ECE533 – Closed-Loop Digital SMPS Design Project

Student #1: Mubarek Abdela (1001027738)

Student #2: Alok Deshpande (999859309)

Date: 20/4/2018

Introduction

This report details the development of a synchronous boost converter with a digital controller implemented in an FPGA. Both the converter hardware and software components are designed and subsequently tested. The report has two sections. Part A details the open-loop design, simulation, and testing. Part B details the closed-loop control design and testing. The report concludes with a summary of the results showing that the controller meets specifications.

Part A

This part describes the open-loop design. After designing the hardware components, the design is simulated to determine the effect of load on duty cycle and efficiency, as well as to verify that it meets open-loop requirements. Finally, the experimental testing results are shown.

1) DC Analysis

Below, the DC analysis of the designed boost converter with inclusion of the non-idealities is presented. The ideal conversion ratio for a boost converter is:

$$M_{ideal}(D) = \frac{V_o}{V_g} = \frac{1}{D'}$$

However, considering the conduction losses, this ratio must be re-derived.

From IVSB on the inductor:

$$< v_L > = D\left(V_g - I_L R_L - I_L R_{ON}\right) + D'\left(V_g - I_L R_L - I_L R_{ON} - V_o\right) = 0$$

Rearranging:

$$\frac{V_o}{V_g} = \frac{1}{D'} - \frac{I_L(R_L + R_{ON})}{D' V_g}$$

From CCB on the output capacitor:

$$< i_C > = D\left(-\frac{V_o}{R}\right) + D'\left(I_L - \frac{V_o}{R}\right) = 0$$

This gives:

$$I_o = D' I_L$$

This gives the non-ideal conversion ratio to be:

$$M_{non-ideal}(D) = \frac{1}{D'}\left(\frac{1}{1 + \frac{R_L + R_{ON}}{R}}\right)$$

a. Rang of Duty Cycle

By considering the output voltage to be at the nominal value of 32V, and heavy load operating current of 1 A and thus, 32Ω load resistance, the operating duty cycle range is obtained. The duty cycle should not be lower than:

$$\frac{(32V)}{(13V)} = \frac{1}{D'}\left(\frac{1}{1 + \frac{10.3m\Omega + 19m\Omega}{32}}\right)$$

$$D_{min} = 0.594$$

Similarly, the maximum duty-cycle would be: $D_{max} = 0.719$

Therefore, the range of allowable duty cycles is:

$$0.594 < D < 0.719$$

b. Range of Load Resistance

The load resistance is:

$$R = \frac{V_o}{I_o}$$

One can determine the maximum resistance by considering maximum output voltage for minimum output current:

$$R_{max} = \frac{(34.6V)}{(0.2A)} = 173\Omega$$

Likewise, the converse can be done to find the minimum resistance:

$$R_{max} = \frac{(29.5V)}{(1A)} = 29.5\Omega$$

Hence, the testing range of the load resistance is: $29.5\Omega < R < 173\Omega$

c. <u>Efficiency due to conduction loss</u>

From the above derivation of conversion ratios, the ideal and non-ideal conversion ratios were determined. Based on these, the efficiency is:

$$\eta = \frac{M_{non-ideal}(D)}{M_{ideal}(D)}$$
$$= \left[\frac{1}{D'}\left(\frac{1}{1 + \frac{R_L + R_{ON}}{R}}\right)\right] \div \left(\frac{1}{D'}\right)$$
$$= \frac{1}{1 + \frac{R_L + R_{ON}}{R}}$$

One may determine the lowest efficiency for the maximum output current, maximum input supply, and minimum duty cycle. Doing so gives a minimum efficiency of:

$$\eta = 1 - \frac{(1A)(10.3\text{m}\Omega + 19\text{m}\Omega)}{(1 - 0.595)(13V)}$$
$$\cong 0.967$$

This meets the specification, and so the design is feasible.

2) **Component selection**

a. <u>Output Capacitor</u>

The main conduction losses is mainly affected by the equivalent series resistance (ESR) of the inductor resistance (R_L) and the transistor ON resistance (R_{ON}). For the selected components, these were determined to be $R_L = 10.3m\Omega$ and $R_{ON} = 19m\Omega$.

To determine the minimum value of C_{out}, one need just meet the output voltage ripple specification:

$$I_O = C_{out}\frac{dV_o}{dt}$$
$$= C_{out}\frac{2\Delta V_o}{DT_s}$$

$$C_{out} = \frac{I_O}{\Delta V_o}\frac{D}{2f_s}$$

The switching frequency of 150k Hz was selected and the value of C_{out} was maximized to ensure successful operation under the worst condition as shown below:

$$C_{out_{min}} = \frac{(1A)}{(350mV)}\frac{(0.7182)}{2(150kHz)}$$
$$\cong 19.75\mu F$$

Hence, this is the minimum capacitance required. Based on the calculation and component tolerance a $22\mu F$ capacitor was chosen.

b. Inductor

The inductor was selected to ensure operation in CCM under all conditions:

$$I_L = \Delta i_L$$

$$\frac{I_o}{D'} = \frac{V_g}{\mathrm{L}}\frac{D}{2f_s} \Rightarrow L = \frac{V_g}{\mathrm{I_o}}\frac{D'D}{2f_s}$$

$$L = \frac{(13V)}{(0.2\mathrm{A})}\frac{(1-0.595)(0.595)}{2(150kHz)}$$
$$\cong 56\mu H$$

Based on the above calculation, and components available on Digi-Key, a $68\mu H$ inductance was chosen.

c. <u>Transistor</u>

The RMS current $i_{Q,RMS}$ is given by:

$$i_{Q,RMS} = I_L\sqrt{D\left(1+\frac{1}{3}\left(\frac{\Delta i_L}{I_L}\right)^2\right)}$$

The DC component is given by:

$$I_L = \frac{I_o}{D'}$$

This value is maximized at the maximum duty cycle and output current so:

$$I_{L_{max}} = \frac{(1A)}{(1-0.718)} \cong 3.56A$$

For the selected inductance, the maximum ripple is:

$$\Delta i_L = \frac{V_g}{\text{L}} \frac{D}{2f_s}$$

$$= \frac{(13V)}{(68\mu\text{H})} \frac{(0.718)}{2(150kHz)}$$

$$\cong 0.490A$$

Hence, the RMS transistor current is:

$$i_{Q,peak} = I_L \sqrt{D\left(1 + \frac{1}{3}\left(\frac{\Delta i_L}{I_L}\right)^2\right)}$$

$$\cong 3.02A$$

The peak voltage across the high-side transistor is V_{out}. Hence, the transistors should be rated above 34.6V. In addition, by considering the switching frequency, the gate driver current and Figure of Merit (FOM), two transistors with the same data sheet were selected (i.e. rated voltage of 60 V, rated current of 7A, Ron of 19mΩ and gate charge of $20nC$).

d. Gate driver selection:

The gate driver must charge equal gate capacitances (having selected the same transistors), but the required charging time differs, based on the duty cycle.

For a switching frequency of $T_{sw} = \frac{1}{150kHz} = 6.67\mu s$, each transistor is ON for its duty cycle time. For M0, this is at minimum:

$$D_{min}T_{sw} = (0.595)(7.14\mu s) \cong 4.25\mu s$$

Whereas for the transistor M1:

$$D'_{min}T_{sw} = (1 - 0.718)(7.14\mu s) \cong 2.01\mu s$$

Choose the transistors to turn within a very short period of this time (e.g. 50x faster switching). These periods are, respectively for M0 and M1:

$$t_0 \cong 85ns \text{ and } t_1 \cong 40ns$$

The gate capacitances must charge to $V_{gs,min}$, which is chosen more than the threshold voltage in order to turn on the transistors with low R_{ON}. Knowing the gate charge $Q_{gs,min}$ associated with $V_{gs,min} = 10V$ from the datasheet of the transistor, one can determine the required driver output current to charge the gate within a time Δt as:

$$I_{drive} = \frac{Q_{gs,min}}{\Delta t}$$

For $Q_{gs,min} = 20nC$, as indicated in the MOSFET datasheet, the required output currents for the low-side and high-side drivers are to charge within times $\Delta t = t_0 \cong 85ns$ and $\Delta t = t_1 \cong 40ns$, respectively. These calculations give the required output currents to be, respectively:

$$I_{drive,0} \cong 394mA \text{ and } I_{drive,1} \cong 832mA$$

Choosing the higher of these two values, this means that the gate driver must be able to provide at least this much output current.

e. Bootstrap circuit design

The bootstrap design was done based on the theory presented in one application note [1].

The minimum capacitance in the bootstrap circuit depends on the charge that it must store to supply the driver (Q_{cb}), and the voltage the high side driver must provide this charge at (ΔV_{boot}).

The charge Q_{cb} must be sufficient to charge the gate capacitance and then ensure the gate driver supplies its quiescent current (I_{supply}) to keep the driver voltage ON for a period of at most $D_{max}T_{sw}$. Hence, the charge is calculated to be:

$$\begin{aligned} Q_{cb} &= Q_{gs,min} + D_{max}T_{sw}I_{supply} \\ &= (20nC) + (0.718)(7.14\mu s)(2.6mA) \\ &\cong 33.3nC \end{aligned}$$

The voltage at which this must then be supplied is equal to the maximum allowable change in voltage across the capacitance during a switching cycle (i.e. maximum ripple), since:

$$\Delta Q = Q_{cb} = C_b \Delta V_{boot}$$

The maximum ripple is chosen to be an arbitrarily small value as is done in [1] (2% of the DC gate drive voltage), so it is:

$$\begin{aligned}\Delta V_{boot} &= 0.02V_{drive} \\ &= 0.02(10V) \\ &= 0.2V\end{aligned}$$

Note that here the DC drive voltage was taken to be 10V, since that is the value in the range of input voltages for which the low R_{ON} design has been developed, as previously mentioned.

Knowing Q_{cb} and ΔV_{boot}, the bootstrap capacitance is:

$$\begin{aligned}C_{b,min} &= \frac{Q_{cb}}{\Delta V_{boot}} \\ &= \frac{(33.3nC)}{(0.2V)} \\ &\cong 0.23\mu F\end{aligned}$$

The actual value used was 0.23uF.

Note that: the selected gate driver needs no additional bootstrap diode or resistor.

