of these 3 cases the first MSB is 0 because it is a signed bit. We are talking about the positive coefficient of the controller. So, all the bits will be 0, all cases the MSB will be 0. Then, the other bit if you see it is approximately equal to 8. So, using K d to be 8, but remember these are the control value where there are other scaling factors. So, it is not actual, if whenever we did the MATLAB simulation, we have not considered the voltage feedback gain, current feedback gain, and other factors. So, we have to consider it.

Now, we will bring back our MATLAB model again and we will put those real scaling factors and want to simulate, and in the subsequent lecture, we will see how the simulation result, whether it is different from the experimental result or they are close. So, you want to discuss. But these are the values.

(Refer Slide Time: 09:38)



```
// 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  // Capturing output voltage samples
N_out<={adc_data[9:1],1'b0};
end
assign N_e=N_ref-N_out;
```

Then once we have to also differentiate this is the parameter, N ref nominal, this is the reference voltage, one is the nominal value, another is the delta. So, it is you can say V ref nominal corresponding voltage will come from this Q format. It is Q 1 dot 9 format this is also Q 1 dot 9 format. So, you can get the analogous reference voltage and nominal.

Similarly, this will give you delta V ref. Again in Q format. So, why do we need it? Because you remember that in Simulink we use a step command, right? So, here it is like a V ref to V ref plus delta V ref. When do you want it? If you want to make a reference transient. So, this is possible that we can make a reference transient and we want to check the response, ok.

So, the actual reference; means, we are using a mux here, mux here, where this will be our V ref and this will be V ref plus delta V ref. The summing block is 0 or 1, whatever, and this is, so this is like a nominal value. So, this is the actual V ref. So, that means, the reference which is coming out, is a mix of nominal value, nominal plus delta V ref. So, we can make a load reference transient, that is why we have to use a wire variable and so N ref temper, you know temp what is the temporary value. So, I will go with these for internal definition.

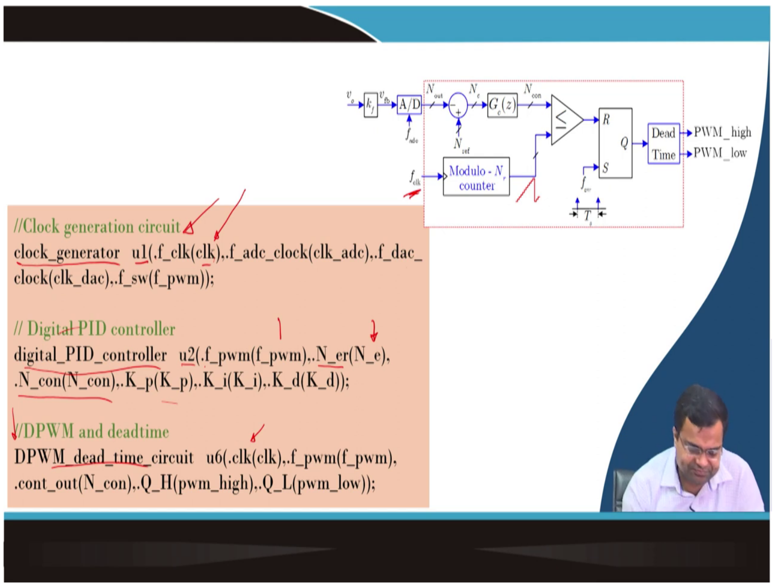
Now, at always at the switching frequency clock; that means, whenever the switching frequency clock will come, this is my fsw clock. In this edge, we are capturing the data coming out from the ADC, and this data we are talking about N out. That means N out is captured from ADC data. That means, N out is the data coming from ADC data at some instant of time that is defined by the activity at the edge of the fsw clock; that means, it will only take, that means, a clock-synchronized operation. So, it will take the ADC data, only at the edge rising edge of the switching frequency clock.

Now, we made some concatenation. So, we made some concatenation; you know we have con where so ADC data we know that. What is the ADC data? It is in Q 1 dot 9 formats. That means it has a total of 10 bit. The first bit is a sign bit and all other bits. But we are capturing Q 1 dot 8 data first and padding with a 0. So, we are intentionally reducing the LSB and padding with 0, so that we want to decrease the resolution. Why it is needed?

Because in digital voltage mode control, we need to make sure the resolution of the DPWM, should be finer than the resolution of the ADC and that is why we are discarding bit, it is 10-bit data, ok. So, that is why we are padding. Even if we want to lose two bits, then what we will do? We will do N out; two bits if you want to lose the concatenation will do ADC data, we are taking from 9 if the MSB 2 2 and padding with 2 binary digits 0; that means, 2 bits are 0. So, this is the 2-bit number binary number 0.

