FEM Assignment

The Finite Element Method

Felicia Segui
Ebba Rickard

May 2020

Contents

1 Introduction

A Netflix server hall has seen an increase in usage since the COVID-19 virus has forced more people to stay at home. This increase in data consumption proportionally increases the heat generated in the integrated circuits that run the Netflix server halls. The heat generates in its turn a degree of deformation in the data chip. To decrease the streaming quality is a way to decrease the data consumption, as well as the heat generated in the circuit. The aim of this project is to investigate how the chip is affected if Netflix decreases their streaming quality, assuming a proportional relationship between heat generation and data consumption, presented in equation 1.

$$Q \propto data \tag{1}$$

The dimensions of a cross-section of the chip are presented in figure 1. The chip has a thickness of 10 mm and is fixated at the bottom of the structure. It is symmetric around the symmetry line and has convection at the upper boundary, as shown in figure 1. The heat is generated in the die made from silicon.

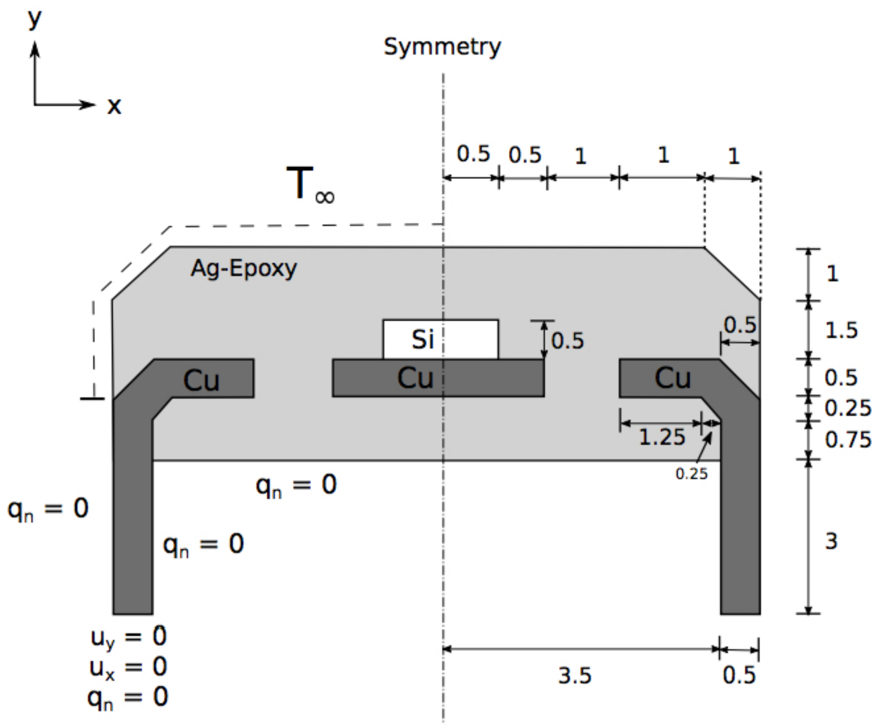


Figure 1: The dimensions in mm of a cross-section of the chip.

The material constants for the materials in the chip are presented in table 1.

Table 1: Material constants.

Constant	**Ag-epoxy**	**Silicon**	**Copper**
Young's modulus, E [GPa]	7	165	128
Poisson's ratio, v [-]	0,3	0,22	0,36
Expansion coefficient, α [1/K]	$4 \cdot 10^{-5}$	$2,6 \cdot 10^{-6}$	$17,6 \cdot 10^{-6}$
Density, ρ [kg/m^3]	2500	2530	8930
Specific heat, c_p [J/(Kg K)]	1000	703	386
Thermal conductivity, k [W/(m K)]	5	149	385

In table 2 other constants and given values that were used throughout the project are presented.

Table 2: The specific numbers used when assessing the heat development in the chip.

