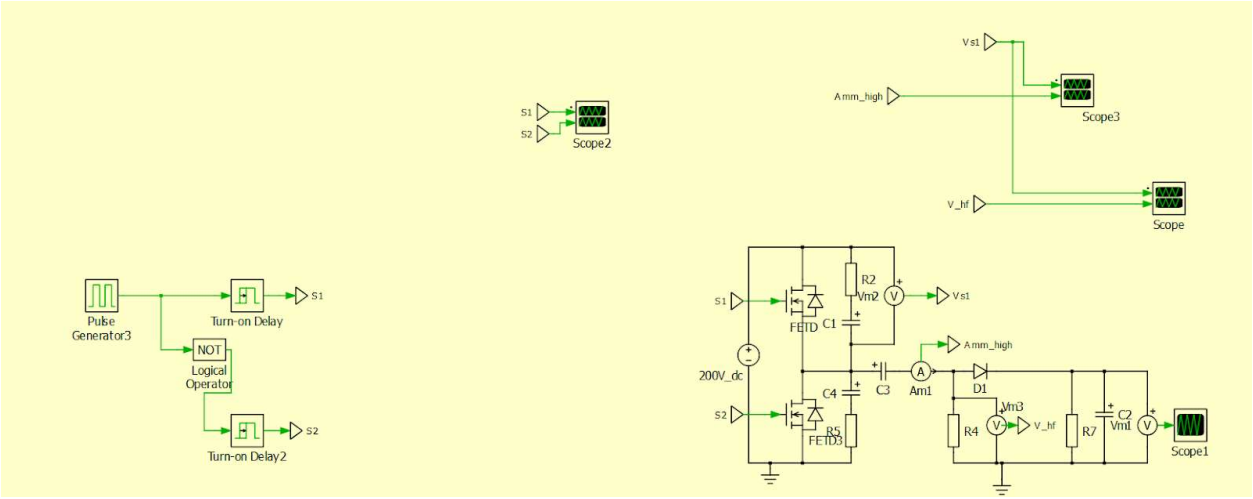
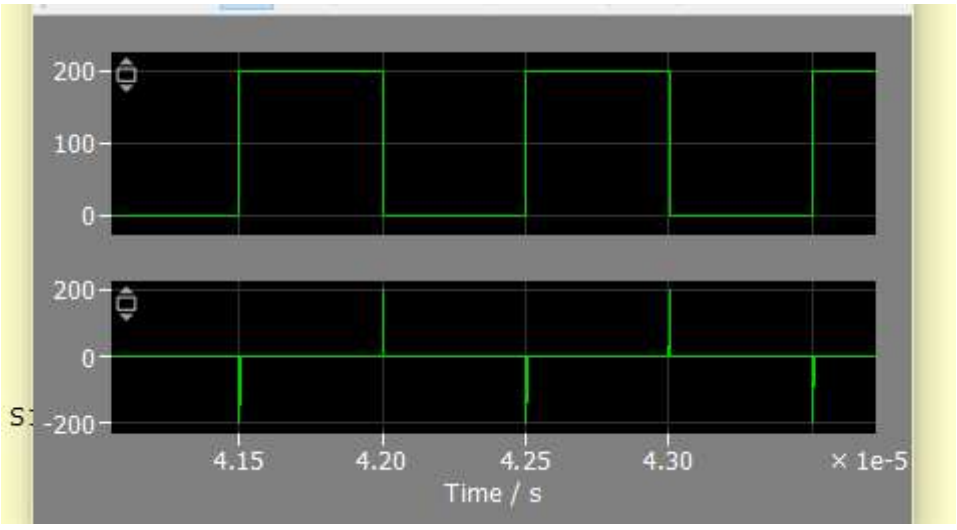
SURA PROJECT

You also need to be vary - because sometimes the topology can do soft switching and the reason it is unable to do so is because the inductor current itself is not enough to

Running the Initial simulation on the half Bridge converter shows some really surprising results to me



This is the current Topology that I am testing out.



The Upper plot is Vs1 and the lower is V_hf - now the issue with this is - it is exactly opposite to what the other topology itself predicted.

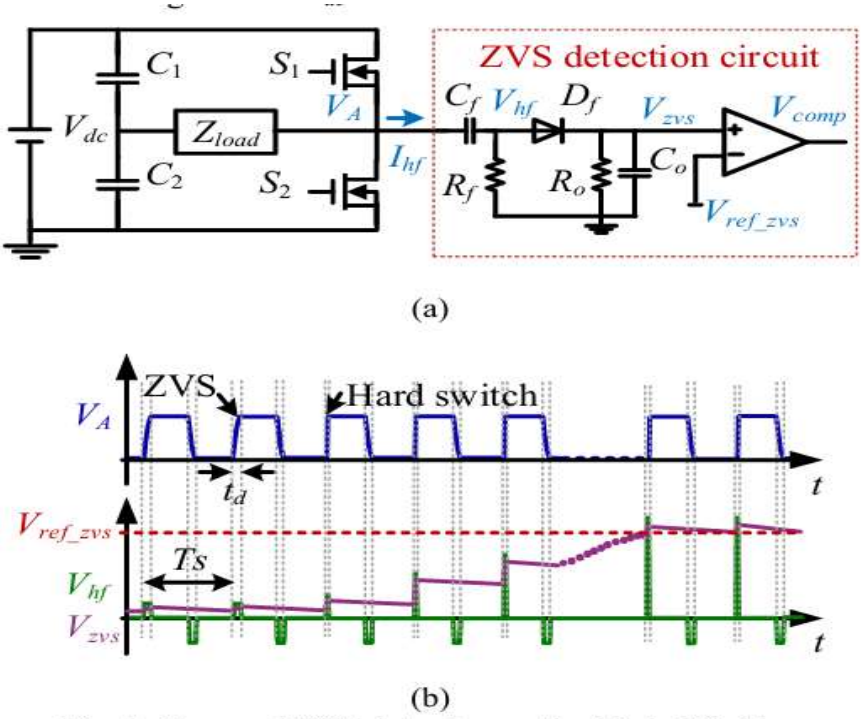


Fig. 1: Proposed ZVS detection method in half-bridge

What was actually proposed in the paper. Where is the difference arising? - 1) I think it is because of presence of a Z_Load factor which is not in this case.

The actual working was - the rate of change of Va will be recorded as a current that will be passing thru the high pass filter and to theground - which will unleash a barrage of voltage.

Till V Hf will also start following it - that did not happen in our case.

We record a Vzvs of 0.33 V and a Vhf of 200V - I mean yes it saves me the complications of letting my comparator suffer high voltages because of it - though you get the idea which we are conveying.

It is that the behaviour is totally unexpected.

Even the Vsi being square wave depictst tha teh Voltage V*i itself is going to 200 and therefore the capacitor itself is not getting charged.

After further experimentation it was found that the entire voltage drop was occuring due to the high current of the diode and because of the on resistance - most of the drop was taking place i the on resistance.

Because of the insane amount of current within the circuit - I_hf was 2*10^9 amperes - I have to change that

I have to keep the on resistance of the diode as 10^-10 to get some changes visible within the circuit.

Reason for such high current - Maybe it is because of the square pulsed nature of Vs1 - It is also partially because or maybe entirely because of the fact that the capacitor itself is not getting charged.

Which is causing more problems - as the current is basically a dirac delta(is what it is).

Yep definitely something wrong with the circuit - the current is 10^16 amperes.
Haha.

Maybe both the switches are on at the same time? Maybe something else IDK? Let us iterate and figure out.

Have taken a scope of S1 and then Vs1 , S2 and then Vs2 to compare what is actually transpiring within the circuit.

Note: I have a dead time of 2 ns given to both of them - In form of turn on Delay.
Which is if My duty is 0.5 it will remain 0.5 - par I say ki 0.5 - is 10 ns – and I say 2 ns delay - toh it will try to start at 0 ns to 10 ns - par with 2 ns delay - it will be from 2ns to 10 ns itself.

I have added and compared them - (ELL201) does help in this logical construction
THis is the diagram

Such Insane currents could very well be due to 1) Hard switching (I have no idea how those waveforms look like)
Definitely something else happening in those 2 ns - maybe both actually stay on for atime.
Or maybe **hard switching current levels are soo high - but hard switching is more of loss oriented not increasing the currents to 1/(planck constant) .**

Maybe they are on for some time - Have to test this hypothesis

2) We also have not added the regulatory Inductor and could very well be the reason for the behaviour of the circuit being the way it is right now.
As this is reminding me of **resonant pole inverter**

2ns acts as my delay and the dead time between the switches 3) I could increase the dead time and I think verify the hard switching wale claims that I carry.

Yes the 2ns will play a role - I increased the dead time to 200ns and apart from shifting of waveforms and all. There is no change in the final values of the waveforms or the way they look - Yes it will play a part - Though I do not think it is playing a part very well right now in the circuit.

In our case there is 1 more caveat - it is that the voltage across RC is following Square wave which can only happen if the capacitor itself is not getting charged.

And the current is going thru the resistor -that could very well also tell me why the current is so high - that resistance in itself is very small and could be the cause of such high current values.

Another hypo - If changing the resistor value - does something to whatever **current - that could tell me why the circuit is behaving the way it is.**

Yep - This is most definitely true- Most of the current is going thru the resistor - I reduced the magnitude of the resistor and the values changed. LUCKILY - ALL OF THEM!!!

Though the solver was becoming stiff for some intermediary resistance values - **This is something I have to figure.**

New tasks

1) **Why solver stiffness for intermediary values**
2) **How can I make sure that my capacitance is the unit that is getting charged**
3) **I have not added the load or the inductance if I call it properly - how will things change when I add the inductance into the system. Would it help with 2 ?**
4) **How does hard switching really look like -** Then I guess we would be able to proceed with Resonant pole inverter mein changes I guess.

I did remove the line connecting them - basically Zload as infinity - and yes the capacitor is getting charged in this case - **which does mean - that whatever is in my zload - does influence the rest of my circuit !**

It was obvious - but how it influences is something we need to understand.

And why the behaviour was the way it was if the zload was a short circuit line is also something I have to understand - Task number 5

After debugging and talking to a senior of mine - the conclusion came to be - The Zload plays an important factor in all of this

The solutions and the simulation files definitely show this

Realise ki jab hum losses dekh rahe hai toh actually kisme dekh rahe hai

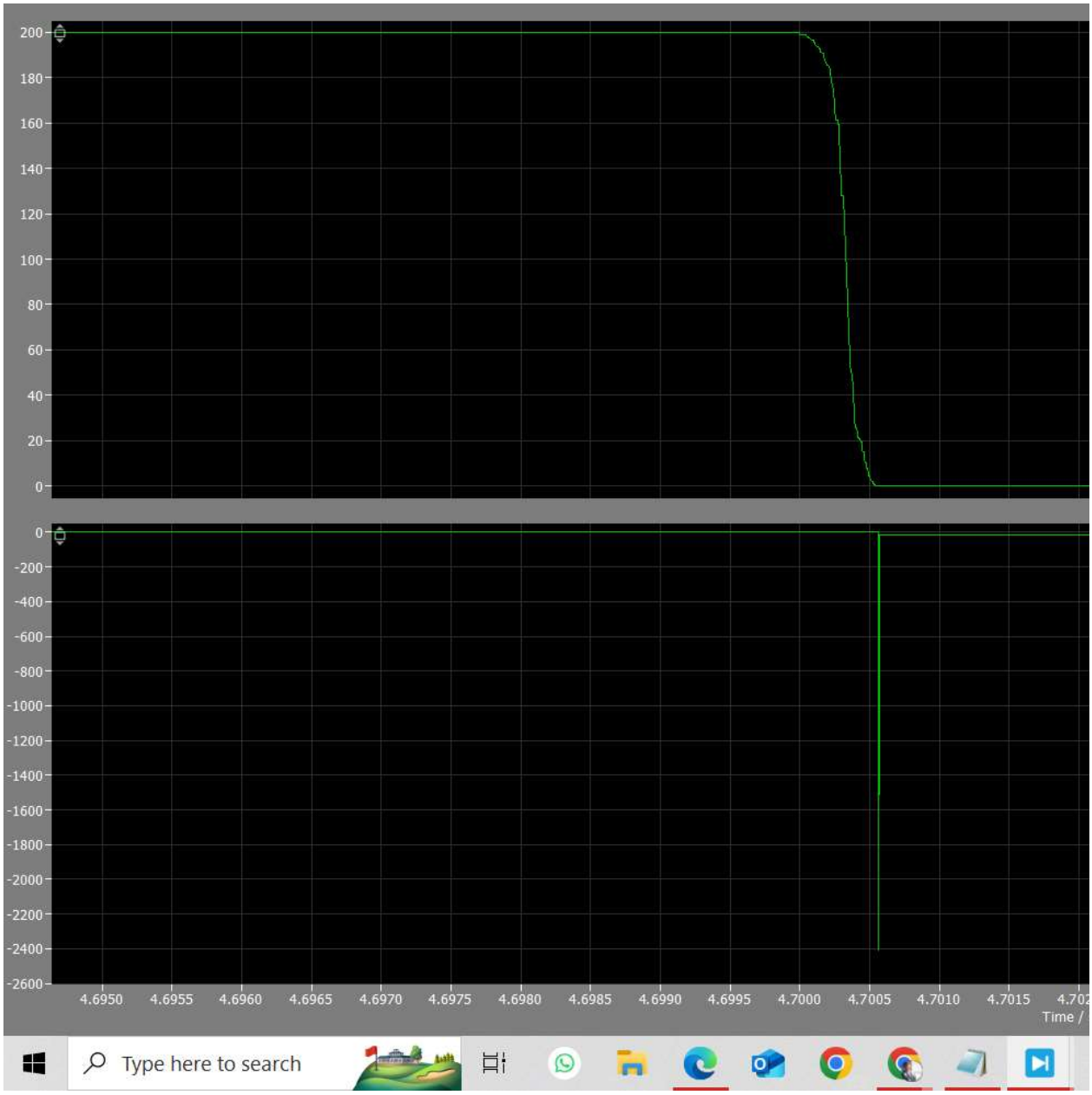
Toh the bright idea is that we have S1 and S2 and to perform hard switching what I want is to start the switch after the voltage across it is zero else the voltage will drop too fast and the losses will be immense within the circuit.

So If My S1 is on and s2 is off - then s2 has got the gyat. Therefore I will be looking at Turn on Losses of S2 when I am switching off s1 .

Because s1 off - both off - voltage shud inverse - then s2 on - Toh isme we are looking at the losses which s2 will be incurring to us.

Similarly when s2 is on and s1 is off - then we will have to turn s1 on - therefore we will be looking at the losses given to us by s1.

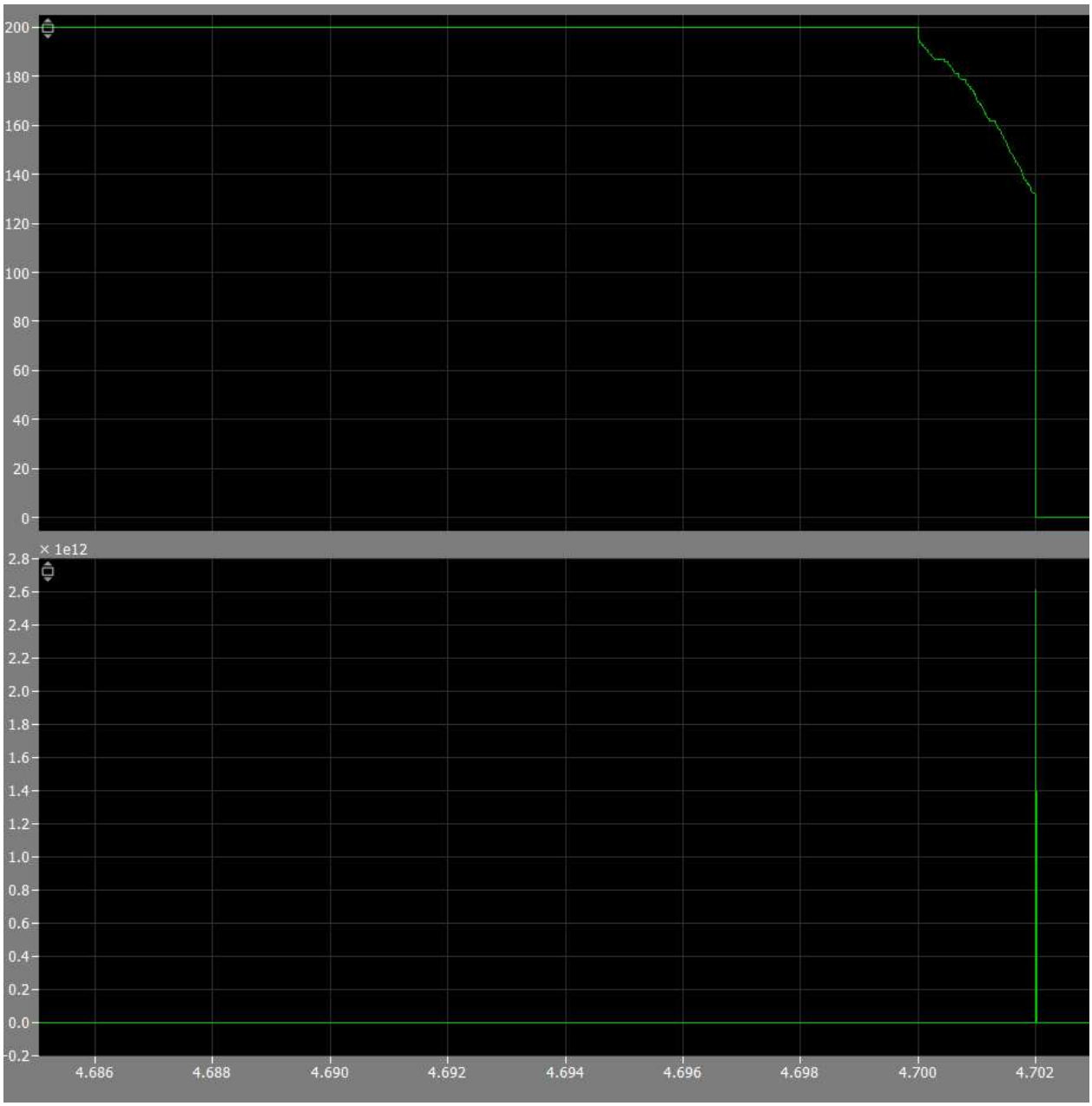
Basically joh bhi uss waqt on hai - uske complementary switch mein losses dekhne hai.



In my case I am seeing something weird - which is when S1is losing voltage = the switch is turning on.

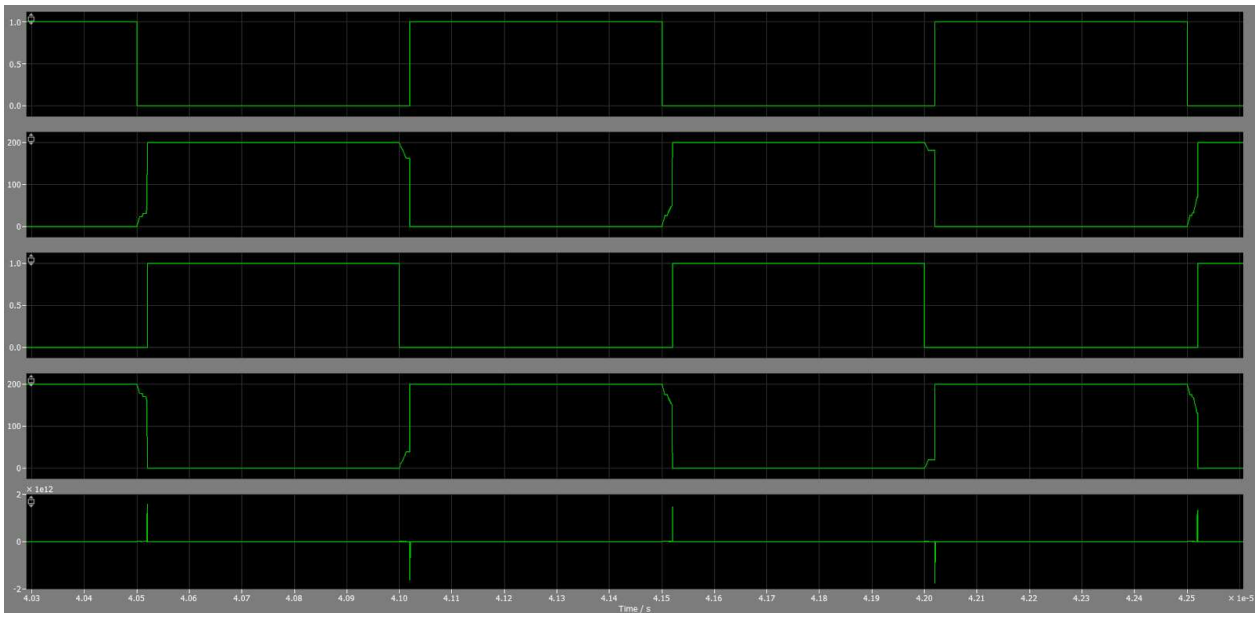
The current shows a very big spike after the voltage has become zero.

l **Increased the value of the inductor from 2 uH to 10 uH -**



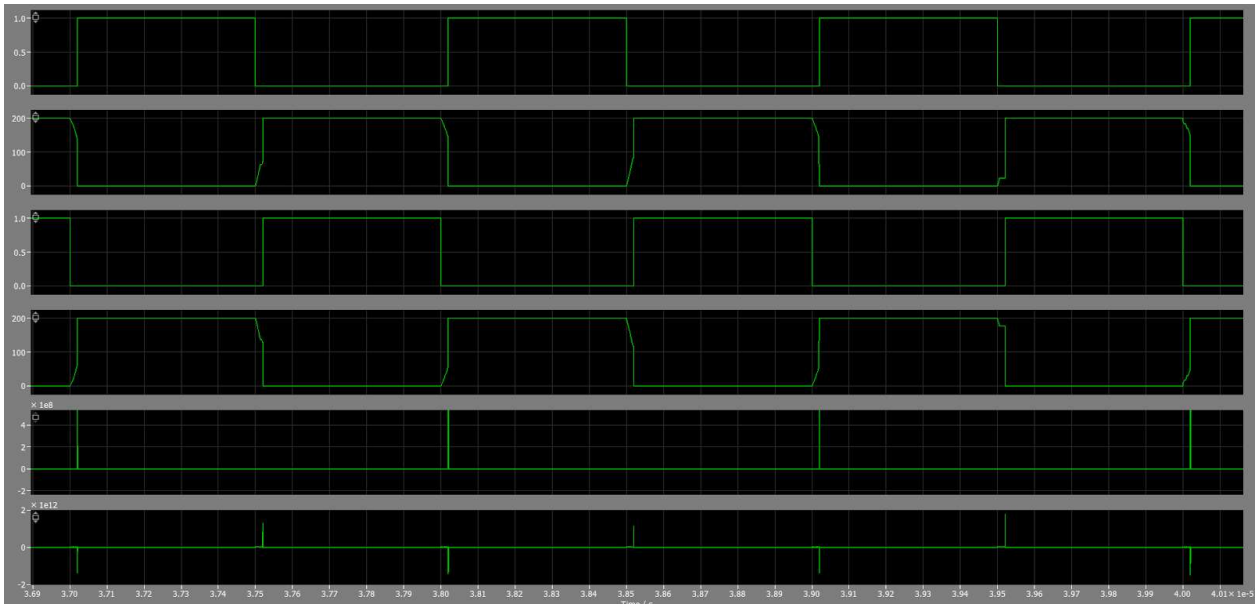
The voltage and the current are intersecting as you can see - though the simulation itself is not explicitly showing this - there are some other factors that are coming into play here.

1) How does the value of Zload itself matter - because in our case that is currently an inductor - so increasing the value of inductance should technically also change some other things like the increase and decrease in current will be much gradual



For 10 uH this is how it looks like .

S1 - Vs1 - S2 - Vs2 then current in the parasitic of switch 1



(top)S1 - Vs1 - S2 - Vs2 - Then current in high pass filter cap then current in the parasitic of switch 1(bottom)

The Behaviour of the high pass filter does make sense - as mentioned in the paper - the value is getting supplied to it by the load value itself.

Another issue is - The magnitude of current observed and the solver is facing issues because the rise of voltage it says is v = ldi/dt or i = cdv/dt and the voltage rise it gives sometimes is far too sudden which does not make sense.

It will not break the nature with which it is ascending immediately. What is causing this issue?
This is visible when you compare Vs2 and the Amm_high current that the current is so large that it should not be so large.

