A. Appendix

A1 Thermistor datasheet

5mm EPOXY COATED DISC TYPE
THERMISTORS FOR TEMPERATURE
COMPENSATION,MEASUREMENT AND CONTROL

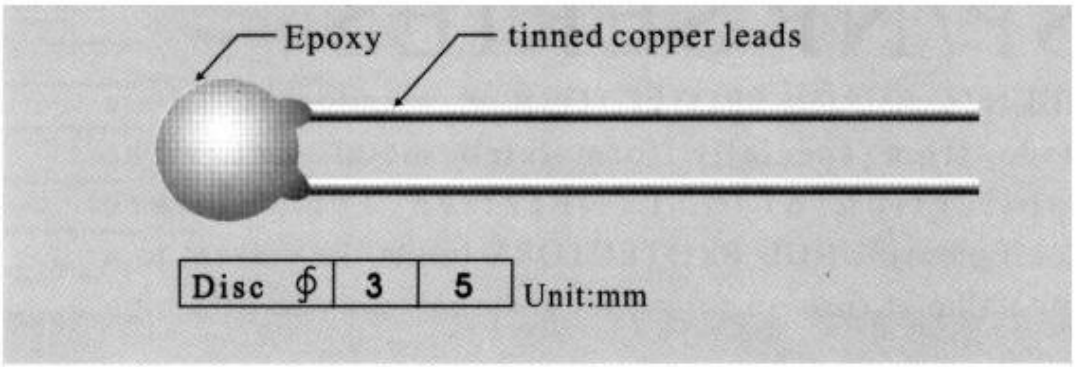
RATING

Operating Temperature Range:-20℃~+125℃
Maximum Power Rating:500mW

SPECIFICATIONS

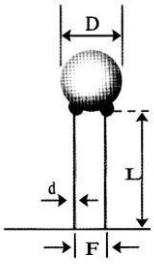
5mm Series

Part No.	Nominal Resistance at 25℃ (ohms)	Beta Value (◦K)	Max. permissible Current at 25℃ (mA)	Thermal Dissipation Constant (mW/℃)	Thermal Time Constant (sec)
RN3430	15	2900	250	6	18
RN3432	100	3100	200	6	19
RN3434	500	3800	100	6	16
RN3436	1,000	3700	80	6	17
RN3438	4,700	4100	45	6	22
RN3440	10,000	4100	30	6	15
RN3442	47,000	4400	20	6	18
RN3444	100,000	4400	15	6	16



Resistance Tolerance: K=±10%

DIMENSION SPECIFICATIONS



FORMING TYPE : WTYPE

FORMING SPECIFICATIONS

Type	Disc Dia.	Forming Type	d (±0.02)	D (max.)	T (max.)	F (±1.0)	L (min.)
NTC	5Φ	W2	0.5	6.5	4.0	3.5	25

A2 EEONTEX Heating Fabric

Eeonyx Product Information Sheet

CORPORATION Rev: 07/09
750 Belmont Way, Pinole, CA 94564 Tel (510)741-3632 Fax (510) 741-3657 E-mail info@eeonyx.com

EEONTEX™ CONDUCTIVE NONWOVEN FABRIC

Part of the EeonTex™ Product Line

EeonTex™ NW170-PI-15 is a microfiber nonwoven coated with a conductive-polymer formulation. This material is used in resistive heating applications.

Parameters of EeonTex™ conductive nonwoven

Part Number: **EeonTex™** NW170-PI-15
Filament blend: polyester/nylon 6 (70/30)
Web bonding: hydrolace
Mass per unit area: 170 g/m²
Thickness: 0.80 mm
Tensile Strength measured: > 450 N
Elongation at break: 40%
Tear Resistance measured: 12 N
Surface Resistivity: 15 ohm/sq +/- 10% (8 to 105 ohm/sq per customer specifications)

Advantages of heaters made of EeonTex™:

- Soft and pliable
- Energy efficient
- Uniform surface heating
- Tunable resistance
- Durable to abrasion & flexing
- Can be designed to any size & shape
- Low thermal mass
- Low watt density
- Completely safe and durable
- Cost effective & low labor manufacture

EeonTex™ nonwoven can be coated with a protective coating to enhance stability and to make it flameresistant and/or bactericidal.