Constant	**Value**
Thickness, A [mm]	10
Thermal transmittance, α_c [W / (m^2 K)]	40
Initial circuit temperature, T_0 [°C]	30
Temperature surrounding the chip, T_∞ [°C]	18
Heat generated in Si die (full streaming quality), Q [W/m^3]	$5 \cdot 10^7$
Heat generated in Si die (decreased streaming quality), Q [W/m^3]	$3,75 \cdot 10^7$

2 Procedure

Everything in the theory sections is collected from the book *Introduction to the Finite Element Method*, written by Niels Ottosen and Hans Petersson, 1992. Information about the problem is from the PDF *Assignment in The Finite Element Method, 2020* written by the Division of Solid Mechanics, LTH, 2020.

2.1 Stationary temperature distribution

2.1.1 Theory

To solve the stationary temperature distribution an FE formulation needed to be derived. In this case it was derived from the weak form of two-dimensional heat flow, and then translated to three dimensions through multiplication with the thickness of the chip. The weak form is given by equation 2 where v is an arbitrary weight function, t is the thickness of the body, $\boldsymbol{D}$ is the constitutive matrix, T is the temperature, h is a constant flux on the boundary $\mathscr{L}_\text{h}$, q_n is the flux on the boundary $\mathscr{L}_\text{g}$ where the temperature is known as $T = g$ and finally Q is the internally generated heat per unit time and unit volume.

$$\int_A (\nabla v)^T t\boldsymbol{D}\nabla T \mathrm{d}A = -\int_{\mathscr{L}_h} vht\mathrm{d}\mathscr{L} - \int_{\mathscr{L}_g} vq_n t\mathrm{d}\mathscr{L} + \int_A vQt\mathrm{d}A \tag{2}$$

The temperature of the body T is approximated using the nodal temperatures in the array $\boldsymbol{a}$ and the global shape functions $\boldsymbol{N}$ as described in equation 3. Introducing $\boldsymbol{B}$ as the gradient of $\boldsymbol{N}$ generates the expression in equation 4, as $\boldsymbol{a}$ being the nodal temperatures doesn't vary element wise in space.

$$T = \boldsymbol{N}\boldsymbol{a}, \qquad \boldsymbol{N} = [N_1 \ N_2 \ \dots \ N_n], \quad \boldsymbol{a} = [T_1 \ T_2 \ \dots \ T_n]^T \tag{3}$$

$$\boldsymbol{B} = \nabla \boldsymbol{N}, \quad \Longrightarrow \quad \nabla T = \nabla \boldsymbol{N}\boldsymbol{a} = \boldsymbol{B}\boldsymbol{a} \tag{4}$$

Moreover, the weight function v is approximated using the Galerkin method resulting in equation 5, where $\boldsymbol{c}$ can be chosen arbitrarily. Since it's chosen using the global shape functions the gradient of v can also be described using $\boldsymbol{B}$.

$$v = \boldsymbol{N}\boldsymbol{c}, \quad \Longrightarrow \quad \nabla v = \nabla \boldsymbol{N}\boldsymbol{c} = \boldsymbol{B}\boldsymbol{c} \tag{5}$$

Inserting 3, 4 and 5 as well as using that $v = v^T$ into the weak form in equation 2 gives

$$\boldsymbol{c}^{\mathrm{T}}\left(\int_A \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{B}t\mathrm{d}A\boldsymbol{a} = -\boldsymbol{c}^{\mathrm{T}}\int_{\mathscr{L}_h}\boldsymbol{N}^{\mathrm{T}}ht\mathrm{d}\mathscr{L} - \boldsymbol{c}^{\mathrm{T}}\int_{\mathscr{L}_g}\boldsymbol{N}^{\mathrm{T}}q_n t\mathrm{d}\mathscr{L} + \boldsymbol{c}^{\mathrm{T}}\int_A \boldsymbol{N}^{\mathrm{T}}Qt\mathrm{d}A\right),$$

which since $\boldsymbol{c}$ is an arbitrary constant gives the final FE formulation in equation 6.