The rise of the voltage itself is very artificial

The current relative tolerance of the simulator is e-4 - it is possible that it is causing a possible issue
Having seen how suprem sometimes does things - i wont be surprised by it

I have reduced the relative tolerance to e-5

The simulation behaves much much better than before - much smoother.

The values or atleast certain values look much more believable than before

The current is discharging not via the resistor but via the capacitor.

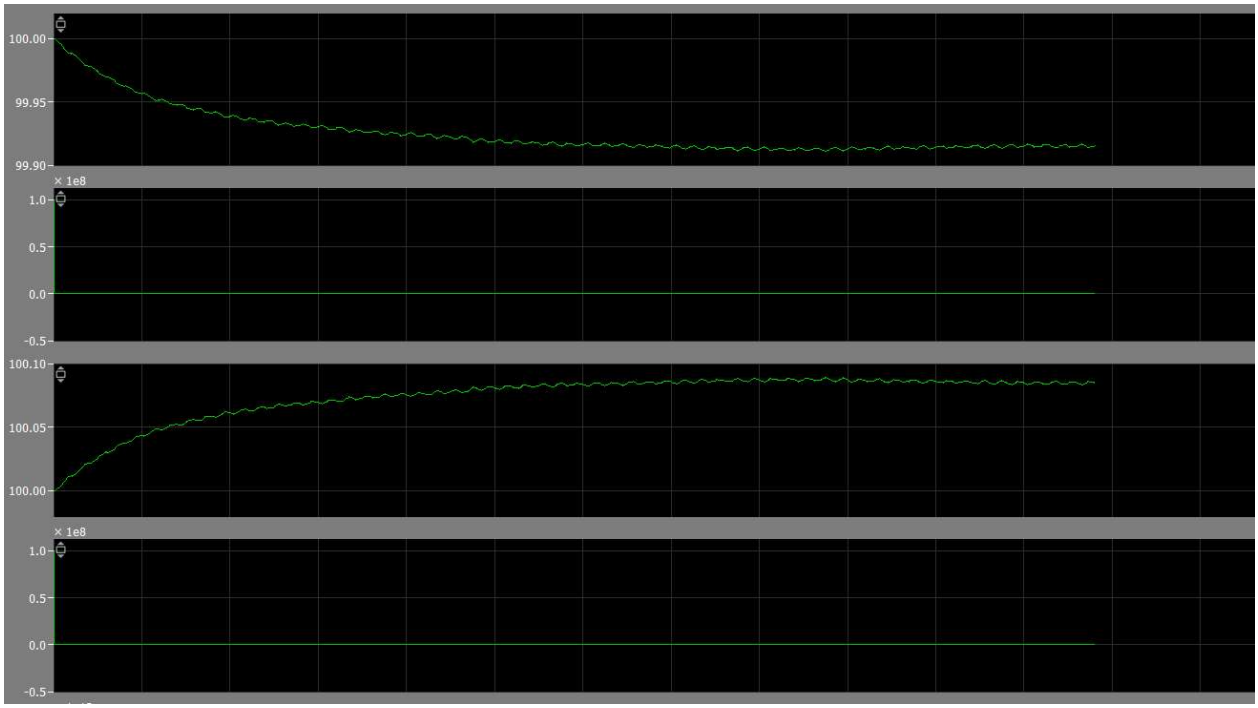
1) **We still have to figure out why it is discharging the way it is discharging**
2) **The current issue is still persistent**

The high pass was producing current values of 2 A etc

The parasitic was producing current values of 1000A etc - These values themselves are farr to high.

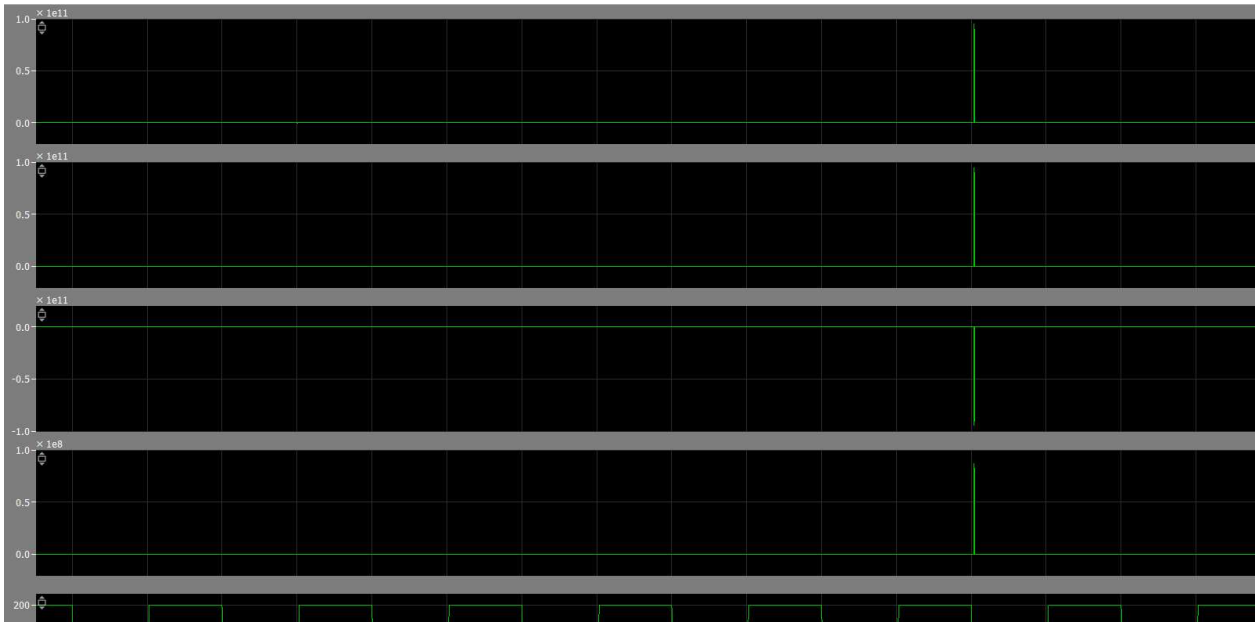
1)Have to figure out the implications of such high currents themselves -because the parasitic resistance is doing the job and the capacitor as well.
\

The way I think the simulation currently works -



I dont know why this small natural bias exists within the circuit - It does - I want to know why..

But on the bright side it does mean that it is working the way I want it to



The simulation also shows a random peak in the middle of nowhere at 36 us.

HTis is all the Amm_high and current and IG graph.

Have written in the book the current implications of the circuit.

What I now wish to understand although there are many things that still need to be rectifieWhat will happen If I change the loads

1) What will happen if I change the dead time

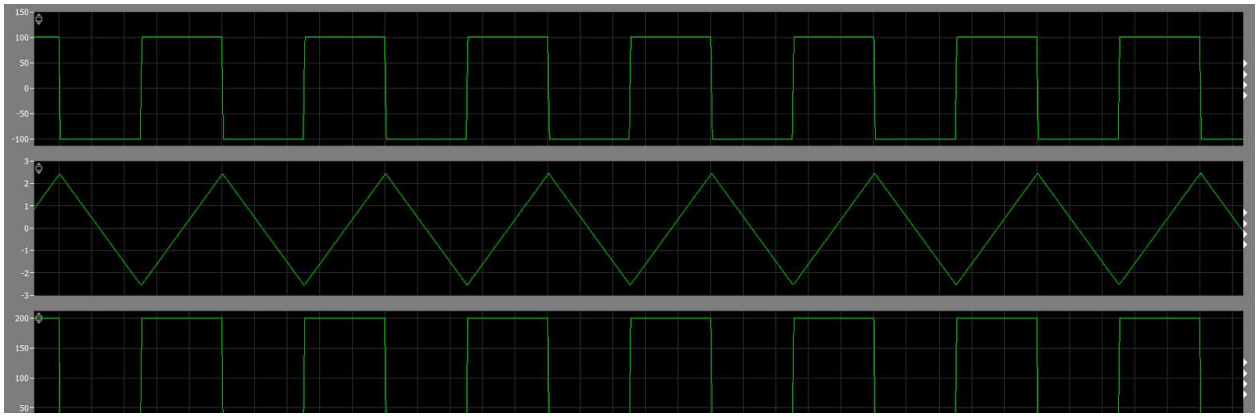
These might give me more information on the circuit and the way it is operating .

d within the simulation itself -

2) What will happen if I further reduce the relative tolerance

Changing the relative tolerance has definitely changed the nature of the simulation

From e-5 to e-6 it is



This is the load ka voltage and current values that we are witnessing right now.

Something I noticed right about now is that the rate at which Va decreases and the rate at which Va increases are not the same.

The rate at which it decreases is almost 2 times the rate at which it increases which is very surprising.

Why this happens is a mystery we will have to solve.

In our case the rough value of V_zvs that we are getting is 0.27 V which is also what the steady state predicts. That bit is definitely correct.

The simulation works wonders and perfectly -

The only thing that is left to test is the dead time and usme variation

We shall test it under different conditions - 1) When it is soft switching
2) When it is hard switching

We have already tested it under soft switching by default - we tested it with a dead time of 20 ns - the max it required was is 12.9 ns or something

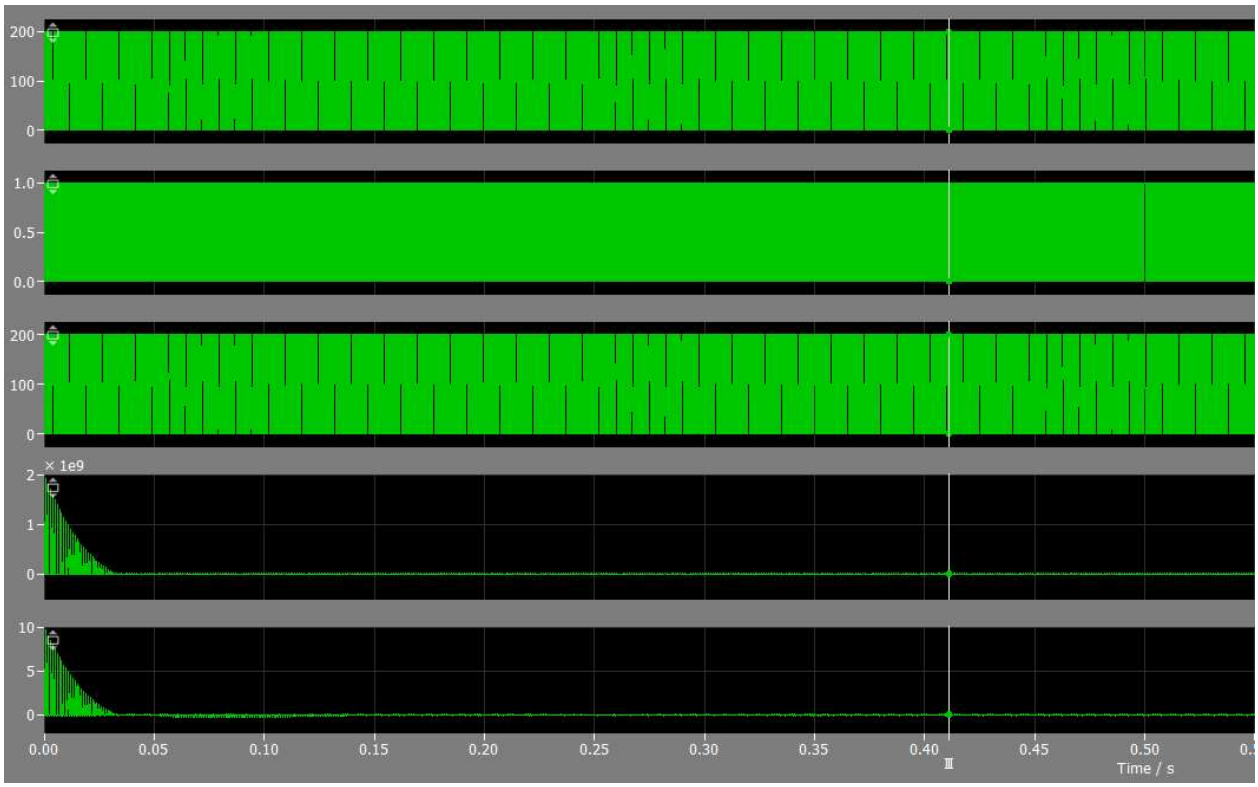
The least it required was 8ns.

I am unable to get the simulation running if I set the dead time below 8ns or equal to it - **I think it likes the very idea of operating under steady state condition**

We need it

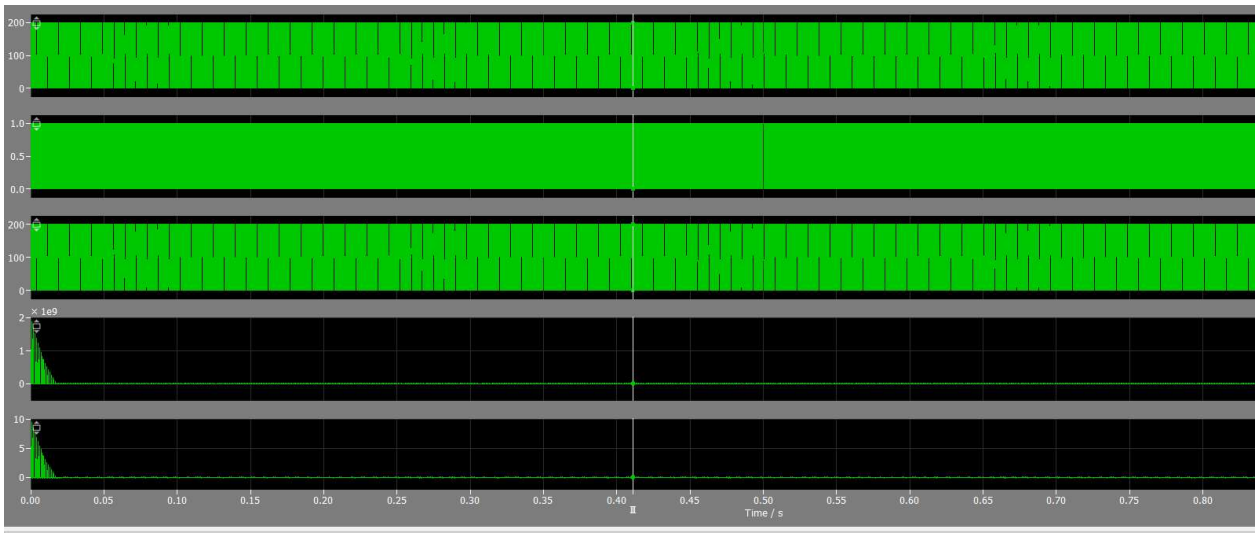
When I keep the deadtime as 8.9ns

This is the overall gist of the current that I see

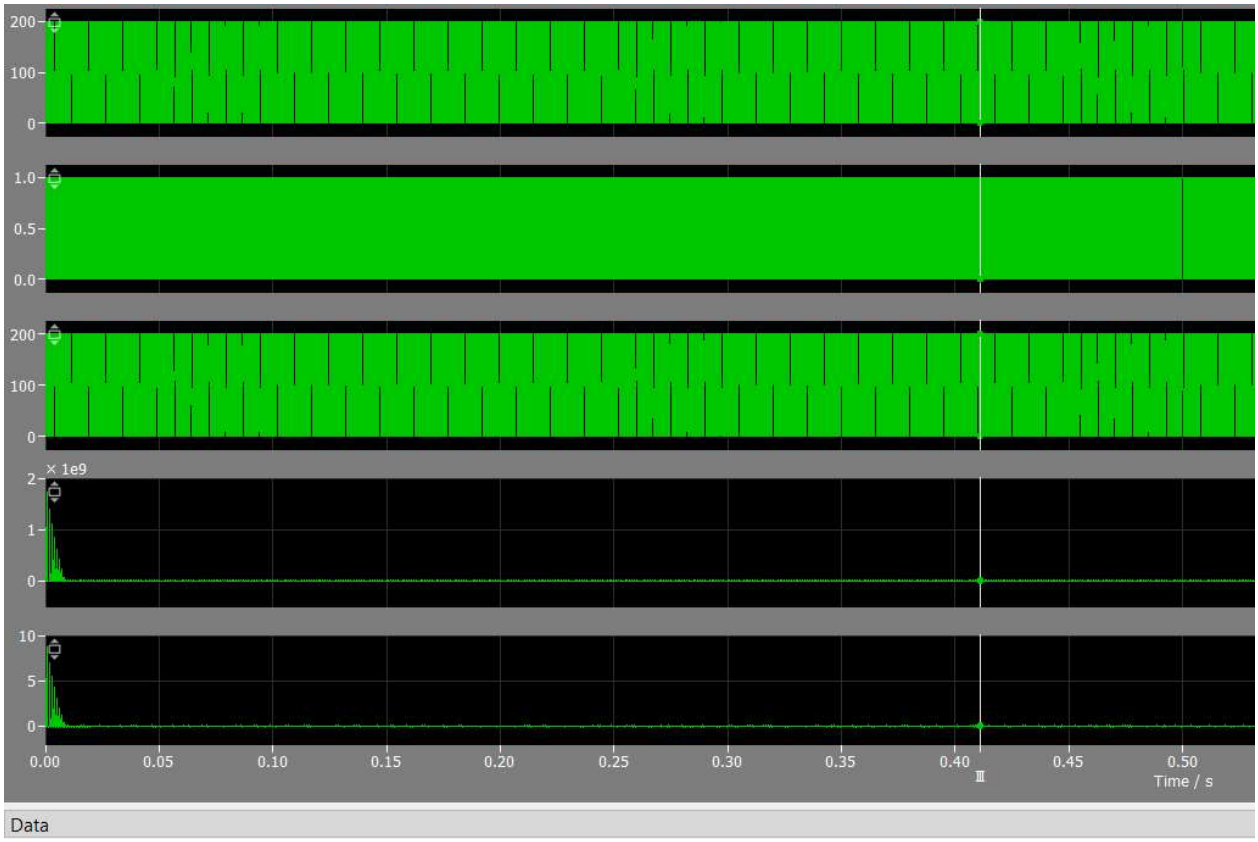


They are in milli seconds

When I keep it as 11ns

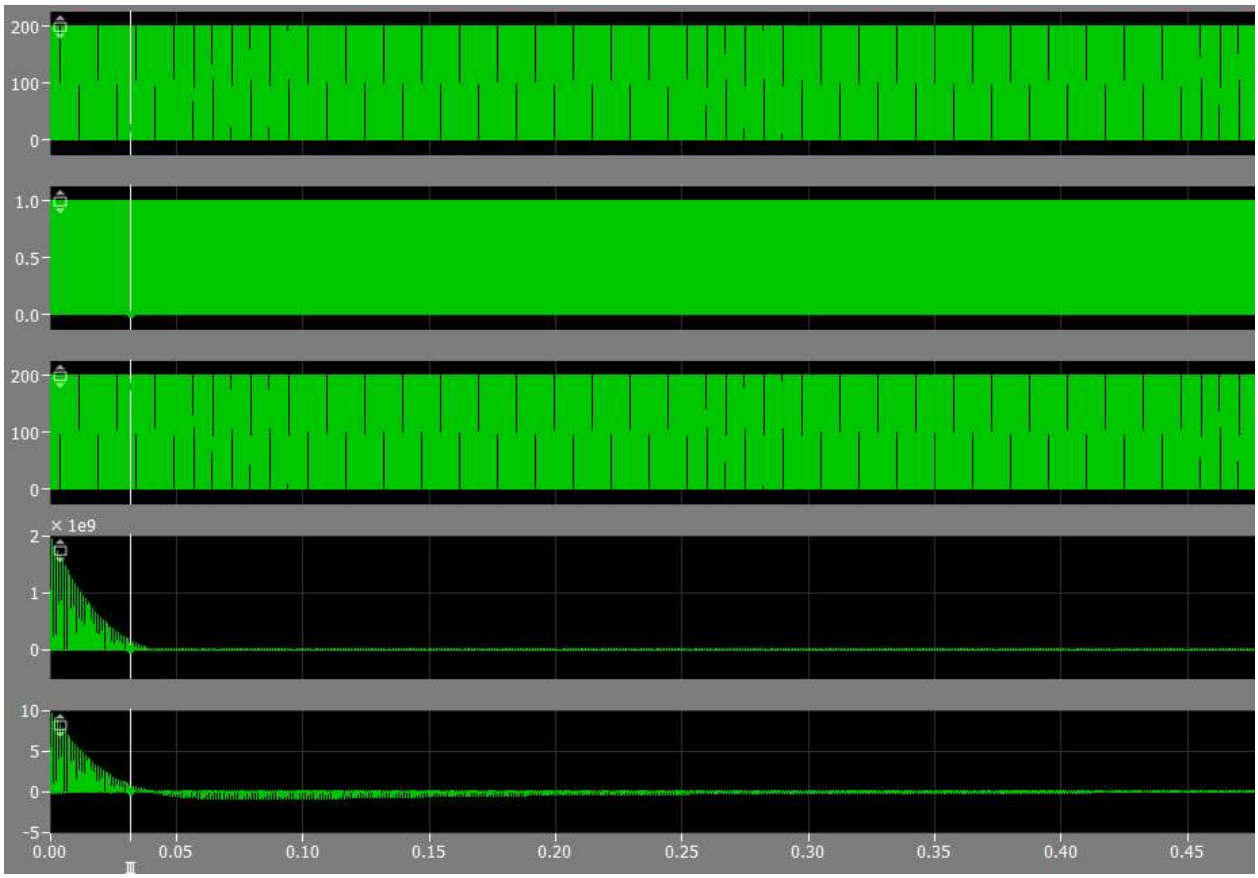


The decrease is evident



At 15 ns

When I do 8.4 ns(the simulation throws an error If I try to go below it)

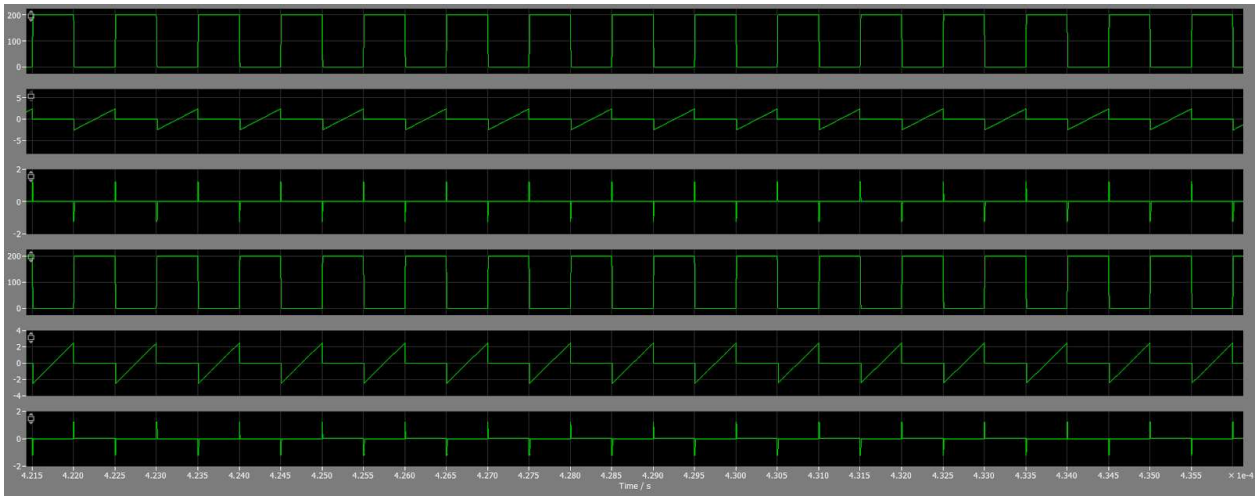


The nature of the graph is self evident.

