PROJECT

Analysis and Design of Push-Pull DC-DC Converter

Rajat Sankhla Renuka

Group 7 Rajat (213079008)
Renuka (203070043)

Design of Push-Pull topology of DC-DC Converter

Introduction:

A dc-dc converter is an electromechanical device which either levels up or levels down the input dc voltage.

Depending upon the technique used they can be divided into linear and switched.

The switched-mode converter uses highly efficient switches operated at high frequency to do this task. Due to their high-frequency operation, they can be designed very compact which makes their application open to almost every gadget out in the market which works on a dc source.

The linear converter is also known as the linear power supply is not as highly efficient as the switched-mode but uses very simple circuits and lesser components which makes them very economical for low-cost applications.

Depending upon the application they can either step up or step down the input voltage level. Due to their wide range of applications and regular advances in the semiconductor industry research is always going on to make them efficiently drive a wide range of loads. They are even be used to isolate high voltage and low voltage circuits which makes them even useful for microprocessor applications that work on low voltage levels but may require integration with higher voltage circuits.

Why use the isolated type converter?

In some applications, we may require control of higher voltage circuits from the low voltage side which may require a person to do the task. Safety of personnel is of utmost importance in such cases so we had to have some isolation strategy which can be thought of by creating a separate ground on either side.

Depending upon the application they are integrated to there can be a variety of topologies available in the market our concern is to design one such topology for our study purpose.

Group 7

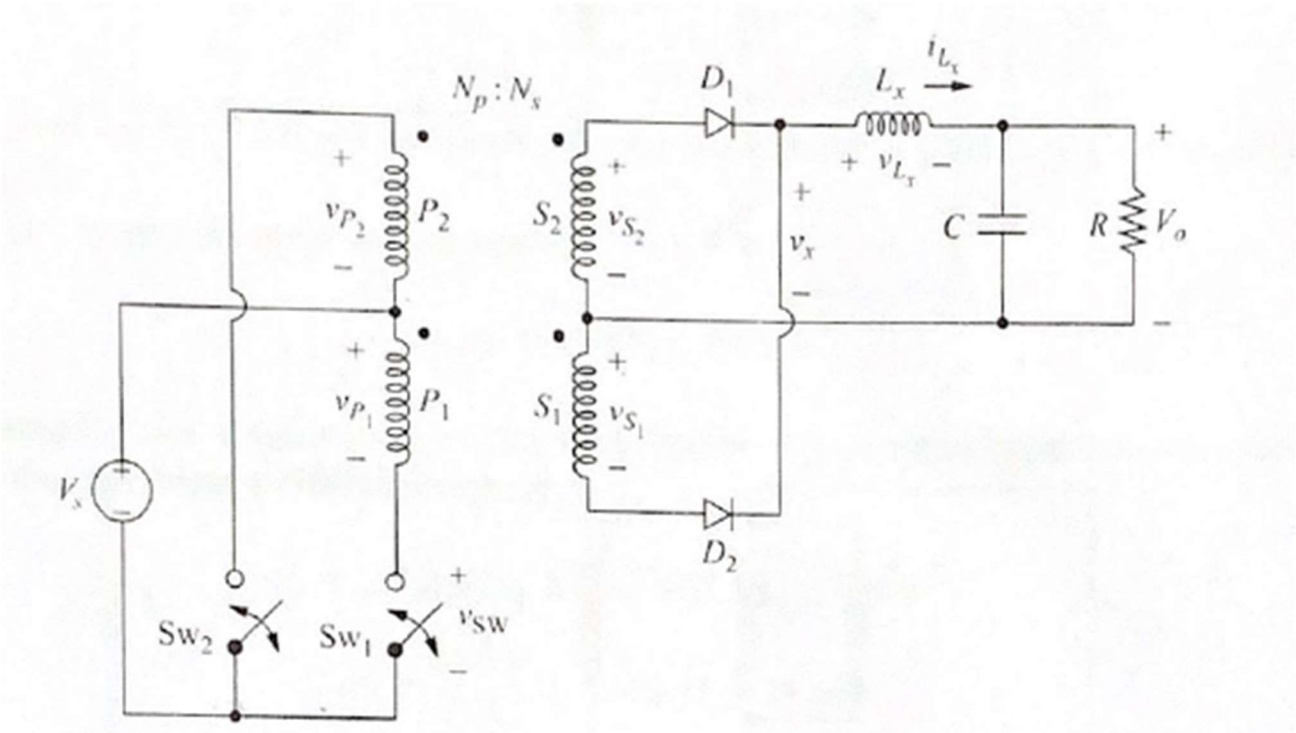
Rajat (213079008)
Renuka (203070043)

Objective:

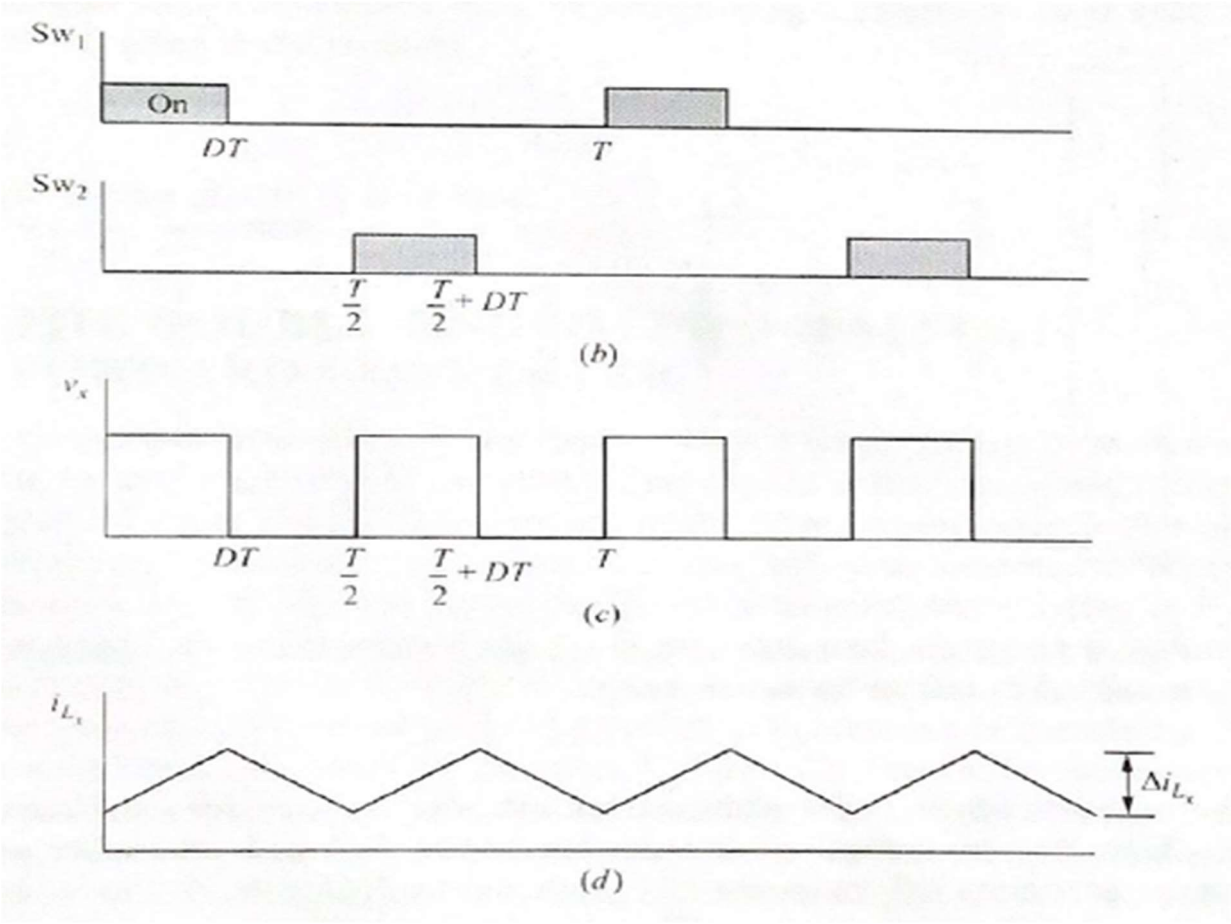
To design and analyse the PUSH-PULL topology of the DC-DC converter.

Why the name push-pull?

The term *push-pull* is sometimes used to generally refer to any converter with bidirectional excitation of the transformer. For example, in a full-bridge converter, the switches (connected as an H-bridge) alternate the voltage across the supply side of the transformer, causing the transformer to function as it would for AC power and produce a voltage on its output side. However, *push-pull* more commonly refers to a two-switch topology with a split primary winding.



Rajat (213079008)
Renuka (203070043)



Rajat (213079008)
Renuka (203070043)

1.

Assumptions

① Voltage drops across switches during condition is Zero
② Inductor is ideal (No parasetic resistance or DCR)
③ Forward voltage drop of diodes = 0.
④ Steady state operation & Continuous conduction mode (I_O = const)

Q1.)