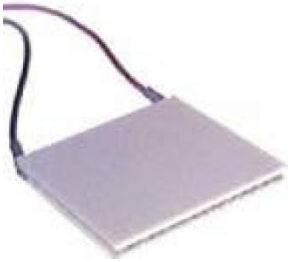
A3 Peltier Plater Datasheet

Thermoelectric Cooler

TEC1-12706

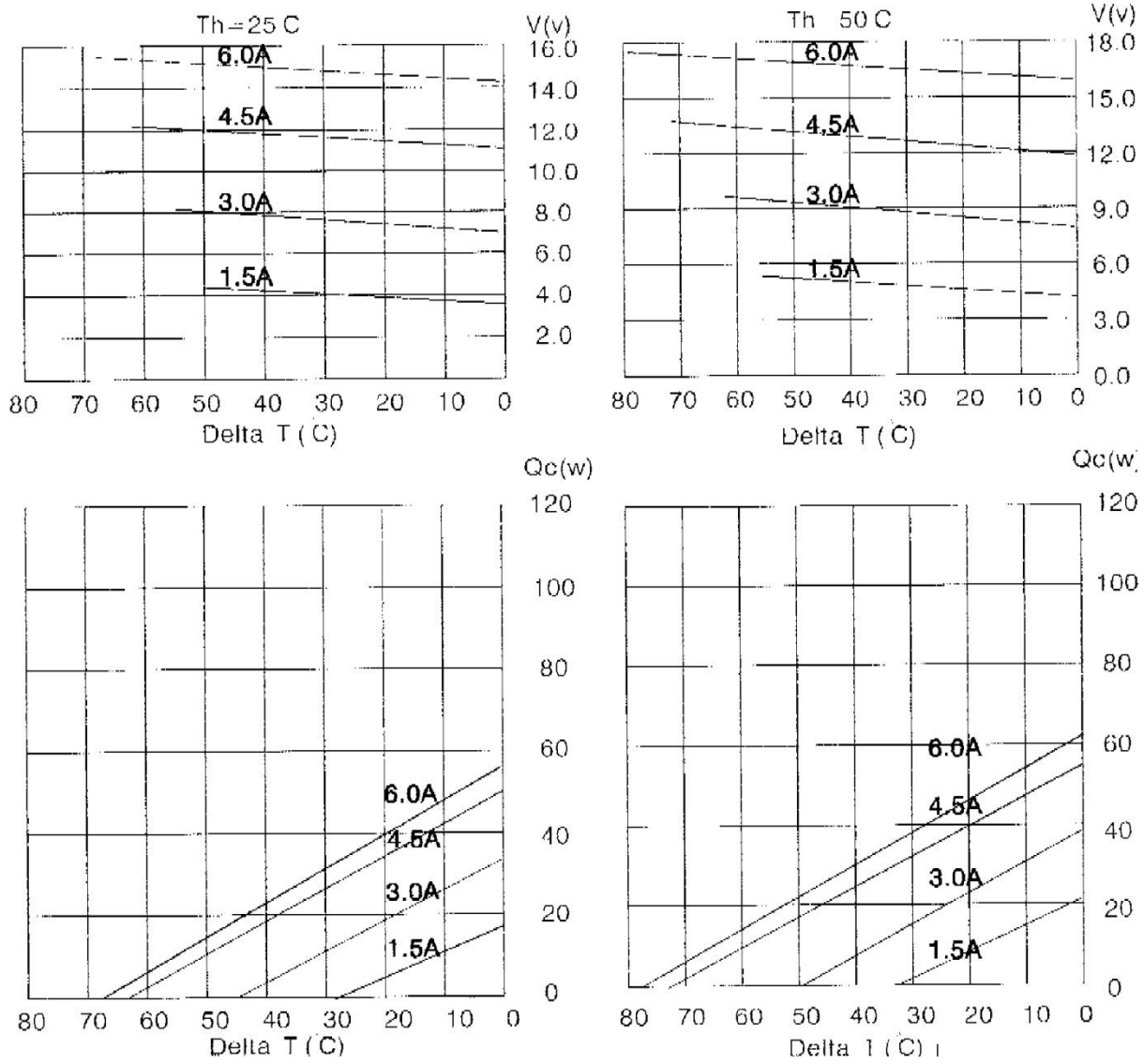
Performance Specifications

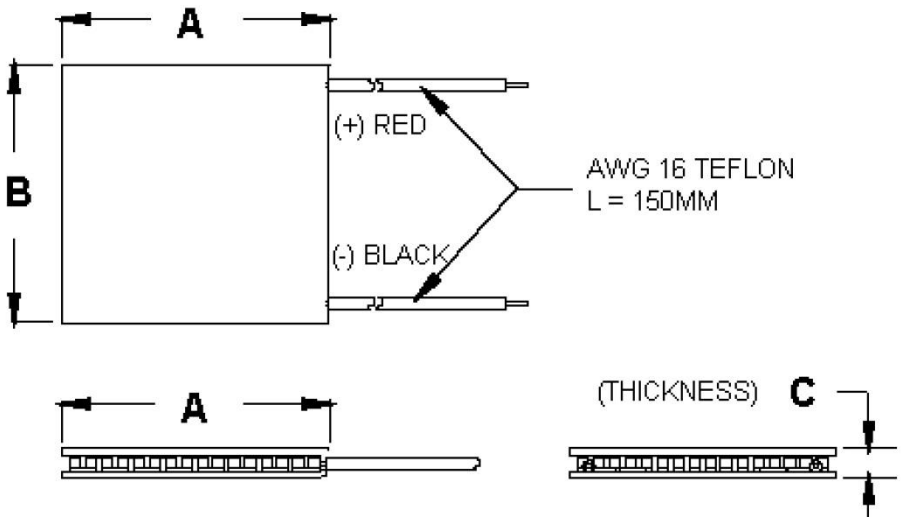
Hot Side Temperature (ºC)	25ºC	50ºC
Qmax (Watts)	50	57
Delta Tmax (ºC)	66	75
Imax (Amps)	6.4	6.4
Vmax (Volts)	14.4	16.4
Module Resistance (Ohms)	1.98	2.30



Performance curves on page 2

 Rev 2.03





Ceramic Material: Alumina (Al_2O_3)
Solder Construction: 138ºC, Bismuth Tin (BiSn)

Size table:

A	B	C			
40	40	3.8			

Operating Tips

- Max. Operating Temperature: $138^{o}C$
- Do not exceed Imax or Vmax when operating module.
- Life expectancy: 200,000 hours
-

- Please consult HB for moisture protection options (seeling).
- Failure rate based on long time testings: 0.2%.

 Rev 2.03

585460

LI-POLYMER BATTERY PACKS

Specification

Type:585460 2000mAh

Prepared/Date	Auditing/Date	Approved/Date
WANG MAR 16, 2006	LI MAR 16, 2006	XIONG MAR 16, 2006

Doc. No.	2006.3.16
Edition No.	2.0
Sheet	1/5

PRODUCT SPECIFICATION

1 Scope

This product specification describes UNIONFORTUNE polymer lithium-ion battery. Please using the test methods that recommend in this specification. If you have any opinions or advices about the test items and methods, please contact us. Please read the cautions recommended in the specifications first, take the credibility measure of the cell's using.

2 Product Type, Model and Dimension

- **Type Polymer lithium-ion battery**
- **Model 585460**

2.3 Cell Dimension(Max, Thickness×Width×Length mm^3) 5.8× 54× 60
Pack Dimension(Max, Thickness×Width×Length mm^3) None

3 Specification

Item		Specifications	Remark
Nominal Capacity		2000 mAh	0.2C_5A discharge
Nominal Voltage		3.7V	Average Voltage at 0.2C_5A discharge
Charge Current		Standard 0.2 C_5A Max 1C_5A	Working temperature 0 40
Charge cut-off Voltage		4.20±0.03V	
Standard Discharge Current		0.2C_5A	Working temperature -20 60
Max Discharge Current		2.0C_5A	Working temperature 0 60
Discharge cut-off Voltage		2.75 V	
Cell Voltage		3.7-3.9 V	When leave factory
Impedance		300 m	AC 1KHz after 50% charge
Weight		Approx: 37g	
Storage temperature	1month	-20 45	Best 20±5 for long-time storage
	3month	0 30	
	6month	20±5	
Storage humidity		65±20% RH	

4 General Performance

Definition of Standard charging method At 20±5 charging the cell initially with constant current 0.2C_5A till voltage 4.2V, then with constant voltage 4.2V till current declines to 0.05C_5A.

Item		Test Methods	Performance
4.1	0.2C Capacity	After standard charging, laying the battery 0.5h, then discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	300min
4.2	1C Capacity	After standard charging, laying the battery 0.5h, then discharging at 1C_5A to voltage 2.75V, recording the discharging time.	51min
4.3	Cycle Life	Constant current 1C_5A charge to 4.2V, then constant voltage charge to current declines to 0.05C_5A, stay 5min constant current 1C_5A discharge to 2.75V stay 5min. Repeat above steps till continuously discharging time less than 36min.	300times
4.4	Capability of keeping electricity	20±5 , After standard charging, laying the battery 28days, discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	240min

PRODUCT SPECIFICATION

5 Environment Performance

Item		Test Methods	Performance
5.1	High temperature	After standard charging, laying the battery 4h at 60 , then discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	270min
5.2	Low temperature	After standard charging, laying the battery 4h at 0.2C_5A, then discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	210min
5.3	Constant humidity and temperature	After standard charging, laying the battery 48h at 40±2 , RH 93±2%. Recording 0.2C_5A discharging time	No distortion No electrolytes leakage 270 min
5.4	Temperature shock	After standard charging, battery stored at -20 for 2 hours, then stored at 50 for 2 hours. Repeat 10 times.	No electrolytes leakage

6 Mechanical Performance

Item		Test Methods	Performance
6.1	Vibration	After standard charging, put battery on the vibration table. 30 min experiment from X,Y,Z axis. Scan rate: 1 oct/min; Frequency 10-30Hz, Swing 0.38mm; Frequency 30-55Hz, Swing 0.19mm.	No influence to batteries© electrical performance and appearance.
6.2	Collision	After vibration test, batteries were laying on the vibration table about X, Y, Z axis. Max frequency acceleration: 100m/s^2; collision times per minutes: 40~80; frequency keeping time 16ms; all collision times 1000±10.	No influence to batteries© electrical performance and appearance.
6.3	Drop	Random drop the battery from 10m height onto concrete one times.	No explosion or fire

7 Safety Test

Test conditions The following tests must be measured at flowing air and safety protection conditions. All batteries must standard charge and lay 24h.