3) Summary

The following table summarizes the selected components:

Table 1: Specifications of Selected Components

Parameter	**Value**	**Unit**
L	68	μH
R_l	10.3	$m\Omega$
R_{on1} @ $V_{in,min}$	19	$m\Omega$
R_{on2} @ $V_{in,min}$	19	$m\Omega$
Q_{gt1} @ $V_{in,min}$	20	nC
Q_{gt2} @ $V_{in,min}$	20	nC
C_{out}	22	μF
Volume of L + C	14350+399=14750	mm^3
Cost of L+C	9784.36	$/ 1000 units

Table 2: Selected Components	
Component	**Digikey/Newark Part #**
Gate driver IC	296-46403-1-ND
L	732-11714-ND
M_1	785-1438-5-ND
M_2	785-1438-5-ND
C_{out}	493-9884-1-ND

4) Simulation of the Design

The following test setup was used to verify that the steady-state conditions met the specification, and to estimate the conduction loss in the setup. Note that the only the parasitic losses associated with the on resistance of the MOSFETs and the ESR of the output capacitor and inductor were modelled. The switching losses and losses due to the gate driver and sensors were not modeled.

a. Duty Cycle (Vin=9V and Vin=14V)

Ideally, for load variations, there should be no need to change duty cycle to maintain constant output voltage (i.e. constant conversion ratio). However, when the converter is non-ideal, this is no longer the case.

Based on this setup, the variation in duty cycle with load current was first simulated to estimate the effect of some of the parasitic. Upon simulating, the following results were obtained:

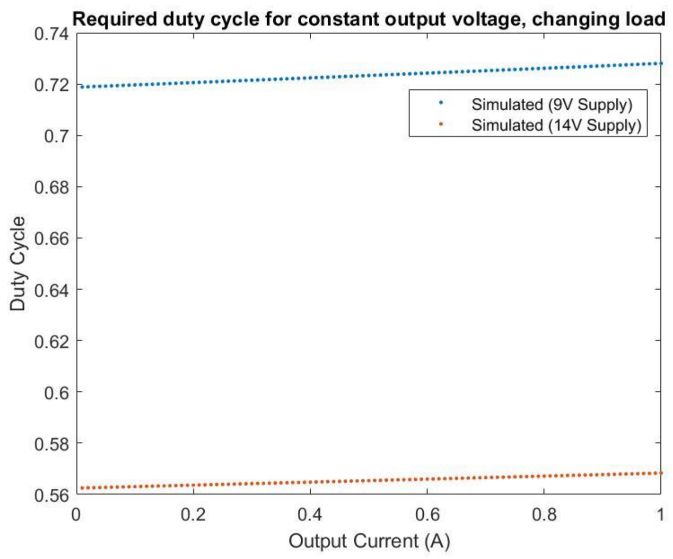


Figure 1: Variation in duty cycle (simulation only)

This shows a higher duty cycle is required to maintain the same conversion ratio as conduction losses increase.

b. Output voltage ripple

The output voltage ripple was one steady-state requirement to meet. Upon testing it in simulation at all combinations of the extreme cases of load and duty cycle, the following results were obtained:

Table 3: Measured output voltage ripple

		Output Voltage Ripple (V)	
		Output Current: 0.1A	Output Current: 1A
Supply Voltage (V)	9	0.027V	0.26V
	14	0.03V	0.2V

This shows the ripple requirement (ripple less than 0.35V) is theoretically met under all conditions.

c. Efficiency (Vin=9V and Vin=14V)

The other requirement tested was the efficiency, only accounting for conduction since other losses were not modelled. By simulating under different load conditions in the same was as done in experimental testing, the following result was obtained:

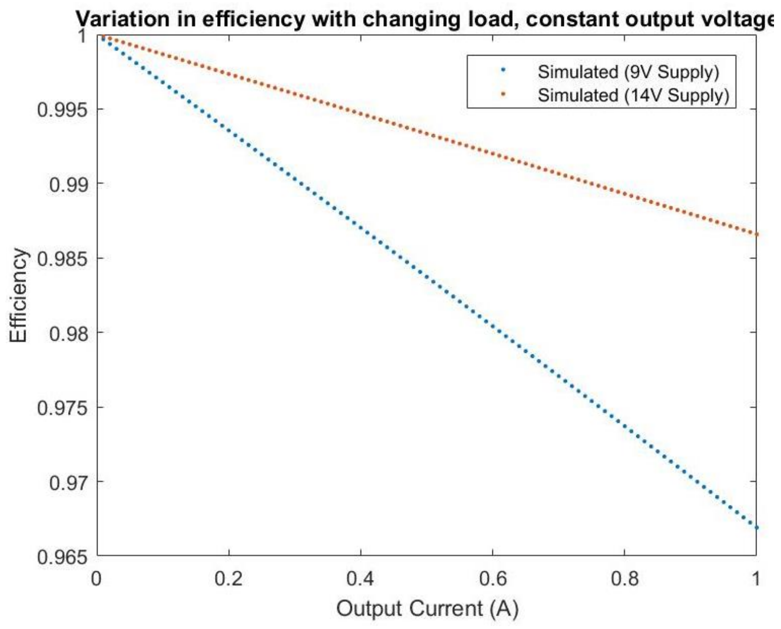


Figure 2: Variation in efficiency (simulation only)

This graph shows that the conduction loss is extremely minimal. It also shows that conduction loss increases with duty cycle and load, as expected. There is also higher efficiency at high supply voltage, which is reasonable because the current is reduced at higher input voltage for constant output voltage.

Note that the efficiency approaches 100% in the simulation for the case of no output current because the simulation only captures conduction losses. In actuality, the efficiency drops at light load due to the dominance of switching loss, as can be seen in the next section. Furthermore, the curve appears to be linear, but it would be non-linear if the switching losses were to be included

5) Experimental Testing

a. <u>Duty Cycle (Vin=9V and Vin=14V)</u>

This test was done by varying the load resistance to achieve a particular output current, and varying D to achieve a fixed V_{out} of 32V. The duty cycle was adjusted by using the switches in FPGA. Figure 4 shows the experimental results, alongside the simulation results for comparison:

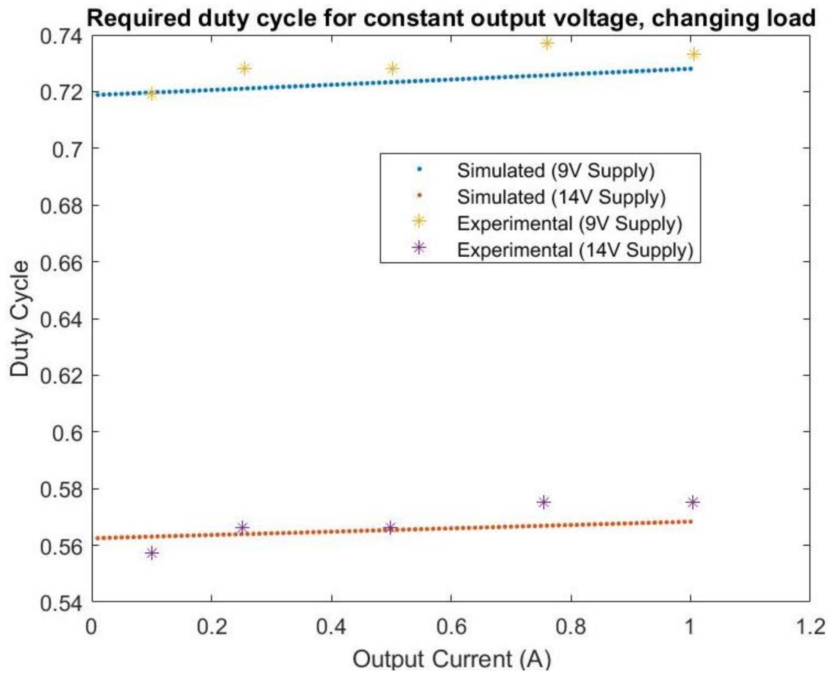


Figure 3: Variation in duty cycle (simulated and experimental)

These results show that the duty cycle varies very little with output current, just as in the simulated case. The experimental and simulation results deviate slightly, since switching and gate driver losses are not included. For the small variation that is present, this also shows that a higher duty cycle is required to produce the same amount of output current.

b. <u>Efficiency (Vin=9V and Vin=14V)</u>

The efficiency of the converter was tested at different load at a fixed output voltage of 32V. Figure 4 shows the results:

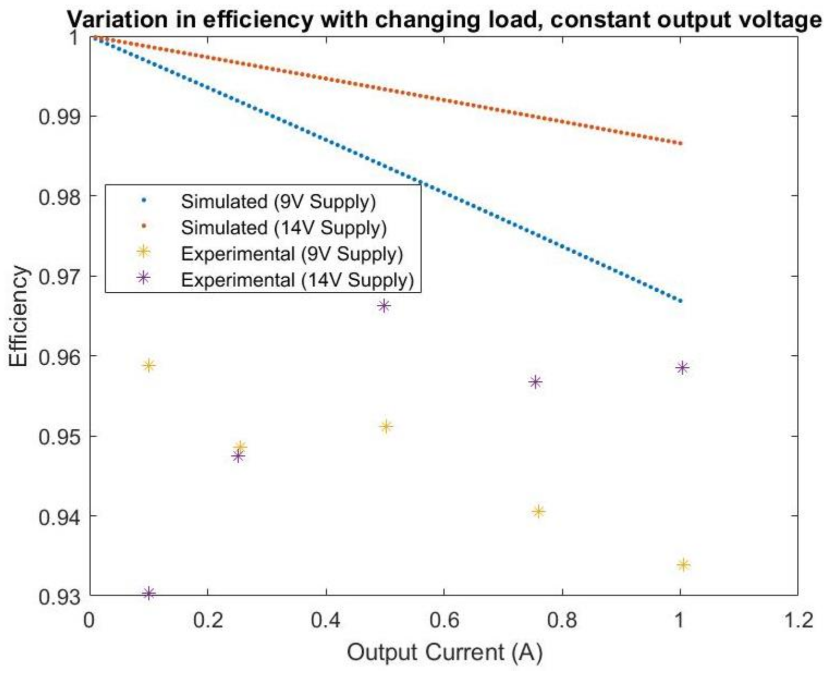


Figure 4: Variation in efficiency (simulated and experimental)

Firstly, this shows that under all load and supply conditions, the efficiency is great than 93% (thus, the specification is always met). It also shows that at low supply voltage, the conduction loss dominates, whereas at high supply voltage the conduction loss is reduced since the input current is lower.

c. <u>Output voltage ripple (for V_{in} =14V and I_{out}=0.1A)</u>

The ripple was measured at one condition of load (0.1 A) and supply (14V). The following plot shows the peak-to-peak output voltage ripple (130mV) is less than the specification (350mV) even experimentally.