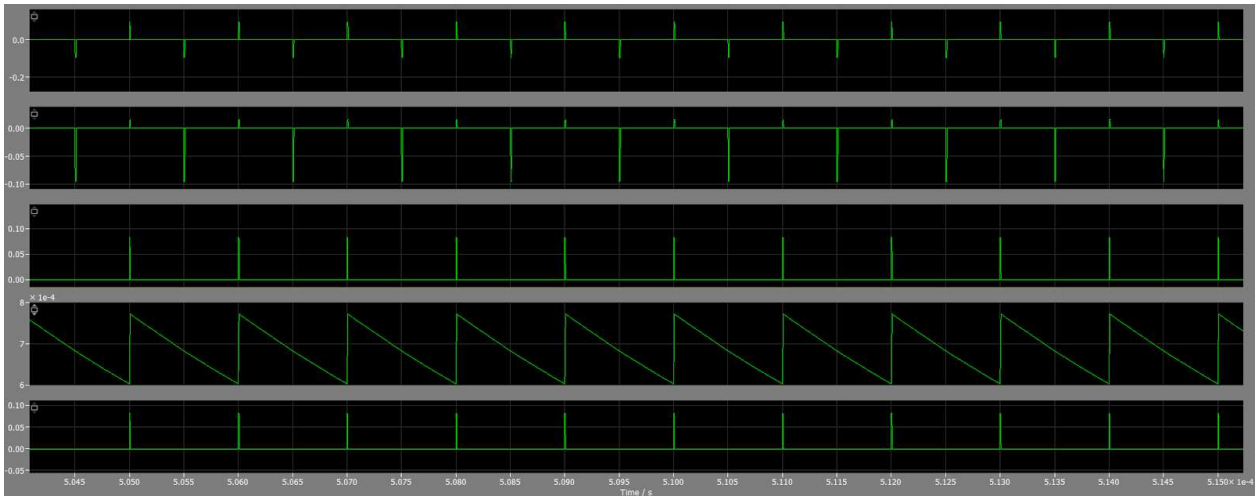
We now have to figure out a solution to this hard switching simulations not working.

Though the graph is still pretty misleading if you do not properly analyse it at each dead time.

So at 12 ns



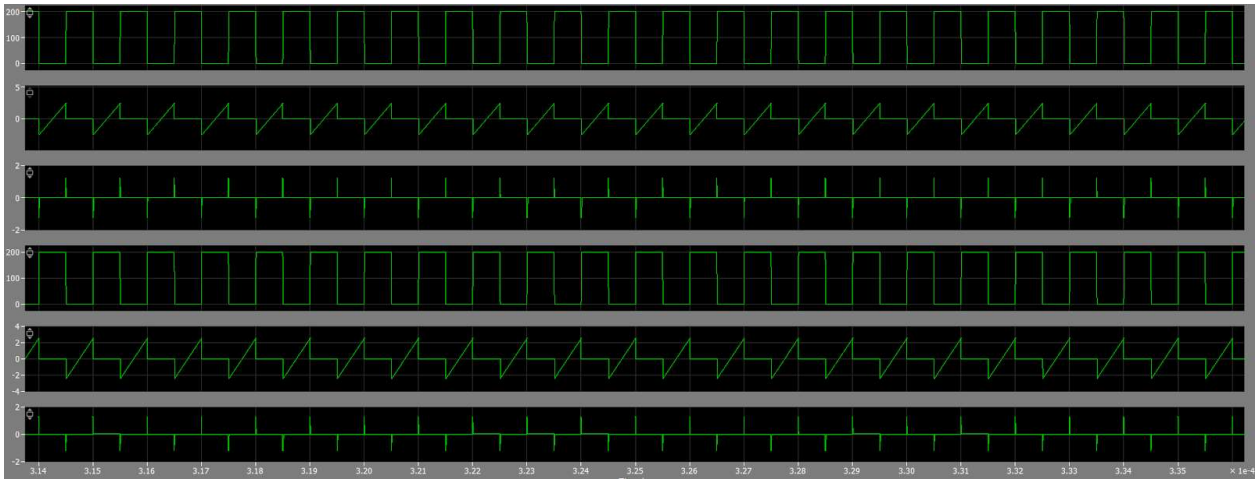
This is how the switch voltage , current and parasitic currents look like at that dead time



This is how Amm_high , I_hf , diode curr and others look like - though The magnitude of Ihf looks small surprisingly

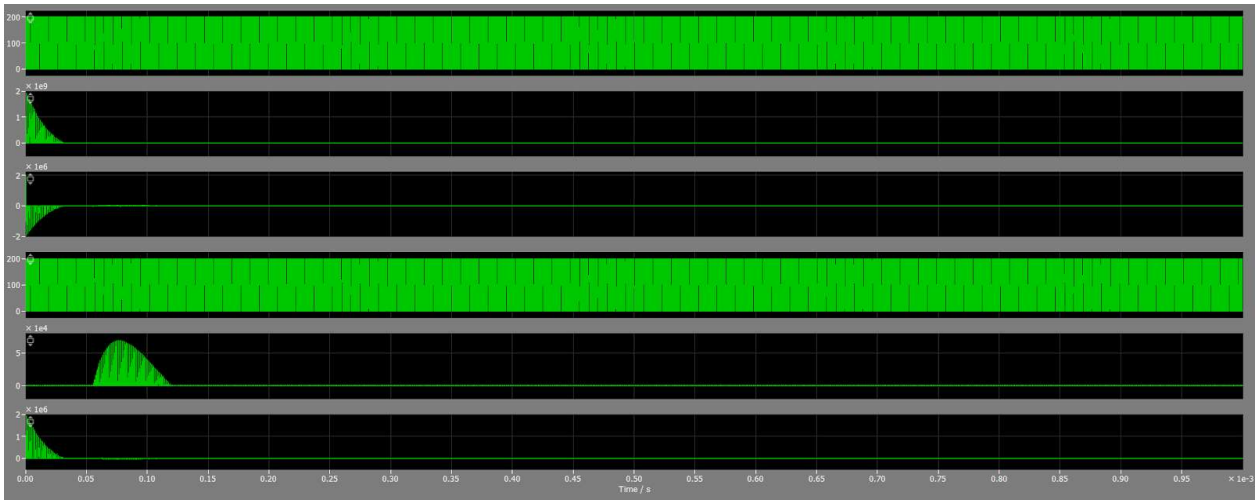
Running the mathematics you do realise that it should be running that way - You do see it - that in the reverse direction the magnitude is -0.1A which ideally should be when you run the mathematics of it.

If I now change the dead time to 9ns



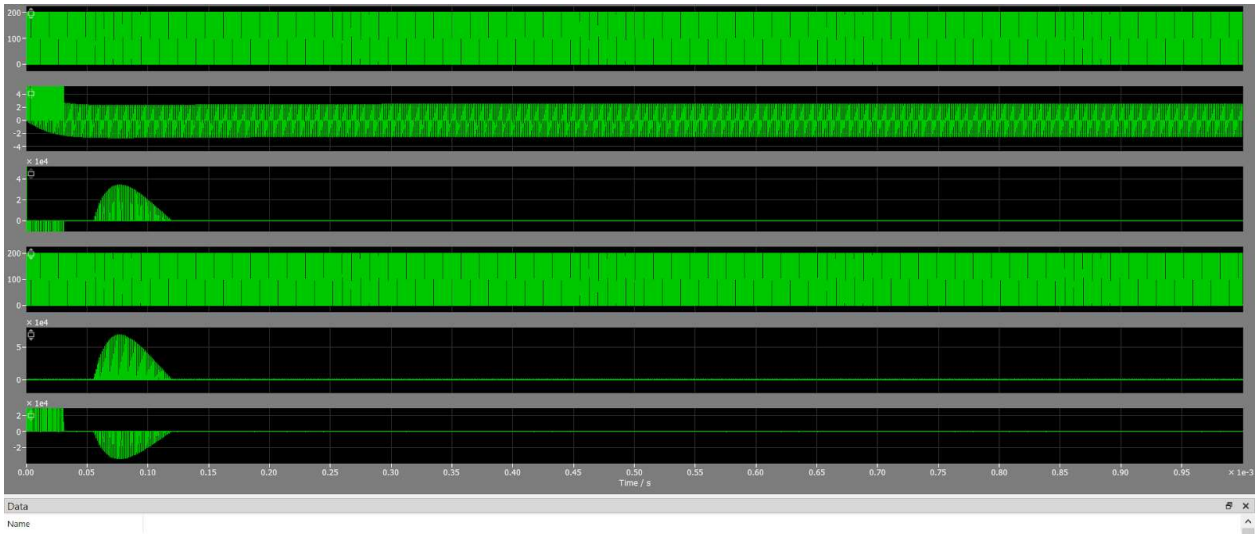
This would again be at steady state itself - when we put or see the switch voltage , current and parasitic current.

The transient behaviour of it has changed its nature



This is the overall nature of the current which we see when we have 9ns dead time

The increase in the middle is unexpected



A better view of the problem at hand

This was occurring at the positive increase itself and not at the decrease.

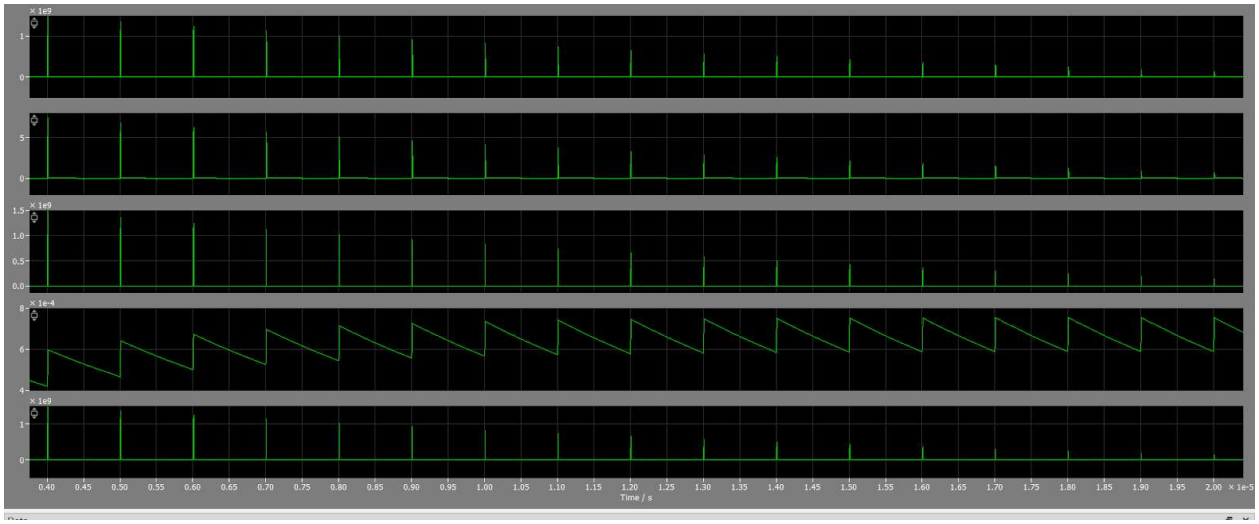
It is entirely possible that the circuit was hard switching at that interval and was not at some other interval

That is a possibility.

Another fact is that - now running the simulation at 20ns - the values of the deadtime which the simulation itself is showing is different.

Therefore

1) **You need to explore this hard switching circuitry wala fact and - also explore maybe multiple times the different values of deadtime that you are achieving as a result of it.**



You can see that the magnitude of the current going to the resistor of high pass filter is very high - the 2nd plot from above - **this is the current that the amm_high is seeing during transient when the dead time is less than the time**

It is during hard switching - what is really transpiring within the circuit is what this tells us about.

5A is very High and we can see that the

Amm_high also falls abruptly when the circuits starts to soft switch within the limits. To roughly 0.1A which it at steady state should.

Even Vhf reduced from its initial high value to 0.3V which is what it serves

At 10 ns I was seeing no secondary curves because the 0 to 200 part starts high and then falls

And the decrease part starts small - caps out at roughly 9ns and then falls to 8ns.

So what I should be seeing when I decrease these things further is that there will be a curve like what I saw 2 graphs before.

Because then hard switching will be happening when Va is decreasing and not increasing.

Thus is the reason for the graph being like this

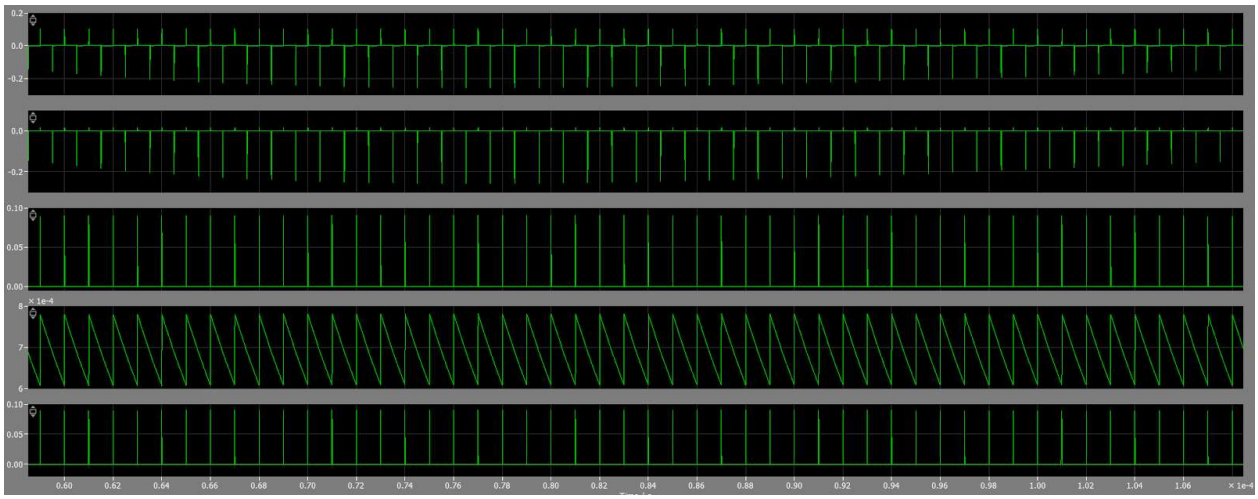
1) **Try this for different values of Dead time and see what values you are getting and the effect on V_hf and possibly V_zvs.**
2) As we are to design it for a general load - replace the L with a LCR - try different values and then try to get the things done - Run some theoretical calculations as well -

3)Get the simulation working when the circuit is only hard switching and not soft switching.

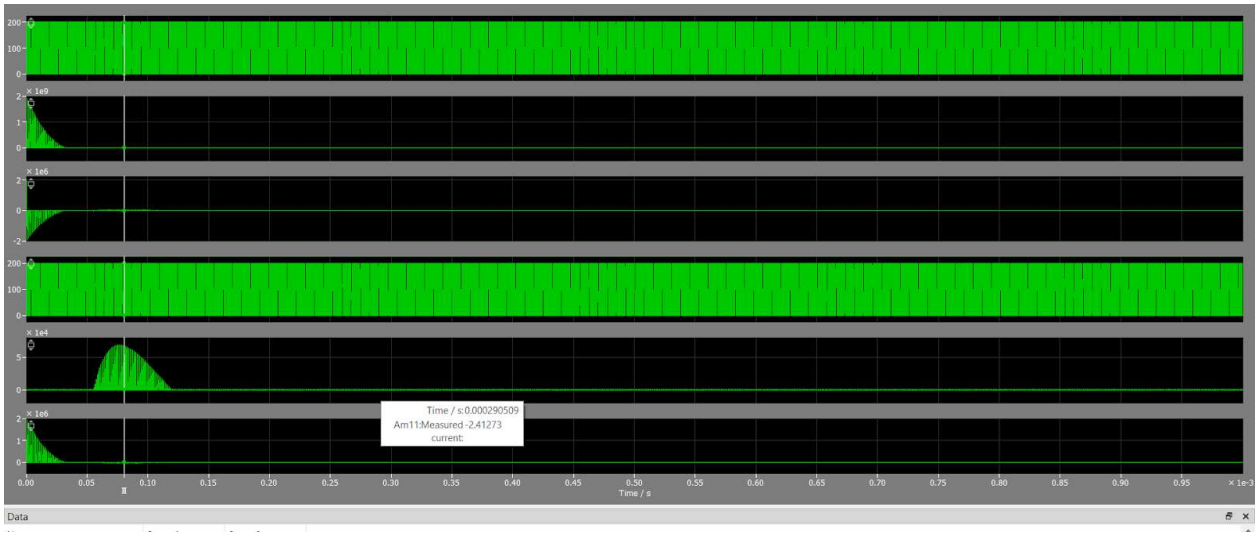
4) Can we get a baseline prediction to what the value of the current will be when I am hard switching - as to what values I should be seeing?

You see that at 9.2ns Dead time - there is no response from decrease to increase part

But there is at 9.1ns - so that is the maximum we can get .



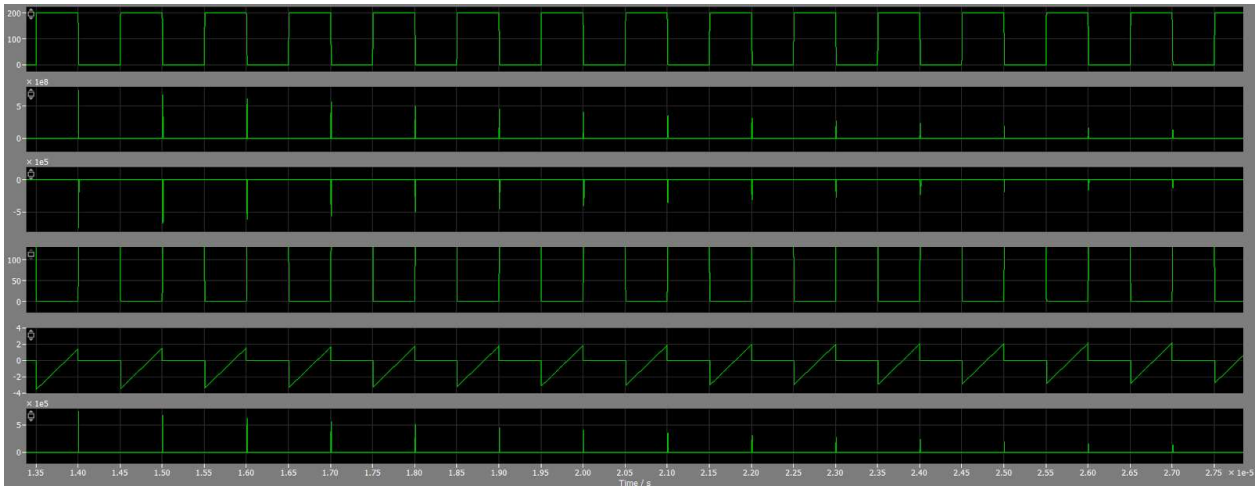
At 9ns - these first 2 plots from the top are the - Amm_high and I_hf which makes sense



When I am hard switching it is .

The magnitude of current in switch 2 is e^4 order- **The issue is- theeke - the current should be high and all - what metric do I have - baseline to judge whether it is true or not.**

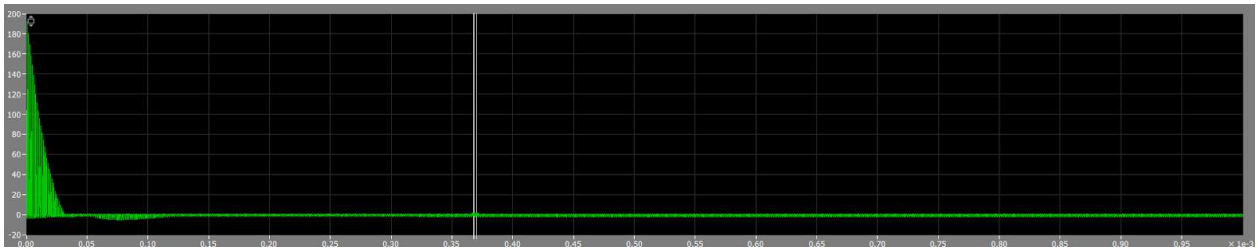
Everything behaves as expected - it is just that the metric itself is lacking



This is the graph during the initial hard switching - As predicted the current in the switch will be much when it is turn off to ON transition cause then only can it conduct.

Though because the voltage takes a jump - **The current which we witness in the parasitic capacitors is of high magnitude as well.**
That makes sense.

I am not commenting on the magnitude yet - only on the behaviour - as I cannot comment on the magnitude



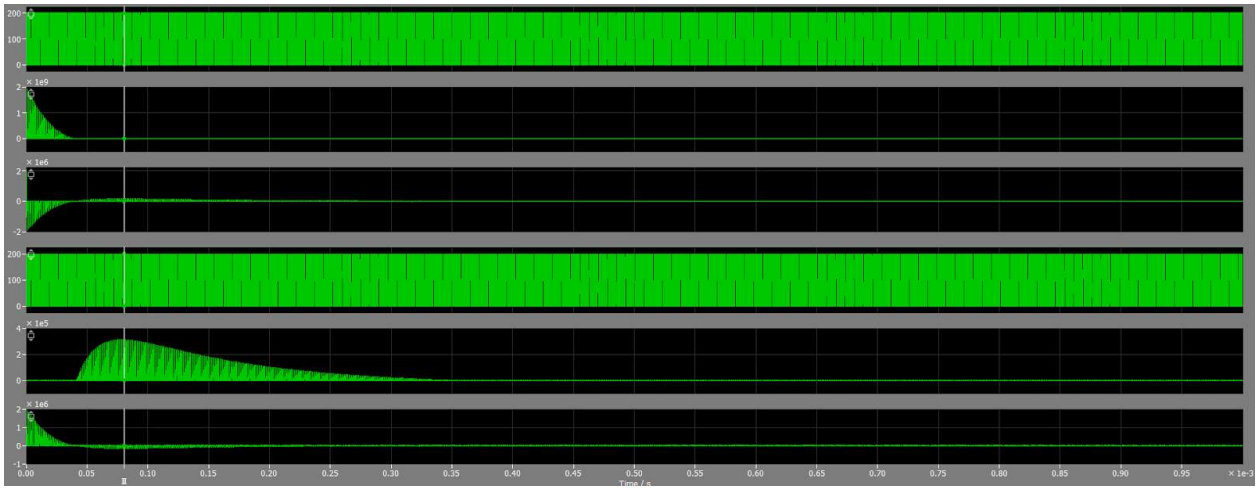
This is the graph of Vhf.

Ye the-ve voltage seems tad bit too less - it is because the simulation says the reverse current is of similarly less magnitude.

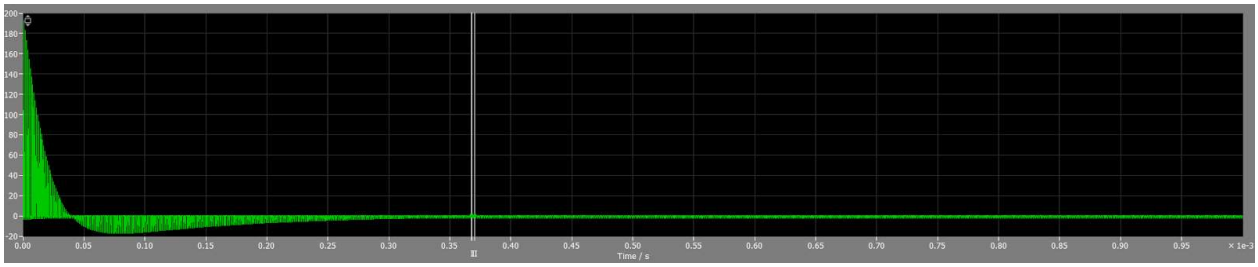
The order of the current in switch 2 during hard switching is of e^4 magnitude.

But in the other switches it of much much higher magnitude.

AT 8.5 ns delay what we notice is



The order of magnitude did change from power 4 to power 5



The effects of which can be seen in V_hf.

There is no lasting change in V_zvs - so given the current dynamics we wont be able to tell anything.

We are unable to simulate the circuit below a certain threshold - Have to cross it.

I changed the load from 10e-6 to 5 e-6 - the di/dt doubles and you do see that - the current is doubled in almost everything.

Except I_hf which is the current in the resistor of the high pass because the steady state voltage value that we are achieving with only an inductive change is that the td time reduced and the Amm_high current increases which effectively cancel each other out.

The theory is all set - we need to see and model the behaviour when we have an LCR circuit in tow.

I initially tried with an inductive value of 5e-6 and a capacitive value of 2e-7 - sort of to try it at resonant frequency itself.