Item		Test Methods	Performance
7.1	Over charge	At 20±5 , charging batteries with constant current 3C_5A to voltage 4.8V, then with constant voltage 4.8V till current decline to 0. Stop test till batteries' temperature 10 lower than max temperature.	No explosion or fire
7.2	Over discharge	At 20±5 , discharge battery with 0.2C_5A continuously 12.5h.	No explosion or fire
7.3	Short-circuit	At 20±5 , connect batteries' anode and cathode by wire which impedance less than 50m , keep 6h.	No explosion or fire
7.4	Extrusion	At 20±5 , put the battery in two parallel steal broad, add pressure 13kN.	No explosion or fire
7.5	Thermal shock	Put the battery in the oven. The temperature of the oven is to be raised at 5±1 per minute to a temperature of 130±2 and remains 60 minutes.	No explosion or fire

8 Cautions

1. **Cautions of batteries' operation**

The batteries must be careful of proceed the operation for it's soft package.

Aluminum packing materials

The aluminum packing material was easily damaged by the sharp edge part, such as nickel-tabs.

forbid to use the sharp part touching the battery;

should cleaning working condition, avoiding the sharp edge part existence; forbid to pierce the battery with nail and other sharp items;

the battery was forbidden with metal, such as necklace, hairpin etc in transportation and storage. Sealed edge

Sealing edge is very easily damaged and don't bend it.

The Al interlayer of package has good electric performance. It's forbidden to connect with exterior component for preventing short-circuits.

Folding edge

The folding edge is formed in batteries' processes and passed all hermetic tests, don't open or deform it. The Al interlayer of package has good electric performance. It's forbidden to connect with exterior component for preventing short-circuits.

Tabs

The batteries' tabs are not so stubborn especially for aluminum tabs. Don't bend tabs.

Mechanical shock

Don't fall, hit, bent the batteries' body.

Short-circuit

Short-circuit is strictly prohibited. It should damage batteries badly.

2. **Standard Test Environment for polymer lithium-ion batteries**

Environment temperature:
20±5 Humidity: 45-85%

3. **Cautions of charge & discharge**

charge

Charging current should be lower than values that recommend below. Higher current and voltage charging may cause damage to cell electrical, mechanical, safety performance and could lead heat generation or leakage.

Batteries charger should charging with constant current and constant voltage mode; Charging current should be lower than (or equal to)1C_5A;

Temperature 0 40 is preferred when charging;

Charging voltage must be lower than 4.25V.

discharge

Discharging current must be lower than (or equal to)2C_5A;

Temperature 0 60 is preferred when discharging;

Discharging voltage must not be lower than 2.75V. over-discharge

It should be noted that the cell would be at an over-discharge state by its self-discharge. In order to prevent over-discharge, the cell shall be charged periodically to keeping voltage between 3.6-3.9V. Over-discharge may cause loss of cell performance. It should be noted that the cell would not discharge till voltage lower than 2.5V.

4.Storage of polymer lithium-ion batteries

The environment of long-time storage: Temperature: 20±5 ;

Humidity: 45-85%;

Batteries were 40 60% charged.

5.Transportation of polymer lithium-ion batteries

The batteries should transportation with 10 50% charged states.

6.Others

Please note cautions below to prevent cells' leakage, heat generation and explosion. Prohibition of disassembly cells;

Prohibition of cells immersion into liquid such as water or seawater; Prohibition of dumping cells into fire;

Prohibition of using damaged cells. The cells with a smell of electrolyte or leakage must be placed away from fire to avoid firing.

In case of electrolyte leakage contact with skin, eye, physicians shall flush the electrolyte immediately with fresh water and medical advise is to be sought.

9. Notice of Designing Battery Pack

9.1 Pack design

Battery pack should have sufficient strength and battery should be protected from mechanical shock. No sharp edge components should be inside the pack contain the battery.

9.2 PCM design

The overcharge threshold voltage should not be exceed 4.25V.

The over-discharge threshold voltage should not be lower than 2.75V. The PCM should have short protection function built inside.

9.3 Tab connection

Ultrasonic welding or spot welding is recommended to connect battery with PCM or other parts. If apply manual solder method to connect tab with PCM, the notice below is very important to ensure battery performance.

The electric iron should be temperature controlled and ESD safe; Soldering temperature should not exceed 350 ;

Soldering time should not be longer than 3s, keep battery tab cold down before next soldering; Soldering times should not exceed 5 times;

Directly heat cell body is strictly prohibited, battery may be damaged by heat above approx. 100 .

9.4 Cell fixing

The battery should be fixed to the battery pack by its large surface area. No cell movement in the battery pack should be allowed.

9.5 Cells replacement

The cell replacement should be done by professional people.

Prohibit short-circuit between cells' Al package and exterior component.

10. Cell Drawing:

A5 TIP41C and TIP42C Datasheet

TIP41A/41B/41C
TIP42A/42C

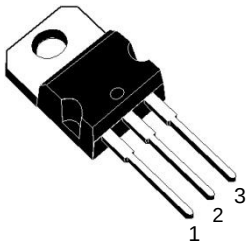
COMPLEMENTARY SILICON POWER TRANSISTORS

- COMPLEMENTARY PNP - NPN DEVICES

DESCRIPTION

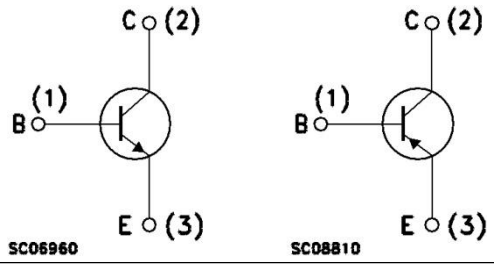
The TIP41A, TIP41B and TIP41C are silicon Epitaxial-Base NPN power transistors mounted in Jedec TO-220 plastic package. They are intented for use in medium power linear and switching applications.

The TIP41A and TIP41C complementary PNP types are TIP42A and TIP42C respectively.