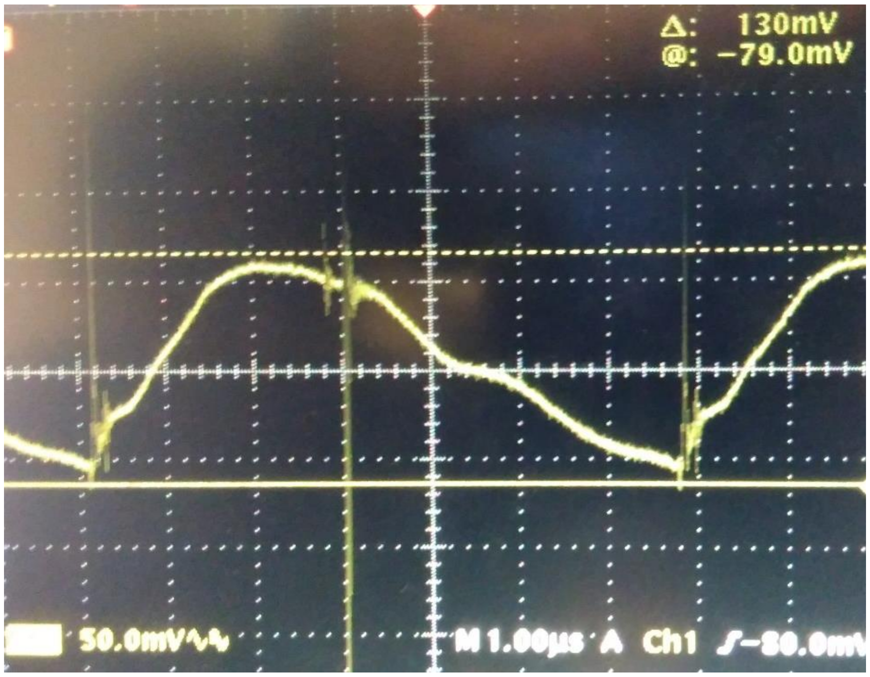


Figure 5: Output voltage ripple

The period of the ripple was also measured:

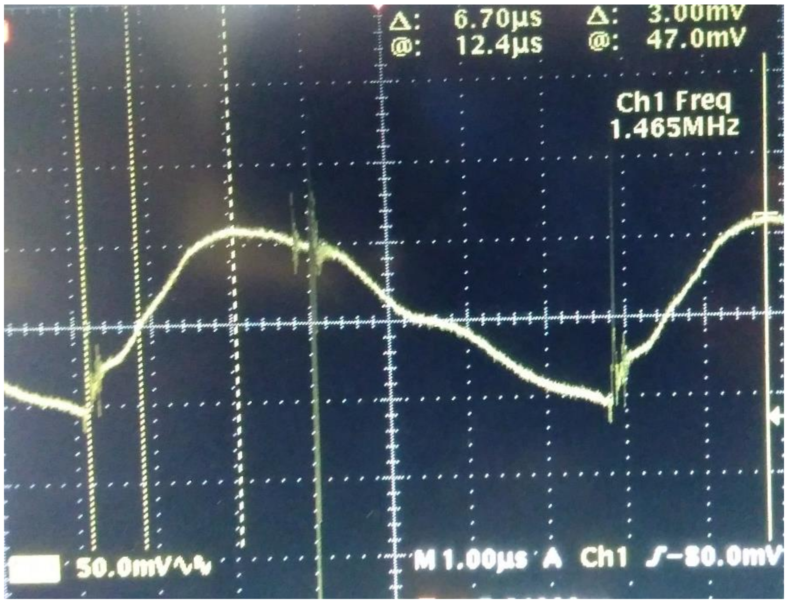


Figure 6: Output voltage ripple, showing switching period

The period is found to correspond to a frequency of $f = \frac{1}{6.7\mu s} = 150kHz$, which matches the switching frequency. This confirms that the viewed waveform is indeed the ripple, and not noise.

Note that in the above figures, the DC voltage has been filtered out, clearly show the ripple.

d. Soft start testing

Soft start was implemented for this converter by initially operating in open-loop mode until the output voltage reached the specified value. The duty cycle was ramped linearly to prevent overshoot in the output voltage. (This would otherwise occur during the transience of closed-loop startup.) The following figure shows the ramp in output voltage as the duty cycle increases over a period of 250ms.

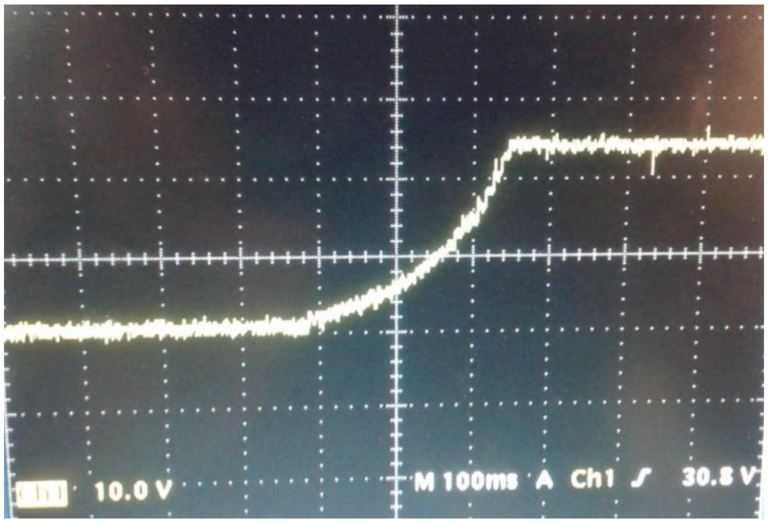


Figure 7: Output voltage during soft start

Part B

1) Sensor Design

Output Voltage Sensing

As shown in Appendix A, the output voltage sensor is a resistive divider with a capacitor in parallel to reduce noise at high frequency. Hence, the DC gain is:

$$K_{sense} = \frac{R_7}{R_7 + R_8}$$

And the pole must be:

$$\omega_{sense} = \frac{1}{C(R_7||R_8)}$$

This gives a transfer function of:

$$\begin{aligned} H_{sense}(s) &= \frac{K_{sense}}{1 + \frac{s}{\omega_{sense}}} \\ &= \frac{R_7}{R_7 + R_8} \cdot \frac{1}{1 + sc(R_7||R_8)} \end{aligned}$$

The resistors were selected to step down the output voltage from its DC value (nominally 32V) to the reference value of the ADC (3.5V). To achieve this, resistor values of $R_7 = 80.6k\Omega$ and $R_8 = 10k\Omega$ were chosen.

$$K_{sense} = \frac{R_7}{R_7 + R_8} = \frac{3.5}{32}$$

It is also desired to reduce the power loss due to the sensor resistors, so the resistors should be chosen to be large.

The capacitance C$_2$ was chosen to be 1nF. This gives a pole of $\omega_{sense} = 112.41\ k\ rad/s$.

Based on these values, the sensor transfer function is:

$$H_{sense}(s) = \frac{0.1104}{8.896 \times 10^{-6}s + 1}$$

Sensor design for the input voltage is given in Appendix B.

2) Controller Design

In order to design a compensator, the small-signal model of the converter is derived as shown in Appendix B and based on the analysis, the following control to output transfer function was obtained.

$$G_{vd}(s) = \frac{\left(\frac{1}{sC} + R_c\right) || R_{load} \left(\frac{V_o}{D'}\right)}{\left(\frac{1}{sC} + R_c\right) || R_{load} + \frac{sL}{D'^2} + \frac{R_L + R_{on}}{D'^2}} - I_L \left(\frac{sL + R_L + R_{on}}{D'^2} || \left(\frac{1}{sC} + R_c\right) || R_{load}\right)$$

The plant transfer function $G_{vd}(s)$ of the converter is plotted using MATLAB as shown below.

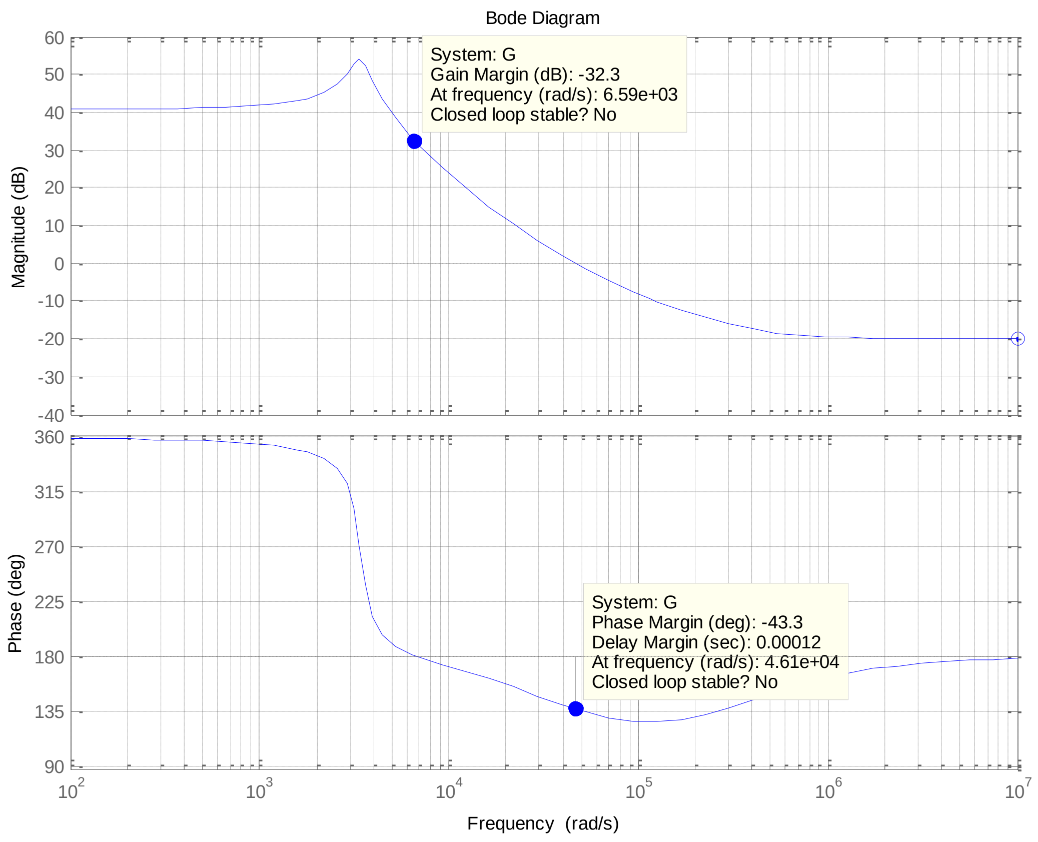


Figure 8: control to output transfer function

The controller was designed for the worst case (9V input voltage and 1A output current) operating conditions. Given the above transfer function, the PWM gain and the output sensor transfer function, the uncompensated open-loop gain is:

$$T(s) = \frac{1}{V_m} G_{vd}(s) H_{sense}(s)$$

Bode plot of the uncompensated system is given in Figure 9 and the figure shows that the system is unstable.

