Appendix A

Codes for Co-Optimization and Design Process

This Appendix presents the codes for the design process discussed in Chapter 4. In this appendix, first the MATLAB script for designing and co-optimization process of magnetics and circuits is presented. 50000 desings are generated, analysed and saved in an initial file called ”AllDesigns.txt”. In the design process a function called OptimizeT is used for optimizing the circuit and calculating the required inductance. OptimizeT is using other functions in itself: ScaleT for calculating the scaling factor of the transistor, EstLosses for estimating the losses inside the transistor and the diode, FindInductor for finding the required inductance for the circuit at the operating frequency. Finally a MATLAB script is presented for filtering the designs based on the process presented in Section 4.2.6. Among the scripts and functions presented here, functions OptimizeT, ScaleT, EstLosses, and FindInductor are originally written by the circuit team; see Acknowledgments.

```
clear all;
close all;
tic;

format shortG

```

```
%%%%%%%%%%%%%%%%%%%
% CONSTANTS
%%%%%%%%%%%%%%%%%%%

mu0=4*pi*1e-7;              % Permeability of Air
epsilon0=8.85e-12;          % Permittivity of Air
mu=mu0*1;                   % Permeability of Inductor Core

%%%%%%%%%%%%%%%%%%%%%%%%
% CONSTANT and FIXED PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%

Vin_V=100;                  % Input Voltage [V]
Vled_V=35.8;                % LED Voltage [V]
Power_W=25;                 % LED Power Delivery Goal [W]
Height=1000e-6;             % Height of Inductor [m]
ConductorThickness=30e-6;
SigmaCopper=5.8e7;                          % Conductivity of Copper at 25 C [S/m]
SigmaCopper=SigmaCopper/(1+0.0039*(100));   % Conductivity at 125 C [S/m]
WindingGap=100e-6; % Gap between the winding [m]
X0=Height/2;                % Half of the Inductor Height [m]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generating Random Designs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for iRand=1:50000                       % Defining Number of Random Designs
```

```
Ro=3.5e-3+rand*(4e-3);            % Defining Outer
    Radius Range [m]
MinFeatureSize=90e-6+rand*80e-6;  % Defining Minimum
    Feature Size Range [m]

Freq_MHz=10+rand*20;              % Defining Frequency
    Range [MHz]
Freq_Hz=Freq_MHz*1e6;             % Frequency [Hz]
Omega=2*pi*Freq_Hz;

Cind_F=5e-12+rand*(5e-12);        % Defining Inductor
    Capacitance Range [F]

% The inductor Height can be either defined as a random
    variable or as a fixed parameter. Random Variable:
    Height=500e-6+rand*500e-6; X0=Height/2;

Cind_pF=Cind_F/1e-12;
Cind_F=Cind_F;

[ scaleT, P_W, PDiode_W, PSwitch_W, L_H, Ton_s,
    Vswitch_V, Iswitch_A, Vdiode_V, Idiode_A, Vind_V,
    Iind_A, T_s] =...
     OptimizeT( Vin_V, Vled_V, Power_W, Freq_Hz,
        @MITHEMT_Coss, @STPS10170C_Cj, Cind_F, 0.9, .3,
        .69, .5);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Designing The Inductor Ro and Ri
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

L_nH=L_H/1e-9;      % Required Inductance from Optimization Code
Ri=Ro/4;            % Initial Value for Inner Radius
RoOvRi=Ro/Ri;       % Initial Value Ro Over Ri Ratio
for RoOvRiStep=1:10
    RoOvRi = pi*Ro*sqrt(2)*sqrt(mu0*Height*log(RoOvRi))/((MinFeatureSize+GapRi)*sqrt(L_nH*1e-9*pi));
end

Ri=abs(Ro/RoOvRi);                              % Inner Radius of the Inductor [m]
N=round(2*pi*Ri/(MinFeatureSize+GapRi));        % Number of Turns for Toroidal Inductor
MinFeaturesize=2*pi*Ri/N-WindingGap;

TopLength=Ro-Ri;
BottomAngle=2*pi/N;
BottomLength=sqrt(Ri^2+Ro^2-2*Ro*Ri*cos(BottomAngle));

Length=(BottomLength+TopLength+2*Height)*N; % Total Length of Windings

LRecal=(1/2)*N^2*mu0*Height*log(Ro/Ri)/pi;  % Recalculating the value of the inductance to check if the solution has converged

ErrorInL=(LRecal-L_H)/L_H*100;                  % Error between the Recalculated Inductance and Required
```

```
    Inductance.


