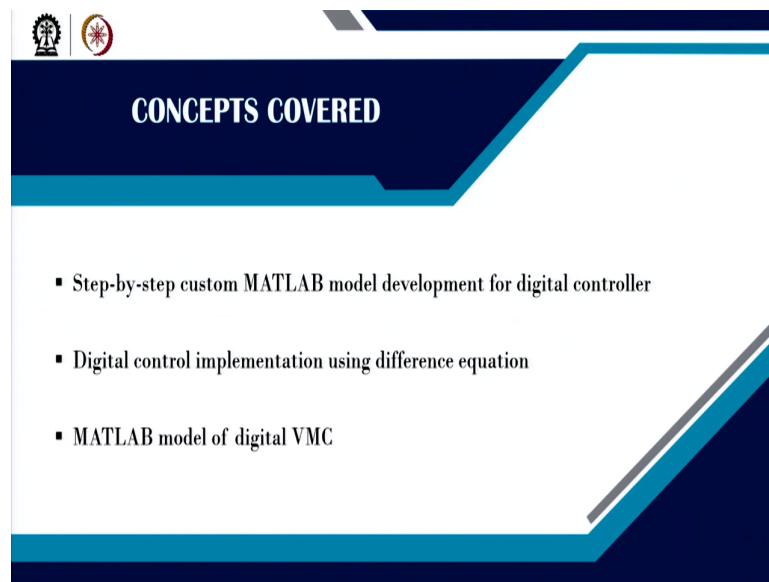


**Digital Control in Switched Mode Power Converters and FPGA-based Prototyping**  
**Prof. Santanu Kapat**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Kharagpur**

**Module - 03**  
**MATLAB Custom Model Development under Digital Control**  
**Lecture - 24**  
**MATLAB Models for Digital Controllers using Difference Equations**

Welcome back. So, this is in this lecture we are going to talk about MATLAB Model Development for Digital Controller using Difference Equations.

(Refer Slide Time: 00:34)

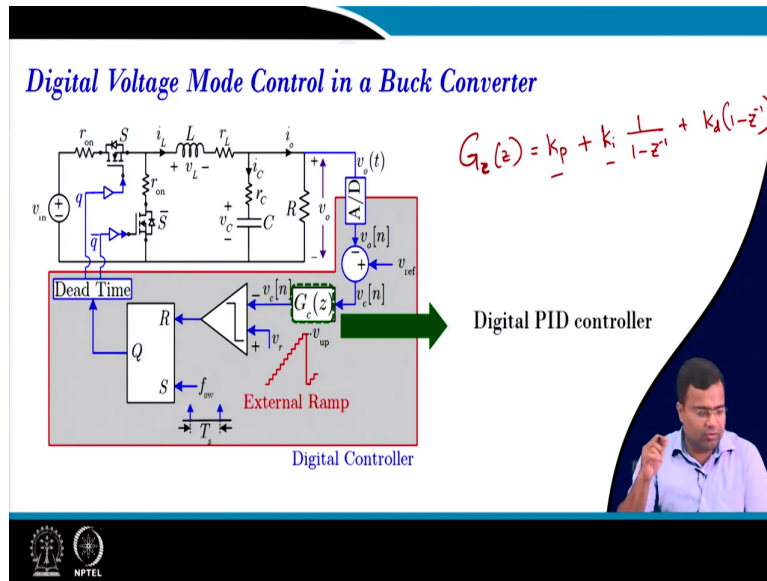


The slide features a dark blue background with a light blue geometric shape on the right side. At the top left, there are two small circular logos. The main title 'CONCEPTS COVERED' is centered in white. Below it, three bullet points are listed in white text.

- Step-by-step custom MATLAB model development for digital controller
- Digital control implementation using difference equation
- MATLAB model of digital VMC

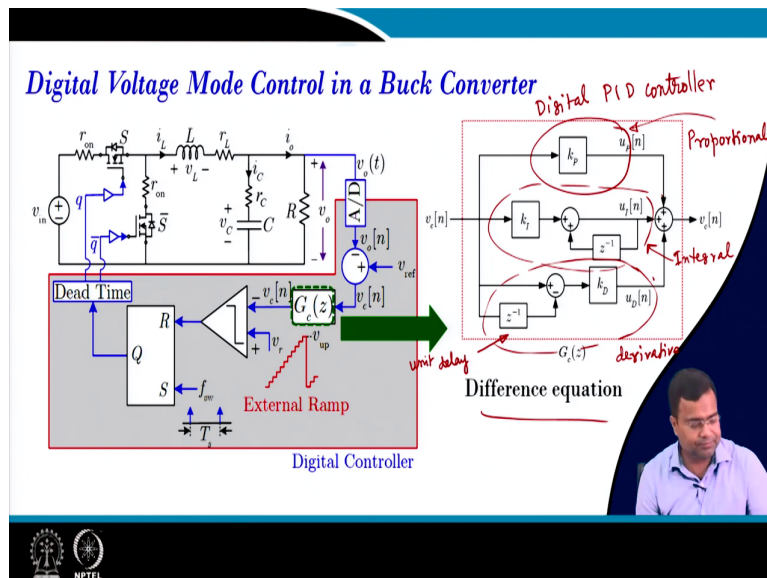
So, in this class we are going to talk about step-by-step custom MATLAB model development for the digital controller, then digital control implementation using difference equation and the MATLAB model for digital voltage mode control.

(Refer Slide Time: 00:47)



So in fact, this is the continuation of the previous lecture where we have considered this digital voltage mode control architecture. And what we did do in the previous lecture? We have considered a digital PID controller, where this PID controller we have considered this is PID controller.

(Refer Slide Time: 01:04)



So, in this PID controller what we have considered? That means, this  $G_c(z)$  sorry  $G_c(z)$  we have considered proportional control plus the integral control then  $1/(1-z^{-1})$  plus the derivative control  $(1-z^{-1})$ . So, this we have directly

plugged in; that means, we have considered one block from Simulink and z domain block and we have simply plugged in these values of  $k_p$ ,  $k_i$  and  $k_d$  and we have also written this transfer function in terms of a polynomial of  $z$  ok.

Now, in this course in this class, what do we have to do? We have to write this in the difference equation and we want to ask why we need to do that. So, the same PID controller if you take, this is the proportional controller part proportional part of the PID controller.

So, this is like you know your digital PID controller. Here this part is the proportional term, so this is proportional; one minute, so this part is the proportional term. Then this part is an integral part and this term is the derivative and all using difference equation, ok, because it is a differential equation we are talking about discrete time domain discrete domain.

Now, the first question what was the difficulty? We can directly plug in. This is one of the realizations. Because ultimately we have to realize using a controller because when you go to implementation we have to implement this block. So, we cannot directly plug in this transfer function, so that is one of the reasons.

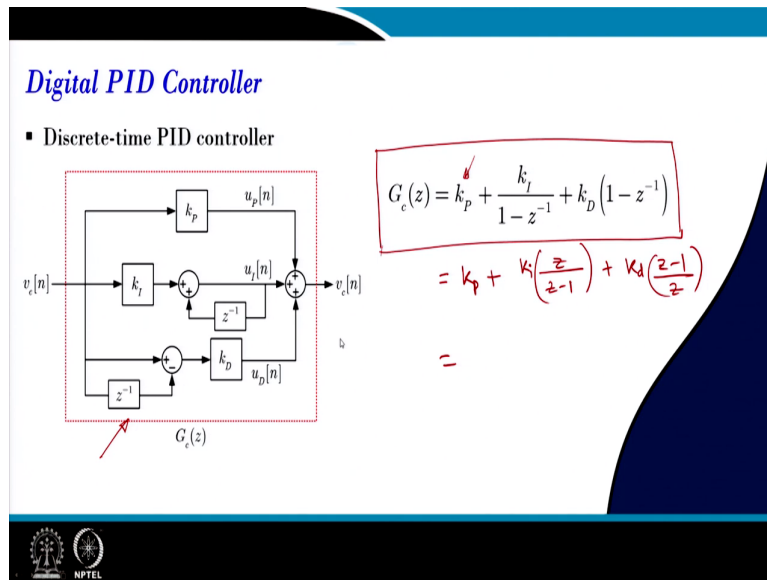
And for implementation, each of these is represented by a delay. So, this is a unit delay, this is a unit delay and this delay is computed with respect to the time; that means, a uniform sampling clock ok.

(Refer Slide Time: 03:41)

*Digital Voltage Mode Control in a Buck Converter*

So, this is what I am showing a digital voltage mode controller and I am going to discuss about this MATLAB model here, But this is the screenshot of the MATLAB Simulink model and this represents our digital voltage mode controller.

(Refer Slide Time: 03:56)



Now the first thing we want to write the digital PID controller and this can be this is the realization using that unit delay. That means, the digital PID controller can be realized using a like unit delays and also gains. And this is the overall transfer function and in the previous class we plugged in this transfer function directly from a transfer function block ok and where we can plug in these parameter values.