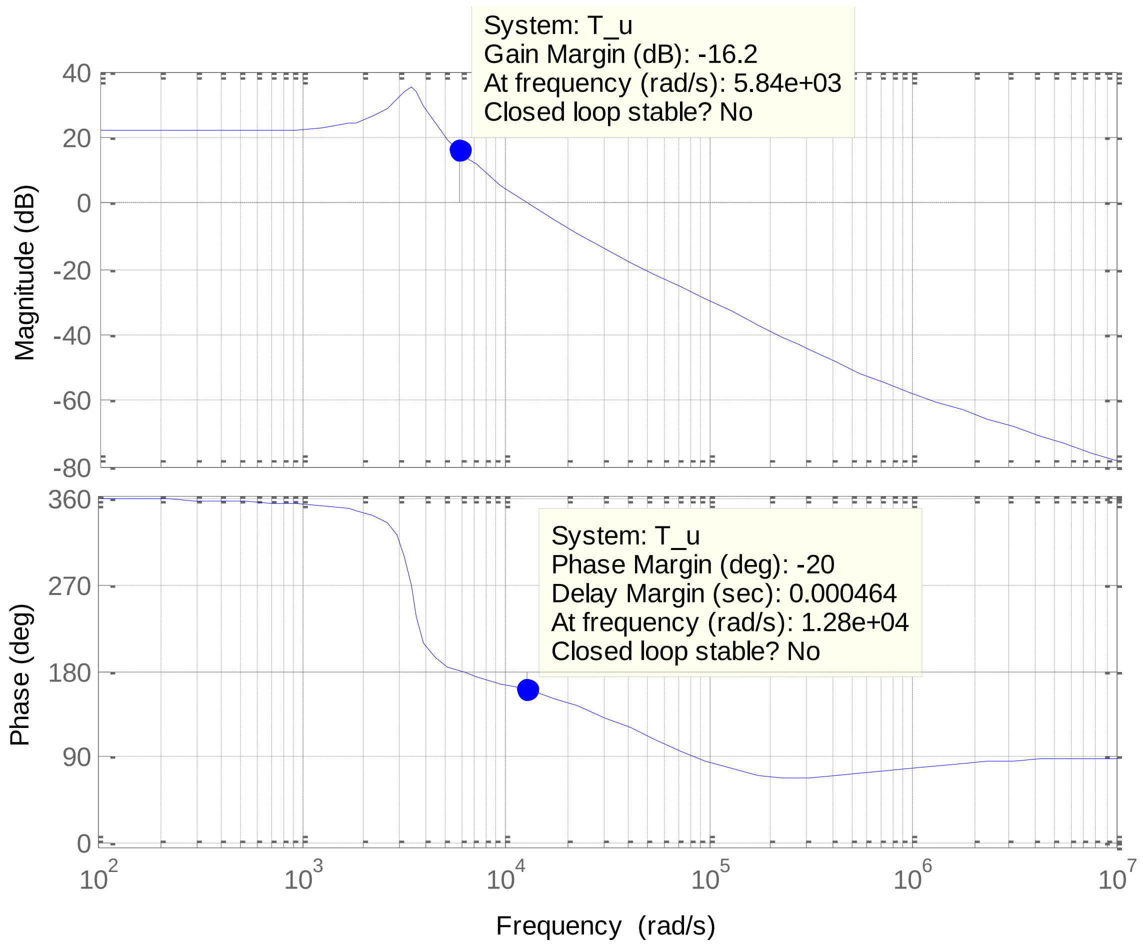


Figure 9: Bode plot of the uncompensated open-loop transfer function

The controller was designed using SISOTOOL in MATLAB to achieve a phase margin above 45° and a crossover frequency below one-tenth of the switching frequency. Furthermore, the sampling frequency of the ADC was taken into consideration. A PID controller meeting the specifications was designed and the transfer function is given below:

$$Gc(s) = \frac{34.246(s^2 + 2976\,s + 2.318 \times 10^6)}{s(s + 5.34 \times 10^5)}$$

The plots obtained from SISOTOOL, which includes the compensated system in Figure 10.

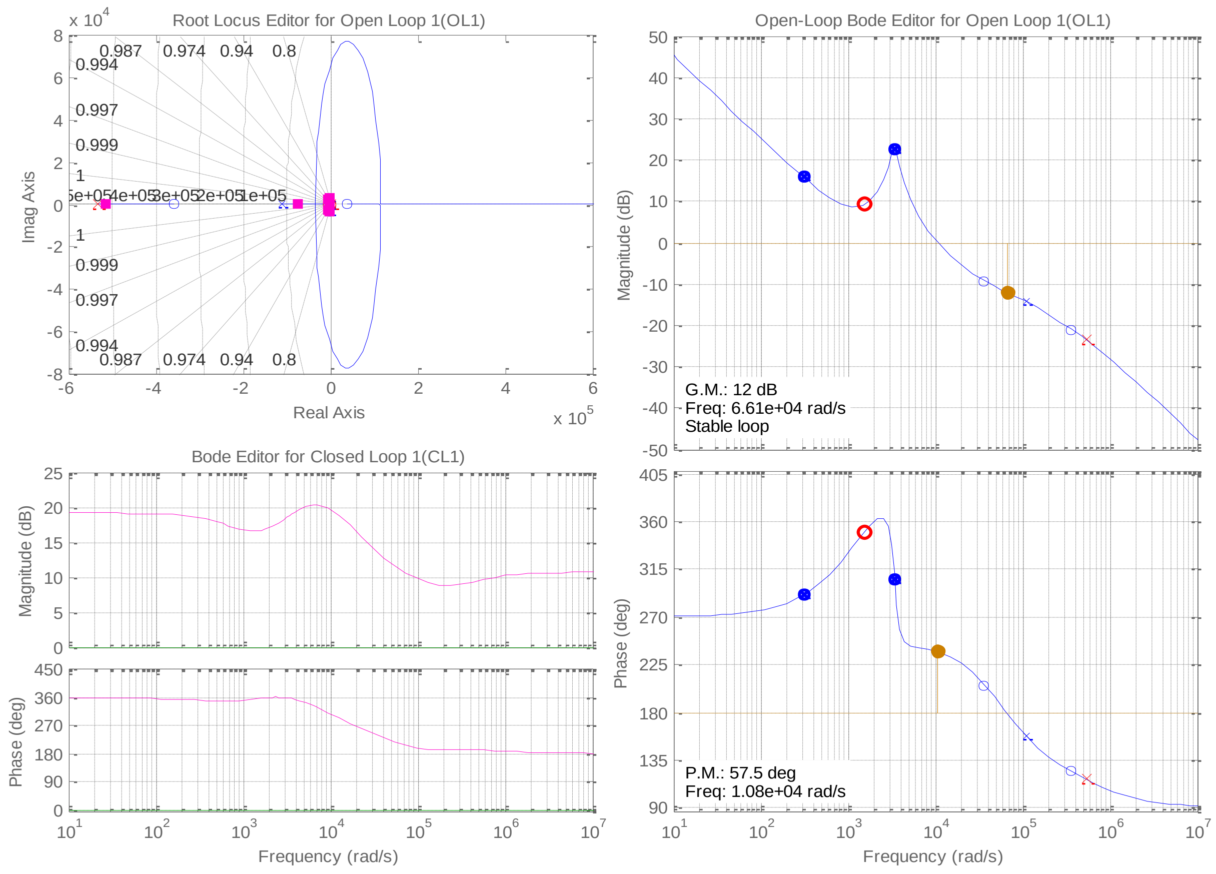


Figure 10: The results from SISOTOOL.

The controller achieves a phase margin of 57.5°, infinite DC gain and a crossover frequency of$1.08 \times 10^4 rad/s$, which is much less than the switching frequency of 150 kHz, but still it could provide a reasonably fast response.

Since we need a digital controller, a bilinear transformation of the compensator transfer function was obtained by using a sampling frequency of 250 kHz as shown below:

$$Gc = \frac{34.246(z^2 - 1.995\,z + 0.9951)}{(z-1)(z-0.1181)}$$

Upon taking the inverse transform, the resulting compensator in the time domain is:

$$d(t+1) = 1.1181d(t) - 0.1181d(t-1) + 34.246(e(t) - 1.995e(t-1) + 0.9951e(t-2))$$

3) Experimental and Simulation Test Results

The discrete controller was tested by using a code in MATLAB with PLECS block-set and Figure 11 given bellow shows the block diagram used for testing the controller.