LengthOver2=((2*pi*((Ro+Ri)/2)/N)-WindingGap)/2; % Half of Conductor Width (Y/2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Voltage and Current Waveforms at the Inductor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V0=(max(Vind_V)-min(Vind_V))/2;                        % Amplitude of First Harmonic of Voltage Waveform
IMaxAC=(max(Iind_A(:,1))-min(Iind_A(:,1)))/2;          % Amplitude of First Harmonic of Current Waveform
I0DC=(max(Iind_A(:,1))+min(Iind_A(:,1)))/2;            % Amplitude of the DC Component of the Current Waveform

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

IMax=1;                                                % Amplitude of a 1-Amper single harmonic current

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating AC Loss By Energy Models + Deffusion Equations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Defining Solved Free Paramters Equations:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Asinyc=(-(2*X0+pi*WindingGap)/(2*WindingGap));
Asinyt= (12.0*WindingGap^2*(-8*WindingGap^3+2*X0*pi*WindingGap^2+...
    pi^2*WindingGap^3+2*pi*X0^3+pi^2*X0^2*WindingGap)/(pi*...
    (-12*pi*X0^5+5*pi^3*X0^3*WindingGap^2+pi^2*X0^2*WindingGap^3+...
    6*pi*X0^3*WindingGap^2-18*X0*pi*WindingGap^4-24*WindingGap^5+...
    3*pi^2*WindingGap^5+2*pi^3*X0^5+pi^4*X0^2*WindingGap^3+...
    pi^4*X0^4*WindingGap+3*pi^3*X0*WindingGap^4)));
Bsinxl=(-48.0*WindingGap^3*X0^2/(pi*(-12*pi*X0^5+5*pi^3*X0^3*...
    WindingGap^2+pi^2*X0^2*WindingGap^3+6*pi*X0^3*WindingGap^2 -...
    18*X0*pi*WindingGap^4-24*WindingGap^5+3*pi^2*WindingGap^5+...
    2*pi^3*X0^5+pi^4*X0^2*WindingGap^3+pi^4*X0^4*WindingGap+...
    3*pi^3*X0*WindingGap^4)));
betaly = (1.0*pi/X0);
betatx =(1.0*pi/WindingGap);

Ralpha=(1/2)*sqrt(2)*sqrt(Omega*mu0*SigmaCopper);
RalphaL=Ralpha*2*LengthOVer2;
RalphaT=Ralpha*ConductorThickness;

```

```
k0=Asinyt*pi/((-2*Asinyt*pi*(-1+exp(-betatx*ConductorThickness)) ...
    /(betatx*WindingGap)+(2*(-Bsinxl*pi+betaly*LengthOver2+Bsinxl*...
    pi*exp(-betaly*LengthOver2)))/(betaly*X0))*WindingGap);
DenkL=(-2*Asinyt*pi*(-1+exp(-betatx*ConductorThickness))/...
    (betatx*WindingGap)+(2*(-Bsinxl*pi+betaly*LengthOver2+...
    Bsinxl*pi*exp(-betaly*LengthOver2)))/(betaly*X0))*X0;
RalphaBTh=Ralpha*ConductorThickness;

LossShortEdges= -(1/4)*Ralpha*k0^2*(2*cos(RalphaL)*sinh(RalphaL)+...
    2*cosh(RalphaL)*sin(2*Ralpha*LengthOver2)+sinh(4*Ralpha*...
    LengthOver2)+sin(4*Ralpha*LengthOver2))*(-1+exp(-2*betatx*...
    ConductorThickness))/((sinh(2*Ralpha*LengthOver2)^2*...
    cos(RalphaL)^2+cosh(2*Ralpha*LengthOver2)^2*...
    sin(RalphaL)^2)*SigmaCopper*betatx);

LossLongEdge= -(1/16)*Ralpha*(-Bsinxl^2*pi^2+4*Bsinxl*pi+...
    Bsinxl^2*pi^2*exp(-2*betaly*LengthOver2)-4*Bsinxl*pi*...
```

```
        exp(-betaly*LengthOver2)+2*log(exp(-betaly*LengthOver2)))*...
        (exp(4*RalphaT)*cos(RalphaT)^2+exp(4*RalphaT)+8*cos(RalphaT)*...
        sin(RalphaT)*exp(2*RalphaT)-cos(Ralpha*ConductorThickness)^2-...
        1+exp(4*RalphaT)*sin(RalphaT)^2-sin(RalphaT)^2)*exp(-2*RalphaT)/...