$$\left(\int_A \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{B}t\mathrm{d}A\right)\boldsymbol{a} = -\int_{\mathscr{L}_h}\boldsymbol{N}^{\mathrm{T}}ht\mathrm{d}\mathscr{L} - \int_{\mathscr{L}_g}\boldsymbol{N}^{\mathrm{T}}q_n t\mathrm{d}\mathscr{L} + \int_A \boldsymbol{N}^{\mathrm{T}}Qt\mathrm{d}A \tag{6}$$

Equation 6 is equal to equation 7 when defining the matrices as presented in (8).

$$\boldsymbol{K}\boldsymbol{a} = \boldsymbol{f}_{\mathrm{b}} + \boldsymbol{f}_{\mathrm{l}} \tag{7}$$

$$\begin{aligned}
\boldsymbol{K} &= \int_A \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{B}t\mathrm{d}A \\
\boldsymbol{f}_{\mathrm{b}} &= -\int_{\mathscr{L}_h}\boldsymbol{N}^{\mathrm{T}}ht\mathrm{d}\mathscr{L} - \int_{\mathscr{L}_g}\boldsymbol{N}^{\mathrm{T}}q_n t\mathrm{d}\mathscr{L} \\
\boldsymbol{f}_{\mathrm{l}} &= \int_A \boldsymbol{N}^{\mathrm{T}}Qt\mathrm{d}A
\end{aligned} \tag{8}$$

The chip has two different types of boundary conditions, convection and isolation. Therefore there is no $\mathscr{L}_{\mathrm{h}}$ boundary, meaning there is no boundary condition with a constant temperature, leaving only the integral over $\mathscr{L}_{\mathrm{g}}$ in equation 8. On the top boundary, as showed in figure 1, Newton convection gives the flux $q_n = \alpha_c(\boldsymbol{T} - T_\infty)$

on the boundary $\mathscr{L}_\mathrm{c}$, a subboundary of $\mathscr{L}_\mathrm{g}$. On the remainder of the boundary the flux is equal to zero, $q_n = 0$, and therefore the boundary force vector only is only contributed to by the convection boundary. This results in

$$\begin{aligned}\boldsymbol{f}_\mathrm{b} &= -\int_{\mathscr{L}_g} \boldsymbol{N}^\mathrm{T} q_n t \mathrm{d}\mathscr{L} = -\int_{\mathscr{L}_c} \boldsymbol{N}^\mathrm{T} \alpha_c (\boldsymbol{T} - T_\infty) t \mathrm{d}\mathscr{L} = [\boldsymbol{T} = \boldsymbol{N}\boldsymbol{a}] = \\ &= -\int_{\mathscr{L}_c} \boldsymbol{N}^\mathrm{T} \boldsymbol{N} \alpha_c t \mathrm{d}\mathscr{L} \boldsymbol{a} + \int_{\mathscr{L}_c} T_\infty \alpha_c t \mathrm{d}\mathscr{L}\end{aligned}$$

which now contains the $\boldsymbol{a}$ matrix. Therefore it both has a contribution to the boundary force vector but also to the $\boldsymbol{K}$ matrix. This give the new formulation of equation 7 in equation 9, with the matrices defined in (10).

$$(\boldsymbol{K} + \boldsymbol{K}_\mathrm{c})\,\boldsymbol{a} = \boldsymbol{f}_\mathrm{b} + \boldsymbol{f}_\mathrm{l} \tag{9}$$

$$\begin{aligned}\boldsymbol{K} &= \int_A \boldsymbol{B}^\mathrm{T} \boldsymbol{D} \boldsymbol{B} t \mathrm{d}A \\ \boldsymbol{K}_\mathrm{c} &= \int_{\mathscr{L}_c} \boldsymbol{N}^\mathrm{T} \boldsymbol{N} \alpha_c t \mathrm{d}\mathscr{L} \\ \boldsymbol{f}_\mathrm{b} &= \int_{\mathscr{L}_c} T_\infty \alpha_c t \mathrm{d}\mathscr{L} q_n t \mathrm{d}\mathscr{L} \\ \boldsymbol{f}_\mathrm{l} &= \int_A \boldsymbol{N}^\mathrm{T} Q t \mathrm{d}A\end{aligned} \tag{10}$$

