Extension of working range of a current sensorless unity power factor utility interface

by

Ilya Panfilov
Diploma of Engineer, Ivanovo State Power University, 2010

A Report Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering,

Supervisory Committee

Extension of working range of a current sensorless unity power factor utility interface

by

Ilya Panfilov
Diploma of Engineer, Ivanovo State Power University, 2010

Supervisory Committee

Dr. Subhasis Nandi, (Department of Electrical and Computer Engineering)
Supervisor

Dr. Jens Bornemann, (Department of Electrical and Computer Engineering)
Supervisor

Dr. Poman So, (Department of Electrical and Computer Engineering)
Departmental Member

Abstract

Supervisory Committee
Dr. Subhasis Nandi, (Department of Electrical and Computer Engineering)
Supervisor

Dr. Jens Bornemann, (Department of Electrical and Computer Engineering)
Supervisor

Dr. Poman So, (Department of Electrical and Computer Engineering)
Departmental Member

The presented work shows further improvement made to the boost-type switch mode rectifier developed in [1]. Modern industry offers various analog controllers for power factor correction purposes. Their manufacturers claim to provide the capability to control a broad range of input and output parameters. The rectifier in [1] has several major advantages over conventional power factor correction (PFC) converters. It is controlled by a fully programmable digital signal processor, it does not utilize a current sensing resistor for current feedback, it is capable of calculating real values of rectifying inductor parameters employed for current values calculation, etc. However, the developed converter has one significant drawback - limited input voltage range. The focus of the present project is an extension of alternating voltage range that can be supplied to the described circuit.

Table of Contents

List of Tables

List of Figures

List of symbols, Sub- and Superscripts and Abbreviations

Symbols

C	capacitance
D	duty cycle
d	instantaneous duty cycle
f	frequency
G	transfer function
h	harmonic number
I	current
i	instantaneous current
K	controller gain, gain multiplier
k	k-factor
L	inductance
N,n	number of cycles, register value
P	real power, resistance of pot
PF	power factor
R	resistance
S	apparent power
s	Laplace variable
T	period
t	time
v	instantaneous voltage
V	voltage (RMS unless othervise noted)
φ	phase angle
ω	angular frequency
κ	voltage controller output

Sub- and superscript

*	reference value
av	average
b	boost
c	controller, crossover
est	estimated value
i	current loop
in	input value
L	inductor
m	margin (phase margin)
max	maximum value
min	minimum value
o	output value
on	on-state
p	pole
peak	peak value
pl	plant
Q	switch
rip	ripple
s	switching
v	voltage loop
z	zero

Abbreviations

AC	alternating current
ADC	analog to digital converter
CCS	code composer studio
CENELEC	European Committee for Electrotechnical Standardization
DC	direct current
DSP	digital signal processor
IEC	International Electrotechnical Commission
MOSFET	metal oxide semiconductor field effect transistor
PFC	power factor correction
RMS	root mean square
SC	switching cycle
SM	switched mode
THD	total harmonic distortion
VAC	volts alternating current
ZCD	zero crossing detection

Acknowledgments

My supervisor Dr. Nandi provided numerous priceless ideas. Without his contribution and expertise the work would not have succeeded.

Dr. Bhat was generously sharing his experience when I had difficulties with the converter prototype.

1 Introduction

Many industrial and domestic applications require direct voltage input while the supply is sinusoidal. The easiest rectifier to use would be a diode bridge. The single-phase diode bridge rectifier outputs a rectified sinusoid which is not acceptable in a vast majority of cases. Bulky and expensive capacitors and inductors on the secondary side might improve output to an acceptable level, but it would result in a highly distorted input current waveform heavily contaminated with harmonics. Industrial standardization organizations such as the International Electrotechnical Commission (IEC) and the European Committee for Electrotechnical Standardization (CENELEC) issued regulations for allowed harmonic content. Hence, further actions should be taken to match the requirements. A good existing solution would be an active current shaping by means of switched mode converters. Switched mode converters not only allow drawing undistorted sinusoidal current but also provide regulated constant output at the desired level. Boost alternative current (AC) to direct current (DC) switched mode converter topology was employed for this project,

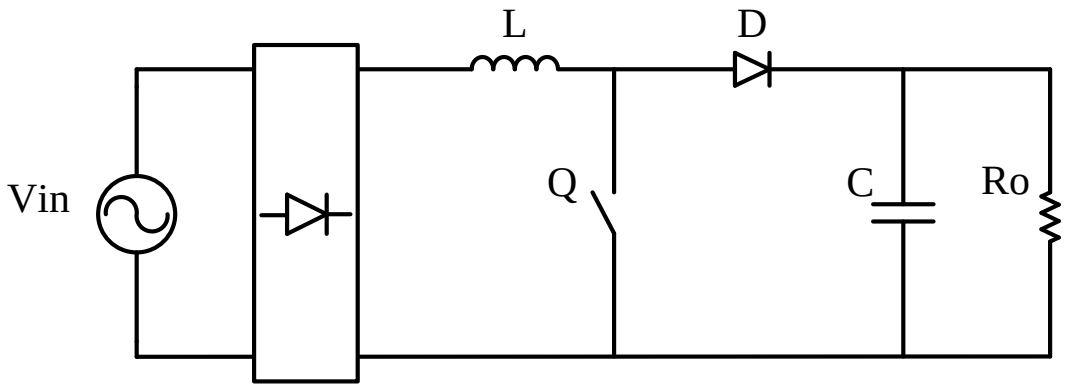


Figure 1.1: Boost converter topology

The energy accumulated in the inductor L while the switch Q is on is then transferred to the output filtering capacitor C and the load Ro when the switch is turned off (Fig. 1.1). The output voltage is calculated as follows:

$$V_o = V_{in} \cdot \frac{1}{1-D} \tag{1.1}$$

where D is the duty cycle or amount of time when the switch is closed in each switching cycle.

$$D = \frac{t_{on}}{T} \tag{1.2}$$

As one can conclude from these relations, the boost converter cannot output a voltage lower than that at the input.

The switched mode (SM) power factor controller contains two control loops. One regulates input current waveform, the second regulates the level of DC voltage output. The undistorted shape of a sinusoidal signal indicates a low harmonics content and therefore a higher power factor (PF). The power factor is a ratio between real power P and apparent power S. Since input voltage is assumed to be ideally sinusoidal, only the fundamental harmonic can contribute to real power:

$$PF = \frac{P}{S} = \frac{I_{in(1)}}{\sqrt{\sum_{n=1}^{\infty} I_{in(n)}^2}} \cos(\varphi_1) \tag{1.3}$$

The boost AC/DC power factor correction converter developed in [1] has several remarkable enhancements compared to conventional implementations, as described briefly in the following. The converter has sensorless current feedback. Most regular topologies have a current sensing resistor of small value. Voltage across this resistor is measured and converted into current value. This indirect method is straight forward and

reliable but associated with power losses in the sensor. Also, the resistor requires heat dissipation and its resistance can vary with temperature. Sensorless current feedback, developed in [1], helps to eliminate these problems. The instantaneous value of the current in the inductor is derived from the voltage across the inductor using the known relation

$$v_L = R_L \cdot i_L + L\frac{di_L}{dt} \tag{1.4}$$

This approach requires accurate values of inductor resistance and inductance. This accuracy would suffer when parameters are not measured individually in the process of mass production, or because of element aging. Therefore, an attempt to develop a parameters estimation procedure during steady state operation was made. The estimation is based on the integration of voltage across the inductor within one line half cycle. Assuming the inductor current is purely sinusoidal,

$$i_L = I_{L\ peak} \cdot \left|\sin(\omega t)\right| \tag{1.5}$$

and integrating expression (1.4) over one line half cycle, after replacing (1.5) in (1.4), the following relations were derived in [1]:

$$R_L^{est} = \frac{\omega}{2I_{L\ peak}} \int_0^{T_2} v_L\ dt \tag{1.6}$$

$$L^{est} = \frac{1}{I_{L\ peak}} \left(\int_0^{T/2} v_L\ dt - \frac{1}{2} \int_0^{T/4} v_L\ dt \right) \tag{1.7}$$

Despite all improvements, the circuit has one major disadvantage. The converter was designed for a narrow input voltage range of 120VAC ±10% deviation.

It is a well known fact that supply voltage standards differ from country to country. Therefore, international manufacturers strive to accommodate this variation of parameters

and produce universal products rather than numerous region-oriented types. The same applies to PFC devices. Therefore, the main purpose of the presented project is broadening the input AC voltage range that the converter can correctly work with. Power factor controller ML4821 by Fairchild Semiconductor was taken as an example. Datasheet [2] states a 90 - 260 VAC input range. A band of 80 - 260 VAC was adopted as a project goal. Aside from that, a number of associated improvements were introduced.

1.1 Operation conditions

Hence, the converter should draw sinusoidal current having measured and computed inductor current feedback as well as computed feedback with parameters adaptation enabled at the following operation conditions:

- 80 - 260 VAC input voltage range,
- 380 VDC output voltage,
- 100 - 200W resistive load range.

1.2 Contributions

The main goal of the project to extend input voltage range of existing boost-type converter from 120 ±10% VAC to 80-260 VAC was reached. Also, the stability of operation and precision of feedback signals acquisition were improved along the way.

2 Recalculation of controller parameters

The control of the boost converter is based on two closed loops: output voltage and inductor current controllers. Expressions for controller parameters were derived from a linearized converter model. Details on function derivation and model linearization based on the state-space averaging approach are given in section (3) of [1].

Controller parameters were recalculated according to new requirements. Input voltage V_{in} is now assumed to be 260 volts RMS. The boost inductor was modified in order to allow for existing current ripple requirements, which is described in section 4.2 of the current report. Hence, inductor resistance R_L and inductance L are now 1.96 Ω and 17.8 mH, respectively. Controller parameters were calculated for the working conditions that result in the least phase margin of the plant transfer functions for both control loops. For this case, these would be 80 V RMS input voltage at a quarter of rated load (50 W or load resistance of 2888 Ω).

2.1 Current controller parameters

The current control loop consists of a plant and controller (Fig.2.1).

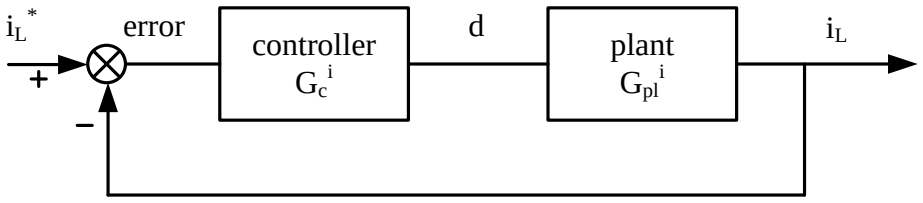


Figure 2.1: Current controller block diagram

From [1] the plant transfer function is as follows:

$$G_{pl}^{i}(s)=\frac{\frac{2\sqrt{2}}{\pi}V_{in}}{\frac{R_L}{R_o}+(1-D)_{av}^{\ 2}}\cdot\frac{(1-D)_{av}}{L}\cdot\frac{s+\frac{2}{R_oC}}{\left(s+\frac{R_L}{L}\right)\left(s+\frac{1}{R_oC}\right)} \qquad (2.1)$$

$$\kappa_{av}=\frac{1}{2R_L}-\frac{\sqrt{V_{in}^2-4P_oR_L}}{2V_{in}R_L}=\frac{1}{2\cdot 1.96\ \Omega}-\frac{\sqrt{(80\ V)^2-4\cdot 50\ W\cdot 1.96\ \Omega}}{2\cdot 80\ V\cdot 1.96\ \Omega}=0.00794\ \frac{A}{V} \quad (2.2)$$

From here

$$(1-D)_{av}=\frac{2\sqrt{2}}{\pi}\cdot\frac{V_{in}}{V_o}\cdot\left(1-R_L\kappa_{av}\right)=\frac{2\sqrt{2}}{\pi}\cdot\frac{80\ V}{380\ V}\cdot\left(1-1.96\ \Omega\cdot 0.00794\right)=0.187 \quad (2.3)$$

This can be plugged into eq.(2.1)

$$G_{pl}^{i}(s)=\frac{\frac{2\sqrt{2}}{\pi}80}{\frac{1.96}{2888}+0.187^2}\cdot\frac{0.187}{17.8\cdot 10^{-3}}\cdot\frac{s+\frac{2}{2888\cdot 270\cdot 10^{-6}}}{\left(s+\frac{1.96}{17.8\cdot 10^{-3}}\right)\left(s+\frac{1}{2888\cdot 270\cdot 10^{-6}}\right)}= \qquad (2.4)$$

$$=21271\cdot\frac{s+2.56}{(s+110)\cdot(s+1.282)}$$

Previously chosen in [1], the open loop transfer function bandwidth of current control loop did not provide stable control of the implemented circuit. Therefore, the bandwidth was decreased to 1/10 f_{sw} =2 kHz [7]. The controller design was performed using the *k*-factor approach [3], a technique that allows obtaining controller parameters using only a few algebraic equations. For employed type 2 error amplifier the *k*-factor is defined as the square root of the ratio of the pole frequency to the zero frequency. The zero and the pole are located such that their geometric mean is the gain crossover frequency, $\sqrt{\omega_z\omega_p}=\omega_c$.

Selecting $\omega_c=2\cdot\pi\cdot 2000\ Hz=12570\ rad/s$ and a phase margin of 60^o [1]:

The phase of the plant transfer function for current controller is

$$\varphi_{pl}(\omega_c)=\arg\left\{G_{pl}^{i}(j\omega_c)\right\}=-90^o \qquad (2.5)$$

The phase boost, required from current controller, is

$$\varphi_b = \varphi_m - \left(90^o + \varphi_{pl}\right) = 60^o - (90^o - 90^o) = 60^o \tag{2.6}$$

The *k*-factor is

$$k = \tan\left(45^o + \frac{\varphi_b}{2}\right) = \tan\left(45^o + \frac{60^o}{2}\right) = 3.732 \tag{2.7}$$

Respective zero, pole and gain of the type 2 error amplifier, adopted for current control

$$\omega_z = \frac{\omega_c}{k} = \frac{12570}{3.732} = 3367\ \ s^{-1} \tag{2.8}$$

$$\omega_p = k \cdot \omega_c = 3.732 \cdot 12570 = 46900\ \ s^{-1} \tag{2.9}$$

$$K = \frac{1}{\left| G_{pl}^{i}(j\omega_c) \cdot \frac{j\omega_c + \omega_z}{j\omega_c(j\omega_c + \omega_p)} \right|} = \\ = \frac{1}{\left| 21271 \cdot \frac{j12570 + 2.565}{(j12570 + 110) \cdot (j12570 + 1.282)} \cdot \frac{j12570 + 3367}{j12570(j12570 + 46900)} \right|} = 27710 \tag{2.10}$$

2.2 Voltage controller parameters

The voltage control loop is shown in Fig.2.2.

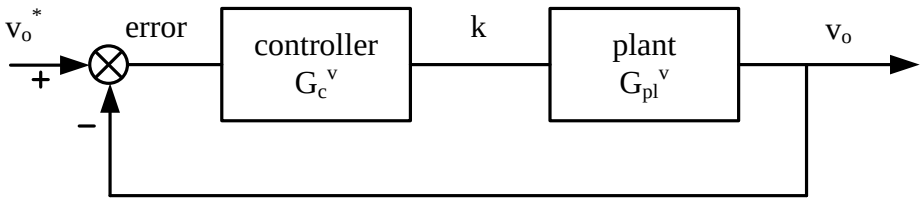


Figure 2.2: voltage controller block diagram

From [1], the plant transfer function is as follows:

$$G_{pl}^{v}(s)=\frac{2\sqrt{2}}{\pi}V_{in}\cdot\frac{L}{R_oC(1-D)_{av}}\cdot\frac{-s+\frac{1}{L}\cdot\left[R_o(1-D)_{av}{}^2-R_L\right]}{s+\frac{2}{R_oC}}=$$

$$=\frac{2\sqrt{2}}{\pi}80\cdot\frac{17.8\cdot10^{-3}}{2888\cdot270\cdot10^{-6}\cdot0.187}\cdot\frac{-s+\frac{2888\cdot0.187^2-1.96}{17.8\cdot10^{-3}}}{s+\frac{2}{2888\cdot270\cdot10^{-6}}}=8.812\cdot\frac{-s+1302}{s+2.565} \tag{2.11}$$

Applying the *k*-factor approach and selecting a crossover frequency of 10 Hz, we get

$$\varphi_{pl}(\omega_c)=\arg\left\{G_{pl}^{v}(j\omega_c)\right\}=-88.31^o \tag{2.12}$$

$$\varphi_b=\varphi_m-\left(90^o+\varphi_{pl}\right)=60^o-(90^o-88.31^o)=58.31^o \tag{2.13}$$

$$k=\tan\left(45^o+\frac{\varphi_b}{2}\right)=\tan\left(45^o+\frac{58.31^o}{2}\right)=3.524 \tag{2.14}$$

$$\omega_z=\frac{\omega_c}{k}=\frac{2\pi\cdot10\ Hz}{3.524}=17.83\ s^{-1} \tag{2.15}$$

$$\omega_p=k\cdot\omega_c=3.524\cdot2\pi\cdot10\ Hz=221.4\ s^{-1} \tag{2.16}$$

$$K=\frac{1}{\left|G_{pl}^{v}(j\omega_c)\cdot\frac{j\omega_c+\omega_z}{j\omega_c(j\omega_c+\omega_p)}\right|}=0.2852 \tag{2.17}$$

Hence, the current and voltage controller transfer functions are as follows:

$$G_c^i=27710\frac{s+3367}{s(s+46900)} \tag{2.18}$$

$$G_c^v=0.2852\frac{s+17.83}{s(s+221.4)} \tag{2.19}$$

3 Simulation of the converter

Simulations of converter improvements were performed in MATLAB Simulink by MathWorks. The model of the converter was adopted from [1] and altered according to recalculated parameters. Computer model (Fig. 3.1) examination concentrated on the ability to operate with the computed inductor current in steady state and transient modes 80 V RMS, 120 V RMS and 260 V RMS input. The following are captured transients.

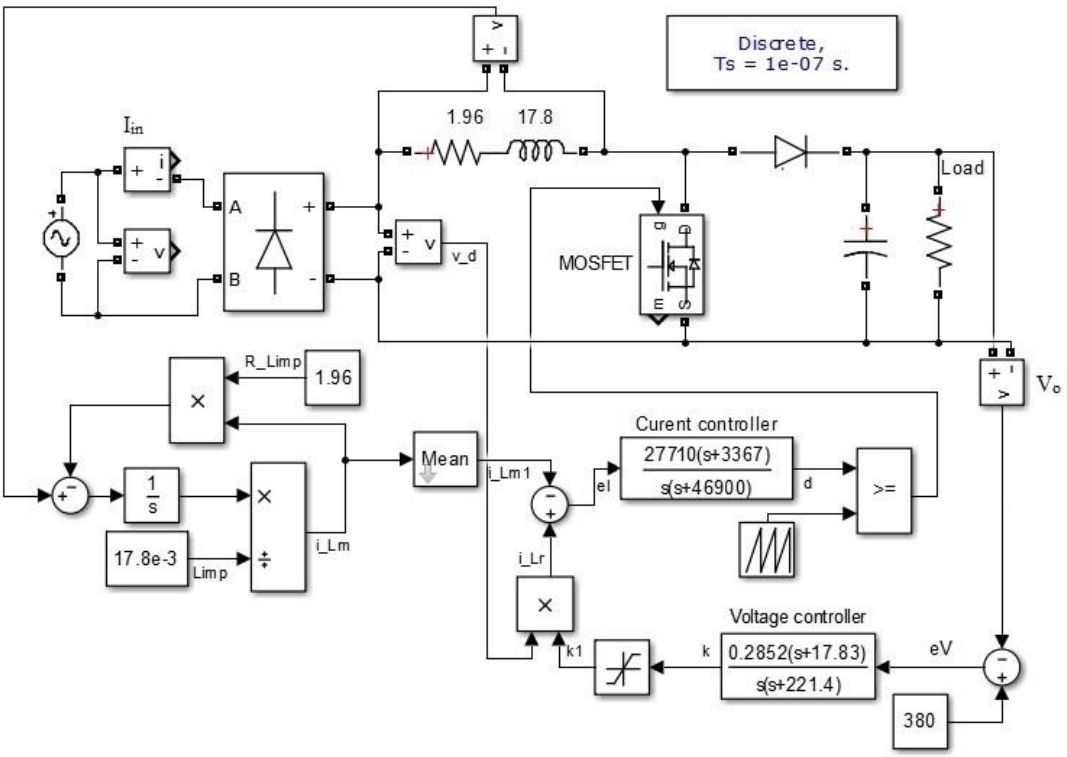


Figure 3.1: Simulink model of the converter

3.1 Load change transient

Step change of a load from 100% (200 W) to 50% (100W) and backward at 80 V as shown in Fig. 3.2, 120 V (Fig. 3.3) and 260 V (Fig. 3.4) were simulated in this test.

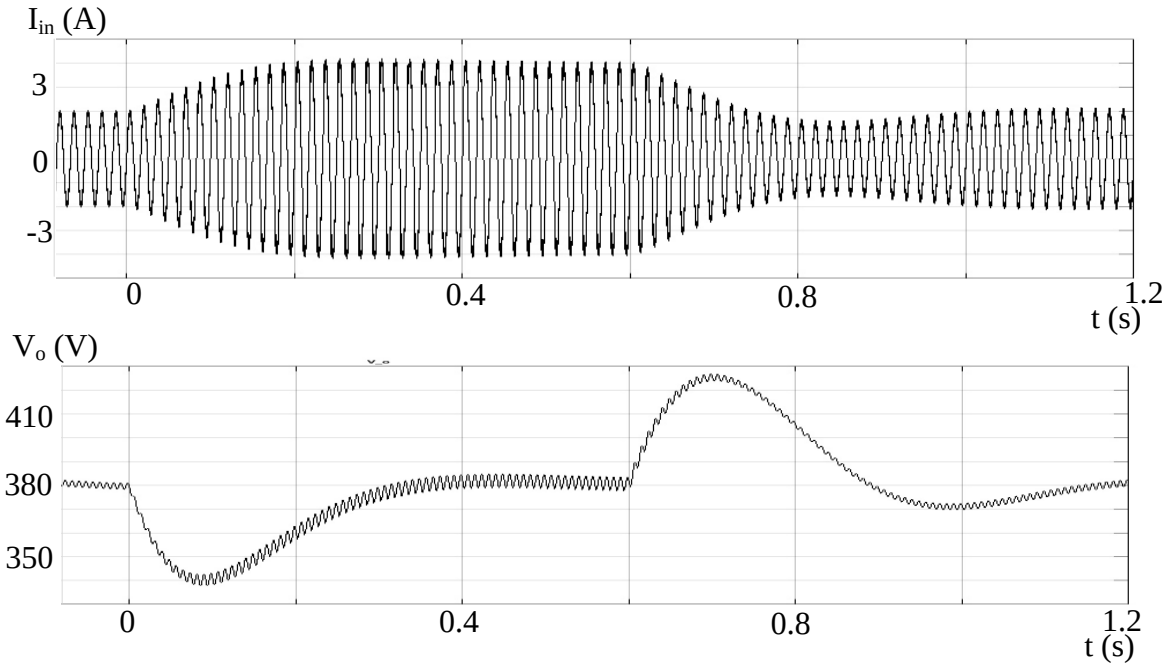


Figure 3.2: Input 80 V RMS, 100 W - 200 W – 100 W load change

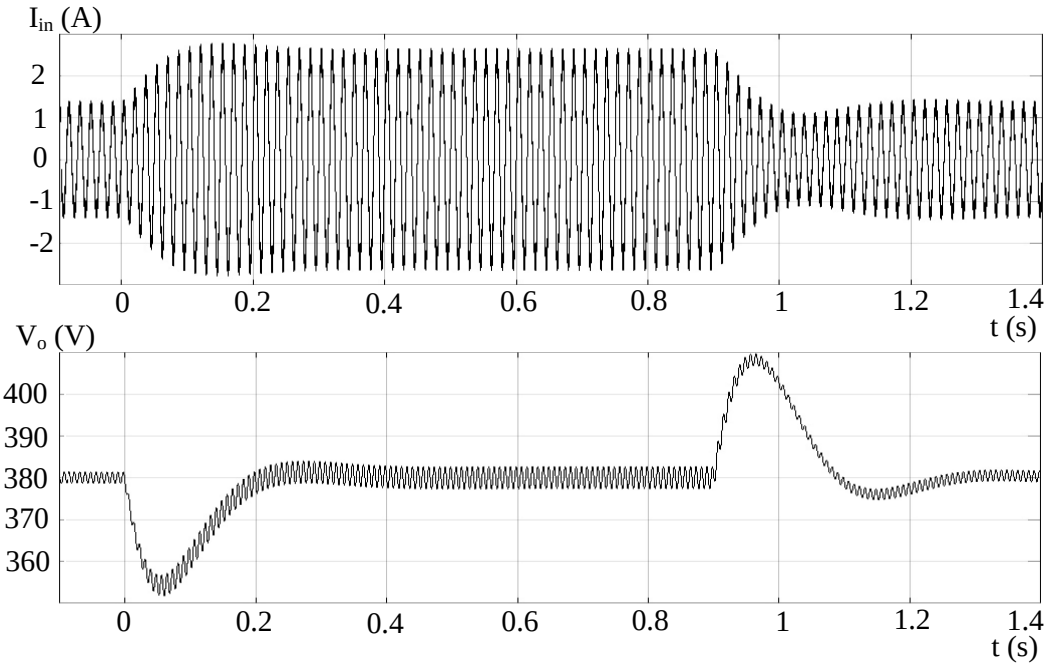


Figure 3.3: Input 120 V RMS, 100 W - 200 W – 100 W load change

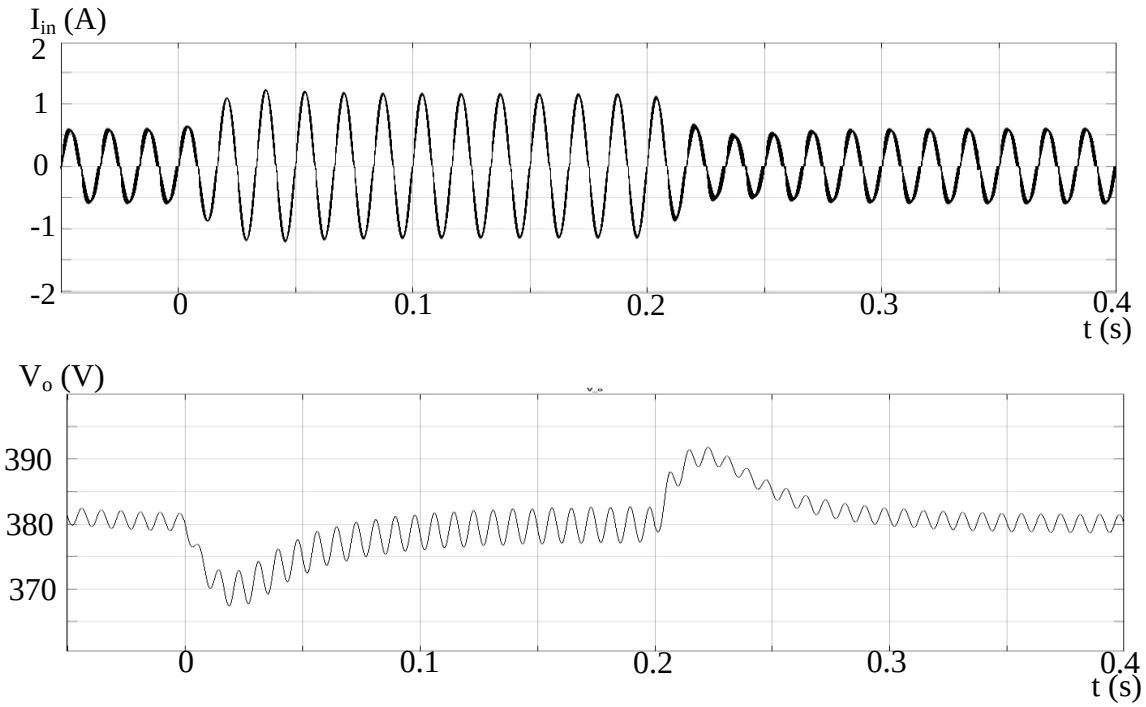


Figure 3.4: Input 260 V RMS, 100 W - 200 W - 100W load change

3.2 Input voltage change transient

In this test, input voltage step changes were applied to the model. Because the absolute maximum and minimum values of input voltage are taken as operating points in the presented work, a step change of 20% or 16 V in addition to an 80 V operating point (Fig.3.5) , ± 20% of 120 V operating point (Fig.3.6, 3.7) and 50 V drop for a 260 V operating point (Fig. 3.8) were simulated.