(Refer Slide Time: 04:28)

### Difficulties in Direct Implementation of Digital PID Controller

$$G_c(z) = k_p + \frac{k_i}{1-z^{-1}} + k_d(1-z^{-1})$$

Start the simulation

$V_e[n]$  →  $G(z)$  →  $V_c[n]$

fixed Sampling time

1. Non-uniform sampling cannot be implemented  
2. Cannot adjust the sampling instant

$t=0$   $t=T$   $t=2T$

$t_{sim}$

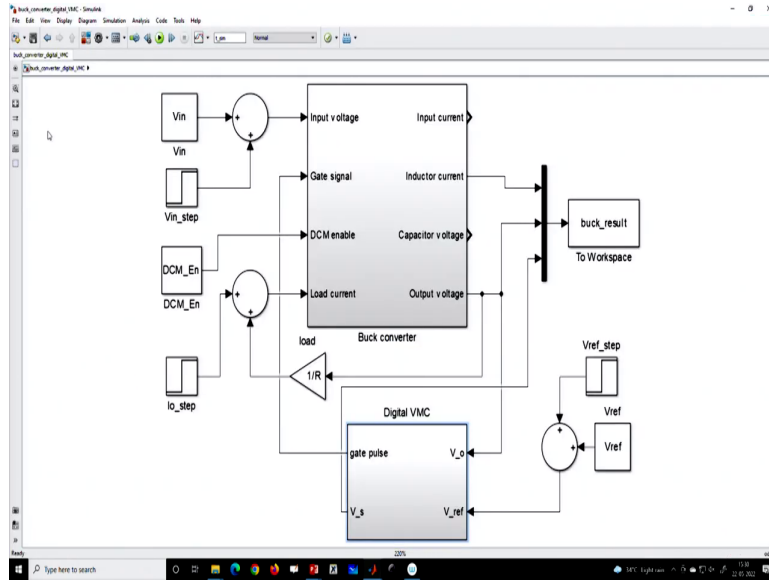
Now, what is the difficulty in this direct PID controller? First of all, whenever you plugged in let us say we are talking about a MATLAB block. And in MATLAB if you plugged in this; that means, there will be an error voltage which is in discrete time and you will get the control voltage. So, when you block when you implement such a block.

(Refer Slide Time: 04:57)

```
1- clc; close all; clear;
2-
3- %% Define parameters
4- buck_parameter;
5- f_sw=1/T;
6- Vin=12; Vref=1; R=1;
7-
8- %% Modulator gain
9- V_m=10; Fm=1/V_m; tau_d=T/10; t_s=0*0.2*T; t_c=0*0.3*
10-
11- %% PID Controller Design (analog)
12- K_p=10; K_i=50000; K_d=0.1*C; t_d=T/5;
13- % K_p=50; K_i=10000; K_d=0.5*C; t_d=T/5;
14-
15- %% PID Controller Design (digital)
16- K_pd=K_p; K_id=K_i*T; K_dd=K_d/T;
17-
18- %% Transient parameters and plots
19-
20- t_sim=5e-3; t_step=2e-3;
```

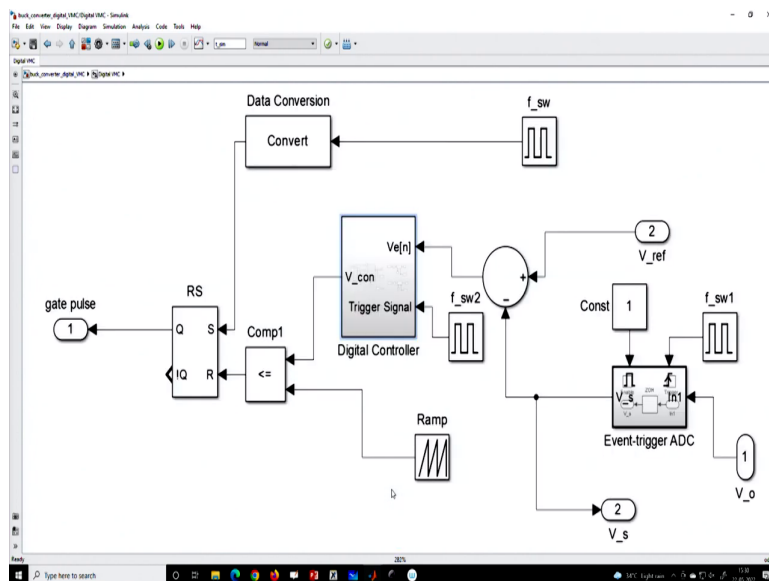
Let us go to the MATLAB and see if you know our MATLAB realization. So, if I go to this MATLAB block.

(Refer Slide Time: 05:07)



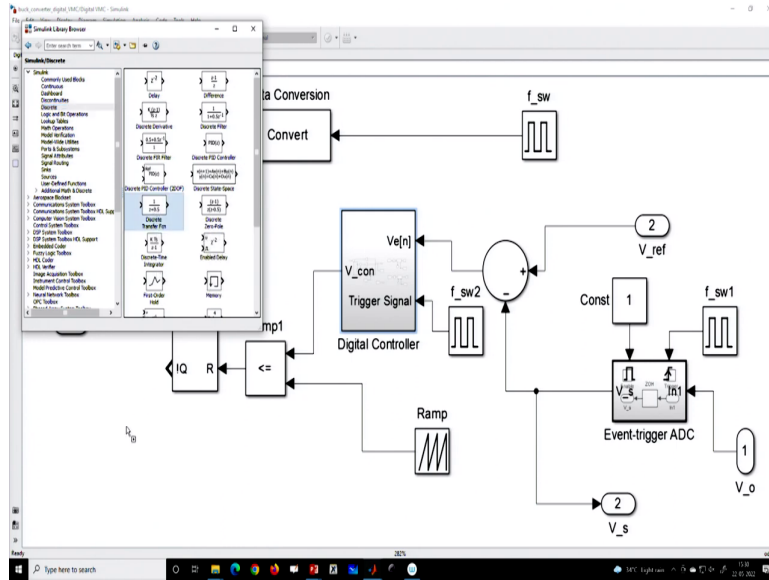
Let us, so I will go through the detailed block diagram.

(Refer Slide Time: 05:09)



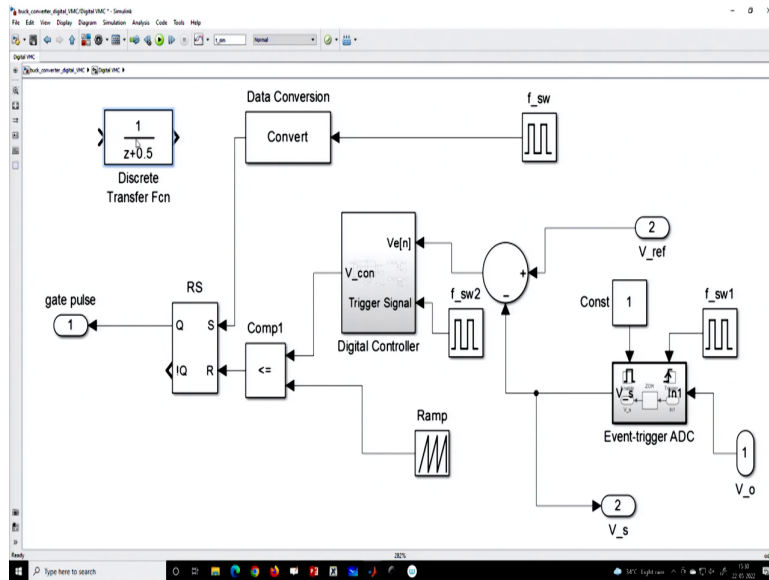
In this block let us say if we want to plug in any transfer function. For example, we want to you know take it from the library ok. So, we discussed in the last class if we want to take one transfer function from the library.

(Refer Slide Time: 05:28)



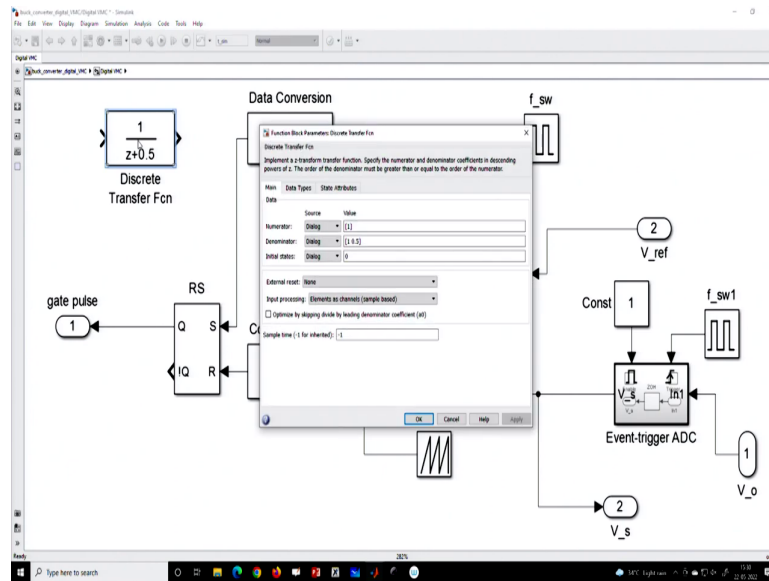
Ok. So, here will be the disc continuous-discrete ok. So, we can take the discrete-time transfer function ok.

(Refer Slide Time: 05:40)



So, what we can do?

(Refer Slide Time: 05:48)



We can actually if you go to this transfer function, if you open this block we can create this numerator and denominator coefficient. That means if we map to this thing. So, these are the coefficient; that means, we can write if you go to the previous slide, so, we can write in terms of  $z$ ; that means, here this will be  $k p$  plus  $k i z$  by  $z$  minus  $1$  plus  $k d$  into; you know what we will get?  $z$  minus  $1$  by  $z$ .

And if we get the whole polynomial then we will get some numerator polynomial of  $z$  and denominator polynomial of  $z$  and here whatever in the MATLAB I am showing. So, these are the polynomial that we have to plug in, these polynomials are in (Refer Time: 06:34). And here we have to also mention if you go to the MATLAB block, let us go. So, here we have also maintained mention the sampling time.

That means we have to specify the sampling time which here we are talking about uniform sampling  $t$  and then you plugged in the polynomial value. So, now, we can realize this using this block; that means, this polynomial which will have a numerator and denominator and you have to only provide the sampling time.