2.1.2 Procedure in Matlab

Due to symmetry only half of the chip was handled in the analysis. The chip was divided into triangular elements using the Matlab PDE tool resulting in the mesh in figure 2. Subdomains were created in the areas where the chip consisted of different materials in order to be able to handle them differently. The resulting edge and subdomain numbering can be seen in figure 3. From the PDE tool matrices were directly extracted and imported into Matlab, containing the node coordinates, edge and element numbering, which edges and elements were connected to which nodes as well as how the triangle elements were divided into subdomains.

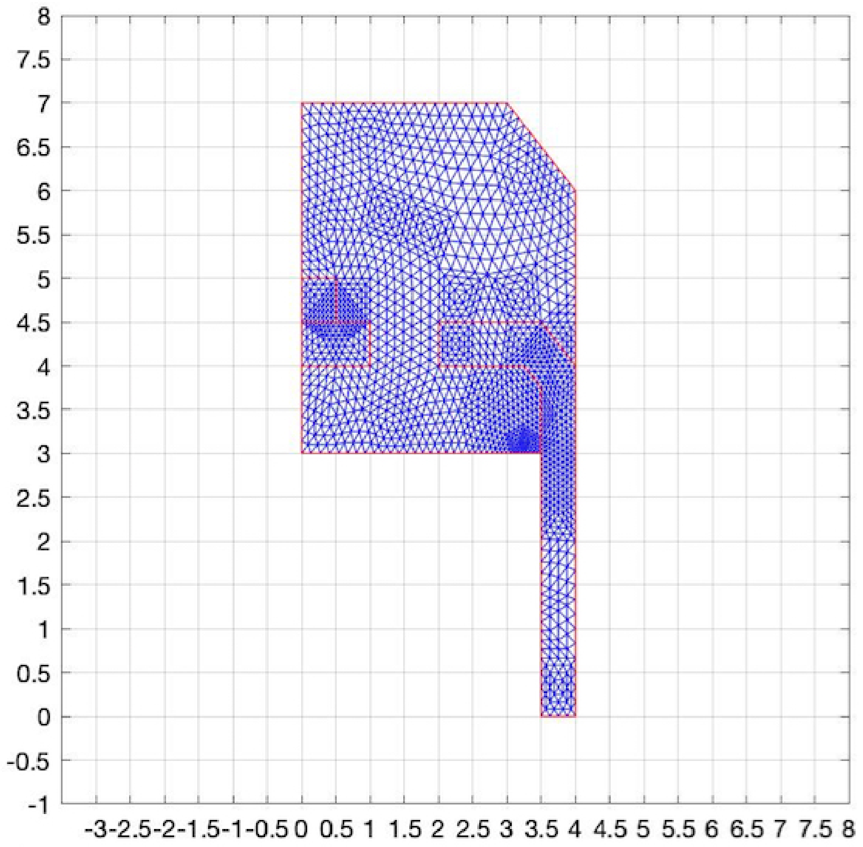


Figure 2: Triangle element mesh.