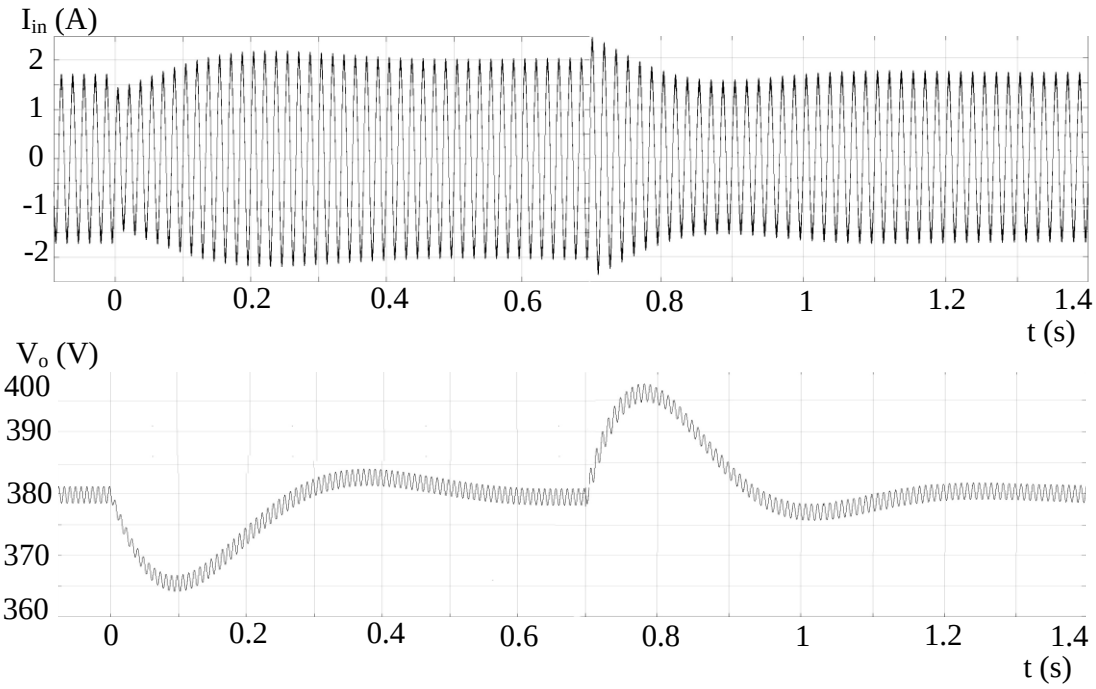


Figure 3.5: 200 W load 80 V - 96 V - 80 V input voltage change

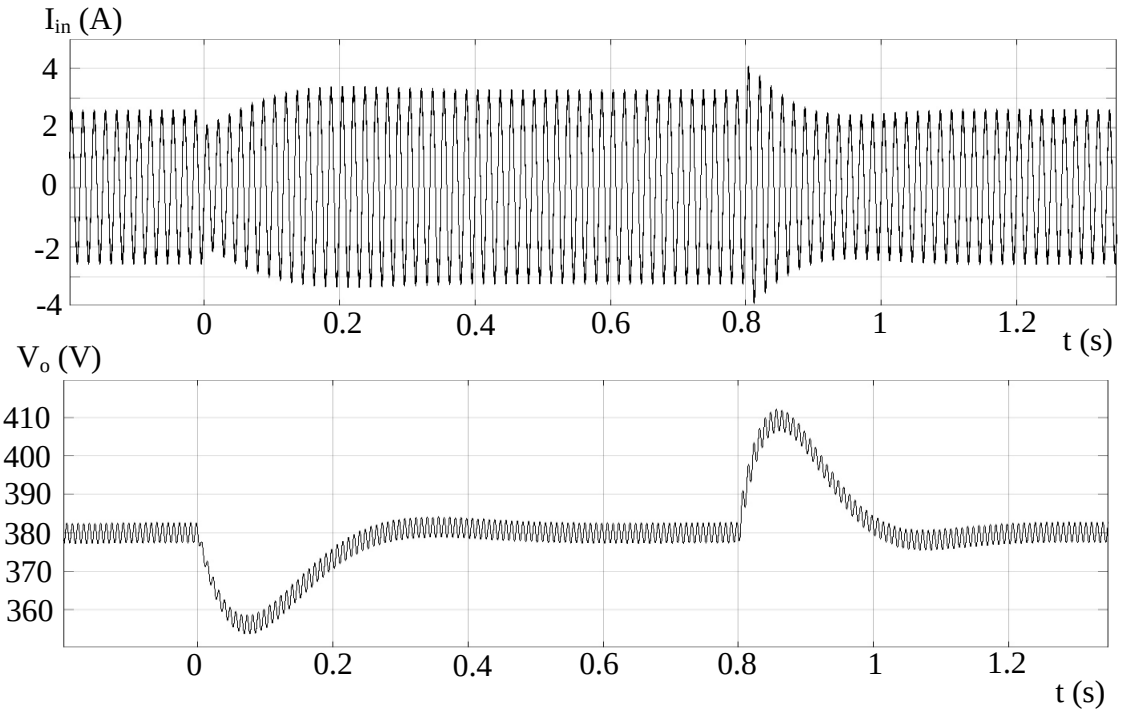


Figure 3.6: 200 W load 120 V - 96 V - 120 V input voltage change

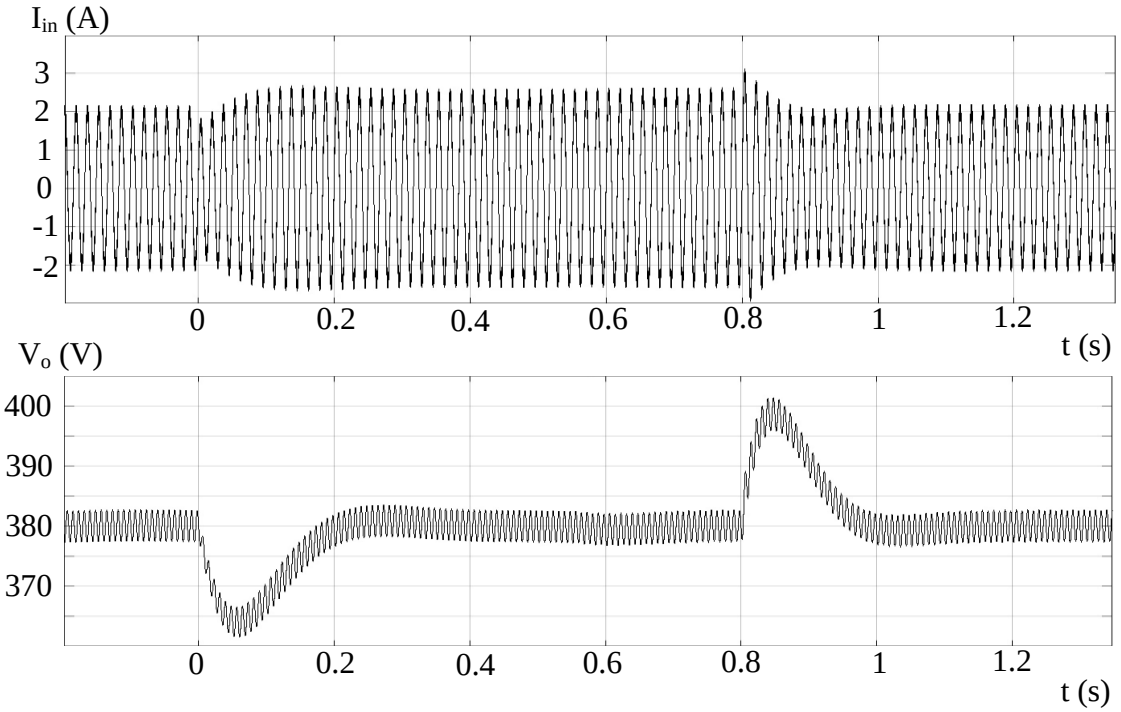


Figure 3.7: 200 W load 144 V - 120 V - 144 V input voltage change

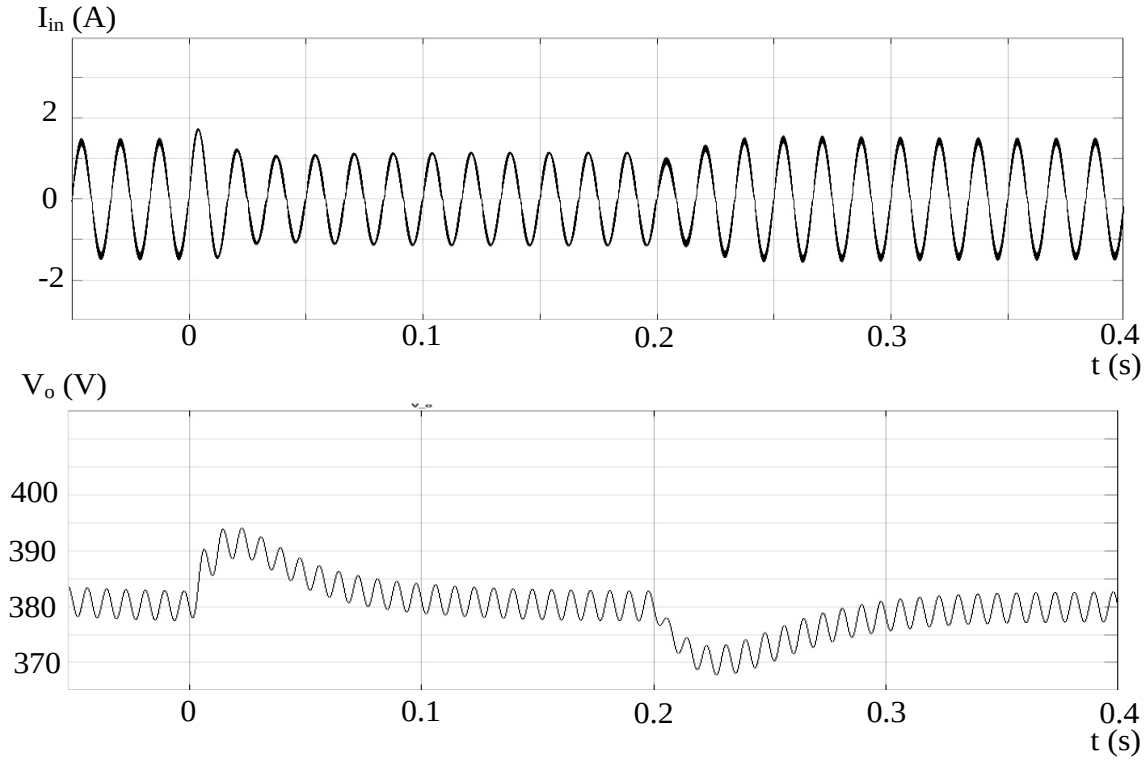


Figure 3.8: 200 W load 210 V - 260 V - 210 V input voltage change

4 Hardware modifications

The presented project is based on the previously built converter circuit [1] and can be found in Fig. 4.1 along with a detailed component list in Table 4.1. In [1] the following components were integrated into the basic circuit presented in Fig 1.1:

- Small value resistor R_{LL} in series with the inductor is used to prove the workability of the parameter adaptation concept. Jumper J_2 removes or adds this resistance to the circuit (was not used in current project).
- Current sensing resistor R_c that can be shorted by jumper J_1 when the converter works with calculated feedback and does not require direct feedback from R_c.
- Diode D_b provides a path for the output capacitors charging current to bypass the inductor and the boost diode. It helps to avoid inductor core saturation and shorten turn on transients. When output becomes higher than the peak of input, D_b turns reverse biased and has no effect on the circuit.
- Light emitting diode at the converter output indicates presence of the output voltage.
- C_f helps to reduce propagation of switching noise to the utility grid.

The converter design in [1] is based on 120 V RMS input. The first step of transition to higher voltages was the analysis of electrical component compatibility. Since the load wattage remains the same and when converter losses are neglected, current and voltage on the input side are inversely proportional.

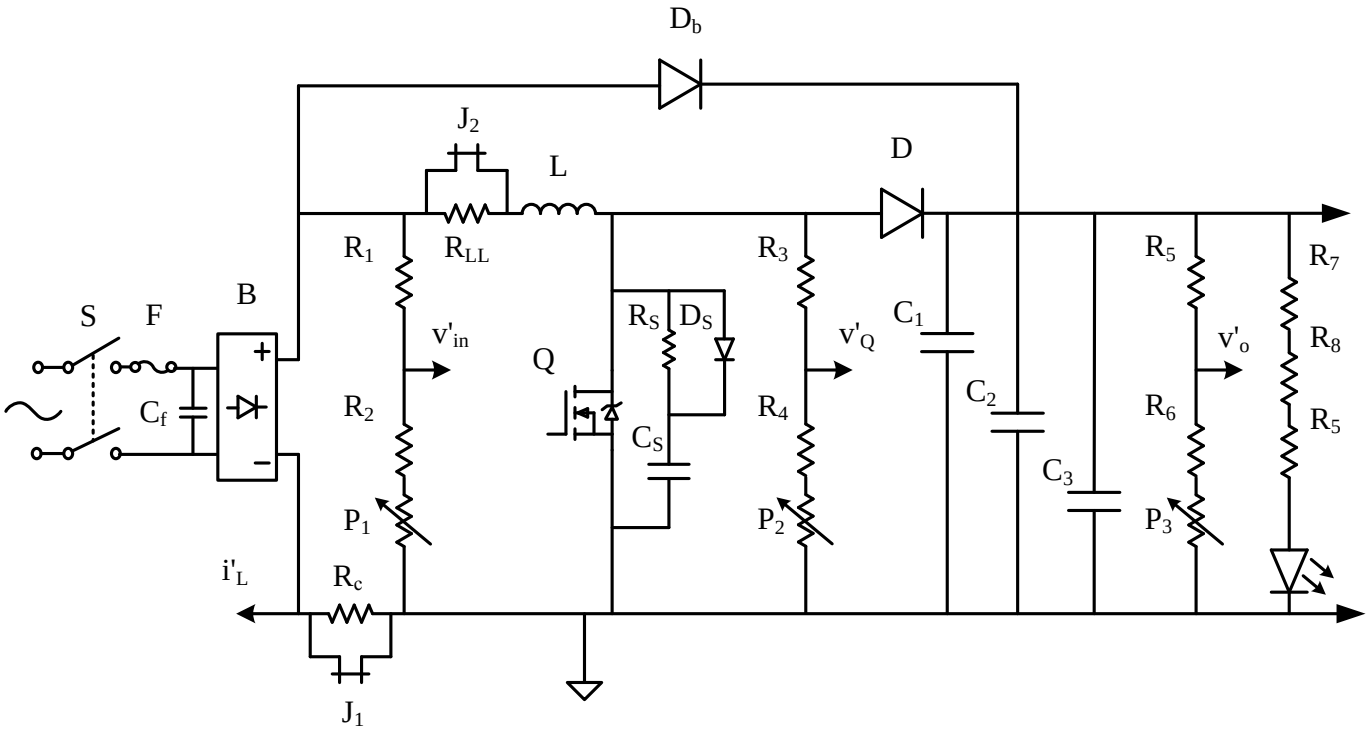


Figure 4.1: Hardware implementation of the circuit [1].

Table 4.1: Circuit elements [1].

Circuit element	Parameters
inductor L	resistance $R_L = 1.96\ \Omega$ inductance $L = 17.8\ mH$ detailed description in Section 4.2
transistor Q	power MOSFET International Rectifier, IRF840A $V_{DSS} = 500\ V$, $I_D = 8\ A$
diode D	stealth diode Fairchild Semiconductor, ISL9R460PF2 $V_{RRM} = 600\ V$, $I_F = 4\ A$
output capacitor C_1	aluminum electrolytic capacitor Panasonic, ECO-S2WB271DA $C_1 = 270\ \mu F$, rated voltage 450 V
output capacitor C_2	ceramic capacitor AVX, SV09AC105KAR $C_2 = 1\ \mu F$, rated voltage 1000 V
output capacitor C_3	ceramic capacitor TDK Corporation, FK26X7R2J103K $C_3 = 10\ nF$, rated voltage 630 V
bridge rectifier B	single phase diode bridge rectifier GBU806 $I_{(av)} = 8\ A$
current sense resistor R_C	wirewound resistor with low-inductive Ayrton-Perry winding Vishay, MRA-05R5000FE12 $R_C = 0.5\ \Omega$, 5 W
voltage divider resistors $R_1 \ldots R_6$	$R_1 = 470\ k\Omega$, ¼ W $R_2 = 3.25\ k\Omega$ ½ W (2 parallel 7.5 $k\Omega$, ¼ W) $R_3 = 470\ k\Omega$, ¼ W $R_4 = 2.7\ k\Omega$, ¼ W

	R_5 = 820 kΩ, ½ W R_6 = 2.7 kΩ, ¼ W
resistors	R_7 = 820 Ω, ½ W Caddock Electronics, MP925-15.0K-1% R_8 = 15 kΩ, 25 W R_9 = 820 Ω, ½ W Ohmite, 15FR250E R_{LL} = 0.25, 5 W
trimmer potentiometers P_1…P_3	$P_1 = P_2 = P_3$ = 2 kΩ, ¼ W
snubber capacitor C_S	metallized polyester film capacitor Kemet, R76PD1220SE00J C_S = 2.2 nF, rated voltage 630 V
snubber resistor R_S	cemented wirewound resistor Vishay, AC07000002001JAC00 R_S = 2 kΩ, 7W
snubber diode D_S	fast recovery rectifier diode Fairchild Semiconductor, EGP10J V_{RRM} = 600 V, $I_{F(av)}$ =3 A
diode D_b	Fairchild Semiconductor, 1N5406 V_{RRM} = 600 V, $I_{F(av)}$ =3 A
switch S	two-pole two-position miniature rocker NKK Switches, M2022TZW13-JB
fuse F	fast-acting glass tube fuse, rated current 5 A
C_f	ceramic capacitor AVX, SV09AC105KAR C_f = 1 μF, rated voltage 1000 V

The highest current flows through the inductor at lowest boundary of input voltage range when maximum load is supplied. The highest voltage on the primary side is equal to the peak of the input voltage.

Assuming the worst efficiency for these parameters to be 0.7 [7], then

$$I_{L\ \max} = \sqrt{2} \cdot \frac{P_{o\ \max} / \eta}{V_{in\ \min}} = \sqrt{2} \cdot \frac{200\ W}{80\ V \cdot 0.7} = 5.05\ A \tag{4.1}$$

$$V_{in\ \max} = \sqrt{2} \cdot V_{in} = \sqrt{2} \cdot 260\ V = 367.7\ V \tag{4.2}$$

It appears that all the elements were capable of withstanding the desired voltage due to the initially designed rating margins. The next step was to determine whether the circuit, being supplied with higher input voltage, still matches the initial design requirements. It was determined that the following modifications are necessary.

4.1 Voltage divider ratio for input voltage sensor

The converter is controlled by Texas Instrument digital signal processor (DSP) TMS320F2812 embedded on an evaluation board eZdsp F2812 by Spectrum Digital [4,5]. The DSP has an analog to digital converter (ADC) with 0-3 V analog input [6]. The implemented circuit has four voltage dividers for scaling the sensed values down to acceptable levels. The scaling factor in this case is

$$\frac{R_2 + P_1}{R_1 + R_2 + P_1} \tag{4.3}$$

Assuming that the potentiometer P1 is adjusted such that the voltage divider output is below 3 V, the scaling factor of the existing voltage divider would be:

$$\sqrt{2} \cdot 132 \cdot \frac{R_2 + P_1}{R_1 + R_2 + P_1} < 3V \Rightarrow P_1 = \frac{R_1 + (1 - \sqrt{2} \cdot \frac{132}{3}) \cdot R_2}{\sqrt{2} \cdot \frac{132}{3} - 1} = 0.177\ \Omega \tag{4.4}$$

In order to accommodate higher input voltage, R_2 was halved and P_1 readjusted, such that

$$\sqrt{2}\cdot 260\cdot\frac{0.5\cdot R_2+P_1}{R_1+0.5\cdot R_2+P_1}<3V\Rightarrow P_1=\frac{R_1+(1-\sqrt{2}\cdot\frac{260}{3})\cdot 0.5\cdot R_2}{\sqrt{2}\cdot\frac{260}{3}-1}=0.116\ \Omega \quad (4.5)$$

4.2 Boost inductor

Initially, the inductance was chosen such that the inductor current ripple stays below 25% of the peak inductor current. Peak current ripple happens when $V_{in} = V_o/2 = 380/2 = 190V$ [7] as shown in Fig. 4.2.

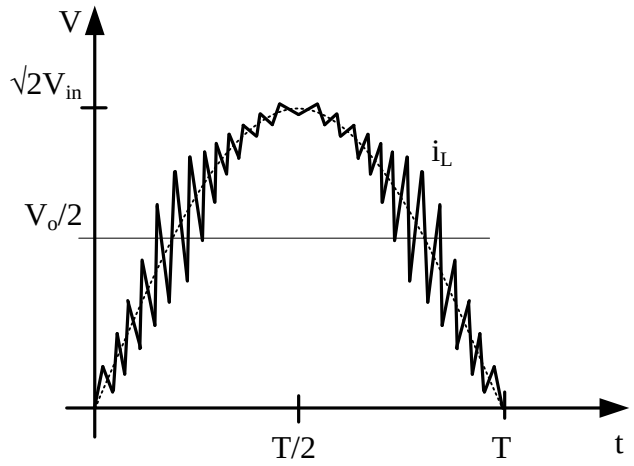


Figure 4.2: Inductor current ripple

Given that initially, the V_{in} peak was equal to $\sqrt{2}\times 120 = 170$ V and did not reach the Vo/2 value, maximum current ripple was at that input voltage. In the context of higher input, according to [7], the peak current ripple will be:

$$I_{L\ \max\ rip}=\frac{V_o}{4\cdot f_s\cdot L} \quad (4.6)$$

From where:

$$L=\frac{V_o}{4\cdot f_s\cdot 0.25\cdot\sqrt{2}\,\frac{P_o}{V_{in\ \max}}}=\frac{380}{4\cdot 20000\cdot 0.25\cdot\sqrt{2}\cdot\frac{200}{260}}=0.0175\ H \quad (4.7)$$

Therefore, it was decided that a new power inductor should be designed. Several factors were taken into account: the linearity of the hysteresis curve, the size and shape of a core, and inductor fabrication costs. Micrometals, Inc. was chosen as a powder core supplier. The design was performed using the Inductor Design Software, a tool provided by the manufacturer. Resulting inductor details are given in Table 4.2.

Table 4.2: Inductor parameters

Core	type	E shaped iron powder core
	part number	E305-2
	core material	mix No. -2
	length [in/mm]	3.051/77.49
	width [in/mm]	3.051/77.49
	height [in/mm]	0.933/23.7
	average AC flux density, *B* [G/mT]	86/8.6
	AL value, [nH]	75
Winding	*N*	489
	wire gauge	AWG 19
	wire diameter [in/mm]	0.039/0.991
	wire length [ft/m]	226.9/69.156
	theoretical DC resistance, R_L [Ω]	2.08
	measured DC resistance, R_L [Ω][1]	1.96
	theoretical inductance, *L* [mH]	17.5
	measured inductance, *L* [mH][1]	17.8

[1] Measured with Philips PM 6303 RLC meter

Material mix. No -2 has the most linear B-H curve (Fig. 4.3) among other compounds. This prevents inductor current distortion during the second half of the period [1].

