SAND2019-14228R

Final Letter Report for AEgis Technologies NMSBA Project

Sheraline Lawles
Bobby D. Middleton

November 2019

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND DEFINITIONS

Click here, then press delete to remove guidance.

Delete this page if you do not have acronyms or definitions.

Abbreviation	Definition

1. FINAL LETTER REPORT FOR AEGIS TECHNOLOGIES NMSBA PROJECT

This letter report signals the end of the NMSBA project to help AEgis Technologies (AEGis) find a way to solve a coupled heat transfer equation for a laser-heated Silicon wafer. Accompanying this letter report is a MATLAB live script that documents the work done to date and provides a first attempt at a solution.

The scope of work provided for Sandia National Laboratories (SNL) to document the problem in a general form in this phase of the work, with the goal of applying for additional funding in Calendar Year 2020 to attempt a complete solution. SNL staff analytically solved the differential heat equation and found solutions that reproduce reasonable shapes for heat flux. However, the required information to provide a complete solution is not available.

The problem is currently stated as two coupled differential heat equations.

$$\begin{aligned} C_1 \frac{dT_1}{dt} &= q - G_{1s}(T_1 - T_s) - G_{12}(T_1 - T_2) \\ C_2 \frac{dT_2}{dt} &= G_{12}(T_1 - T_2) - G_{2s}(T_2 - T_s) \end{aligned} \tag{0.1}$$

This form of the equation treats a single point on the experimental arrangement; this point is assumed to be a temperature sensor. T_1 is the temperature of the irradiated material (assumed to be Silicon); T_2 is the temperature of the FR4 composite substrate. However, the exact makeup of the FR4 is unknown to either AEgis personnel or to SNL staff. Since C_1 and C_2 are heat capacity constants, it is highly likely that the value for C_2 is suspect. The only other data that are available are measured values for T_1 taken at a measurement rate of 10 samples per second.

There are also data that have been collected for a different experiment (known as the free-standing configuration) that allows for calculation of an upper bound on G_{1s}. This upper bound on G_{1s} was found to be G = 0.0324 J/(cm^2-sec-K). In the free-standing configuration, the only heat transfer is from the silicon to air. In the coupled configuration, heat is also transferred via thermal contact to the FR4 substrate. Since the conductance to the FR4 is much higher than from silicon to air, the value for G_{1s} should be higher in the free-standing configuration.

The basic process for solving the two equations from (1.1) is to set q to zero and solve the coupled, linear, homogeneous, ordinary differential equations (ODEs) via the use of their common eigenvalues. This process is outlined in the accompanying MATLAB live script. These equations are used to find the values for G_{12} and G_{2s} in terms of G_{1s}, C_1, and C_2. Since only an upper bound is known for G_{1s}, a vector of values for G_{1s} was calculated and the corresponding values for G_{12} and G_{2s} were then calculated. Since the equations (1.1) assume positive values for the constants, any combinations with negative values were discarded as being physically irrelevant. For the MATLAB live script, a value of 0.1*G was used for G_{1s}, giving values of:

$$\begin{aligned} G_{12} &= 0.0187 \\ G_{2s} &= 0.0044 \end{aligned} . \tag{0.2}$$

Fitting a curve to the T_1 data for the time period that q>0, MATLAB's *dsolve* function was used to calculate T_2 during this same time. Once that calculation is performed, the result was then substituted into the first equation to calculate q.

Although this outlines the basic process for calculating q, it is noted that the scope of work only required that SNL set up the problem, not solve it completely. This is because discussions determined that AEGis would likely apply for a second round of funding in CY 2020. As such, the work produced thus far is actually beyond the original scope of work. The authors of this letter report urge AEGis to apply for further funding in CY 2020.

Some ways to improve results include:

1. Finding a way to record temperature measurements on both the Silicon and the FR4 substrate.
2. Determining the exact makeup of the FR4 so the value of C_2 is accurate.
3. Ensuring that the reflectivity of the Silicon is known and provided to SNL for any future analysis.