TO-220

INTERNAL SCHEMATIC DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter		Value			Unit
		NPN	TIP41A	TIP41B	TIP41C	
		PNP	TIP42A		TIP42C	
V_{CBO}	Collector-Base Voltage (I_E = 0)		60	80	100	V
V_{CEO}	Collector-Emitter Voltage (I_B = 0)		60	80	100	V
V_{EBO}	Emitter-Base Voltage (I_C = 0)		5			V
I_C	Collector Current		6			A
I_{CM}	Collector Peak Current		10			A
I_B	Base Current		3			A
P_{tot}	Total Dissipation at $T_{case} \leq 25\ ^{\circ}C$ $T_{amb} \leq 25\ ^{\circ}C$		65 2			W W
T_{stg}	Storage Temperature		-65 to 150			^{o}C
T_j	Max. Operating Junction Temperature		150			^{o}C

For PNP types voltage and current values are negative.

THERMAL DATA

$R_{thj\text{-}case}$	Thermal	Resistance	Junction-case	Max	1.92	$^oC/W$
$R_{thj\text{-}amb}$	Thermal	Resistance	Junction-ambient	Max	62.5	$^oC/W$

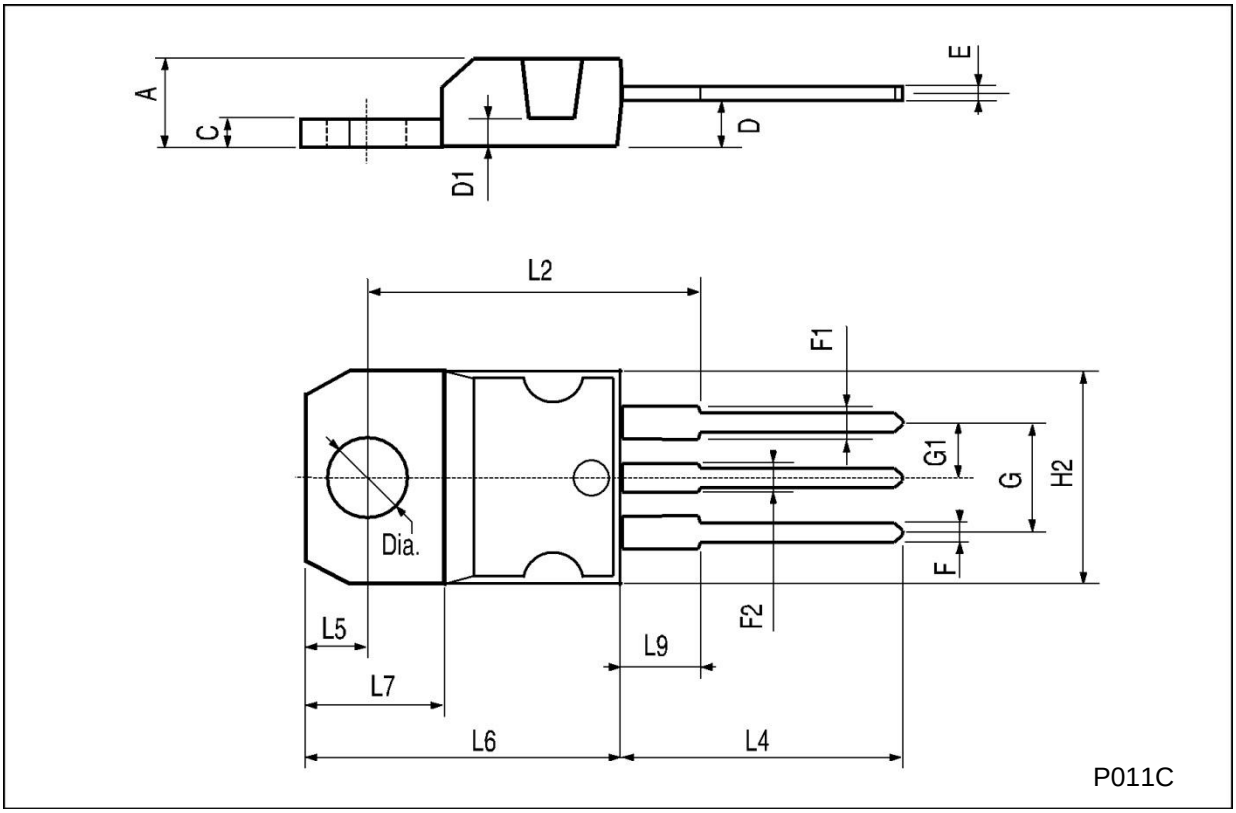
ELECTRICAL CHARACTERISTICS (T_{case} = 25 ^oC unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
I_{CEO}	Collector Cut-off Current (I_B = 0)	for **TIP41A/42A** V_{CE} = 30 V for **TIP41B/41C/42C** V_{CE} = 60 V			0.7 0.7	mA mA
I_{CES}	Collector Cut-off Current (V_{BE} = 0)	for **TIP41A/42A** V_{CE} = 60 V for **TIP41B** V_{CE} = 80 V for **TIP41C/42C** V_{CE} = 100 V			0.4 0.4 0.4	mA mA mA
I_{EBO}	Emitter Cut-off Current (I_C = 0)	V_{EB} = 5 V			1	mA
$V_{CEO(sus)}$*	Collector-Emitter Sustaining Voltage (I_B = 0)	I_C = 30 mA for **TIP41A/42A** for **TIP41B** for **TIP41C/42C**	 60 80 100		 	 V V V
$V_{CE(sat)}$*	Collector-Emitter Saturation Voltage	I_C = 6 A I_B = 0.6 A			1.5	V
$V_{BE(on)}$*	Base-Emitter Voltage	I_C = 6 A V_{CE} = 4 V			2	V
h_{FE}*	DC Current Gain	I_C = 0.3 A V_{CE} = 4 V I_C = 3 A V_{CE} = 4 V	30 15		 75	
h_{fe}	Small Signal Current Gain	I_C = 0.5 A V_{CE} = 10 V f = 1 KHz I_C = 0.5 A V_{CE} = 10 V f = 1 MHz	20 3			

∗ Pulsed: Pulse duration = 300 μs, duty cycle ≤ 2 % For
PNP types voltage and current values are negative.

TO-220 MECHANICAL DATA

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A	4.40		4.60	0.173		0.181
C	1.23		1.32	0.048		0.051
D	2.40		2.72	0.094		0.107
D1		1.27			0.050	
E	0.49		0.70	0.019		0.027
F	0.61		0.88	0.024		0.034
F1	1.14		1.70	0.044		0.067
F2	1.14		1.70	0.044		0.067
G	4.95		5.15	0.194		0.203
G1	2.4		2.7	0.094		0.106
H2	10.0		10.40	0.393		0.409
L2		16.4			0.645	
L4	13.0		14.0	0.511		0.551
L5	2.65		2.95	0.104		0.116
L6	15.25		15.75	0.600		0.620
L7	6.2		6.6	0.244		0.260
L9	3.5		3.93	0.137		0.154
DIA.	3.75		3.85	0.147		0.151



A6 2n2222 Datasheet

Central™
Semiconductor Corp.
145 Adams Ave., Hauppauge, NY 11788 USA
Phone (631) 435-1110 FAX (631) 435-1824
Manufacturers of World Class Discrete Semiconductors
www.centralsemi.com

2N2221A
2N2222A

NPN SILICON TRANSISTOR
JEDEC TO-18 CASE

DESCRIPTION:

The CENTRAL SEMICONDUCTOR 2N2221A, 2N2222A types are Silicon NPN Planar Epitaxial Transistors designed for small signal general purpose and switching applications.