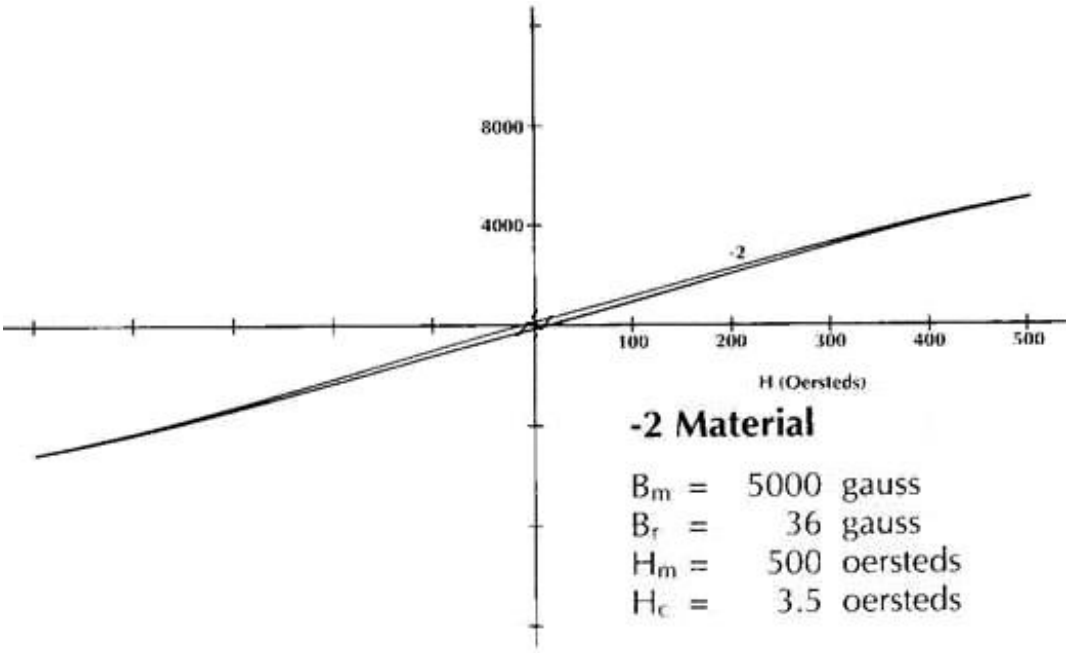


Figure 4.3: Material mix No. -2 BH curve. *Note: from* http://www.micrometals.com/

Provided that the E- shaped core consists of two halves that are assembled around a bobbin, wire winding becomes much more available locally. The specified inductor and its parameters were used for all the calculations and experiments in this report.

4.3 Signal channel noise protection

The digital signal processor collects four feedback signals: the voltage across the current sensing resistor, the input voltage, the output voltage and the voltage across the switch. Signals are transferred in several stages. First, voltages are sensed at different points of the circuit. Then, they are scaled down to an ADC acceptable level of 0-3 V. Further, according to recommendations in [6], the signals are passed through operational

amplifiers configured as a voltage follower. At the final stage, signals are passed through signal wires with decoupling capacitors at ADC inputs (Fig. 4.4).

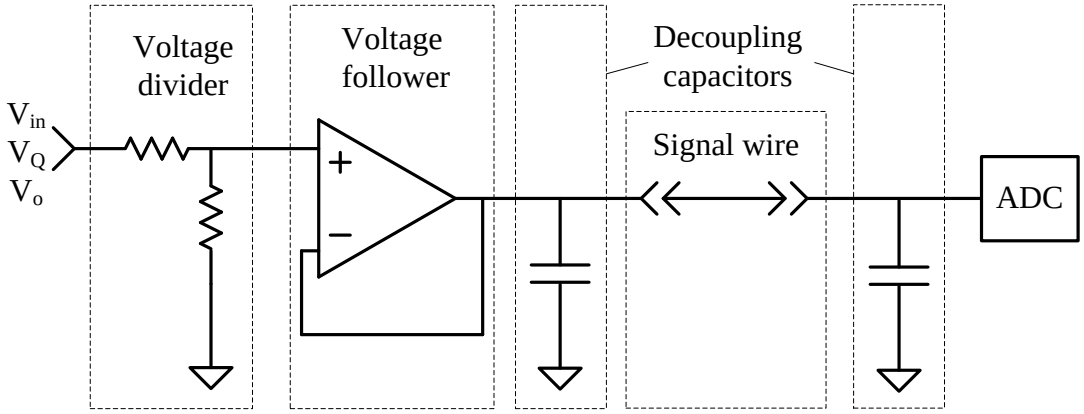


Figure 4.4: Signal transmission circuit before modification

While performing experiments, it was noticed that the signal wires received electromagnetic interference. This noise absorption severely distorted current waveform and therefore impaired overall converter performance. The channel for a voltage across the switch with high frequency discontinuous signal was the worst-affected. Utilization of a twisted pair for signal transmission would be a standard solution in case of electromagnetic compatibility issues. Further experiments proved that these measures resolved the problem.

5 Software modifications

The DSP drives the MOSFET according to the control code written in the language C. Texas Instrument provides the Code Composer Studio (CCS) [8], a powerful software for programming and communicating with the chip. CCS allows not only writing, compiling and burning a code to a processor, but also acquiring values from registers in real time. It significantly facilitates the debugging process. Apparently, the program had to undergo particular modifications, along with the hardware. The most important of them are described in the following sections. The complete code including all the improvements can be found in Appendix A.

5.1 Duty-cycle precalculation

Despite all the versatility of the code, it would not be able to return the correct control output when higher voltage is supplied without one major alteration. The DSP is timed by a 150 MHz high-speed peripheral clock. The MOSFET is driven with a constant frequency of 20 kHz and a respective period $T_s = \frac{1}{f_s} = \frac{1}{20000} = 50\ \mu s$. This frequency is generated by Event Manager 2 and leaves about $N = \frac{150\ MHz}{20\ kHz} = 7500$ processor clock cycles per switching cycle (SC). The ADC continuously samples required signals. At the n^{th} switching cycle, the processor converts data acquired during the previous $(n\text{-}1)^{th}$ switching cycle into real values. Then, within the same switching cycle *n*, it does all required calculations using the results of conversion and returns a duty-cycle value to drive the MOSFET at the n^{th} switching cycle.

According to the definition, the boost converter duty cycle *D* is [7]:

$$D = 1 - \frac{V_{in}}{V_o} \tag{5.1}$$

Since V_o is assumed to be constant, the duty cycle will have its least value at $V_{in\ \max}$. Previously, in the worst case scenario, it was:

$$D_{\min} = 1 - \frac{V_{in\ \max}}{V_o} = 1 - \frac{\sqrt{2} \cdot 132}{380} = 0.509 \tag{5.2}$$

Hence, the processor would have $7500 \cdot 0.509 = 3816$ clock cycles for the computations before it should produce the duty-cycle value. If the input voltage is to be increased up to 260 V RMS, the minimum duty cycle would be:

$$D_{\min} = 1 - \frac{V_{in\ \max}}{V_o} = 1 - \frac{\sqrt{2} \cdot 260}{380} = 0.032 \tag{5.3}$$

At this point, the processor would have only $7500 \cdot 0.032 = 242$ clock cycles. In other words, the higher the input, the less time the processor has for computation. Implementation of the control code in adaptation mode up to the moment when the duty cycle is calculated and the compare register is renewed, takes 1264 clock cycles. Hence, the minimum possible duty cycle in this case would be $\frac{1264}{7500} = 0.1685$. This value corresponds to a maximum RMS value of the input voltage:

$$V_{in\ \max} = \frac{(1 - 0.1685) \cdot 380}{\sqrt{2}} = 223.425\ V \tag{5.4}$$

So, when V_{in} was reaching this value, the processor was finishing computation of the duty cycle after the latter should have been applied to the gate of the MOSFET. It caused complete disruption of the converter operation. The most obvious solution would be optimization of the code. However, it appeared that the code does not have much room

for optimization and it is not possible to complete all the computations within 242 clock cycles. Therefore, another strategy was adopted. Since the processor does not have enough computation time to produce a duty cycle timely at the current switching cycle, it was decided to apply the current duty cycle value in the following SC. Thus, the sequence would be: *(n-1)* switching cycle - ADC acquires signals; *(n) SC* - D is computed; *(n+1)* SC - gating signal sent according to the previously calculated duty cycle. Experiments showed that this approach helps to break through the input limit but causes a noticeable distortion of a current waveform. The problem was that the duty cycle sent to the gate has a lag of three SC, or 150 μs. Therefore, a compensation for this lag was required.

Knowing that a sinusoidal signal is relatively smooth by nature and the switching frequency is much higher than the line frequency, the difference between the following duty cycle and the current was presumed to be approximately equal to the difference between the current and the previous duty cycle as illustrated in Fig. 5.1:

$$D_{(n+1)} \approx D_{(n)} + [D_{(n)} - D_{(n-1)}] \tag{5.5}$$

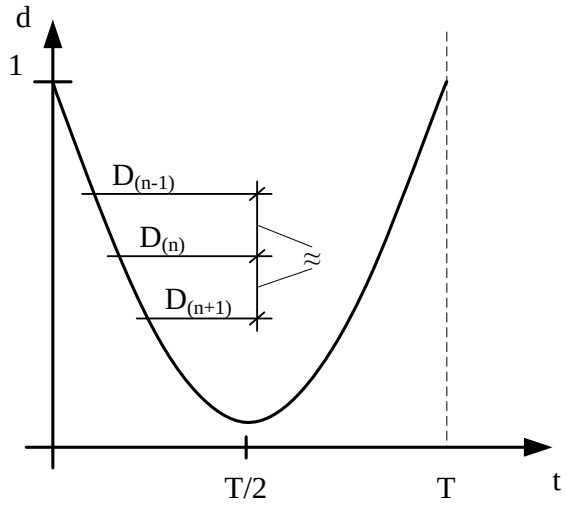


Figure 5.1: Duty cycle approximation

This mechanism indeed cannot predict the exact value of the following duty cycle, but the provided compensation gives acceptable results. Hence, the duty cycle is calculated and the value is written to the respective compare register at the beginning of the following SC. Implementation of this approach can be found in the code in Appendix A.

5.2 ADC recalibration

As mentioned before, the scaling voltage divider for the input voltage signal was altered and, therefore, the ADC requires recalibration [1]. The procedure was analogous to the one in [1]. ADC readings can be found in Table 5.1.

Table 5.1: ADC calibration measurements

V_{in} [V]	N	V_o [V]	N	V_Q [V]	N
0	300	0	800	0	200
40	7130	40	6270	40	6450
80	14220	80	12650	80	12610
120	21450	120	18980	120	18850
160	28620	160	25190	160	24940
200	35870	200	31600	200	31100
240	43200	240	37950	240	37350
280	50430	280	44480	280	43620
320	58000	320	51060	320	50010
360	65390	360	57710	360	56610
		400	64270	400	63340

These values plotted on a graph are presented in Fig. 5.2.

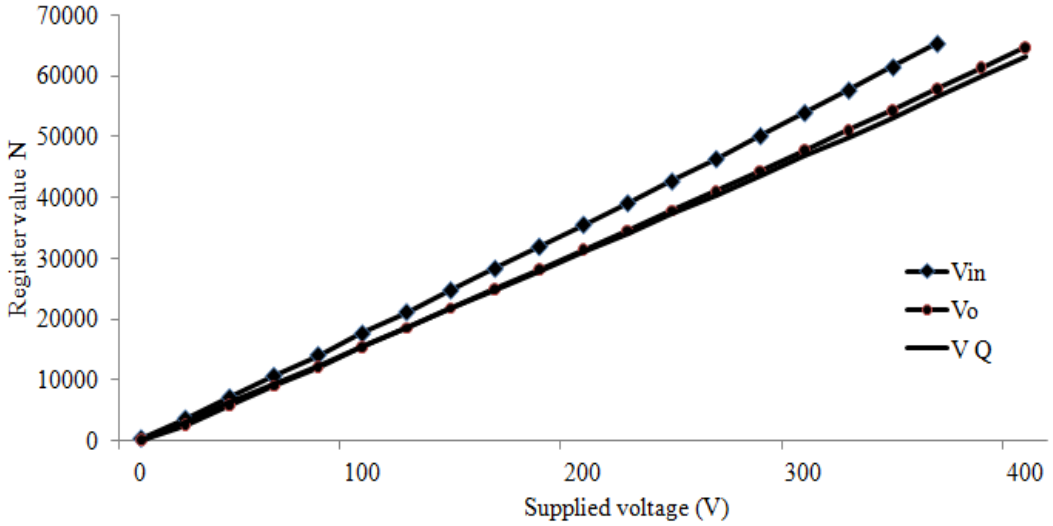


Figure 5.2: ADC calibration measurements

One can estimate the linear functions from ADC register values using MATLAB and use it to obtain the relationship between the signals and register values:

$$N = 181.13 \cdot V_{in} - 142 \rightarrow V_{in} = \frac{1}{181.13}(N + 142)\ V \tag{5.6}$$

To derive the final conversion function for the code, it is necessary to multiply the above by 16 to accommodate 4 bit right register shifts and divide the register value N by 50, the number of samples (details on the sampling frequency can be found in Section 5.3).

$$V_{in} = \frac{16}{181.13}\left(\frac{N}{50} + 142\right) = \frac{16}{181.13 \cdot 50}\left(N + 7100\right)\ V \tag{5.7}$$

A similar procedure is applied to the rest of signals (Table 5.2):

Table 5.2: ADC to real signal values conversion

Signal to ADC register linear function	Reversed linear function	Conversion to real values
$N = 181.13 \cdot V_{in} - 142$	$V_{in} = 0.005521(N + 142)$	$V_{in} = 0.0017667(N + 7100)$
$N = 159.57 \cdot V_o - 8$	$V_o = 0.006267(N + 8)$	$V_o = 0.0020054(N + 400)$
$N = 156.93 \cdot V_Q - 15$	$V_Q = 0.006372(N + 15)$	$V_Q = 0.002039(N + 750)$

5.3 ADC sample frequency

The voltage across the switch is discontinuous and is either equal to V_o, when it is off, or zero, when it is on. The processor computes its average value over a switching cycle and stores it for further computations. These average values plotted for one line half cycle would have a waveform similar to the sinusoidal input (Fig. 5.3). One can observe a noticeable fluctuation of values around the peak of the sinusoid. Since the V_Q measurement has a great impact on the calculation of current reference, it can cause current waveform distortion. Therefore, it was decided to take measures aimed at fluctuation smoothing.

Previously, the signal acquisition routine was supposed to sum 40 samples of each signal over one SC and write it to a buffer. Then, this sum was divided by the number of samples for averaging. The moment when V_Q goes from high to low is defined by the duty cycle, which in turn inevitably fluctuates from cycle to cycle. Therefore, it might happen that at one SC, the ADC would collect for example 37 “high” samples and 36 or 38 at another (Fig. 5.4). In practice, it would mean that there is

$\frac{38}{40} \cdot V_o - \frac{37}{40} \cdot V_o = \frac{1}{40} \cdot 380 = 9.5\ V$ of difference between adjacent average values.

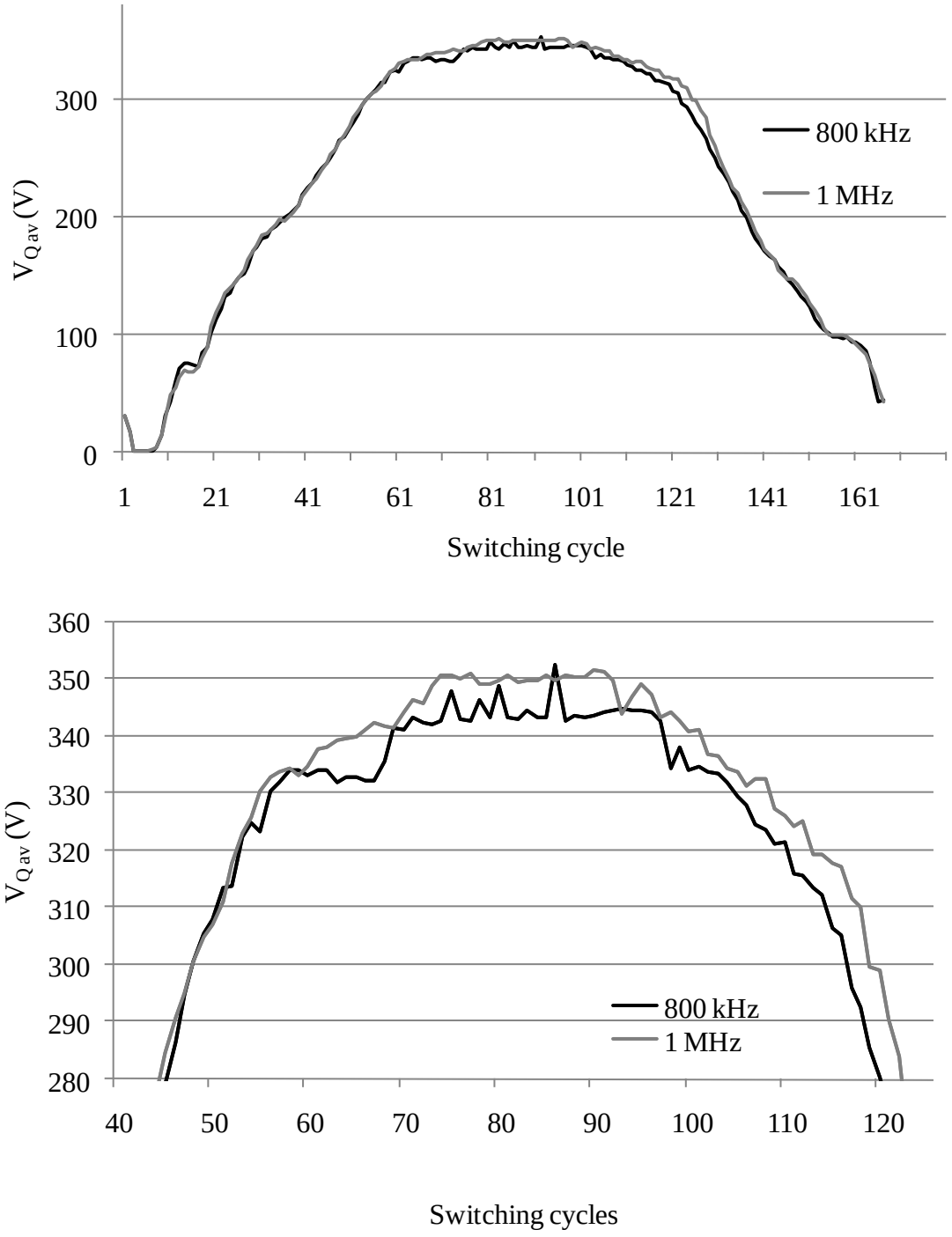


Figure 5.3: Averaged measured VQ over one line half cycle. Line half cycle (first) and enlarged peak portion (second)

This phenomenon causes the mentioned fluctuation around peak values of the input voltage. A rise in sampling frequency would relieve the problem.

The analog-to-digital converter TMS320F2812 potentially has 25 MHz sampling frequency capability [6]. The previous sampling frequency was $20000 \cdot 40 = 800\ kHz$. Now, the duty-cycle precalculation approach helped to release some computation power within each SC. However, the following limitations should have been taken into account as well. Each signal acquisition interrupt takes around 40 clock cycles [1]. Also, the main routine execution takes a significant amount of processing time. In practice, a sampling frequency of 1 MHz or 50 samples per SC was achieved. The flattening of fluctuations can be seen in Fig. 5.3.

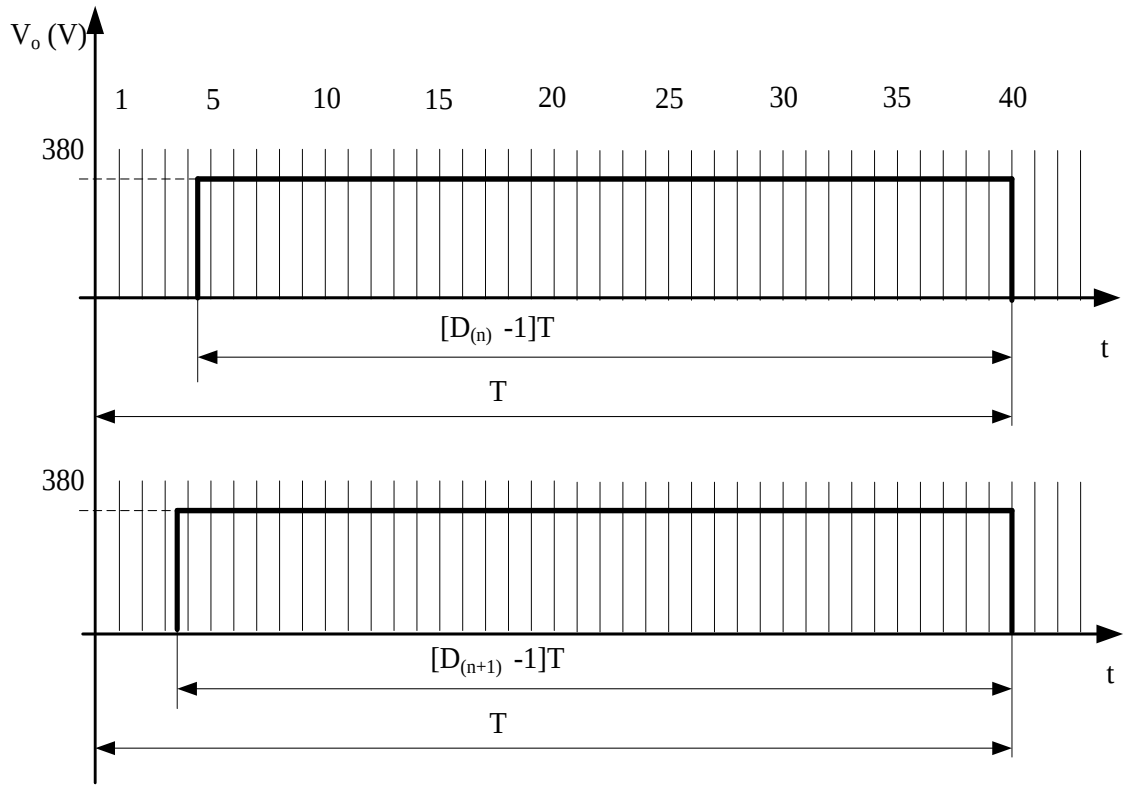


Figure 5.4: Discontinuous signal sampling

5.4 Software fault protection

The given prototype has both software and hardware protection against an overvoltage or overcurrent. The hardware protection is implemented as a fast acting glass fuse, rated for 5 A. Software protection should switch the converter off if the measured output voltage or inductor current exceeds a particular level. Early experiments show that both protections are ineffective. The fuse is too slow to blow out before the power MOSFET gets damaged. Software protection reacts to feedback from a circuit which is slow and absent when the sensing resistor is shorted or when current is not sensed at all. Also, it was determined that faults in the given circuit happen when the processor sends a large or even unity duty cycle to the gate of the MOSFET, while input voltage is already too far from zero crossing. In this case, the switch happens to be shorted at high voltage. This might be caused by transient or miscomputation. Hence, excessively high current is detected too late to avoid circuit damage.

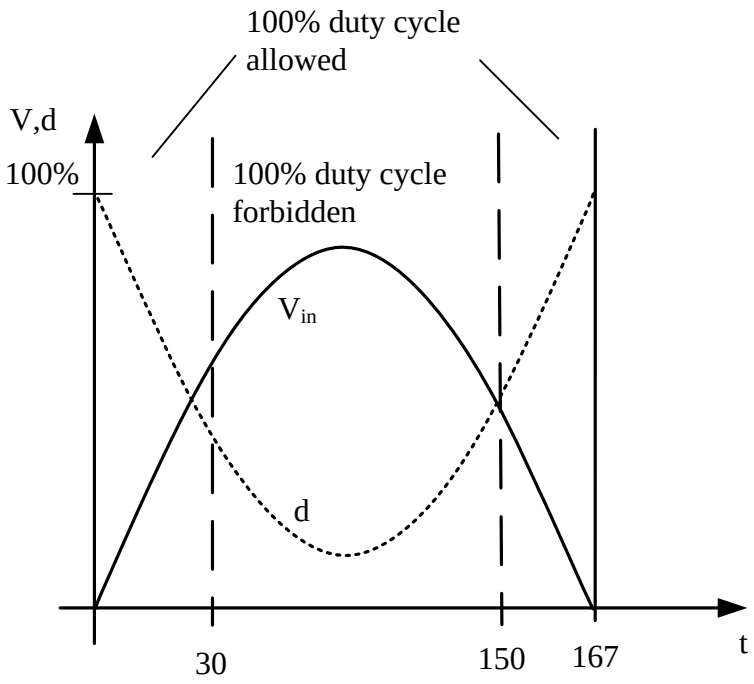


Figure 5.5: Software protection

New protection logic is based on the fact that faults were caused by an erroneously high duty cycle. The rectified input has a frequency of 120 Hz. Respectively, there are $\frac{20000\ Hz}{120\ Hz} = 166$ SC per line half cycle. The new protection safely switches off the MOSFET if the computed duty cycle is 100% between the 30^{th} and 150^{th} switching cycle as illustrated on Fig. 5.5.

This saves the circuit from excessive current and voltage. The benefit of this approach is that the protection trips before an erroneous duty cycle is applied and current becomes dangerously high. The effectiveness of the applied approach was proven by numerous experiments as follows. In fact, not a single case of hardware damage was registered after the protection had been put into operation, whereas before it, was happening repeatedly.

5.5 Zero crossing detection correction

The point where the line voltage crosses the zero level serves as a reference point for numerous computations and, therefore, should be detected in a reliable and consistent manner. The input voltage signal is used for zero crossing detection (ZCD). As it is known, the signal acquisition system receives a rectified sinusoid. Taking into account electromagnetic interference and respective distortion, ADC inaccuracy, switching frequency step-like data input etc., one can anticipate that the register values for signals do not always go to zero. This stipulates the necessity of a sophisticated ZCD procedure. Such procedure was developed and works in the following way. Ten consecutive values of input voltage signal are averaged to alleviate noise and acquisition errors. Then the average is compared with a threshold derived from the peak value experimentally to

compensate for the natural lag in computing the average. This approach allows accurate detection of the zero crossing at any level of input.

Despite thorough theoretical analysis, it was noticed that the ZCD algorithm does not work correctly in some conditions. Careful search in the code revealed that the signal average was being miscalculated. The sum of ten samples was divided by eight instead of ten. This discrepancy was eliminated and the ZCD has been working correctly since then.

6 Experimental results

The following experiments were undertaken to prove that the converter is able to operate in desired conditions. Experiments were concentrated around six operating points: 80 VAC RMS / 120 VAC RMS/ 260 VAC RMS input at rated (200W) and half (100W) resistive load.