The analysis is done with either of the switches closed and with both switches open.

Closing SW_1 and SW_2 as open

$$V_{P_1} = V_S$$

Due to transformer action

$$V_{S_1} = V_S\left(\frac{N_S}{N_P}\right)$$

$$V_{S_2} = V_S\left(\frac{N_S}{N_P}\right)$$

$$V_{P_2} = V_S$$

$$V_{SW_2} = 2V_S$$

Diode D_1 → Forward biased.
Diode D_2 → Reverse biased

$$V_R = V_{S_2} = V_S\,\frac{N_S}{N_P}$$

$$V_{L_x} = V_{L_x} - V_O = V_S\left(\frac{N_S}{N_P}\right) - V_O \quad \left[\, 0 \le t \le T_{ON} \,\right] \left(T_{ON} = DT \right.$$

Rajat (213079008)
Renuka (203070043)

Both switches open

When SW_1 is opened, a small delay is given before closing SW_2 so to avoid shorting of supply voltage via conduction of both SW_1 and SW_2, this is termed as dead time.

During dead time inductor releases its energy to the load resulting in D_1 & D_2 becoming forward biased.

As the source is disconnected so $V_{S_1} = V_{S_2} = 0$

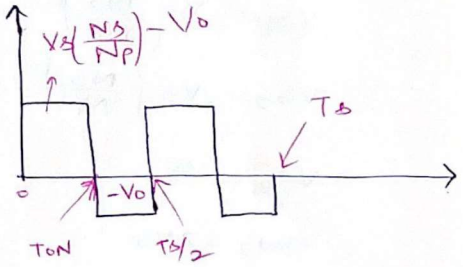
and $V_x = 0$

$$\boxed{V_{Lx} = V_x - V_o = -V_o} \quad \left[T_{ON} \leq t \leq \frac{T_S}{2}\right]$$

For steady state conditions,

$$\langle V_{Lx} \rangle_{Avg} = 0 \quad \left\{\text{Voltage across the inductor at the beginning and end of 1 cycle will be same}\right\}$$

$$\left(V_S\left(\frac{N_S}{N_P}\right) - V_o\right)\frac{T_{ON}}{T_S/2} = V_o\left(\frac{T_S}{2} - T_{ON}\right)\frac{1}{T_S/2}$$



V_L

$$\frac{T_{ON}}{T_S} = D$$

$$\left(V_S\left(\frac{N_S}{N_P}\right) - V_o\right)(2D) = V_o(1 - 2D)$$

$$2D\left(V_S\left(\frac{N_S}{N_P}\right) - V_o\right) = V_o - 2V_oD$$

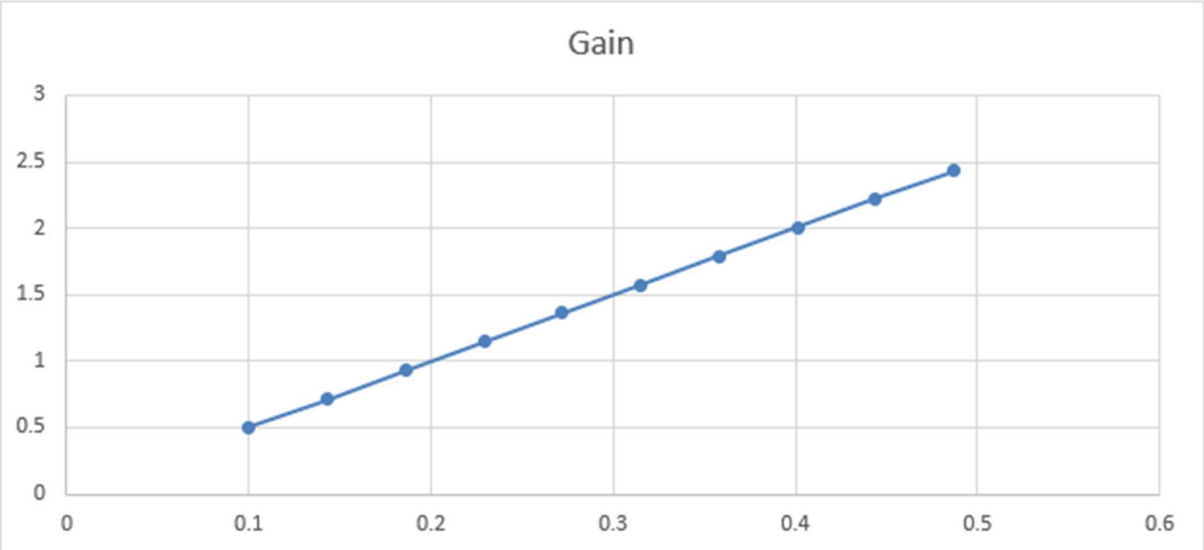
$$2D\,V_S\left(\frac{N_S}{N_P}\right) = V_o$$

$$\frac{V_o}{V_S} = 2D\frac{N_S}{N_P}$$

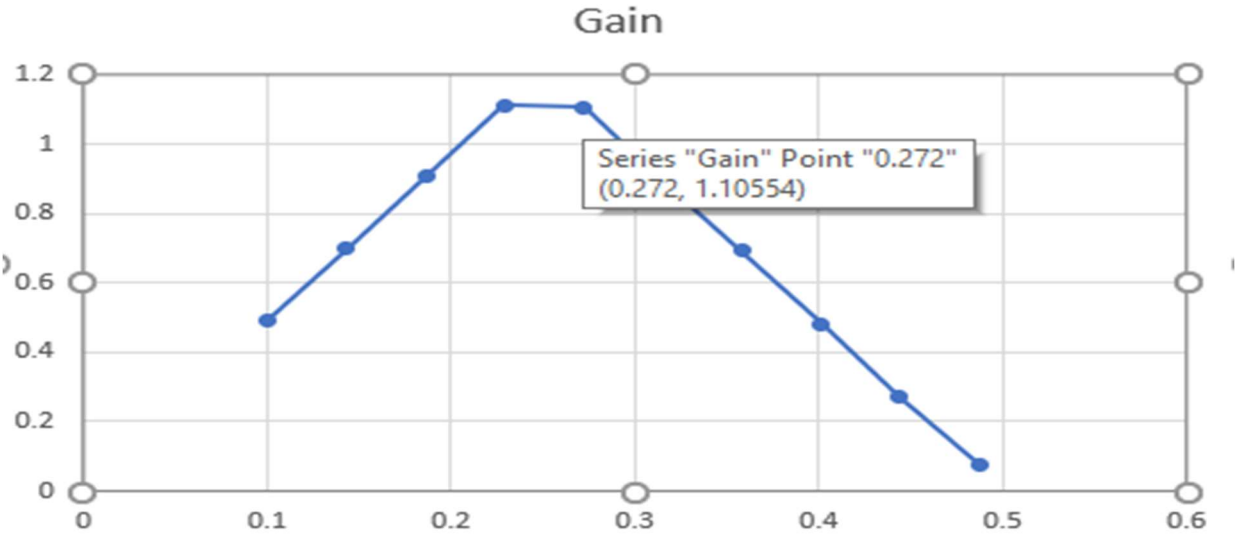
$$\boxed{G_1 = 2D\frac{N_S}{N_P}}$$

Rajat (213079008)
Renuka (203070043)

Value Plot



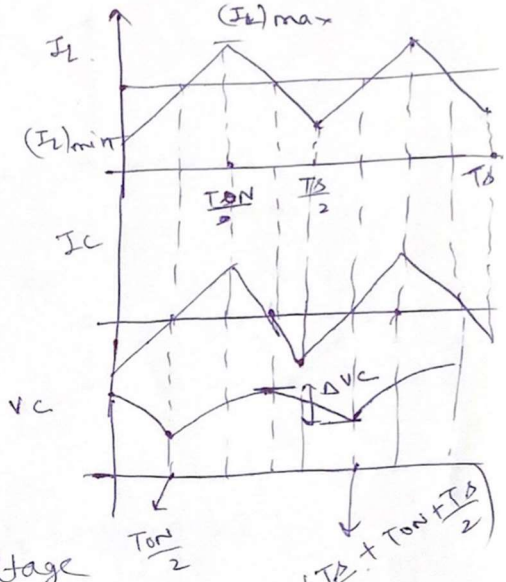
Simulation Results



Rajat (213079008)
Renuka (203070043)

2.

Q2. From the Analysis done in Ques 1 we have drawn waveforms of V_L, I_L, I_C and V_C, lets Reproduce some of those here directly.



$$\langle I_L \rangle_{Avg} = (I_L)_{min} + \frac{1}{2}\left(\frac{T_S}{2}\right)\left(\frac{\Delta I_L}{T_S/2}\right)$$

$$\langle I_L \rangle_{Avg} = (I_L)_{min} + \frac{\Delta I_L}{2}$$

$$\langle I_L \rangle_{Avg} = (I_L)_{min} + \frac{(I_L)_{max} - (I_L)_{min}}{2}$$

$$\boxed{\langle I_L \rangle_{Avg} = \frac{(I_L)_{max} + (I_L)_{min}}{2}}$$

Lets approximate the capacitor voltage waveform as triangular shape

$$\langle V_C \rangle_{Avg} = (V_C)_{min} + \frac{1}{2}\left(\frac{\frac{T_S}{2} + \frac{T_{ON}}{2} - \frac{T_{ON}}{2}}{T_S/2}\right)(\Delta V_C)$$

$$\langle V_C \rangle_{Avg} = (V_C)_{min} + \frac{\Delta V_C}{2}$$

$$\boxed{\langle V_C \rangle_{Avg} = \frac{(V_C)_{max} + (V_C)_{min}}{2}}$$

Rajat (213079008)
Renuka (203070043)

3.