        ((sinh(RalphaBTh)^2*cos(RalphaBTh)^2+cosh(RalphaBTh)^2*...
        sin(RalphaBTh)^2)*DenkL^2*SigmaCopper*betaly);

    ACLossForOneAmperSingleHarmonicPerMeter=LossShortEdges+LossLongEdge;
    Rac=ACLossForOneAmperSingleHarmonicPerMeter/(0.5*IMax^2)*Length; % IMax =1

    TotalACLoss=Rac*(IMaxAC^2*0.5); % Using Inductor Current Waveform

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculating DC Loss
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    SF=1+(pi*Ri/N/(Ro-Ri))^2;
    RDC=1/SigmaCopper*N^2/ConductorThickness*((Height-2*ConductorThickness)/(2*pi*Ri-N*WindingGap)+(Height-2*ConductorThickness)/(2*pi*Ro-N*WindingGap)+SF/pi*log((2*pi*(Ro)-N*WindingGap)/(2*pi*(Ri)-N*WindingGap)));
    LossDC=RDC*I0DC^2;

```

```
TotalLossOnGlass=PDiode_W+PSwitch_W+LossDC+TotalACLoss;
CircuitEfficiencyGlass=25/(25+TotalLossOnGlass);

AreaOfInductor=pi*Ro^2;
InductorPowerDensityWPmSq=25/AreaOfInductor;

Qac=L_H*Omega/Rac;

% Operating Frequency
InitialDesignTable(iRand,1)=Freq_MHz;

% Designed Inductor Geometry
InitialDesignTable(iRand,2)=N;
InitialDesignTable(iRand,3)=Ri/1e-3;
InitialDesignTable(iRand,4)=Ro/1e-3;
InitialDesignTable(iRand,5)=Height/1e-6;
InitialDesignTable(iRand,6)=ConductorThickness/1e-6;
InitialDesignTable(iRand,7)=WindingGap/1e-6;
InitialDesignTable(iRand,8)=BottomLength;
InitialDesignTable(iRand,9)=TopLength;
InitialDesignTable(iRand,10)=MinFeaturesize;

% Optimized Transistor Area:
InitialDesignTable(iRand,11)=scaleT;

% Design Material Properties
InitialDesignTable(iRand,12)=mu/mu0; % Core Relative
    Permeability
InitialDesignTable(iRand,13)=SigmaCopper;

```

```
    % R, L, C Values
    InitialDesignTable(iRand,14)=L_nH;
    InitialDesignTable(iRand,15)=RDC;
    InitialDesignTable(iRand,16)=Rac;
    InitialDesignTable(iRand,17)=Qac;
    InitialDesignTable(iRand,18)=ErrorInL;
    InitialDesignTable(iRand,19)=Cind_pF;

    % Waveforms at The Inductor
    InitialDesignTable(iRand,20)=V0;
    InitialDesignTable(iRand,21)=IMaxAC;
    InitialDesignTable(iRand,22)=I0DC;

    % Inductor, Switch and Diode Loss
    InitialDesignTable(iRand,23)=TotalACLoss;
    InitialDesignTable(iRand,24)=LossDC;
    InitialDesignTable(iRand,25)=LossDC+TotalACLoss;
    InitialDesignTable(iRand,26)=PDiode_W;
    InitialDesignTable(iRand,27)=PSwitch_W;
    InitialDesignTable(iRand,28)=P_W;
    InitialDesignTable(iRand,29)=TotalLossGlass;

    % Circuit Efficiency and Inductor Power Density
    InitialDesignTable(iRand,30)=CircuitEfficiencyGlass;
    InitialDesignTable(iRand,31)=InductorPowerDensityWPmSq;

end

RandomSortedLossGlass=sortrows(InitialDesignTable,29);
```

```
dlmwrite('AllDesigns.txt', RandomSortedLossGlass, 'delimiter', '\t', ...
    'precision', 5);

toc

function [scaleT, P_W, PDiode_W, PSwitch_W, L_H, Ton_s, Vswitch_V, Iswitch_A, Vdiode_V, Idiode_A, Vind_V, Iind_A, T_s] =...
