DIFFERENTIAL POWER PROCESSING FOR MISMATCHED THERMOELECTRIC GENERATORS IN PARALLEL

BY

YUJIA ZHANG

THESIS

University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Robert Pilawa-Podgurski
ABSTRACT

This paper presents a differential power processing architecture to increase the energy capture of parallel-connected thermoelectric generators (TEGs) which have mismatched I-V characteristics. The design employs Ćuk converters regulating individual branch current to achieve maximum power output at each TEG. Each converter handles only the differential power between neighboring branches, targeting minimum power loss. In this research, a test-bed is designed to evaluate the I-V characteristics of TEG modules under non-uniform temperature gradients. The power outputs are compared between the proposed system and a directly-parallel-connected TEG system. A hardware prototype was built to validate the performance of the system.

Subject Keywords: Thermoelectricity; Differential Power Processing; Ćuk Converter; Energy Harvesting
ACKNOWLEDGMENTS

I would like to express my special appreciation and thanks to my adviser, Professor Pilawa, for his encouragements, guidance and for giving me the opportunity to do the exploration. Working in the power electronics lab has been an enjoyable learning experience for me. My gratitude also goes to Enver Candan, Yutian Lei, Shibin Qin and Rodrigo Serna. Their knowledge and research experience helped get me up to speed on the project. I would also thank my team member Corinne Cannavale for working with me on the characterization of TEG modules. Finally, a special thanks to Jeff Wheeler, who provided support when I encountered difficult stages and shared my excitement when I reached every breakthrough in the project.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER NUMBER</th>
<th>CHAPTER TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>TEG WITH NON-UNIFORM TEMPERATURE GRADIENT CHARACTERIZATION</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Experimental Setup</td>
<td>2</td>
</tr>
<tr>
<td>2.2</td>
<td>Characterization Results</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>PROPOSED ARCHITECTURE</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Mismatch Issue</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Differential Power Processing Architecture</td>
<td>7</td>
</tr>
<tr>
<td>3.3</td>
<td>Čuk Converter</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>SIMULATION ANALYSIS</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>HARDWARE VERIFICATION</td>
<td>15</td>
</tr>
<tr>
<td>5.1</td>
<td>Circuit Schematic</td>
<td>15</td>
</tr>
<tr>
<td>5.2</td>
<td>Test Results</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>CONCLUSION</td>
<td>20</td>
</tr>
<tr>
<td>A</td>
<td>TEG IV SWEEP CODE</td>
<td>21</td>
</tr>
<tr>
<td>A.1</td>
<td>Keithly Class</td>
<td>21</td>
</tr>
<tr>
<td>A.2</td>
<td>Source Meter IV sweep</td>
<td>24</td>
</tr>
<tr>
<td>A.3</td>
<td>C2000 Microcontroller PWM signals generation</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>32</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

As primary energy sources are limited by their finite supply and pollution generation, some non-conventional energy sources such as thermoelectric generator (TEG) have gained increased attention. TEG uses Seebeck effect [1] to convert heat into electrical energy, in which the heat flow, generated by a temperature gradient across a conducting material, diffuses charge carriers and results in a voltage between the hot and cold region. TEG is a good candidate for providing power to various applications, such as wearable devices, electrical vehicles and space probes, as it has no moving parts and is reliable for long term usage.

However, TEG is also constrained by its small output power and high output resistance, which incurs the need to connect multiple thermoelectric generators in parallel to power applications that requires large input current. The IV characteristics of TEG output are similar to those of a Thevenin circuit [1], namely a resistor connected in series with a voltage source, whose value increases as the temperature gradient across the TEG rises. As in practice, when several TEGs are connected in parallel, it is common to have non-uniform temperature gradient distribution among the modules, which can induce I-V characteristics mismatch at generator outputs and prevent the system from operating at maximum power output.