APPENDIX A. TEXT VERSION OF MATLAB CODE

Importing shot 37 data, a sample set from the thermally coupled sensor array, temperature data is animated to help visualize it and choose a centrally located sensor to work with:

```
clear, clc, close all
load('TofT-Shot-37.mat')
T = RTD.T(1:20,1:20,1:293); % T1


% animated for visualization
t1 = T(1:20,1:20,1);
figure(1)
h = image(t1);
colorbar
for i = 1:293
    set(h, 'cdata', T(1:20,1:20,i))
    pause(0.05)
end
```

Ambient temperature, T_s , is found by averaging the values before the signal, q, is turned on.

```
sig = RTD.trig;
for i=1:293
    if sig(i) > 0.001
        sig(i) = 1;
    else
        sig(i) = 0;
    end
end
%figure; plot(sig*50)
firstsig = find(sig,1,'first'); %first timestep of signal = 23
lastsig = find(sig,1,'last'); %last timestep of signal = 123
Tavg = zeros(20);
for m = 1:20
    for n = 1:20
        Tavg(m,n) = nanmean(T(m,n,1:41));
    end
end
Ts = nanmean(Tavg,'all');
```

Looking at a temperature stillshot of the last timestep when the signal is on, we find that a reasonable 5x5 centroid range of sensors is found from (9,9) to (13,13). A plot of the temperature data at sensor (9,9) and then for all centroid sensors is shown.

```
figure(2)
t146 = T(1:20,1:20,146); % 20x20 array at t = 146
image(t146)
colorbar
title('Sensor Array at t = 14.6(s)')
```

```
Tsingle = squeeze(T(9,9,1:293));
figure(3)
plot(Tsingle); title('Raw Temperature at Centroid, (9,9)');


%Plot of temperature vectors for centroid sensors
figure(4)
for i=9:13
    for j=9:13
        sensorT = squeeze(T(i,j,:));
        plot(sensorT)
        hold on
    end
end
title('Temperature for Centroid Sensors (Raw), Shot 37')


Tcent = T(9:13,9:13,1:293); %temp at 5x5 of centroid sensors
```

We will be exploring a system of equations of the form:

$$C_1\dot{T}_1 = q - G_{1s}(T_1 - T_s) - G_{12}(T_1 - T_s)$$

$$C_2\dot{T}_2 = G_{12}(T_1 - T_2) - G_{2s}(T_2 - T_s)$$

$\frac{\mathrm{dT}}{\mathrm{dt}}$ or $\dot{T}_1$ at the centroid sensors is found using central differencing:

```
dTdt_cent = zeros(5,5,293);
%figure(5); title('dT/dt of Centroid Sensors (i=9:13, j=9:13)')
for m = 9:13
    for n = 9:13
        Tmn = squeeze(T(m,n,1:293));
        dT = zeros(1,length(Tmn));
        dt = 0.1;
        for i=1:length(dT)
            if i==1
                dT(i)=2*(Tmn(i+1) - Tmn(i));
            elseif i==length(dT)
                dT(i)=2*(Tmn(i) - Tmn(i-1));
            else
                dT(i)=Tmn(i+1) - Tmn(i-1);
            end
        end
        dTdt = (dT')./(2*dt);
        dTdt_cent(m-8,n-8,1:293) = dTdt;
        %plot(dTdt); hold on
    end
```

```
end
```

Using only sensor (9,9) now as a starting point, we let $\Delta T_{1s} = T_1 - T_s$ and fit a two-term exponential of the form $ae^{bt} + ce^{dt}$ to the portion of ΔT_{1s} where $q = 0$. The figure shows the raw cooling temperature data, as well as the exponential fit.

```
deltaTcent = Tcent - Ts;
CdeltaT = squeeze(deltaTcent(1,1,150:293));
t = [0.1:0.1:14.4]';
Fit1 = fit(t,CdeltaT,'exp2');
CE = coeffvalues(Fit1);
a = CE(1); b = CE(2); c = CE(3); d = CE(4);
Fit2 = a*exp(b*t) + c*exp(d*t);
figure(6), plot(CdeltaT); hold on
plot(Fit2)
legend('data','fit')
title('Curve Fit of Cooling Temperature at (9,9)')
```

Using our system of equations above, we let $x = T_1 - T_s$ and $y = T_2 - T_s$. This also gives $\dot{x} = \dot{T}_1$, $\dot{y} = \dot{T}_2$, and $T_1 - T_2 = x - y$.