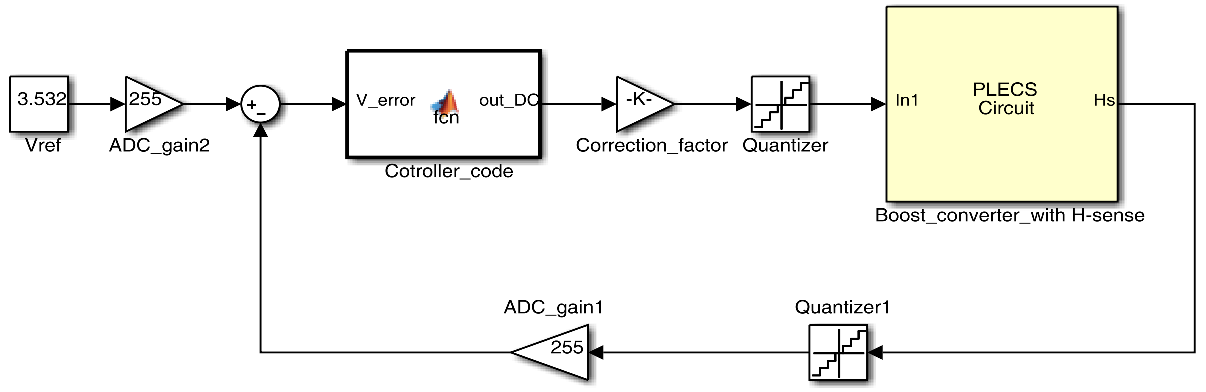


Figure 11: The block used for testing the discrete controller.

In addition, the variation of phase margin of the controller with the supply/input voltage and the output load is given in Figure 12. The figure shows that the controller meet the specification under all conditions.

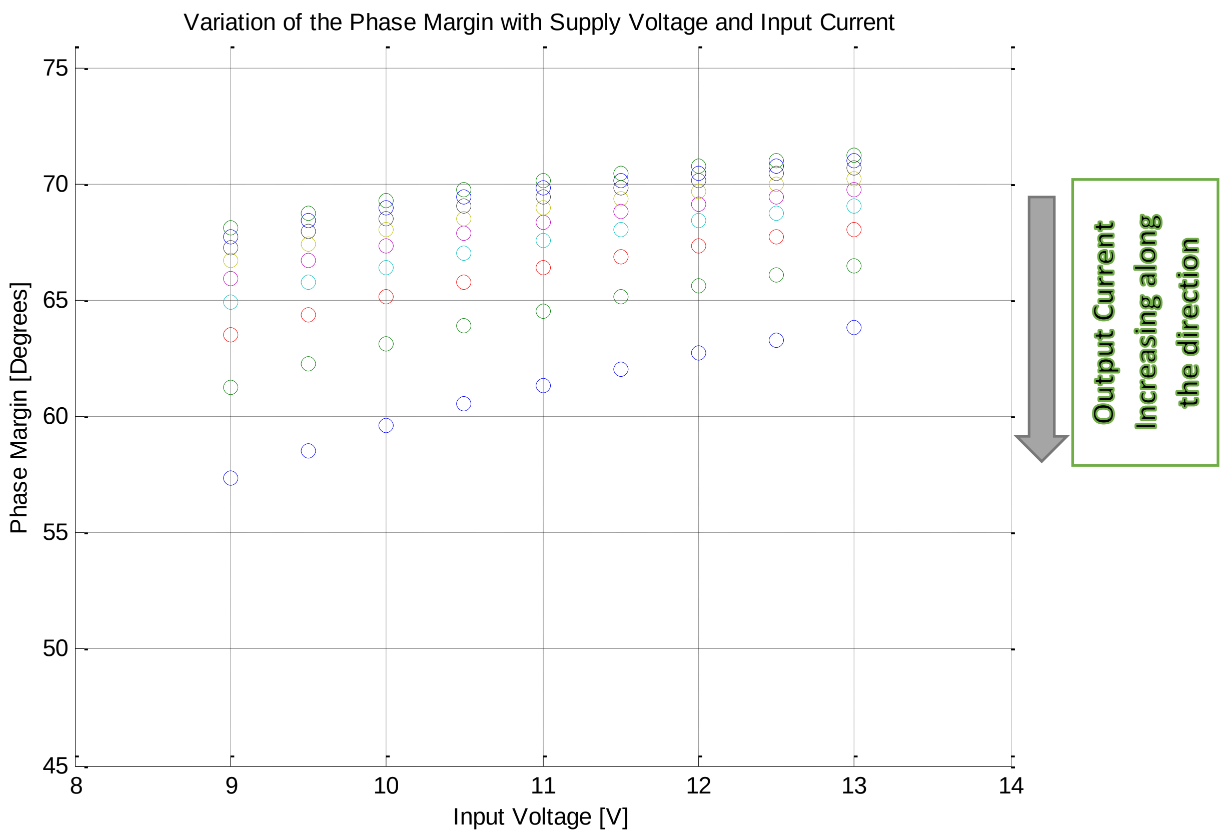


Figure 12: variation of the phase-margin with input voltage and load

Based on the above compensator, the full closed-loop system is tested in simulation. The system is tested with a load step response having the following characteristic:

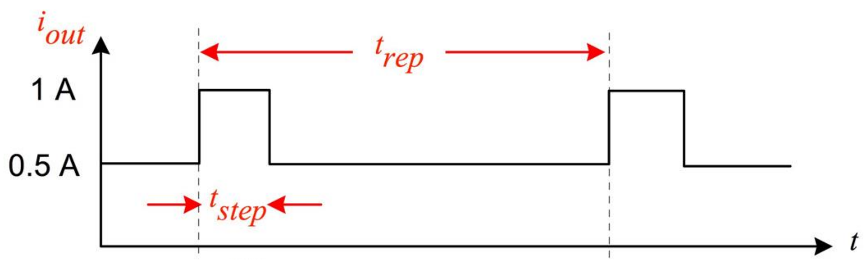


Figure 13: load step for controller test

Here, t_{step} is a step time that is designed to be longer than the settling time, and t_{rep} is a given step repetition time ($t_{rep} = \frac{1}{50Hz} = 20ms$).

To achieve the above response the resistor varied from:

$$R_1 = \frac{V_o}{I_{out}} = \frac{32V}{0.5A} = 64\Omega$$

to:

$$R_2 = \frac{V_o}{I_{out}} = \frac{32V}{1A} = 32\Omega$$

and back.

Testing with such a step in simulation using the Simulink model given in Figure 11, one obtains the following result:

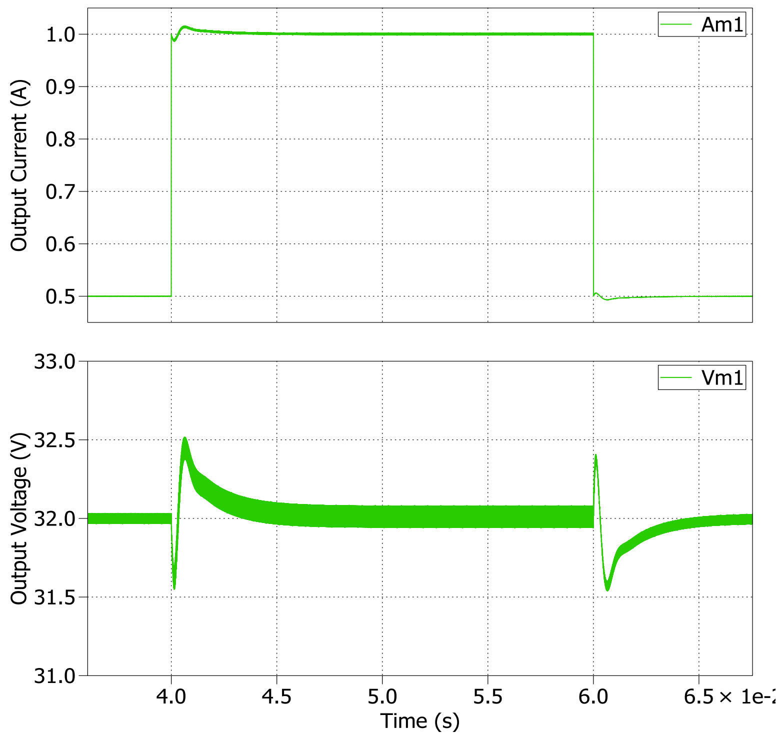


Figure 14: Load step response of the closed loop system

The voltage overshoot is up to 32.42V, and undershoot is down to 31.67V, for a steady-state voltage of 32V. This is within the range requirements (29.5V-34.6V), so this controller meets the specification.

Figure 15 shows experimental load step response of the controller from 160Ω (0.2 A) to 80Ω (0.4A) using the electronic load. The output voltage settles to the controlled value within 3ms. The sharp falling edge of the transient response of the above plot shows that the impact of the equivalent series resistance on the response of the controller is significant. Deviation of the voltage from the controlled value is limited to approximately 700mV.

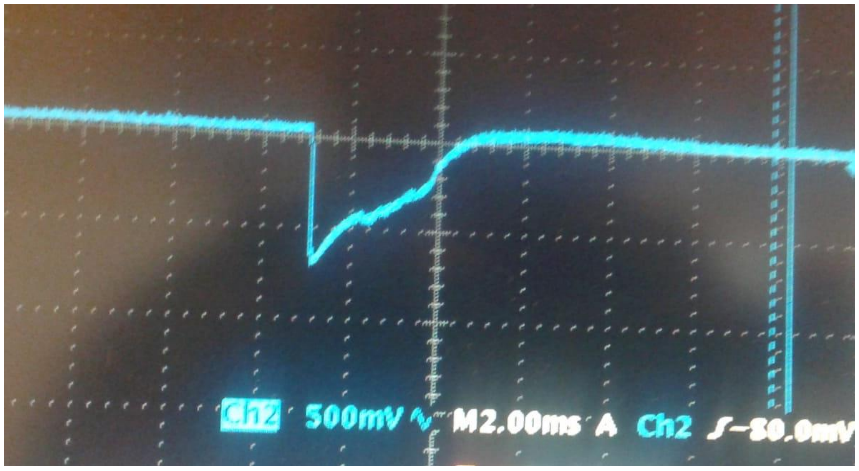


Figure15: Experimental result for load step 160Ω to 180Ω.

Please note that unfortunately, we were unable to plot the controller response for the resistance step of 80Ω to 160Ω on the same plot as the above, since our gate driver for the load step is not functioning properly.

For a resistance step of 160Ω to 32Ω the result shown in Figure 16 was obtained. The result shows that there is a significant delay in the transient response of the controller. As it can be clearly seen from the figure there are spikes/noises. These can be potentially due to the Electromagnetic interference during probing (since the controller is clearly not responding to these changes although they are within the sensor bandwidth).