So, padding with 0, means, we are losing 2 bits. So, these things are possible. And then once this is done, so this will be a register because it has to store the value and will only take at the edge of this clock signal, so that is why it is defined as a register variable here. In the previous example, we mentioned. So, this is the register variable, ok. And error voltage is a reference minus N out. How are you getting references? I am going to that point.

(Refer Slide Time: 14:12)



```
//Clock generation circuit
clock_generator  u1(.f_clk(clk),.f_adc_clock(clk_adc),.f_dac_
clock(clk_dac),.f_sw(f_pwm));

// Digital PID controller
digital_PID_controller  u2(.f_pwm(f_pwm),.N_er(N_e),
.N_con(N_con),.K_p(K_p),.K_i(K_i),.K_d(K_d));

//DPWM and deadtime
DPWM_dead_time_circuit  u6(.clk(clk),.f_pwm(f_pwm),
.cont_out(N_con),.Q_H(pwm_high),.Q_L(pwm_low));
```

So, we are now this code is continuing. You know we may provide this full code in a one Word file or maybe a PDF file, you know because this is just breaking the code into because in one slide it is difficult. So, in the clock generation circuit, clock generation, we have to give the instant name, and the clock generates is a module name. And we also know that when you instantiate a module, inside a module, we can instantiate the module instantiation by name.

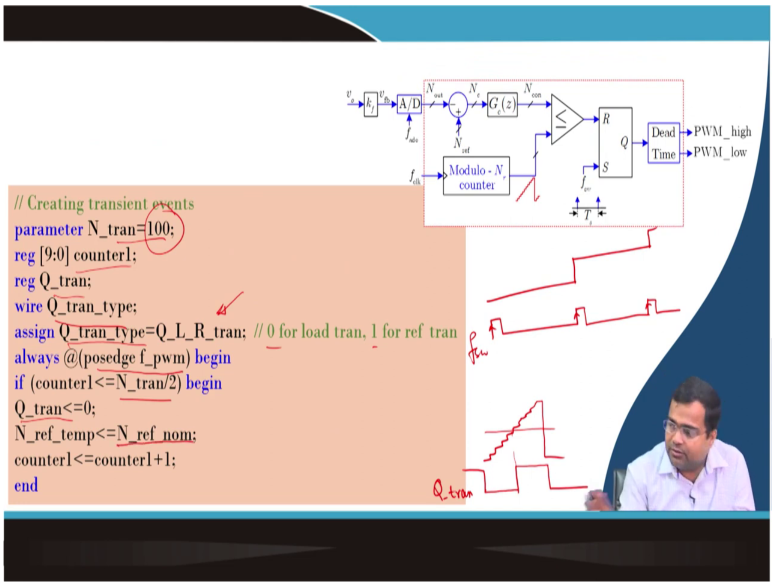
So, that is why the dot is coming where the actual clock generator module will have the variable input-output port name f underscore clock, f adc clock, and so on. We will go to that. And these are the local variable because, in the main module, we are taking the switching frequency clock sorry, the digital controller clock which is this clock that clk. So, it is connecting. That means, the clock generator takes this clock as the input and it is generating the output ADC clock, DAC clock, and the switching frequency clock and this is the local value.

Then, DPWM, the module name is this, and the instant name is this. And then again the dot means we are instantiating by name and these names are consistent with the module input-output variable name and these are the local variable, ok. So, for the digital PID controller N er is the name of the input signal vector which is the error you are connecting with the error of this main module's local variable. Similarly, the controller output K p, K i, K d. So, this naming part we have already discussed in the I think previous week, where we have discussed in detail the instantiation tile and all.

Then, we are also instantiating another module called DPWM underscore dead time because we discussed that we need to consider a top-down design methodology where the main module will be broken into 3 pieces. It will call, we will create one submodule of the clock generator, one submodule of PWM digital PID controller another submodule of DPWM plus dead time. So, this is the circuit.

And it again takes one of the inputs as a high-frequency clock because this clock is needed to adjust the dead time. After all, we can increase our resolution of the dead time defined by the high-frequency clock, that is our 100 megahertz clock. So, we can change the number of clock units concerning this clock. That means this clock period is 10 nanosecond. So, we can change the dead time in the order of 10 nanosecond. That means, 10, 20, and 30, but we cannot change it to 15 nanosecond, because the 5 resolution clock is not available, ok.