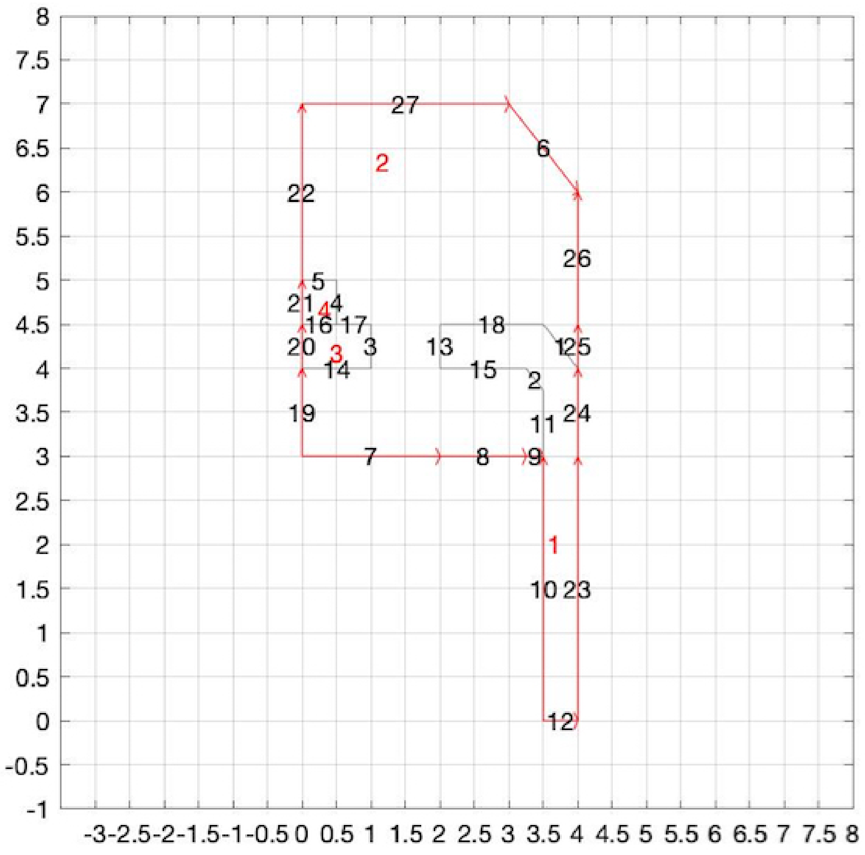


Figure 3: Chip geometry and division into subdomains.

The matrices in (10) together with equation 9 were used to solve the stationary temperature distribution in the chip. The matrices were calculated element wise and then assembled according to the edof matrix which was extracted from the PDE tool. The function flw2te.m in CALFEM was used to calculate the $\boldsymbol{K}_{\mathrm{e}}$ matrix. From figure 3 it was deducted that the convection boundary, $\mathscr{L}_{\mathrm{c}}$, was composed of the edge segments 6, 25, 26 and 27. The nodes connected to each boundary segment were then used to derive the $\boldsymbol{K}_{\mathrm{ce}}$ matrix using the plantml.m function, and to derive the $\boldsymbol{f}_{\mathrm{be}}$ matrix using the assemfb.m function, see chapter 5.2. Heat is only generated in the silicon die, corresponding to subdomain 4 in figure 3. Therefore only the elements in subdomain 4 contributes to the $\boldsymbol{f}_{\mathrm{l}}$ matrix. The stationary temperature distribution is given by the solution to equation 9, which in Matlab was done by using solveq.m.

2.2 Transient temperature distribution

2.2.1 Theory

The strong form of general heat flow is given in equation 11. Here c is the constant heat capacity and ρ is the density. From this the weak form of transient heat flow is derived by multiplying with an arbitrary weight function v and using the Green-Gauss theorem, resulting in equation 12.

$$c\rho\dot{T} + div(\boldsymbol{q}) - Q = 0 \tag{11}$$

$$\int_{\mathscr{L}} v\boldsymbol{q}^T\boldsymbol{n}td\mathscr{L} - \int_A (\nabla v)^T \boldsymbol{q}tdA - \int_A vQtdA + \int vc\rho\dot{T}tdA = 0 \tag{12}$$