OptimizeT(Vin_V, Vled_V, Power_W, Freq_Hz, Coss_F, Cj_F, Cind_F, Rsw_Ohm, TranCRes_Ohm, DiodeDrop_V, DiodeCRes_Ohm)
%  Inputs
%  Vin_V - scalar - Input supply voltage [Volts]
%  Vled_V - scalar - LED string voltage [Volts]
%  Power_W - scalar - LED power delivery goal [Watts]
%  Freq_Hz - scalar - Operating frequency goal [Hz]
%  Cj_F - function - Incremental diode capacitance versus reverse voltage
%  [Volts to Farads]
%  Coss_F - function - Incremental transistor output capacitance versus
%  drain source voltage [Volts to Farads]
%  Cind_F - scalar - Inductor capacitance to ground [Farads]
%  Rsw_Ohm - scalar - Transistor on resistance [Ohms]
%  TranCRes_Ohm - scalar - resistance in series with the FET capacitance [Ohms]
%  DiodeDrop_V - scalar - diode voltage drop [Volts]
```

```
%   DiodeCRes_Ohm - scalar - resistance in series with the diode capacitance [Ohms]
%
%   Outputs
%   scaleT - scalar - optimum scaling factor of the transistor die
%   P_W - scalar - amount of power lost in both the diode and transistor
%   PDiode_W - scalar - power loss in the diode
%   PSwitch_W - scalar - power loss in the transistor
%   L_H - scalar - required inductor value [Henries]
%   Ton_s - scalar - on time [seconds]
%   Vswitch_V - vector - switch drain voltage [Volts]
%   Vdiode_V - vector - diode reverse voltage [Volts]
%   Iind_A - matrix - inductor current - first column is inductive
%   current, second column is capacitive current from Cind [Amperes]
%   Idiode_A - matrix - inductor current - first column is forward
%   current, second column is capacitive current from Cj [Amperes]
%   Iswitch_A - matrix - switch current - first column is on-state
%   current, second column is capacitive current from Coss [Amperes]
%   T_s - vector - time [seconds]

[scaleT,P_W] = fminbnd(@(x) ScaleTLoss( Vin_V, Vled_V, Power_W, Freq_Hz, Coss_F, Cj_F, Cind_F, Rsw_Ohm,
```

```
TranCRes_Ohm, DiodeDrop_V, DiodeCRes_Ohm, x),0.01,10);
[~, PDiode_W, PSwitch_W, L_H, Ton_s, Vswitch_V, Iswitch_A, Vdiode_V, Idiode_A, Vind_V, Iind_A, T_s] = ScaleTLoss(Vin_V, Vled_V, Power_W, Freq_Hz, Coss_F, Cj_F, Cind_F, Rsw_Ohm, TranCRes_Ohm, DiodeDrop_V, DiodeCRes_Ohm, scaleT);

end

function [P_W, PDiode_W, PSwitch_W, L_H, Ton_s, Vswitch_V, Iswitch_A, Vdiode_V, Idiode_A, Vind_V, Iind_A, T_s] = ScaleTLoss(Vin_V, Vled_V, Power_W, Freq_Hz, Coss_F, Cj_F, Cind_F, Rsw_Ohm, TranCRes_Ohm, DiodeDrop_V, DiodeCRes_Ohm, scaleT)
% ESTLOSSES Finds the losses total from scaling the transistor using the proper inductor.
%
%   Inputs
%   Vin_V - scalar - Input supply voltage [Volts]
%   Vled_V - scalar - LED string voltage [Volts]
%   Power_W - scalar - LED power delivery goal [Watts]
%   Freq_Hz - scalar - Operating frequency goal [Hz]
%   Cj_F - function - Incremental diode capacitance versus reverse voltage
%   [Volts to Farads]
%   Coss_F - function - Incremental transistor output capacitance versus
%   drain source voltage [Volts to Farads]
%   Cind_F - scalar - Inductor capacitance to ground [Farads]
```

```
%   Rsw_Ohm - scalar - Transistor on resistance [Ohms]
%   TranCRes_Ohm - scalar - resistance in series with the FET capacitance [Ohms]