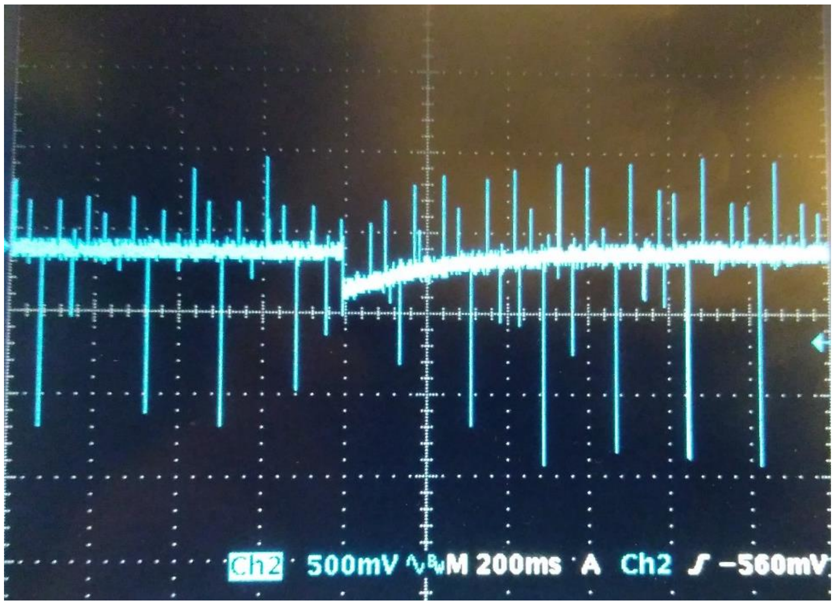


Figure 16: Experimental result for load step 60Ω to 32Ω

Remark and Conclusion

A reasonably fully operational converter with a digital controller was successfully designed. However, there were challenges in the process of designing both the hardware and software components. The list of issues raised during the project are given below:

1) *Ringing at the gate driver:* this was a main issue particularly at the beginning of the project, the high-side MOSFET was damaged once. Initially we had 5Ω gate resistors, but due to ringing issues these values had to be increased to a gate resistance of 10Ω for the low-side switch and 30Ω for the high-side switch (the inductance of the bootstrapping capacitor might have increased the ringing for the high-side MOSFET). We used sockets for the MOSFETs and the inductance of the sockets might have contributed to the increased ringing.
2) *Issue with gate driver*: in this project about 3 different gate driver were damaged. Two gate drivers were destroyed during closed loop testing of the final digital controller.
3) *Short circuits*: there were many incidents of shorting in the circuit due to soldering issues and also failure of the MOSFETs.
4) *Working with FPGA*: implementing the designed controller in FPGA was a big challenge, in particular operation with signed numbers in FPGA caused many problems and it required a long and time consuming debugging of the code.
5) *Soldering ADC and Buffer*: soldering these components required sharp soldering iron tip or hot air gun which were not easily accessible in the lab.

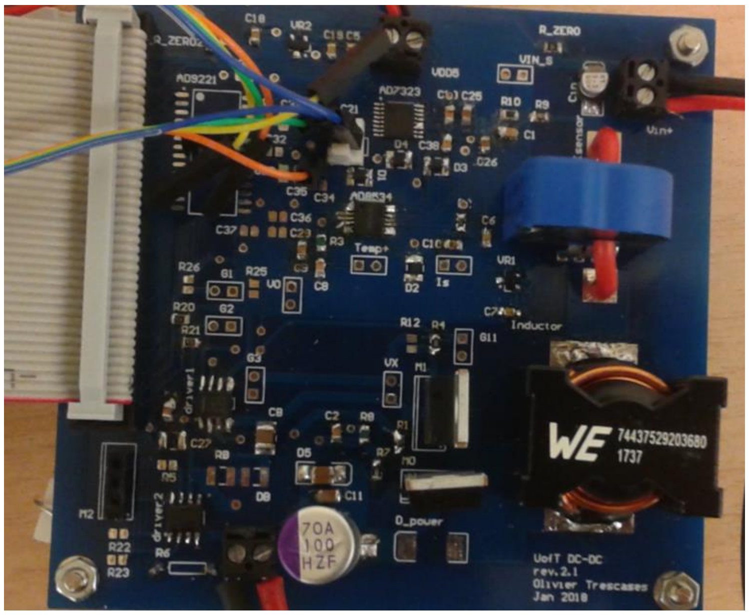


Figure 17: the designed boost converter

Reference

[1] Silicon Labs, “AN486: High-Side Bootstrap Design Using ISO Drivers in Power Delivery Systems”. Available at: https://www.silabs.com/documents/public/application-notes/AN486.pdf

Appendix A

Small signal modelling of the designed boost converter

For 0<t<d(t)T$_s$:

$$v_L(t) = v_g(t) - i_L(t)R_L - i_L(t)R_{ON}$$

$$i_c(t) = -\frac{v_o(t)}{R}$$

$$i_g(t) = i_L(t)$$

For d (t) T$_s$ < t < T$_s$:

$$v_L(t) = v_g(t) - i_L(t)R_L - i_L(t)R_{ON} - v_o(t)$$

$$i_c(t) = i_L(t) - \frac{v_o(t)}{R}$$

$$i_g(t) = i_L(t)$$

The averaged model is:

$$\boldsymbol{< v_L(t) >= d(t)(< v_g(t) > -< i_L(t) > R_L -< i_L(t) > R_{ON}) + d'(t)(< v_g(t) > -< i_L(t) > R_L -< i_L(t) > R_{ON} -< v_o(t) >)}$$

$$< i_c(t) >= d(t)\left(-< \frac{v_o(t)}{R} >\right) + d'(t)\left(< i_L(t) > -< \frac{v_o(t)}{R} >\right)$$

$$< i_g(t) >=< i_L(t) >$$

Perturbing and linearizing these equations:

$$L\frac{I_L + \widehat{\imath_L}}{dt} = (D + \hat{d})(V_g + \widehat{v_g} - (I_L + \widehat{\imath_L})R_L - (I_L + \widehat{\imath_L})R_{ON}) + (D - \hat{d})\left(V_g + \widehat{v_g} - (I_L + \widehat{\imath_L})R_L - (I_L + \widehat{\imath_L})R_{ON} - (V_g + \widehat{v_g})\right)$$

$$C\frac{V_o + \widehat{v_o}}{dt} = (D + \hat{d})\left(-\frac{V_o + \widehat{v_o}}{R}\right) + (D - \hat{d})\left((I_L + \widehat{\imath_L}) - \frac{V_o + \widehat{v_o}}{R}\right)$$

$$I_g + \widehat{\imath_g} = I_L + \widehat{\imath_L}$$

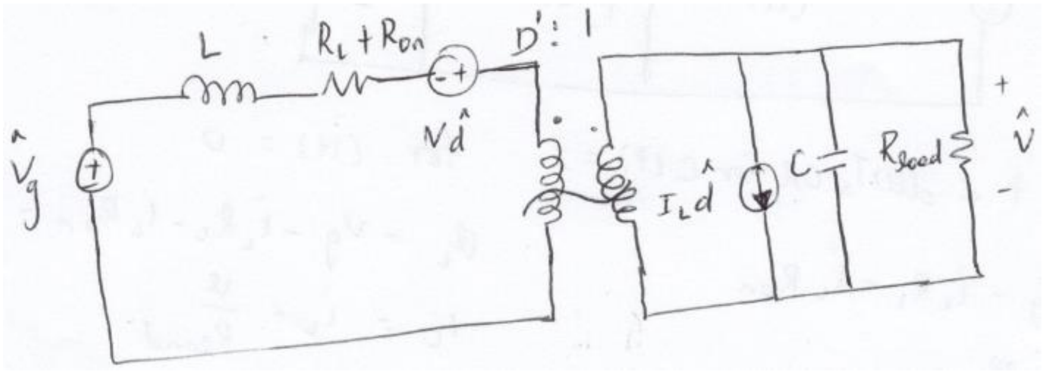
Removing the DC components and 2nd order terms, one obtains the AC small-signal dynamics:

$$L\frac{\widehat{\imath_L}}{dt} = \widehat{v_g} + \widehat{\imath_L}(R_L + R_{ON}) - D'\widehat{v_o} + V_O\hat{d}$$

$$C\frac{\widehat{v_o}}{dt} = -\hat{d}I_L + D'\widehat{\iota_L} - \frac{\widehat{v_o}}{R}$$

$$\widehat{\iota_g} = \widehat{\iota_L}$$

Upon taking the Laplace transforms, the equivalent circuit models are:



Setting all other perturbation set to zero, and solving for $G_{vd}(s) = \frac{\widehat{v_o}}{\hat{d}}$, the transfer function becomes:

$$G_{vd}(s) = \frac{\frac{1}{sC}||R\left(\frac{V_o}{D'}\right)}{\frac{1}{sC}||R + \frac{sL}{D'^2} + \frac{R + R_{on}}{D'^2}} - I_L\left(\frac{sL}{D'^2} + \frac{R + R_{on}}{D'^2}\right)||\frac{1}{sC}||R$$

The control to output transfer function of the system including the equivalent series resistance of the output capacitor is given below.

$$G_{vd} = \frac{\left(\frac{1}{sC} + R_{ESR}\right)||R\left(\frac{V_o}{D'}\right)}{\left(\frac{1}{sC} + R_{ESR}\right)||R + \frac{sL}{D'^2} + \frac{R + R_{on}}{D'^2}} - I_L\left(\frac{sL}{D'^2} + \frac{R + R_{on}}{D'^2}\right)||\left(\frac{1}{sC} + R_{ESR}\right)||R$$

Appendix B: input Voltage Sensing

Similar to the output voltage, the input voltage sensor also has the same configuration with the low pass filter. The DC gain is:

$$K_{sense} = \frac{R_9}{R_9 + R_{10}}$$

With the input voltage being at most 14V, this voltage must be stepped down to the level of the ADC. Because a signed value is not needed for the input voltage to indicate deviation away from a midrange value (i.e. it is not used in the linear control loop as an error), it suffices to just step down the maximum voltage to 5V. Hence, the DC gain is:

$$K_{sense} = \frac{R_9}{R_9 + R_{10}} = \frac{5V}{14V}$$

To design this, the resistors chosen are then $R_9 = 50k\Omega$ and $R_{10} = 100k\Omega$, approximately achieving this ratio.