6.1 Test setup

The test setup (Fig. 6.1) consisted of the following components:

- Designed converter
- Step-up transformer: Hammond H HQ4P, 1 phase, 120/240 V, 2000 VA, 60 Hz
- Variac: Powerstat 136B, 1 phase 120/0-140 V, 22 A, 3.1 kVA
- Variable load resistance
- DC breaker

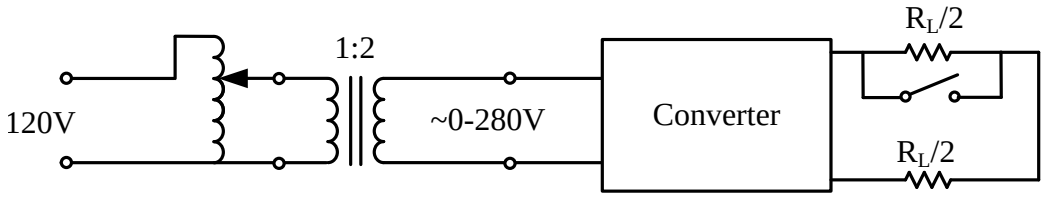


Figure 6.1: Test setup

The variac is necessary to precisely adjust the voltage to the desired level and allows increasing input voltage gradually from zero as a start up safety precaution. The step-up transformer boosts the voltage from the variable transformer and provides galvanic isolation from the supply utility. The variable load consists of several power resistances

that can be individually connected. The breaker is used for partially bypassing load resistances to simulate a step load change (c.f. Fig. 3.1).

List of measurement equipment:

– oscilloscope: digital four-channel oscilloscope Tektronix TDS 224,

– voltage probes: high voltage differential probes Tektronix P5200,

– current probe: Tektronix A622, bandwidth: 100 kHz,

– multimeter to measure V_{in}: digital multimeter Philips PM 2519,

– multimeter to measure V_o: digital multimeter Fluke 8010A,

– multimeter to measure I_o: digital multimeter Fluke 8050A,

– data acquisition module to store measurements in the computer: National Instruments USB-6259 BNC, 16 16-bit analog inputs, max. 1,250,000 samples per second divided by number of used channels.

Experiments were performed in all three modes of operation: with sensed current feedback, with computed current feedback without parameter adaptation and with computed current and parameter adaptation algorithm enabled. Conditions listed below were tested: steady state at various input voltage and load levels as well as transients at turn on and after step load change.

Converter operating at 80 V RMS input and full 200W load with measured current feedback requires highest input current.

6.2 Steady state operation.

First set of data was taken at 80 V (Fig. 6.2), 120 V (Fig. 6.3) and 260 V RMS (Fig. 6.4) input:

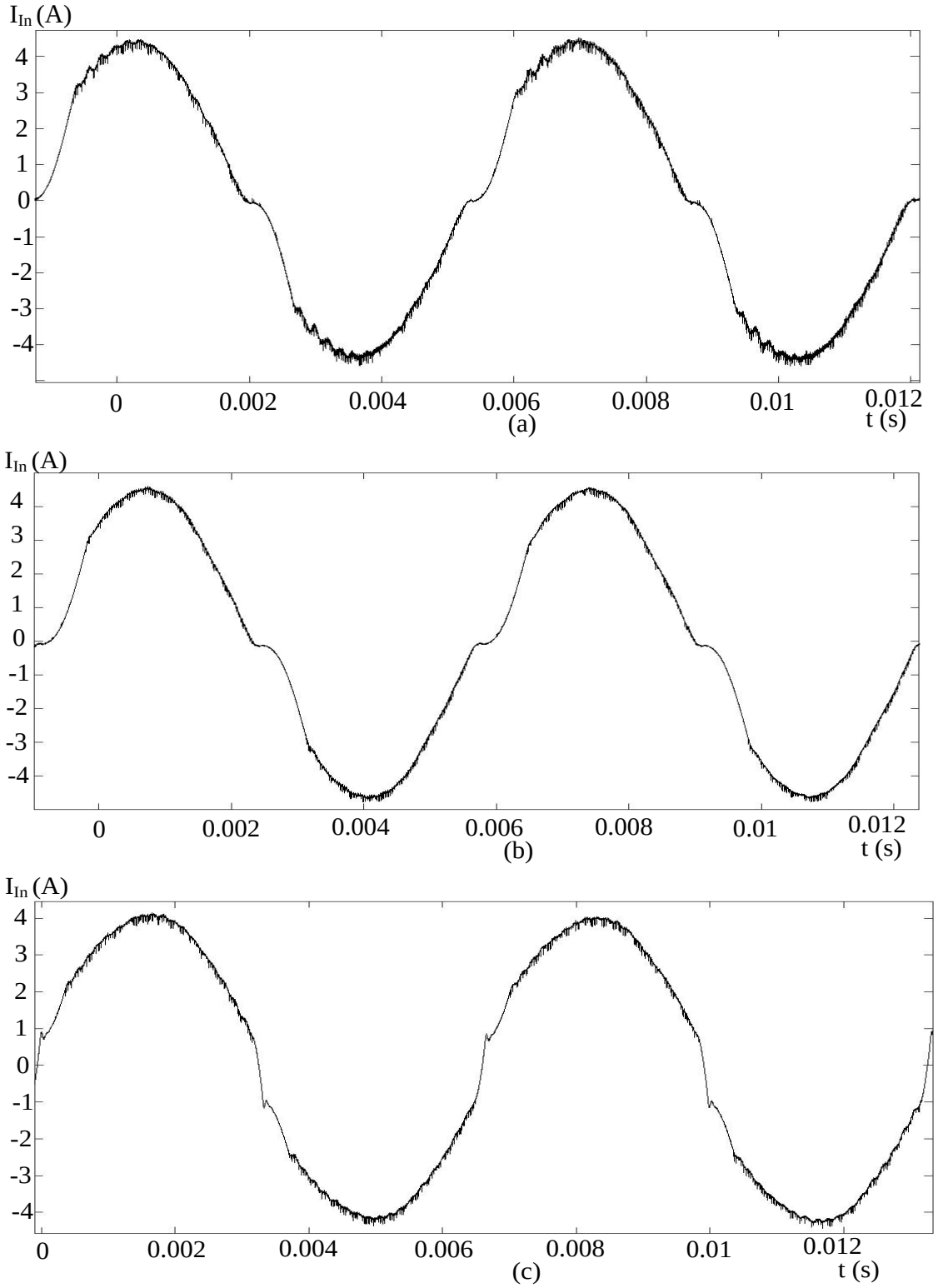


Figure 6.2: Line current at Vin = 80V and 200W load (a) measured feedback, (b) computed feedback, (c) computed feedback with parameter adaptation enabled

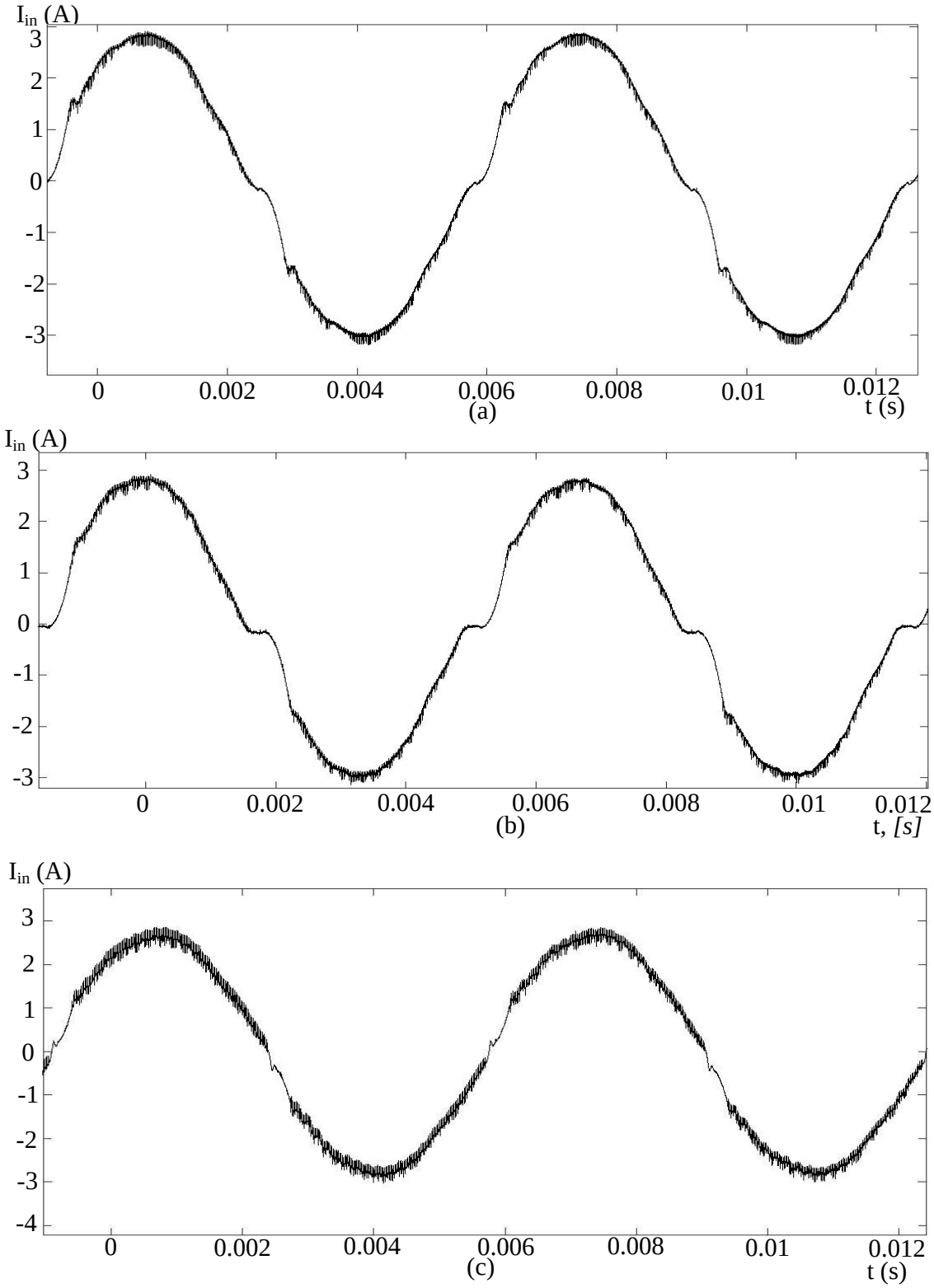


Figure 6.3: **L**ine current at V_{in} = 120V and 200W load (a) measured feedback, (b) computed feedback, (c) computed feedback with parameter adaptation enabled

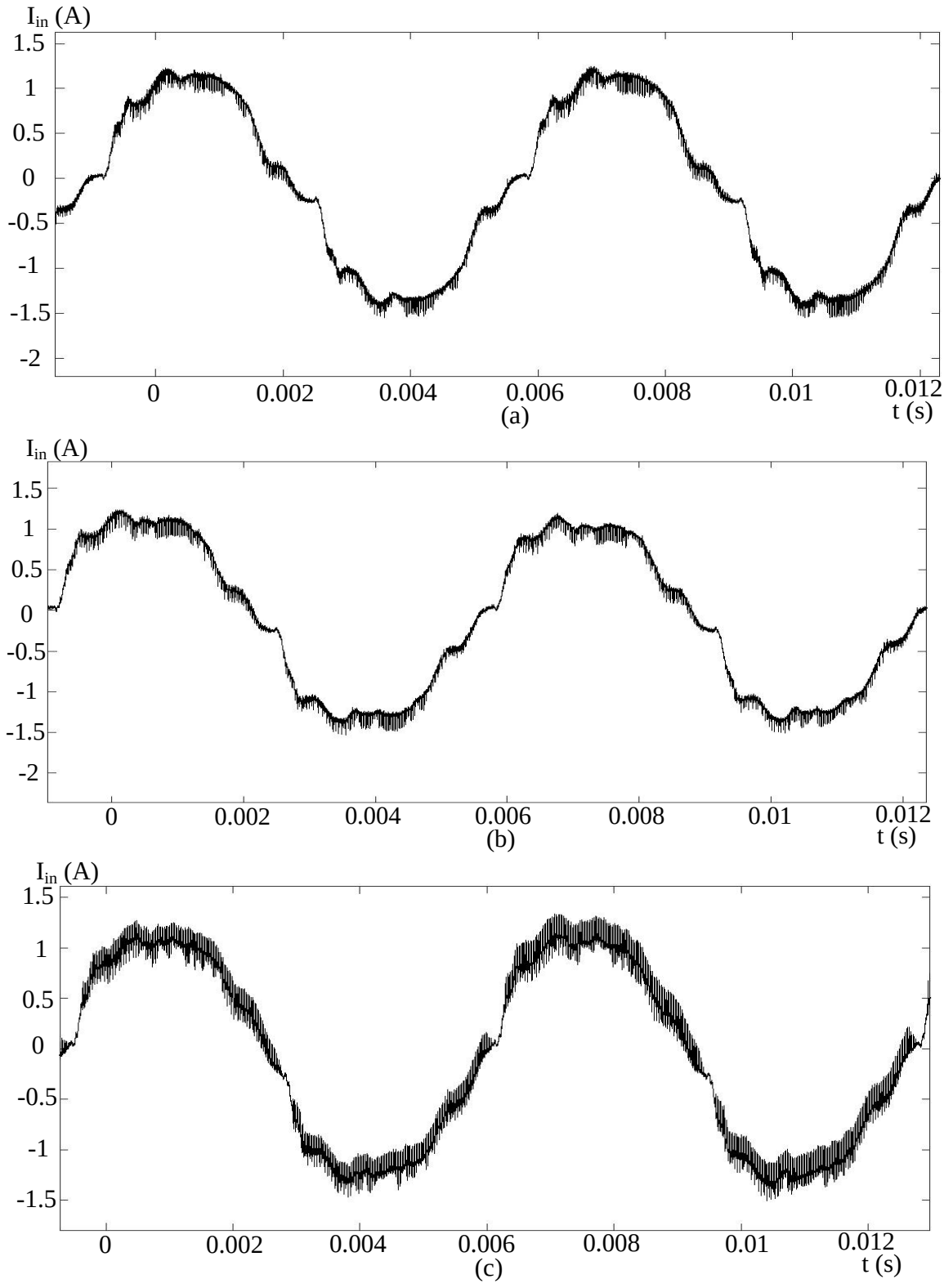


Figure 6.4: Line current at V_{in} = 260V and 200W load (a) measured feedback, (b) computed feedback, (c) computed feedback with parameter adaptation enabled

Previously mentioned measured parameters of the inductor (17.8 mH and 1.96 Ω) were plugged for computation of the parameters. These values do not reflect any resistance and inductance deviations due to temperature or voltage changes. Such deviations introduce noticeable distortion to the current waveform (see Fig. 6.2 b – 6.4 b). In order to improve the waveform it is necessary to perform fine manual calibration and plug in individual L and R_L for each of the built in the future devices. Fig. 6.2 c – 6.4 c illustrate significant improvements of line current waveform in terms of low order harmonics (see Section 6.4) provided by the parameter adaptation algorithm without any additional manual adjustment. High frequency noise has low power and can be easily filtered out.

6.3 Transients

Observation of the transient response provides important information about controller performance and circuit behaviour. Relationships between parameters employed in the parameters adaptation algorithm hold only at steady state operation [1]. Therefore, the performance of the converter with parameter adaptation procedure enabled was not tested in the present project.

The capability to operate normally at turn on with computed current feedback was examined first. In order to capture respective waveforms (Fig. 6.5 – 6.7) at different operating points, the converter was turned on by the switch S (see Fig. 4.1) at preset input voltage.

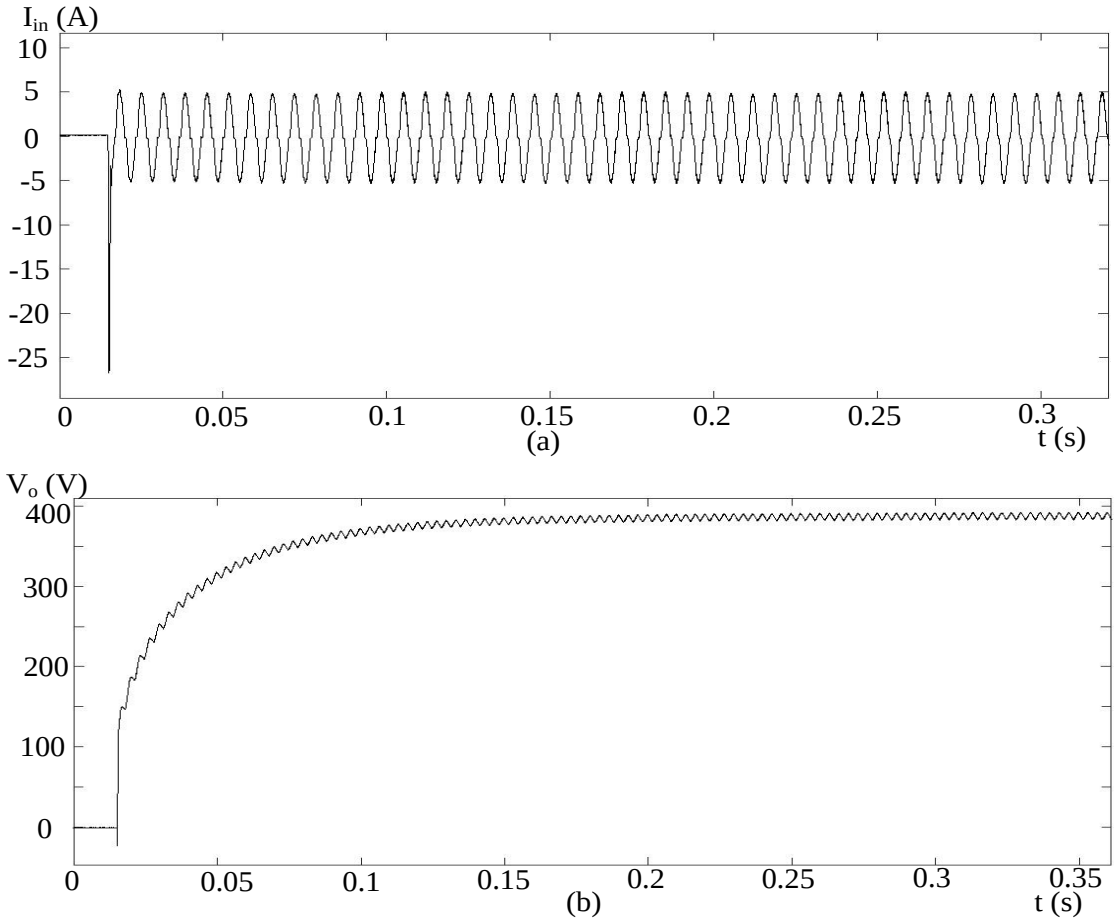
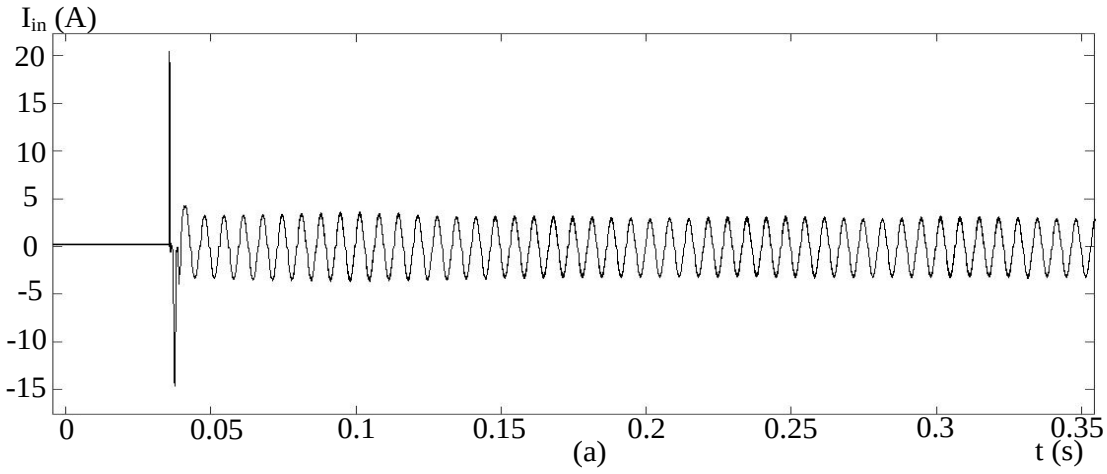


Figure 6.5: Turn on at 80 V RMS input and 200 W load (a) line current, (b) output voltage



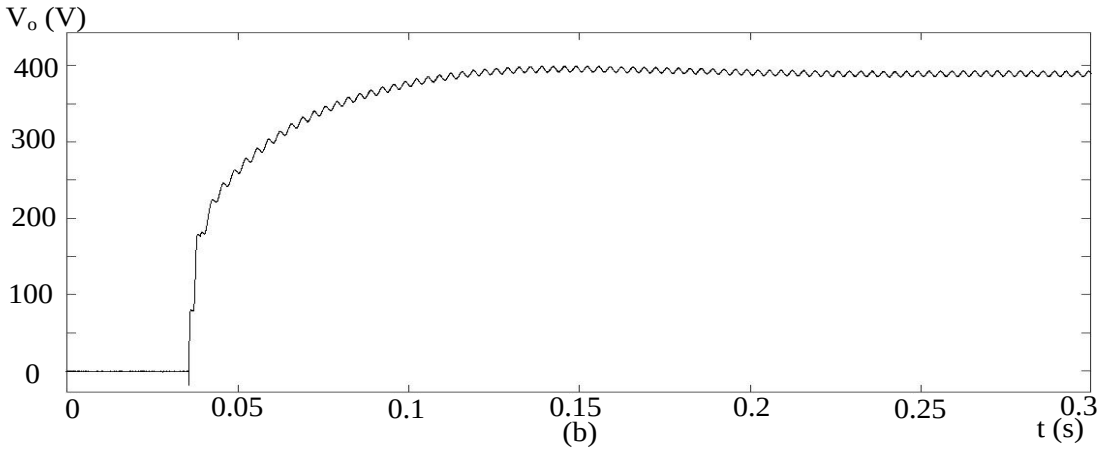


Figure 6.6: Turn on at 120 V RMS input and 200 W load (a) line current, (b) output voltage

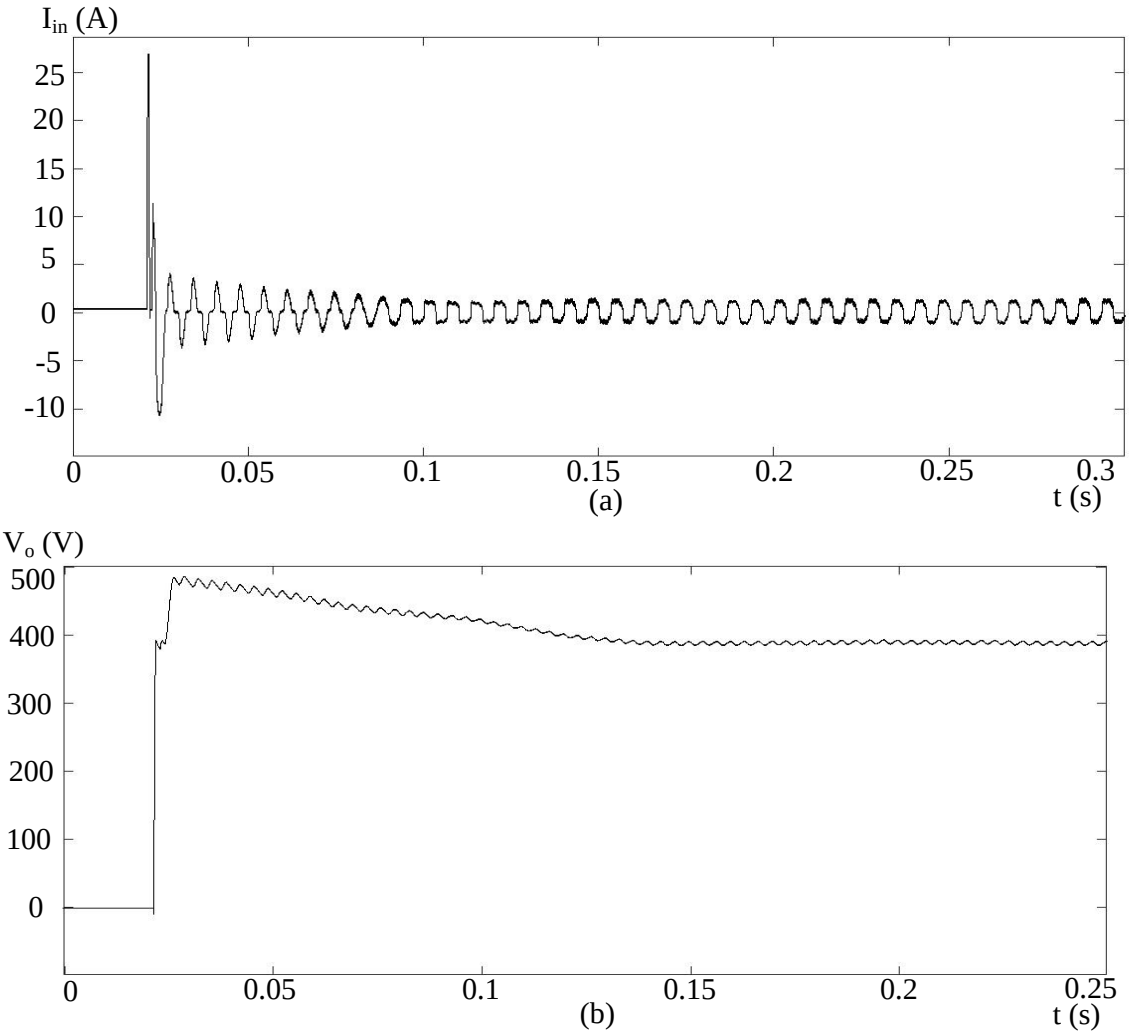


Figure 6.7: Turn on at 260 V RMS input and 200 W load (a) line current, (b) output voltage

The pictures above reveal current leaps of large amplitude at all operating points. Also, at turn on with 260 V RMS supplied output voltage reaches almost 500 V value. Thus the electrolytic and other noise suppression capacitors at the output should be chosen with proper voltage rating.