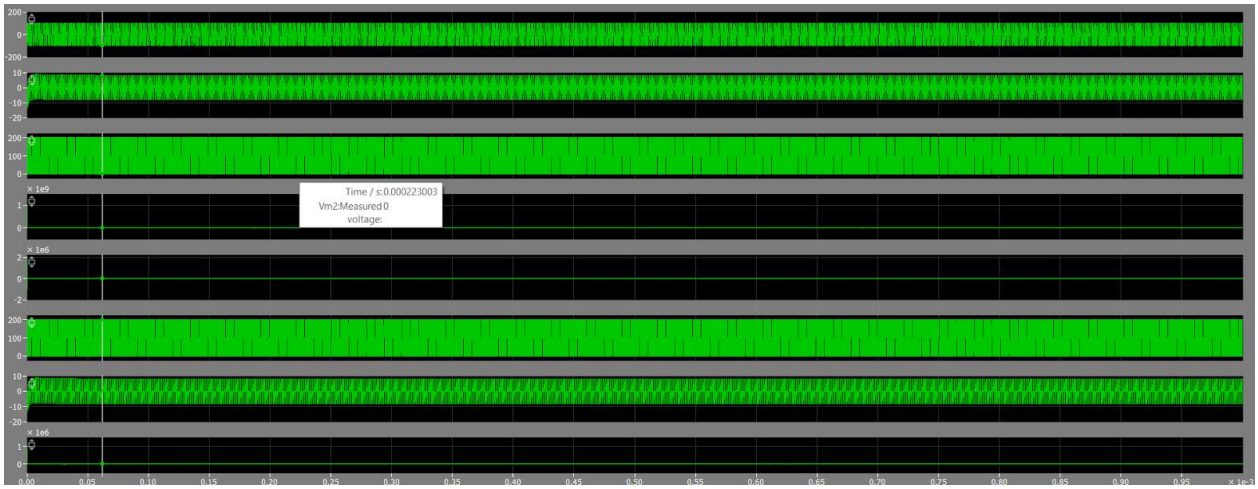
Then I tried it with an inductive value of 10e-6 and a capacitive value of 1e-7 and it did not work.

Though it works when I reduced the relative tolerance from 10^-7 to 10^-4

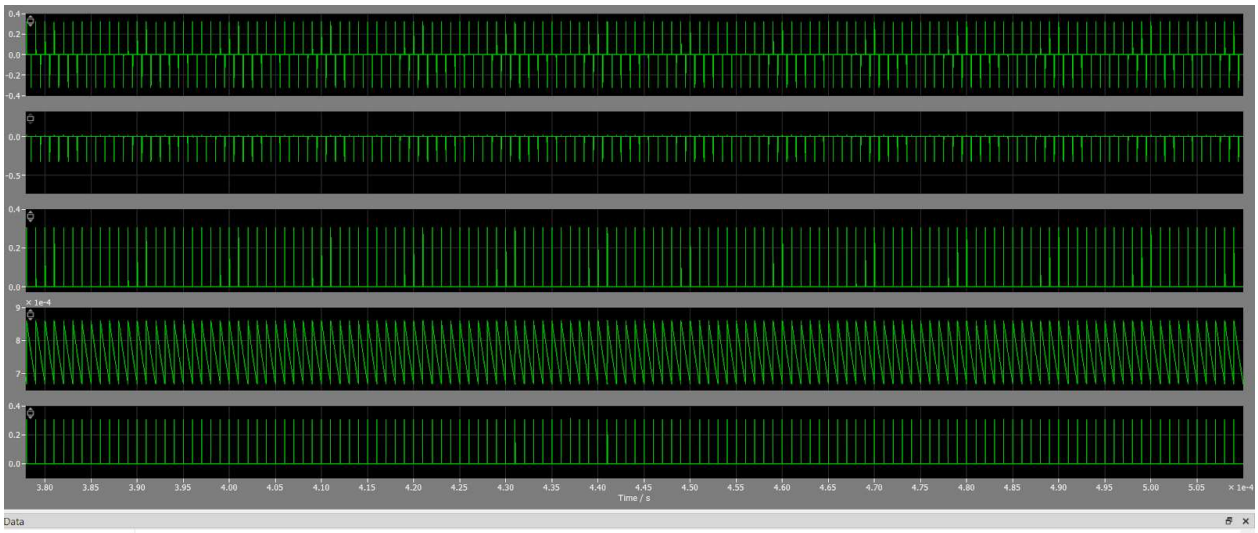
Why? I have no idea about it

I am varying the load - currently both the capacitor and the inductor are at 10u . I have varied the resistance values from 1u to 0.1 ohm - I have seen some difference but as IR is still small - the changes are barely noticeable.

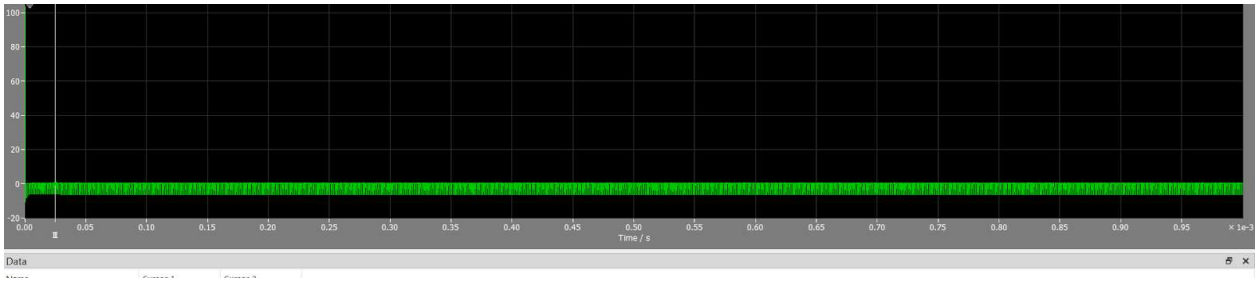
I have now changed the resistance to 10 ohms and I am witnessing a decent enough change that the previous simulation times are wrong -because the resistance drop has become significant enough and there are other considerations.



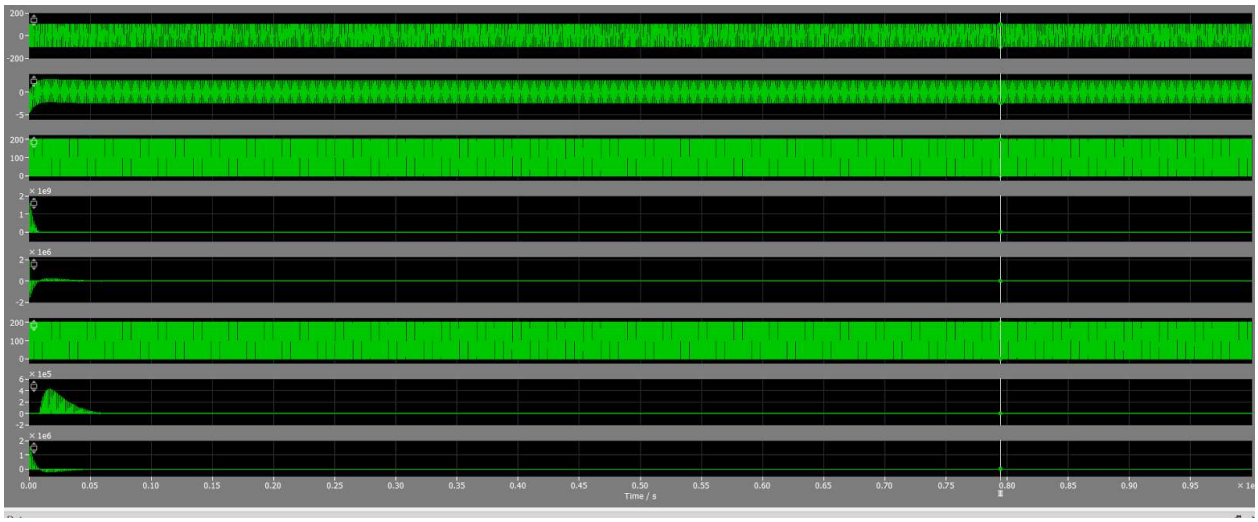
This is the graph we see with R=2 as the load resistance
Tht as you can see is V load , I load and Vs1 Is1 and so on



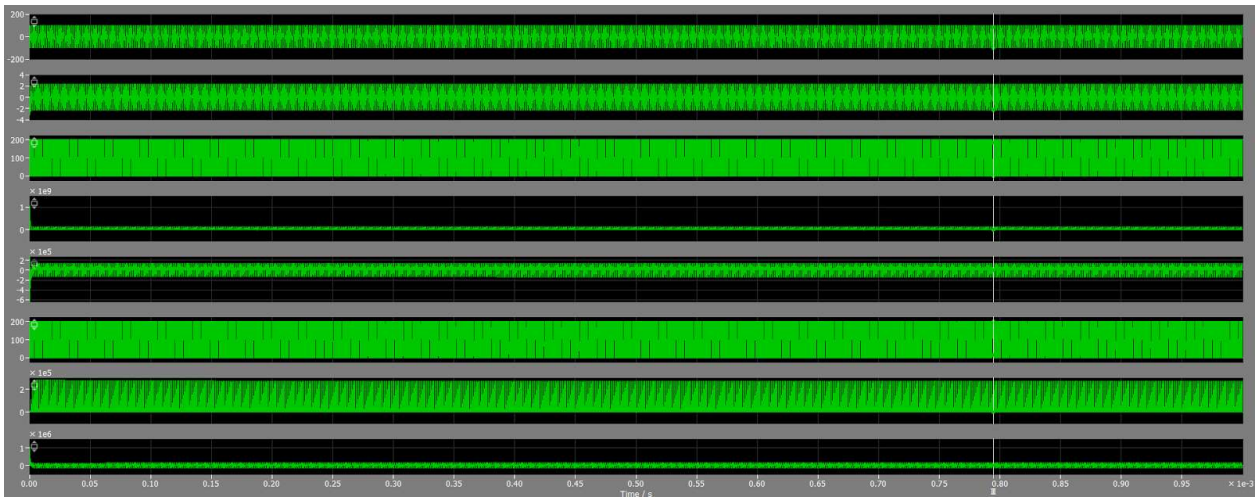
This is the amm high I hf graph and so om



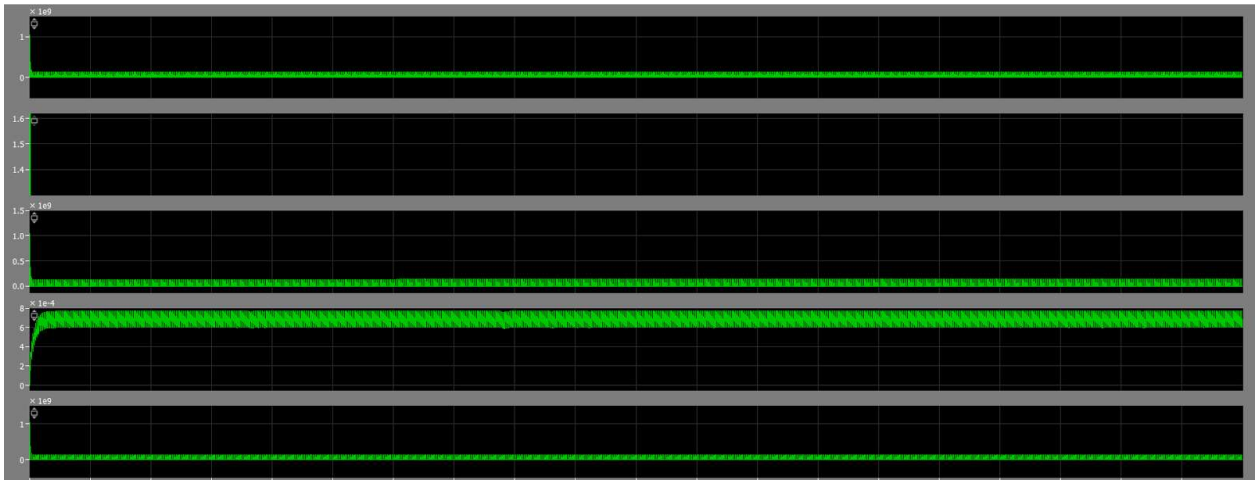
This is how the V_hf looks like



This is how the graph looks like when I am having L as 10e-6 instead of 3e-6 as before and resistance is 2 ohm

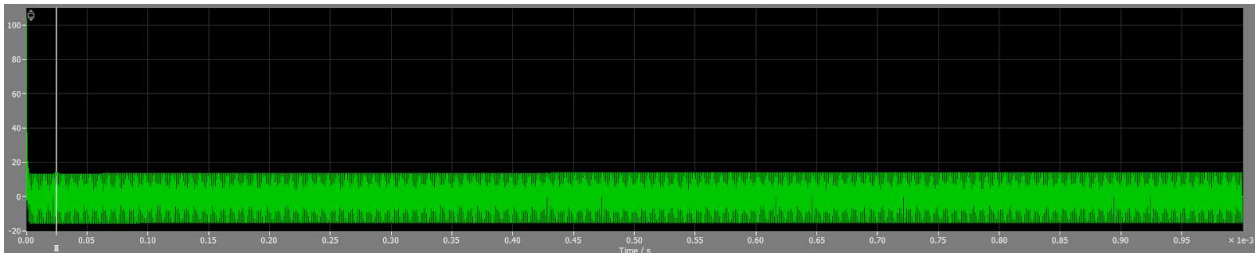


THis is with R=20 - which shows that there is consistently hard switching taking place atleast in the negative interval



Supported by the humongous currents that we witness with Amm High and others - of the order of e^9 which does not make sense but I think eventually will have to .

I mean a simulation cannot be wrong at such a stage of operation.



This is V_hf - goes from +20 to -20 and also that V_zvs remains unchanged from 0.31 V - it is an immovable force it seems

Changing ever so slightly which we cannot really use as a differentiator between hard or soft switching.

Changed certain things in the simulation - Added Rds on the mosfet - It was because the simulation was showing **That the capacitor voltage is taking a very sudden jump - which showed the lack of it.**

WE removed the parasitic capacitance and added an internal resistance of 10 m or 1m .

The simulation now works and it works in a different fashion from what it did previously.

It is able to simulate the circuit below the required dead time - it is entirely in the hard switching zone - which I never really expected it to work in. I have no idea why.

The simulation is much smoother than before as well - The dead times - the curves - the everything - they are far more smoother.

The issue was with the steady state values which we are obtaining now - they are different and they work wonders themselves.

Changing the R from 400 to 40000 changed the steady state values - and we are getting higher and higher voltage values when we are hard switching as compared to when we are soft switching - that is a good addition to the system.

The major change has been implemented - **You wont see much change now if you change the resistance values.**

The only thing left in this is to figure out is - how the inductive loads or the type of loads we want to cater will vary.

How the current waveforms for different resistance values look like and why

1)Variations in dead time as well as resistor values and what they look like

Depending on things we might have to also add a saturator and an AND gate.

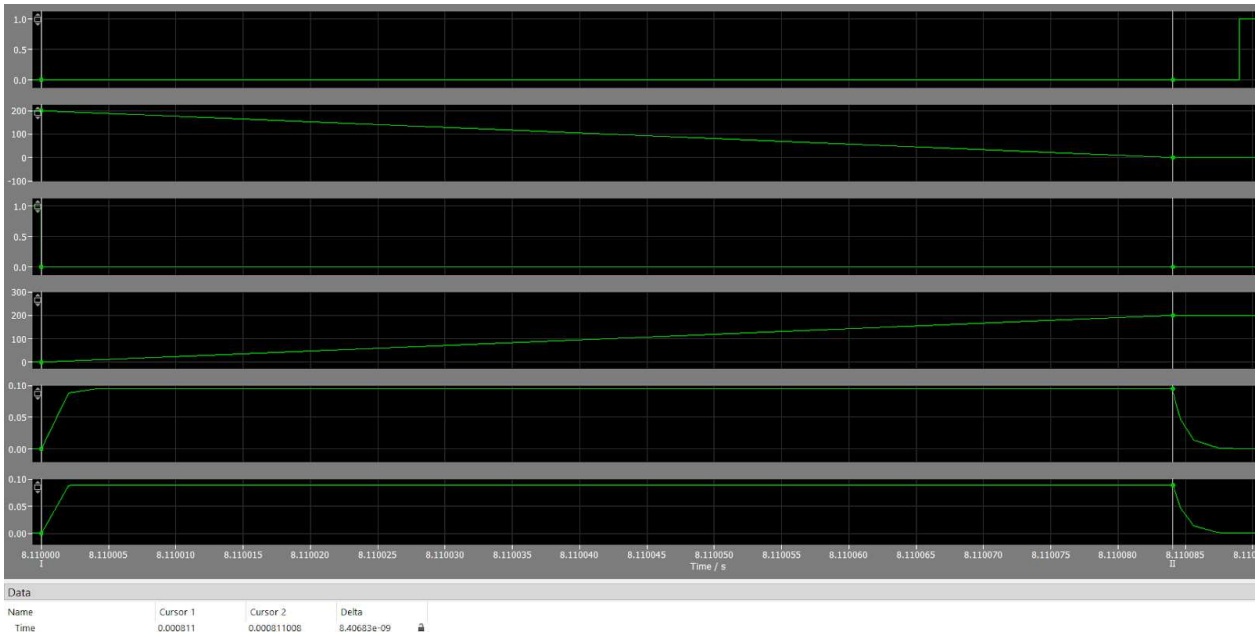
Yes a saturator does look like something we will have to add to the system In the event the dead time turns out to be too small

THe V-zvs can shoot up.

What is transpiring is - **we are having 4ns dead time and for soft switching we need 8 ns**

We will have to again analyse certain things because the simulation now behaves in a different manner.

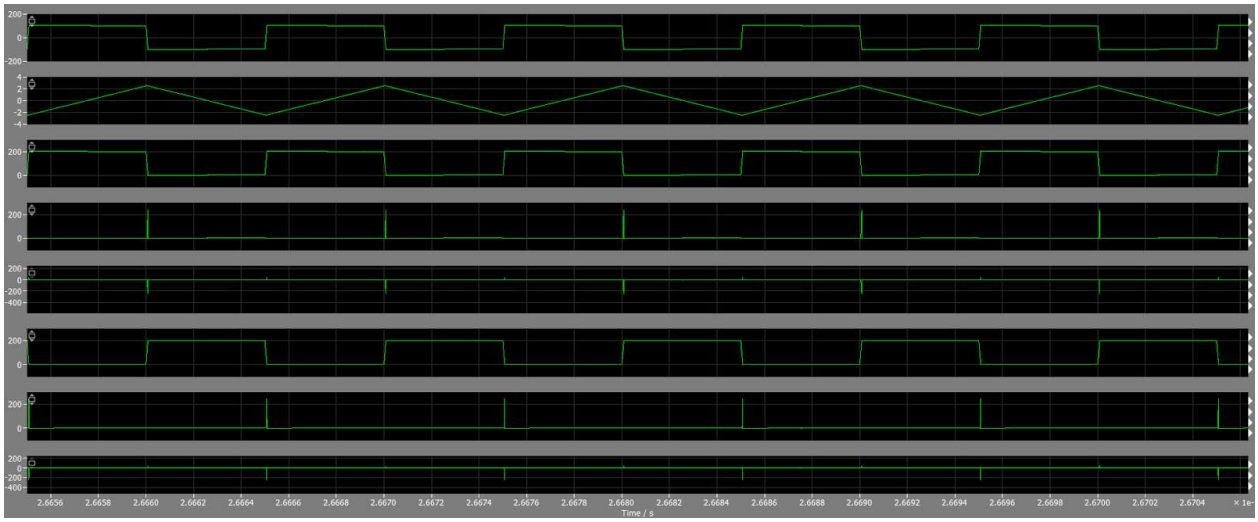
The steady state actual dead time is 8.4062 ns - This does suggest we should change the dead time by 0.2 ns intervals or something akin to it

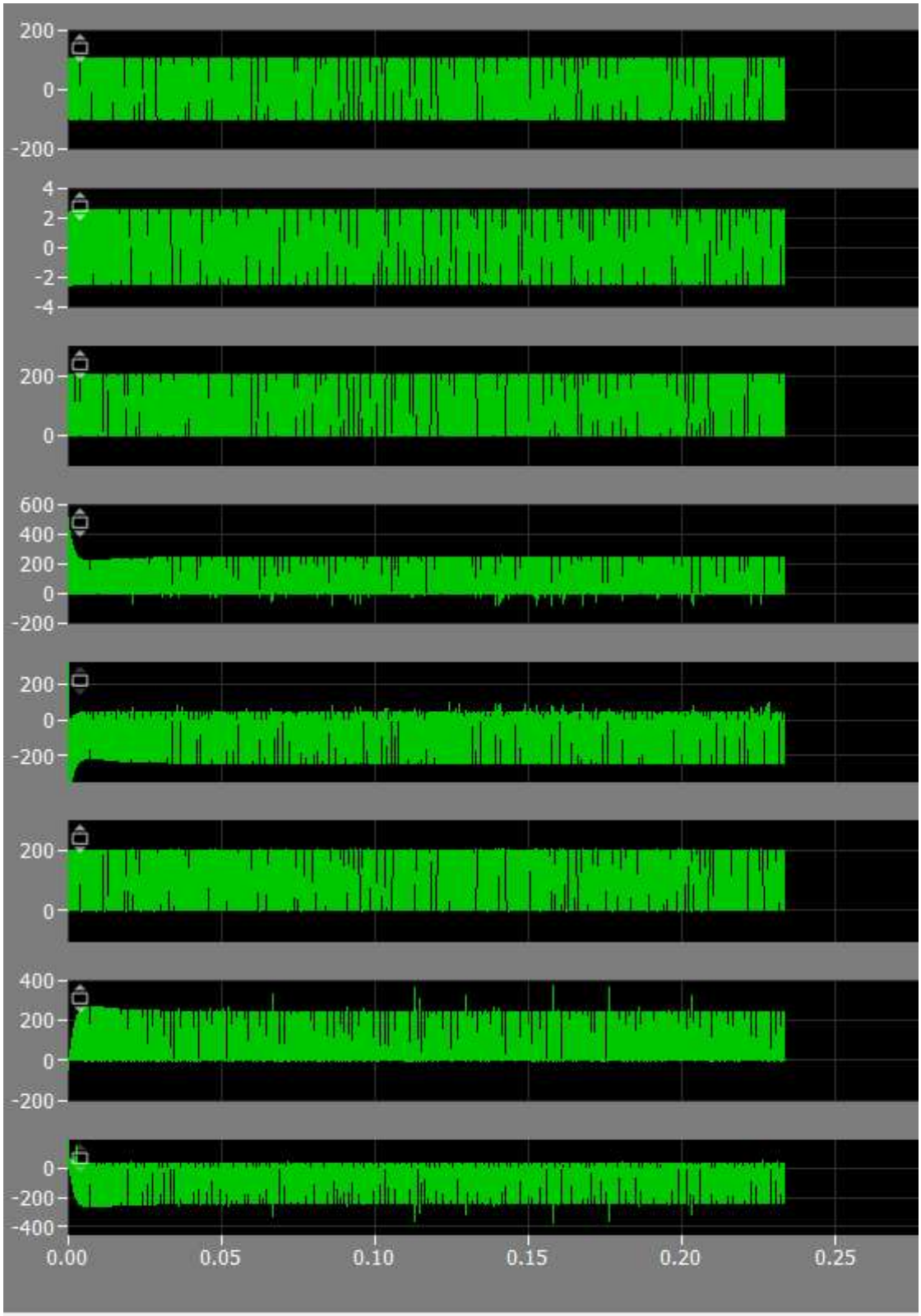


Now let us understand the variations in the dead time and the RC values and the dynamics which they now offer.