The pole for the filter is still the same form:

$$\omega_{sense} = \frac{1}{C_1(R_9||R_{10})}$$

This time, however, the pole must only be placed so that the signal being sampled by the ADC is not aliased as a result of the sampling. Based on a sampling rate of $f_{samp} = 250kHz$, the Nyquist frequency is $f_{Ny} = 125kHz$. The signal must be significantly attenuated at this frequency to avoid aliasing and remove high frequency, so the pole is placed at a significantly lower. Choosing a low pole frequency of $f_{sense} = 5kHz$ based on the components available, a suitable capacitance of:

$$\begin{aligned} C_1 &= \frac{1}{2\pi(R_9||R_{10})} \\ &= 1nF \end{aligned}$$

can be used.

Appendix C:

Code used for testing the converter.

```verilog
//MAIN

module DPWM (
    input [17:0] SW          // Selection Bits
//, input [RESOLUTION-1:0] SW    // Value Bits
,   input [3:0] KEY
,   input CLOCK_50

,   inout [35:0] GPIO_0
,   output [35:0] GPIO_1
,   output [3:0] LEDG
,   output [17:0] LEDR
);

wire adc_CLK;

pll DPWM_PLL (
    .inclk0(CLOCK_50),
    .c1(adc_CLK),
);


wire [15:0] adc_output;
spi ad7324(
    .DOUT(GPIO_0[1]),
    .CLK_IN(adc_CLK),
    .R(KEY[3]),
    .HOLD(CLK1MHz),
    .READ_ALL(1'b0),

    .DIN(GPIO_0[7]),
    .CS(GPIO_0[5]),
    .CLK_OUT(GPIO_0[3]),
    .DATA_READ(adc_output)
);
//assign LEDR[15:0] = adc_output;

reg [9:0] maxcount;
always@ (*)
    case (SW[3:0])

        4'b0000: maxcount=10'b1111101000;//1000 and freq = 50khz
        4'b0001: maxcount=10'b1100000001;//769 and freq = 65khz
        4'b0010: maxcount=10'b1001110001;//625 and freq = 80khz
        4'b0011: maxcount=10'b1000001110;//526 and freq = 95khz
        4'b0101: maxcount=10'b0110010000;//400 and freq = 125khz
        4'b0110: maxcount=10'b0101100101;//357 and freq = 140khz
        4'b1011: maxcount=10'b0101001110;//334 and freq =
        150khz
        4'b0111: maxcount=10'b0101000010;//322 and freq = 155khz
        4'b1000: maxcount=10'b0100100110;//294 and freq = 170khz
        4'b1001: maxcount=10'b0100001110;//270 and freq = 185khz
        4'b1010: maxcount=10'b0011111010;//250 and freq = 200khz
        default: maxcount=10'b1111101000;
    endcase


wire S1, S2;


assign GPIO_0[32] = S1;
assign GPIO_0[34] = S2;


wire CLK1MHz;
//clock divider for the ADC input
```

```
clk divider GET 1MHz(
.clk(adc_CLK),
.reset(0),
.clk_out(CLK1MHz)
);

wire CLK10kHz;
//clock divider for the ADC input
clk_divider_10kHz GET_10kHz(
.clk(CLK1MHz),
.reset(0),
.clk_out(CLK10kHz)
);


assign GPIO_1[32] = CLK10kHz;



 //Soft start
 reg [7:0]Duty_Cycle;
//The following code pertains to the Controller
reg [1:0]identifier;
reg [11:0]Temp, Vin, Iin;
reg  signed [59:0]Vout;
reg  [59:0]controller_DC;
reg  signed[59:0]V_ref;
reg  signed [59:0]V_error, V_error1, V_error2;

reg  signed [59:0]out_DC, DClk;


always@ (posedge CLK10kHz)
begin
    identifier <= adc_output[14:13];
    V_ref <= $signed(60'b001111100);   // dropped the last 4 bits from 60'b00101101001101
    Vout <= $signed(adc_output[12:4]);

   if(identifier==2'b00)


   //2 controllers were developed, as shown below. Only Type 2 was ultimately selected
   //Using TYPE 2 CONTROLLER (PI)
   begin

        V_error = (V_ref - Vout); //255 ADC gain

        if (SW[17]==1) begin
           out_DC =60'b0011111100*255;
        end
        else begin
            out_DC = (DClk*524288 + 16811*V_error);
            out_DC = out_DC>>>19;  //dividing by 524288
             if(out DC > 60'b0101101000101101l) begin
                 out_DC =60'b0011111100*255; //since it will end up beign devided
            end
             DClk = out_DC;

        end
    end

    //TYPE 1 CONTROLLER (PID)
    /* begin

        Vout <= 0;    //Vout <= adc_output[11:3];
        V_error = (V_ref - Vout); //255 ADC gain
        controller_DC = 334*((11181*DC1k) - (1181*DC2k) + ((100000*V_error)-(199500*V_error1)+(99510*V_error2))); //334 DPWM gain

        //Setting History terms
        if (controller_DC > 60'b111110000000110011111010000)  begin //the value of controller_DC should not exceed (130050000) or 111110000000110011111010000


         controller_DC = 60'b1;//60'b111110000000110011111010000
         DC2k <=DC1k;
         DC1k <=controller_DC;
         V_error2 <=V_error1;
         V_error1 <=V_error;

        end
        else begin

        DC2k <=DC1k;
        DC1k <=controller_DC;
        V_error2 <=V_error1;
        V_error1 <=V_error;

    end */




    end // identifier  for cases  2'b01, 2'b10 and 2'b11


    wire [59:0] quotient; //Value of quotient should not exceed 255

    division (out_DC, 60'b011111111, quotient);//dividing by 255


 //here assign quotient[9:0] to duty Cycle
 gating (.CLK_50(CLOCK_50), .DutyCycle(quotient[7:0]), .maxcount(10'b0101001110), .reset(KEY[0]),.SET(KEY[1]), .S1(S1), .S2(S2));


endmodule



module gating(CLK_50, DutyCycle, maxcount, reset, SET, S1,  S2);

  input [9:0]DutyCycle, maxcount;
  input reset, SET, CLK_50;
  output S1, S2;
  reg [9:0] counter;//is this initalized to zero by default???
  reg  C_1, C_2; //temporary variables
    always @(posedge CLK_50, negedge reset)
    begin
       if (!reset)
            begin
                C_1<=0;
                C_2<=0;
                counter = 10'b0;
            end
        else if(!SET)
            begin
                C_1<=0;
                C_2<=0;
            end
        else
            begin
```

```
                        if (counter < maxcount) // at every 0.5 seconds, activate
                            counter = counter + 1;
                        else
                            counter = 0;
                //the duty cycle adjusted based on the switch input
                if (counter > DutyCycle)
                    begin
                       C_2<=1'b0;
                        C_1 <= 1'b1;
                    end
                else
                    begin
                        C_1<=1'b0;
                        C_2 <= 1'b1;
                    end

                end

        end

    wire bit_value1, bit_value2;


    shiftn forDT1(3'b110, C_1, CLK_50, bit_value1);
    shiftn forDT2(3'b110, C_2, CLK_50, bit_value2);

    assign S2 = bit_value1 && C_1;
    assign S1 = bit_value2 && C_2;

    endmodule



    module shiftn (binary, w, Clock, bit_value);
    input [2:0]binary;
    integer n;
    always@ (binary)
        case (binary)

            3'b001: n=0;
            3'b010: n=1;
            3'b011: n=2;
            3'b100: n=3;
            3'b101: n=4;
            3'b110: n=5;
            default: n=0;
        endcase


    input w, Clock;

    reg [5:0]Q;
    integer k;
    output reg bit_value;
    always @(posedge Clock)
            begin
                for (k = 0; k < 5; k = k+1)
                    Q[k] <= Q[k+1];

                Q[n] <= w;
                bit_value <= Q[0];
            end

    endmodule

    module DutyCylceConverter (SW_value, clk2, frequency, duty);
```

```
input clk2;
input [3:0] frequency;
input [7:0] SW_value;
output reg [9:0] duty;

always@ (*)
    case (frequency)

        4'b0000: duty = SW_value*8'b0000_0100;//1000 and freq = 50khz
        4'b0001: duty = SW_value*8'b0000_0100;//769 and freq = 65khz
        4'b0010: duty = SW_value*8'b0000_0100;//625 and freq = 80khz
        4'b0011: duty = SW_value*8'b0000_0100;//526 and freq = 95khz
        4'b0100: duty = SW_value*8'b0000_0010;//455 and freq = 110khz
        4'b0101: duty = SW_value*8'b0000_0010;//400 and freq = 125khz
        4'b0110: duty = SW_value*8'b0000_0010;//357 and freq = 140khz
        4'b0111: duty = SW_value*8'b0000_0010;//322 and freq = 155khz
        4'b1000: duty = SW_value*8'b0000_0010;//294 and freq = 170khz
        4'b1001: duty = SW_value*8'b0000_0010; //270 and freq = 185khz
        4'b1010: duty = SW_value*8'b0000_0001; //250 and freq = 200khz, whe should not
        go beyond 250!!!!!!!!!

        default: duty = SW_value*8'b0000_0100;
    endcase
endmodule


module deadtime_generator #(parameter RESOLUTION = 12)
(   input wire hf_clock
,   input wire [RESOLUTION-1:0] hs_deadtime
,   input wire [RESOLUTION-1:0] ls_deadtime
,   input wire HPWM
,   input wire LPWM
,   input wire reset

,   output wire pwm
,   output wire cpwm
);

//WIRES
wire [RESOLUTION-1:0] hs_dt_counter;
wire [RESOLUTION-1:0] ls_dt_counter;
wire deadtime_hs_mask = hs_dt_counter >= hs_deadtime;
wire deadtime_ls_mask = ls_dt_counter >= ls_deadtime;

positive_counter #( .WIDTH(RESOLUTION) ) HS_DT_CTR (
    .clk(hf_clock),
    .reset(~HPWM | reset),
    .enable(1'b1),
    .count(hs_dt_counter)
);

positive_counter #( .WIDTH(RESOLUTION) ) LS_DT_CTR (
    .clk(hf_clock),
    .reset(~LPWM | reset),
    .enable(1'b1),
    .count(ls_dt_counter)
);

assign pwm  = (HPWM & deadtime_hs_mask) & (~reset);
assign cpwm = (LPWM & deadtime_ls_mask) & (~reset);

endmodule


//
Reference://http://verilogcodes.blogspot.ca/2015/11/synthesisable-verilog-code-for-divisi
```

```
on.html

module division(A, B, Res);
 //the size of input and output ports of the division module is generic.
    parameter WIDTH = 40;
    //input and output ports.
    input [WIDTH-1:0] A;
    input [WIDTH-1:0] B;
    output [WIDTH-1:0] Res;
    //internal variables
    reg signed [WIDTH-1:0] Res = 0;
    reg signed [WIDTH-1:0] a1,b1;
    reg signed [WIDTH:0] p1;
    integer i;

    always@ (A or B)
    begin
        //initialize the variables.
        a1 = A;
        b1 = B;
        p1= 0;
        for(i=0;i < WIDTH;i=i+1)    begin //start the for loop
            p1 = {p1[WIDTH-2:0],a1[WIDTH-1]};
            a1[WIDTH-1:1] = a1[WIDTH-2:0];
            p1 = p1-b1;
            if(p1[WIDTH-1] == 1)    begin
                a1[0] = 0;
                p1 = p1 + b1;   end
            else
                a1[0] = 1;
        end
        Res = a1;
    end

endmodule


module input_sanitizer #( parameter RESOLUTION = 12 ) (
    input wire [RESOLUTION-1:0] dc
,   input wire [RESOLUTION-1:0] fs
,   input wire [RESOLUTION-1:0] dt1
,   input wire [RESOLUTION-1:0] dt2

,   output wire [RESOLUTION-1:0] san_dc
,   output wire [RESOLUTION-1:0] san_fs
,   output wire [RESOLUTION-1:0] san_dt1
,   output wire [RESOLUTION-1:0] san_dt2
);

// WIRES
wire [RESOLUTION-1:0] fs_inv;

// MAIN CODE

// Frequency Select Sanitization
assign fs_inv = {RESOLUTION{1'b1}} - fs;
assign san_fs = (fs_inv < 2) ? 2 : fs_inv;

// Duty Cycle Sanitization
assign san_dc = (dc > san_fs) ? san_fs : dc;

// Deadtime1 Sanitization
assign san_dt1 = (dt1 > san_dc) ? san_dc : dt1;

// Deadtime2 Sanitization
assign san_dt2 = (dt2 > (san_fs - san_dc)) ? (san_fs - san_dc) : dt2;
```

```

endmodule


module clk_divider #(parameter WIDTH = 5, parameter N = 20) (
input clk,
input reset,
output clk_out);


//input clk;
//input reset;
//output clk_out;

reg [WIDTH-1:0] pos_count, neg_count;
wire [WIDTH-1:0] r_nxt;

 always @(posedge clk)
 if (reset)
 pos_count <=0;
 else if (pos_count ==N-1) pos_count <= 0;
 else pos_count<= pos_count +1;

 always @(negedge clk)
 if (reset)

 neg_count <=0;

 else  if (neg_count ==N-1) neg_count <= 0;

 else neg_count<= neg_count +1;

assign clk_out = ((pos_count > (N>>1)) | (neg_count > (N>>1)));

endmodule

module clk_divider_10kHz #(parameter WIDTH = 7, parameter N = 100) (
input clk,
input reset,
output clk_out);


//input clk;
//input reset;
//output clk_out;

reg [WIDTH-1:0] pos_count, neg_count;
wire [WIDTH-1:0] r_nxt;

 always @(posedge clk)
 if (reset)
 pos_count <=0;
 else if (pos_count ==N-1) pos_count <= 0;
 else pos_count<= pos_count +1;

 always @(negedge clk)
 if (reset)

 neg_count <=0;

 else  if (neg_count ==N-1) neg_count <= 0;

 else neg_count<= neg_count +1;

assign clk_out = ((pos_count > (N>>1)) | (neg_count > (N>>1)));