Q3.

Considering 1st Half of Switching cycle.

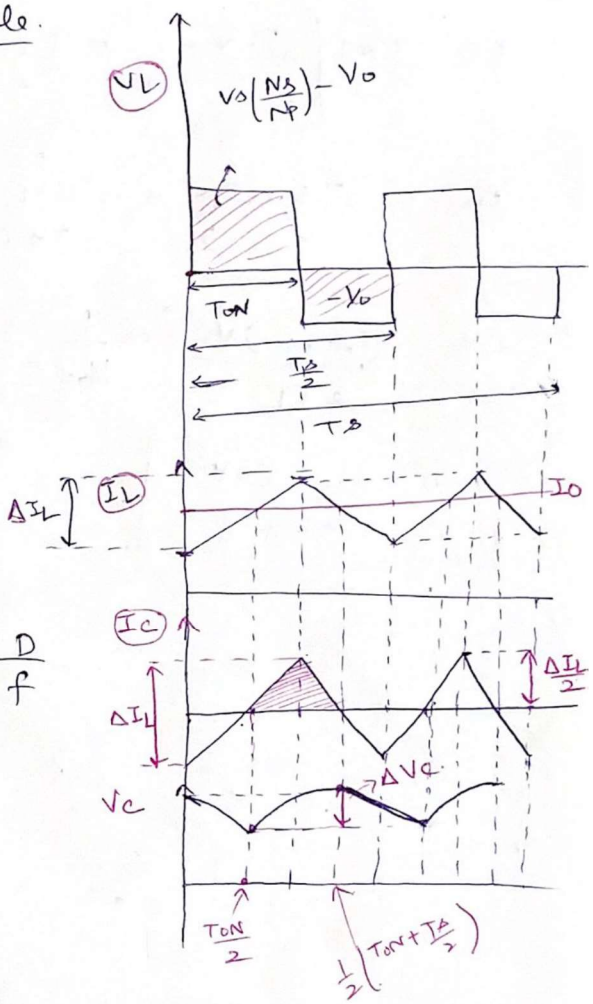
$$V_L = L \frac{dI_L}{dt}$$

$$\int_{(I_L)_{min}}^{(I_L)_{max}} dI_L = \frac{V_L}{L} \int_0^{T_{ON}} dt$$

$$\int_{(I_L)_{min}}^{(I_L)_{max}} dI_L = \left(V_s\left(\frac{N_s}{N_p}\right) - V_o\right) \cdot \frac{1}{L} \cdot \left[T_{ON}\right]$$

$$(I_L)_{max} - (I_L)_{min} = \left(\frac{V_o}{2D} - V_o\right) \cdot \frac{1}{L} \frac{D}{f}$$

$$\boxed{\Delta I_L = \frac{(1-2D)V_o}{2fL}}$$



$I_L = I_c + I_o$ is valid when either of switches are closed and when both are open.

$$I_c = I_L - I_o$$

$$C \frac{dV_c}{dt} = I_c$$

$$\int_{V_{c(min)}}^{V_{c(max)}} dV_c = \frac{1}{C} \int_{T_{ON}/2}^{\frac{1}{2}\left(T_{ON} + \frac{T_s}{2}\right)} I_c \, dt$$

$$\int_{V_{cmin}}^{V_{c(max)}} dV_c = \frac{1}{C}\left(\text{Area under the curve}\right)$$

Rajat (213079008)
Renuka (203070043)

$$\Delta V_C = \frac{1}{C} \times \frac{1}{2}\left[\left(\frac{T_{ON}}{2} + \frac{T_S}{4} - \frac{T_{ON}}{2}\right)\right] \frac{\Delta I_L}{2}$$

$$\Delta V_C = \frac{T_S \Delta I_L}{8 C \times 2} = \frac{\Delta I_L}{8fC} \times \frac{1}{2} = \frac{\Delta I_L}{16fC}$$

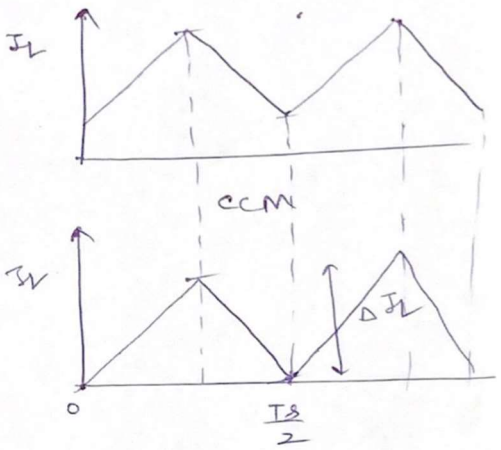
$$\Delta V_C = \frac{(1-2D)V_O}{2fL} \times \frac{1}{16fC}$$

$$\boxed{\Delta V_C = \frac{(1-2D)V_O}{32f^2LC}}$$

4.

Qy.) From the analysis performed till Now we can produce current (Inductor's) directly below.

At Boundary condition.



$$I_L(t=0) = I_L\left(t=\frac{T_S}{2}\right) = 0$$

$$\langle I_L \rangle_{Avg} = \frac{\frac{1}{2}\left(\frac{T_S}{2}\right)\left(\Delta I_L\right)}{T_S/2} = \frac{\Delta I_L}{2}$$

$$\langle I_L \rangle_{Avg} = \frac{\Delta I_L}{2}$$

From Ques 3 we can write $\Delta I_L = \frac{(1-2D)V_o}{2fL}$.

$$\langle I_L \rangle_{Avg} = \frac{(1-2D)V_o}{4fL}$$

$$\langle I_L \rangle_{Avg} = \frac{V_o}{R}$$

$$\frac{(1-2D)V_o}{4fL} = \frac{V_o}{R}$$

At DCM ;

$$\langle I_L \rangle_{Avg} < \frac{\Delta I_L}{2}$$

$$\frac{(1-2D)V_o}{4fL} > \frac{V_o}{R}$$

$$L_{crit} < \frac{(1-2D)R}{4f}$$

For CCM

$$\langle I_L \rangle_{Avg} > \frac{\Delta I_L}{2}$$

$$\frac{V_o}{R} > \frac{(1-2D)V_o}{4fL}$$

$$L_{crict} > \frac{(1-2D)R}{4f}.$$

Also. If we are varying 'R' then,

$$R_{critical} < \frac{4fL}{1-2D}$$

5.

Q5) Requirements

Converter must operate in CCM at Buck as well as Boost operation.

Given $\frac{\Delta I_L}{I_o} < 20\%$; $\frac{\Delta V_c}{V_o} < 5\%$

Assumptions – Consider all elements ideal.

$f = 100KHz$; $V_s = 400V$; $V_{out} = V_o = 600V$ (Boost) & $200V$ (Buck)

P_o output Power $P_o = 5kW$ (Buck as well as Boost)

→ L & C.

→ If we find Design Parameters for Boost operation it will also work for Buck operation.

→ Practically the maximum value of duty ratio D is 0.5, this is done to maintain a small blanking time to avoid turning both the switches on simultaneously.

$$\boxed{0 < D < 0.5}$$

with the help of this lets design the turns ratio. $\left(\frac{N_S}{N_P}\right)$

WKT $V_o = 2D\frac{N_S}{N_P} \cdot V_s$

For Boost condn $V_o = 600V$; $V_s = 400V$

$$D = \frac{V_o}{V_s}\frac{N_P}{N_S} \cdot \frac{1}{2} = \frac{600}{400} \cdot \frac{N_P}{N_S} \cdot \frac{1}{2} = 0.75\frac{N_P}{N_S}$$

$$0 < 0.75\frac{N_P}{N_S} < 0.5$$

$$\boxed{\frac{N_S}{N_P} > 1.5}$$

Same Analysis for Buck converter gives

$$D = \frac{200}{400}\cdot\frac{N_p}{N_s}\cdot\frac{1}{2} = \frac{N_p}{N_s}\cdot\frac{1}{4}$$

$$0 < D < 0.5$$

$$\boxed{\frac{N_s}{N_p} > 2}$$

So for opⁿ of Both Buck & Boost condⁿ we have to take $\frac{N_s}{N_p} > 2$

Let take $\boxed{\frac{N_s}{N_p} = 2.5}$

In the further design $\frac{N_s}{N_p} = 2.5$ will be taken.