This research project first examined the output power-voltage (P-V) characteristics of TEGs at non-uniform temperature gradient distribution environment; then developed a differential power processing (DPP) architecture using Čuk converters to achieve maximum power output for multiple parallel-connected TEGs. The analytical results for the proposed system is presented and compared to the conventional parallel-connected TEG system. A hardware prototype was built to test with one DPP and two Thevenin circuit branches.
CHAPTER 2

TEG WITH NON-UNIFORM TEMPERATURE GRADIENT CHARACTERIZATION

2.1 Experimental Setup

A test-bed is designed to evaluate the TEG P-V performance under different temperature gradients. As shown in Figure 2.1, an aluminum plate is used to host at maximum 9 TEGs in three rows and columns. Six power resistors are mounted on one end of the aluminum board to act as heating elements, whereas two CPU fans are installed on the other end as cooling devices. Therefore, a horizontal temperature difference is established when the heat generated by resistors flows to the CPU fan’s direction. Figure 2.2. Is a thermal picture captured when both resistors and CPU fans are powered. The area near heating elements has the highest temperature, and the temperature of the aluminum plate decreases in the direction toward the fans.

Heat sink is mounted on each slot that hosts TEG. (Figure 2.3) When TEGs are inserted between the heat sink and aluminum board, each TEG sees a temperature difference across its top and bottom sides; the TEG that is placed close to the heating elements has larger temperature gradient than the TEG that is set close to the cooling elements.

The selected TEG is a Peltier module with specifications shown in Table 2.1.

Table 2.1: TEG Specification (reproduced from [2])

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Dimension</th>
<th>Maximum Hot Side Plate Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP85438</td>
<td>40mm x 40mm</td>
<td>90 °C</td>
</tr>
</tbody>
</table>

As demonstrated in Figure 2.4 is a characterization test setup. The FG 501A function generator (#1) on top outputs PWM signal that controls the speed of the CPU fans. The DC power supply on bottom right (#4) powers the heating resistors on the test-bed. The Agilent 34970A Data Acquisi-
Figure 2.1: Test-bed Block Diagram

Figure 2.2: Thermal Picture of TEG Test-bed
Figure 2.3: TEG Slot

Figure 2.4: Characterization Setup
tion Unit (#2) is equipped with thermocouples to measure the temperature across both sides of each TEG. Once the thermal equilibrium is reached, the Keithley 2420 source meter (#3) will do an I-V sweep across a designated TEG: the source meter is connected in series with the TEG and acts as a load; it steps the load and records the current and voltage values at each point until zero current is reached. The data documented by the source meter is transferred to computer via GPIB. A python program was written to control the IV sweep of the source meter and plots the obtained power and voltage points in graph (Appendix A).

2.2 Characterization Results

An example characterization consists of three TEGs placed in a column is presented as Figure 2.5. Table 2.2 denotes the temperature gradient, open circuit voltage, and maximum power for each TEG. The open circuit voltage is the potential difference measured across the TEG terminals when the output current is zero. The output power is computed as

\[ P = I \times V \] (2.1)

<table>
<thead>
<tr>
<th>TEG</th>
<th>Temperature Gradient ( T_d ) (°C)</th>
<th>Open Circuit Voltage ( V_S ) (V)</th>
<th>Maximum Power ( P_M ) (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42</td>
<td>0.82</td>
<td>85</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>0.2</td>
<td>5.5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0.06</td>
<td>0.49</td>
</tr>
</tbody>
</table>

As shown in Table 2.2, higher temperature gradient induces larger open circuit voltage and peak output power.

Figure 2.6 is the output P-V curve for TEG1, which is closest to the heating element. Same as the output P-V relationship for Thevenin circuit, the curves shape is a parabola, and the maximum power is reached at half of the open circuit voltage point.
Figure 2.5: TEG in Test

Figure 2.6: P-V Curve of TEG
CHAPTER 3
PROPOSED ARCHITECTURE

3.1 Mismatch Issue

When connecting TEGs with different P-V characteristics in parallel, based on superposition, the output voltage is not half of the open circuit voltage for all branches. To illustrate this issue, an example system is set up as Figure 3.1: A voltage source and a series resistor is used to represent the TEG model; each branch voltage source has a different value. Figure 3.2 plots the P-V curves for each TEG. As the peak of each curve is not aligned horizontally, it is impossible to extract the maximum power output from all three branches simultaneously.