Now, what is the problem here? The problem is that whenever you run a simulation, that means, if you start the simulation start your simulation that means, let us say this is the time, this is the time  $t$  and your it is a  $0$  time this is a  $0$  time and let us say this is our total simulation time  $sim$ . At time  $t$  this block will be computed; that means, this block will be computed at the beginning.

That means, at the beginning as you can imagine there is a virtual clock, again the next clock virtually comes like this. So that means, this interval is just  $t$ . That means, if you take the simulation time at  $t$  equal to 0 this block will be computed then at  $t$  equal to capital  $T$  this block will be computed, and the next  $t$  equal to twice  $T$  will be computed. So, first of all, the computation of this block will always happen with respect to clock time  $t$  and it will start from 0 times; that means, the start of the simulation.

So, first of all, there are two problems. One problem; you cannot use; means, you cannot use any non-uniform sampling. Non-uniform or event-based sampling cannot be used. Non-uniform sampling cannot be implemented because here in this way, you cannot simply adjust the period inside the block because this block is fixed.

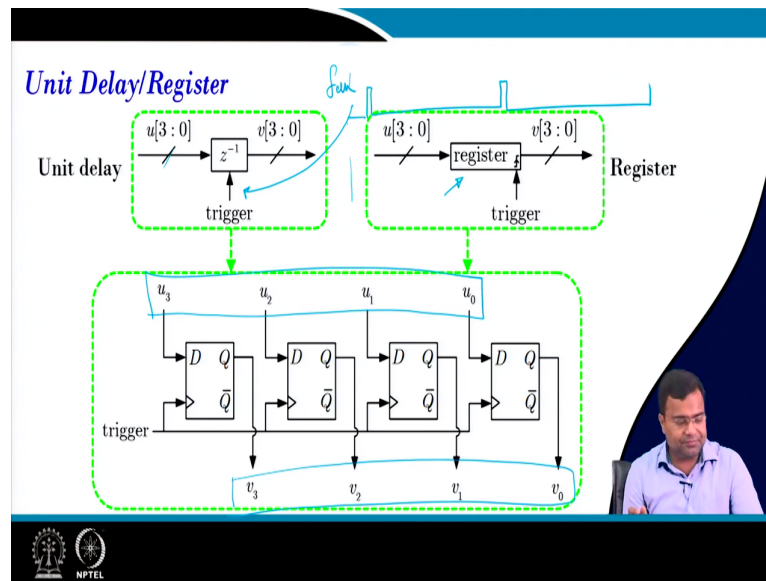
So, this is a fixed sampling time, so you cannot implement using this block. So, this is number 1. Number 2; you cannot adjust the sampling instant. What does it mean? That means, suppose I want to sample not exactly at that point, I want to sample here, I want to sample here. It is time  $t$  only, but I want a different instance of time.

That means, I want to sample which rate is  $t$ ; that means, the sampling interval is  $t$ , it is the same, but the point of sampling I want to change because this is necessary if you want to customize, let us say if you go to the datasheet of any A to D converter. So, we need to understand that when you send the sampling clock then it will take time to convert the data.

So, you want to you should provide sufficient time for that your ADC conversion time to happen as well as controller computation to happen, and then only you will start the clock. So, there has to be a difference in the time instant between your switching clock and the sampling clock. So, that means we want an adjustable sampling clock.

That means, where the time period may be the same it may be uniform sampling, but the point of sampling should be customizable and this block does not allow that. Secondly, this block does not allow. So that means, these two are not possible that is the difficulty. So, you should not plug in directly this block.

(Refer Slide Time: 10:38)



Then how to overcome this? And this course today class is dedicated to that purpose. So, instead of directly plugging the transfer function sorry plug in the transfer function from the library we want to implement this transfer function using a difference equation. That means, if you go back to this, this is the difference equation where we will use unit delay, but we still need to synchronize with certain clock ok.

So, how to do that? So, first, we need a delay block. How does the delay work? So, the delay means unit delay. So, that means, you need to delay with respect to a clock. Now, that clock can be even a time-varying clock; that means, a non-uniform clock or it can be a uniform clock or that clock edges can be different. But we need just one delay; that means, between two edges of the clock.

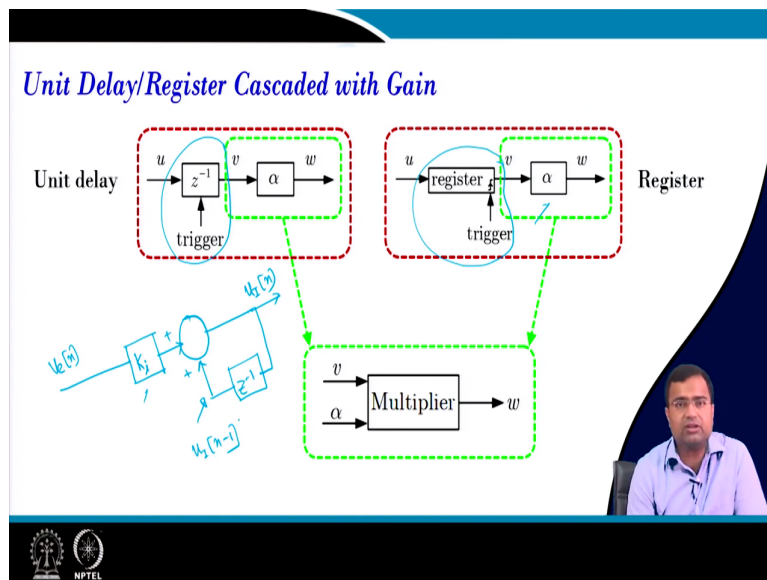
That means, if I want this kind of clock this is my  $t$ . So, this is some clock and I want to delay this signal with respect to this clock. So that means how to do that? So, this will act like a trigger pulse; that means this pulse will be used as a trigger pulse. Whenever this trigger comes, then it will be, it will be synchronized with respect to that clock and it will create a delay of one unit.

But the data that you are delaying is vector data, it is not one bit because if you are going for digital; that means, you need to convert the analog signal to digital and then it will be either there should be some bit size whether it is an 8-bit data, 12-bit data and so on. So, it is vector data and the data has to be delayed by one unit of time.

So, we need to use a register and it is because if it is a single data then it is simply a D flip-flop. So, you define. In the D flip-flop, we know what happened. Whenever the clock edge comes then only the output will take the input D. The Q will be equal to D when the edge comes. So, if such multiple like 8 such D flops are connected in parallel then we create an 8-bit register ok and all this will be common, it will have a common clock.

And this is the realization. So, that means you have a such clock. So, these are the 4-bit data; that means,  $u_0$  to  $u_3$  are the 4-bit data and it will generate another 4-bit delayed data and that delay will be controlled by this trigger clock. So, we want to implement a clock base delay.

(Refer Slide Time: 13:09)



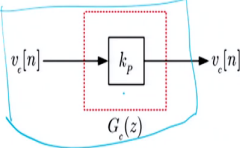
So that means, now we know how to generate delay and I will show you in MATLAB how to do that. And once you use that delay in actual realization using a register, then you have to multiply with the coefficient because this multiplication will give you because if you take about the PID controller, you know what we have seen.

If we have that  $Ve$  of  $n$  then what does the controller do? So, you need one; that means, first of all, it has to be multiplied by the integral gain. And this will be added with what? Whatever the output that will be delayed by one unit and this is the output of the integrator. So, this part they are all added, this part is  $u$  I of  $n$  minus 1 ok. So, that means we need a scaling factor as well as we need a delay.

(Refer Slide Time: 14:13)



*Digital Proportional Controller*

- Proportional Controller



Uniform sampling

Difference equation:  $v_c[n] = k_p v_c[n]$

$$G_c(z) = k_p$$


So, now, in a proportional controller, you can make this multiplication with respect to the clock because when you go to; because what will happen? This proportional gain is just a multiplier, but we may have to use this whole operation in synchronizing with a clock. Because when we talk about Verilog HDL we want to discuss that we want to compute this PI k P, k I, k D within synchronized with the clock.

Because their computation time is due to the proportional controller, the computation time of the integral controller and the computation time of the derivative controller may be different. But if we synchronize them with respect to the clock then all of them will be updated at one time. So, we will do a clock synchronize PID controller. So, that is why here also you can use a clock to synchronize and it is just a simple multiplication.



(Refer Slide Time: 15:07)

### Digital PI Controller

- Proportional-Integral Controller

Handwritten equations:

$$u_p[n] = k_p v_e[n]$$

$$u_i[n] = u_i[n-1] + k_i v_e[n]$$

Difference equations:

$$v_c[n] = k_p v_c[n] + u_i[n]$$

$$u_i[n] = k_i v_c[n] + u_i[n-1]$$

Handwritten note: "delayed version of  $u_i[n]$ "

Clock driven sampling

If you do a digital PI controller again this is a delay we have discussed. So, this represents a unit delay and this can be realized by a register and which will be shifted by a clock; that means, it is a clock-driven register and this can be realized. So, as you can see this part is the proportional part where the proportional part is simply  $k_P$  into  $V_e[n]$ . And we will be going to Verilog then we will see this will be like a data flow modeling which will be this.

Then integral part will have  $u_I$  of  $n$  minus 1 plus  $k_i$  into  $V_e$  of  $n$ . So, this is there and this delay is generated by this register.