Now as explained earlier, value of L & C calculated for Boost operation will also work for Buck condⁿ.
Hence lets proceed.

Let our converter give Buck operation at $D = 0.1$ & Boost operation at $D = 0.3$

$$P_{out} = 5000W = V_o I_o$$

$$(I_o)_{Boost} = \frac{5000}{600} = 8.33A.$$

$$\frac{\Delta I_L}{I_o} < 20\%$$

$$\Delta I_L < 0.2 \times 8.33$$

$$\Delta I_L < 1.666$$

$$\frac{(1-2D)V_o}{2fL} < 1.666$$

Rajat (213079008)
Renuka (203070043)

$$\frac{1-2(0.3)(600)}{2\times 100\,KHz\times 1.666} < L$$

$$\boxed{L > 720.28\,MH}$$

With ripple conditions.

$$\frac{\Delta V_c}{V_o} < 0.05$$

$$\Delta V_c < 0.05 \times 600$$

$$\Delta V_c < 30$$

$$\frac{\Delta V_c}{V_o} = \frac{(1-2D)}{32LCf^2}$$

$$\frac{1-2D}{32LCf^2} < 0.05$$

$$C > \frac{(1-2D)}{32\times 0.05\,Lf^2}$$

$$C > \frac{1-2(0.3)}{32\times 0.05\times 720.28H\times 10^{10}}$$

$$\boxed{C > 34.70\,nF}$$

If we do similar calculations for Buck

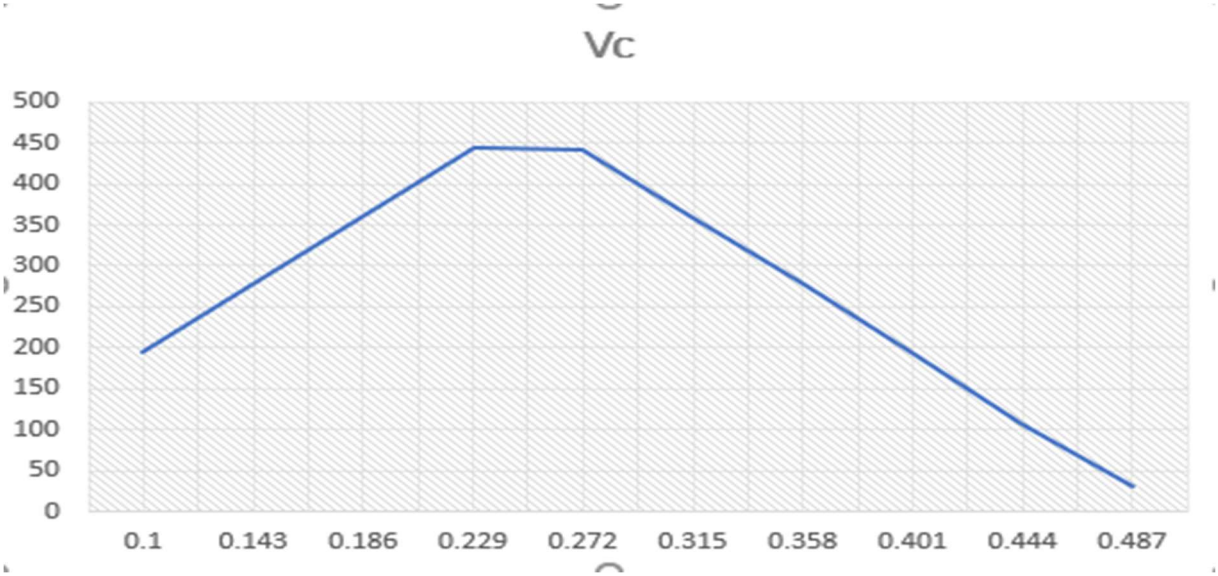
$$L > 160\,MH \text{ or } C > 312.5nF$$

For dual operation of converters with considering ripple conditions we take both the values higher, so for our practical design we will chose

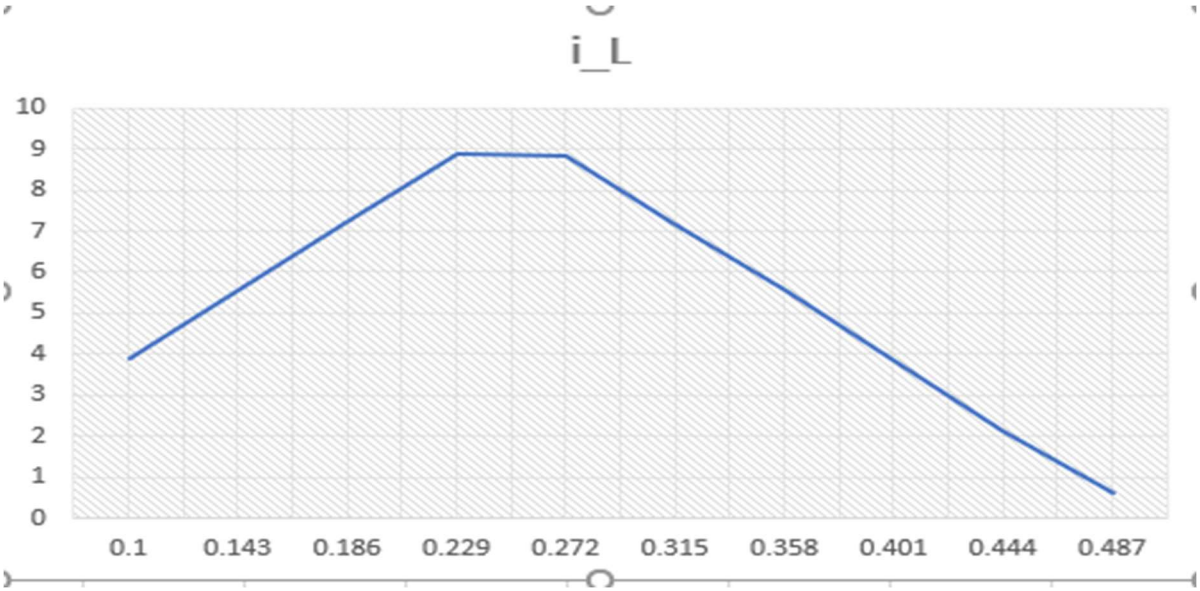
$$\boxed{L > 720MH \text{ & } C > 312.5nF}$$

Rajat (213079008)
Renuka (203070043)