3.2 Differential Power Processing Architecture

In order to solve the issue presented in Section 3.1 and to achieve the maximum power output for parallel connected TEGs, a differential power processing system is proposed as Figure 3.3. The architecture is developed by

![Figure 3.1: TEG in Parallel](image)
This system consists of five TEGs as an example network, but is also scalable to higher or lower number of TEGs. Instead of linking TEGs directly in parallel, the architecture first connects a TEG in series with DC/DC converters and then arrange the resulted branches into parallel connections. In general, a system consists of N TEGs will need (N-1) converters. Each TEG on the middle branches is connected in series with two DC/DC converters; the TEGs on the outside branches have only one converter in series.

The DC/DC converters act as voltage compensation modules for TEGs. They add positive or negative voltages to its branch to make the voltage across TEG is half of its own open circuit voltage. Assume the voltage at output is $V_O$, the voltage across each TEG is $V_{TEG}$, and the voltage across
the converter is $V_{conv}$. Based on KVL, the three voltage values will obey the following equation:

$$V_{TEG} + V_{CONV} = V_O$$  \hspace{1cm} (3.1)

Adjusting the duty ratio of each DC/DC converter will change the current flow through the converter as well as the voltage across the converter. Let $I_M$ be the current, at which a TEG outputs the maximum power. For each TEG, $V_S$ is the open circuit voltage and R is output resistance. $I_M$ can be expressed as following

$$I_M = \frac{V_S}{2R}$$  \hspace{1cm} (3.2)

For TEGs that have static open circuit voltage and output resistance, the value of $I_M$ is fixed, thus when they are connected in the DPP system, the duty ratios for each converter is also fixed and related to each $I_M$.\cite{5}

When the system is operating at maximum power output, due to the voltage compensation from the converters, the voltage across each TEG is half of its open circuit voltage. Thus each TEG is producing the maximum output power. Since each converter only sees a differential voltage between the TEG and the output, it processes only a fractional power. Thus minimizing the fundamental circuitry power loss.

### 3.3 Čuk Converter

The DC/DC converter used in this architecture is Čuk converter (Figure 3.4), which is the dual of the buck-boost converter used in \cite{6}. A Čuk converter consists of one capacitor, two inductors and two switches. The two switches in the system are operated complementary to each other. By turning on and off the switches, the capacitor connects alternatively to the input and output, thus transfers energy. \cite{7}

The current through the input, $I_{in}$ and output, $I_{out}$ as denoted in Figure 3.1 follows the equation (3.3), where D is the duty ratio of the PWM switching signal

$$\frac{I_{OUT}}{I_{IN}} = \frac{1 - D}{D}$$  \hspace{1cm} (3.3)
In the proposed architecture, since the converter input or output terminals are connected in series with the TEG, the input and output currents corresponds to the TEG output currents.
CHAPTER 4

SIMULATION ANALYSIS

The proposed DPP system is verified in LTspice with three TEGs. Thevenin circuit is used to model TEG and assuming each branch has the same series resistance 5 Ohm to simplify the analysis. The open circuit voltages are listed as in Table 4.1.

Table 4.1: TEG Open Circuit Voltages for Analysis

<table>
<thead>
<tr>
<th>TEG</th>
<th>Open Circuit Voltage, $V_S$ (V)</th>
<th>Current at max power, $I_M$ (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>2.1</td>
</tr>
</tbody>
</table>

The open circuit voltages are higher than characterized result for single modules. These were chosen for the convenience of practical converter design. In practice, the voltages can be achieved by stacking TEG modules in series. The value of $I_M$ is obtained by equation (3.2).

Theoretically, the maximum power output can be obtained by adding the maximum value of each TEG branch, which is calculated as following

$$P_{\text{max}} = P_{1,\text{max}} + P_{2,\text{max}} + P_{3,\text{max}} = \frac{1}{2} \times \left( \frac{3^2}{10} + \frac{7^2}{10} + \frac{21^2}{10} \right) = 24.95\text{W} \quad (4.1)$$

Two systems are simulated in LTspice: one is the TEG with DPP which consists of three TEGs and two Ćuk converters; the other is TEG only, which connects three TEGs in parallel directly. The two systems have the same TEG characteristics. The I-V sweep is done by stepping the load resistance connected in parallel with the system from 0.1 to 5 Ohms.