(Refer Slide Time: 16:05)

### Digital PID Controller

- Proportional-Integral-Derivative Controller

Handwritten equations:

$$v_c[n] = k_p v_c[n] + u_i[n] + u_D[n]$$

$$u_i[n] = k_i v_c[n] + u_i[n-1]$$

$$u_D[n] = k_D (v_c[n] - v_c[n-1])$$

Difference equations:

$$v_c[n] = k_p v_c[n] + u_i[n] + u_D[n]$$

$$u_i[n] = k_i v_c[n] + u_i[n-1]$$

$$u_D[n] = k_D (v_c[n] - v_c[n-1])$$

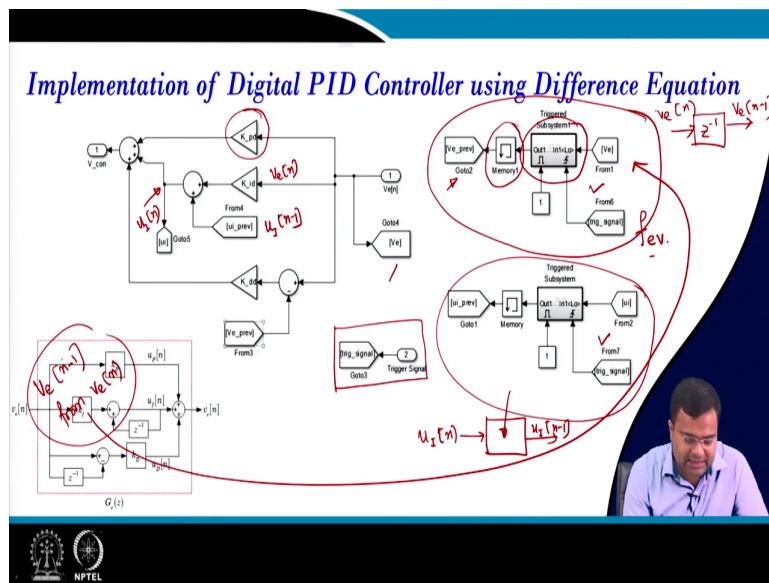
Clock driven sampling

Similarly, if you go to; so, you can see it is a clock-driven delay. That means whatever difference equation you are getting that one cycle delay of this that means this portion is nothing, but a delayed version of  $u_i$  of  $n$ . That means this one is a delayed version. And this delay will happen concerning the clock. Although it is a one-unit delay, we also want to customize the edge of updating as well as the edge of delay.

So, it is not completely like a transfer function. Similarly, for the PI controller, we can also generate. So, here is what we need. This has to be  $v_{e,n-1}$ , this is  $v_n$  because this is  $v_e$ . And this delay means this is a delayed version of this. So, this delay is again created by a register and a clock. So, this is a clocked-driven sampling.

That means the update is happening with respect to clock ok. So, this is the proportional part, this is an integral part and this is a derivative part and the integral part is written like this, the derivative part is written like this, and the overall implementation is like this.

(Refer Slide Time: 17:23)



This is the overall PID controller implementation using MATLAB and this is a screenshot and I am going to show this entire implementation. So, you can see how can we implement MATLAB with the clock driven. So, this is our clock which will decide; that means, you can say this is our event.

Now this clock can be uniform sampling; that means, the clock frequency can be fixed or this clock can even be a variable frequency clock because we will extend the same approach. The

same approach is applicable when you go to variable frequency modulation with event base sampling.

So, the same block is applicable. What does it do? So, this block we have discussed this A to D converter block. So, this is a triggered ADC; which means this ADC is not just a simple ADC. Here we will take the sample with respect to an event base; that means the sampling will happen with respect to one event. And if you take any A to D converter; means, A to D converter the conversion process starts a clock edge comes.

So, that is why you are creating virtually a similar A to D operation using MATLAB simulation. So, this block is to create the error signal. Here the error signal is already generated. That means it is coming from the output block; that means, if you are talking about you know I will go to the complete diagram, but here I am just showing.

If you have a discrete time error then this error is supposed if you want to create  $V_{e n - 1}$  from  $V_{e n}$  and this process is shown here. That means, and this memory is needed at the start of the simulation because if you start the simulation since there is no value here, so; which means, it cannot start the simulation it will show an error because it has to have something here.

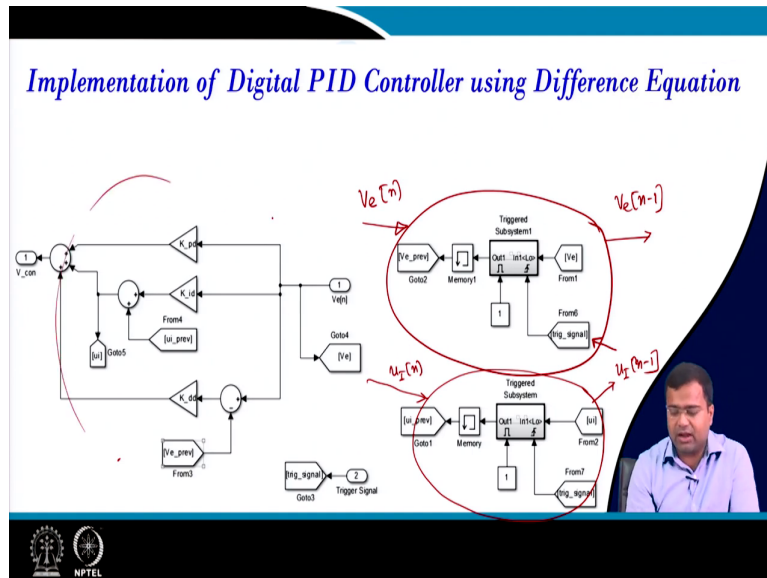
So, that is why the memory block is there and I will go to the simulation show. So, this block is to perform this particular operation; that means, it will if this block will generate just a  $z$  inverse  $V_{e n}$  and it will generate  $V_{e n - 1}$  ok. What is this block? This block will generate  $u_{I n}$  and if it passes through this block then it will generate  $u_{I n - 1}$ .

So, this is another delayed block that will delay the integral term and the proportional term is just a scaling factor. So, it is there. This is the integral block you can see, the whole block is the integral block, this is the previous value ok. So, if we ok; so I am just highlighting. This is our  $u_{I n - 1}$ , this is our  $V_{e n}$  and this particular signal is our  $u_{I n}$  and this is going to the go-to block.

And this is the trigger pulse that is coming which is used in various blocks. You can see the trigger clock is used here, the trigger clock is used here. So that means, this is the clock that will synchronize all the updates of this delay operation ok and this is the realization of this PID controller.

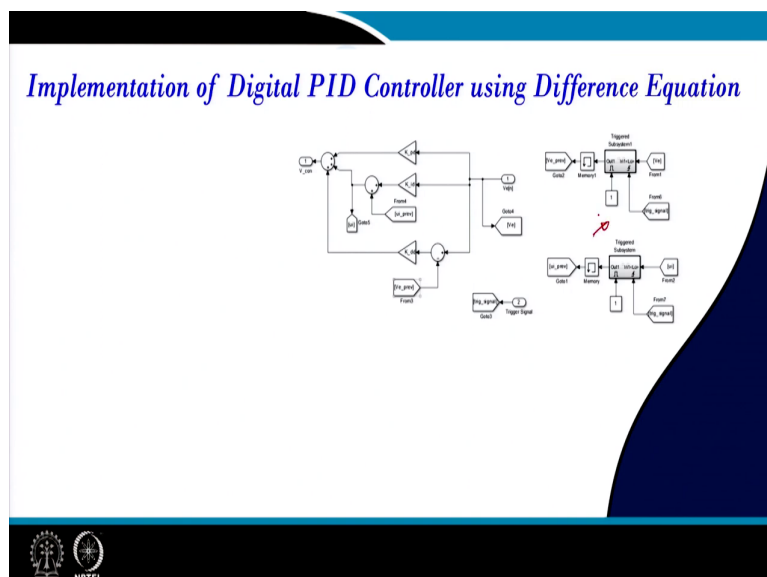
So, if you rub up this part, yeah. So, if this block is the; so, this whole block is realized by; that means, this whole block is realized here ok.

(Refer Slide Time: 21:27)



Now, again I am showing. So, this is the what block? We have discussed this block too. What is the task if this is the block, input to this block is what?  $V_e$  error  $n$ . And the output to this block is what?  $V_e$   $n$  minus 1 and it is driven by this external clock. Then what is this block? Again the input to this block is  $u_I$  of  $n$  and the output to this block is  $u_I$  of  $n$  minus 1 ok and then this is a complete PID controller that I have shown here ok.

(Refer Slide Time: 22:10)



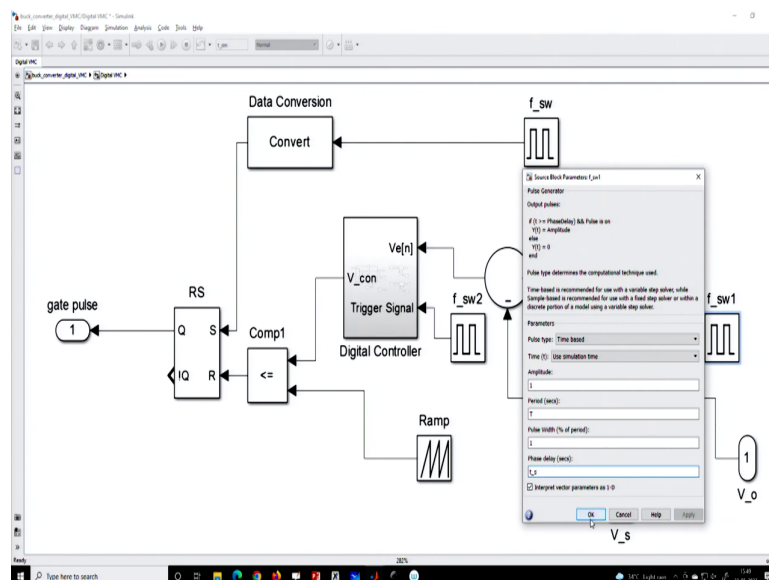
So that means, now we will go to MATLAB to show the implementation of the PID controller. So, we are not going to use this transfer function because we have identified this way of implementation is not going to solve our purpose. After all, we want to sample with respect to time.