For $q = 0$, this gives us the following equations:

$$\dot{x} = -\left(\frac{G_{1s} + G_{12}}{C_1}\right)x + \left(\frac{G_{12}}{C_1}\right)y$$

$$\dot{y} = \left(\frac{G_{12}}{C_2}\right)x - \left(\frac{G_{12} + G_{2s}}{C_2}\right)y$$

This system can then be expressed as a matrix equation of the form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\frac{G_{1s} + G_{12}}{C_1} & \frac{G_{12}}{C_1} \\ \frac{G_{12}}{C_2} & -\frac{G_{12} + G_{2s}}{C_2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Analytically, the solution of this system is of the form:

$$\begin{bmatrix} x \\ y \end{bmatrix} = A\overrightarrow{x_+}e^{\lambda_+ t} + B\overrightarrow{x\_}e^{\lambda\_ t}$$

where $\overrightarrow{x_+}$ is the eigenvector (2x1) for scalar-valued λ_+ and $\overrightarrow{x_-}$ is the eigenvector for λ_-.

```
syms G1s G12 G2s C1 C2


a11 = (G1s + G12)/C1;
a12 = G12/C1;
```

```
a21 = G12/C2;
a22 = (G2s + G12)/C2;


M = [-a11, a12; a21, -a22];
[V,D] = eig(M);
lambda_m = D(1,1); % eigenvalues as functions of G- and C-values
lambda_p = D(2,2);
x_m = V(:,1); %eigenvectors as functions of G- and C-values
x_p = V(:,2);
```

The components of this solution correspond with the coefficients of the curve fit for cooling ΔT_1 as follows:

$\lambda_- = b$

$\lambda_+ = d$

$B = \frac{a}{\overrightarrow{x_-}}$

$A = \frac{c}{\overrightarrow{x_+}}$

```


eqn1 = b == lambda_m;
eqn2 = d == lambda_p;
eqns = [eqn1, eqn2];
vars = [G12, G2s];
S = solve(eqns,vars);
```

The above code solves for G_{12} and G_{2s} in terms of $C_1, C_2,$ and G_{1s}.

C_1 and C_2 are calculated using the following values:

```
rho1 = 2.33; %g/cm^3, density
thk1 = 0.05; %cm, thickness of sensor (Vol/Area)
cp1 = 0.71; %J/gK, avg. specific heat for Silicon in temperature range
C1 = cp1*rho1*thk1;


rho2 = 1.8; %g/cm^3 or 1800 kg/m^3 (what Jim is using)
thk2 = 0.159; %cm
cp2 = 0.8; %J/gK
C2 = cp2*rho2*thk2;
```

G_{1s} is calculated as a fraction of the G value computed in shot 25. The matrix *Gcompare* outputs pairs of G_{12} and G_{2s} corresponding to values of G_{1s} calculated as $i * G$ for i from 0.1 to 1.0 in increments of 0.1.

```
G = 0.0324; % from shot 25, rounded to 4 decimals
G12 = zeros(1,10);
G2s = zeros(1,10);
for i = 1:10
```

```
    G1s = 0.1*i*G;
    G12(i) = double(subs(S.G12(1,:))); % gives numeric solutions for G12 and G2s
    G2s(i) = double(subs(S.G2s(1,:)));
end
Gcompare = [G12;G2s]


G1s = 0.1*G; %manually chosen from Gcompare
G12 = Gcompare(1,1);
G2s = Gcompare(2,1); % for results (A), set G2s = G1s
```

Values for $\lambda_-, \lambda_+, \vec{x_-}$, and $\vec{x_+}$ are now calculated by substituting in numeric values for $C_1, C_2, G_{1s}, G_{12}, \text{and } G_{2s}$. The solutions, $x = T_1 - T_s$ and $y = T_2 - T_s$ are calculated and plotted.

```
lambda_m = double(subs(lambda_m)); %gives numeric eigenvalues
lambda_p = double(subs(lambda_p));
lambda_m = lambda_m(1); lambda_p = lambda_p(1);


x_m = double(subs(x_m)); %gives numeric eigenvectors
x_p = double(subs(x_p));


B = a/x_m(1); %gives A & B values
A = c/x_p(1);


t = [0.1:0.1:14.4];
x = A*x_p(1)*exp(lambda_p*t) + B*x_m(1)*exp(lambda_m*t);
y = A*x_p(2)*exp(lambda_p*t) + B*x_m(2)*exp(lambda_m*t);


figure(7); plot(t,x,t,y);
legend('x','y'), title('x and y, using G1s = 0.1*G, G2s \= G1s')
xlabel('x = T1 - Ts'), ylabel('y = T2 - Ts');
```

From here, we turn to a computation of T_2 while signal q is on, leading to the desired final computation of q.