To obtain the FE formulation for the transient heat flow the same procedure is followed as with the stationary temperature model. The weight function is chosen to be $v = \boldsymbol{N}(x)\boldsymbol{c}$ according to the Galerkin method, and the temperature in each node is represented by $T = \boldsymbol{N}(x)\boldsymbol{a}(t)$ where $\nabla\boldsymbol{N} = \boldsymbol{B}$. Also making use of Fourier's law, $\boldsymbol{q} = -\boldsymbol{D}\nabla\boldsymbol{T}$, equation 12 can be rewritten as equation 13.

$$\boldsymbol{c}^T\left(\int_{\mathscr{L}} \boldsymbol{N}^T q_{\text{n}}td\mathscr{L} + \int_A \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}tdA\boldsymbol{a} - \int_A \boldsymbol{N}^TQtdA + \int_A \boldsymbol{N}^Tc\rho\boldsymbol{N}tdA\dot{\boldsymbol{a}}\right) = 0 \tag{13}$$

Equation 13 is equal to equation 14 after defining the matrices in (15).

$$\boldsymbol{C}\dot{\boldsymbol{a}} + \boldsymbol{K}\boldsymbol{a} = \boldsymbol{f}_b + \boldsymbol{f}_l \tag{14}$$

$$\begin{aligned}
\boldsymbol{C} &= \int_A \boldsymbol{N}^Tc\rho\boldsymbol{N}tdA \\
\boldsymbol{K} &= \int_A \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}tdA \\
\boldsymbol{f}_{\text{b}} &= -\int_{\mathscr{L}} \boldsymbol{N}^Tq_{\text{n}}td\mathscr{L} \\
\boldsymbol{f}_{\text{l}} &= \int_A \boldsymbol{N}^{\text{T}}Qt\text{d}A
\end{aligned} \tag{15}$$

To solve the transient heat distribution, in other words to solve equation 14, we are now looking to solve a differential equation. To solve this numerically, the time derivative is approximated using the definition of derivation in equation 16.

$$\dot{\boldsymbol{a}} = \frac{\boldsymbol{a}(t_{n+1}) - \boldsymbol{a}(t_n)}{t_{n+1} - t_n} \tag{16}$$

Implicitly approximating $\boldsymbol{a}$ and $\boldsymbol{f}$ and inserting them into equation 14 results in equation 17.

$$\boldsymbol{C}\frac{\boldsymbol{a}(t_{n+1}) - \boldsymbol{a}(t_n)}{t_{n+1} - t_n} + \boldsymbol{K}\boldsymbol{a}(t_{n+1}) = \boldsymbol{f}(t_{n+1}) \tag{17}$$

2.2.2 Procedure in Matlab

Rewriting equation 17 to equation 18 gives the solution to $\boldsymbol{a}$ at $\boldsymbol{t}_{n+1}$. Iterating through the desired time interval then gives the heat distribution at any time. See chapter 5.4 for details of how this was implemented in Matlab.

$$\boldsymbol{a}(t_{n+1}) = [\boldsymbol{C} + \Delta t\boldsymbol{K}]^{-1}[\Delta t\boldsymbol{f}(t_{n+1}) + \boldsymbol{C}\boldsymbol{a}(t_n)] \tag{18}$$

2.3 Stress distributions

The strains in a plane are described by equation 19, where ε are the strains in the material, ε_0 are the initial strains and $\boldsymbol{D}$ is given by (20) in which E is Young's modulus and ν is Poisson's ratio.

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\varepsilon} - \boldsymbol{D}\boldsymbol{\varepsilon}_0 \tag{19}$$

$$\boldsymbol{D} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix} \tag{20}$$

The initial strains, ε_0, are the strains for zero stresses and can be caused by thermal expansion. In the case of thermoelasticity, ε_0 is given by (21). The temperature increase is calculated as the average temperature increase of an element, i.e.

$$\Delta T = \frac{1}{3}\sum_{i=1}^{3} T_i - T_0$$

where T_i are the node temperatures and T_0 is the initial temperature.

$$\varepsilon_0 = (1+\nu)\alpha\Delta T\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \tag{21}$$

The relationship between the displacement vector $\boldsymbol{u}$ and the strain vector $\boldsymbol{\varepsilon}$ is presented in equation 22.

$$\boldsymbol{\varepsilon} = \tilde{\boldsymbol{\nabla}}\boldsymbol{u} \tag{22}$$

In (23) the form of $\boldsymbol{u}$, $\tilde{\boldsymbol{\nabla}}$ and $\boldsymbol{\varepsilon}$ are presented for plain strain conditions.