So, you do not want to start sampling may be at the start of the simulation, so we want to customize. Sometimes in digital control, we want to speed up know sampling process during the transient. So; that means, we need an adaptively varying sampling process during the run time. Another requirement maybe we want to use event-based sampling.

So, to accommodate all these kinds of customized sampling methods, we should not use any by default transfer function block, we should realize using difference equation. So, here in this MATLAB, this is a complete block diagram that I have demonstrated. Then in this block, you can see; that, first of all, this is a power stage converter which you have explained multiple times. Then there is the input voltage, DCM enables, load step and everything is explained.

Then this is the controller block. It takes the output voltage and the reference voltage and it generates the gate signal and this is a sample point. If you go inside you know this output voltage pass through an A-to-D converter. So, this is our A-to-D converter. So, you can see this is an A-to-D converter, and this A-to-D converter is driven by a sampling process given by a clock.

(Refer Slide Time: 23:37)

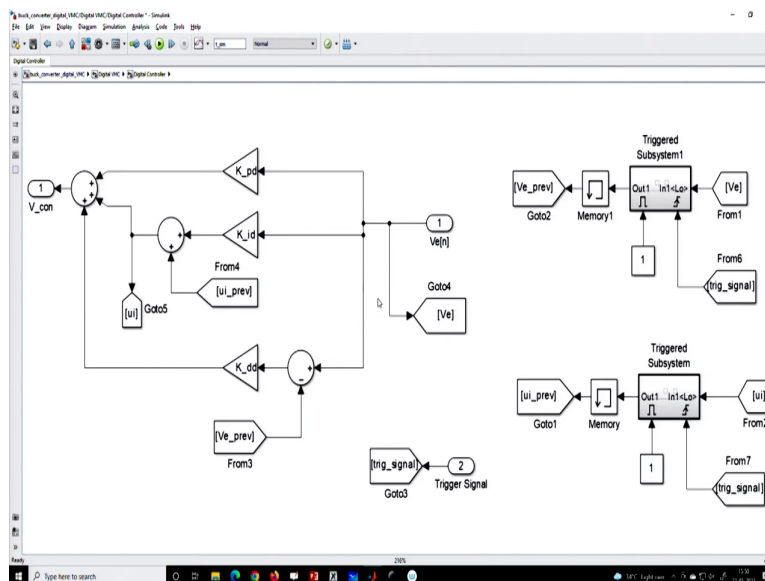


And this clock we can delete this signal; that means, now we can customize the sample edge ok. Or we can use any other clock which is respectful to the event and that will be discussed in the subsequent class. Then this is our sample voltage; that means, this is our sample voltage you can see. It is going to an output which means, we can see the scope. Then this is our reference voltage minus the sample voltage. So, this is the error sample error voltage and which is going to the digital controller.

Now, the controller computation will be carried out with respect to this clock; that means it may not be the same clock of ADC that will be used for computation. Because even if you send the start command of the sampling command and that sampling process will take time, because ADC conversion there is a finite time for ADC conversion time. Once the data is ready and it passed through the error then only the controller can start computing otherwise it will take the wrong data.

So, this clock has to be delayed. So, this clock has to be delayed with respect to this clock so that the difference can accommodate the ADC conversion time. Now, if we go inside this controller what is there? The rest of the blocks are the same, which means you can see this modulator is a trailing edge modulator with respect to the switching frequency clock. This is a sawtooth waveform. You can also add a quantized block here.

(Refer Slide Time: 25:05)



But the objective here if you go inside, is that we have explained. So, this block is to generate the delay of the error voltage; that means,  $V_{e,n}$  this block will generate  $V_{e,n-1}$ . Similarly, this block is the same, but it is used to generate  $u_n$  from  $u_{n-1}$ .

Because, if we talk about the integral it is an incremental integral process discrete time, where  $u_n$  is equal to  $u_{n-1} + k_i \text{ into the error voltage}$ . So, this block is generating that previous term that is generated and it is used here. This block is the trigger pulse and we are taking it from outside. So, as I said that controller computation will happen with respect to a clock and number 2 is this clock.

So, this is the port and this clock is given here and here; that means all the operations like a delayed operation etcetera is happening with respect to this clock. And finally, this is the overall implementation of the PID controller where this is the proportional part, this part is the if you take this part, overall is the integral part where this is your  $U_i$  of  $n$  and this part is the derivative part and all are added together to generate the PID controller ok.

And if we go to the simulation; that means, that this is the simulation. Now, I have shown in the previous class, so if you set  $K_p$ ,  $K_i$  some value, this we have taken in the analog domain.

(Refer Slide Time: 26:44)

```

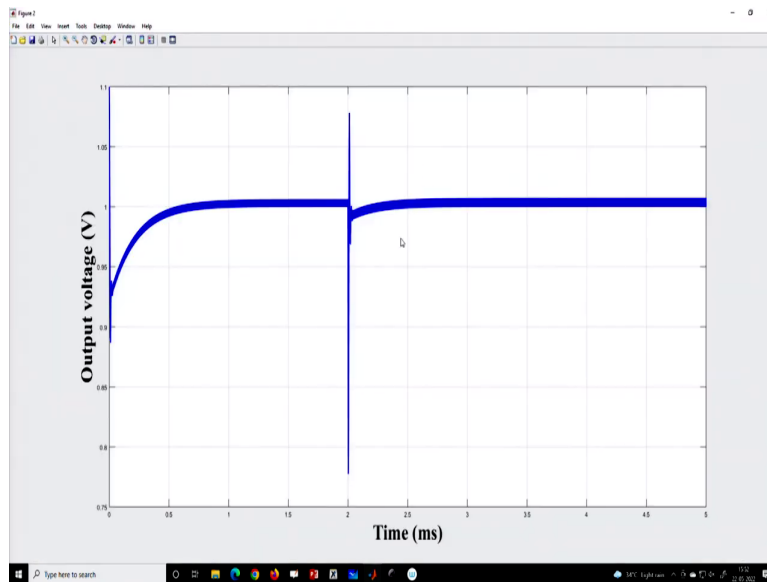
4- buck_parameter;
5- f_sw=1/T;
6- Vin=12; Vref=1; R=1;
7
8 %% Modulator gain
9 V_m=10; Fm=1/V_m; tau_d=T/10; t_s=0*0.2*T; t_c=0*0.3*
10
11 %% PID Controller Design (analog)
12 K_p=10; K_i=50000; K_d=0.1*C; t_d=T/5;
13 % K_p=50; K_i=10000; K_d=0.5*C; t_d=T/5;
14
15 %% PID Controller Design (digital)
16 K_pd=K_p; K_id=K_i*T; K_dd=K_d/T;
17
18 %% Transient parameters and plots
19
20 t_sim=5e-3; t_step=2e-3;
21 delta_io=20; delta_Vin=0; delta_Vref=0;
22
23 buck_converter_simulation;

```

Then I also discussed in the last class how to convert into a digital domain. That means the discrete time proportional gains remain the same. The integral gain is nothing but analog integral gain multiplied by the sampling time and the derivative gain in the digital domain is

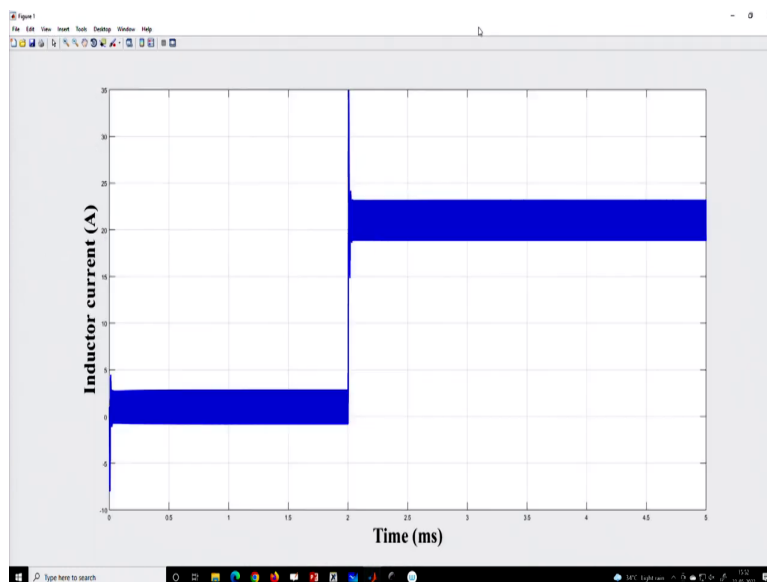
nothing but a continuous time derivative that will be divided by the sampling time. In that way, we have run and if you run the simulation, so I am just carrying out a load transient of a digital voltage mode converter.

(Refer Slide Time: 27:10)



And this is the one which is shown.