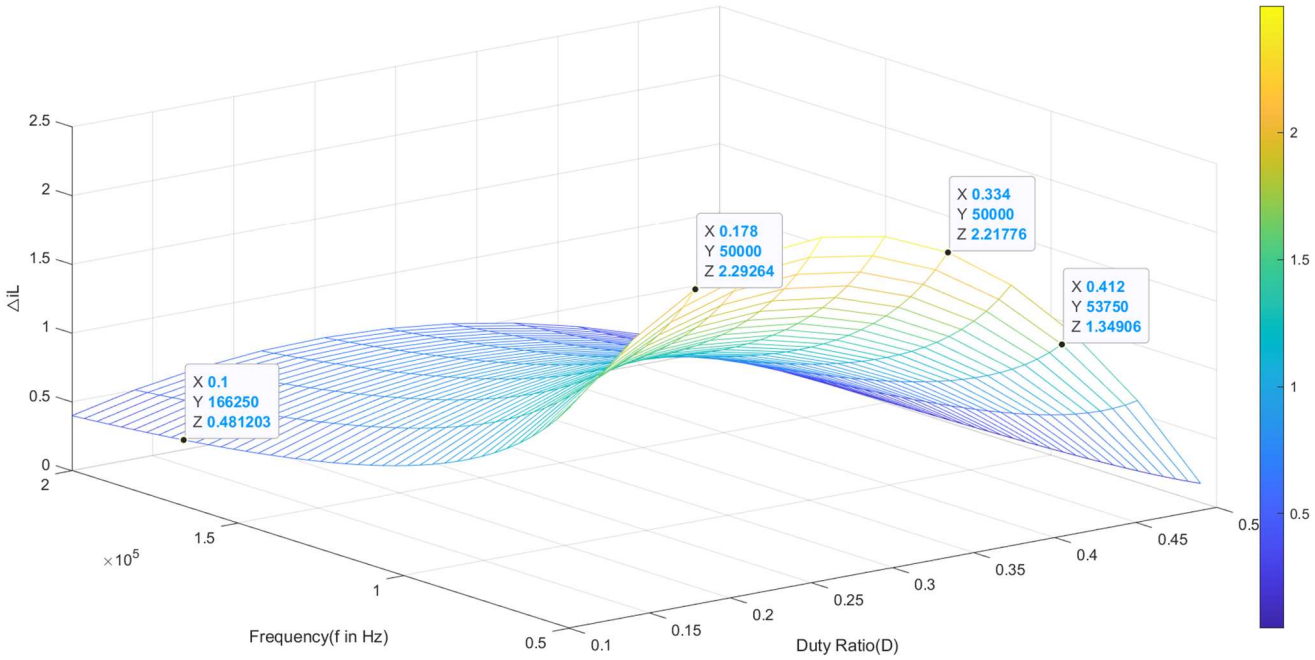
6. Vc V/s Duty Ratio

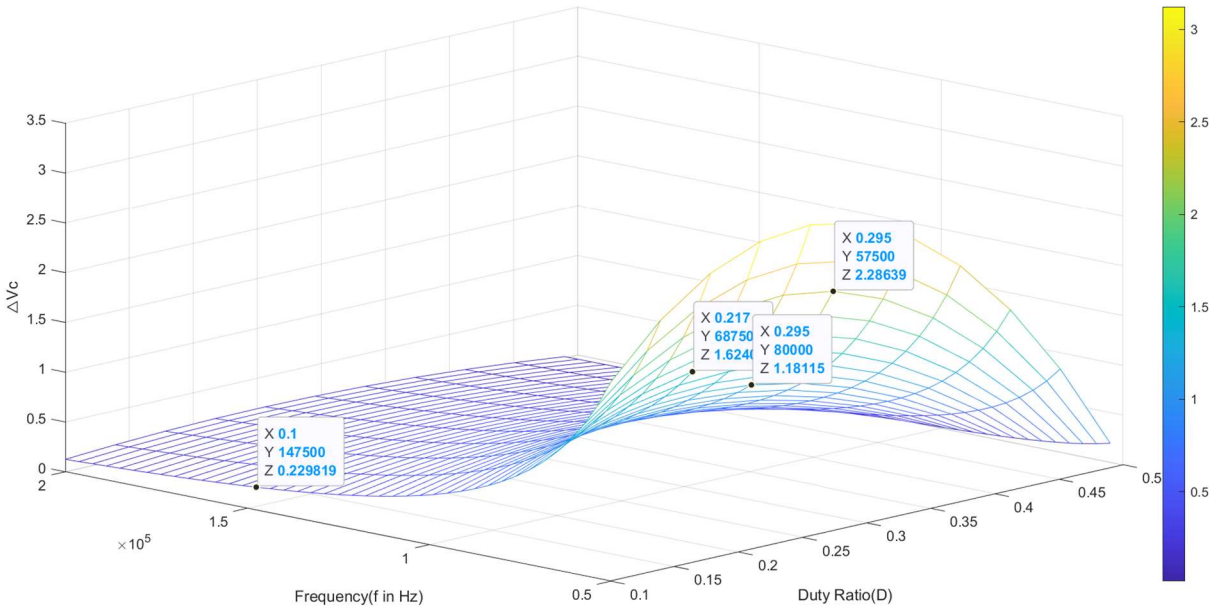


iL V/s Duty Ratio



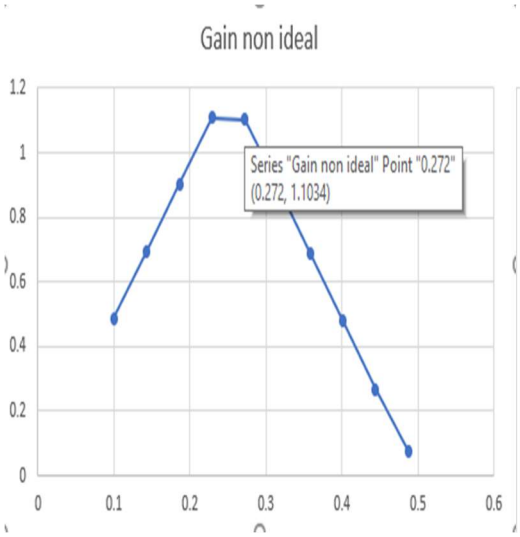
7.


X 0.1
Y 166250
Z 0.481203
X 0.178
Y 50000
Z 2.29264
X 0.334
Y 50000
Z 2.21776
X 0.412
Y 53750
Z 1.34906
ΔiL
×10^5
Frequency(f in Hz)
Duty Ratio(D)

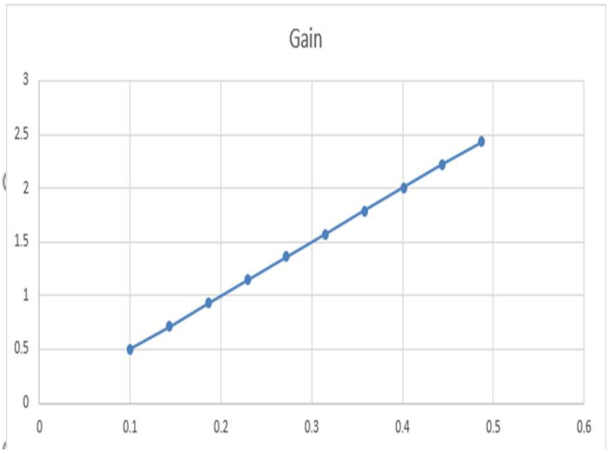

X 0.1
Y 147500
Z 0.229819
X 0.217
Y 68750
Z 1.624
X 0.295
Y 80000
Z 1.18115
X 0.295
Y 57500
Z 2.28639
ΔVc
×10^5
Frequency(f in Hz)
Duty Ratio(D)

8. **Gain V/S Duty ratio**

Non-Ideal case



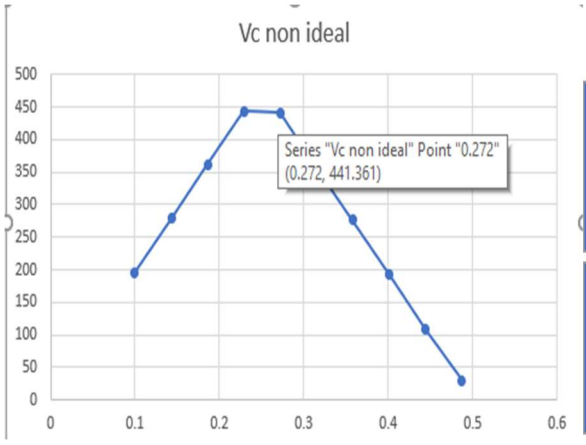
Ideal case



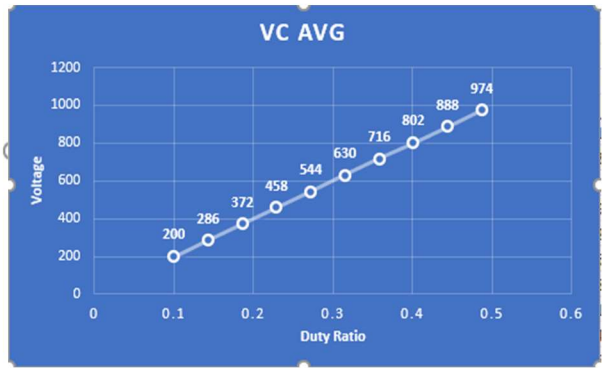
It is not possible to take all ideal components while simulating in LT spice so while considering non-linearity in only inductance we can see that at the same value of duty ratio, the gain of the converter is less. The difference in values of gain will be more visible if we could simulate only non-linearity in inductance and rest switches and diode are assumed ideal.

Vc V/S Duty

Non ideal case



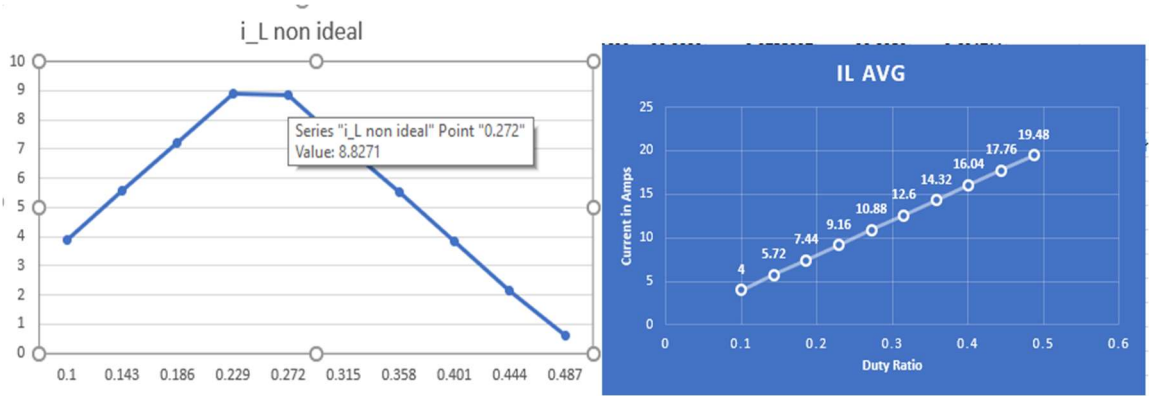
ideal case



The output voltage is less at the same duty ratio in the case of non-idealistic design, again the difference is not that much visible due to simulation constraints. But the curve of the non-ideal case will be lower than that of the ideal case.