MAXIMUM RATINGS: (T_A=25°C)

	SYMBOL		UNITS
Collector-Base Voltage	V_{CBO}	75	V
Collector-Emitter Voltage	V_{CEO}	40	V
Emitter-Base Voltage	V_{EBO}	6.0	V
Collector Current	I_C	800	mA
Power Dissipation	P_D	400	mW
Power Dissipation (T_C=25°C)	P_D	1.2	W
Operating and Storage Junction Temperature	T_J, T_{stg}	-65 to +200	°C
Thermal Resistance	Θ_{JA}	438	°C/W
Thermal Resistance	Θ_{JC}	146	°C/W

ELECTRICAL CHARACTERISTICS: (T_A=25°C unless otherwise noted)

		2N2221A		2N2222A		
SYMBOL	TEST CONDITIONS	MIN	MAX	MIN	MAX	UNITS
I_{CBO}	V_{CB}=60V		10		10	nA
I_{CBO}	V_{CB}=60V, T_A=150°C		10		10	∝A
I_{EBO}	V_{EB}=3.0V		10		10	nA
I_{CEV}	V_{CE}=60V, V_{EB}=3.0V		10		10	nA
BV_{CBO}	I_C=10∝A	75		75		V
BV_{CEO}	I_C=10mA	40		40		V
BV_{EBO}	I_E=10∝A	6.0		6.0		V
$V_{CE(SAT)}$	I_C=150mA, I_B=15mA		0.3		0.3	V
$V_{CE(SAT)}$	I_C=500mA, I_B=50mA		1.0		1.0	V
$V_{BE(SAT)}$	I_C=150mA, I_B=15mA	0.6	1.2	0.6	1.2	V
$V_{BE(SAT)}$	I_C=500mA, I_B=50mA		2.0		2.0	V
h_{FE}	V_{CE}=10V, I_C=0.1mA	20		35		
h_{FE}	V_{CE}=10V, I_C=1.0mA	25		50		
h_{FE}	V_{CE}=10V, I_C=10mA	35		75		
h_{FE}	V_{CE}=10V, I_C=10mA, T_A=-55°C	15		35		
h_{FE}	V_{CE}=10V, I_C=150mA	40	120	100	300	
h_{FE}	V_{CE}=1.0V, I_C=150mA	20		50		

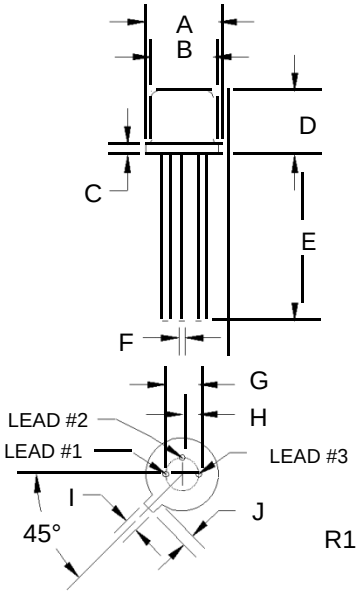
h_{FE}	V_{CE}=10V, I_C=500mA	25	40

(Continued)

ELECTRICAL CHARACTERISTICS: Continued

SYMBOL	TEST CONDITIONS	2N2221A MIN	2N2221A MAX	2N2222A MIN	2N2222A MAX	UNITS
f_T	V_{CE}=20V, I_C=20mA, f=100MHz	250		300		MHz
C_{ob}	V_{CB}=10V, I_E=0, f=100kHz		8.0		8.0	pF
C_{ib}	V_{EB}=0.5V, I_C=0, f=100kHz		25		25	pF
h_{ie}	V_{CE}=10V, I_C=1.0mA, f=1.0kHz	1.0	3.5	2.0	8.0	kΩ
h_{ie}	V_{CE}=10V, I_C=10mA, f=1.0kHz	0.2	1.0	0.25	1.25	kΩ
h_{re}	V_{CE}=10V, I_C=1.0mA, f=1.0kHz		5.0		8.0	$x10^{-4}$
h_{re}	V_{CE}=10V, I_C=10mA, f=1.0kHz		2.5		4.0	$x10^{-4}$
h_{fe}	V_{CE}=10V, I_C=1.0mA, f=1.0kHz	30	150	50	300	
h_{fe}	V_{CE}=10V, I_C=10mA, f=1.0kHz	50	300	75	375	
h_{oe}	V_{CE}=10V, I_C=1.0mA, f=1.0kHz	3.0	15	5.0	35	∝mhos
h_{oe}	V_{CE}=10V, I_C=10mA, f=1.0kHz	10	100	25	200	∝mhos
$rb'C_c$	V_{CB}=10V, I_E=20mA, f=31.8MHz		150		150	ps
NF	V_{CE}=10V, I_C=100∝A, R_S=1.0kΩ, f=1.0kHz				4.0	dB
t_d	V_{CC}=30V, V_{BE}=0.5, I_C=150mA, I_{B1}=15mA		10		10	ns
t_r	V_{CC}=30V, V_{BE}=0.5, I_C=150mA, I_{B1}=15mA		25		25	ns
t_s	V_{CC}=30V, I_C=150mA, I_{B1}=I_{B2}=15mA		225		225	ns
t_f	V_{CC}=30V, I_C=150mA, I_{B1}=I_{B2}=15mA		60		60	ns

TO-18 PACKAGE - MECHANICAL OUTLINE



DIMENSIONS				
	INCHES		MILLIMETERS	
SYMBOL	MIN	MAX	MIN	MAX
A (DIA)	0.209	0.230	5.31	5.84
B (DIA)	0.178	0.195	4.52	4.95
C	-	0.030	-	0.76
D	0.170	0.210	4.32	5.33
E	0.500	-	12.70	-
F (DIA)	0.016	0.019	0.41	0.48
G (DIA)	0.100		2.54	
H	0.050		1.27	
I	0.036	0.046	0.91	1.17
J	0.028	0.048	0.71	1.22

TO-18 (REV: R1)

LEAD CODE:

1) Emitter
2) Base
3) Collector

Central™ Semiconductor Corp.
145 Adams Ave., Hauppauge, NY 11788 USA
Phone (631) 435-1110 FAX (631) 435-1824
Manufacturers of World Class Discrete Semiconductors
www.centralsemi.com