(Refer Slide Time: 17:12)



```
// Creating transient events
parameter N_tran=100;
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 creating an event, in the main module we want to create a transient event. So, we are giving 100 cycles. It can be 500, it can be 1000, and it is user-defined. How does it operate? So, first, we have to define a counter, the counter is getting updated. Then, the transient register, ok, and the transient type, where, again it is user defined whether you want to define it to be load transient or reference transient.

And we can have given transient type from the external. So, when I will go to the live demo, I will show you a switch that can be placed in either the 0 or 1 position. If you set it to 0, it will make the load transient. If you set it to 1, it will make reference transient, ok. So, our transient type can be set from the outside or you can define it inside. The only issue is the inside because it takes time to you know to compile the code and again load it to the target device.

To avoid such kind of multiple-time loading and you know kind of compilation we can simply provide an interface switch so that you can change it. And it will happen in real-time. Then, we are using a PWM clock edge. That means, we are using the switching frequency clock edge sorry, we are using this switching frequency clock edge which is like this. So, this is my fsw clock. At the edge, we are incrementing like a counter is incrementing like this.

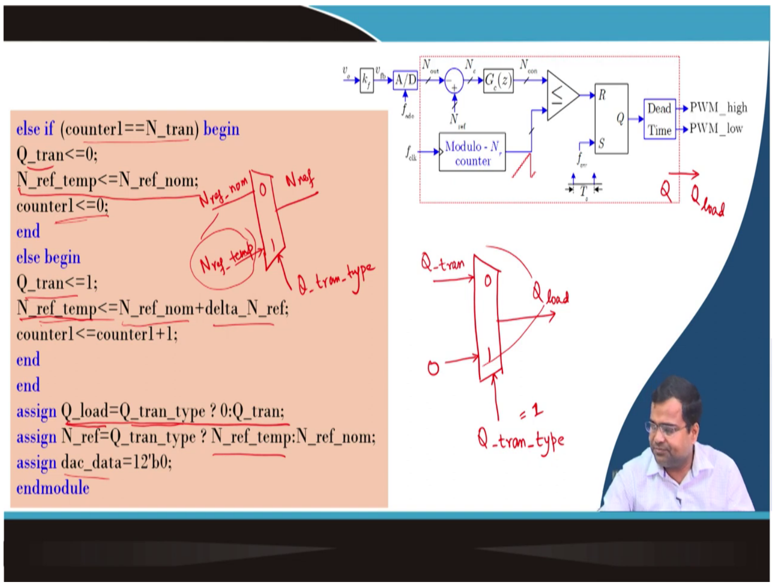
When a counter hit the half value of this then the transient up to that point will be 0. That means, as long as, that means, we are making like this kind of staircase, like up to full value.

When it makes hits the transient type will be 0, then it will be 1, 0, 1 like that. And this is my Q tran and we are using it for load transient.

That means I will show you if it can be a load or referenced. So, we will make 50 cycles in one condition, and another 50 cycles in another condition. If it is a load transient 50 cycles of one load resistance, then another 50 cycles of another load resistance, the combination.

If you make a reference transient, 50 cycle V ref nominal, another 50 cycle V ref nominal plus delta V ref, ok. So, this is fine, so Q 0 or Q 1. If it is this, else what will happen? Else, you know it is continuing; up to this point when equal, all will be equal. We will not update the counter, the counter will be reset.

(Refer Slide Time: 19:54)