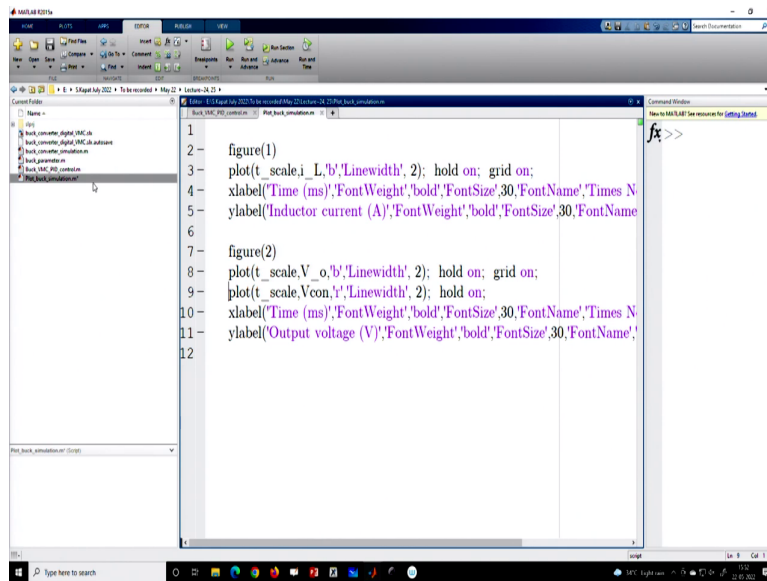
(Refer Slide Time: 27:15)



So, you can see that you know this is the operation.



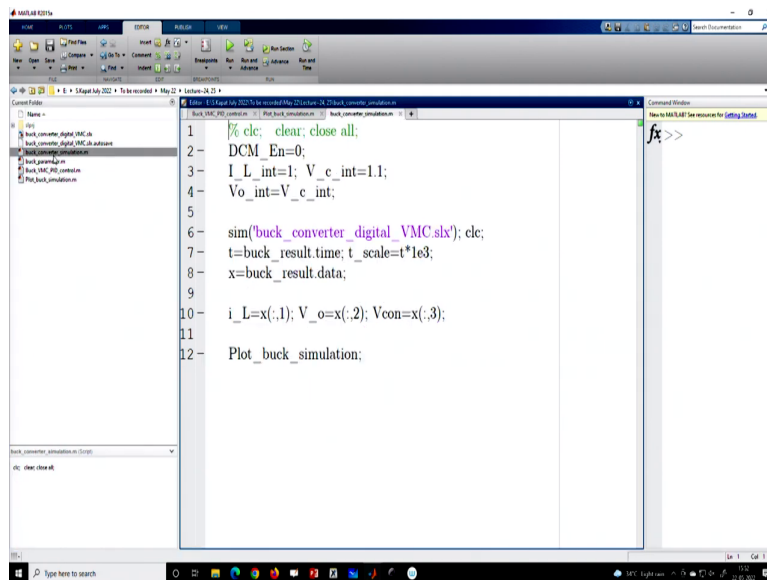
(Refer Slide Time: 27:21)



```
1  
2- figure(1)  
3- plot(t_scale,i_L,'b','Linewidth',2); hold on; grid on;  
4- xlabel('Time (ms)',FontWeight,'bold',FontSize',30,FontName','Times N  
5- ylabel('Inductor current (A)',FontWeight','bold',FontSize',30,FontName'  
6  
7- figure(2)  
8- plot(t_scale,V_o,'b','Linewidth',2); hold on; grid on;  
9- plot(t_scale,Vcon,'r','Linewidth',2); hold on;  
10- xlabel('Time (ms)',FontWeight','bold',FontSize',30,FontName','Times N  
11- ylabel('Output voltage (V)',FontWeight','bold',FontSize',30,FontName','  
12
```

Now in this plot, if I go to plot Simulink and if you want to know plot that; that means if you want to see the sample point.

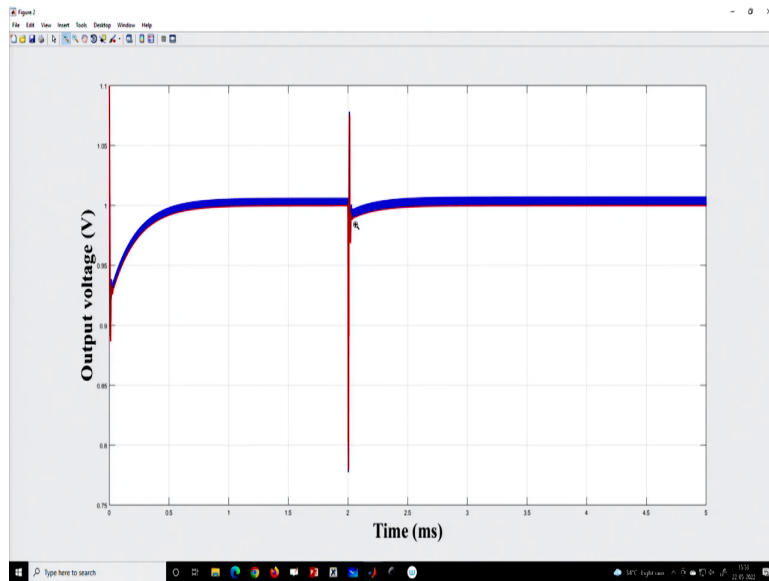
(Refer Slide Time: 27:35)



```
1  
2- % cfc; clear; close all;  
3- DCM_En=0;  
4- I_L_int=1; V_c_int=1.1;  
5- Vo_int=V_c_int;  
6- sim('buck_converter_digital_VMC.slx'); cfc;  
7- t=buck_result.time; t_scale=t*1e3;  
8- x=buck_result.data;  
9  
10- i_L=x(:,1); V_o=x(:,2); Vcon=x(:,3);  
11  
12- Plot_buck_simulation;
```

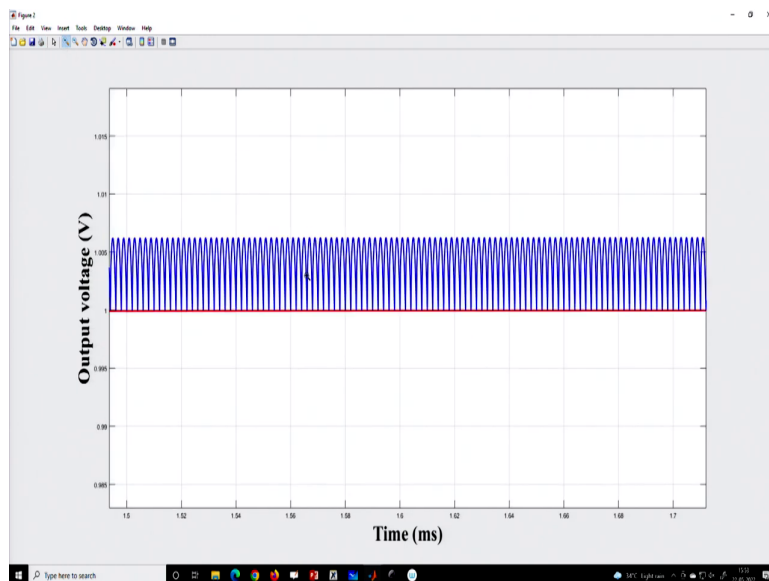
So, let me check that whatever we are capturing is the third channel. So, what is the third channel here? So, the third channel is V\_c sample point ok. So, let us go to the simulation and now run this plot simulation.

(Refer Slide Time: 27:50)



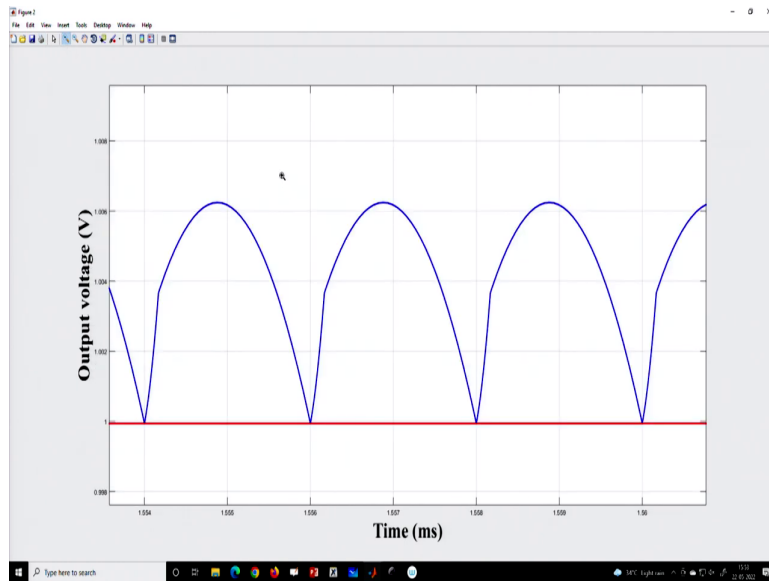
So, we have the capture and you can see the red one is indicate the sample value.

(Refer Slide Time: 27:56)



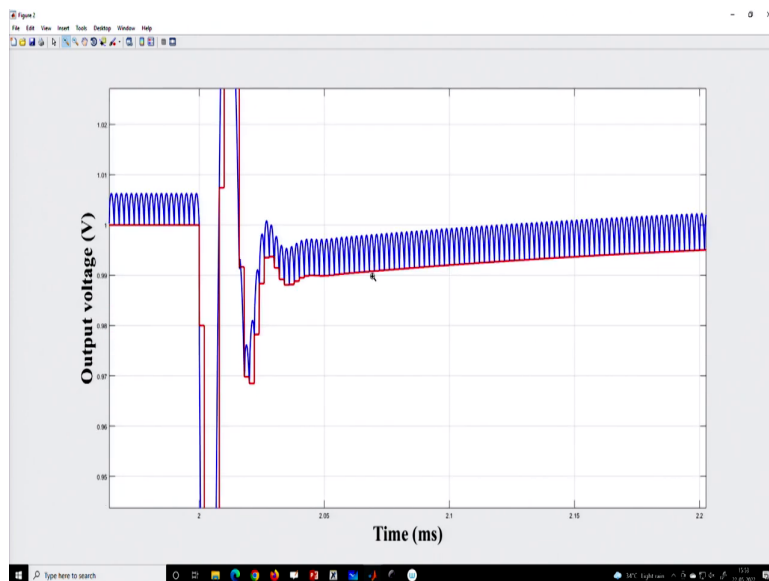
And we want to show that.