%   DiodeDrop_V - scalar - diode voltage drop [Volts]
%   DiodeCRes_Ohm - scalar - resistance in series with the diode capacitance [Ohms]
%   scaleT - scalar - amount the transistor die area is increased
%
%   Outputs
%   P_W - scalar - amount of power lost in both the diode and transistor
%   PDiode_W - scalar - power loss in the diode
%   PSwitch_W - scalar - power loss in the transistor
%   L_H - scalar - required inductor value [Henries]
%   Ton_s - scalar - on time [seconds]
%   Vswitch_V - vector - switch drain voltage [Volts]
%   Vdiode_V - vector - diode reverse voltage [Volts]
%   Iind_A - matrix - inductor current - first column is inductive
%   current, second column is capacitive current from Cind [Amperes]
%   Idiode_A - matrix - inductor current - first column is forward
%   current, second column is capacitive current from Cj [Amperes]
%   Iswitch_A - matrix - switch current - first column is on-state
%   current, second column is capacitive current from Coss [Amperes]
```

```
%   T_s - vector - time [seconds]

[ L_H, Ton_s, ~ ] = FindInductor(Vin_V, Vled_V, Power_W, Freq_Hz, @(Vds) Coss_F(Vds)*scaleT, Cj_F, Cind_F);
[ Vswitch_V, Iswitch_A, Vdiode_V, Idiode_A, Vind_V, Iind_A, T_s ] = SimulateWaveforms( Vin_V, Vled_V, Ton_s, L_H, @(Vds) Coss_F(Vds)*scaleT, Cj_F, Cind_F, Rsw_Ohm/scaleT);
[PDiode_W, PSwitch_W] = EstLosses( Iswitch_A, Idiode_A, T_s, DiodeDrop_V, DiodeCRes_Ohm, Rsw_Ohm/scaleT, TranCRes_Ohm);
P_W = PDiode_W + PSwitch_W;


end

function [ L_H, Ton_s, T_s ] = FindInductor( Vin_V, Vled_V, Power_W, Freq_Hz, Coss_F, Cj_F, Cind_F)
%FindInductor - This function gives you the inductance, diode scaling
%factor, and switch on-time necessary for the provided operating conditions
%
%   Inputs
%   Vin_V - scalar - Input supply voltage [Volts]
%   Vled_V - scalar - LED string voltage [Volts]
%   Power_W - scalar - LED power delivery goal [Watts]
%   Freq_Hz - scalar - Operating frequency goal [Hz]
%   Cj_F - function - Incremental diode capacitance versus reverse voltage
%   [Volts to Farads]
```

```
%   Coss_F - function - Incremental transistor output capacitance versus
%   drain source voltage [Volts to Farads]
%   Cind_F - scalar - Inductor capacitance to ground [Farads]
%
%   Output
%   L_H - scalar - required inductor value [Henries]
%   T_s - scalar - cycle time [seconds]
%   Ton_s - scalar - on time [seconds]

Cnode_F = @(V) ((Coss_F(V)+Cj_F(Vin_V-V)+Cind_F));

tRingScaled_s_rootH = RingTime( Vin_V, Vled_V, Cnode_F);

QSum_C = reimann(Cnode_F,0,Vin_V,100);

L_H=800e-9;

Tgoal_s=1/Freq_Hz;
Igoal_A=Power_W/Vled_V;
Qtot_C=Igoal_A*Tgoal_s;

while 1==1
    Ton_s=sqrt(Qtot_C/(Vin_V^2/(2*L_H*Vled_V)-Vin_V/(2*L_H)));
    Ipk_A=Ton_s*(Vin_V-Vled_V)/L_H;
    Toff_s=Ipk_A*L_H/Vled_V;
    Trise_s=QSum_C/Ipk_A;
    Tring_s=sqrt(L_H)*tRingScaled_s_rootH;
```

```
        T_s = Ton_s+Toff_s+Trise_s+Tring_s;
        scaleT = T_s/Tgoal_s;
        if (0.995 < scaleT && scaleT < 1.05)
            break;
        else
            L_H=L_H/scaleT;
        end
    end


end


function [ Vswitch_V, Iswitch_A, Vdiode_V, Idiode_A, Vind_V, Iind_A, T_s ] = ...