iL Average V/S Duty

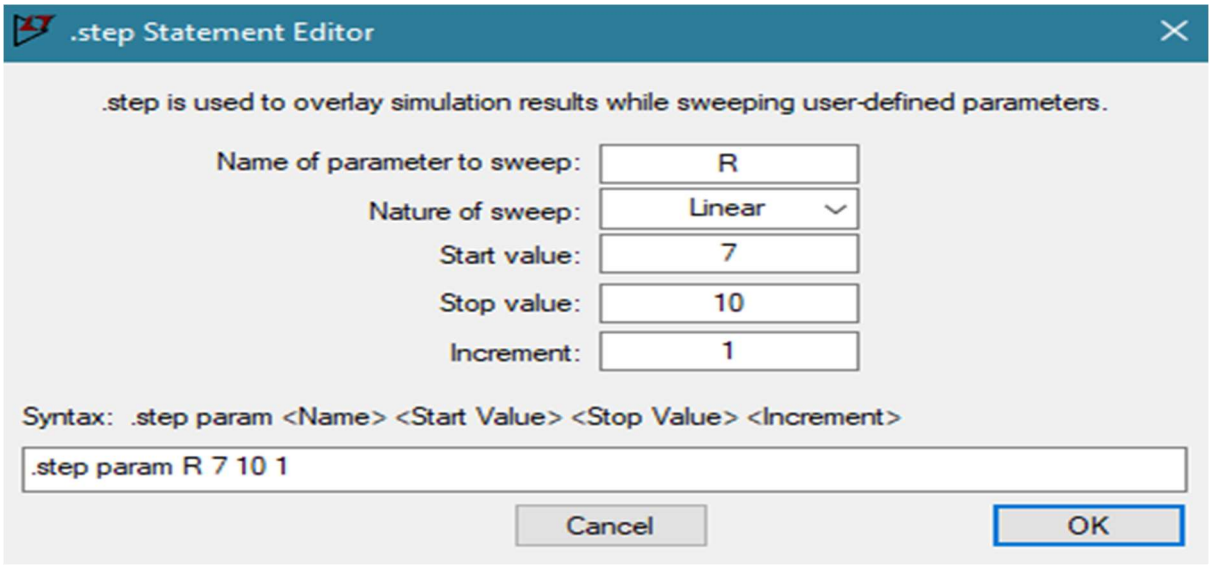
Non ideal case Ideal case



Inductor current was less in at same duty ratio in the case of the converter with a non-ideal inductor. The difference in values would be more evident if the larger nonlinearity is considered.

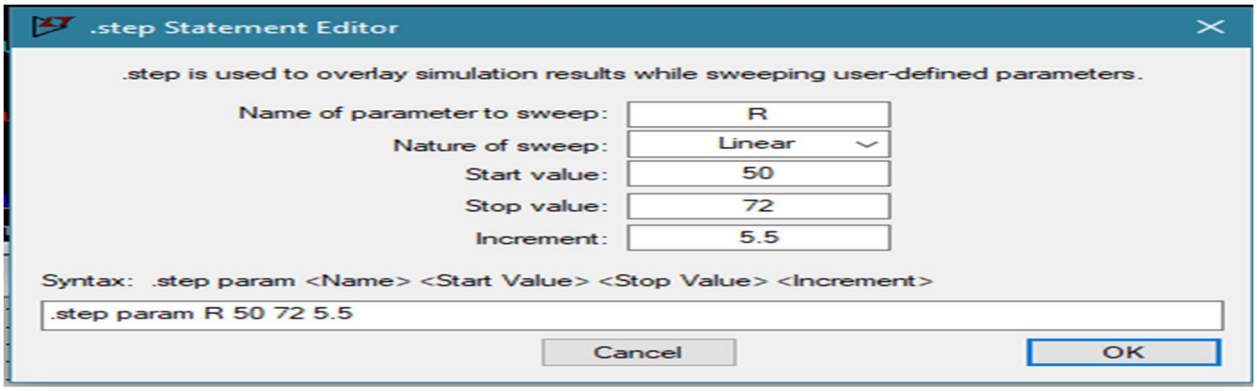
9. For Buck operation

```
Measurement: pout
  step            AVG(v(vout)*i(r1))  FROM            TO
     1            5656.5              0               0.01
     2            4987.78             0               0.01
     3            4493.66             0               0.01
     4            4091.09             0               0.01

Measurement: pin
  step            AVG(v(vin)*(-i(v1)))                FROM
     1            7698.02             0               0.01
     2            6047.9              0               0.01
     3            5406.51             0               0.01
     4            4890.29             0               0.01

Measurement: efficiency
  step            pout/pin
     1            0.734799
     2            0.824713
     3            0.831156
     4            0.836574
```



As we reach the rated value of load current for which the converter is designed, efficiency increases.

For Boost Operation



SPICE Error Log: C:\Users\Rajat Sankhla\Documents\LtspiceXVII\push_pull_final.log

```
easurement: pout
 step            AVG(v(vout)*i(r1))  FROM            TO
    1            7044.84             0               0.01
    2            6378.83             0               0.01
    3            5828.8              0               0.01
    4            5370.48             0               0.01
    5            4972.76             0               0.01

easurement: pin
 step            AVG(v(vin)*(-i(v1)))                FROM
    1            7908.36             0               0.01
    2            7582.35             0               0.01
    3            7007.39             0               0.01
    4            5927.88             0               0.01
    5            5580                0               0.01

easurement: efficiency
 step            pout/pin
    1            0.890809
    2            0.841274
    3            0.831809
    4            0.90597
    5            0.891177

easurement: average_value
 step            AVG(v(vout))        FROM            TO
    1            593.5               0               0.01
```

Variation of Load resistance up to the load consuming rated power

As we are moving away from the rated load for which the converter is designed, efficiency decreases.

10. How it is different from the Buck-Boost converter?

Features:

Unlike buck-boost converter, it uses a transformer to transfer energy from source to load in both the half cycles except a small dead-time which eliminates the need for loop stabilization circuits that are needed in buck-boost converters to maintain the load at

constant voltage when the load is disconnected from the source for a long time at higher duty ratios. So, if the input to the transformer is well regulated the push-pull converter doesn't require any loop stabilization circuits and hence it is **simple** in design.

When compared with its counterparts performing similar operations it is **efficient** because it delivers energy to the load in both the half cycles due to which peak current in the switches and windings is slightly higher than load side so losses are less.

This topology efficiently uses the transformer core by energising the core in both directions due to which magnetic field intensity and field density both crosses zero which facilitates a fresh demagnetised core for each operation which allows using smaller transformers in compact designs.

It is **immune** to electromagnetic transients.

Limitations:

a. Care has to be taken in providing switching pulses to the switches to prevent turning on both the switches at the same time because this will cancel the flux in the transformer making it draw larger currents which can damage the switches. We can ensure that both switches don't turn on together by providing a dead time in between operations and following break before make a strategy.

b. For each cycle of operation, the stress on switches is very large, it is difficult to design and operate the switch if an application requires higher voltage levels.

   The stress on each switch can be reduced by using multiple switches in series but this will affect the efficiency of the converter as each switch will having a small value of resistance during ON time.

11.

The push-Pull converter seems the best fit for automotive applications where you can utilise its features like the lower size which reduces the space occupied in the car and reduces the car weight. It is desired that an automotive must have a lower form factor as space is premium.

The power supply should be very reliable when you add need advanced features to the automobiles like cruise control and others, we don't want our power supplies to fail during its usage. Now, this can be ensured in the Push-pull topology as it will cancel the increase in flux in the primary winding which may have increased due to the flow of transients because of difference in voltages in primary and secondary sides.