```

```
endmodule

module clk_divider_100Hz #(parameter WIDTH = 19, parameter N = 500000) (
input clk,
input reset,
output clk_out);


//input clk;
//input reset;
//output clk_out;

reg [WIDTH-1:0] pos_count, neg_count;
wire [WIDTH-1:0] r_nxt;

 always @(posedge clk)
 if (reset)
 pos_count <=0;
 else if (pos_count ==N-1) pos_count <= 0;
 else pos_count<= pos_count +1;

 always @(negedge clk)
 if (reset)

 neg_count <=0;

 else  if (neg_count ==N-1) neg_count <= 0;

 else neg_count<= neg_count +1;

assign clk_out = ((pos_count > (N>>1)) | (neg_count > (N>>1)));

endmodule









module LCD_display(
  input CLOCK_50,    //    50 MHz clock
  input [1:0] KEY,      //    Pushbutton[0]
  input [1:0] SW,       //Using 2 switches
  output [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7,  // Seven Segment Digits
  input [3:0]   VinH,VinL,VoutH,VoutL,IoutH,IoutL,TempH,TempL,
  //output [1:0] LEDR,  //    LED Red
//    LCD Module 16X2
  output LCD_ON,    // LCD Power ON/OFF
  output LCD_BLON,    // LCD Back Light ON/OFF
  output LCD_RW,    // LCD Read/Write Select, 0 = Write, 1 = Read
  output LCD_EN,    // LCD Enable
  output LCD_RS,    // LCD Command/Data Select, 0 = Command, 1 = Data
  inout [7:0] LCD_DATA    // LCD Data bus 8 bits
);
//Add reset button
wire RST;
assign RST = ~KEY[0]; //KEY 0 for reset (key is normally high)

// reset delay gives some time for peripherals to initialize
wire DLY_RST;
Reset_Delay r0(    .iCLK(CLOCK_50),.oRESET(DLY_RST) );

// Send switches to red leds
//assign LEDR = SW[1:0];
```

```

// turn LCD ON
assign    LCD_ON       =    1'b1;
assign    LCD_BLON     =    1'b1;

//Take ADC input
//OPTION1: 8 bits x 4 measurements
//Will write to each output unit as hex character, not bit, so padded with 3 zeros at
start
/* reg [3:0] hex[31:0]; //32 output units, each a hex character
wire [3:0] output[31:0];
integer i, j;
always@(*) begin
    for (i = 0; i < 3; i = i +1) begin
        hex[i][0]=VinL[i];
        for (j = 1; j < 3; j = j +1) begin
            hex[i][j]=1'b0;
        end
    end
 end
assign output=hex; */

wire [3:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7;
//OPTION2: 2 hex x 4 measurement (+names)
assign hex0 = VinL; assign hex1 = VinH; assign hex2 = VoutL; assign hex3 = VoutH; assign
 hex4 = IoutL;
assign hex5 = IoutH; assign hex6 = TempL; assign hex7 = TempH;

LCD_Display u1(
// Host Side
   .iCLK_50MHZ(CLOCK_50),
   .iRST_N(DLY_RST),
   .hex0(hex0),.hex1(hex1),.hex2(hex2),.hex3(hex3),
   .hex4(hex4),.hex5(hex5),.hex6(hex6),.hex7(hex7),
// LCD Side
   .DATA_BUS(LCD_DATA),
   .LCD_RW(LCD_RW),
   .LCD_E(LCD_EN),
   .LCD_RS(LCD_RS)
);


// blank unused 7-segment digits
assign HEX0 = 7'b111_1111; assign HEX1 = 7'b111_1111; assign HEX2 = 7'b111_1111;
assign HEX3 = 7'b111_1111; assign HEX4 = 7'b111_1111; assign HEX5 = 7'b111_1111;
assign HEX6 = 7'b111_1111; assign HEX7 = 7'b111_1111;

endmodule

`ifndef  CUSTOM_LCD_DISPLAY_STRING
module LCD_display_string(index,out,hex0,hex1,hex2, hex3, hex4, hex5, hex6, hex7);
input [4:0] index;
input [3:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7;
output [7:0] out;
reg [7:0] out;
// ASCII hex values for LCD Display
// Enter Live Hex Data Values from hardware here
// LCD DISPLAYS THE FOLLOWING:
//----------------------------
//| Vin=hh Vout=hh           |
//| Iout=hh Temp=hh          |
//----------------------------

/*--                        ASCII HEX TABLE
--  Hex                        Low Hex Digit
-- Value  0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
```

```
------\----------------------------------------------------------------
--H  2 |  SP  !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
--i  3 |  0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
--g  4 |  @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
--h  5 |  P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
--   6 |  `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
--   7 |  p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ DEL
-----------------------------------------------------------------------
-- Example "A" is row 4 column 1, so hex value is 8'h41"
-- *see LCD Controller's Datasheet for other graphics characters available
*/

// Line 1
   always
     case (index)
    5'h00: out <= 8'h56;
    5'h01: out <= 8'h69;
    5'h02: out <= 8'h6E;
    5'h03: out <= 8'h3D;
    5'h04: out <= {4'h0,hex1};
    5'h05: out <= {4'h0,hex0};
    5'h06: out <= 8'h56;
    5'h07: out <= 8'h6F;
    5'h08: out <= 8'h75;
    5'h09: out <= 8'h74;
    5'h0A: out <= 8'h3D;
    5'h0B: out <= {4'h0,hex3};
    5'h0C: out <= {4'h0,hex2};
// Line 2
    5'h10: out <= 8'h49;
    5'h11: out <= 8'h6F;
    5'h12: out <= 8'h75;
    5'h13: out <= 8'h74;
    5'h14: out <= 8'h3D;
    5'h15: out <= {4'h0,hex5};
    5'h16: out <= {4'h0,hex4};
    5'h17: out <= 8'h54;
    5'h18: out <= 8'h65;
    5'h19: out <= 8'h6D;
    5'h1A: out <= 8'h70;
    5'h1B: out <= 8'h3D;
    5'h1C: out <= {4'h0,hex7};
    5'h1D: out <= {4'h0,hex6};
    default: out <= 8'h20;
     endcase
endmodule

`endif



module    Reset_Delay(iCLK,oRESET);
input        iCLK;
output reg     oRESET;
reg     [19:0]     Cont;

always@(posedge iCLK)
begin
    if(Cont!=20'hFFFFF)
    begin
        Cont     <=     Cont+1'b1;
        oRESET     <=     1'b0;
    end
    else
    oRESET     <=     1'b1;
```

```verilog
end

endmodule

```