SimulateWaveforms( Vin_V, Vled_V, Ton_s, L_H, Coss_F, Cj_F, Cind_F, Rsw_Ohm)
%SimulateWaveforms Simulate a complete single cycle of the circuit
%
%   Inputs
%   Vin_V - scalar - Input supply voltage [Volts]
%   Vled_V - scalar - LED string voltage [Volts]
%   Ton_s - scalar - on time [nanoseconds]
%   L_H - scalar - Inductor inductance [nanoHenries]
%   Coss_F - function - Incremental transistor output capacitance versus
%   drain source voltage [Volts to Farads]
%   Cj_F - function - Incremental diode capacitance versus reverse voltage
```

```
%   [Volts to Farads]
%   Cind_F - scalar - Inductor capacitance to ground [Farads]
%   Rsw_Ohm - scalar - Transistor on resistance [Ohms]
%
%   Outputs
%   V_switch_V - vector - switch drain voltage [Volts]
%   V_diode_V - vector - diode reverse voltage [Volts]
%   I_ind_A - matrix - inductor current - first column is inductive
%   current, second column is capacitive current from Cind [Amperes]
%   I_diode_A - matrix - inductor current - first column is forward
%   current, second column is capacitive current from Cj [Amperes]
%   I_switch_A - matrix - switch current - first column is on-state
%   current, second column is capacitive current from Coss [Amperes]
%   T_s - vector - time [seconds]
%   Vzvs - scalar - voltage at which switching is occuring [Volts]

Cnode_F = @(V) ((Coss_F(V)+Cj_F(Vin_V-V)+Cind_F));


[ Vring_V, Iring_A, Tring_s ] = SimulateRing( Vin_V, Vled_V,
    Cnode_F, L_H, 0);

```

```
[ Vdisc_V, IdiscR_A, Idisc_A, Tdisc_s ] =SimulateDisc( Vin_V, Vled_V, Cnode_F, L_H, Rsw_Ohm, Vring_V(end), Iring_A(end), Tring_s(end));

[ Von_V, Ion_A, Ton_s ] = SimulateOn( Vin_V, Vled_V, Ton_s, L_H, Idisc_A(end), Tdisc_s(end));

[ Vrise_V, Irise_A, Trise_s ] = SimulateRise( Vin_V, Vled_V, Cnode_F, L_H, Ion_A(end), Ton_s(end));

[ Voff_V, Ioff_A, Toff_s ] = SimulateOff( Vin_V, Vled_V, L_H, Irise_A(end), Trise_s(end));



Vswitch_V = [Vring_V; Vdisc_V(2:end); Von_V; Vrise_V(2:end); Voff_V];
Vdiode_V = Vin_V-Vswitch_V;
Vind_V = Vin_V-Vled_V-Vswitch_V;


ICind_A = -Cind_F./(Cnode_F(Vswitch_V)).*[Iring_A; Idisc_A(2:end); 0 ; 0 ; Irise_A(2:end); 0 ;0];
ICswitch_A = Coss_F(Vswitch_V)./(Cnode_F(Vswitch_V)).*[Iring_A; Idisc_A(2:end); 0 ; 0 ; Irise_A(2:end); 0 ; 0];
ICdiode_A = Cj_F(Vdiode_V)./(Cnode_F(Vswitch_V)).*[Iring_A; Idisc_A(2:end); 0 ; 0 ; Irise_A(2:end); 0 ; 0];

T_s = [Tring_s; Tdisc_s(2:end); Ton_s; Trise_s(2:end); Toff_s];
```

```
Iind_A = [Iring_A' Idisc_A(2:end)' Ion_A' Irise_A(2:end)' Ioff_A' ; ICind_A']';
Iswitch_A = [zeros(size(Iring_A')) IdiscR_A(2:end)' Ion_A' zeros(size([Irise_A(2:end)' Ioff_A'])) ; ICswitch_A']';
Idiode_A = [zeros(size([Iring_A' Idisc_A(2:end)' Ion_A' Irise_A(2:end)'])) Ioff_A' ; ICdiode_A']';

end

function [ PDiode_W, PSwitch_W ] = EstLosses( Iswitch_A, Idiode_A, T_s, DiodeDrop_V, DiodeCRes_Ohm, TranRon_Ohm, TranCRes_Ohm )
% ESTLOSSES Find the losses in the devices given the current waveforms
%
%   Inputs
%   I_switch _A - matrix - switch current - first column is on-state
%   current, second column is capacitive current from Coss [Amperes]
%   I_diode _A - matrix - inductor current - first column is forward
%   current, second column is capacitive current from Cj [Amperes]
%   T_s - vector - time [seconds]
%   DiodeDrop_V - scalar - diode voltage drop [Volts]
%   DiodeCRes_Ohm - scalar - resistance in series with the diode capacitance [Ohms]
%   TranRon_Ohm - scalar - FET on resistance [Ohms]
```

```
%   TranCRes_Ohm - scalar - resistance in series with the FET capacitance [Ohms]
%
%   Outputs
%   PDiode_W - scalar - power loss in the diode
%   PSwitch_W - scalar - power loss in the transistor



DT_s=T_s(2:end) - T_s(1:end-1);

PswitchCap_W = ((Iswitch_A (2:end,2)'.^2)*DT_s*TranCRes_Ohm)/T_s(end);
PswitchCond_W = (((Iswitch_A(1:end-1,1))'.^2+(Iswitch_A(2:end,1))'.*(Iswitch_A(1:end-1,1))'+(Iswitch_A(2:end,1))'.^2)/3*DT_s*TranRon_Ohm)/T_s(end);
PDiodeDrop_W = ((Idiode_A(2:end,1))'+(Idiode_A(1:end-1,1))')*DT_s*DiodeDrop_V/2/T_s(end);
PDiodeCap_W = ((Iswitch_A (2:end,2)'.^2)*DT_s*DiodeCRes_Ohm)/T_s(end);

PSwitch_W = PswitchCap_W + PswitchCond_W;
PDiode_W = PDiodeDrop_W + PDiodeCap_W;


end


function [ Coss ] = MITHEMT_Coss( Vds )
```

```
%This function shows the voltage/capacitance relationship
    for an MIT FET
Coss = (-8e-5*Vds.^2+0.0074*Vds+7.118)*1e-11;
end


%%%%%%%%%%%%%%%%%%
% This script filters the designs generated in the first
    step. All generated designs have been saved in the file "
    AllDesigns.txt". Here, all of those designs are checked
    for the filtering process. First the hard requirements
    are checked. Next the Pareto-Filtering process is used to
     keep only the superior designs. See the filtering
    process in "Chapter 4: Inductor Design" of the thesis.