1. EPwm.c file:

```
/*****************************************************************
* File: EPwm.c -- Solution File
* Devices: TMS320F28x7x
* Author: C2000 Technical Training, Texas Instruments
*****************************************************************/

#include "Lab.h"                        // Main include file


/*****************************************************************
* Function: InitEPwm()
*
* Description: Initializes the Enhanced PWM modules on the F28x7x
*****************************************************************/
void InitEPwm(void)
{
asm(" EALLOW");                                 // Enable EALLOW protected
register access

      // Configure the prescaler to the ePWM modules.  Max ePWM input clock is
100 MHz.
      ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV = 1;         // EPWMCLK divider from
PLLSYSCLK.  0=/1, 1=/2

      // Must disable the clock to the ePWM modules if you want all ePWM modules
synchronized.
      CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;

      asm(" EDIS");                             // Disable EALLOW protected
register access


//---------------------------------------------------------------------
//--- Configure ePWM2 to trigger ADC SOCA at a 50 kHz rate
//---------------------------------------------------------------------
      asm(" EALLOW");                                       // Enable EALLOW
protected register access
      DevCfgRegs.SOFTPRES2.bit.EPWM2 = 1;           // ePWM2 is reset
      DevCfgRegs.SOFTPRES2.bit.EPWM2 = 0;           // ePWM2 is released from
reset
      asm(" EDIS");                                     // Disable EALLOW
protected register access

      EPwm2Regs.TBCTL.bit.CTRMODE = 0x3;            // Disable the timer

      EPwm2Regs.TBCTL.all = 0xC033;                 // Configure timer control
register
// bit 15-14     11:     FREE/SOFT, 11 = ignore emulation suspend
// bit 13        0:      PHSDIR, 0 = count down after sync event
// bit 12-10     000:    CLKDIV, 000 => TBCLK = HSPCLK/1
// bit 9-7       000:    HSPCLKDIV, 000 => HSPCLK = EPWMCLK/1
// bit 6         0:      SWFSYNC, 0 = no software sync produced
// bit 5-4       11:     SYNCOSEL, 11 = sync-out disabled
// bit 3         0:      PRDLD, 0 = reload PRD on counter=0
// bit 2         0:      PHSEN, 0 = phase control disabled
// bit 1-0       11:     CTRMODE, 11 = timer stopped (disabled)
```

```
      EPwm2Regs.TBCTR = 0x0000;                     // Clear timer counter
      EPwm2Regs.TBPRD = ADC_SAMPLE_PERIOD;    // Set timer period
      EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;           // Set timer phase

      EPwm2Regs.ETPS.all = 0x0100;                  // Configure SOCA
// bit 15-14     00:     EPWMxSOCB, read-only
// bit 13-12     00:     SOCBPRD, don't care
// bit 11-10     00:     EPWMxSOCA, read-only
// bit 9-8       01:     SOCAPRD, 01 = generate SOCA on first event
// bit 7-4       0000:   reserved
// bit 3-2       00:     INTCNT, don't care
// bit 1-0       00:     INTPRD, don't care

      EPwm2Regs.ETSEL.all = 0x0A00;                 // Enable SOCA to ADC
// bit 15        0:      SOCBEN, 0 = disable SOCB
// bit 14-12     000:    SOCBSEL, don't care
// bit 11        1:      SOCAEN, 1 = enable SOCA
// bit 10-8      010:    SOCASEL, 010 = SOCA on PRD event
// bit 7-4       0000:   reserved
// bit 3         0:      INTEN, 0 = disable interrupt
// bit 2-0       000:    INTSEL, don't care

      EPwm2Regs.TBCTL.bit.CTRMODE = 0x0;            // Enable the timer in count
up mode


//-----------------------------------------------------------------
//--- Configure ePWM1 for 2 kHz symmetric PWM on EPWM1A pin
//-----------------------------------------------------------------
      asm(" EALLOW");                                          // Enable EALLOW
protected register access
      DevCfgRegs.SOFTPRES2.bit.EPWM1 = 1;           // ePWM1 is reset
      DevCfgRegs.SOFTPRES2.bit.EPWM1 = 0;           // ePWM1 is released from
reset
      asm(" EDIS");                                       // Disable EALLOW
protected register access

      EPwm1Regs.TBCTL.bit.CTRMODE = 0x3;            // Disable the timer

      EPwm1Regs.TBCTL.all = 0xC033;                 // Configure timer control
register
// bit 15-14     11:     FREE/SOFT, 11 = ignore emulation suspend
// bit 13        0:      PHSDIR, 0 = count down after sync event
// bit 12-10     000:    CLKDIV, 000 => TBCLK = HSPCLK/1
// bit 9-7       000:    HSPCLKDIV, 000 => HSPCLK = EPWMCLK/1
// bit 6         0:      SWFSYNC, 0 = no software sync produced
// bit 5-4       11:     SYNCOSEL, 11 = sync-out disabled
// bit 3         0:      PRDLD, 0 = reload PRD on counter=0
// bit 2         0:      PHSEN, 0 = phase control disabled
// bit 1-0       11:     CTRMODE, 11 = timer stopped (disabled)

      EPwm1Regs.TBCTR = 0x0000;                     // Clear timer counter
      EPwm1Regs.TBPRD = PWM_HALF_PERIOD;            // Set timer period
      EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;           // Set timer phase


      EPwm1Regs.CMPA.bit.CMPA = PWM_DUTY_CYCLE;     // Set PWM duty cycle
      EPwm1Regs.CMPB.bit.CMPB = PWM_DUTY_CYCLE;   // Set PWM duty cycle
```

```
      EPwm1Regs.CMPCTL.all = 0x0002;               // Compare control register
// bit 15-10     0's:    reserved
// bit 9         0:      SHDWBFULL, read-only
// bit 8         0:      SHDWAFULL, read-only
// bit 7         0:      reserved
// bit 6         0:      SHDWBMODE, don't care
// bit 5         0:      reserved
// bit 4         0:      SHDWAMODE, 0 = shadow mode
// bit 3-2       00:     LOADBMODE, don't care
// bit 1-0       10:     LOADAMODE, 10 = load on zero or PRD match

      EPwm1Regs.AQCTLA.all = 0x0060;         // Action-qualifier control register
A
      EPwm1Regs.AQCTLA.bit.ZRO = 0x0010;
      EPwm1Regs.AQCTLA.bit.PRD = 0x0000;
      EPwm1Regs.AQCTLA.bit.CAU = 0x0001;
      EPwm1Regs.AQCTLA.bit.CAD = 0x0000;
// bit 15-12     0000:     reserved
// bit 11-10     00:       CBD, 00 = do nothing
// bit 9-8       00:       CBU, 00 = do nothing
// bit 7-6       01:       CAD, 01 = clear
// bit 5-4       10:       CAU, 10 = set
// bit 3-2       00:       PRD, 00 = do nothing
// bit 1-0       00:       ZRO, 00 = do nothing

    EPwm1Regs.AQCTLB.all = 0x0060;      // Action-qualifier control register B
    EPwm1Regs.AQCTLB.bit.ZRO = 0x0000;
    EPwm1Regs.AQCTLB.bit.PRD = 0x0010;
    EPwm1Regs.AQCTLB.bit.CBU = 0x0000;
    EPwm1Regs.AQCTLB.bit.CBD = 0x0001;
// bit 15-12     0000:     reserved
// bit 11-10     00:       CBD, 00 = do nothing
// bit 9-8       00:       CBU, 00 = do nothing
// bit 7-6       01:       CAD, 01 = clear
// bit 5-4       10:       CAU, 10 = set
// bit 3-2       00:       PRD, 00 = do nothing
// bit 1-0       00:       ZRO, 00 = do nothing


      EPwm1Regs.AQSFRC.all = 0x0000;         // Action-qualifier s/w force
register
// bit 15-8      0's:    reserved
// bit 7-6       00:     RLDCSF, 00 = reload AQCSFRC on zero
// bit 5         0:      OTSFB, 0 = do not initiate a s/w forced event on output B
// bit 4-3       00:     ACTSFB, don't care
// bit 2         0:      OTSFA, 0 = do not initiate a s/w forced event on output A
// bit 1-0       00:     ACTSFA, don't care

      EPwm1Regs.AQCSFRC.all = 0x0000;        // Action-qualifier continuous s/w
force register
// bit 15-4      0's:    reserved
// bit 3-2       00:     CSFB, 00 = forcing disabled
// bit 1-0       00:     CSFA, 00 = forcing disabled

      EPwm1Regs.DBCTL.bit.OUT_MODE = 0;      // Deadband disabled
      EPwm1Regs.PCCTL.bit.CHPEN = 0;         // PWM chopper unit disabled
      EPwm1Regs.TZDCSEL.all = 0x0000;        // All trip zone and DC compare
actions disabled
```

```
	EPwm1Regs.TBCTL.bit.CTRMODE = 0x2;	// Enable the timer in count up/down
mode

//-----------------------------------------------------------------
//--- Enable the clocks to the ePWM module.
//--- Note: this should be done after all ePWM modules are configured
//--- to ensure synchronization between the ePWM modules.
//-----------------------------------------------------------------
	asm(" EALLOW");                                          // Enable EALLOW
protected register access
	CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;  // TBCLK to ePWM modules enabled
	asm(" EDIS");                                        // Disable EALLOW
protected register access

} // end InitEPwm()


//--- end of file -----------------------------------------------------
```