B1 Code to Measure Battery Voltage

```
int analogpin=0;
double v = 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  v = analogRead(analogpin) * 0.0049; //Read voltage
  Serial.println(v); //Print Voltage value
  delay(5000); //Read every 5 seconds
  if (v==0){ //Battery depleted, print finish
    Serial.print("finish");
    delay(3600000);
  }
}
```

B2 Final Arduino Code

```
//Code written by Giuseppe Di Cillo (www.coagula.org) was used as a
template to create this program. The original code
//is released under GNU Public License, redistribution and/or
modification is permitted under the terms
//of the GNU General Public License as published by the Free Software
Foundation

#include <MenuBackend.h>      //MenuBackend library - copyright by
Alexander Brevig
#include <LCD3Wire.h>         //LCD3Wire library - available from
Arduino Playground
#include <Tlc5940.h>          //Tlc5940 library - created by Alex
Leone
#include <PID_Beta6.h>        //PID_Beta6 libary - created by Brett
Beauregard



int avg = 0;                  //variable to store current average
temperature
int setAvg = 0;               //variable to store the target average
temperature
int hBridgePin0 = 0;          //variables to set the current
direction for Hbridge(ie. cooling or heating of peltier plate)
int hBridgePin1 = 1;

int valTemp = 0;              //a temporary value used in the
calculation of temperature
double beta = 4100;           //beta value of the thermistors, used
to calculate temperature
double Rt = 0;                //used to store the resistance of a
thermistor in the calculation of temperature
```

```
//arrary to store the current temperature of different modules
double curTemp[6] = {0,0,0,0,0,0};
//arrary to store the target temperature of different modules(when
heating is required)
double setTemp[6] = {0,0,0,0,0,0};
//arrary to store the target temperature of different modules(when
cooling is required)
double setCool[6] = {100,100,100,100,100,100};
//arrary to store the output(amount of current to output through
peltier plate) when heating and cooling
double output[6] = {0,0,0,0,0,0};
double outputCool[6] = {0,0,0,0,0,0};

//flags to indicate if devices should be heating, cooling or neither
int onoff = 0;
int cool = 0;
int set;

//PID objects defined in the PID_Beta6 libary used to calcuate the
appropiate outputs based on PID calcuations
//the first three arguments are the input, output and target,the last
three are the Proportional,
//Integration and the Derivative parameters of the PID calcuation
PID pid0(&curTemp[0], &output[0], &setTemp[0], 30 , 4, 1);
PID pid1(&curTemp[1], &output[1], &setTemp[1], 30 , 4, 1);
PID pid2(&curTemp[2], &output[2], &setTemp[2], 30 , 4, 1);
PID pid3(&curTemp[3], &output[3], &setTemp[3], 30 , 4, 1);
PID pid4(&curTemp[4], &output[4], &setTemp[4], 30 , 4, 1);
PID pid5(&curTemp[5], &output[5], &setTemp[5], 30 , 4, 1);

const int buttonPinLeft = 2;      // pin for the Up button
const int buttonPinRight = 12;    // pin for the Down button
const int buttonPinEsc = 4;     // pin for the Esc button
const int buttonPinEnter = 5;   // pin for the Enter button

int lastButtonPushed = 0;

int lastButtonEnterState = LOW;   // the previous reading from the
Enter input pin
int lastButtonEscState = LOW;   // the previous reading from the Esc
input pin
int lastButtonLeftState = LOW;   // the previous reading from the
Left input pin
int lastButtonRightState = LOW;   // the previous reading from the
Right input pin


long lastEnterDebounceTime = 0;  // the last time the output pin was
toggled
long lastEscDebounceTime = 0;  // the last time the output pin was
toggled
long lastLeftDebounceTime = 0;  // the last time the output pin was
toggled
long lastRightDebounceTime = 0;  // the last time the output pin was
toggled
long debounceDelay = 500;    // the debounce time

//a character array used to convert an integer to characters that can
be displayed on the LCD
char temp[2];
```

```
//LCD objects defined in the LCD3Wire libary used for controlling the
LCD display
LCD3Wire lcd = LCD3Wire(2, 7, 8, 6);

//Menu variables
MenuBackend menu = MenuBackend(menuUsed,menuChanged);
//initialize menuitems
MenuItem all = MenuItem("All");
MenuItem allCheck = MenuItem("All Check");
MenuItem allSet = MenuItem("All Set");

MenuItem module0 = MenuItem("Module 0");
MenuItem module0Check = MenuItem ("Module0 Check");
MenuItem module0Set = MenuItem ("Module0 Set");

MenuItem module1 = MenuItem("Module 1");
MenuItem module1Check = MenuItem ("Module1 Check");
MenuItem module1Set = MenuItem ("Module1 Set");

MenuItem module2 = MenuItem("Module 2");
MenuItem module2Check = MenuItem ("Module2 Check");
MenuItem module2Set = MenuItem ("Module2 Set");

MenuItem module3 = MenuItem("Module 3");
MenuItem module3Check = MenuItem ("Module3 Check");
MenuItem module3Set = MenuItem ("Module3 Set");

MenuItem module4 = MenuItem("Module 4");
MenuItem module4Check = MenuItem ("Module4 Check");
MenuItem module4Set = MenuItem ("Module4 Set");

MenuItem module5 = MenuItem("Module 5");
MenuItem module5Check = MenuItem ("Module5 Check");
MenuItem module5Set = MenuItem ("Module5 Set");

MenuItem stopAll = MenuItem("Stop All");
MenuItem stoppedAll = MenuItem("Stopped All");

void setup()
{
  //set the input/output modes of the pins
  pinMode(hBridgePin0, OUTPUT);
  pinMode(hBridgePin1, OUTPUT);
  pinMode(buttonPinLeft, INPUT);
  pinMode(buttonPinRight, INPUT);
  pinMode(buttonPinEnter, INPUT);
  pinMode(buttonPinEsc, INPUT);

  //setup of the PID objects
  pid0.SetOutputLimits(0,1023);
  pid0.SetMode(AUTO);
  pid1.SetOutputLimits(0,1023);
  pid1.SetMode(AUTO);
  pid2.SetOutputLimits(0,1023);
  pid2.SetMode(AUTO);
  pid3.SetOutputLimits(0,1023);
  pid3.SetMode(AUTO);
  pid4.SetOutputLimits(0,1023);
  pid4.SetMode(AUTO);
  pid5.SetOutputLimits(0,1023);
```

```
  pid5.SetMode(AUTO);

  //Initialise the Tlc(PWM) chip and the lcd
  Tlc.init();
  lcd.init();

  //configure menu
  menu.getRoot().add(all);
  all.addRight(module0);
  module0.add(module0Check);
  module0Check.add(module0Set);
  module0.addRight(module1);
  module1.add(module1Check);
  module1Check.add(module1Set);
  module1.addRight(module2);
  module2.add(module2Check);
  module2Check.add(module2Set);
  module2.addRight(module3);
  module3.add(module3Check);
  module3Check.add(module3Set);
  module3.addRight(module4);
  module4.add(module4Check);
  module4Check.add(module4Set);