%%%%%%%%%%%%%%%%%%

clear all;
close all;

load('AllDesigns.txt')
Sorted=AllDesigns;

BudgetVectStp1=2.5-Sorted(:,28); % Budget loss for the
    magnetics considering a total loss budget of 2.5 Watts.
MinFeatureSizeStp1=Sorted(:,10); % Minimum feature size of
    the inductor
InductorLossStp1=Sorted(:,25); % AC and DC loss for the
    inductor
LossVectStp1=Sorted(:,29); %Total circuit loss
```

```
ErrorInLStp1=abs(Sorted(:,18)); %Error in calculating inductor

HRStep=0; % A counter for the designs that pass the hard requirements

% The following loop checks that the designs satisfy the hard requirements and have numerically converged in calculating the inductor value and losses.

for i=1:size(LossVectStp1)
    if (   (InductorLossStp1(i)<(1.5*BudgetVectStp1(i)) )  &&...
           (ErrorInLStp1(i)<5)  )  &&...
           (BudgetVectStp1(i)<2.5)
         % (MinFeatureSizeStp1(i)>75e-6)  )
         HRStep=HRStep+1;
         PassedHardReq(HRStep,:)=Sorted(i,:); % Only the designs that pass the hard requirements are saved in "PassedHardReq"
    end
end

BudgetVect=2.5-PassedHardReq(:,28);
MinFeatureSize=PassedHardReq(:,10);
InductorLoss=PassedHardReq(:,29);
LossVect=PassedHardReq(:,18);
FreqMHz=PassedHardReq(:,1); % Operating Frequency [MHz]
RoVect=PassedHardReq(:,4); % Outer Radius [mm]
ConvEfficiency=PassedHardReq(:,30);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following loop is for the Pareto-Filtering process.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 for i=1:HRStep
     jStep=0;

     for j=[1:(i-1),(i+1):HRStep]

         if (  (LossVect(i)>LossVect(j)) &&...
                 (RoVect(i)>RoVect(j)) )   &&...
                 (MinFeatureSize(i)<MinFeatureSize(j) )
             jStep=jStep+1;
         end
     end

     if(  (jStep == 0)    )

         kStep=kStep+1;
         Filtered(kStep,:)=PassedHardReq(i,:);
        %   OptMu(kStep)=muVect(i); ADD THIS LINE FOR RANDOM
             MU


     end

end
dlmwrite('FilteredDesigns.txt', Filtered, 'delimiter', '\t',
     ...