```
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;
assign dac_data=12'b0;
endmodule
```

Else, we will do this to be 1 and then this N, so we can make also N ref; that means we are just updating. Now, the load transient command which is going out there is a Q load, Q load. Now, this Q load we are generating by muxing. This is my Q load, 0 and 1. If it is 0, then it will take Q tran. What is Q tran? It is a register. So, Q tran means for 50 cycles it will be 0, 50 cycles it will be 1.

So, that means, Q load will make a load transient. And there is a select line, we will come to that. The rest will be 0. That means if you set the select line which is the Q tran type. If this is set to 0, then it will make a load transient because the Q will depend on Q tran and which will change in half of this 50 cycle sorry; we have defined the total cycle to be 100. So, the first

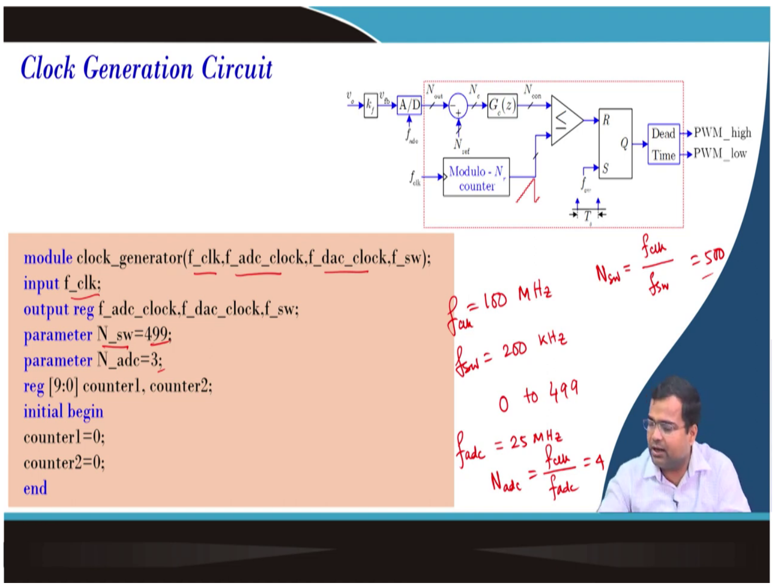
50 cycles will have a Q load will be 0, and the remaining 50 cycle Q load will be 1 because it is connected to the Q tran.

But if this transient type you know if this transient type is set to 1, then there will be no load transient, it will be connected to this point. But in that case, we will be, that means, this is the command of the data flow modeling of this mux. Similarly, here we are making N ref. So, N ref again we are making like this 0 and 1. So, 0 means it is your N ref nominal, and 1 means we are making are getting this value plus there is a delta N ref.

And this is your N ref. And again the select line is Q tran type. So, if the Q tran type is 0, the reference voltage will be fixed. If the Q tran type is 1, then sorry there is another layer. So, here is another layer which is Q tran temp, which means, it will be N ref temp. What is this? N ref temp you can see, for the 50 cycles it will be nominal value, another 50 cycles you see, for the first 50 cycles it will be if you go back, the first 50 cycles it will take the nominal value, remaining 50 cycles it will take nominal plus delta N ref.

So, this is the changing variable concerning 50 cycles, 50 cycles, but this is a fixed value. So, if the Q tran type is set to 1, when the load transient will be 0, there is no load transient, but there will be a reference transient. So, this arrangement is made to make sure that switch is placed to see either load transient or reference transient. So, you can change it, it is a user-defined thing. So, you can change it. Then, assign the DAC type. We are just sending 0 data. I mean this is what we are not using in DAC.

(Refer Slide Time: 23:44)



Clock Generation Circuit