For the DPP system, the circuit schematic is shown in Figure 4.1, the duty ratio for the two converters are obtained by applying equation (4.2) and (4.3):
Figure 4.1: Hardware Test Circuit Schematic
\[ D_1 = \frac{1}{1 + \frac{I_{M.2}}{I_{M.1}}} = \frac{1}{1 + \frac{0.7}{0.3}} = 0.3 \]  
\[ D_2 = \frac{1}{1 + \frac{I_{M.3}}{I_{M.2}}} = \frac{1}{1 + \frac{2.1}{0.7}} = 0.25 \]  

The \( P_{out} \) vs. \( R \) is plotted for both configurations. Figure 4.2 shows the result for non-DPP and Figure 4.3 shows the result for DPP. The maximum output power for system with DPP is able to reach 23.9W, which is close to the theoretical prediction; however, the peak value for system without DPP is 16W (Figure 4.2). In this case, the peak output power of system with DPP is 33.1% higher than that of non-DPP system.

It can also be observed from the curves that not only at peak output power point, but also other load resistances, the output power of DPP is higher in general.

<table>
<thead>
<tr>
<th>Converter</th>
<th>Duty Ratio</th>
<th>Frequency (kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 (S1, S2)</td>
<td>0.3</td>
<td>100</td>
</tr>
<tr>
<td>#2 (S3, S4)</td>
<td>0.25</td>
<td>100</td>
</tr>
</tbody>
</table>
Figure 4.2: $P_{\text{out}}$ vs. $R$, no DPP

Figure 4.3: $P_{\text{out}}$ vs. $R$, DPP
CHAPTER 5

HARDWARE VERIFICATION

5.1 Circuit Schematic

The circuit schematic is shown as following. The two switches in the Ćuk were realized by N type MOSFETs. A TI C2000 microcontroller was programmed to output a pair of pulse width modulated signals (PWM), in which the two signals are complementary to each other. The PWM signals are feed into gate drivers in order to control the turn-on and off of the MOSFETs. As shown in the schematic below, M1 is connected in opposite direction to M2. For M1, its source is connected to the converter, and drain is connected to ground. Thus in order to drive M1, it requires isolated power and PWM signals, which is provided by Iso-power chip ADUM5201. The details of components are shown as in Table 5.1.

5.2 Test Results

As shown in Figure 5.2, the Ćuk converter is integrated with two Thevenein branches: power supply with series resistor.

Before measuring the output power capture of the system, a validation of

<table>
<thead>
<tr>
<th>Component</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI3410DV NMOS(M1, M2)</td>
<td>Switches</td>
</tr>
<tr>
<td>TI C2000 LaunchPad</td>
<td>Generates PWM signals</td>
</tr>
<tr>
<td>ADUM 5201</td>
<td>Isolates power and PWM signals</td>
</tr>
<tr>
<td>LTC 1693</td>
<td>Gate driver</td>
</tr>
<tr>
<td>Capacitor (C1)</td>
<td>100uF</td>
</tr>
<tr>
<td>Inductors (L1, L2)</td>
<td>68uH</td>
</tr>
</tbody>
</table>
the functionality of the Ćuk converter was performed. Table 5.2 shows the results.

Table 5.2: Ćuk Converter Validation

<table>
<thead>
<tr>
<th>$V_1$ (V)</th>
<th>$V_2$ (V)</th>
<th>Set Duty Ratio</th>
<th>Frequency (kHz)</th>
<th>$I_1$ (A)</th>
<th>$I_2$ (A)</th>
<th>Experimental Duty Ratio</th>
<th>Duty Ratio Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3V</td>
<td>4.5V</td>
<td>40%</td>
<td>100</td>
<td>0.21</td>
<td>0.29</td>
<td>42%</td>
<td>2%</td>
</tr>
</tbody>
</table>

The experimental duty ratio was calculated from the currents flowing through the power supplies, $I_1$ and $I_2$:

$$\text{Experimental Duty Ratio} = \frac{1}{1 + \frac{I_2}{I_1}} = \frac{1}{1 + \frac{0.29}{0.21}} = 42\% \quad (5.1)$$

Figure 5.3 is the switching nodes waveform for the Ćuk converter. Channel 1 is the PWM1 signal generated by the C2000 microcontroller. Channel 2 and Channel 3 are the voltages at the two terminals of the capacitor: Channel 2 is SW_Drain1 and Channel 3 is SW_Drain2 as shown in the Figure 5.3.