```

```
    'precision', 5); % Saving the filtered resuls in a file.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here, the filtered designs are devided into 3 categories
    based on the minimum feature size: 1) Between 90 um to
    110 um, 2) between 110 um to 130 um, 3) between 130 um to
     170 um
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FirstMFS=110e-6;
SecondMFS=130e-6;

AboveSecondStep=0;
BelowFirstStep=0;
BtwFirstandSecondStep=0;

for i=1:size(Filtered(:,10),1)
    if Filtered(i,10)<FirstMFS
        BelowFirstStep=BelowFirstStep+1;
        BelowFirst(BelowFirstStep,:)=Filtered(i,:);
    else
        if Filtered(i,10)<SecondMFS
            BtwFirstandSecondStep=BtwFirstandSecondStep+1;
            BtwFirstandSecond(BtwFirstandSecondStep,:)=
               Filtered(i,:);
        else

            AboveSecondStep=AboveSecondStep+1;
            AboveSecond(AboveSecondStep,:)=Filtered(i,:);
        end
```

```
        end
end

%
figure;
plot(BelowFirst(:,4),BelowFirst(:,29),'k^',BtwFirstandSecond
    (:,4),BtwFirstandSecond(:,29),'rs', AboveSecond(:,4),
    AboveSecond(:,29),'.b');
title('Min Feature Size: Black 90-110 um; Red: 110-130 um;
    Blue: 130-170 um');
xlabel('Ro[mm]');
ylabel('Circuit Loss [W]');
grid on;
grid(gca,'minor');

%
figure;
plot(BelowFirst(:,4),BelowFirst(:,1),'k^',BtwFirstandSecond
    (:,4),BtwFirstandSecond(:,1),'rs',AboveSecond(:,4),
    AboveSecond(:,1),'.b');
title('MinFeatSize black 90-110 um; Red: 110-130 um; Blue:
    130-170 um');
xlabel('Ro[mm]');
ylabel('Frequency [MHz]');
grid on;
grid(gca,'minor');

BelowFirst(:,4);
%
```

```
figure;
plot(BelowFirst(:,4),BelowFirst(:,11),'k^',BtwFirstandSecond
    (:,4),BtwFirstandSecond(:,11),'rs',AboveSecond(:,4),
    AboveSecond(:,11),'.b');
title('MinFeatSize black 90-110; red: 110-130; blue: 130-170
    um');
xlabel('Ro[mm]');
ylabel('ScaleT');
grid on;
grid(gca,'minor');

figure;
plot(BelowFirst(:,30),BelowFirst(:,31),'k^',
    BtwFirstandSecond(:,30),BtwFirstandSecond(:,31),'rs',
    AboveSecond(:,30),AboveSecond(:,31),'.b');
title('MinFeatSize black 90-110; red: 110-130; blue: 130-170
    um');
xlabel('Converter Efficiency');
ylabel('Inductor Power Density [W/m^2]');
grid on;
grid(gca,'minor');
```

Appendix B

Four Components of Inductance

```
% LAP : Equivalent inductance for poloidal magnetic fields inside air
% LAT : Equivalent inductance for torooidal magnetic fields inside air
% LWP : Equivalent inductance for poloidal magnetic fields inside the winding
% LWT : Equivalent inductance for toroidal magnetic fields inside the winding
% Ro : Outer radius
% Ri : Inner radius
% T : Conductor thickness
% D : Inductor Height
% Freq: Frequency of calculating the inductance
% sigma: Conductivity of Copper

omega=2*pi*Freq;
Ralpha=(1/2)*sqrt(2)*sqrt(omega*mu0*sigma)
LAP= (1/2)*(Ro+Ri)*mu0*(log((8*(Ro+Ri))/(Ro-Ri))-2);
LAT=(1/2)*mu0*N^2*(-log(Ri+T)+log(Ro-T))*(D)/pi;
```

```
LWP=0;
LWT=2*((1/128)*mu0*N^2*(2*pi*Ri-N*Delta)*(exp(4*Ralpha*T)*
    cos(Ralpha*T)^2+exp(4*Ralpha*T)-8*cos(Ralpha*T)*sin(
    Ralpha*T)*exp(2*Ralpha*T)-cos(Ralpha*T)^2-1+exp(4*Ralpha*
    T)*sin(Ralpha*T)^2-sin(Ralpha*T)^2)*exp(-2*Ralpha*T)*D/(
    pi^2*(Ri+T)^2*(sinh(Ralpha*T)^2*cos(Ralpha*T)^2+cosh(
    Ralpha*T)^2*sin(Ralpha*T)^2)*Ralpha)+(1/128)*mu0*N^2*(2*
    pi*Ro-N*Delta)*(exp(4*Ralpha*T)*cos(Ralpha*T)^2+exp(4*
    Ralpha*T)-8*cos(Ralpha*T)*sin(Ralpha*T)*exp(2*Ralpha*T)-
    cos(Ralpha*T)^2-1+exp(4*Ralpha*T)*sin(Ralpha*T)^2-sin(
    Ralpha*T)^2)*exp(-2*Ralpha*T)*D/(pi^2*(Ro-T)^2*(sinh(
    Ralpha*T)^2*cos(Ralpha*T)^2+cosh(Ralpha*T)^2*sin(Ralpha*T
    )^2)*Ralpha) -(1/16)*mu0*N^2*(-exp(4*Ralpha*T)+4*cos(
    Ralpha*T)*sin(Ralpha*T)*exp(2*Ralpha*T)+1)*exp(-2*Ralpha*
    T)*(-log(Ri+T)+log(Ro-T))/(pi*(sinh(Ralpha*T)^2*cos(
    Ralpha*T)^2+cosh(Ralpha*T)^2*sin(Ralpha*T)^2)*Ralpha));
```