(Refer Slide Time: 27:58)



Now, this is in a steady state.

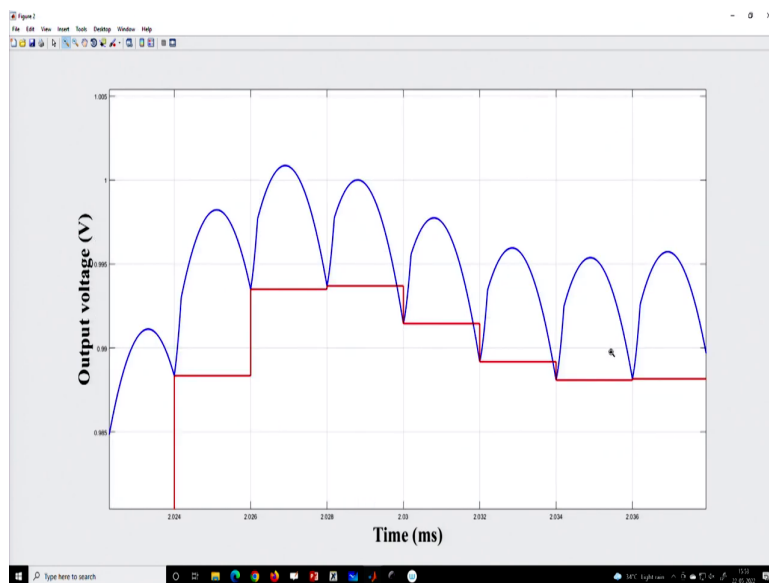
(Refer Slide Time: 28:01)



(Refer Slide Time: 28:03)

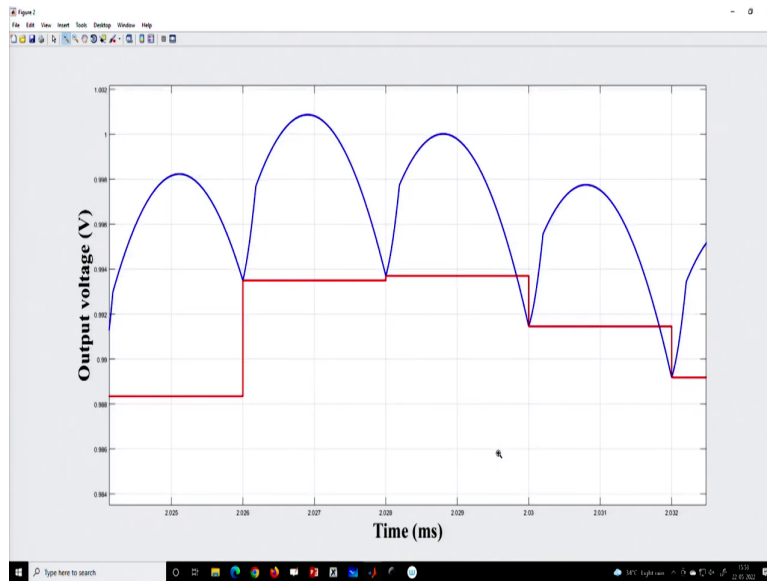


(Refer Slide Time: 28:05)



Now during transient if you see that the switching point and sampling point can be even shifted.

(Refer Slide Time: 28:08)



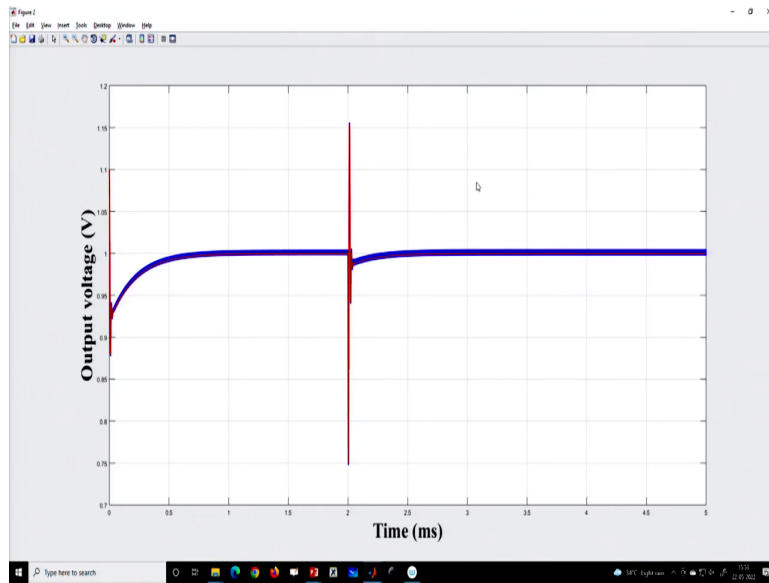
So, here we have not used any delay, but if you use any delay; that means, you know we have the provision to adjust the delay. So, let us go back to our code because here we have used  $t_s$  to be this, but suppose we want to use this  $t_s$  which is a delay.

(Refer Slide Time: 28:29)

```
1 clear; close all; clear;
2
3 %% Define parameters
4 buck_parameter;
5 f_sw=1/T;
6 Vin=12; Vref=1; R=1;
7
8 %% Modulator gain
9 V_m=10; Fm=1/V_m; tau_d=T/10; t_s=0.9*T; t_c=0*0.3*T;
10
11 %% PID Controller Design (analog)
12 K_p=10; K_i=50000; K_d=0.1*C; t_d=T/5;
13 K_p=50; K_i=10000; K_d=0.5*C; t_d=T/5;
14
15 %% PID Controller Design (digital)
16 K_pd=K_p; K_id=K_i*T; K_dd=K_d/T;
17
18 %% Transient parameters and plots
19
20 t_sim=5e-3; t_step=2e-3;
```

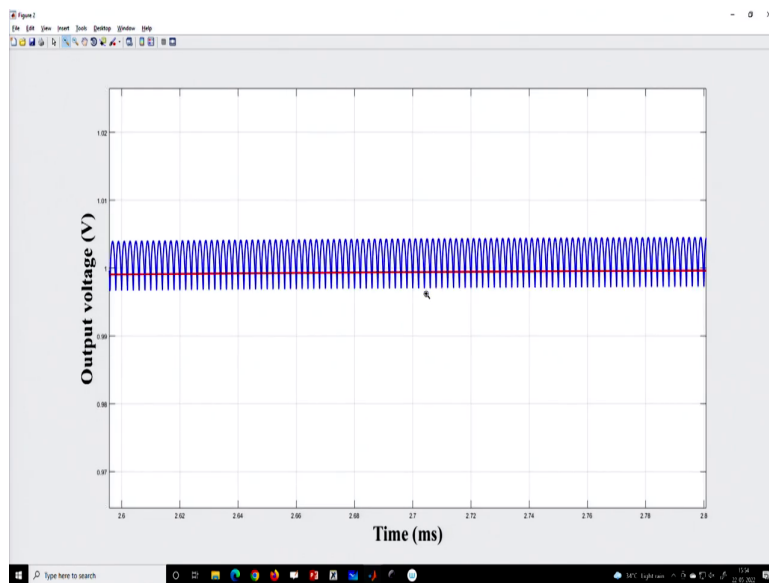
And maybe we can just use a 0.9, let us run it. We are just delaying you know.

(Refer Slide Time: 28:38)



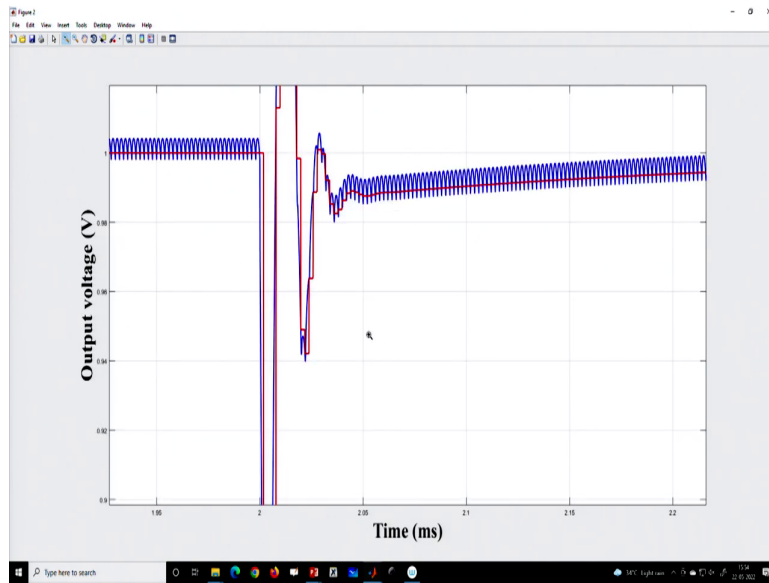
So, let me check.