8.4ns gives 1.065 to 1.070 volt

7.9ns gives 1.165 to 1.170




200
0
-200
4
2
0
-2
-4
200
0
600
400
200
0
-200
200
0
-200
200
0
400
200
0
-200
0
-200
-400
0.00
0.05
0.10
0.15
0.20
0.25

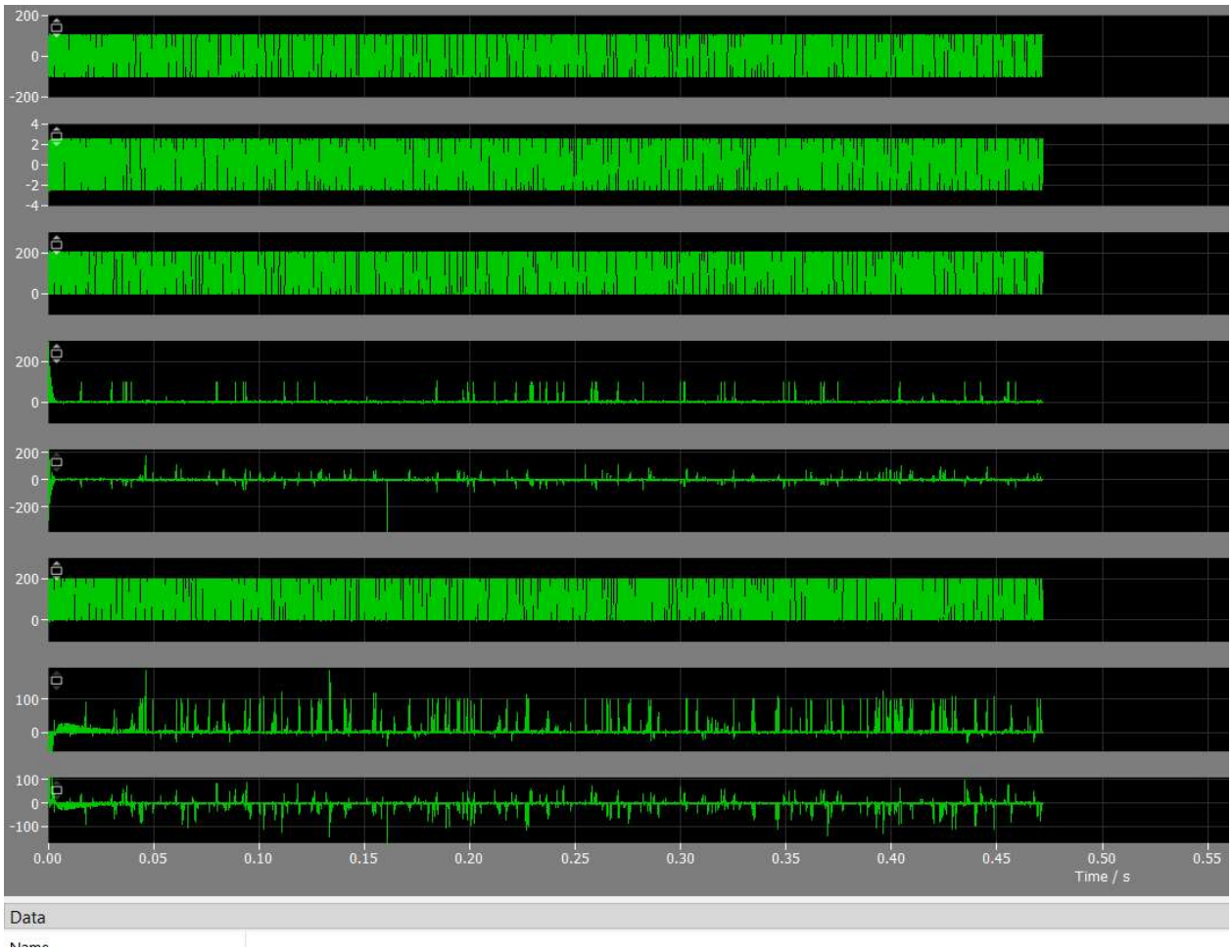
Data

Name

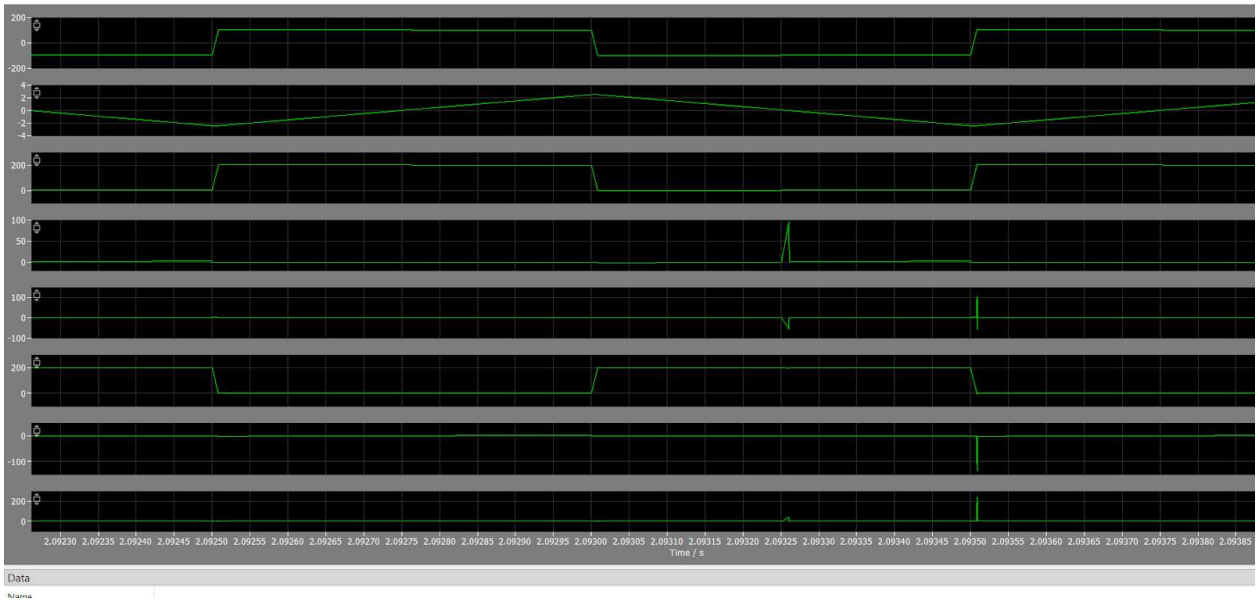
These are the switch parameters which we are looking at - The currents are farr too high - and they make sense because we focus on the Turn OFF to ON transition and not the other transitions.

‘

At 8.4ns



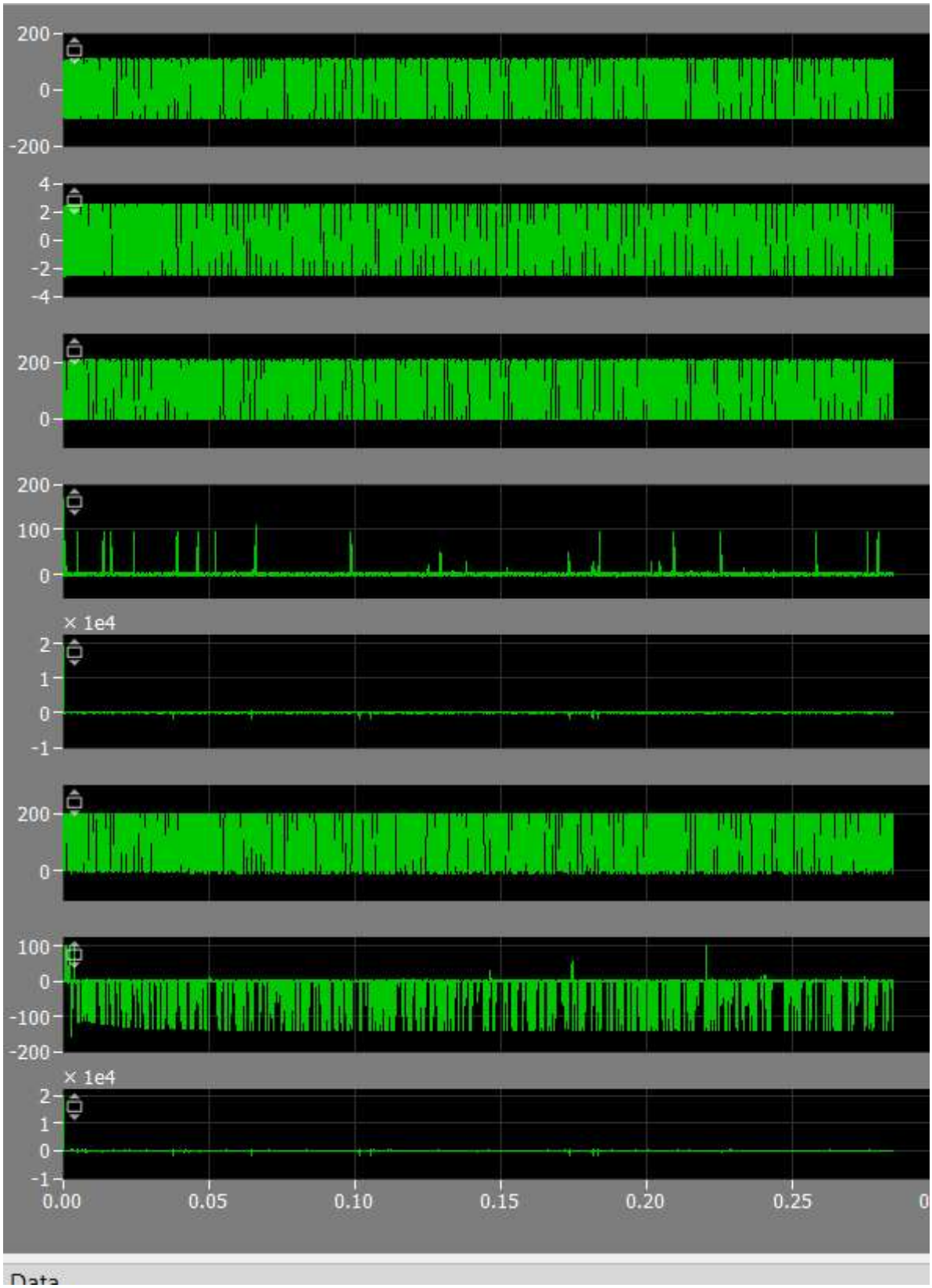
At steady state lets say 8.7ns

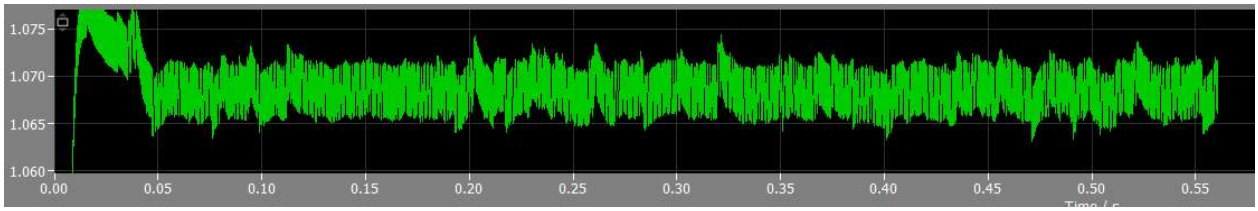


This is a simulation caveat - Sometimes randomly in the middle the voltage will start rising.

This is something that we will have to fix.

There is no fixed pattern to it as well- Maybe a simulation error or something is happening in the middle of the circuit.

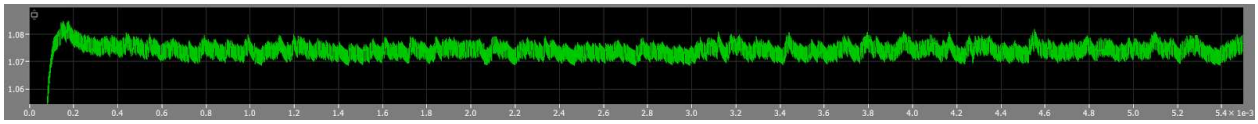

200
0
-200
4
2
0
-2
-4
200
0
200
100
0
× 1e4
2
1
0
-1
200
0
100
0
-100
-200
× 1e4
2
1
0
-1
0.00
0.05
0.10
0.15
0.20
0.25
Data



This is Vzvs at 8.4ns

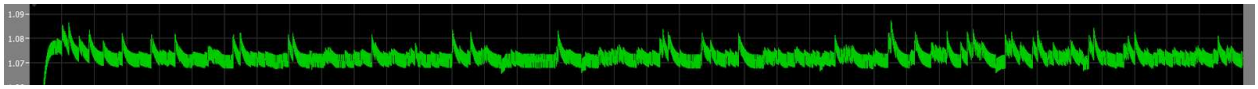
Now

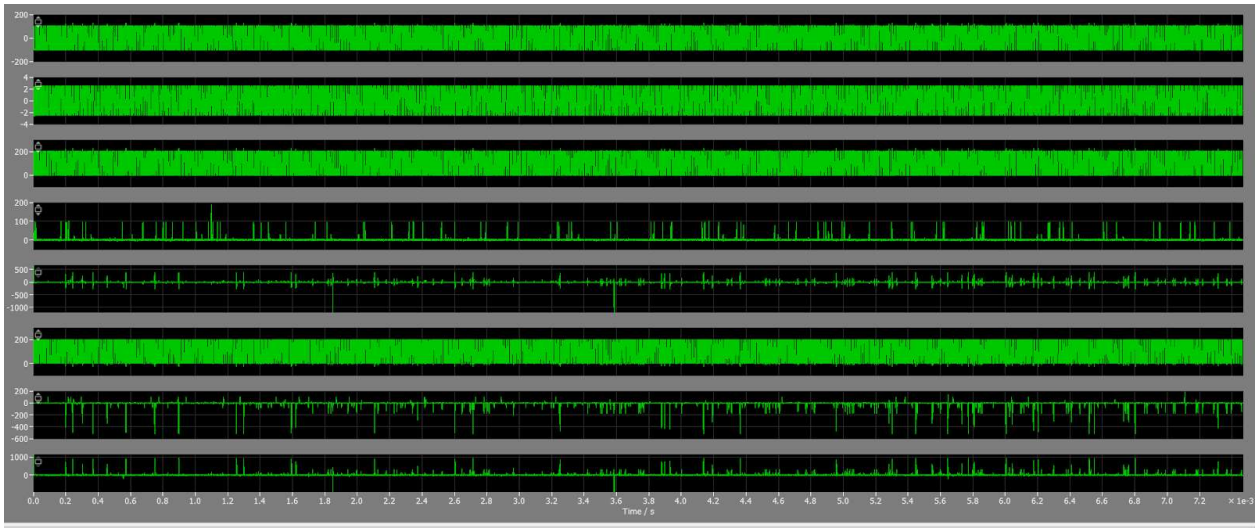
Vzvs at 8.7ns



I have no idea why this is happening

At 9.5 ns





This is what I see for the switches at 9.5ns which is a bit farr to surprising.

Unlike our previous simulations - There are phases where the body diode itself is conducting in this case - that is something that needs to be looked upon .

The reverse recovery of diode is slow and does lead to many losses.

The disparity can be due to simulation errors themselves.

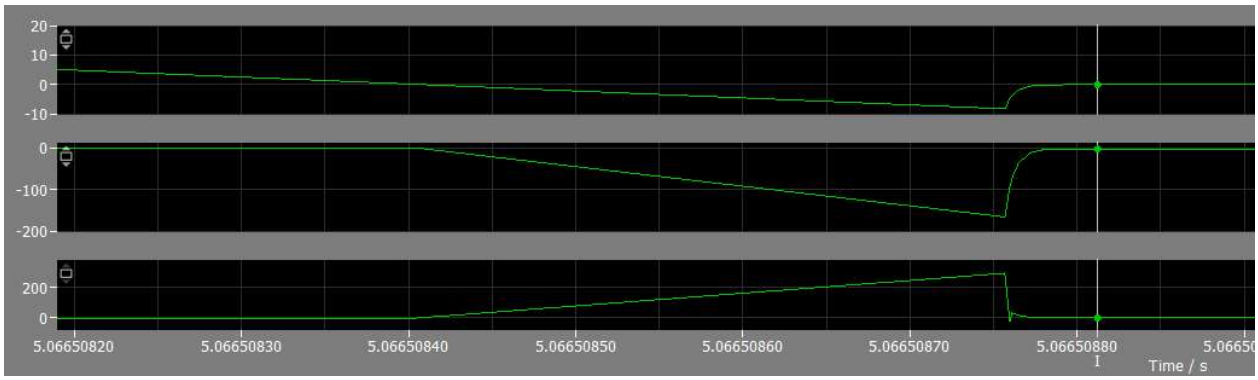
The reverse diode losses are also huge - Therefore the best solution that you need to have is Not too much dead time or too less dead time - because they will lead to losses

Just the perfect dead time so that the **diode does not conduct.**

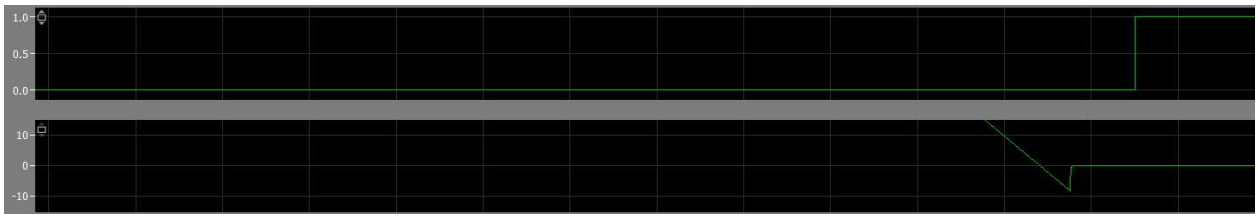
What we see is

In the usual simulation - there are certain times where the body diode does not conduct and there are certain times where the body diode does conduct - as to why it really does it - I have no clue.

It should be periodic the simulation - why it is not - I have no idea



The lower switch ke voltages , currents and parasitic current



Upper is the gate pulse and lower is the voltage of the switch.

Yes , The simulation is showing some sort of an error and therefore we have decided that we should try to simulate this circuit on ltspice.

We will now try to contain the dead time itself

That which - we will do by 2 methods

1) **C-scripting which is a powerful method**
2) **State machine - Yes plecs has that option**

We are learning more about what we can do with plecs as a software itself it is amazing - I did not think this functionality would be available and yet it is

⏳ Delay Order and Sample Time in PLECS

In PLECS (or any discrete-time simulation), the **total time delay** introduced by a delay block like z^{-1} depends on **two things**:

1. **Delay Order** (aka "Number of samples")
2. **Sample Time** (simulation time step)

- **Delay Order = 3**
- **Sample Time = 50 µs (0.00005 s)**

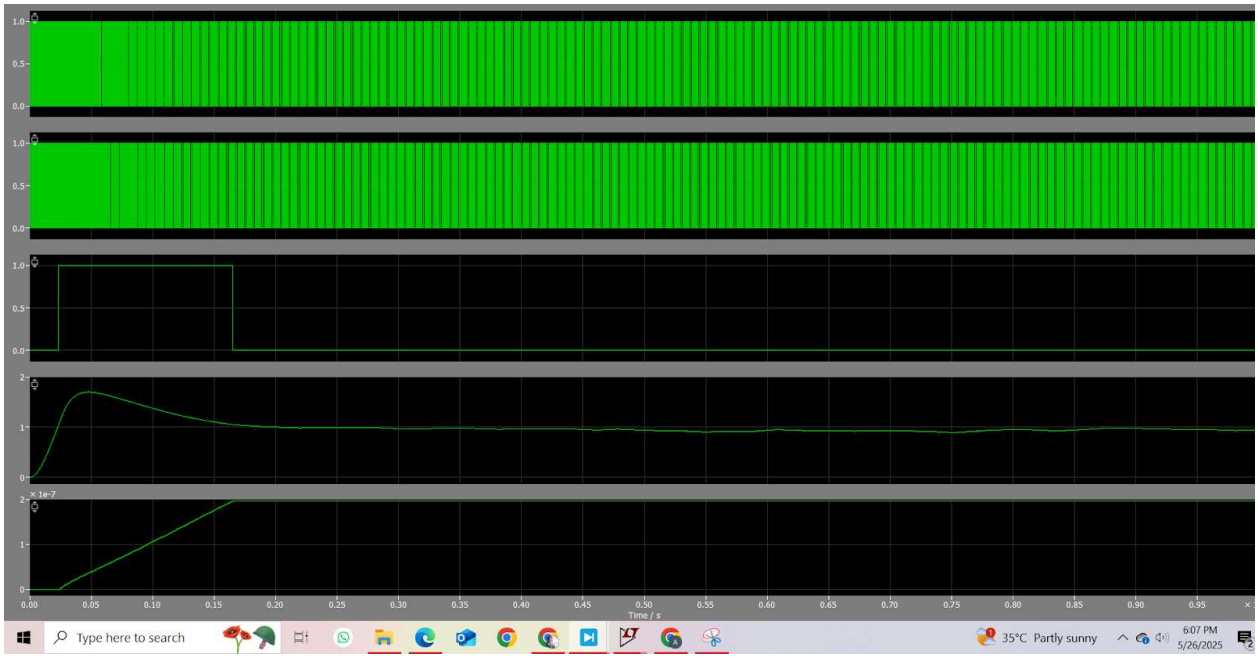
Then:

```
text                                                    Copy    Edit

Total Delay = 3 × 0.00005 = 0.00015 s = 150 µs
```

The output will be the input **150 µs ago**.

The current questions that are in my mind right now are

1) **How do I set an initial value of the dead time via the state machine**
2) **How do I make sure it does not go haywire - When I am running it manually - the simulation time steps are killing it**
3) **How do I get the animation part of it running without it being so slow**



The 2nd lowest is the comparator output , the lowest is the dead time of the system

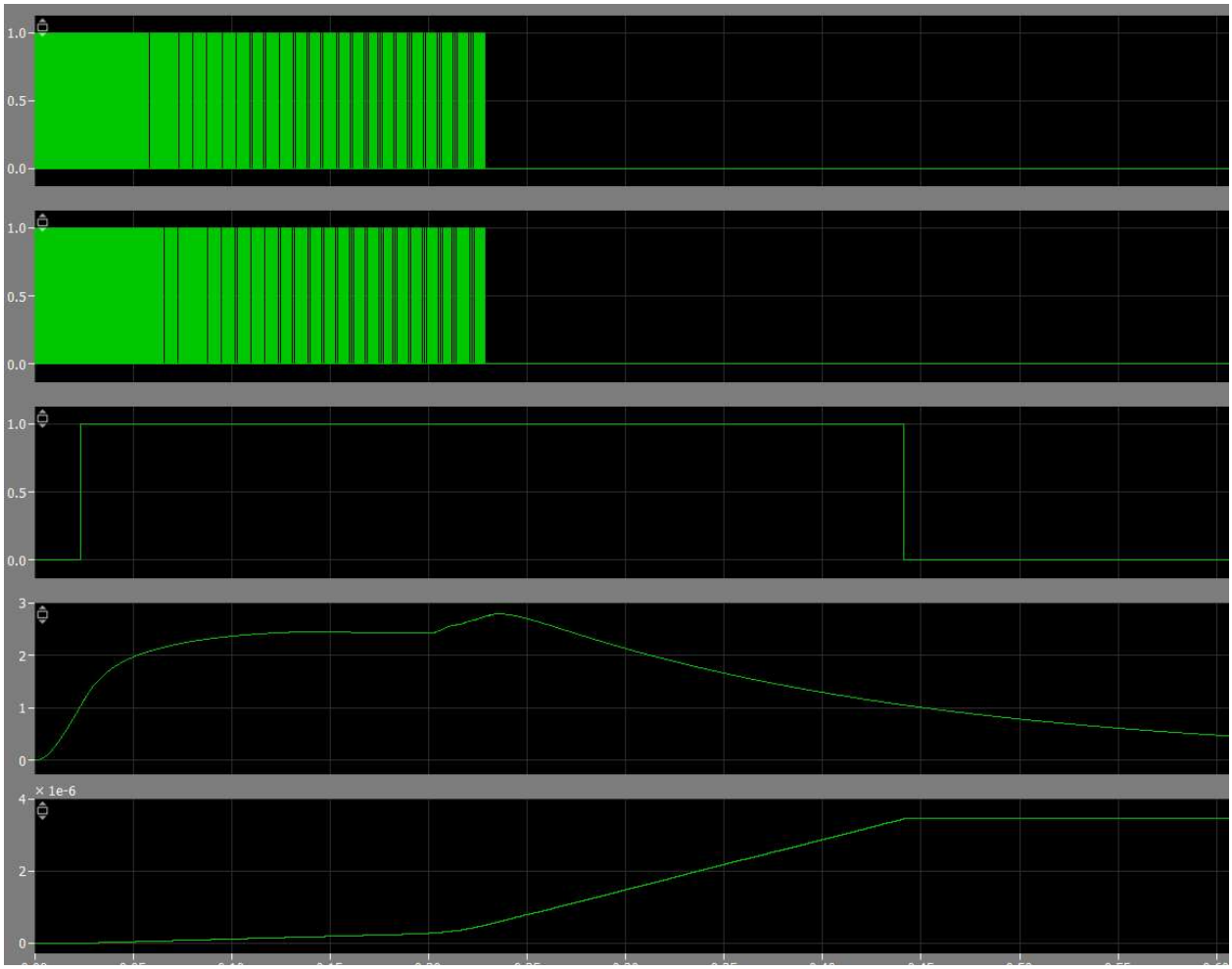
The 3rd is the comparator output where as the top 2 guys being the switch parameters.

The Caveat at hand is

1) Yes the initial value of the dead time itself starts off as very small

But the bigger caveat is that it is not behaving as expected because the state machine runs whenever a time step is running - therefore - when the simulation step sizes become too small - the dead time is increasing in magnitude farr too fast.

As compared to the switching frequency itself which is a bad metric.



This is when the load is 5e-6 - the issue is - even when we obtain the optimum dead time - because the simulation is running - but the circuit switching frequency is a bit slower - the optimum dead time is not getting evaluated and so it keeps on increasing and increasing

Until the voltage value itself collapse.

The solution to this is to make sure that the state machine itself is not running at the frequency at which my variable time solver is running - but at a frequency at which or a multiple of the switching frequency so that we can see it getting implemented itself.