The next stage was testing of converter's performance during step load change from 200 W to 100 W and other way around. Captured line current and output voltage waveforms are in Fig.6.8 – 6.19:

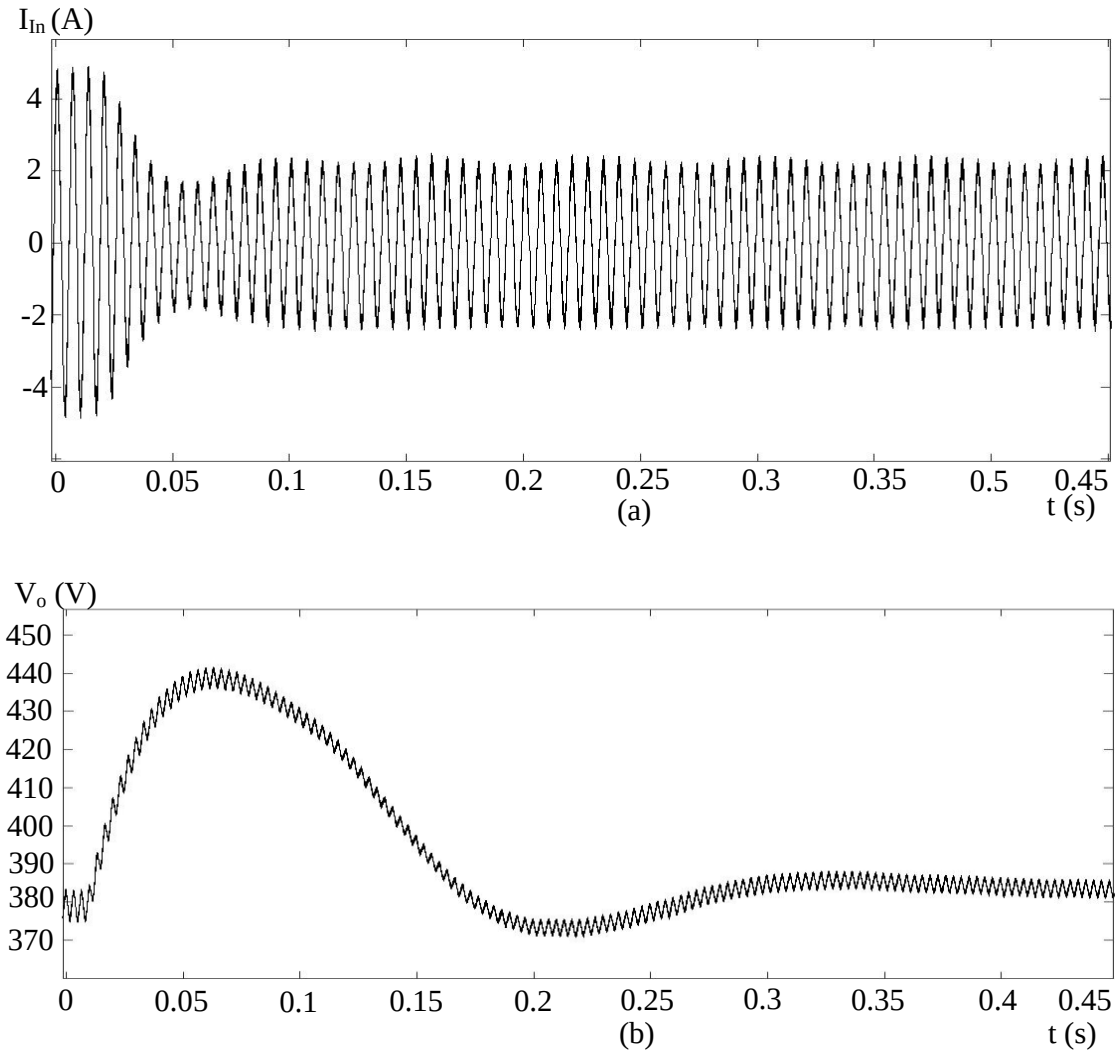


Figure 6.8: 200 W to 100 W step load change at 80 V RMS input and measured feedback, (a) line current, (b) output voltage

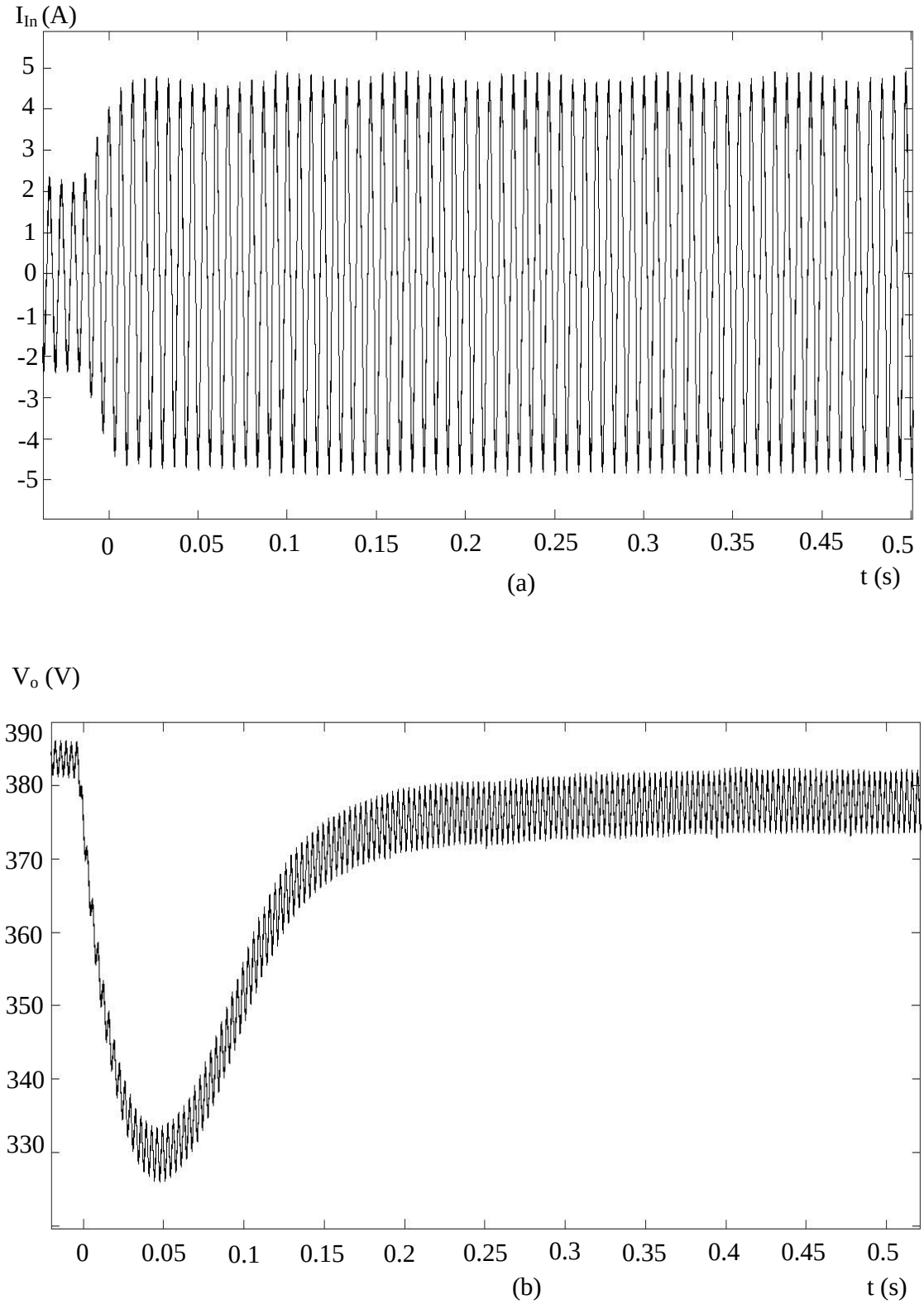


Figure 6.9: 100 W to 200 W step load change at 80 V RMS input and measured feedback, (a) line current, (b) output voltage

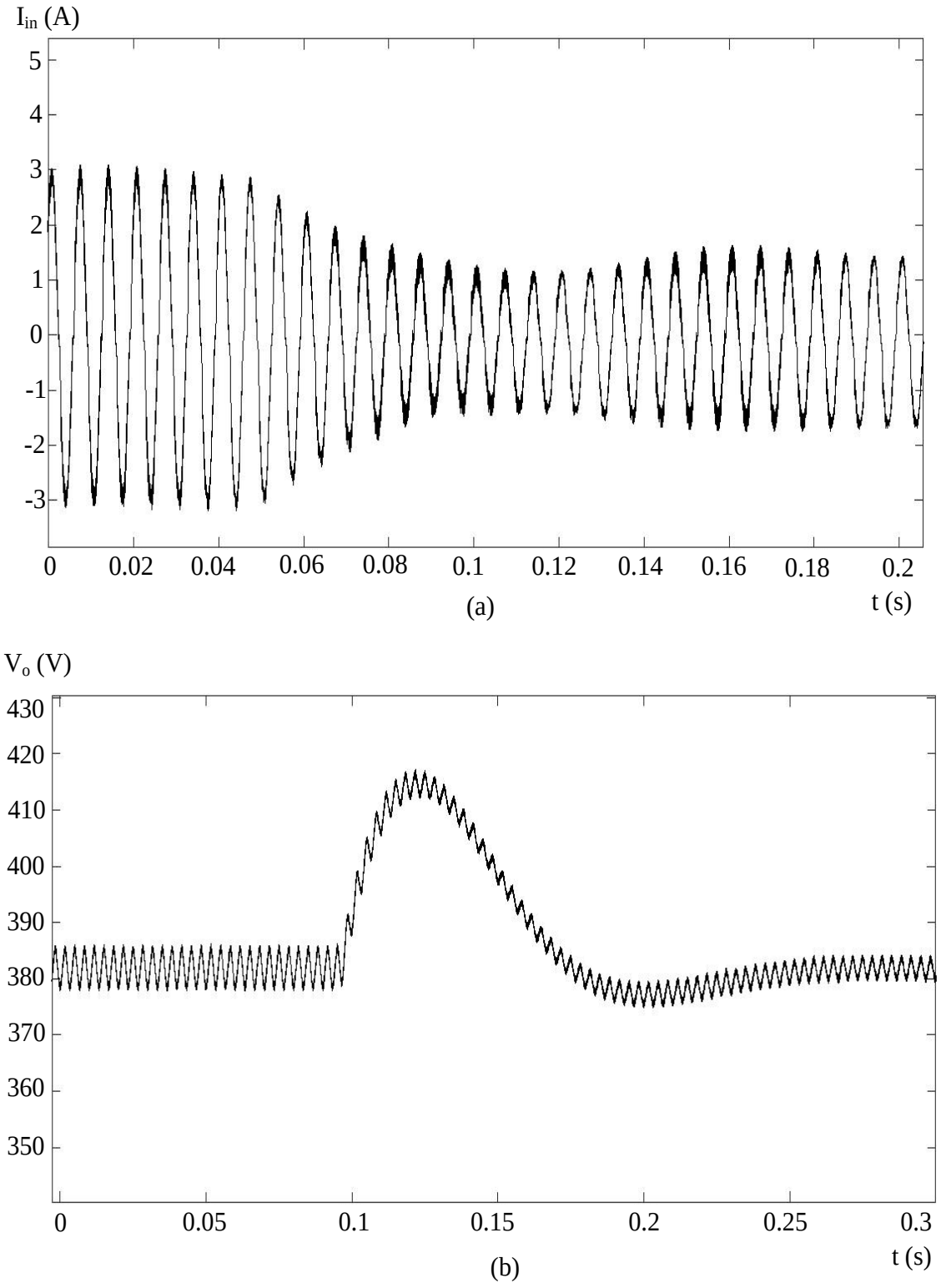


Figure 6.10: 200 W to 100 W step load change at 120 V RMS input and measured feedback, (a) line current, (b) output voltage

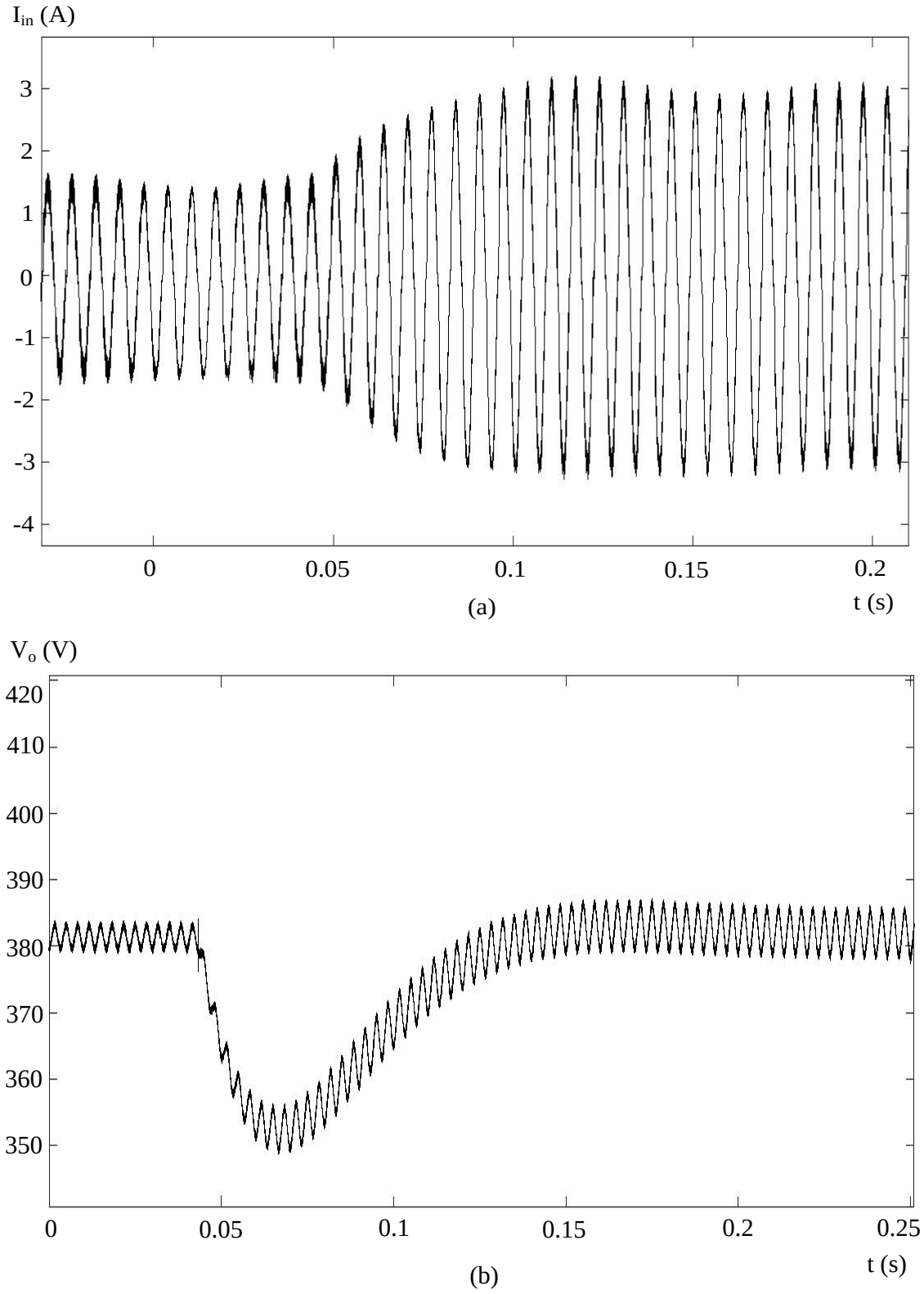


Figure 6.11: 100 W to 200 W step load change at 120 V RMS input and measured feedback, (a) line current, (b) output voltage

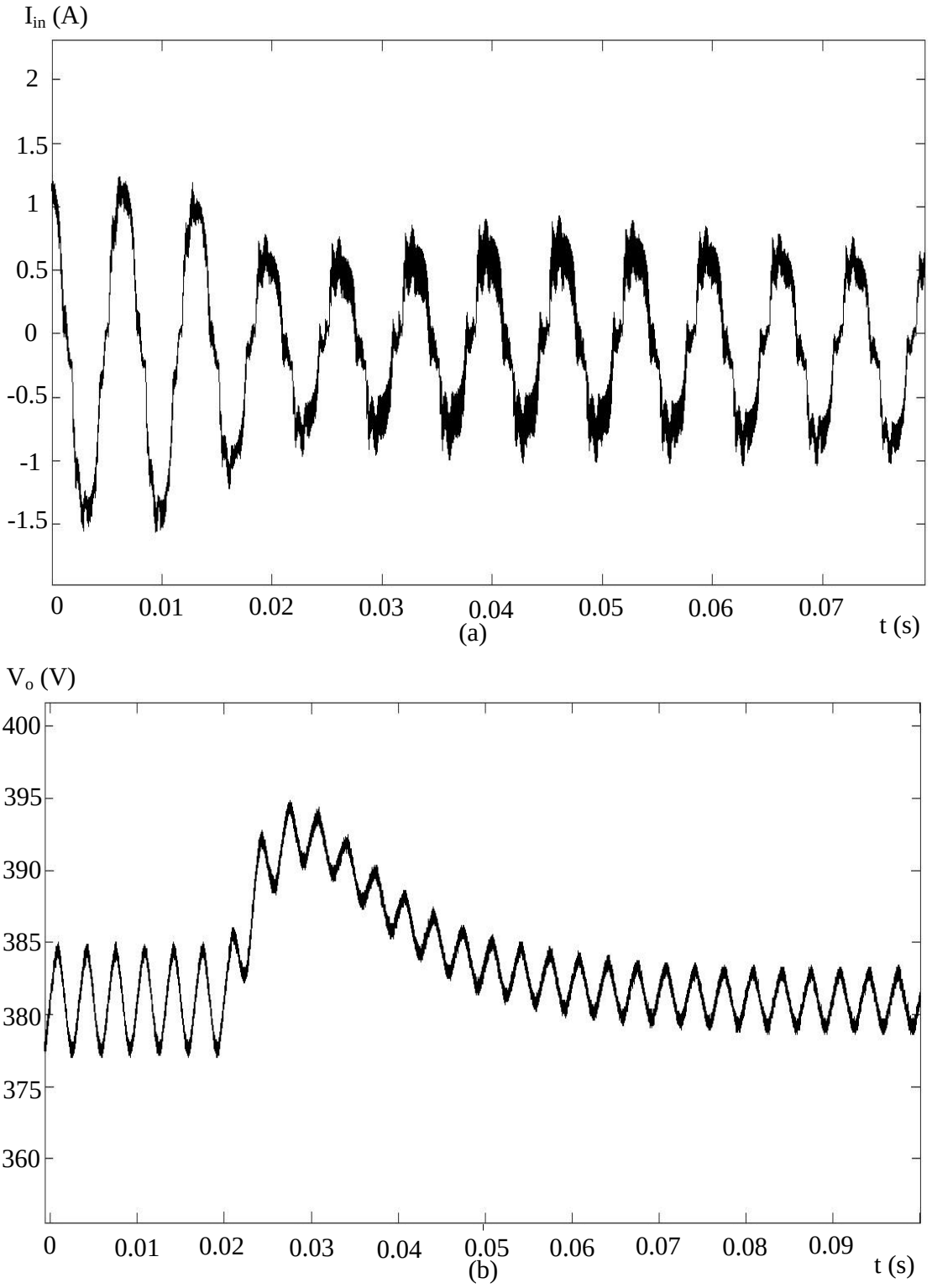


Figure 6.12: 200 W to 100 W step load change at 260 V RMS input and measured feedback, (a) line current, (b) output voltage

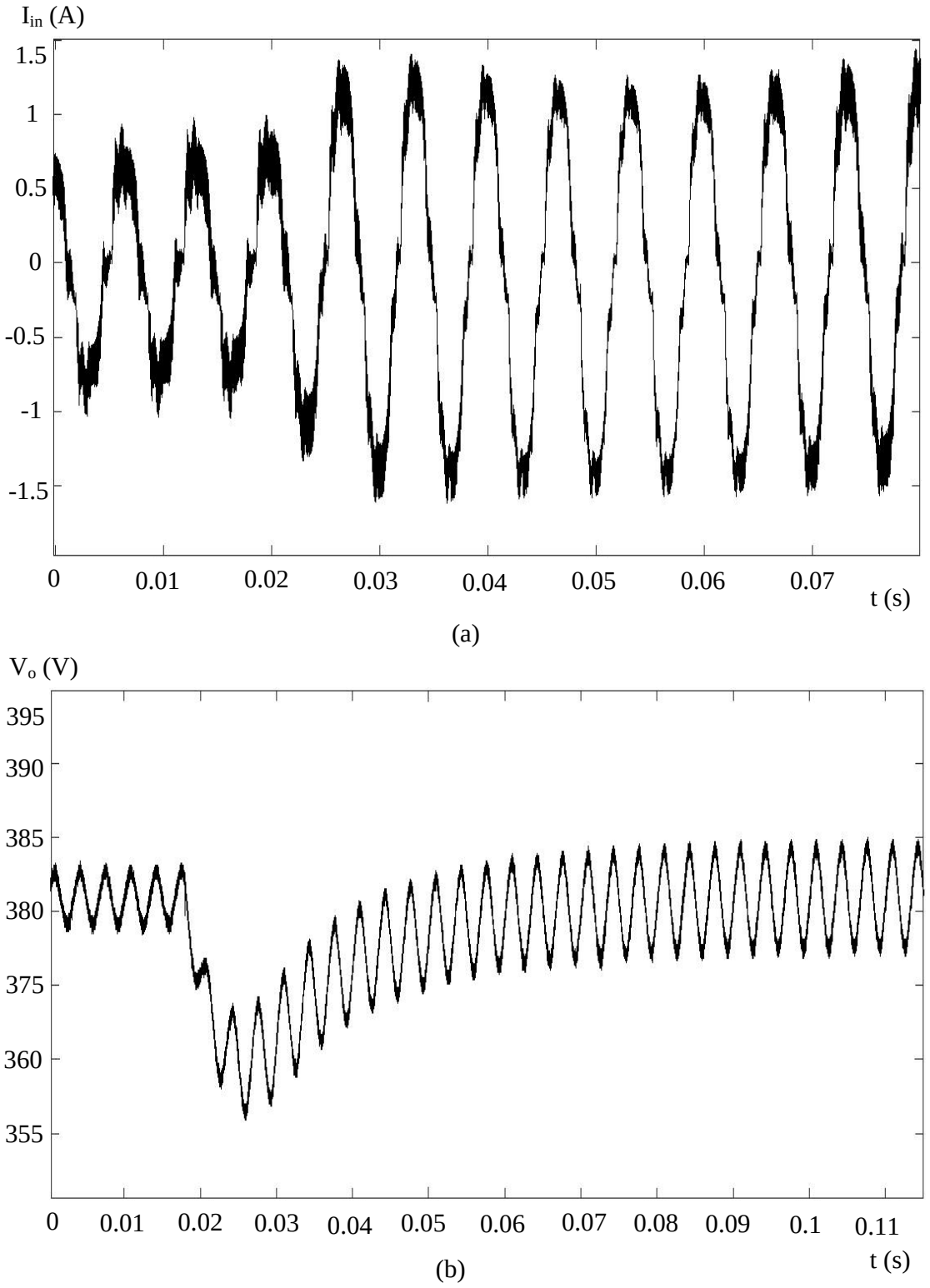


Figure 6.13: 100 W to 200 W step load change at 260 V RMS input and measured feedback, (a) line current, (b) output voltage

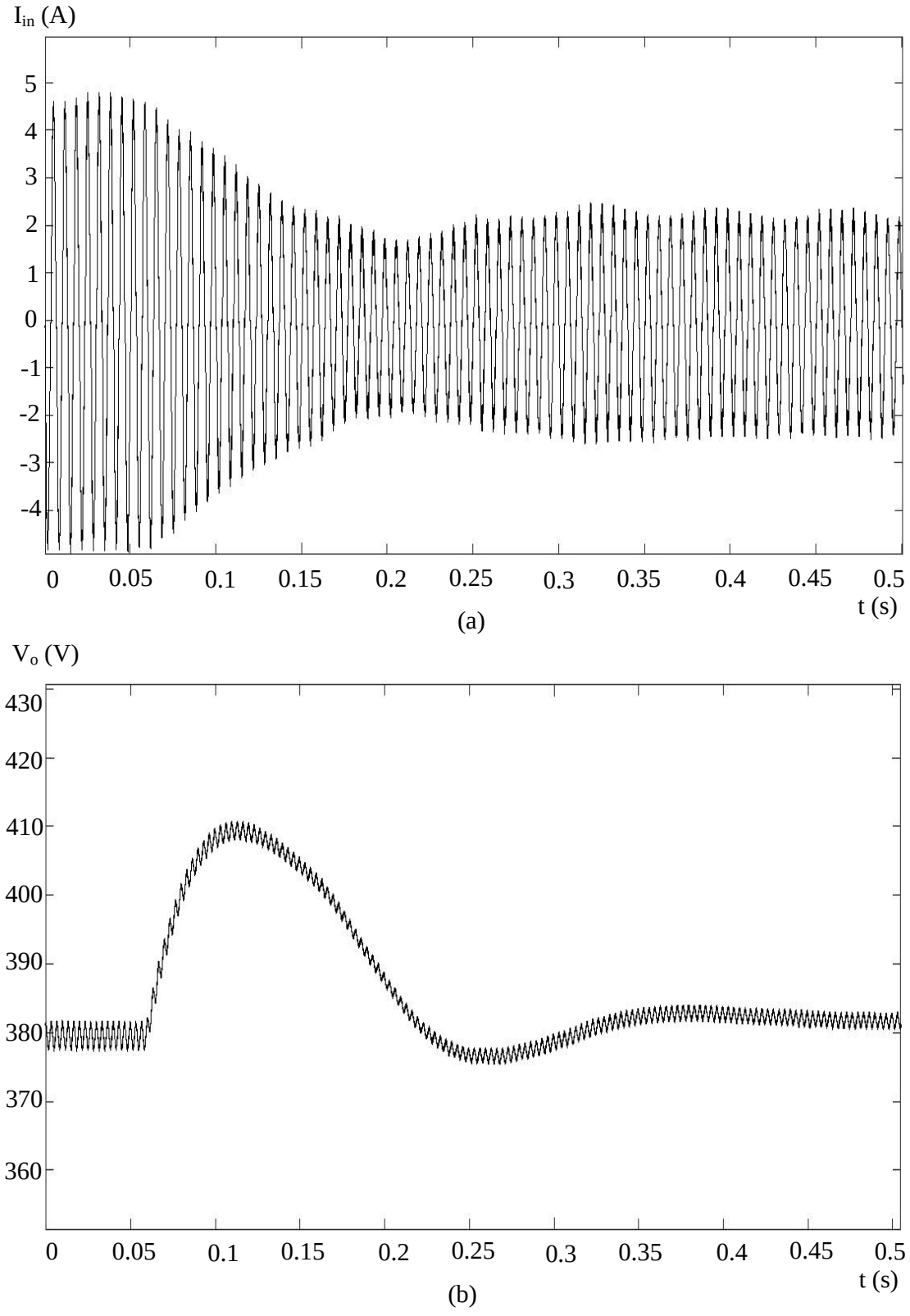


Figure 6.14: 200 W to 100 W step load change at 80 V RMS input and computed feedback, (a) line current, (b) output voltage