Rearranging the second equation in our system,

$$C_1\dot{T}_1 = q - G_{1s}(T_1 - T_s) - G_{12}(T_1 - T_s)$$

$$C_2\dot{T}_2 = G_{12}(T_1 - T_2) - G_{2s}(T_2 - T_s)$$

we get a differential equation

$$\dot{T}_2 = -\left(\frac{G_{12} + G_{2s}}{C_2}\right)T_2 + \left(\frac{G_{12}}{C_2}\right)T_1 + \left(\frac{G_{2s}}{C_2}\right)T_s$$

To use Matlab's *dsolve* function, we fit a function to the T_1 measured data for the period that q is on and then solve for T_2 symbolically. (Alternately, one may choose to use the *ode45* function and the data itself.)

```
% Fit for T1 while q is on, from timestep 42 to 142
t2 = [0.1:0.1:10.1]';
T1_on = squeeze(Tcent(1,1,42:142));
figure(10), plot(T1_on), title('T1 while q is on')
fit_deltaT_on = fit(t2, T1_on, 'exp2')
ab = coeffvalues(fit_deltaT_on);
a = ab(1); b = ab(2); c = ab(3); d = ab(4);


syms T2(t) %G12 G2s C2 - add these back to simplify trying multiple values
beta = (G12 + G2s)/C2;
T1eqn = a*exp(b*t) + c*exp(d*t);
eqn = diff(T2) == -beta*T2 + (G12/C2)*(T1eqn) + (G2s/C2)*Ts; %another option is
ode45
T2eqn = dsolve(eqn)
```

The symbolic solution includes an arbitrary constant in the form of the letter C with some appended number (other than 1 or 2), e.g. C7. This constant is given a new appended number every time the code is run during an open session, so the following code currently needs to be altered accordingly based on *T2eqn* from the previous step. The constant is then solved for using initial condition T_s at $t = 0$.

```
% solve for constant using initial condition <--- must look set below C
% number to that of constant in T2eqn
t = 0;
evalC3 = Ts == subs(T2eqn);
syms C3
constant = double(solve(evalC3, C3));
C3 = constant;
```

We can now substitute in all necessary values and solve for T_2 over the period when q is turned on. T_1 and T_2 are plotted together over this period.

```
t = [0.1:0.1:10.1]';
T2_on = double(subs(T2eqn));
figure(11), plot(t,T2_on), title('T1 and T2 while q is on'), xlabel('t'),
ylabel('T2')
hold on
plot(t,T1_on), legend('T2','T1')
```

Finally, heat flux is calculated using the first equation in our system, rearranged:

$$q = C_1\dot{T}_1 + G_{1s}(T_1 - T_s) + G_{12}(T_1 - T_2)$$

The code currently uses a raw $\frac{\mathrm{dT}}{\mathrm{dt}}$ that has not been smoothed, which then shows up as noise in our calculated q. The scaling of q values is off, but the shape is showing to be accurate to the signal on and off periods.

```
% Calculating q while q is on


% dT/dt while q is on, discarding constant portion at end of signal
dTdt_on = squeeze(dTdt_cent(1,1,42:142));
figure(9); plot(dTdt_on)
title('Raw dT/dt of T1 while q is on')


q_on = C1*dTdt_on + G1s*(T1_on - Ts) + G12*(T1_on - T2_on);
figure(12), plot(t,q_on), title('Calculated q while q is on')
q_on_smooth = smoothdata(q_on,'gaussian',20)';
hold on, plot(t,q_on_smooth), legend('raw q','smooth q')
```

The following confirms that heat flux is correctly calculated when $q = 0$ and requires smoothing for the same reason as above.

```
% Calculating q while q is off
t = [0.1:0.1:14.4];
T1_off = x + Ts;
T2_off = y + Ts;
dTdt_off = squeeze(dTdt_cent(1,1,150:293));
q_off = C1.*dTdt_off' + G1s.*(T1_off - Ts) + G12.*(T1_off - T2_off);
figure(13), plot(t,q_off),title('Calculated q while q is off')
q_off_smooth = smoothdata(q_off,'gaussian',20);
hold on, plot(t,q_off_smooth), legend('raw q','smooth q')
```