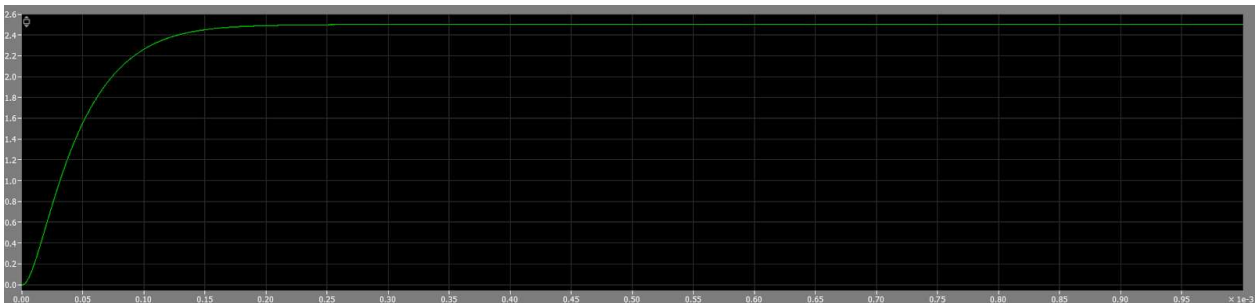
This value of increasing or decreasing - is not a fixed constant value as 1 might think.

It is a function 1) Of How fast the V_zvs itself is rising - (basically the tau with which it was rising

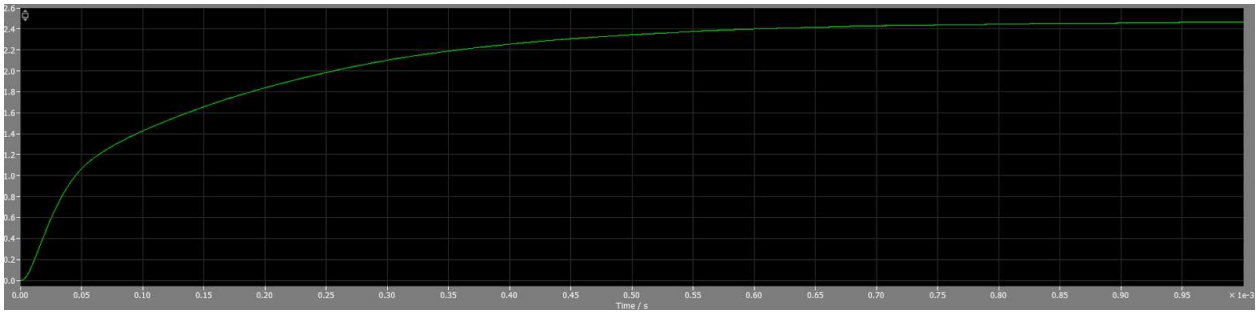
As my voltage or my dead time will be influenced by it,

2) The load - **As the load is what influences Ihf which in turn influences the magnitude of V_zvs.** Therefore these calculations take lot of priority

3) The lower the dead time - the faster will the switches burn out and therefore we also want the delay to be less in this case.



This takes roughly .2ms to reach its peak for the current values deadtime and everything



In this the inductor value is double -
And it takes .5 ms etc

At 7ns dead time

At 4ns dead time it still takes .4to .5ms to reach a certain percentage of its maximum value.

But it is a good bet to say that it reaches 2.5v at 0.07ms for the 10e-6 load

For a 5e-6 load - the required rise in current is more so the required dead time is also less.

It is a function 1) Of How fast the V_zvs itself is rising - (basically the tau with which it was rising
As my voltage or my dead time will be influenced by it,

2)The load - **As the load is what influences Ihf which in turn influences the magnitude of V_zvs.** Therefore these calculations take lot of priority

3) The lower the dead time - the faster will the switches burn out and therefore we also want the delay to be less in this case. (**INcase of same load)**

In case of load inductance increasing - we will need to increase the dead time

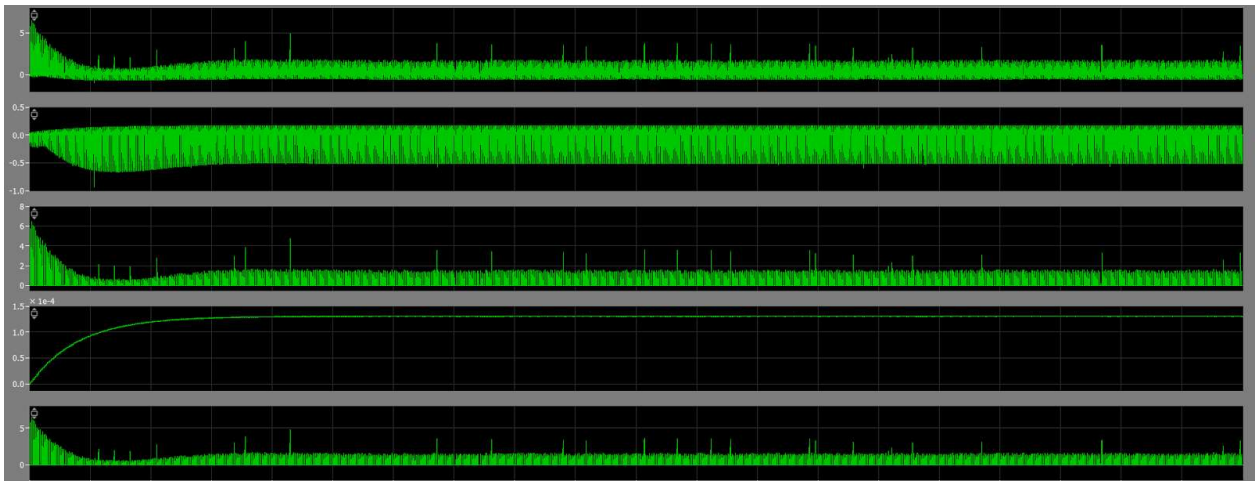
In case of load inductance decreasing - we will have to decrease the dead time as well.

The delay we need to give is a percentage of the delay - the delay is dependent on the value of Ro , and the dead time which we are varying

I now need to check if I can make sure that - it is not very much dependent on the dead time - SO that I can approximate it.

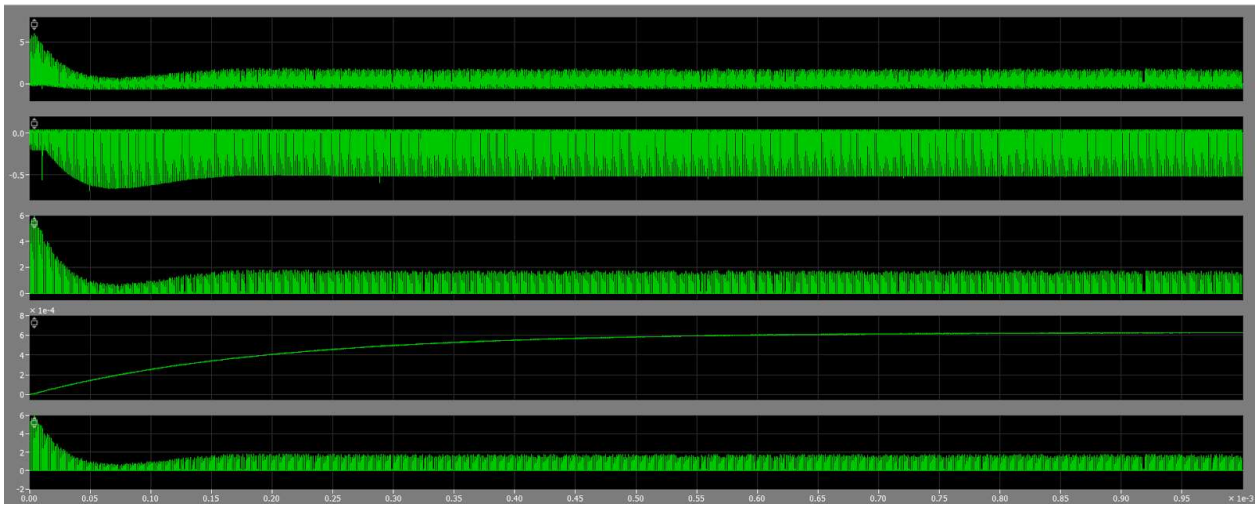
For that **I** will have to keep the RoCo = consntat and exchange some parts of em - let us see whether the circuit likes that or not

The calculations tell - dependent on the value of I_hf — let us check how much of that is true



THis is with Ro= 20k and C =10nF

Load is 5e-6 - dead time is 4ns and output is V_zvs = 2.6 v



These are currents for Ro =200and C = 0.1uF , dead time is 4ns - but the V_zvs is 0.12 which is very concerning.

But will be true- because the tau is also present in the magnitude as well as the time constant.

I think for beginners - let us skip this entire variable delay jargon as well – **we know the dead time for steady state for inductive loads- for others not yet.**

We can figure out a way then - first let us see how the simulation behaves else-wise.

🧠 Quick Note on Syntax

Symbol	Meaning	Use in PLECS
`{}`	Cell array (like a container)	❌ Avoid in State Machines
`[]`	Numeric array (vector/matrix)	✅ Safe to use
`()`	Regular grouping / indexing	✅ Safe

Also it seems that unlike python or other languages - I cannot use Internal cns

❌ Why You're Getting an Error

PLECS is showing a **"C undefined" or calculation error** because:

1. Even though `Co`, `Rf`, `Ro`, and `d` are **defined globally**, they are not **automatically visible** inside the **internal constants** of a state machine.
2. In PLECS, **state machine internal constants cannot directly access global constants** unless:
   - They are passed in via **input ports**, or
   - They are re-declared locally in the **internal constants list**.

Co/((d/Rf) +(1/Ro))

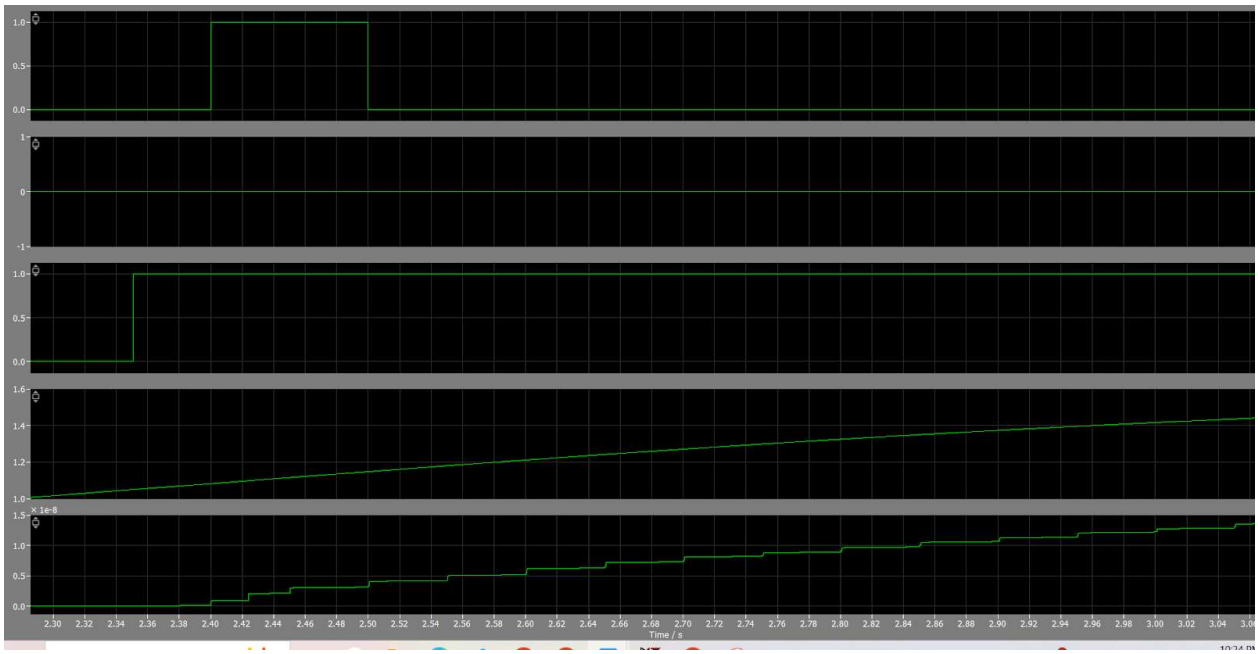
Yes , so this is an issue with the internal constants in plecs - though it is not with the internal variables.

Time-Based Trigger

A time-based trigger is an expression of the form `AFTER(delay)`, where *delay* is an expression that evaluates to a number. If the state machine uses a continuous sample time, the trigger event will be created exactly *delay* seconds after the source state of the transition was entered. If the state machine uses a discrete sample time, the trigger event will be created at the first execution time following the delay period.

This is the reason the state machine was not working - It just delays it - so in our case - basically it is not controlling its frequency - it is just delaying the inevitable by T_delay - so the entire endeavour was in a way a failure.

The trigger event works perfectly.



The issue is that Either I am not implementing it in the correct fashion or it does not work this way - because the increments are also happening when E==0 which does not make sense.

I will abandon this approach mid-way

Though I do know that C-scripts and State machines are something I will have to master If I want to grow further into power electronics.

C-scripting Basics in plecs

In the **C-Script block** of **PLECS**, the dropdown you're looking at lets you define code that runs during **specific simulation phases**. Each option corresponds to a different stage in the simulation loop, and writing code in the right section is crucial for correct behavior.

Here's a breakdown of **each function code section**:

🔷 1. Code declarations

- **Purpose:** Define global constants, macros, variables, and includes.
- **Typical contents:**

```c
#include <math.h>
static double counter;
```

🔷 2. Start function code

- **Purpose:** Code here runs **once at the beginning of the simulation**.
- **Use case:** Initialize variables, check for invalid parameters, setup.
- **Example:**

```c
if (DELAY <= 0.)
{
    SetErrorMessage("Delay must be a positive number.");
    return;
}
counter = 0;
```

🔷 3. Output function code

- **Purpose:** Computes **block outputs** based on inputs and internal variable
- **Executed:** At **every time step** when the solver computes outputs.
- **Example:**

```c
OutputSignal(0) = sin(2 * M_PI * freq * time) + offset;
```

🔷 4. Update function code

- **Purpose:** Updates **internal memory or discrete states**.
- **Executed:** Once per time step, after outputs are computed.
- **Use case:** Useful for implementing discrete logic or delay lines.

🔷 5. Derivative function code

- **Purpose:** Define **derivatives of continuous states**.
- **Used only if** the C-Script block includes **continuous states**.
- **Use case:** Modeling custom continuous dynamic systems.

🔷 6. Terminate function code

- **Purpose:** Executes once **at the end of simulation**.
- **Use case:** Logging, releasing memory (rarely nee[illegible] in typical use).

🔷 7. Store custom state function code

- **Purpose:** Used for **code generation / real-time simulation**, to store the internal state.
- **Used in:** HIL (Hardware-in-the-Loop) or code export applications.

🔷 8. Restore custom state function code

- **Purpose:** Complements the previous one—used to **reload internal state**.

Function Code	When It Runs	Typical Use
`Code declarations`	Always	Global vars, includes
`Start function code`	Once at start	Init, error checking
`Output function code`	Every solver step	Compute output values
`Update function code`	Every solver step	Update memory/discrete states
`Derivative function code`	Every solver step	Custom ODEs
`Terminate function code`	Once at end	Final cleanup
`Store custom state` / `Restore`	During code export/HIL	Save/load internal state

Suppose you're modeling a simple integrator:

$$\dot{x} = u \quad \Rightarrow \quad x(t) = \int u(t)\, dt$$

Your **Derivative Function Code** would be:

```
c

Deriv(0) = InputSignal(0, 0);
```

This tells PLECS:

- `x_c[0]` is your continuous state
- Its **derivative** is the **current input**
- PLECS will integrate it over time

- C code:

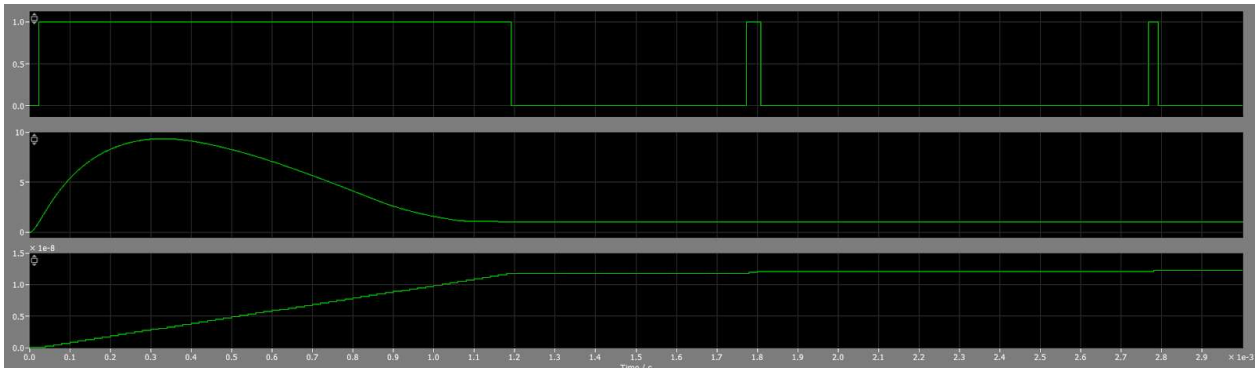
```
c

Deriv(0) = x2;          // dx1/dt = x2
Deriv(1) = -x1 + u;     // dx2/dt = -x1 + u
```

Continuous states as in state variables is what they are talking about. Ki mere X matrix ke entries

Major Steps vs Minor Steps

Aspect	Major Step	Minor Step
Definition	The main time points at which the solver updates the simulation state.	Intermediate points between major steps used by variable-step solvers.
State Update	States (e.g., voltages, currents) are updated.	States are **not** updated; only outputs may be interpolated.
Purpose	Advance the simulation by computing new values.	Used internally by the solver for accuracy (interpolation, zero-crossing detection).
Custom Code	C-Script `Output function` and `Update function` typically execute here.	Your code does **not** execute during minor steps.
Sampled Discrete Logic	Executed only at **major time steps**.	Ignored in minor steps.

Ask the Professor on the multiple sample time delay example which has been posed by the plecs manual - as to ho wto figure all of this out nad zero crossing stuff and all



The upper part is the comparator output - then comes the voltage and then the dead time .

Yes , because the system is a bit sensitive around that 1.05 value - because of very very small variations which still might take place - We have to control it.

For the current load - the best dead time value figured was 8.5 nano etc - but with the current code it goes upto 11 nano.

That will definitely lead to body diode conduction which is something we do not desire at all.

Which means - The sample time of the C script should be increased.

Or there should be some sort of piecewise function that is being executed for the same.\

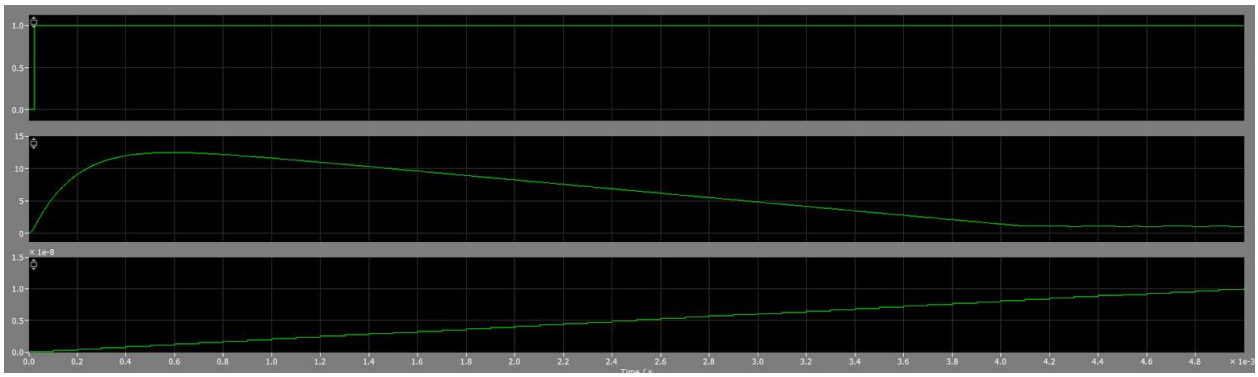
The increase in such dead time can only mean 1 thing - that the V_zvs is not spending much time with the current dead time.

We want to make it spend more time . So am varying the dead time and the sample time of the simulation as well

Sample time - 4e-5 and increment is 2e-10 - finally - 11.4ns.

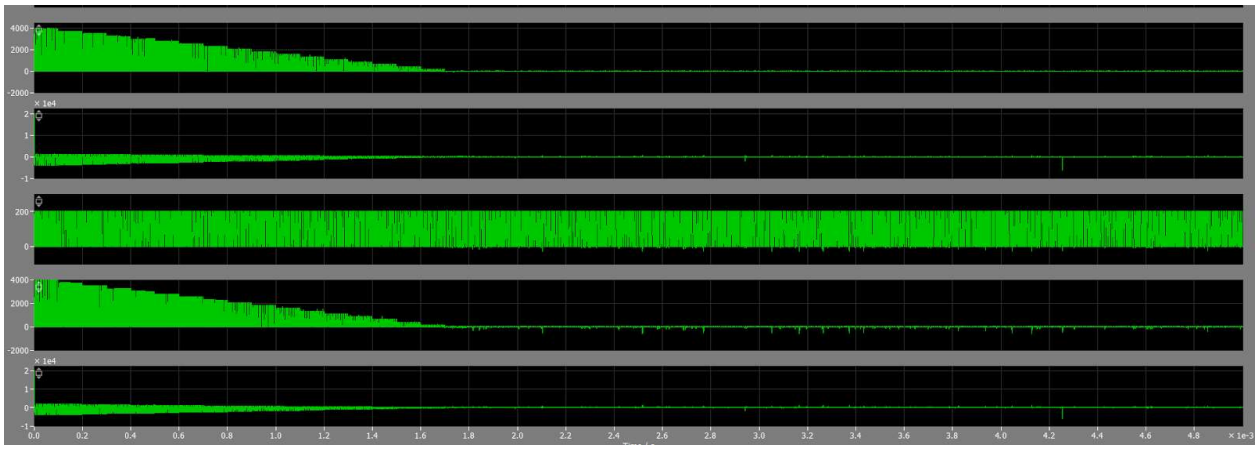
Changing the sample time to 5e-5 yields the same result.

The changes seem to be negligible



THe change is very negligible.

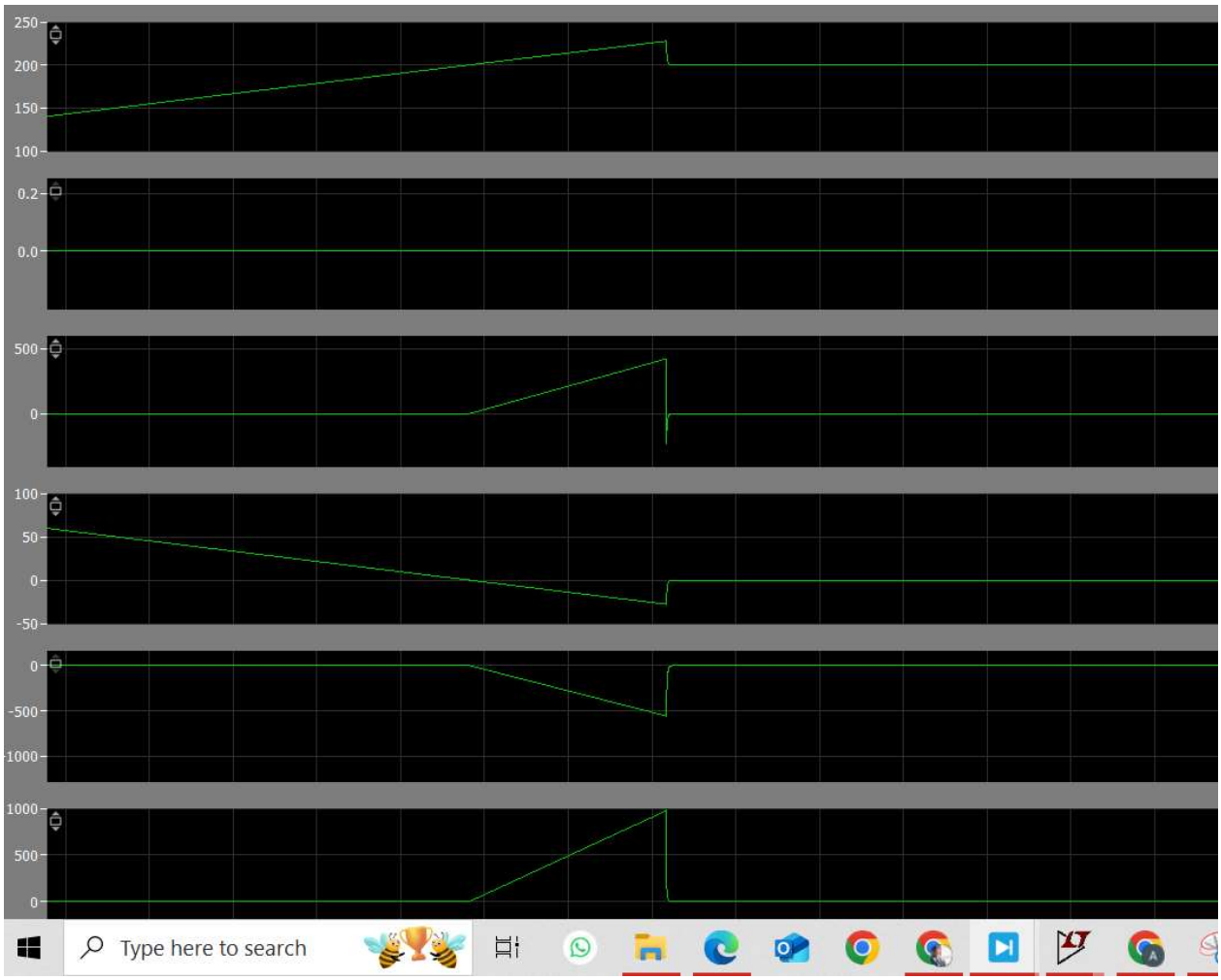
Irrespective of the gain I am setting - the dead time always tends to 11ns etc.