(Refer Slide Time: 28:43)

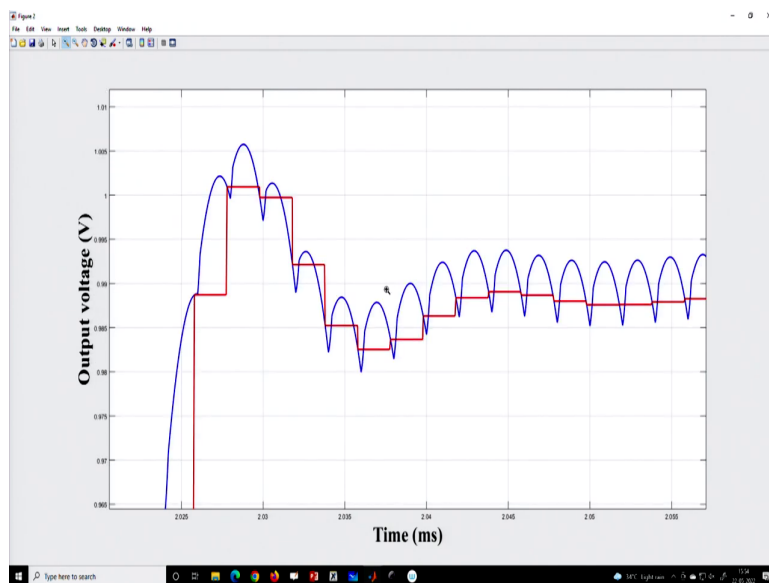


Now we have applied a delayed version of this delay.

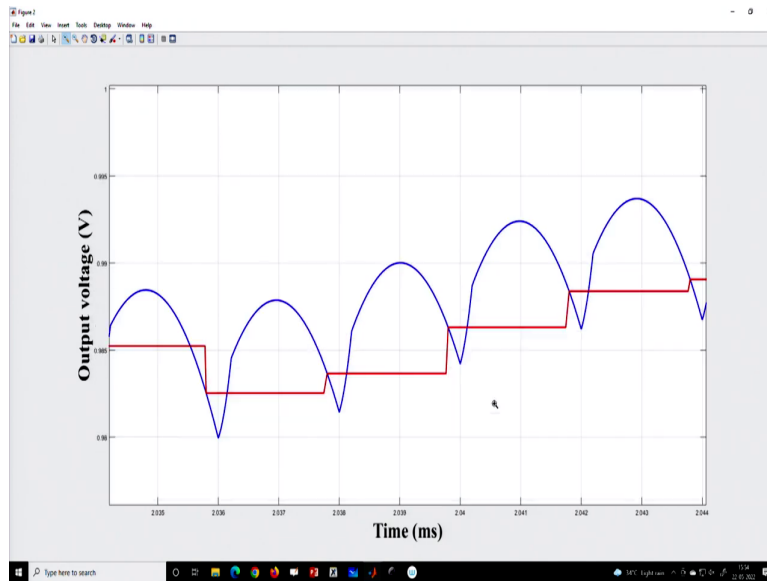
(Refer Slide Time: 28:46)



(Refer Slide Time: 28:48)

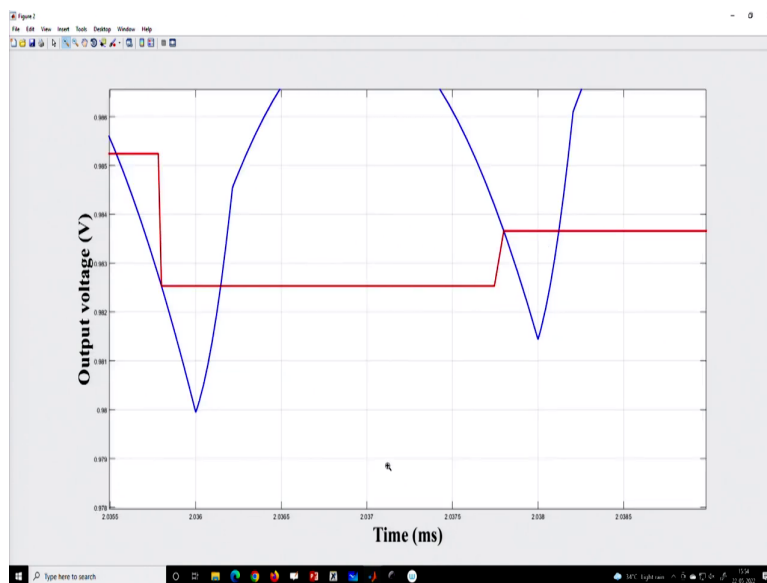


(Refer Slide Time: 28:50)



Now you can see we are sampling earlier than switching. That means the first sampling is happening.

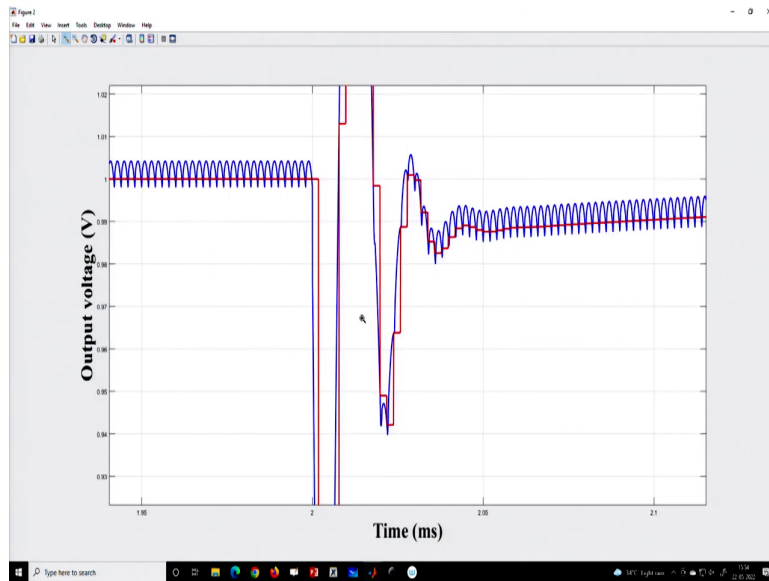
(Refer Slide Time: 28:56)



So, if you look at one cycle during the transient.

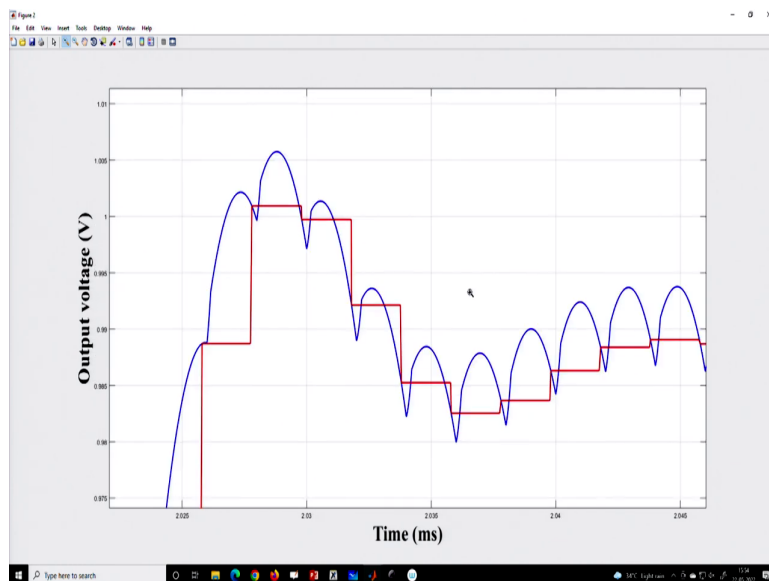


(Refer Slide Time: 28:59)



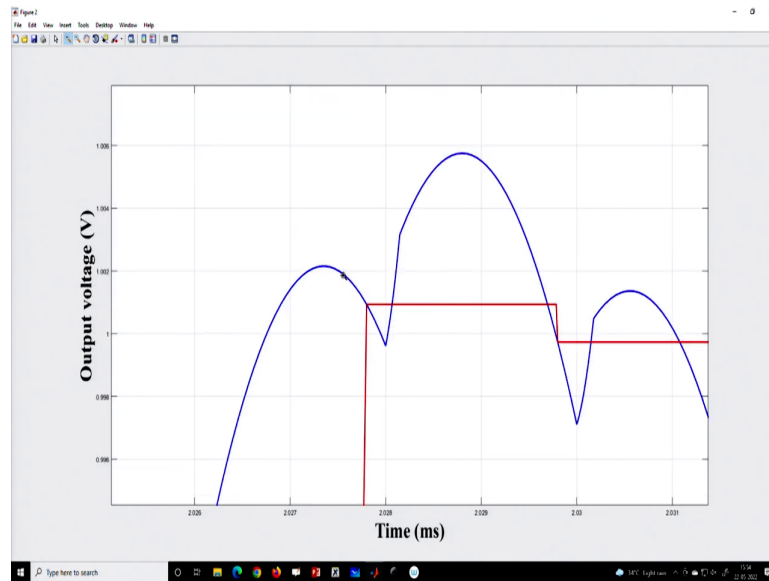
Let us say this is transient.

(Refer Slide Time: 29:01)



If you look here switching point is happening here, but the sampling point is happening here.

(Refer Slide Time: 29:07)



That means if we just zoom in first it is sampling here and then switching sampling here and then switching. So, we can customize this sampling point ok. So that means, we have learned that we can sample in the controller either by event clock or even adjustment we can adjust the edge of sampling with respect to the switching clock, by using a difference equation. And by using this clock-driven difference equation where we can implement any type of controller PID, PI anything.

(Refer Slide Time: 29:41)

## CONCLUSION

- Step-by-step custom MATLAB model development for digital controller
- Digital control implementation using difference equation
- MATLAB model of digital VMC

So, in summary, we have learned step-by-step custom MATLAB model development for digital controllers then digital control implementation using difference equation and MATLAB models for digital voltage mode control. And we will just take once again in the next lecture the digital voltage mode control implementation overall implementation. With that, I want to finish it here.

Thank you very much.