  module4.addRight(module5);
  module5.add(module5Check);
  module5Check.add(module5Set);
  all.addLeft(stopAll);
  stopAll.add(stoppedAll);
  all.add(allCheck);
  allCheck.add(allSet);

  //set all the outputs to zero
  for(int i = 0; i < 12; i++){
    Tlc.set(i, 0);
    Tlc.update();
  }

  //read the current temperature into the curTemp array
  readTemp(curTemp);

  menu.toRoot();
}  // setup()...


void loop()
{
  readButtons();  //I splitted button reading and navigation in two
procedures because
  navigateMenus();  //in some situations I want to use the button for
other purpose (eg. to change some settings)
  //only do turn on output if flag is on
  readTemp(curTemp);
  if(onoff == 1){
    if(cool == 0){
      //set the hBridge configuration to allow heating
      digitalWrite(hBridgePin0, HIGH);
      digitalWrite(hBridgePin1, LOW);
      readTemp(curTemp);
      //compute the outputs requried to reach the targeted
temperatures
      pid0.Compute();
```

```
      pid1.Compute();
      pid2.Compute();
      pid3.Compute();
      pid4.Compute();
      pid5.Compute();
      //output the computed values
      Tlc.set(1, output[0] * 4);
      Tlc.set(2, output[1] * 4);
      Tlc.set(3, output[2] * 4);
      Tlc.set(4, output[3] * 4);
      Tlc.set(5, output[4] * 4);
      Tlc.set(6, output[5] * 4);
      //make sure the the pins used for cooling is off
      for(int i = 7; i < 13; i++){
        Tlc.set(i, 0);
      }
      Tlc.update();
    }
    else{
      //configure Hbridge to allow cooling
      digitalWrite(hBridgePin0, LOW);
      digitalWrite(hBridgePin1, HIGH);
      //run the function that takes care of cooling
      checkCool(curTemp, setCool, outputCool);
    }
  }
  //ensure that all the outputs are 0 when we are not heating or
cooling
  else{
    for(int i = 1; i < 13; i++){
    Tlc.set(i, 0);
  }
    Tlc.update();
  }
} //loop()...


void menuChanged(MenuChangeEvent changed){

  MenuItem newMenuItem=changed.to; //get the destination menu

  lcd.cursorTo(0,0); //set the start position for lcd printing to the
second row

  //Display the menus on the LCD
  if(newMenuItem.getName()==menu.getRoot()){
    lcd.clear();
    lcd.printIn("Main Menu      ");
  }
  else if(newMenuItem.getName() == "Module 0"){
    lcd.printIn("Chest Left");
  }
  else if(newMenuItem.getName() == "Module 1"){
    lcd.printIn("Chest Right");
  }
  else if(newMenuItem.getName() == "Module 2"){
    lcd.printIn("Stomach Left");
  }
  else if(newMenuItem.getName() == "Module 3"){
    lcd.printIn("Stomach Right");
  }
```

```
  else if(newMenuItem.getName() == "Module 4"){
    lcd.printIn("Upperback Left");
  }
  else if(newMenuItem.getName() == "Module 5"){
    lcd.printIn("Upperback Right");
  }

  //procedures to be executed when certain menus are chosen such as
displaying the current temperature on the selected module
  //or setting its target temperature
  else if(newMenuItem.getName() == "Module0 Check"){
    lcd.clear();
    lcd.printIn("ChestL Check");
    lcd.cursorTo(2,0);
    itoa(curTemp[0], temp, 10);
    lcd.printIn(temp);
  }

  else if(newMenuItem.getName() == "Module0 Set"){
    settingTemp(0);
  }

  else if(newMenuItem.getName() == "Module1 Check"){
    lcd.printIn("ChestR Check");
    lcd.cursorTo(2,0);
    itoa(curTemp[1], temp, 10);
    lcd.printIn(temp);
  }

  else if(newMenuItem.getName() == "Module1 Set"){
    settingTemp(1);
  }

  else if(newMenuItem.getName() == "Module2 Check"){
    lcd.printIn("StomachL Check");
    lcd.cursorTo(2,0);
    itoa(curTemp[2], temp, 10);
    lcd.printIn(temp);
  }

  else if(newMenuItem.getName() == "Module2 Set"){
    settingTemp(2);
  }

  else if(newMenuItem.getName() == "Module3 Check"){
    lcd.printIn("StomachR Check");
    lcd.cursorTo(2,0);
    itoa(curTemp[3], temp, 10);
    lcd.printIn(temp);
  }

  else if(newMenuItem.getName() == "Module3 Set"){
    settingTemp(3);
  }

  else if(newMenuItem.getName() == "Module4 Check"){
    lcd.printIn("UpperbackL Check");
    lcd.cursorTo(2,0);
    itoa(curTemp[4], temp, 10);
    lcd.printIn(temp);
  }
```

```
  else if(newMenuItem.getName() == "Module4 Set"){
    settingTemp(4);
  }

  else if(newMenuItem.getName() == "Module5 Check"){
    lcd.printIn("UpperbackR Check");
    lcd.cursorTo(2,0);
    itoa(curTemp[5], temp, 10);
    lcd.printIn(temp);
  }

  else if(newMenuItem.getName() == "Module5 Set"){
    settingTemp(5);
  }

  else if(newMenuItem.getName() == "Stop All"){
    lcd.printIn("Stop All");
  }

  //procedure to stop all current heating/cooling
  else if(newMenuItem.getName() == "Stopped All"){
    lcd.printIn("Stopped All");
    onoff = 0;
    cool = 0;
    delay(2000);
    menu.toRoot();
  }
  else if(newMenuItem.getName()=="All"){
    lcd.printIn("All            ");
  }

  //slightly modified procedures from above to enable heating and
cooling of all modules using average temperatures
  else if(newMenuItem.getName()=="All Check"){
    lcd.printIn("All Check      ");
    lcd.cursorTo(2,0);
    readTemp(curTemp);
    avg = average(curTemp);
    delay(250);
    itoa(avg, temp, 10);
    lcd.printIn(temp);
    setAvg = avg;
  }
  else if(newMenuItem.getName()=="All Set"){
    lcd.printIn("All Set        ");
    lastButtonPushed = 0;
    while(lastButtonPushed != buttonPinEnter){
      readButtons();
      if (lastButtonPushed == buttonPinRight){
        setAvg++;
        itoa(setAvg, temp, 10);
        lcd.clear();
        lcd.cursorTo(2,0);
        lcd.printIn(temp);
      }
      if (lastButtonPushed == buttonPinLeft){
        setAvg--;
        itoa(setAvg, temp, 10);
        lcd.clear();
        lcd.cursorTo(2,0);
```

```
        lcd.printIn(temp);
      }
    }
    //decide whether we need to heat or cool and set the appropiate
arrays
    if(setAvg >= avg){
      for(int i = 0; i < 6; i++){
        setTemp[i] = setAvg;
      }
      cool = 0;
    }
    else{
      for(int i = 0; i < 6; i++){
        setCool[i] = setAvg;
      }
      cool = 1;
    }
    lcd.cursorTo(1,0);
    onoff = 1;
    //Serial.println(avg);
    lcd.printIn("Temp set to:");
    delay(2000);
    menu.toRoot();
  }
}

void menuUsed(MenuUseEvent used){
  lcd.cursorTo(0,0);
  lcd.printIn("You used        ");
  lcd.cursorTo(2,0);
  lcd.printIn((char*)used.item.getName());
  delay(3000);  //delay to allow message reading
  lcd.clear();
  lcd.cursorTo(0,0);
  menu.toRoot();  //back to Main
}