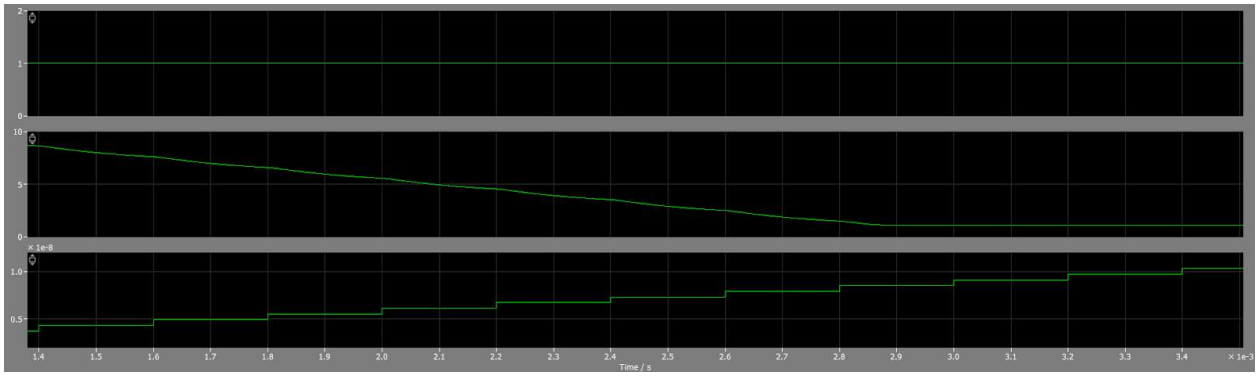
Even the currents that we witness then are too high - 4kA is nothing to scoff at - this is something we have to rectify

Upper is the V_zvs that we see and lower is the filtered V_zvs that we obtain - for 8.4 ns dead time - the value of the V_zvs that we have obtained is 1.068

Then let us try that I set the reference as 1.067 V - the final value of the dead time still reaches to about 11.4ns. And the final value of the voltage still reaches to about 1.05 V



The diode reverse conduction and everything still takes place - the issue at hand is the body diode conduction - which the simulation is not simulating in a correct fashion.



When the dead time was being changed - You realise the fact that

It looked like the voltage was immediately taking a jump.

Just a quick confirmation again - at 11ns the V_zvs DC value is 1.072 volts type - which is higher than 8.4ns dead time voltage.

Another question that arises in my mind is how is the voltage going to 1.05 volts at a dead time which is supposed to give 1.07 volts

🔍 What Are State Variables?

State variables are the internal variables that represent the **history-dependent behavior** of a system.

There are two kinds:

Type	Description	Examples
Continuous	Change smoothly over time, governed by differential equations	Inductor current, capacitor voltage
Discrete	Change only at specific time steps (sampled systems)	Counter value, logic states, FSM flags

↓

2. **Look in the Simulation Log (PLECS Viewer)**

After running a simulation:

- Go to the "**Simulation Log**"
- Open the **States** tab
- You'll see a list of state variables:
  - Their **name or origin block**
  - Whether they are **continuous or discrete**
  - Their values at each time step

\

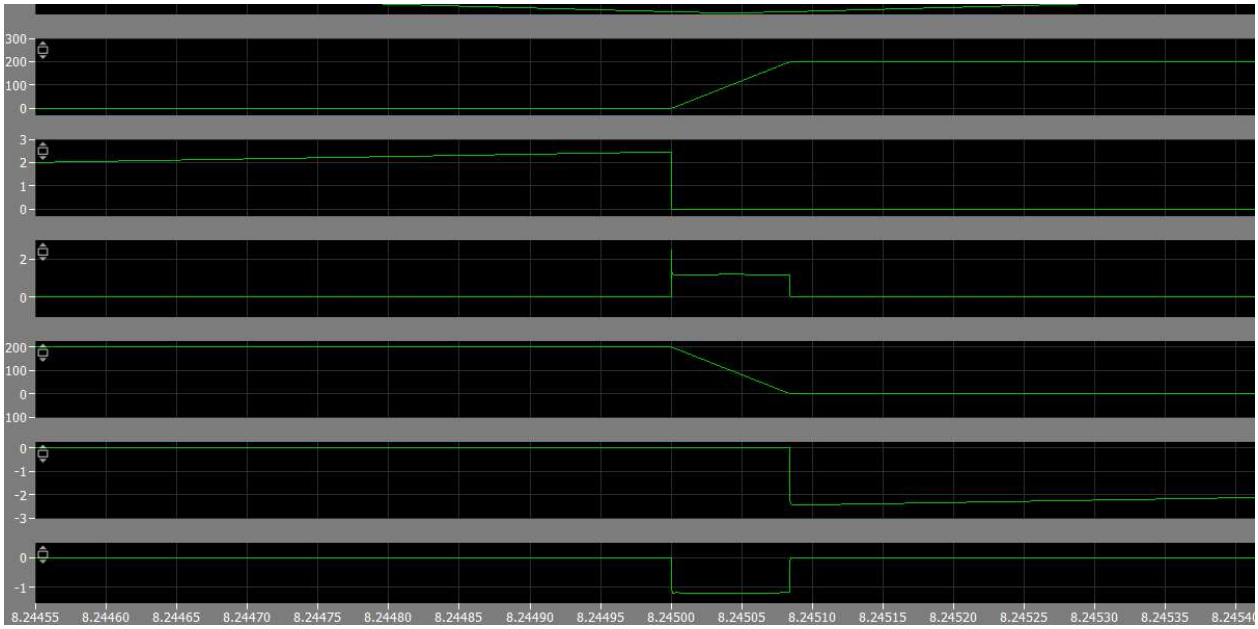
1. Automatic Identification via Blocks

PLECS assigns state variables based on the blocks you use:

Block Type	State Type	Notes
Inductor / Capacitor	Continuous	Holds energy; requires numerical integration
Discrete Transfer Function	Discrete	State is memory of past inputs/outputs
C-Script (with states)	Discrete or Continuous	You choose how many states of each type
State Machine	Discrete	Internal logic memory (e.g. current state)
Integrator / Derivative	Continuous	Mathematical integration (continuous)
Z^{-1} (Unit Delay)	Discrete	1-step delay = 1 discrete state

Checking the I_check , I_g currents and other currents we are witnessing that the Snubber capacitors have current flowing through them and in huge quantity - they are not aligning with the original values which the simulation was showing.

These values also have to be looked into



This is yet another simulation error which we were witnessing which happens as to how the inductor current will be flowing thru the switches during dead time.

For inductance value 10e-6 - the voltage value that is being needed is 1.073 according to theoretical calculations. With a tau of 2.12e-5

Realise that we are initially seeing the peak voltage touch a value of 1.079 and then decreasing to 1.071 - why I do not understand. According to theoretical calculations it should follow some other curve

With L = 5e-6 it is 2.504 the steady state value in the simulation

The calculated value comes out to be roughly 2.525 V

Note in the above calculations - the Ihf we have take in the Ihf the simulation is showing to us - we are not taking some other Ihf into consideration.

Theoretical calculations of Ihf can be easily done - as dead time is farr smaller than the interval of increase or decrease- it can and will be effectively ignored.

But for accuracy of the calculations if we see - even

The delta Ihf will be Codv/dt - and here dv/dt is calculated via the dead time because that is the dt we are looking at

We are getting the calculation as 0.1904 as opposed to 0.1915

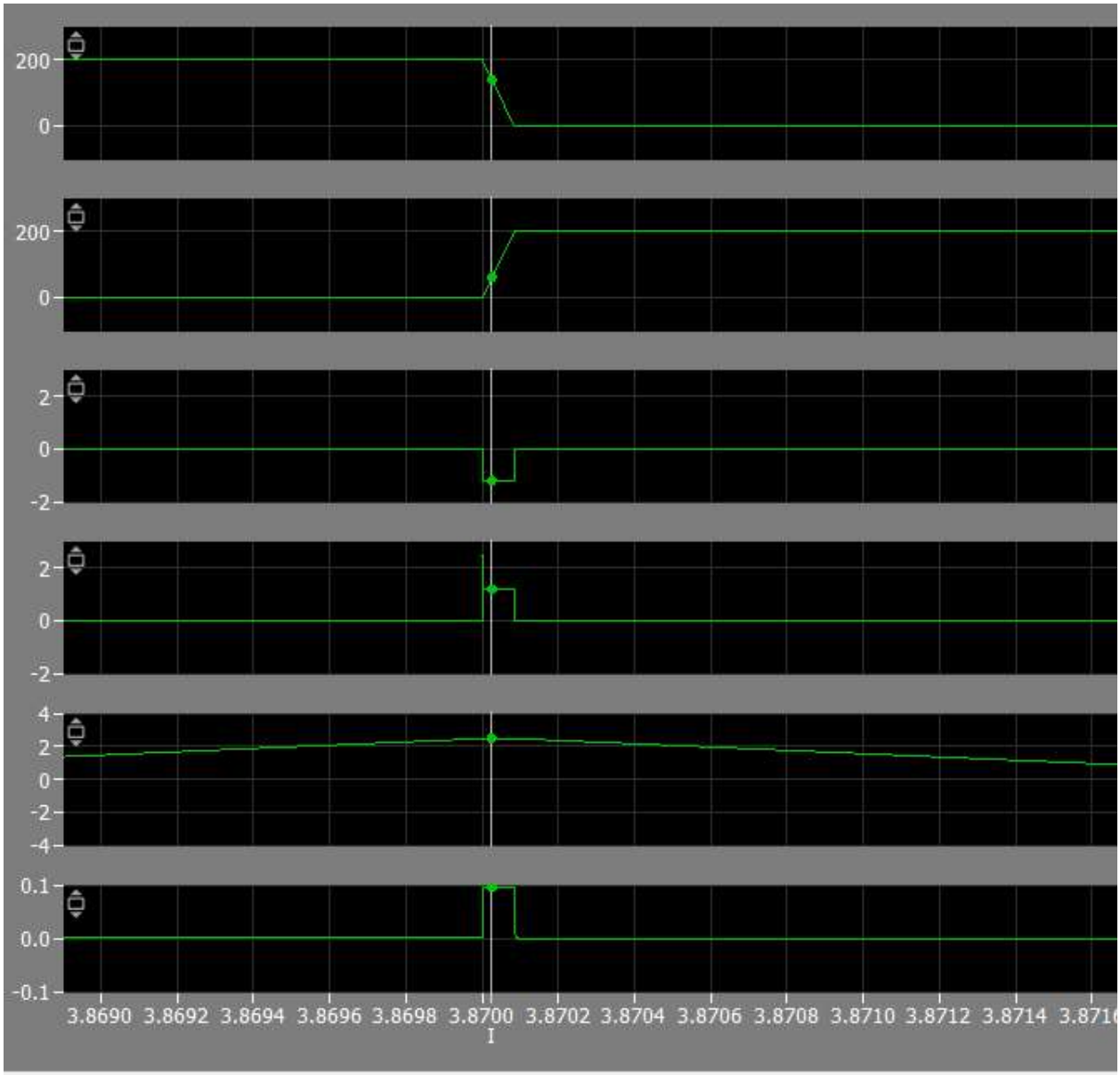
But if I take 0.1904 as opposed to 0.1915 for the V_zvs calculations - you get the Vzvs as 2.507 which is what the simulation is showing.

Going to verify it for 10e-6 - it shows an Ihf of 0.0952 and Vzvs as 1.0765

2.5e-6 - the theoretical calculation shows 0.3809 and the simulation shows 0.3837

Vzvs - 4.6368

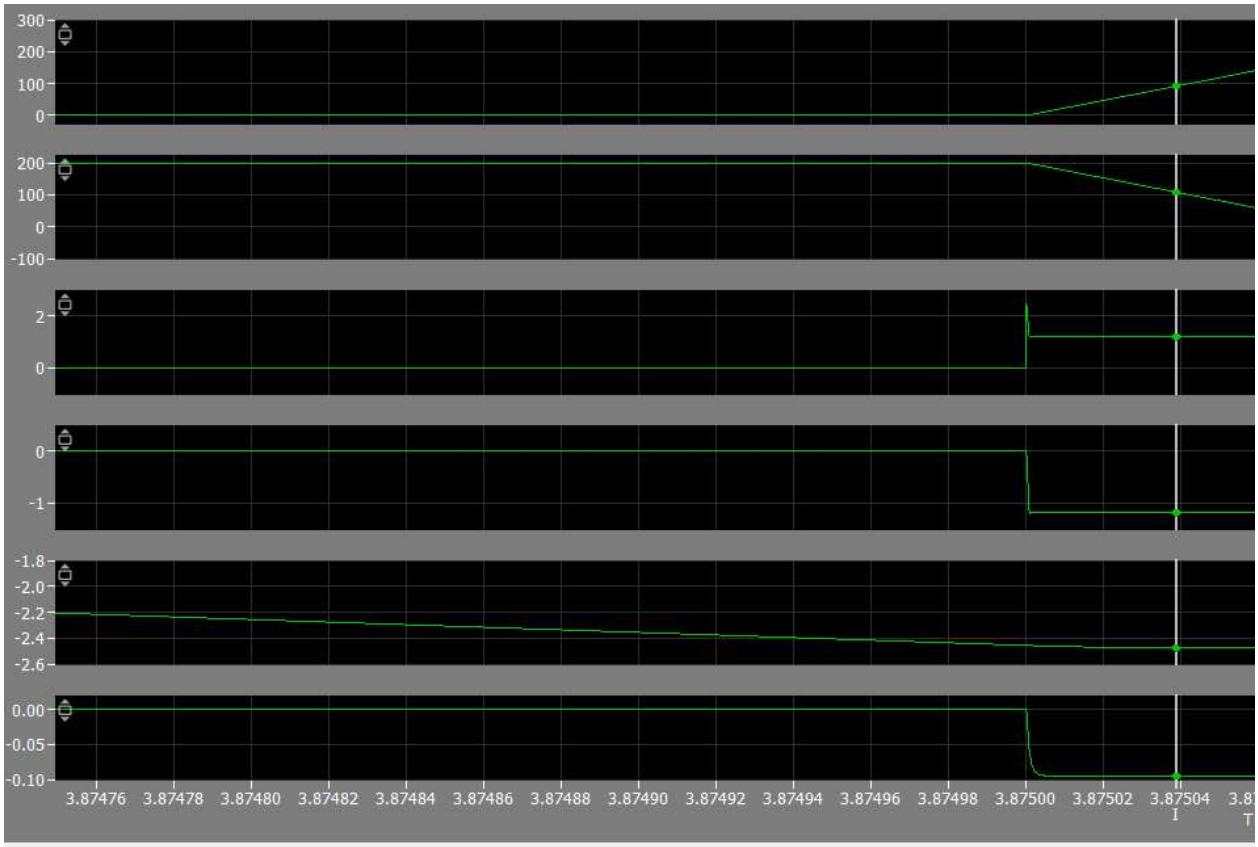
20e-6



Data

Name	Cursor 1	Cursor 2
Time	0.000387003	0.000387444
Plot 1		
	139.318	0.0925173
Plot 2		
	60.6797	199.907
Plot 3		
	-1.19101	0.026573
Plot 4		
	1.191	-0.0192769
Plot 5		
	2.47729	-1.8962
Plot 6		
	0.0952778	-2.11133e-06

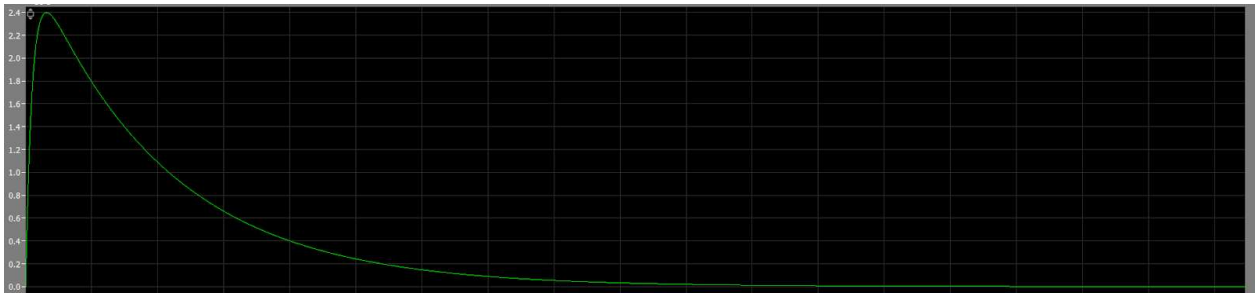
As you can see - the simulation does tell that - when the switch node voltage is rising - then AMM_HIGH + Both parasitic currents = I_load current



Data

Name	Cursor 1	Cursor 2
Time	0.000387504	0.00038752
✔ Plot 1		
	91.9399	200.115
✔ Plot 2		
	108.058	-0.117062
✔ Plot 3		
	1.19106	-0.00029913
✔ Plot 4		
	-1.19106	0.000518773
✔ Plot 5		
	-2.4774	-2.34043
✔ Plot 6		
	-0.0952815	1.98297e-06

The same thing is what you see when the switch node voltage is decreasing.



This is the case when the load is 40e-6 - I have seen inductors in my time of higher value as well - So this is bad news.
The steady state does make sense - it has to be this low.

It was very low for 20e-6 as it is roughly 0.22 v - and therefore is very very low for 40e-6 as it is - There needs to be some sort of amplification mechanism that is taking place.

The overshoot itself is also very worrisome - which I am not sure why it is behaving the way it is.

This needsto be checked. - It literally is kissing zero .

Which makes sense if it is thought about - Because the **Ihf - vd/rf term is governing - As the dead time itself is increasing in magnitude - you must realise that Ihf magnitude reduces.**

It kissing zero poses the issue that the voltage is not being generated enough to actually overcome the diode forward.

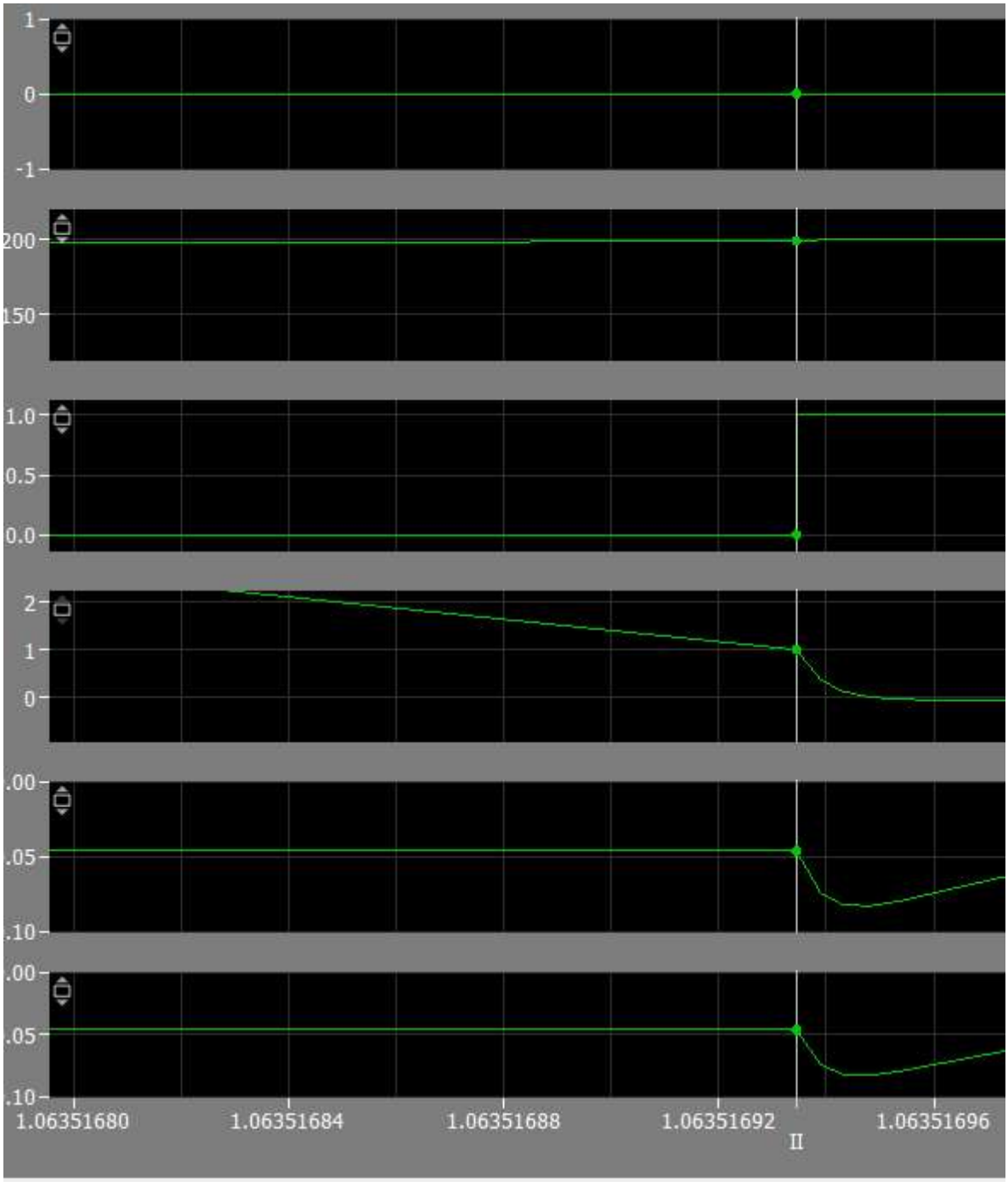
Therefore the resistance value needs to be increased.

Then the other issue that will arise is - It will act less and less of a high pass filter because the corner frequency will keep and keep on shifting to the left - similar issue with decreasing the cap value.

The corner frequency starts shifting to the right and the problem we face becomes much **worse in nature.**

This shall be discussed upon and till now we limit our discussion to 20e-6 inductive loads.

Till that point - the approximation of dead time is perfect in nature.
(we are currently at 1 Mhz switching frequency)



ata

ame	Cursor 1	Cursor 2	Delta
Time	0.0010635	0.00106352	1.69344e-08
Plot 1			
Turn-on Delay	1	0	1
Plot 2			
Vm2:Measured voltage	0.060136	198.994	-198.934
Plot 3			
Turn-on Delay2	0	0	0
Plot 4			
Vm5:Measured voltage	199.939	1.0054	198.934
Plot 5			

When the inductive load is 20e-6 - in that case because the theoretical calculations fall short by 0.3 ns etc - the diode does conduct because the reverse voltage it will be seeing is 1V .

We have a value of the ZVS ready , at least the theoretical implications of it

I have tested it by changing the inductance values.

1)I am yet to measure the effects of what will happen when I vary many different parameters on which it depends and how all the other quantities change accordingly. (Keeping other constant and only changing the inductance gives an acceptable rate.

2) Figuring out how to move to a variable inductive load case.