After validating the proper conversion function, the following set-up was tested to compare the output power capture between Thevenin circuits in-
tegrated with differential power processing and Thevenin circuits connected directly in parallel.

**Table 5.3: TEG Hardware Test Setup**

<table>
<thead>
<tr>
<th>$V_1$ (V)</th>
<th>$V_2$ (V)</th>
<th>$R_1$ (Ohm)</th>
<th>$R_2$ (Ohm)</th>
<th>$R_{LOAD}$ (Ohm)</th>
<th>Duty Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1V</td>
<td>7V</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>12.5%</td>
</tr>
</tbody>
</table>

The voltage across the load resistor, $V_{LOAD}$, was measured to calculate the output power. The first test was connecting the two Thevenin circuit branches in parallel and measuring the output power. The second was to connect the two Thevenin circuit branches with differential power converter and then connected to the load resistor. The two output power values are compared to evaluate the differential power processing system performance. The results are shown in Table 5.4.

**Table 5.4: TEG Hardware Test Setup**

<table>
<thead>
<tr>
<th>$V_{LOAD}$ no DPP (V)</th>
<th>$V_{LOAD}$ DPP (V)</th>
<th>$P_{LOAD}$ no DPP (W)</th>
<th>$P_{LOAD}$ DPP (W)</th>
<th>$P_{LOAD}$ no DPP, theoretical (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8425V</td>
<td>1.9368V</td>
<td>1.13V</td>
<td>1.25V</td>
<td>1.142</td>
</tr>
</tbody>
</table>
The output power are calculated as

\[ P_{load,\,no\,DPP} = \frac{V_{load,\,no\,DPP}^2}{R_{Load}} = \frac{V_{1.8425}^2}{3} = 1.13W \]  \hspace{1cm} (5.2)

\[ P_{load,\,DPP} = \frac{V_{load,\,DPP}^2}{R_{Load}} = \frac{V_{1.9368}^2}{3} = 1.25W \]  \hspace{1cm} (5.3)

The theoretical maximum output power of two Thevenin circuits connected in parallel without differential power processing was obtained through simulation in LTspice. The simulation method is similar as described in Chapter 4 and the waveform is shown in Figure 5.4.

As shown in Table 5.4, the power extracted from the system with differential power processing is higher than the theoretical maximum output power of the system without DPP by 9.45%. This increase is also validated experimentally. With the same load resistance, the output power with DPP is higher than the one without DPP by 10.6%.
CHAPTER 6

CONCLUSION

This research project explored the possibility of connecting mismatched thermal electric generators in parallel, while achieving the maximum power output. The proposed differential power processing system employs Čuk converters to compensate the voltages for series-connected TEGs, in order to operate each TEG at its own maximum power point. The system is tested in simulation and the analytical results verified its functionality. In the hardware prototype, the system with DPP captures more output power than the system without DPP.

Future work includes expanding the hardware prototype from using one DPP to multiple ones, so that the system can work with more than two Thevenin circuit branches. Currently the circuit was power supplies with series resistors, and it is also beneficial to integrate the system with TEG modules and evaluate its performance with applications.
import serial
from time import sleep
import smtplib

class Keithley2420:
    LINETERM = '
    #set the sourcemeter parameters (all commands can be looked up in detail in the SCPI Programming Reference.chm file)
    