void readButtons(){  //read buttons status
  int reading;
  int buttonEnterState=LOW;             // the current reading from
the Enter input pin
  int buttonEscState=LOW;             // the current reading from the
input pin
  int buttonLeftState=LOW;             // the current reading from
the input pin
  int buttonRightState=LOW;             // the current reading from
the input pin

  //Enter button
  // read the state of the switch into a local variable:
  reading = digitalRead(buttonPinEnter);

  // check to see if you just pressed the enter button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonEnterState) {
    // reset the debouncing timer
    lastEnterDebounceTime = millis();
```

```
  }

  if ((millis() - lastEnterDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current
state:
    buttonEnterState=reading;
    lastEnterDebounceTime=millis();
  }

  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonEnterState = reading;


  //Esc button
  // read the state of the switch into a local variable:
  reading = digitalRead(buttonPinEsc);

  // check to see if you just pressed the Down button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonEscState) {
    // reset the debouncing timer
    lastEscDebounceTime = millis();
  }

  if ((millis() - lastEscDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current
state:
    buttonEscState = reading;
    lastEscDebounceTime=millis();
  }

  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonEscState = reading;


  //Down button
  // read the state of the switch into a local variable:
  reading = digitalRead(buttonPinRight);

  // check to see if you just pressed the Down button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonRightState) {
    // reset the debouncing timer
    lastRightDebounceTime = millis();
  }

  if ((millis() - lastRightDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current
state:
    buttonRightState = reading;
```

```
    lastRightDebounceTime =millis();
  }

  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonRightState = reading;


  //Up button
  // read the state of the switch into a local variable:
  reading = digitalRead(buttonPinLeft);

  // check to see if you just pressed the Down button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonLeftState) {
    // reset the debouncing timer
    lastLeftDebounceTime = millis();
  }

  if ((millis() - lastLeftDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current
state:
    buttonLeftState = reading;
    lastLeftDebounceTime=millis();
    ;
  }

  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonLeftState = reading;

  //records which button has been pressed
  if (buttonEnterState==HIGH){
    lastButtonPushed=buttonPinEnter;

  }
  else if(buttonEscState==HIGH){
    lastButtonPushed=buttonPinEsc;

  }
  else if(buttonRightState==HIGH){
    lastButtonPushed=buttonPinRight;

  }
  else if(buttonLeftState==HIGH){
    lastButtonPushed=buttonPinLeft;

  }
  else{
    lastButtonPushed=0;
  }
}

void navigateMenus() {
  MenuItem currentMenu=menu.getCurrent();

  switch (lastButtonPushed){
```

```
  case buttonPinEnter:
    if(!(currentMenu.moveDown())){  //if the current menu has a child
and has been pressed enter then menu navigate to item below
      menu.use();
    }
    else{  //otherwise, if menu has no child and has been pressed
enter the current menu is used
      menu.moveDown();
    }
    break;
  case buttonPinEsc:
    menu.toRoot();  //back to main
    break;
  case buttonPinRight:
    menu.moveRight();
    break;
  case buttonPinLeft:
    menu.moveLeft();
    break;
  }

  lastButtonPushed=0; //reset the lastButtonPushed variable
}

//function to read the current temperature of all the modules and put
the values into the curTemp array
void readTemp(double *curTemp){
  for(int i = 0; i < 6; i++){
    //read the normalised voltage across the thermistor network
    valTemp = analogRead(i);
    //calculate the resistance of the thermistor
    Rt = 553.684711*exp(0.874921 * valTemp * 0.0049);
    //calculate the temperature from the resistance value
    curTemp[i] = (beta/(log(Rt/10000)+(beta/298.15))) - 273.15;
  }
  delay(250);
}

//function to calculate the average temperature from the six modules
int average(double *curTemp){
  int sum = 0;
  for(int i = 0; i < 6; i++){
    sum = sum + curTemp[i];
  }
  return sum/6;
}

//function that enables cooling
void checkCool(double *curTemp, double *setCool, double *outputCool){
  for(int i = 0; i < 6; i++){
    if(curTemp[i] > setCool[i]){
      //determine how much effort we want to put into cooling the
module
      double factor = coolPower(curTemp[i], setCool[i]);
      Tlc.set(i + 7, factor * 4094);
    }
    else Tlc.set(i+7, 0);
  }
  Tlc.update();
}
```

```
//function to determine how much effort(current) needs to used to
cool the module, similar to the concept of PID
//the effort needed is proportional to the temperature difference of
the current and target temperature
double coolPower(double curTemp, double setCool){
  double diff = curTemp - setCool;
  if (diff >= 5) return 1;
  else if(diff > 2.5) return 0.5;
  else if(diff > 1.5) return 0.3;
  else if(diff > 0) return 0.1;
  else return 0;
}

//function to set the target temperature of module
void settingTemp(int i){
  readTemp(curTemp);
  lcd.clear();
  set = curTemp[i];
  lcd.clear();
  switch (i)
  {
    case 0:
    lcd.printIn("ChestL Set"); break;
    case 1:
    lcd.printIn("ChestR Set"); break;
    case 2:
    lcd.printIn("StomachL Set"); break;
    case 3:
    lcd.printIn("StomachR Set"); break;
    case 4:
    lcd.printIn("UpperbackL Set"); break;
    case 5:
    lcd.printIn("UpperbackR Set"); break;
  }
  lcd.cursorTo(2,0);
  itoa(set, temp, 10);
  lcd.printIn(temp);
  lastButtonPushed = 0;
  while(lastButtonPushed != buttonPinEnter){
    readButtons();
    if (lastButtonPushed == buttonPinRight){
      set++;
      itoa(set, temp, 10);
      lcd.clear();
      lcd.cursorTo(2,0);
      lcd.printIn(temp);
    }
    if (lastButtonPushed == buttonPinLeft){
      set--;
      itoa(set, temp, 10);
      lcd.clear();
      lcd.cursorTo(2,0);
      lcd.printIn(temp);
    }
  }
  lcd.cursorTo(1,0);
  lcd.printIn("Temp set to:");
  delay(2000);
  if (set < curTemp[i]){
    setCool[i] = set;
    cool = 1;
```

```
  }
  else{
    setTemp[i] = set;
    cool = 0;
  }
  onoff = 1;
  menu.toRoot();
}
```