```
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
```

$N_{sw} = \frac{f_{clk}}{f_{sw}} = 500$

$f_{clk} = 100$ MHz

$f_{sw} = 200$ kHz

0 to 499

$f_{adc} = 25$ MHz

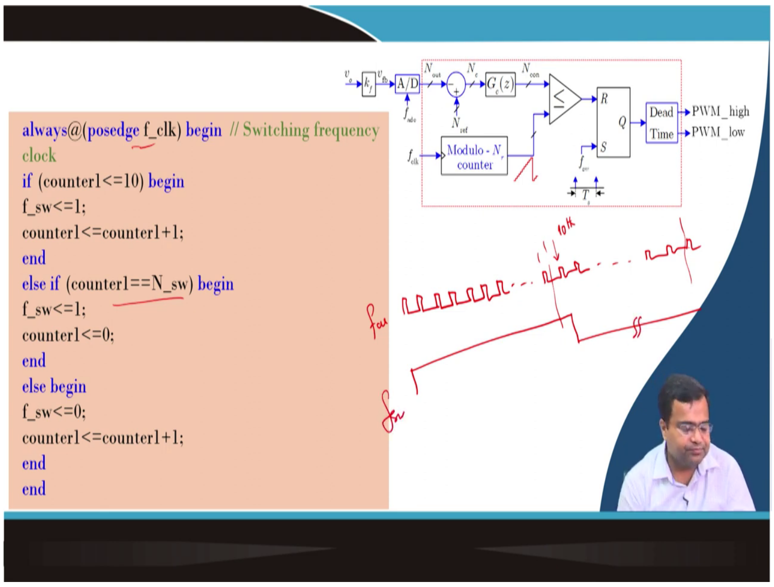
$N_{adc} = \frac{f_{clk}}{f_{adc}} = 4$

So, clock generator circuit. Now, we have instantiated this clock generator circuit. And this lecture we are only talking about clock generators because we have already discussed the DPWM dead time in the previous week, ok. So, we will be discussing digital PID controllers in the subsequent lecture. So, here we will only talk about the clock generator part.

So, what is a clock generator? If you go inside it takes input as a main clock which is a 100 megahertz clock and output as an ADC clock and DAC clock and switching frequency. It is just a counter. So, if you set this number to 500, 499; that means, we are 100 megahertz clock, it is my f clock, which is 100 megahertz. And I want to switch the frequency clock to 200 kilohertz. Then, what will be my N sw which is nothing but f clock divided by you know fsw? So, it will be 500. Did I set it to 499, why? Because it will count from 0 to 499, so, as a result, the total number will be 500.

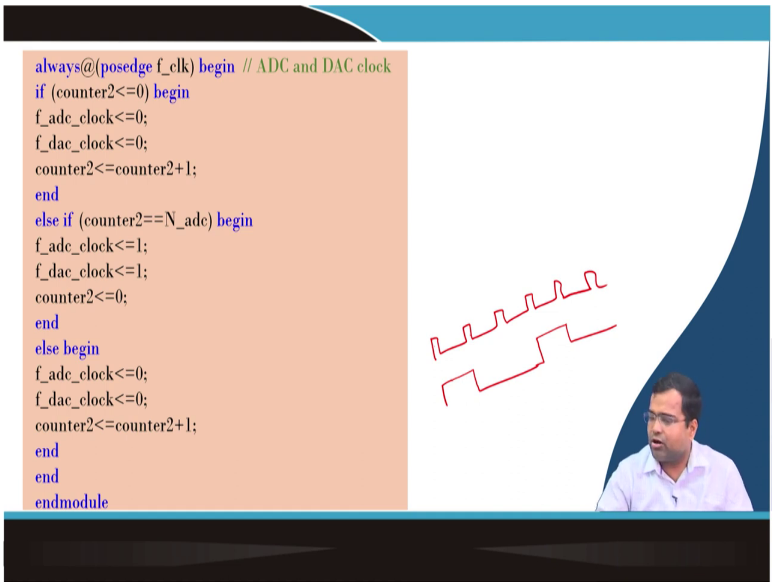
Similarly, I want a dc clock to be let us say 25 megahertz. Then, what will be N adc? It will be f clock divided by f adc, and that is 4. And I said 3 because it will vary from 0 to 3. That is it.

(Refer Slide Time: 25:19)


```
always@(posedge f_clk) begin  // Switching frequency
clock
if (counter1<=10) begin
f_sw<=1;
counter1<=counter1+1;
end
else if (counter1==N_sw) begin
f_sw<=1;
counter1<=0;
end
else begin
f_sw<=0;
counter1<=counter1+1;
end
end
```

So, you need two counter and you take the first clock, set it, I mean just for 10 cycles, then if it is equal to switching clock. That means we are making suppose, if this is your high-frequency clock let us say you know, so dot, dot, dot. So, if this number is 10, that means, if this is your 10th number, this is your 10th cycle f clock. So, my switching clock will remain high up to this, then it goes low, and if it is repeated. So, here we are making like this. So, this is your switching frequency clock.

(Refer Slide Time: 26:02)



```
always@(posedge f_clk) begin // ADC and DAC clock
if (counter2<=0) begin
f_adc_clock<=0;
f_dac_clock<=0;
counter2<=counter2+1;
end
else if (counter2==N_adc) begin
f_adc_clock<=1;
f_dac_clock<=1;
counter2<=0;
end
else begin
f_adc_clock<=0;
f_dac_clock<=0;
counter2<=counter2+1;
end
end
endmodule
```

Now, we can generate the ADC clock. So, it will count from N adc, up to that. So, it will first clock this, then next clock this, like this. This is an ADC, whereas, this clock will be again this clock like this, ok. Maybe my drawing is not perfect. So, you can try drawing. So, maybe we can erase this whole part. So, first, it will be high for one complete pulse, then 1, 2, 3, 4, again at 4th time it will be 1, and so on. So, it will be divided by 4 clocks. So, it is a clock division.

(Refer Slide Time: 26:51)

Summary

- Top Down Design Method for Digital Voltage Mode Control
- Verilog HDL Coding for Overall Digital Voltage Mode Control

IIT Kharagpur
NPTEL

So, in summary, we have discussed the top-down design method for digital voltage mode control. And we have also discussed Verilog HDL coding for overall digital voltage mode control. And in the next lecture, we are going to talk about Digital Voltage Mode Control you know Verilog HDL Implementation. That is it for today.

Thank you very much.