    initcmds = [
        #Restore GPIB defaults
        "*RST",
        #Everything in memory is reinitialized
        "SYSTem:MEMory:INITialize",
        #Clear Status; Clears all event registers
        ____________________#and Error Queue
        "*CLS",
        #Enables or disables auto-zero; Disabling increases
        ____________________#speed/loses accuracy
        "SYSTem:AZEr:STAt:ON",
        #Sets sweep integration speed; When set for one,
        ____________________#sets for voltage, current, and
        resistance
        "SENSe:VOLTage:NPLCycles_MIN",
        #Might be redundant
        "SENSe:CURRent:NPLCycles_MIN",
        #Enables measuring multiple functions at once
        "SENSe:FUNCtion:CONCurrent:ON",
        #Enables current to be measured
"SENSe:FUNCtion:"CURRent"," 
#Start voltage to sweep at 
"SOURce:VOLTage:STARt\_0",
#Stop voltage to sweep to 
"SOURce:VOLTage:STOP\_5",
#Sets the sourcing mode as sweep; 
#Can be FIXed or LIST 
"SOURce:VOLTage:MODE\_SWEep",
#Specify number of source–measure points 
"SOURce:SWEep:POINts\_100",
#Select scale for sweep LINear or LOGarithmic 
"SOURce:SWEep:SPACing\_LINear",
#Set current compliance "TIMEOUT HERE" 
"SENSe:CURRent:PROTection 3",
"SENSe:VOLTage:PROTection 9",
"SENSe:CURRent:PROTection 3",
#Sets auto range to off so that manual range is 
#Automatically turned off when manual range is set (Redundant) 
#Sets auto range to off so that manual range is 
#Automatically turned off when manual range is set (Redundant) 
"SENSe:VOLTage:RANGe\:AUTO ON",
#Sets current range to look at (a bit confused here) 
#Disables digital averaging 
"SENSe:AVerage\_OFF",
#auto output–off disables auto output–off (who named this) 
#source output must be on before INIT or READ can be used 
"SOURce:CLEar\:AUTO\_OFF",
#Manually set source delay so signal can settle 
#before measurement. In seconds. 
"SOURce:DELay\_MIN",
#Setting no delay for Device Action; In seconds. 
"TRIGger:DELay\_0",
#Specifies number of time an operation is performed
#(Should be same as points per sweep)
"TRIGger:COUNt_100",
#Specifies the number of times it should sweep

TRIG:COUN should not exceed 2500
"ARM:COUNt_1",
#Sets the display resolution to 3.5 digits
"DISPlay:DIGits_MIN",
#Sets elements to be returned as ASCII string (See figure 18–1)
"FORMat:ELEMents_VOLT, CURR",
#Sets the data format as ASCII, <length> not used for ASCII
"FORMat:DATA:ASCII",
#Turns source off or on
"OUTPut_ON"

#creates a socket connection between the meter and computer

def __init__(self, port="/dev/ttyUSB0", baudrate=57600, 
               timeout=1, writeTimeout=1):
    self.pms = serial.Serial()
    self.pms.port = port
    self.pms.baudrate = baudrate
    self.pms.timeout = timeout
    self.pms.writeTimeout = writeTimeout
    self.pms.open()

    if self.pms.isOpen():
        #self.idn = self.query("*IDN?")
        #print self.idn
        self.initialize()
        #self.write("*RST")
        #print self.pms.portstr

    #initializes the meter by writing to it the initial SCPI commands listed above
    def initialize(self):
        self.writelist(self.initcmds)

    def query(self, querystr, recvsize=1024):
        self.write(querystr)
        return self.read(recvsize=recvsize)
# reads and returns the data stored in the memory of the meter

def read(self, recvsize=1024, raw=False):
    while True:
        try:
            data = self.pms.read(recvsize)
            break
        if not raw: data = data.strip()
    return data

# writes to the meter any additional SCPI commands

def write(self, cmdstring, raw=False):
    self.pms.write(cmdstring)
    sleep(0.1)
    if not raw: self.pms.write(self.LINETERM)

def writelist(self, cmdlist):
    for c in cmdlist:
        self.write(c)

# closes the socket connection

    #(after communication with the meter is no longer needed)

    def close(self):
        self.query("\*RST")
        self.pms.close()
# Initialize timestamp for folder name in form of yyyyymmdd
date = time.strftime('%Y%m%d')
# Initialize timestamp for file names in form of hhmmss
clock = time.strftime('%H%M%S')

bigTime = time.time()
Vexp = []
Iexp = []
Pexp = []

def mainInit():
    global i, date
    # Create foldername for os.path use
    foldername = './%s' %date
    
    # Check if todays folder exists, if not create a new one
    # If it does, count the number of files and return set updated i
    if not os.path.exists(foldername):
        # Create directory
        os.makedirs(foldername)
    else:
        # Count number of files, divide by two (one file for each meter),
        # and store it into i
        i = (len(glob.glob('./%s/*' %date)) / 2)

def keithleyFunc():
    global i, date, bigTime, Vexp, Iexp, Pexp

    meter = keithleyClass.Keithley2420()

    n = 0
    # Vmpp = 0

    keithleyFile = './%s/keithley_%d' %(date, i)
    readKeithleyF = open(keithleyFile, 'w')
    Vexp = []
    Iexp = []
Pexp = []
meter.write("READ?")
meter.write("++read")