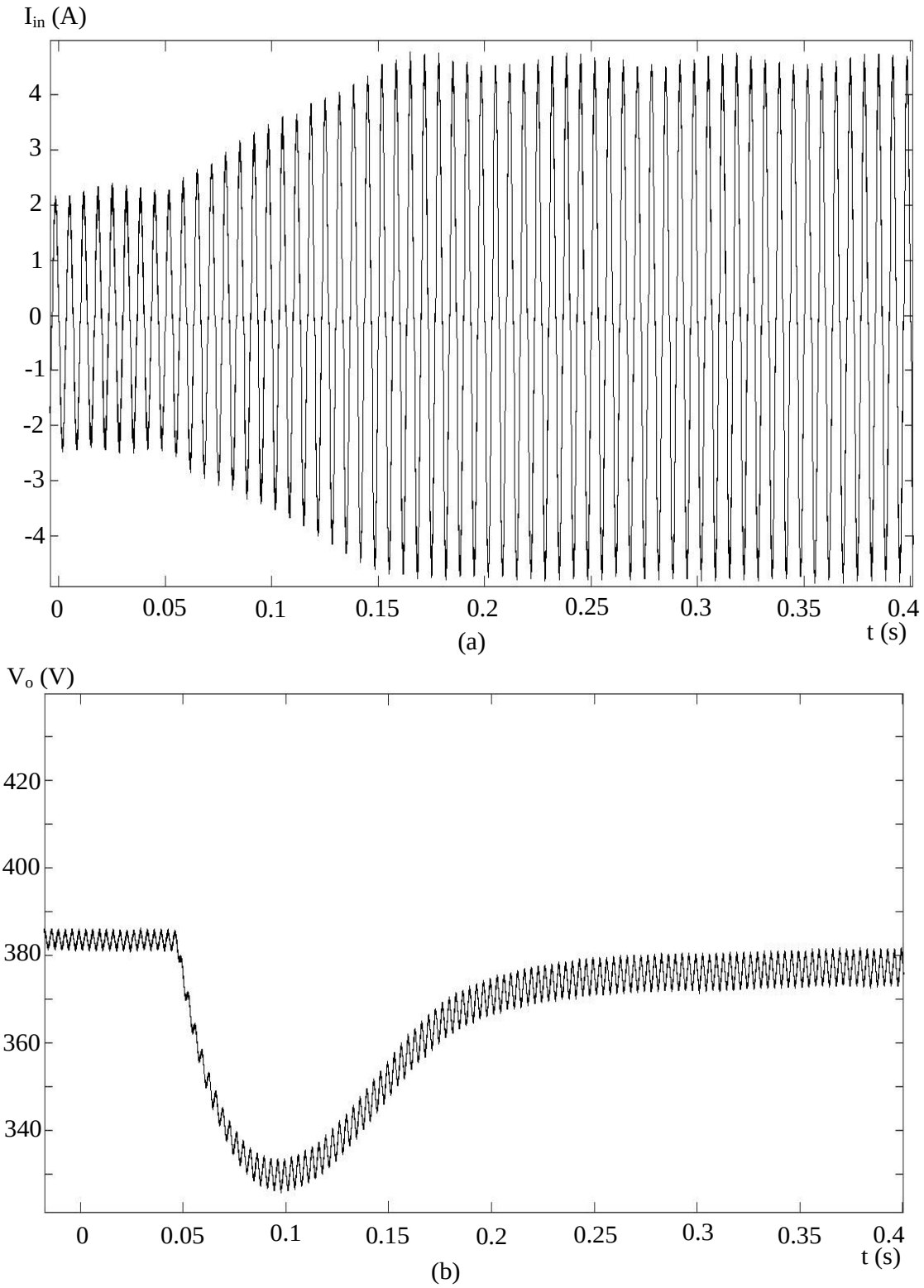


Figure 6.15: 100 W to 200 W step load change at 80 V RMS input and computed feedback, (a) line current, (b) output voltage

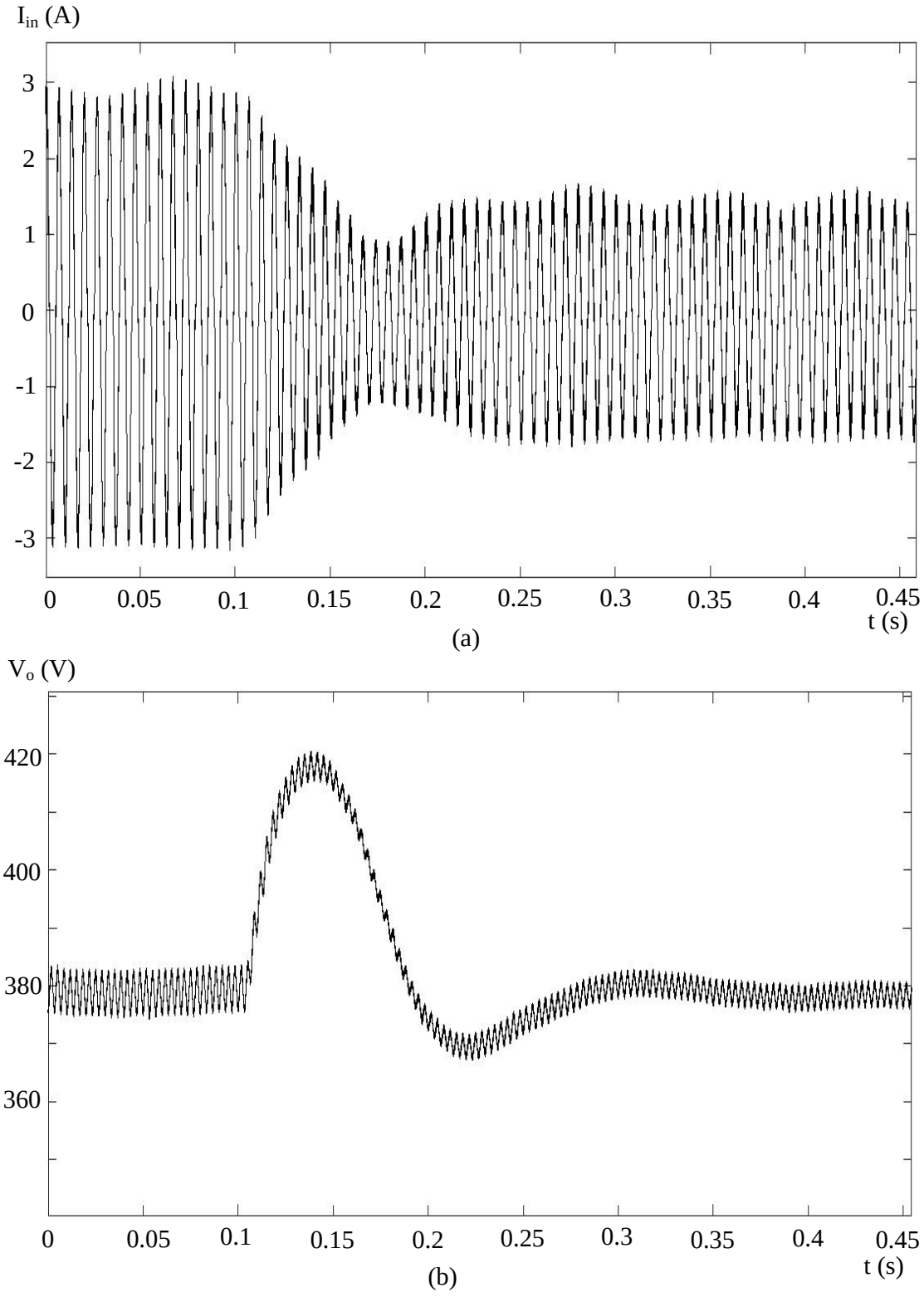


Figure 6.16: 200 W to 100 W step load change at 120 V RMS input and computed feedback, (a) line current, (b) output voltage

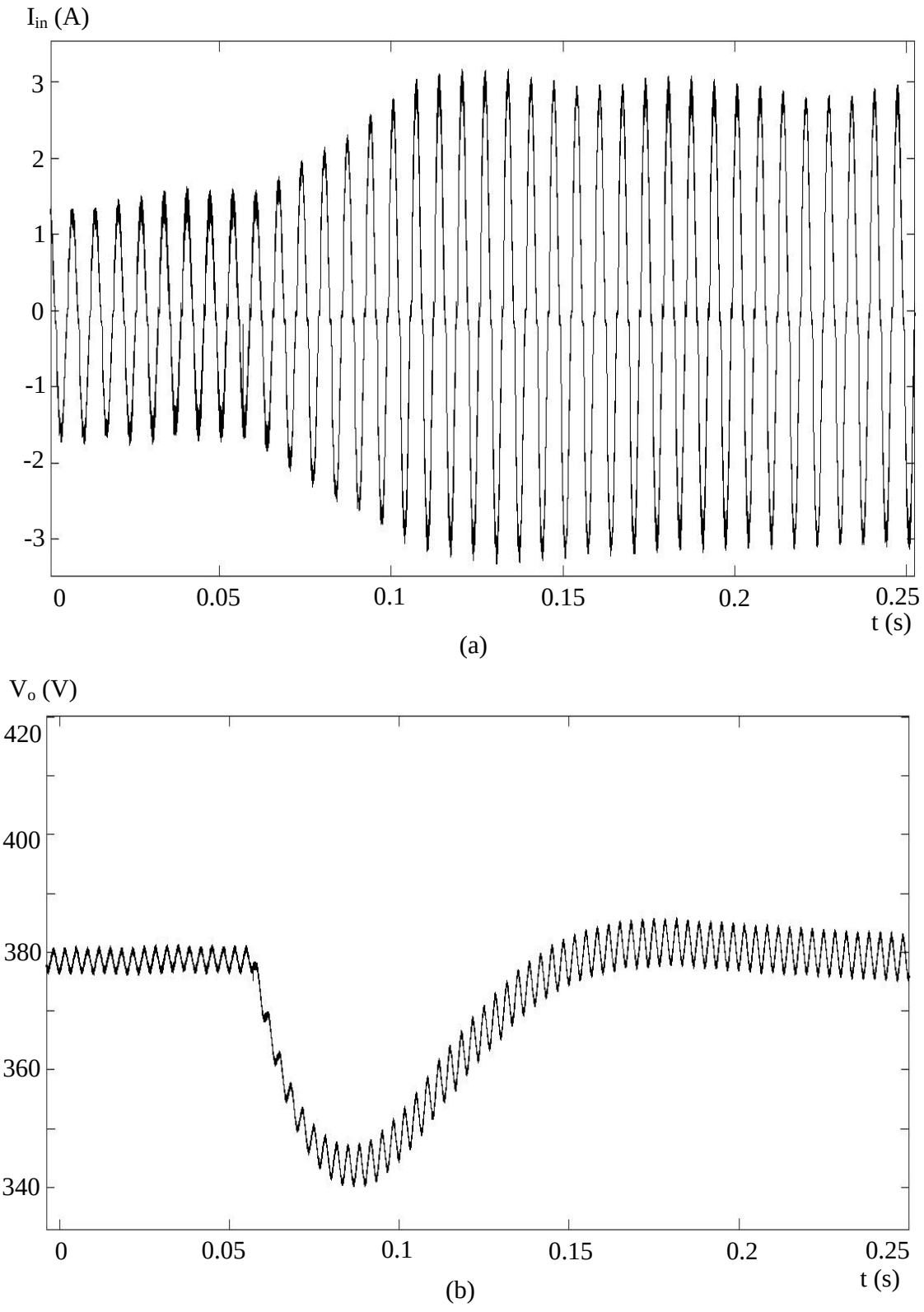


Figure 6.17: 100 W to 200 W step load change at 120 V RMS input and computed feedback, (a) line current, (b) output voltage

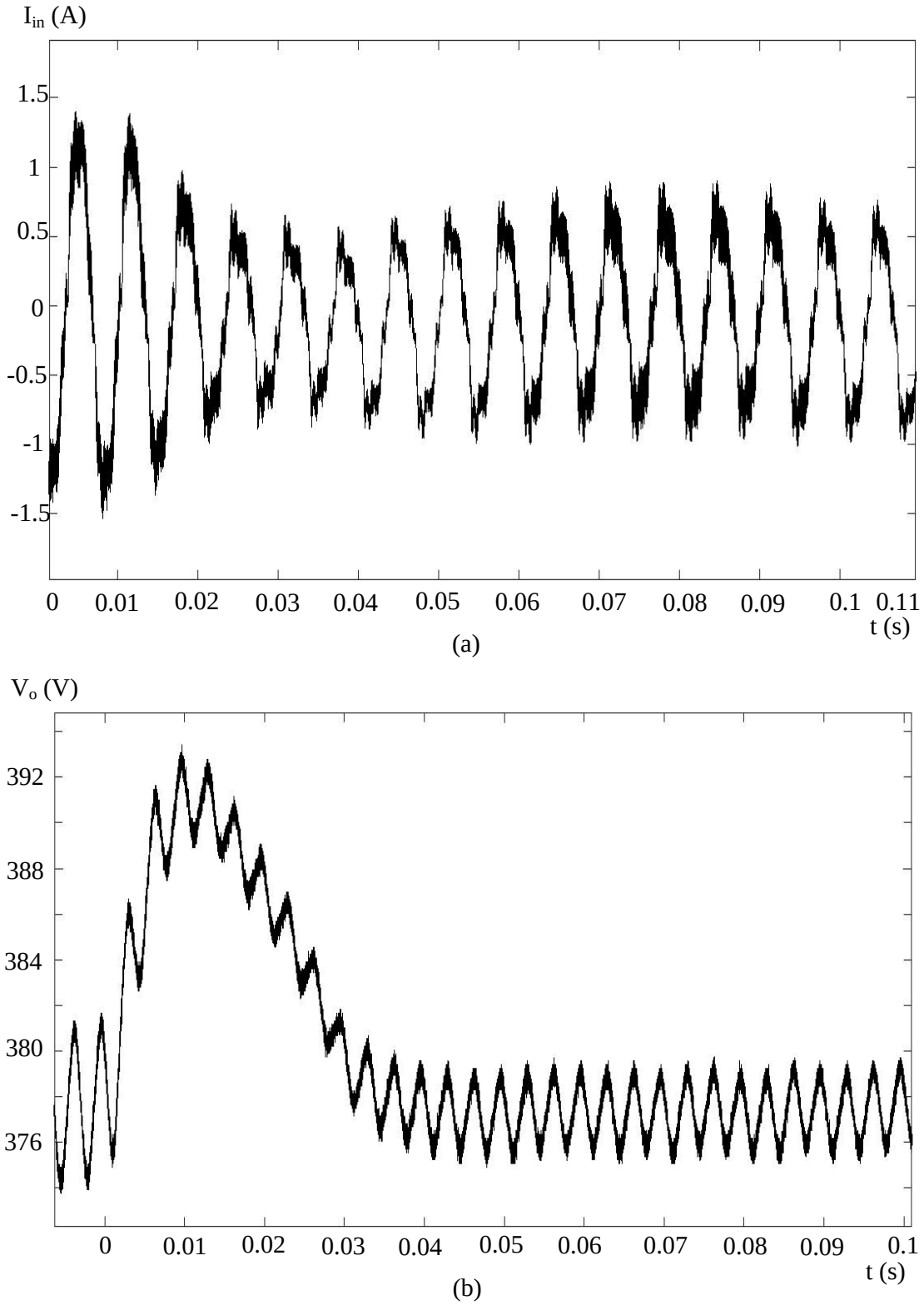


Figure 6.18: 200 W to 100 W step load change at 260 V RMS input and computed feedback, (a) line current, (b) output voltage

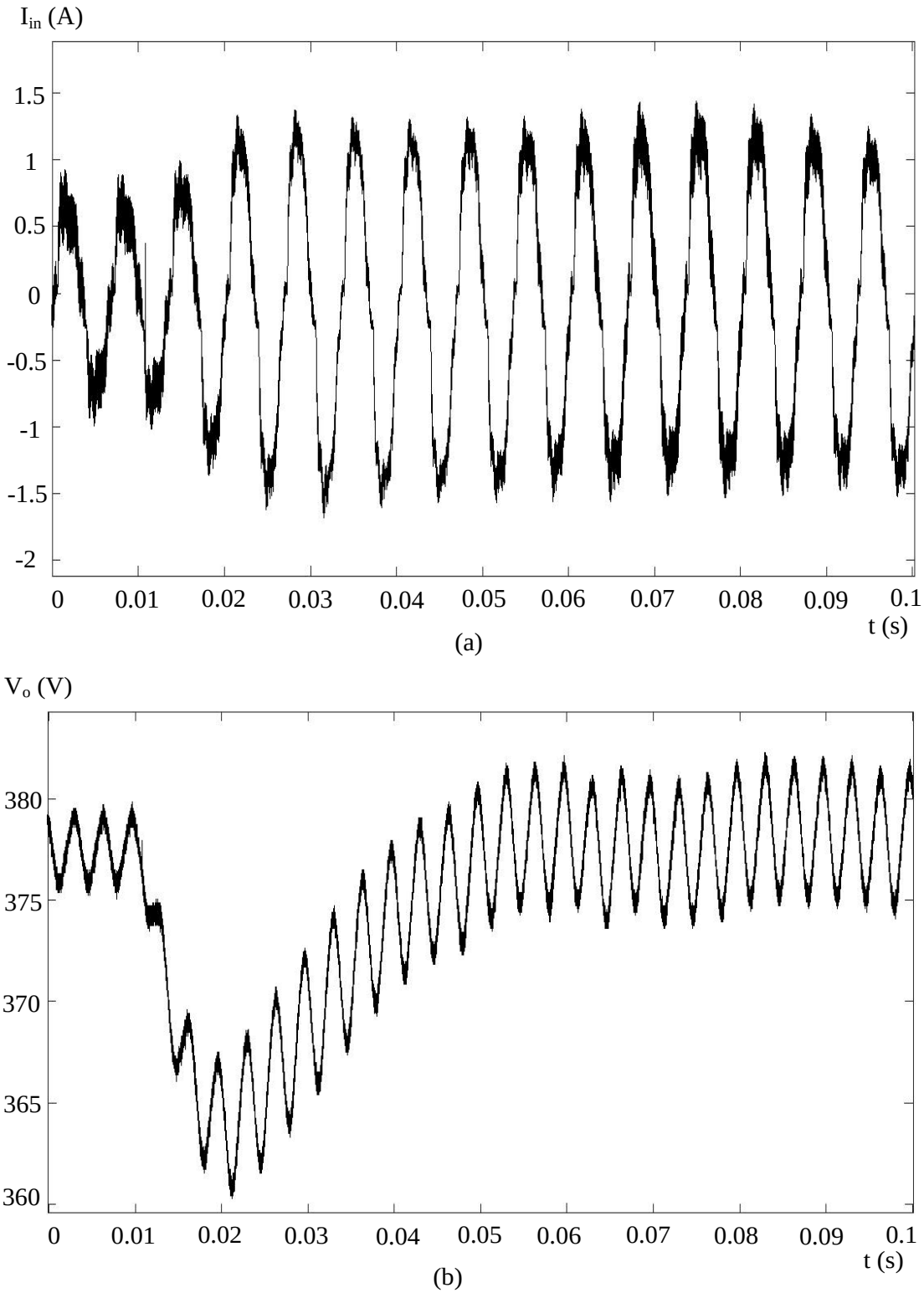


Figure 6.19: 100 W to 200 W step load change at 260 V RMS input and computed feedback, (a) line current, (b) output voltage

The presented plots show that the controller effectively maintains stability and returns the converter to steady-state operation in new conditions after transient. Minor deviations from the desired 380 V output at steady state are assumed to be caused by thermal changes of the voltage divider resistances.

The line current waveform deteriorates at higher input voltages and lower current. This fact is attributed to the limitations of the digital signal processor, sensing accuracy and electromagnetic noise susceptibility of the circuit.

The transient at lower input voltage takes more time and the overshoot has a larger amplitude than at higher voltages. It is explained by the fact that the controller's coefficients were recalculated for high input voltage operating point (see Chapter 2). This influence can be alleviated by using individual sets of parameters according to the consumer's standard voltage level. An experiment with a flexible voltage loop gain multiplier was performed in order to prove the concept. The set up is illustrated in Fig. 6.20. The gain multiplier K was assumed to be proportional to the inverse of the square of the input voltage [2]:

$$K \propto \frac{1}{V_{in}^2} \tag{6.1}$$

The multiplier was chosen to be unity at rated input voltage of 120V RMS (Fig. 6.21) such that the voltage loop gain was equal to 0.0985 at this voltage [1]:

$$G_c^v = K \cdot 0.0985 = \frac{14400}{V_{in}^2} \cdot 0.0985 \tag{6.2}$$

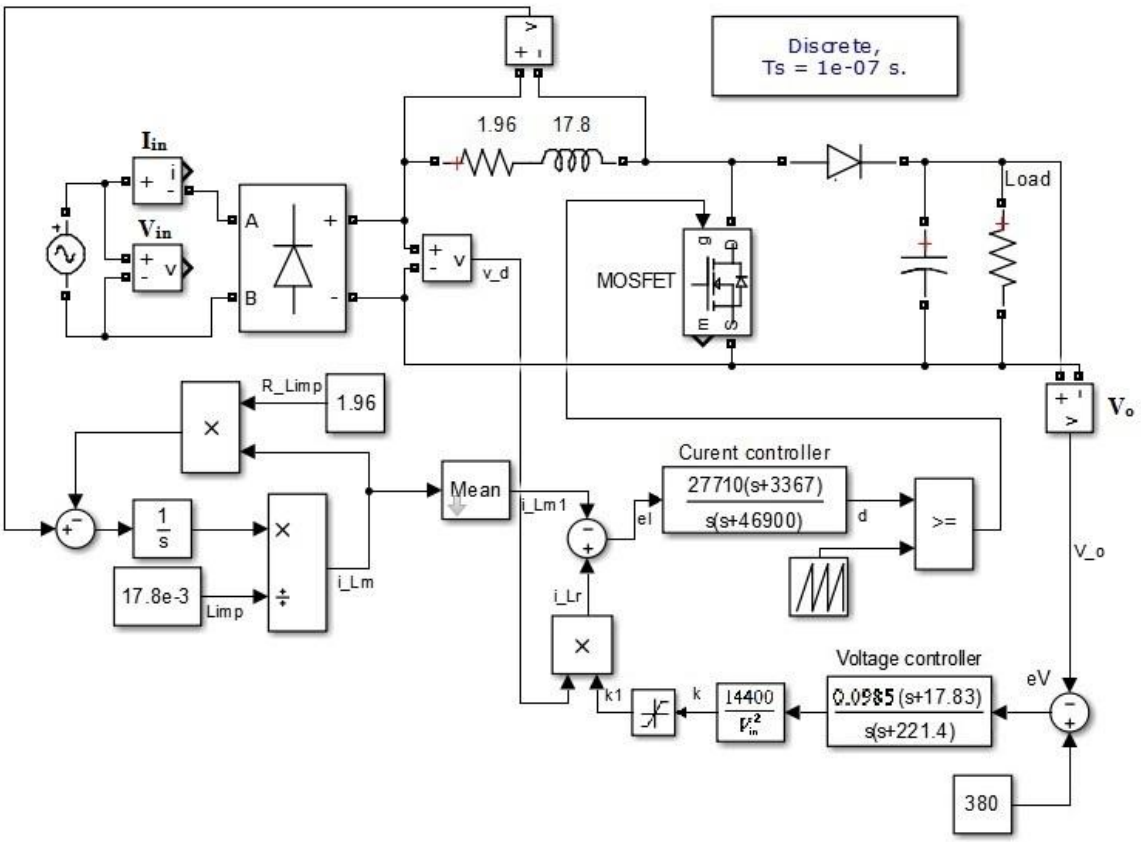


Figure 6.20: Converter diagram with gain multiplier

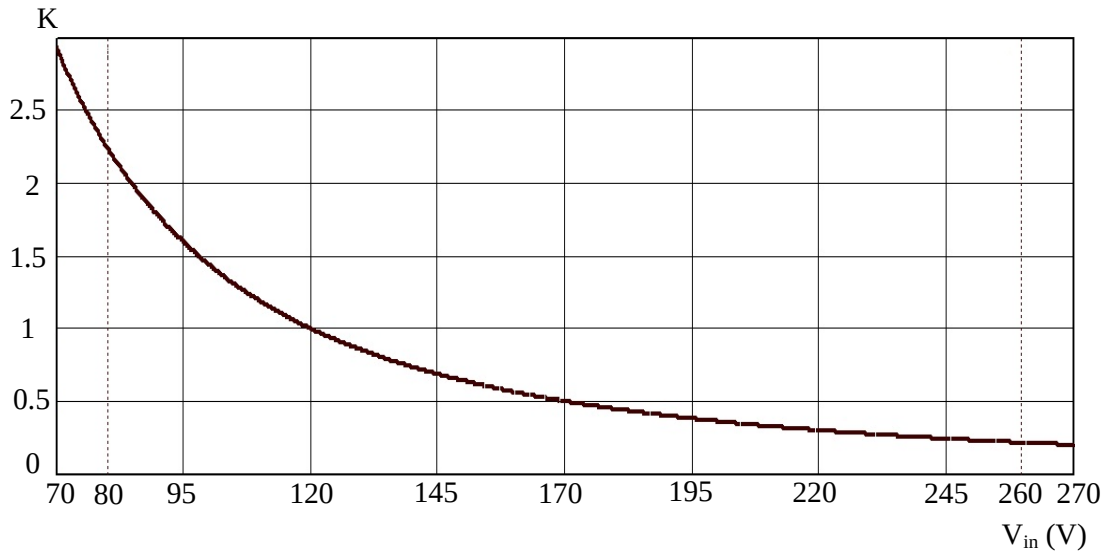


Figure 6.21: Gain multiplier K

The following comparison of transient responses (Fig. 6.22, 6.23) gives an example of the effect of an adjustable voltage loop gain:

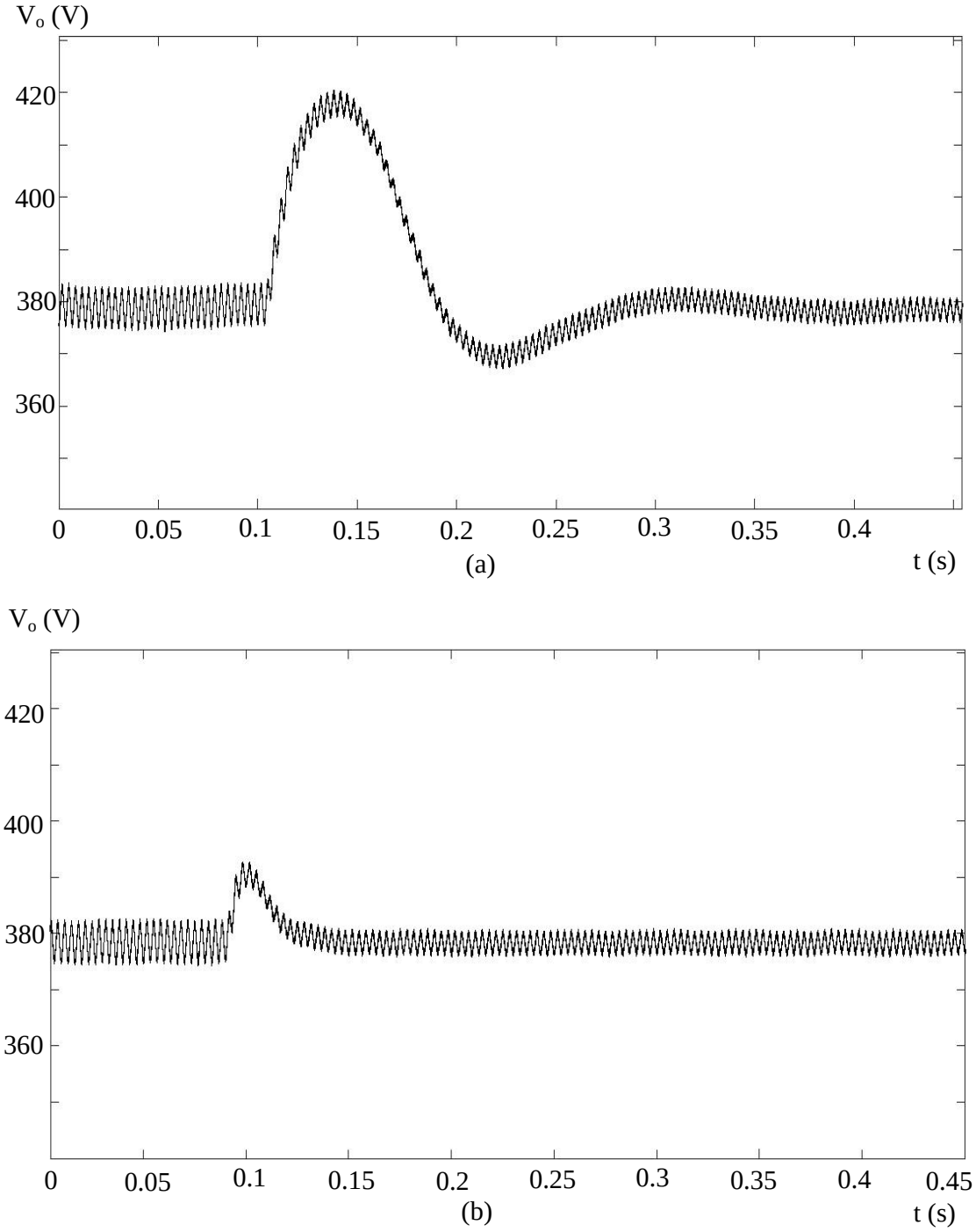


Figure 6.22: Output voltage at 200 W to 100 W step load change, 120 V RMS input and computed feedback, (a) before gain multiplication, (b) after gain multiplication

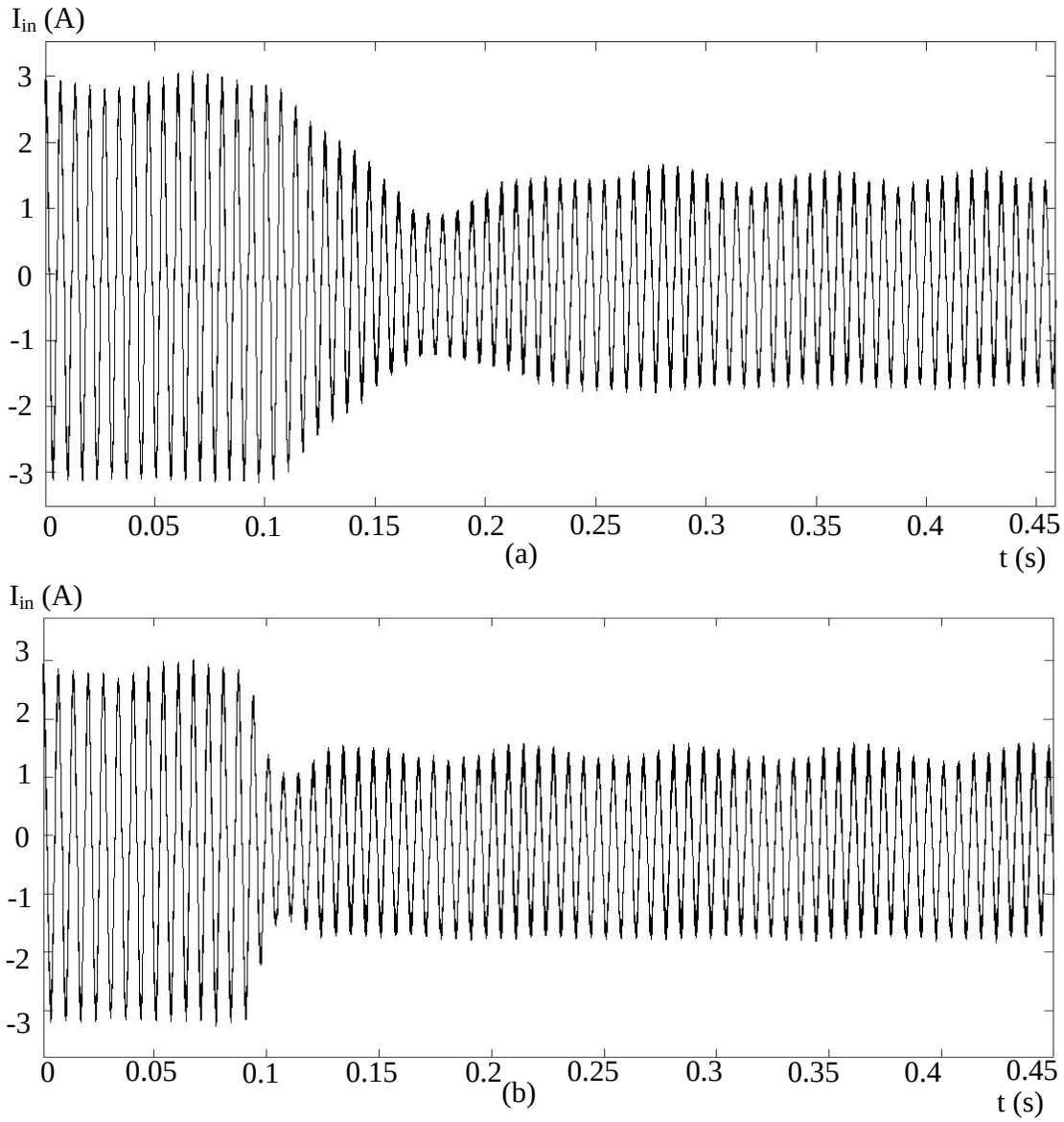


Figure 6.23:Line current at 200 W to 100 W step load change, 120 V RMS input and computed feedback, (a) before gain multiplication, (b) after gain multiplication

The converter is fully programmable and, therefore, allows selection of controller parameters according to the input voltage levels. Hence, one can have an individual region oriented set of parameters after the system identified a particular operation mode.

6.4 Harmonics analysis

An analysis of high frequency harmonics contamination allows to estimate the current shaping performance of the converter. Harmonics analysis was performed by using MATLAB's Fast Fourier Transform functions. Signal samples were acquired for 7.5 seconds at 500 kHz sampling frequency. Harmonic spectra relative to the fundamental are shown in Fig. 6.24, 6.25.

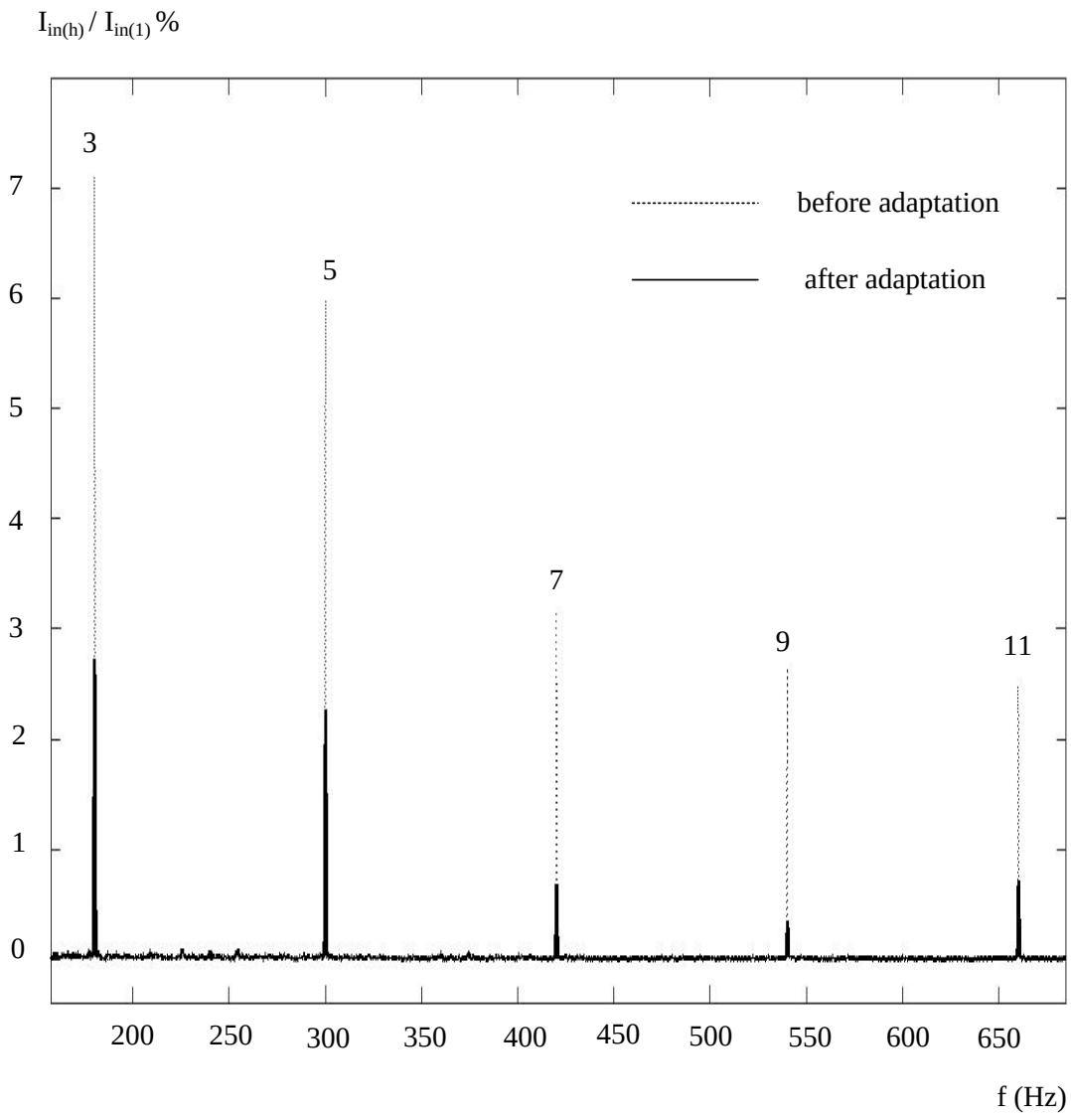


Figure 6.24: Low-order harmonics spectra before and after adaptation enabled at 120 V RMS input 200W load with computed feedback operation.

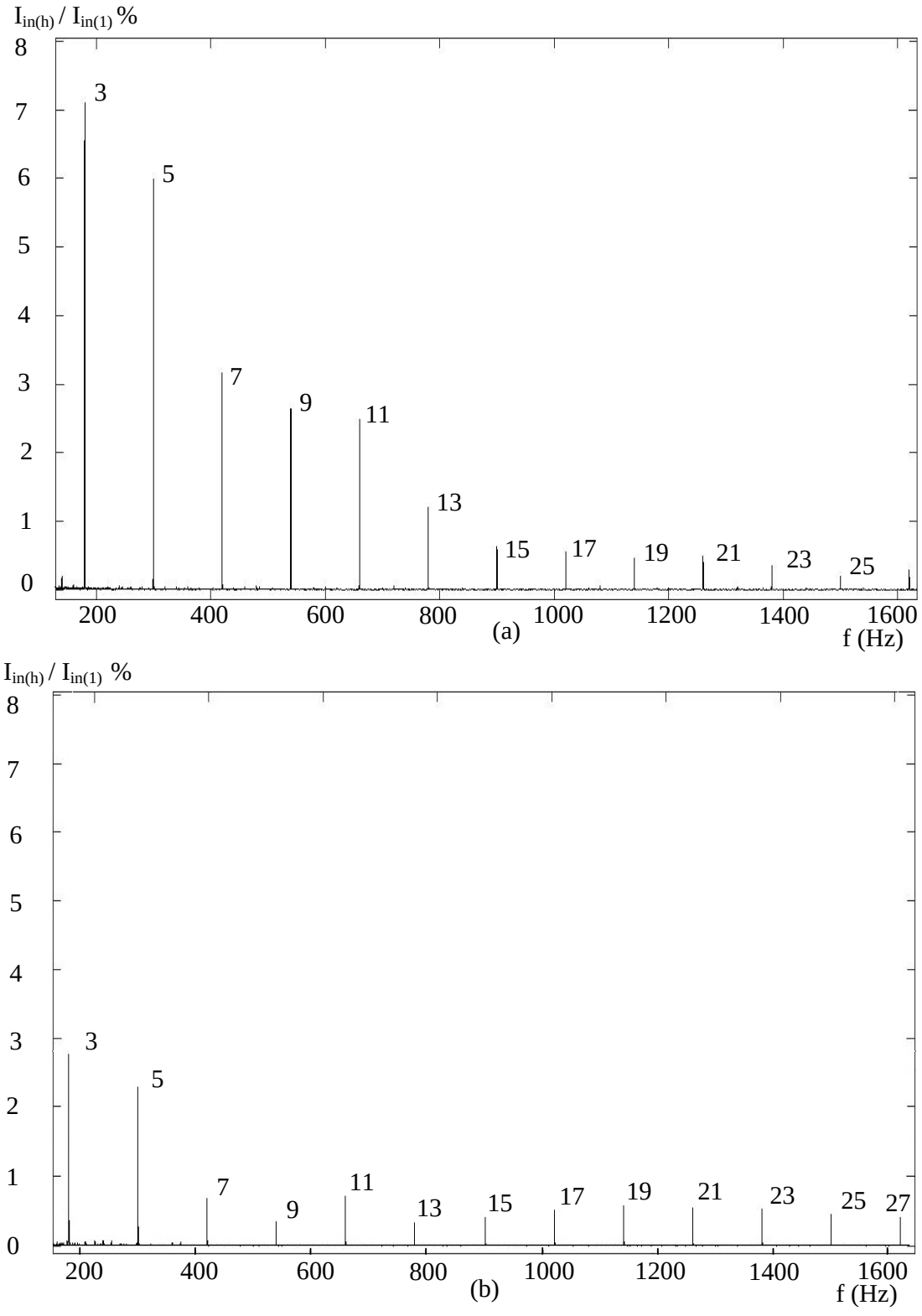


Figure 6.25: Higher harmonics spectra before and after adaptation enabled at 120 V RMS input 200W load with computed feedback operation, (a) before adaptation, (b) after adaptation.

Table 6.1: Harmonics of the input current relative to the fundamental and its total harmonic distortion before and after model adaptation at rated load

Harmonic h	**Frequency (Hz)**	**$I_{in(h)} / I_{in(1)}$ %**	
		Computed feedback without adaptation enabled THD % = 10.605	Computed feedback with adaptation enabled THD % = 3.92
3	180	7.113862	2.731205
5	300	5.985465	2.25937
7	420	3.168112	0.690528
9	540	2.652304	0.344816
11	660	2.484927	0.713005
13	780	1.210026	0.334062
15	900	0.632421	0.430757
17	1020	0.553571	0.516023
19	1140	0.465072	0.569457
21	1260	0.491073	0.554183
23	1380	0.355101	0.53805
25	1500	0.20158	0.470548

Above figures demonstrate enhancement of overall performance provided by the inductor parameters adaptation algorithm. The calculation of Total Harmonic Distortion (THD) for taken samples returned the following values: computed feedback operation 10.605%, computed feedback operation with parameter adaptation enabled 3.92%. Equation (1.3) in terms of THD would be as follows:

$$PF = \cos\varphi_{(1)} \frac{1}{\sqrt{1+THD^2}} \quad (6.3)$$

Respectively, computed feedback operation results in PF = 0.994, whereas adaptation improves it up to 0.999.

Further calculations were conducted to prove the efficiency of model adaptation at various operation points (Tables 6.2,6.3).

Table 6.2: Low-order harmonics of the input current relative to the fundamental and its total harmonic distortion before and after model adaptation as well as with measured feedback at rated load

V_{in} RMS (V)	**Mode**	**$I_{in(h)}$ / $I_{in(1)}$ %**					**THD %**
		h = 3	h = 5	h = 7	h =9	h = 11	
80	Measured	4.81	4.04	3.42	2.4	0.87	7.62
	Computed	7.16	5.31	3.39	2.08	0.64	9.81
	Adaptation enabled	6.5	3.41	1.76	1.14	0.94	8.17
120	Measured	3.16	3.66	1.8	1.94	2.31	4.81
	Computed	7.11	5.99	3.16	2.65	2.49	10.6
	Adaptation enabled	2.73	2.26	0.69	0.34	0.71	3.92
260	Measured	2.12	6.84	3.69	1.85	4.86	9.29
	Computed	3.76	7.19	2.77	2.2	5.15	10.875
	Adaptation enabled	6.95	4.27	2.28	0.97	3.78	9.6

Table 6.3: Low-order harmonics of the input current relative to the fundamental and its total harmonic distortion after model adaptation at various loads

V_{in} RMS (V)	Load (W)	$I_{in(h)} / I_{in(1)}$ %					THD %
		3	5	7	9	11	
80	200	6.50	3.41	1.76	1.14	0.94	8.17
	100	3.41	2.0	0.82	0.22	0.58	4.44
	50	5.97	2.99	3.36	1.47	1.49	7.87
120	200	2.73	2.26	0.69	0.34	0.71	3.92
	100	3.72	2.52	1.97	0.75	1.66	5.41
	50	4.38	2.78	4.43	1.43	3.30	7.99
260	200	6.95	4.27	2.28	0.97	3.78	9.6
	100	12.44	11.34	6.62	4.08	7.08	20.33

7 Conclusions

The project was devoted to universalization of the boost PFC converter developed in [1] by means of broadening of the input voltage. After thorough analysis of [1] and other sources, a number of measures were taken to allow the converter to operate in the desired input range. Along with that, improvements of safety, stability and precision were introduced.

Various experiments were performed to demonstrate the ability of the converter to operate under desired conditions. Further processing of the captured data helped to estimate the performance of active current waveform shaping.

Bibliography

[1] A. Engel, “Current sensorless control of a boost-type switch-mode rectifier using an adaptive inductor model", Master of Applied Science thesis, University of Victoria, Victoria, British Columbia, 2013.

[2] Fairchild Semiconductor Inc., “ML4821 Power Factor Controller datasheet”, 2001, www.fairchildsemi.com

[3] H. D. Venable, "The K factor: A new mathematical tool for stability analysis and synthesis", without year, Linear Technology Reference Reading #4.

[4] Spectrum Digital Inc., "eZdspF2812 Technical Reference", Rev. F, 2003. http://c2000.spectrumdigital.com/ezf2812/docs/ezf2812_techref.pdf.

[5] Spectrum Digital Inc., "eZdspF2812 Schematic", Rev. C, 2003. http://c2000.spectrumdigital.com/ezf2812/docs/ezf2812_schem.pdf.

[6] Texas Instruments Incorporated, “TMS320x281x DSP Analog-to-Digital Converter (ADC) Reference Guide”, 2002. http://www.ti.com.cn/cn/lit/ug/spru060d/spru060d.pdf

[7] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power electronics: converters, applications, and design*, 2nd ed., United States of America: John Wiley & Sons, 1995.

[8] Texas Instruments Incorporated, "Code Composer Studio (CCStudio) Integrated Development Environment (IDE) v5", 2012. http://www.ti.com/tool/ccstudio.