3) Implementing that the dead time can decrease and not only increase in nature

4) tring the c script for other values of inductors. - did try it for a different inductor value - have realised that - we will have to figure out a different sample time and a dead time incremental part for the same.
They just cannot remain same when I am changing my load ka nature in magnitude - have to figure this as well - because the tau is changing its value as well.

For 10e-6 - I had kept the sample time as 12e-5 and tau was 2.12e-5and for 5e-6 I kept it as 24e-5. And tau was 3.83 e-5

6 times of the round function seems like a good approximation. - because it gave 8.5ns for 10e-6 and is giving 4.3ns for 5e-6 which are the best I can have for the given least count.

For 2.5e-6 it gives 2.2ns and sample time is 6*round(6.626e-5)

For 1.25e-6 we took 54e-5 - got 1ns and we need 1.034 ns for soft switch - time was 9.48e-6(edge case) even 60e-5 yields same results - so the round approximation is valid

Have implemented the sample time code- only minor tweaks will be required in this.

Because the tau itself was almost double in nature. These are values of sample time that are giving about the best results when I am applying a dead time of increment 0.3ns always.

Have to give this a formal mathematical push and we shall be done.

We would always want the Vzvs actual at steady state to be a bit less than V_zvs theoretical - so that the comparator functions the way it is supposed to - also realise that it should not be higher than a good margin - because in taht case - **IT wont be a reliable metric against hard_switching - therefore you want your metric to be such**

Making the Vzvs_theo = 1.01* V_zvs calculated - makes the simulation work much more smoother and it takes care of all the worries that I had - which I have written in the paragraph just above - atleast for the range that we have designed it for yet.

When we simply put inductors in parallel and then try to simulate the circuit - it does show an error to it. As the current changes its value immediately there is a huge voltage change associated with it.

To Mitigate this error that is arising - we are to add a very high resistive value in parallel to the inductors - This way - the error does not occur because the current now also has an alternative path to flow through

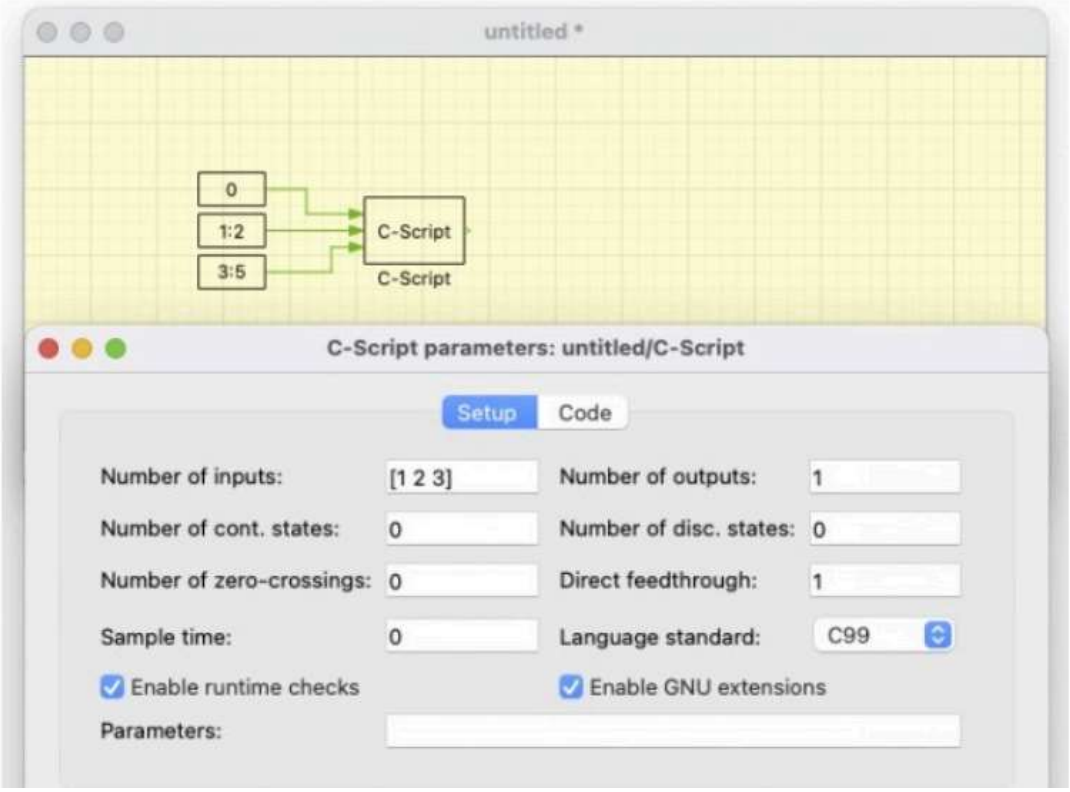
And the system does act the way we want it to - effectively only as 2 inductive loads within the system.

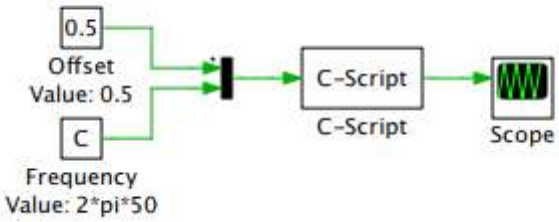
There is a certain issue at hand - it is when I am changing my dead time midway.

We always have this assumption that when we once do some load change - we are taking it to steady state - that we are not changing the value which it shows to us mid way - we let it first settle

The C-Script block accepts multiple, vectorized inputs. You can set the “Number of inputs” field as a vector which will specify the width of each input. Within the C-Script you can access the desired input signal using the “InputSignal(int i, int j)” macro, where the first argument is the input signal and the second argument the j’th element of the i’th input signal (zero indexed). In the screenshot below InputSignal(2, 1) would return 4.

A similar indexing approach is applied to output signals as well.





This Mux of plecs is used to create an array of entries - not the mux which we use in digital electronics. It passes values as an array of elements

The Idea now is that - **We will keep our sampling frequency very high from the get go - and depending on the inductor values - we shall change the rate at which the output code itself will run.**

A vector signal with n inputs has a signal width of n

> Say you have a 3x3 matrix. If I'm understanding your question correctly, one could:
>
> Set the Number of Inputs field be "9". There will be one input port to the C-Script block. Access the matrix entries as InputSignal(0, 3*Row + Column)Set the Number of Inputs field be "[3, 3, 3]". The C-Script will have 3 input ports, each with three elements. Access the matrix entries as InputSignal(Row,Column)

Values entered in the Model Initialization Commands or Mask Initialization Commands can be passed into C-Scripts using the "Parameters" field, configured in the C-Script's "Setup" tab.

From the C-Script documentation, the "Parameters" field is described as: "A comma-separated list of expressions that are passed as external parameters into the C functions. The expressions can reference workspace variables and must evaluate to scalars, vectors, matrices, 3d-arrays or strings."

You can then access the parameter entries using the **ParamRealData()** or **ParamStringData()** macros.

ParamRealData(i,j) is ith col and jth row for some reason

- `i` refers to the **ith user-defined parameter block** for the C-Script.
- `j` refers to the **linear index** into that specific parameter.

🔗 Think of it Like This:

You can define **multiple user parameters** in the C-Script block:

```
text

User Parameter 0: [1, 2, 3]
User Parameter 1: [4 5; 6 7]  → 2×2 matrix
User Parameter 2: 9           → scalar
```

Now:

- `UserParameter(0, 1)` → gets `2` from parameter 0
- `UserParameter(1, 3)` → gets `7` from the matrix in parameter 1
- `UserParameter(2, 0)` → gets `9`

This is what plecs meant when they wanted to convey how you can give parameter inputs.

They should make a better documentation

✅ 2. Make sure you're using **C99 or later**

In the C-Script block settings, make sure the **Language Standard** is set to **C99** (or newer).

- `round()` is part of the C99 standard.
- If you're using C89, the `round()` function **won't be available**, and you'll get a compiler error.

In a PLECS C-Script block, the `Output`, `Update`, and `Derivative` sections are meant to contain **code execution**, **not function definitions**.

Your code:

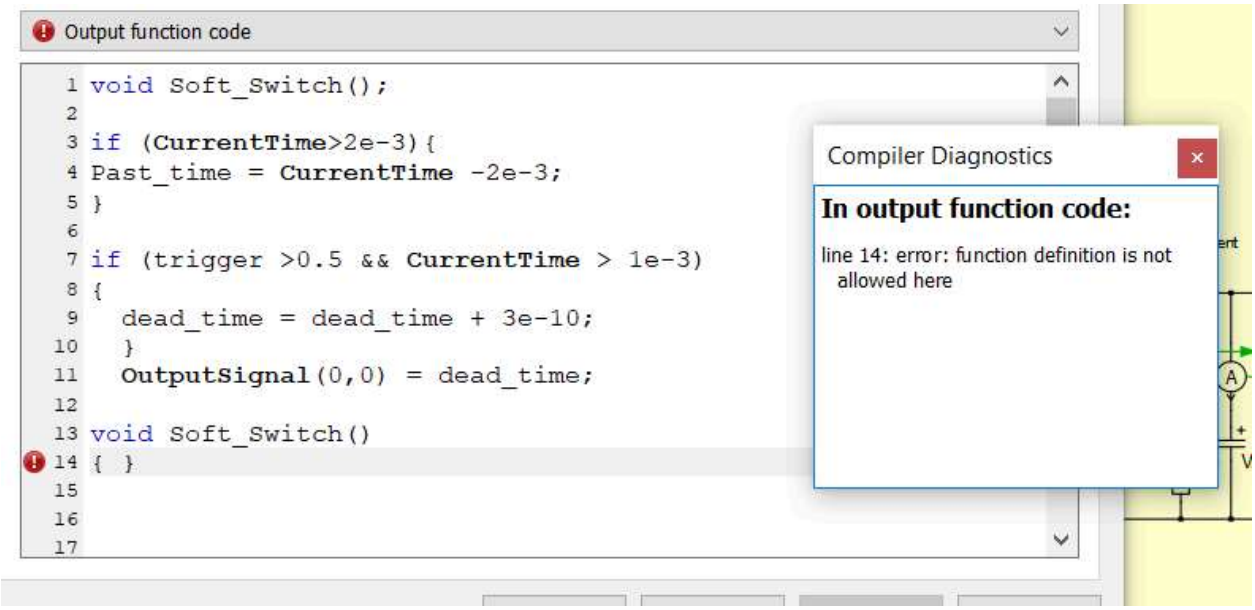
```c
void Soft_Switch();  // Declaration
...
void Soft_Switch()   // ❌ Definition (not allowed here)
{
}
```

You're **defining a new function inside the Output function section**, which leads to the error.

✅ How to Fix It

To define custom functions like `Soft_Switch`, you must place them in the "**Declarations**" section at the top of the C-Script block.

This is most definitely true - because I faced a similar issue with this in recent times



⚠ Why This Error Occurs:

In PLECS C-Script:

- The **Declarations** section is for:
  - Global variables
  - User-defined functions (with only pure C logic)
- It **cannot use simulation-specific functions or variables** like:
  - `CurrentTime`
  - `InputSignal()`
  - `OutputSignal()`
  - `ParamRealData()`

These are only accessible in:

- **Output function code**
- **Update function code**
- **Derivative function code**

This is corresponding to the fact that the logic I had written had initially used these

So I will have to pass these values as arguments

In C language can i directly use the variable that I pass as a function parameter in the code ? and will its value only be changed within the scope of the function or also outside

In C language, **yes**, you can directly use a variable that you pass as a function parameter. However, **whether the value of that variable changes outside the function depends on how you pass it**:

🔷 1. Pass-by-Value (Default behavior in C)

When you pass a variable like this:

c Copy Edit

```c
void modify(int x) {
    x = 10;
}
```

```c
int main() {
    int a = 5;
    modify(a);
    // a is still 5 here
}
```

✅ Inside the function: You can use `x` freely.

❌ Outside: The original variable `a` **remains unchanged**, because only a **copy** was passed in.

🔷 2. Pass-by-Reference (via Pointers)

To **change the original value**, pass a pointer:

c Copy Edit

```c
void modify(int *x) {
    *x = 10;
}

int main() {
    int a = 5;
    modify(&a);
    // a is now 10
}
```

✅ Inside: You dereference the pointer (`*x`) to modify the original value.

✅ Outside: The change **persists**, because you're directly modifying the memory location of `a`.

We shall stick with pass by value - much simpler for us right now

There is a caveat that I had not anticipated at all - which is I now have to run the calculations on I_load - to see how accurate they are to the theoretical model

I have done it for I_hf - but because i wont have access to it for a good bit time - but I can relate with I_load with near perfection - that is what I have to achieve now.

Then plug it in them equations.

🔷 Question 1: If I defined `V_feedback_old` globally and modify it in the function, will it update globally?

✅ **Yes**, it will.

In C, when you declare a variable **outside of all functions**, it's considered a **global variable**. If you reference or assign to that variable inside a function **without declaring a local version**, you're modifying the **global version**.

So this line:

```c
V_feedback_old = feedback(V_feedback, time, T_switch);
```

modifies the global `V_feedback_old`, and the new value is stored globally.

🔷 Question 2: In the 3rd `if` block, if I use `V_feedback_old`, will it reflect the updated value?

✅ **Yes**, it will **use the last updated value** from the global variable — including changes made earlier in the same function.

So here:

```c
if (1.02 * V_feedback_old < V_feedback)
```

You're using the **latest value** of `V_feedback_old`, which you set earlier with:

```c
V_feedback_old = feedback(...);
```

⚠️ Important Caveat: Scope matters

This only works **as long as you don't redefine** `V_feedback_old` **inside the function** like this:

```c
double V_feedback_old = ...; // ❌ This hides the global variable
```

If you do that, you'll have a local version, and the global one will be unaffected.

✅ Conclusion

- ✔️ Yes, your use of `V_feedback_old` globally and within the function is valid and will behave as you expect.
- ✔️ Any changes made inside `Soft_Switch()` will **persist globally**.
- ❌ Just avoid redeclaring it locally (no `double V_feedback_old = ...;` inside the function).

Would you like help refactoring the whole logic for better readability or maintainability?

🔷 Global Variables vs Pass-by-Value

✅ Global Variable Access

When you use a **global variable** inside a function:

- You're **not passing it** into the function as a parameter.
- You're **directly accessing/modifying** the variable from the global memory.

Example:

```c
double global_var = 10;

void change() {
    global_var = 20;  // Directly modifies the global variable
}
```

➡️ After calling `change()`, `global_var` is 20.

✅ Pass-by-Value

In contrast, when you pass a variable **as a parameter** to a function:

- C **copies the value** of the argument into the function parameter.
- Any modification to the parameter inside the function **does NOT affect** the original variable.

Example:

```c
void modify(double x) {
    x = 100;  // Only modifies the copy
}

int main() {
    double a = 50;
    modify(a);
    // a is still 50 here
}
```

➡️ Even after `modify(a)`, `a` remains 50 because `x` is a copy (pass-by-value).

🔍 Why This Happens in PLECS:

In PLECS C-Scripts, inputs, parameters, and some other special variables (like certain outputs or internal model variables) are **read-only** by default — they are provided by the simulation engine and cannot be reassigned.

🛠️ How to Fix It:

If `L_load` is a **parameter or input port**, you **cannot assign a value** to it like this:

```c
L_load = L_new;  // ❌ Not allowed if L_load is read-only
```

Instead, do one of the following:

This is a parameter value - and therefore I am unable to change the value

The code could not be more wrong- everything is zero in the beginning - and even after the transition time - when it should be well above it- There is nothing - the system is not working at all-

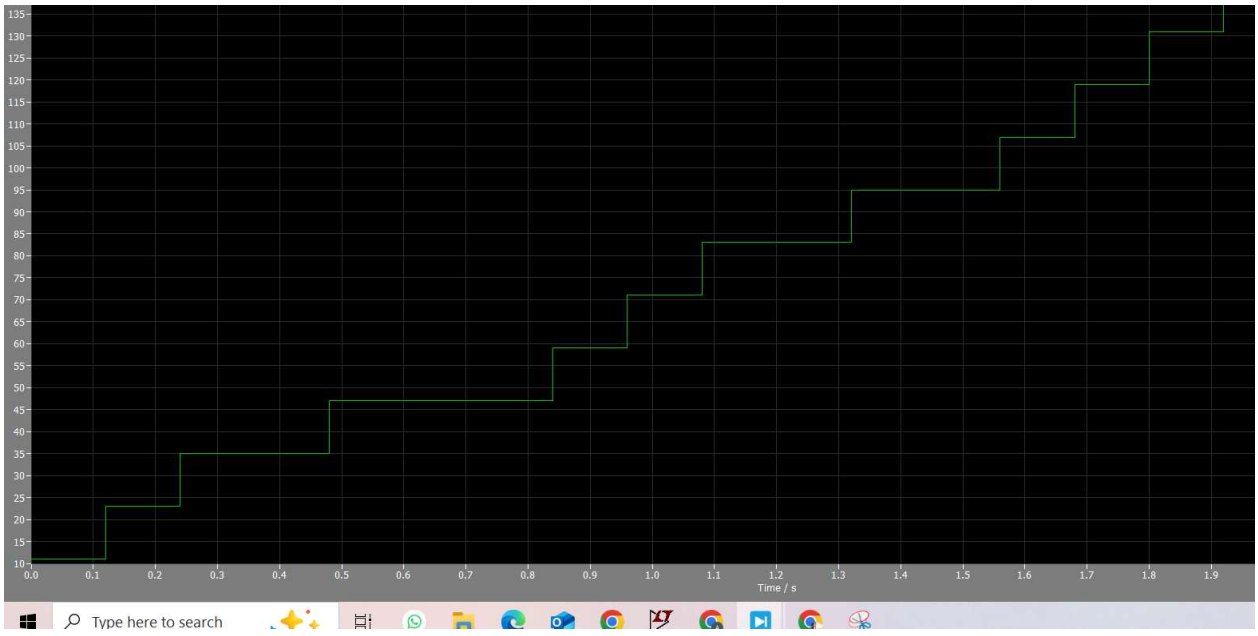
The first step which is the actual_sample_time is its

PS - **I actually forgot to map them to the output signal**

The initial values are all correct - it is just that they are not getting updated - which is Something we have to look at

The reason I am not seeing 3e-10 increments is because at that instant itself - that loop is running for multiple loops and not 1 - so there is a 3ze-10 increment - z is the number of repetitions in the loops.

I figure it is the loops which themselves are running farr too much
The sample time % is not uniform as I had intended



THis is an issue when I was trying to implement it with the code FMODof actual sample time and current time

Becuase it does not exactly yield to 0 - inherent property of the FMOD function

Therefore the advice was to use
static int sample_count = 0;

```
static double last_sample_time = 0;

double sample_interval = Actual_Sample_time;

if (CurrentTime - last_sample_time >= sample_interval - 1e-12) {


   // Update last_sample_time accurately using counting, not actual clock
   sample_count++;
   last_sample_time = sample_count * sample_interval;
}
```

Update it like this - this way that issue will never arrive.

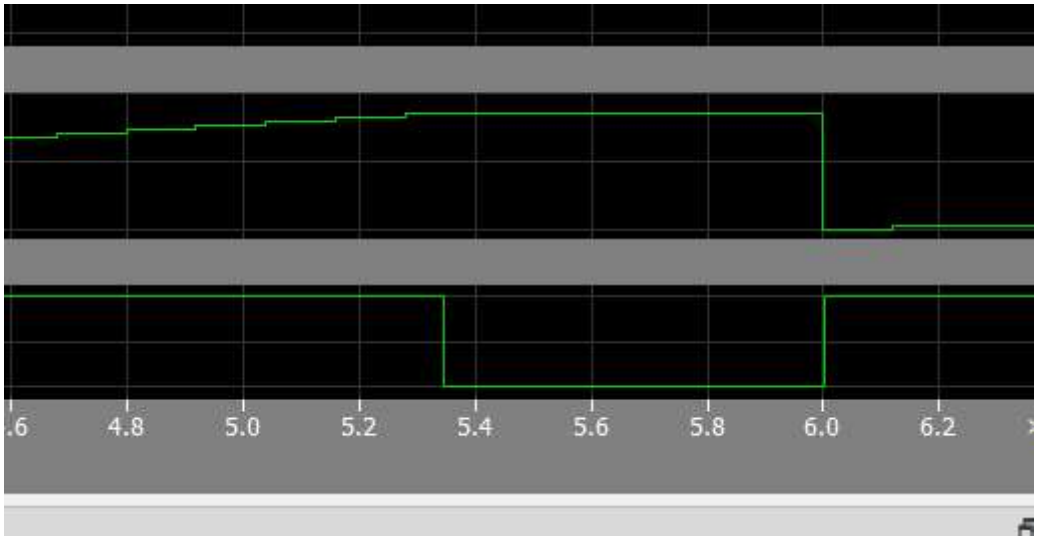
In the code I notice that First_time variable is not getting updated and therefore - it always stays in the first if else block itself - it never proceeds to consider the 2nd or the 3rd at all.

We need to look at the reason for it.

So I think the final update is not happening because the Td!= dead_time condition is itself not being evaluated.

Count variable is not increasing which means the code of Soft_switch is not being evaluated - or if it is then not getting updated

It is going inside the function - though it is not getting updated - that is something we have to see

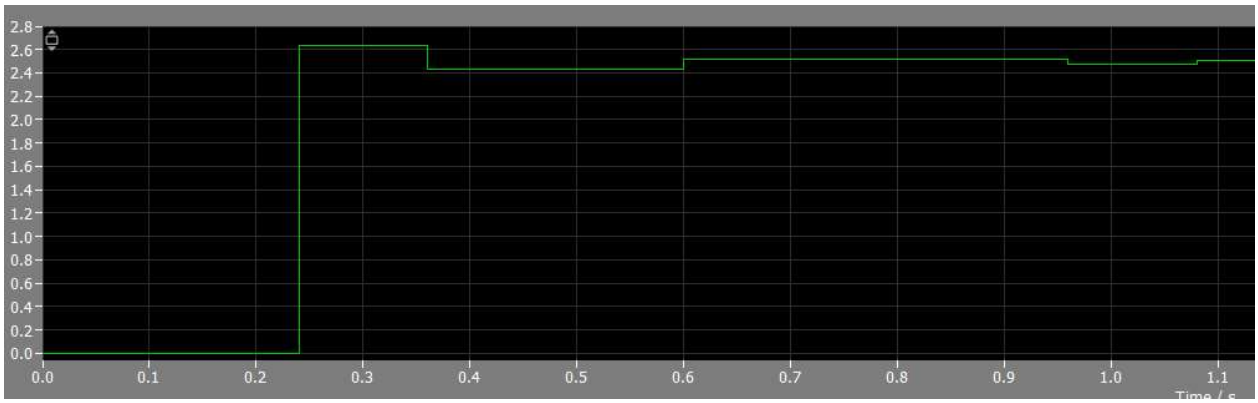


Dead Time is randomly becoming 0 at 6e-3 time interval which makes no sense but here we are.

And I say literal 0 not even 1e-10 which means there is some other gibberish going on

I tried changing the 1e-12 to 2e-12 and nothing happened.

It is like totally random



Now , the thing is - It should not be changing till 1e-3 but it is - that means the condition is being violated - the T_transition wali cheez

We have to get it to work as well

The reason you're seeing `V_feedback_old` being assigned **before** `1e-3` **seconds**, even though your code intends for it not to, likely comes down to this line:

```c
if (trigger > 0.5 && 1e-3 < CurrentTime - T_transition < 2e-3)
```

This line **does not do what you expect in C**. Here's why:

This line:

```c
if (trigger > 0.5 && 1e-3 < CurrentTime - T_transition < 2e-3)
```

is **not** interpreted in C as:

> "Check if `CurrentTime - T_transition` is between `1e-3` and `2e-3`."

Instead, it's interpreted as:

```c
if (trigger > 0.5 && ((1e-3 < CurrentTime - T_transition) < 2e-3))
```

This happens because C evaluates `1e-3 < CurrentTime - T_transition` as a boolean, resulting in either `0` or `1`. Then it checks whether that boolean value is less than `2e-3`, which is **almost always true** if the first part evaluates true.

So even if `CurrentTime - T_transition < 1e-3`, this condition might still evaluate to true and let `V_feedback_old = feedback(...)` execute.

✅ Solution: Split the condition

Use the proper syntax:

```c
if (trigger > 0.5 && (CurrentTime - T_transition > 1e-3) && (CurrentTime - T_transition < 2e-3))
```

This way, you're explicitly checking both ends of the time window.

Did a complete simulation run - Found certain caveats in the code

1) When I am transitioning the load - I see that 1) The V_feedback_old is not even updating after that at all - which makes no sense to me - I mean they wont be giving the same value will they?