#while(0):
while n < 100:
    value = meter.read(28)
    #print sys.getsizeof(value)
    #print value
    readKeithleyF.write(value)
    try:
        Vexp.append(float(re.split(',', value)[0]))
        Iexp.append(-1 * float(re.split(',', value)[1]))
    except:
        print re.split(',', value)[0]
        print re.split(',', value)[1]
    while (0):
        while sys.getsizeof(value) == 21:
            value = meter.read(21)
            readKeithleyF.write(value)
            try:
                Vexp.append(float(re.split(',', value)[0]))
                Iexp.append(-1 * float(re.split(',', value)[1]))
            except:
                print re.split(',', value)[0]
                print re.split(',', value)[1]
            readKeithleyF.write("" + str(time.time()) + "\n")
            Pexp.append(Vexp[n]*Iexp[n])
            readKeithleyF.write(str(Pexp[n]) + "\n")
            n=n+1
            #break
        i = i + 1
    #while(0):
    readKeithleyF.close()
    meter.close()
    plot_PV()
# plot the output data

def plot_PV():
    # global Vexp, Pexp, Iexp
    font = {'family': 'Serif',
            'color': 'darkred',
            'weight': 'normal',
            'size': 16,
            }
    plot(Vexp, Pexp)
    plt.title('P vs. V', fontdict=font)
    plt.xlabel('Voltage (V)', fontdict=font)
    plt.ylabel('Power (W)', fontdict=font)
    show()

if __name__ == "__main__":
    mainInit()
    keithleyFunc()

A.3 C2000 Microcontroller PWM signals generation

[8]

// This code generates a pair of complementary PWM signals with
dead time. The code was modified from example code provided
by Texas Instruments [5]

#include "DSP28x_Project.h"       // Device Headerfile and
                               // Examples Include File
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/pwm.h"
#include "f2802x_common/include/wdog.h"

// Prototype statements for functions found within this file.
void InitEPwm1(void);

// Global variables used in this example
uint32_t EPwm1TimerIntCount;
uint16_t EPwm1_DB(Direction;
#define EPWM1_MAX_DB 0x03FF

#define EPWM1_MIN_DB 8

CLK_Handle myClk;
FLASH_Handle myFlash;
GPIO_Handle myGpio;
PIE_Handle myPie;
PWM_Handle myPwm1

void main(void)
{
    CPU_Handle myCpu;
PPL_Handle myPll;
WDOG_Handle myWDog;
    // Initialize all the handles needed for this application
    myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
    myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
    myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(
        FLASH_Obj));
    myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj)) ;
    myPie = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj));
    myPll = PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));
    myPwm1 = PWM_init((void *)PWM_ePWM1_BASE_ADDR, sizeof(
        PWM_Obj));
    myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj)) ;

    // Perform basic system initialization
    WDOG_disable(myWDog);
    CLK_enableAdcClock(myClk);
    (*Device_cal)();
    CLK_disableAdcClock(myClk);

    // Select the internal oscillator 1 as the clock source
    CLK_setOscSrc(myClk, CLK_OscSrc_Internal);

    // Setup the PLL for x12 /2 which will yield 60Mhz = 10Mhz * 12 / 2
    PLL_setup(myPll, PLL_Multiplier_12,
        PLL_DivideSelect_CLKIn_by_2);
// Disable the PIE and all interrupts
PIE_disable(myPie);
PPI_disableAllInts(myPie);
CPU_disableGlobalInts(myCpu);
CPU_clearIntFlags(myCpu);

// If running from flash copy RAM only functions to RAM
#ifdef FLASH
memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
#endif

// Initialize GPIO
GPIO_setPullUp(myGpio, GPIO_Number_0, GPIO_PullUp_Disable);
GPIO_setPullUp(myGpio, GPIO_Number_1, GPIO_PullUp_Disable);
GPIO_setMode(myGpio, GPIO_Number_0, GPIO_0_Mode_EPWM1A);
GPIO_setMode(myGpio, GPIO_Number_1, GPIO_1_Mode_EPWM1B);

// Setup a debug vector table and enable the PIE
PIE_setDebugIntVectorTable(myPie);
PIE_enable(myPie);