Appendix A

Digital controller source code

```
/* main.c */

/*front matter*/ #if (1)
#include "DSP281x_Device.h"     // define CPU commands, variable types, include all
peripheral header files
#include "DSP281x_Examples.h"   // define CPU clock, include various header files
#include "IQmathLib.h"          // IQmath library

interrupt void adc_isr();
void shutdown();

volatile int samplecount=0;     // counter for the number of samples taken in the
current switching cycle
int NSPSC;                      // number of samples per switching cycle
volatile long v_dsum=0;         // sum of v_d for current switching cycle
volatile long v_Qsum=0;         // sum of v_Q for current switching cycle
volatile long v_osum=0;         // sum of v_o for current switching cycle
volatile long i_Lsum=0;         // sum of i_L for current switching cycle
volatile int processed=1;       // completely sampled switching cycle has been
processed (boolean)
#endif

int main() {

unsigned int      i;            // for use in any small loop or other local temporary
use

/* ============= system control ============================== */ #if (1)

EALLOW;
        SysCtrlRegs.WDCR=0x0068;                // disable watchdog timer
EDIS;
        InitPll(0xA);                           // set clock PLL to 10:
SYSCLKOUT = 5*XCLKIN = 150 MHz
EALLOW;
        SysCtrlRegs.HISPCP.all=0x0000;          // set high-speed peripheral clock
pre-scaler, factor 1
        SysCtrlRegs.LOSPCP.all=0x0002;          // set low-speed peripheral clock pre-
scaler
        SysCtrlRegs.PCLKCR.bit.EVAENCLK=1;      // enable peripheral clock for event
manager A
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=1;      // enable peripheral clock for ADC
EDIS;
#endif
```

```
SysCtrlRegs.PCLKCR.bit.EVAENCLK=1;      // enable peripheral clock for event manager
A
SysCtrlRegs.PCLKCR.bit.ADCENCLK=1;      // enable peripheral clock for ADC
EDIS;
#endif

/* ============= GPIO pins ================================== */ #if (1)

// configure all GPIO pins as digital outputs and set to 0
EALLOW;
        GpioMuxRegs.GPAMUX.all=0;
        GpioMuxRegs.GPBMUX.all=0;
        GpioMuxRegs.GPDMUX.all=0;
        GpioMuxRegs.GPEMUX.all=0;
        GpioMuxRegs.GPFMUX.all=0;
        GpioMuxRegs.GPGMUX.all=0;
        GpioMuxRegs.GPADIR.all=0xFFFF;
        GpioMuxRegs.GPBDIR.all=0xFFFF;
        GpioMuxRegs.GPDDIR.all=0xFFFF;
        GpioMuxRegs.GPEDIR.all=0xFFFF;
        GpioMuxRegs.GPFDIR.all=0xFFFF;
        GpioMuxRegs.GPGDIR.all=0xFFFF;
        GpioDataRegs.GPACLEAR.all=0xFFFF;
        GpioDataRegs.GPBCLEAR.all=0xFFFF;
        GpioDataRegs.GPDCLEAR.all=0xFFFE;
        GpioDataRegs.GPECLEAR.all=0xFFFF;
        GpioDataRegs.GPFCLEAR.all=0xFFFF;
        GpioDataRegs.GPGCLEAR.all=0xFFFF;


        // configure required GPIO pins as peripheral (timer) outputs

        GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=1;
        GpioMuxRegs.GPAMUX.bit.T2PWM_GPIOA7=1;
EDIS;
```

```
#endif

/* ============ timers ==================================== */ #if
(1)

// GP timer 1 setup -- provides time base for sampling
        EvaRegs.T1CON.bit.FREE=1;               // complete timer period on
emulation suspend
        EvaRegs.T1CON.bit.SOFT=0;               // ... second part of it
        EvaRegs.T1CON.bit.TMODE=2;              // continuous up-counting mode
        EvaRegs.T1CON.bit.TPS=0;          // input clock prescaler, factor 1
        EvaRegs.T1CON.bit.TCLKS10=0;         // use HSPCLK as clock
        EvaRegs.T1CON.bit.TCLD10=0;             // reload compare register
T1CMPR at beginning of sampling period
        EvaRegs.T1CON.bit.TECMPR=1;             // enable timer compare
        EvaRegs.T1PR=149;                 // period register = 150MHz/f_s-1
        EvaRegs.T1CNT=0;                  // initialize counter register

// GP timer 2 setup -- provides time base for switching
        EvaRegs.T2CON.bit.FREE=1;               // complete timer period on
emulation suspend
        EvaRegs.T2CON.bit.SOFT=0;               // ... second part of it
        EvaRegs.T2CON.bit.TMODE=2;              // continuous up-counting mode
        EvaRegs.T2CON.bit.TPS=0;          // input clock prescaler, factor 1
        EvaRegs.T2CON.bit.T2SWT1=1;             // start together with T1
        EvaRegs.T2CON.bit.TCLKS10=0;         // use HSPCLK as clock
        EvaRegs.T2CON.bit.TCLD10=2;             // reload compare register
T2CMPR immediately
        EvaRegs.T2CON.bit.TECMPR=1;             // enable timer compare
        EvaRegs.T2CON.bit.SET1PR=0;             // use own period register (rather
than T1's)
        EvaRegs.T2PR=7499;                // period register = 150MHz/f_sw-1
        EvaRegs.T2CMPR=3000;                    // compare register for GP T2
        EvaRegs.T2CNT=0;                  // initialize counter register
```

```
// GP timer control register of EVA (for TxPWM)

        EvaRegs.GPTCONA.bit.T2CTRIPE=0;      // disable trip function that can
drive output to high Z

        EvaRegs.GPTCONA.bit.T1CTRIPE=0;
        EvaRegs.GPTCONA.bit.TCMPOE=1;        // enable compare output
        EvaRegs.GPTCONA.bit.T2TOADC=0;       // do not start ADC
        EvaRegs.GPTCONA.bit.T1TOADC=2;       // start ADC on period match
        EvaRegs.GPTCONA.bit.TCMPOE=1;        // enable timer compare output
        EvaRegs.GPTCONA.bit.T2CMPOE=1;       // enable timer 2 compare output
        EvaRegs.GPTCONA.bit.T1CMPOE=1;       // enable timer 1 compare output
        EvaRegs.GPTCONA.bit.T2PIN=1;         // polarity of T2PWM
        EvaRegs.GPTCONA.bit.T1PIN=1;         // polarity of T1PWM

//      timers are enabled before the main loop is entered
#endif

/* ============= ADC ====================================== */
#if (1)

        AdcRegs.ADCTRL1.bit.RESET=1;         // reset ADC
        asm(" RPT #50 || NOP");              // required delay after reset before
modifying ADC registers

        AdcRegs.ADCTRL3.bit.ADCBGRFDN=0x3;       // power up
bandgap/reference circuitry

        for (i=0;i<2500;i++) asm(" RPT #255 || NOP"); // required 5 ms delay (5.1 ms)

        AdcRegs.ADCTRL3.bit.ADCPWDN=1;           // power up rest of ADC
        for (i=0;i<20;i++) asm(" RPT #255 || NOP");  // required 20 us delay (38 us)


        AdcRegs.ADCTRL1.bit.SUSMOD=1;            // upon emulation suspend
finish ADC sequence

        AdcRegs.ADCTRL1.bit.ACQ_PS=1;            // length of the S/H pulse
in ADC clock cycles 1

        AdcRegs.ADCTRL1.bit.CPS=0;           // ADC clock prescaler, factor 1
        AdcRegs.ADCTRL1.bit.CONT_RUN=0;          // start-stop mode (as
opposed to continuous conversion)
```

```
        AdcRegs.ADCTRL1.bit.SEQ_OVRD=0;      // disable sequencer override, NR
        AdcRegs.ADCTRL1.bit.SEQ_CASC=0;      // dual sequencer mode (as
opposed to cascaded mode)

        AdcRegs.ADCTRL2.bit.RST_SEQ1=1;      // reset sequencer 1
        AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1=1; //enable interrupt request by
SEQ1

        AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1=0; // interrupt after every EOS (as
opposed to every second one)

        AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1=1; // allow event manager A to
start conversion sequence

        AdcRegs.ADCTRL3.bit.ADCEXTREF=0;   // use internal reference sources
        AdcRegs.ADCTRL3.bit.ADCCLKPS=3;      // ADC clock divider, factor 1/6
        AdcRegs.ADCTRL3.bit.SMODE_SEL=1;   // higher sampling freq
        AdcRegs.ADCMAXCONV.bit.MAX_CONV1=1;   // higher sampling freq
        AdcRegs.ADCCHSELSEQ1.bit.CONV00=0; // sample ADCINA0 (v_d)(v_o)
        AdcRegs.ADCCHSELSEQ1.bit.CONV01=2; // sample ADCINA1 (v_Q) (i_L)
        AdcRegs.ADCST.bit.INT_SEQ1_CLR=1;    // clear SEQ1 interrupt flag bit

#endif

/* ============= interrupts ================================== */ #if
(1)

// configure interrupt control and peripheral interrupt expansion table

        InitPieCtrl();                       // clear all PIEIER and PIEIFR registers
        IER = 0x0000;                        // disable all interrupts on CPU level
        IFR = 0x0000;                        // clear CPU IR flag register
        InitPieVectTable();     // initialize and enable PIE vector table EALLOW;
        PieVectTable.ADCINT=&adc_isr;    // define adc_isr as interrupt service
routine

        //PieVectTable.ADCINT = &timer1_p_isr;  //isr for EVA timer 1 period
register

EDIS;
        IER |= M_INT1;                       // enable CPU IR group 1

        EINT;                                // enable interrupts globally
```

```
        PieCtrlRegs.PIECRTL.bit.ENPIE=1;          // enable PIE vector table
        PieCtrlRegs.PIEACK.all=0xFFFF;            // clear PIEACK
        PieCtrlRegs.PIEIER1.bit.INTx6=1;          // enable ADCINT

        ERTM; // clear DBGM (debug enable mask bit): service halt requests and
                // breakpoints, allow emulator to access registers in real time
#endif

/* ===== variable declarations and constant definitions ===== */ #if (1)

        // global Q = 22 defined in IQmathLib.h

      #define       f_sw    20000               // switching frequency;
        static _iq  f_sw_10       =0;           // f_sw>>10 in IQ for controller
parameters

        #define     V_oref          _IQ(380.0)          // reference output voltage
        #define     C       _IQ(270.0e-6*32)    // C <<5
        #define     L_0     _IQ(17.8e-3)        // initially implemented L
        #define     R_L_0 _IQ(2)                // initially implemented R_L

        // controller parameters

        #define     Ki      _IQ(29000.0/1024.0) // current controller gain >>10
        #define     wzi     _IQ(5360.0/1024.0)  // current controller zero >>10
        #define     wpi     _IQ(74600.0/1024.0) // current controller pole >>10
        #define     Kv      _IQ(0.02139*32)     // voltage controller gain <<5
        #define     wzv     _IQ(22.4)           // voltage controller zero
        #define     wpv     _IQ(176.0)          // voltage controller pole


        #define     dmin    _IQ(0.0)            // lower bound for d  0
        #define     dmax    _IQ(1.0)            // upper bound for d 1.0
        #define     kappamin _IQ30(0.0001)      // lower bound for kappa
        #define     kappamax _IQ30(0.022)       // upper bound for kappa
```

```
        #define     eVmin _IQ(-30.0)            // lower bound for eV  -30
        #define     eVmax _IQ(30.0)             // upper bound for eV  30

        static _iq  a1i=0, a2i=0, b0i=0, b1i=0, b2i=0, b0i2=0, b1i2=0, b2i2=0;//
current controller coefficients

        static _iq30 a1v=0, a2v=0, b0v=0, b1v=0, b2v=0;    // voltage controller
coefficients

        static _iq  L_imp =L_0;  // value of L implemented in the inductor model
        static _iq  R_L_imp     =R_L_0;     // value of R_L implemented in the
inductor model

        static _iq  L_est   =L_0;           // estimated value of L
        static _iq  R_L_est     =R_L_0;     // estimated value of R_L
        static _iq  L_corr[2]; // correction of L with respect to L_0 (with previous)
        static _iq  R_L_corr[2];  // correction of R_L with respect to R_L_0 (with
previous)

        static _iq  deltaL[2];      // difference between previous and current
estimated L (with previous)

        static _iq  deltaR_L[2];  // difference between previous and current
estimated R_L (with previous)

        static int  adapt   =0;     // enables/disables adaptation (boolean)

        static int  NSCPLHC;    // number of switching cycles per line half-cycle
        static int  NSCPLHC2;   // NSCPLHC/2
        static long T2per;          // period of timer 2 (register value)
        static int  LHCcount=0; // counter for the number of switching cycles
completed in the current line half-cycle

        static _iq  v_L[2]; //average of v_L over one switching cycle (with previous)
        static _iq  v_o=0;              // average of v_o over one switching cycle
        static _iq  i_LM[2];            // measured i_L (with previous)
        static _iq  i_LC[2];            // computed i_L (with previous)

                // most recent value has index 0: i_L[0]=i_L[k], i_L[1]=i_L[k-1]

        static _iq  V_dp   =_IQ(170.0);  // peak input voltage
        #define     LHCarlen 180        // length of the arrays that store data over
one line half-cycle

        static _iq  v_dLHC[LHCarlen];  // all values of v_d for one line half-cycle
```

```
        static _iq      v_dsum10=0; // sum of v_d from 10 switching cycles (for ZCD)
        static _iq      v_osumLQC1=0; // sum of (v_o-V_oref) for first line quarter-
cycle

        static _iq      v_osumLQC2=0; // sum of (v_o-V_oref) for second line
quarter-cycle

        static _iq      V_o2   =0;      // peak-to-peak value of the output voltage
ripple = peak of 2nd harmonic

        static _iq      i_Lmax         =0;

        static int      zcdavstart; // value of LHCcount at which averaging of v_d
starts

        static _iq      zcdthresh=_IQ(20.0);  // threshold value for line voltage zero
crossing detection

        static _iq      sine[LHCarlen]; // look-up table with one half-cycle of sine
values

        static _iq      v_Lsum         =0;     // for integrating v_L
        static _iq      v_LsumLQC=0;       // to store v_Lsum after one line quarter-
cycle is completed

        static _iq      fi_L1=0, fi_L2=0, fi_L3=0; // factors used in the computation
of i_L

        static _iq      eV[3];                  // output voltage error (with 2 previous)
        static _iq      eI[3];                  // inductor current error (with 2 previous)
        static _iq30 kappa[3];                  // emulated conductance; i_L =
kappa*v_d (with 2 previous)

        static _iq      d[4];                   // duty cycle (with 2 previous)

        #define         tauL    _IQ(0.04)       // time constant for adjustment of L
        static _iq      fL1=0, fL2=0;           // factors used in adjustment of L
        #define         tauR_L  _IQ(0.04)       // time constant for adjustment of R_L
        static _iq      fR_L1=0, fR_L2=0;   // factors used in adjustment of R_L

        static _iq      facL;                           // factor for computing L
        static _iq      facR_L;                         // factor for computing R_L
        #define         oneover120pi _IQ(2.65258e-3)        // = 1/(120*pi)
        static _iq      Vdp_Vo2        =_IQ(1.0);               // V_dp/V_o2 >>8

        static long v_dbuf      =0;                     // switching cycle sum buffers
```

```
        static long v_obuf      =0;
        static long v_Qbuf      =0;
        static long i_Lbuf      =0;

#endif

/* ============= computation of parameters ================== */ #if (1)
        f_sw_10=_IQ(((float)f_sw)/1024.0);           // f_sw>>10 in IQ
   NSPSC=50;                                 // number of samples per switching cycle
        NSCPLHC=f_sw/120.0+.5;   // number of switching cycles per line half-cycle
        NSCPLHC2=NSCPLHC>>1;// NSCPLHC/2
        zcdavstart=NSCPLHC2+10;  // value of LHCcount at which averaging of v_d
starts

        a1i=_IQdiv(f_sw_10<<2,((f_sw_10<<1)+wpi));      // current controller
coefficients

        a2i=_IQ(1.0)-a1i;
        b0i=_IQmpy( _IQdiv(Ki,f_sw_10)>>1, _IQdiv( ((f_sw_10<<1)+wzi),
((f_sw_10<<1)+wpi) ));
        b1i=_IQmpy( _IQdiv(Ki,f_sw_10<<1), _IQdiv( wzi<<1, ((f_sw_10<<1)+wpi)
));
        b2i=b1i-b0i;

        a1v=_IQ30div(f_sw_10<<2,((f_sw_10<<1)+(wpv>>10)));  // voltage
controller coefficients
        a2v=_IQ30(1.0)-a1v;
        b0v=_IQ30rmpy( _IQ30div(Kv,f_sw_10),
_IQ30div((f_sw_10<<1)+(wzv>>10),(f_sw_10<<1)+(wpv>>10)));

        b1v=_IQ30rmpy( _IQ30div(Kv,f_sw_10),
_IQ30div(wzv>>9,(f_sw_10<<1)+(wpv>>10)) );

        b2v=b1v-b0v;
                // all bv are <<16

        fL1=_IQdiv( _IQmpy(tauL,f_sw_10<<1)-(_IQ(1.0)>>10),
_IQmpy(tauL,f_sw_10<<1)+(_IQ(1.0)>>10));

        // factors used in adjustment of L, = (2*tauL*f_sw-1)/(2*tauL*f_sw-1)

        fL2=_IQdiv(_IQ(1.0),_IQmpy(tauL<<7,f_sw_10)+(_IQ(1.0)>>4));
```

```
        // = 1/(2*tauL*f_sw+1) <<4
        fR_L1=_IQdiv( _IQmpy(tauR_L,f_sw_10<<1)-(_IQ(1.0)>>10),
_IQmpy(tauR_L,f_sw_10<<1)+(_IQ(1.0)>>10));

   // factors used in adjustment of R_L, = (2*tauR_L*f_sw-1)/(2*tauR_L*f_sw-1)
        fR_L2=_IQdiv(_IQ(1.0),_IQmpy(tauR_L<<7,f_sw_10)+(_IQ(1.0)>>4));
                // = 1/(2*tauR_L*f_sw+1) <<4

        facR_L=_IQdiv(_IQ(1.0),_IQmpy(_IQmpy(f_sw_10,C),V_oref));  // factor for
computing R_L, = 1 / (f_sw*C*V_oref) <<8
                        // divisor: >>10 from f_sw_10, <<5 from C, >>3 from factor 8;
compensate with >>8 from Vdp_Vo2

        facL=_IQmpy( _IQdiv(_IQ(1.0),_IQmpy(_IQmpy(f_sw_10,C),V_oref)) ,
_IQ(0.021221));         // factor for computing L, =  1 / (f_sw*C*V_oref*15*pi) <<10
                        // divisor: >>10 from f_sw_10, <<5 from C, >>5 from factor
480/15; compensate with >>8 from Vdp_Vo2 and >>2 from v_LsumLQC

        T2per=EvaRegs.T2PR;
        #endif


/* ============= array initializations ====================== */ #if (1)

        i_LM[0]=i_LM[1]=0;
        i_LC[0]=i_LC[1]=0;

        for (i=0;i<4;i++){

                eV[i]=0;                        // previous voltage errors
                eI[i]=0;                        // previous current errors
                kappa[i]=_IQ(.015);             // previous voltage controller outputs
                d[i]=_IQ(.7);                   // previous current controller outputs

                }

        L_corr[0]=L_corr[1]=0;
        R_L_corr[0]=R_L_corr[1]=0;
```

```
        deltaL[0]=deltaL[1]=0;
        deltaR_L[0]=deltaR_L[1]=0;

        for (i=0;i<NSCPLHC;i++) // look-up table with one half-cycle of sine values
                sine[i]=_IQsinPU(_IQdiv(_IQ(i),_IQ(NSCPLHC*2)));

        for (i=NSCPLHC;i<LHCarlen;i++)
                sine[i]=0;
#endif

        EvaRegs.T1CON.bit.TENABLE=1;  // enable timers 1 and 2

// ============= infinite loop ==============================
        while(1){

if (samplecount==NSPSC && processed==0){        // when one entire switching cycle
has been captured and not yet been processed


        EvaRegs.T2CMPR=_IQmpyI32int(d[3],T2per);       // update compare register
with previously computed D


                v_dbuf=v_dsum;              // copy switching cycle sums to buffers
                v_obuf=v_osum;
                v_Qbuf=v_Qsum;
                i_Lbuf=i_Lsum;

                v_dsum=0;                   // reset switching cycle sums
                v_Qsum=0;
                v_osum=0;
                i_Lsum=0;

                samplecount=0;              // reset counter for the number of samples
taken in the current switching cycle
```

```
/*compute switching cycle averages*/ #if (1)

v_L[1]=v_L[0];  // copy values from previous switching cycle to higher indices

i_LM[1]=i_LM[0];

/* taking voltage divider ratio, left alignment of conversion register value,
conversion to voltage
   value from 12 bit integer conversion result and division by NSPSC into
account */

v_dLHC[LHCcount]=_IQmpyI32(_IQ(0.0017667),(v_dbuf+7100)>>4);

v_o=_IQmpyI32(_IQ(.0020054),(v_obuf+400)>>4);

v_L[0]=v_dLHC[LHCcount]-_IQmpyI32(_IQ(.002039),(v_Qbuf+750)>>4) ;

if (LHCcount<20 && v_L[0]<0)
v_L[0]=v_L[1]=0; // restrict v_L to positive voltages at beginning of LHC

i_LM[0]=_IQmpyI32(_IQ(.00052284),(i_Lbuf+17076)>>9); // shift+17076

#endif


/*determine peak input voltage of current line half-cycle*/ #if (1)

// compute the mean value of four values around T/4

if (LHCcount==NSCPLHC2+1){

V_dp=(
(v_dLHC[LHCcount]>>2)+(v_dLHC[NSCPLHC2]>>2)+(v_dLHC[NSCPLHC2-
1]>>2)+(v_dLHC[NSCPLHC2-2]>>2) );

// keeps V_dp within allowed range
        zcdthresh=V_dp>>3;          // threshold for ZCD

}
```

```
        #endif


        /*determine V_o(2)*/ #if (1)

        if (LHCcount<NSCPLHC){   // only consider values from feasible interval

          if (LHCcount<=NSCPLHC2)

          v_osumLQC1+=(v_o-V_oref)>>1; // integrate v_or during first line quarter-
cycle

        else v_osumLQC2+=(v_o-V_oref)>>1; // integrate v_or during second line
quarter-cycle

        // V_o2 is computed at ZC
        }

        #endif


        /*compute estimated L, R_L*/ #if (1)

        v_Lsum+=v_L[0]>>2;

         if (LHCcount==NSCPLHC2){         // line quarter-cycle completed

        v_LsumLQC=v_Lsum;  // v_LsumLQC corresponds to S/H output for L ORIG

      }

        // estimates are computed at ZC

        #endif


        // Enables adaptation only after Vo reaches its reference (380-12 = 368V)
          if (v_o>=V_oref-_IQ(12.0)){

                adapt = 1;            //Switch adaptation on when Vout is as reference
```

```
}

 else {

        adapt = 0; //Switch adaptation off when Vout is lower than reference
}

if (V_dp< _IQ(60)) {    //Resets R_L and L to initial values at Vd RMS ~ 40V

        R_L_imp = R_L_0;
        L_imp = L_0;
}


/*update implemented L, R_L*/ #if (1)

if (adapt){

deltaL[0]=L_est-L_0;            // compute model parameter error

L_corr[0]=_IQmpy(fL1,L_corr[1]) + _IQmpy(fL2,(deltaL[0]+deltaL[1])>>4);
// LP filter for deltaL

L_imp=L_0+L_corr[0];

deltaR_L[0]=R_L_est-R_L_0;

R_L_corr[0]=_IQmpy(fR_L1,R_L_corr[1]) +
_IQmpy(fR_L2,(deltaR_L[0]+deltaR_L[1])>>4);     // LP filter for deltaR_L

R_L_imp=R_L_0+R_L_corr[0];

L_corr[1]=L_corr[0];  // overwrite values from last cycle with current ones

R_L_corr[1]=R_L_corr[0];

deltaL[1]=deltaL[0];

deltaR_L[1]=deltaR_L[0];

}

#endif
```

```
        /*compute i_L */ #if  (1)

        i_LC[1]=i_LC[0]; // copy value from previous switching cycle to higher index
        fi_L1=_IQmpy(f_sw_10,L_imp<<10);
        fi_L2=_IQdiv(fi_L1-(R_L_imp>>1),fi_L1+(R_L_imp>>1));
        fi_L3=_IQdiv(_IQ(.5),fi_L1+(R_L_imp>>1));
        i_LC[0]=_IQrmpy(fi_L2,i_LC[1])+_IQrmpy(fi_L3,v_L[0]+v_L[1]);
        // v_L/(R+sL) as difference equation

        #endif


        /*voltage controller*/ #if (1)

        kappa[2]=kappa[1];                      // shift old values towards higher indices
        kappa[1]=kappa[0];
        eV[2]=eV[1];
        eV[1]=eV[0];
        eV[0]=_IQsat(V_oref-v_o,eVmax,eVmin);   // compute and limit voltage error

// controller equation: kappa[0] = a1v*kappa[1] + a2v*kappa[2] + b0v*eV[0] +
b1v*eV[1] + b2v*eV[2];


// all bv are <<16

        kappa[0]= _IQ30mpy(a1v,kappa[1]) + _IQ30mpy(a2v,kappa[2])+
((_IQ30mpyIQX(b0v,30, eV[0]>>8,GLOBAL_Q) +_IQ30mpyIQX(b1v,30,
eV[1]>>8,GLOBAL_Q) + _IQ30mpyIQX(b2v,30, eV[2]>>8,GLOBAL_Q))>>8);

        kappa[0]=_IQsat(kappa[0],kappamax,kappamin);     // limit kappa

        #endif

        /*current controller*/ #if (1)

        d[2]=d[1];       // copy values from previous switching cycle to higher indices
        d[1]=d[0];
        eI[2]=eI[1];
```

```
        eI[1]=eI[0];

        eI[0]=_IQmpyIQX(_IQmpy(V_dp,sine[LHCcount]),GLOBAL_Q,kappa[0],
30) - i_LC[0];

        eI[0]=_IQsat(eI[0],_IQ(1.0),_IQ(-1.0));                 // limit eI

// controller equation: d[0] = a1i*d[1] + a2i*d[2] + b0i*eI[0] + b1i*eI[1] + b2i*eI[2];

        d[0]=_IQmpy(a1i,d[1]) + _IQmpy(a2i,d[2]) + _IQmpy(b0i,eI[0])+
_IQmpy(b1i,eI[1]) + _IQmpy(b2i,eI[2]);

        d[0]=_IQsat(d[0],dmax,dmin);                                    // limit d
        d[3]=d[0]+d[0]-d[1];          //assumed D for the next switching cycle
        d[3]=_IQsat(d[3],dmax,dmin);

        #endif

        /*protection from faulty high D*/

        if (LHCcount>30 && LHCcount<150  && V_dp >_IQ(150.0)&&
d[3]>=_IQ(1.0)) {

         // switch off if D=1 at specified range

                        shutdown();
                        return 0;
        }


        /*determine whether zero crossing of line voltage occurred */

        #if (1)
        // average v_d over 10 switching cycles (v_dsum10)

        if (LHCcount==zcdavstart) {

v_dsum10=(v_dLHC[zcdavstart]>>3)+(v_dLHC[zcdavstart-
1]>>3)+(v_dLHC[zcdavstart-2]>>3)+(v_dLHC[zcdavstart-3]>>3)+
        (v_dLHC[zcdavstart-4]>>3)+(v_dLHC[zcdavstart-
5]>>3)+(v_dLHC[zcdavstart-6]>>3)+(v_dLHC[zcdavstart-7]>>3);
```

```
            }

        else {if (LHCcount>zcdavstart){

       v_dsum10=v_dsum10-(v_dLHC[LHCcount-8]>>3)+(v_dLHC[LHCcount]>>3);

        // subtract value from 10 switching cycles before and add latest one

        if (v_dsum10<=zcdthresh){

        // when averaged v_d drops below threshold, ZC occurs

        v_dLHC[0]=v_dLHC[LHCcount];    // copy current value to the beginning of
the array

        LHCcount=-1;                  // reset the switching cycle counter
        i_LC[0]=i_LC[1]=0;            // reset computed i_L to avoid integrator drift
        d[0]=d[1]=_IQ(1.0);           // force d to 1 to alleviate cusp distortion

        // compute V_o2

        V_o2=_IQmpy(_IQ(.0188996),v_osumLQC2-v_osumLQC1);
        v_osumLQC1=0;                                   // reset integrals of v_o2
        v_osumLQC2=0;

        // calculate R_L_est and L_est from integration results

        Vdp_Vo2=_IQdiv(V_dp>>8,V_o2);                   // V_dp/V_o2 >>8

        R_L_est=_IQmpy( _IQmpy(v_Lsum,facR_L) , Vdp_Vo2<<2 );     //  compute
estimate for R_L

        R_L_est=_IQsat(R_L_est,_IQ(11.0),0);                    // limitation

   L_est=_IQmpy( _IQmpy(v_LsumLQC>>2,facL) , Vdp_Vo2<<2 ) -
_IQmpy(R_L_est,oneover120pi);      //  compute estimate for L

        L_est=_IQsat(L_est,_IQ(0.025),0);                   // limitation
```

```
                v_Lsum=0;
                v_LsumLQC=0;
                i_Lmax=0;


        }}
        else if (LHCcount==20){
        }
        #endif

        LHCcount++;  // increment counter for the number of switching cycles
completed in the current line half-cycle


           if (LHCcount>=170)          {// failed to detect line voltage zero crossing

                if (v_o>_IQ(50.0)) {    // assures that zero crossing started

           shutdown(); // force gating signal to low, set error flag pin, disable interrupts
           return 0;

                  }                                 // exit program

                }

                processed=1;            // mark current switching cycle as processed
                EALLOW; GpioDataRegs.GPBCLEAR.bit.GPIOB5=1; EDIS;
// clear busy flag

                }       // end of if (samplecount==NSPSC && processed==0)

        }       // end of while(1)

}       // end of main

interrupt void adc_isr(){

        v_dsum+=AdcRegs.ADCRESULT0;
        v_osum+=AdcRegs.ADCRESULT1;
        v_Qsum+=AdcRegs.ADCRESULT2;
        i_Lsum+=AdcRegs.ADCRESULT3;
```

```
AdcRegs.ADCTRL2.bit.RST_SEQ1=1;        // reset sequencer 1

AdcRegs.ADCST.bit.INT_SEQ1_CLR=1;    // clear SEQ1 interrupt flag bit

PieCtrlRegs.PIEACK.all=PIEACK_GROUP1; // clear acknowledge bit for group 1


        samplecount++;                  // increment sample counter
        processed=0;                    // mark current switching cycle as unprocessed

}


void shutdown(){

        EvaRegs.GPTCONA.bit.T2PIN=0;            // force gating signal to low

        EALLOW; GpioDataRegs.GPASET.bit.GPIOA2=1; EDIS; // use GPIOA2
(pin 11) to flag error

        DINT;                                   // disable all interrupts
}
```