Buck Converter

2. Lab.h file

```
/**********************************************************************
* File: lab.h
* Device: TMS320F2837xD
* Author: C2000 Technical Training, Texas Instruments
* Description: Include file for workshop lab exercises.  Include this
* file in all C-source files.
**********************************************************************/

#ifndef LAB_H
#define LAB_H


//---------------------------------------------------------------------------
// Constant Definitions
//
#define ADC_BUF_LEN         50              // ADC buffer length
#define ADC_SAMPLE_PERIOD   1999            // 1999 = 50 kHz sampling w/ 100 MHz
ePWM clock
#define PWM_HALF_PERIOD     500             // period/2 for 100 kHz symmetric PWM
w/ 100 MHz ePWM clock
#define PWM_DUTY_CYCLE      109             // 25% duty cycle
#define PWM_MIN_DUTY            450                     // 10% duty cycle for PWM
modulation
#define PWM_MAX_DUTY            50                      // 90% duty cycle for PWM
modulation
#define PWM_STEP                10                        // Step size change for
PWM modulation
#define FILTER_LEN          5               // filter length
#define SINE_PTS            25              // number of point in sine wave


//---------------------------------------------------------------------------
// Include Standard C Language Header Files
```

```
// (Header file <string.h> not supported by CLA compiler)
//
#if !defined(__TMS320C28XX_CLA__)
    #include <string.h>
#endif


//---------------------------------------------------------------------------
// Include any other Header Files
//
#include "F2837xD_Cla_typedefs.h"        // CLA type definitions
#include "F2837xD_device.h"              // F2837xD header file peripheral address
definitions
#include "F2837xD_Adc_defines.h"         // ADC definitions
#include "F2837xD_defaultisr.h"          // ISR definitions
#include "F2837xD_Pie_defines.h"         // PIE definitions


//---------------------------------------------------------------------------
// Function Prototypes
//
extern void AdcSetMode(Uint16, Uint16, Uint16);
extern void CalAdcINL(Uint16);
extern void DelayUs(Uint16);
extern void InitAdca(void);
extern void InitCla(void);
extern void InitDacb(void);
extern void InitDma(void);
extern void InitECap(void);
extern void InitEPwm(void);
extern void InitFlash(void);
extern void InitGpio(void);
extern void InitPieCtrl(void);
extern void InitSysCtrl(void);
extern void InitWatchdog(void);
extern void InitXbar();
extern void SetDBGIER(Uint16);
extern void UserInit(void);


//---------------------------------------------------------------------------
// CLA Function Prototypes
//
extern interrupt void Cla1Task1();
extern interrupt void Cla1Task2();
extern interrupt void Cla1Task3();
extern interrupt void Cla1Task4();
extern interrupt void Cla1Task5();
extern interrupt void Cla1Task6();
extern interrupt void Cla1Task7();
extern interrupt void Cla1Task8();


//---------------------------------------------------------------------------
// Global symbols defined in the linker command file
//
extern Uint16 cla1Funcs_loadstart;
extern Uint16 cla1Funcs_loadsize;
extern Uint16 cla1Funcs_runstart;
```

```
extern Uint16 secureRamFuncs_loadstart;
extern Uint16 secureRamFuncs_loadsize;
extern Uint16 secureRamFuncs_runstart;
extern Uint16 Cla1Prog_Start;


//---------------------------------------------------------------------------
// Global Variables References
//
extern float32 xDelay[FILTER_LEN];
extern float32 coeffs[FILTER_LEN];
extern Uint16 AdcBuf[ADC_BUF_LEN];
extern Uint16 AdcBufFiltered[ADC_BUF_LEN];
extern Uint16 AdcBufRaw[2*ADC_BUF_LEN];
extern Uint16 ClaFilteredOutput;
extern Uint16 DacOffset;
extern Uint16 DacOutput;
extern Uint32 PwmDuty;
extern Uint32 PwmPeriod;
extern Uint16 AdcResult;
extern Uint16 DacData;
extern Uint16 SineData;
extern Uint16 DEBUG_TOGGLE;
extern Uint16 SINE_ENABLE;
extern Uint16 PWM_MODULATE;
extern int QuadratureTable[SINE_PTS];
extern const struct PIE_VECT_TABLE PieVectTableInit;    // PieVectTableInit is
always extern


//---------------------------------------------------------------------------
// Macros
//

// The following pointer to a function call calibrates the ADC reference,
// DAC offset, and internal oscillators
#define Device_cal (void   (*)(void))0x070282

// The following pointers to functions calibrate the ADC linearity.  Use this
// in the AdcSetMode(...) function only
#define CalAdcaINL (void   (*)(void))0x0703B4
#define CalAdcbINL (void   (*)(void))0x0703B2
#define CalAdccINL (void   (*)(void))0x0703B0
#define CalAdcdINL (void   (*)(void))0x0703AE

// The following pointer to a function call looks up the ADC offset trim for a
// given condition. Use this in the AdcSetMode(...) function only.
#define GetAdcOffsetTrimOTP (Uint16 (*)(Uint16 OTPoffset))0x0703AC


//---------------------------------------------------------------------------
#endif  // end of LAB_H definition


//--- end of file -----------------------------------------------------
```

Boost Converter

3. Lab.h file

```
/**********************************************************************
* File: lab.h
* Device: TMS320F2837xD
* Author: C2000 Technical Training, Texas Instruments
* Description: Include file for workshop lab exercises.  Include this
* file in all C-source files.
**********************************************************************/

#ifndef LAB_H
#define LAB_H


//---------------------------------------------------------------------------
// Constant Definitions
//
#define ADC_BUF_LEN         50              // ADC buffer length
#define ADC_SAMPLE_PERIOD   1999            // 1999 = 50 kHz sampling w/ 100 MHz
ePWM clock
#define PWM_HALF_PERIOD     500             // period/2 for 100 kHz symmetric PWM
w/ 100 MHz ePWM clock
#define PWM_DUTY_CYCLE      307             // 25% duty cycle
#define PWM_MIN_DUTY            450                     // 10% duty cycle for PWM
modulation
#define PWM_MAX_DUTY            50                      // 90% duty cycle for PWM
modulation
#define PWM_STEP                10                        // Step size change for
PWM modulation
#define FILTER_LEN          5               // filter length
#define SINE_PTS            25              // number of point in sine wave


//---------------------------------------------------------------------------
// Include Standard C Language Header Files
// (Header file <string.h> not supported by CLA compiler)
//
#if !defined(__TMS320C28XX_CLA__)
    #include <string.h>
#endif


//---------------------------------------------------------------------------
// Include any other Header Files
//
#include "F2837xD_Cla_typedefs.h"       // CLA type definitions
#include "F2837xD_device.h"             // F2837xD header file peripheral address
definitions
#include "F2837xD_Adc_defines.h"        // ADC definitions
#include "F2837xD_defaultisr.h"         // ISR definitions
#include "F2837xD_Pie_defines.h"        // PIE definitions


//---------------------------------------------------------------------------
// Function Prototypes
//
```

```
extern void AdcSetMode(Uint16, Uint16, Uint16);
extern void CalAdcINL(Uint16);
extern void DelayUs(Uint16);
extern void InitAdca(void);
extern void InitCla(void);
extern void InitDacb(void);
extern void InitDma(void);
extern void InitECap(void);
extern void InitEPwm(void);
extern void InitFlash(void);
extern void InitGpio(void);
extern void InitPieCtrl(void);
extern void InitSysCtrl(void);
extern void InitWatchdog(void);
extern void InitXbar();
extern void SetDBGIER(Uint16);
extern void UserInit(void);


//---------------------------------------------------------------------------
// CLA Function Prototypes
//
extern interrupt void Cla1Task1();
extern interrupt void Cla1Task2();
extern interrupt void Cla1Task3();
extern interrupt void Cla1Task4();
extern interrupt void Cla1Task5();
extern interrupt void Cla1Task6();
extern interrupt void Cla1Task7();
extern interrupt void Cla1Task8();


//---------------------------------------------------------------------------
// Global symbols defined in the linker command file
//
extern Uint16 cla1Funcs_loadstart;
extern Uint16 cla1Funcs_loadsize;
extern Uint16 cla1Funcs_runstart;
extern Uint16 secureRamFuncs_loadstart;
extern Uint16 secureRamFuncs_loadsize;
extern Uint16 secureRamFuncs_runstart;
extern Uint16 Cla1Prog_Start;


//---------------------------------------------------------------------------
// Global Variables References
//
extern float32 xDelay[FILTER_LEN];
extern float32 coeffs[FILTER_LEN];
extern Uint16 AdcBuf[ADC_BUF_LEN];
extern Uint16 AdcBufFiltered[ADC_BUF_LEN];
extern Uint16 AdcBufRaw[2*ADC_BUF_LEN];
extern Uint16 ClaFilteredOutput;
extern Uint16 DacOffset;
extern Uint16 DacOutput;
extern Uint32 PwmDuty;
extern Uint32 PwmPeriod;
extern Uint16 AdcResult;
extern Uint16 DacData;
```

```
extern Uint16 SineData;
extern Uint16 DEBUG_TOGGLE;
extern Uint16 SINE_ENABLE;
extern Uint16 PWM_MODULATE;
extern int QuadratureTable[SINE_PTS];
extern const struct PIE_VECT_TABLE PieVectTableInit;    // PieVectTableInit is
always extern


//---------------------------------------------------------------------------
// Macros
//

// The following pointer to a function call calibrates the ADC reference,
// DAC offset, and internal oscillators
#define Device_cal (void   (*)(void))0x070282

// The following pointers to functions calibrate the ADC linearity.  Use this
// in the AdcSetMode(...) function only
#define CalAdcaINL (void   (*)(void))0x0703B4
#define CalAdcbINL (void   (*)(void))0x0703B2
#define CalAdccINL (void   (*)(void))0x0703B0
#define CalAdcdINL (void   (*)(void))0x0703AE

// The following pointer to a function call looks up the ADC offset trim for a
// given condition. Use this in the AdcSetMode(...) function only.
#define GetAdcOffsetTrimOTP (Uint16 (*)(Uint16 OTPoffset))0x0703AC


//---------------------------------------------------------------------------
#endif  // end of LAB_H definition


//--- end of file -----------------------------------------------------
```