// Register interrupt handlers in the PIE vector table
// PIE_registerPieIntHandler(myPie, PIE_GroupNumber_3,
// PIE_SubGroupNumber_1, (int Vect_t)&epwm1_isr);
CLK_disableTbClockSync(myClk);

InitEPwm1();
CLK_enableTbClockSync(myClk);

// Initialize counters:
EPwm1TimerIntCount = 0;
// Enable CPU INT3 which is connected to EPWM-3 INT
CPU_enableInt(myCpu, CPU_IntNumber_3);

// Enable EPWM INTn in the PIE: Group 3 interrupt 1
PIE_enablePwmInt(myPie, PWM_Number_1);

// Enable global Interrupts and higher priority real-time debug events
CPU_enableGlobalInts(myCpu);
CPU_enableDebugInt(myCpu);

for (;;) {
  asm(".NOP");
}
```c
void InitEPwm1()
{
    CLK_enablePwmClock(myClk, PWM_Number_1);
    PWM_setPeriod(myPwm1, 300);  // Set timer period
    PWM_setPhase(myPwm1, 0x0000); // Phase is 0
    PWM_setCount(myPwm1, 0x0000); // Clear counter

    // Setup TBCLK
    PWM_setCounterMode(myPwm1, PWM_CounterMode_UpDown); // Count up
    PWM_disableCounterLoad(myPwm1);   // Disable phase loading
    PWM_setHighSpeedClkDiv(myPwm1, PWM_HspClkDiv_by_1); // Clock ratio to SYSCLKOUT
    PWM_setClkDiv(myPwm1, PWM_ClkDiv_by_1);

    // Load registers every ZERO
    // PWM_setShadowMode_CmpA(myPwm1, PWM_ShadowMode_Shadow);  //
    // PWM_setShadowMode_CmpB(myPwm1, PWM_ShadowMode_Shadow);
    // PWM_setLoadMode_CmpA(myPwm1, 50);
    // PWM_setLoadMode_CmpB(myPwm1, 100);
    PWM_setLoadMode_CmpA(myPwm1, PWM_LoadMode_Zero);
    PWM_setLoadMode_CmpB(myPwm1, PWM_LoadMode_Zero);

    // Setup compare
    PWM_setCmpA(myPwm1, 120);
    // PWM_setCmpB(myPwm1, 100);

    // Set actions
    PWM_setActionQual_CntUp_CmpA_PwmA(myPwm1, PWM_ActionQual_Set);
    PWM_setActionQual_CntDown_CmpA_PwmA(myPwm1, PWM_ActionQual_Clear);
    PWM_setActionQual_CntUp_CmpA_PwmB(myPwm1, PWM_ActionQual_Clear);
    PWM_setActionQual_CntDown_CmpA_PwmB(myPwm1, PWM_ActionQual_Set);

    // Active Low PWMs – Setup Deadband
    //PWM_setDeadBandOutputMode(myPwm1, PWM_DeadBandOutputMode_EPWMxA_Rising_EPWMxB_Falling);
    PWM_setDeadBandOutputMode(myPwm1, PWM_DeadBandOutputMode_AHC);
    PWM_setDeadBandPolarity(myPwm1, PWM_DeadBandPolarity_EPWMxB_Inverted);
    // PWM_setDeadBandPolarity(myPwm1,
```

PWM_DeadBandPolarity_EPWMxA_Inverted_EPWMxB_Inverted);
127 PWM_setDeadBandInputMode(myPwm1,
PWM_DeadBandInputMode_EPWMxA_Rising_and_Falling);
128 PWM_setDeadBandRisingEdgeDelay(myPwm1, EPWM1_MIN_DB);
129 PWM_setDeadBandFallingEdgeDelay(myPwm1, EPWM1_MIN_DB);
130 EPwm1_DB_Direction = DB_UP;
131 // Interrupt where we will change the Deadband
132 // PWM_setIntMode(myPwm1, PWM_IntMode_CounterEqualZero); //
133 // Select INT on Zero event
134 // PWM_enableInt(myPwm1); //
135 // Enable INT
136 // PWM_setIntPeriod(myPwm1, PWM_IntPeriod_ThirdEvent); //
137 // Generate INT on 3rd event
138 }
REFERENCES