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} \qquad \tilde{\boldsymbol{\nabla}} = \begin{bmatrix} \frac{\partial}{\partial \mathrm{x}} & 0 \\ 0 & \frac{\partial}{\partial \mathrm{y}} \\ \frac{\partial}{\partial \mathrm{y}} & \frac{\partial}{\partial \mathrm{x}} \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} \mathrm{u}_x \\ \mathrm{u}_y \end{bmatrix} \tag{23}$$

The derivation of the FE formulation for two-dimensional thermal expansion problems is similar to the derivation of the FE formulation for temperature distributions, shown in chapter 2.1.1. In equation 24 the weak form for the displacements in a plane is presented.

$$\int_A (\tilde{\boldsymbol{\nabla}}\boldsymbol{v})^T t dA = \oint_{\mathscr{L}} \boldsymbol{v}^T \boldsymbol{t} t d\mathscr{L} + \int_A \boldsymbol{v}^T \boldsymbol{b} t dA \tag{24}$$

Approximating the displacement vector as $\boldsymbol{u} = \boldsymbol{N}\boldsymbol{a}$, the weight function as $\boldsymbol{v} = \boldsymbol{N}\boldsymbol{c}$ as well as approximating the $\boldsymbol{B}$ matrix as $\boldsymbol{B} = \tilde{\boldsymbol{\nabla}}\boldsymbol{N}$ and inserting this into the weak form in equation 24 results in equation 25. It was used that $\boldsymbol{c}^T$ can be chosen arbitrarily.

$$\int_A \boldsymbol{B}^T \boldsymbol{\sigma} t dA = \oint_{\mathscr{L}} \boldsymbol{N}^T \boldsymbol{t} t d\mathscr{L} + \int_A \boldsymbol{N}^T \boldsymbol{b} t dA \tag{25}$$

The constitutive law presented in equation 19 together with the relationship presented in equation 22 and $\boldsymbol{B} = \tilde{\boldsymbol{\nabla}}\boldsymbol{N}$ results in equation 26.

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{B}\boldsymbol{a} - \boldsymbol{D}\boldsymbol{\varepsilon_0} \tag{26}$$

As the only non zero force vector is the initial thermal strain vector $\boldsymbol{f}_{\Delta \mathrm{T}}$, equation 26 inserted in equation 25 finally results in the FE formulation in equation 27.

$$(\int_A \boldsymbol{B}^T \boldsymbol{\sigma} t dA)\boldsymbol{a} = \int_A \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{\varepsilon_0} t dA \tag{27}$$

The stiffness matrix $\mathbf{K}$ and the initial thermal strain vector $\boldsymbol{f}_{\Delta \mathrm{T}}$ are defined in (28).

$$\boldsymbol{K} = \int_A \boldsymbol{B}^T \boldsymbol{\sigma} t dA \qquad \boldsymbol{f}_{\Delta \mathrm{T}} = \int_A \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{\varepsilon_0} t dA \tag{28}$$

The von Mises stress is given by (29) where $\boldsymbol{\sigma} = \begin{bmatrix}\sigma_{xx} & \sigma_{yy} & \tau_{xy}\end{bmatrix}^T$ and $\sigma_{zz} = \nu(\sigma_{xx} + \sigma_{yy}) - \alpha E \Delta T$.

$$\sigma_{eff} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{xx}\sigma_{zz} - \sigma_{yy}\sigma_{zz} + 3\tau_{xy}^2 + 3\tau_{xz}^2 + 3\tau_{yz}^2} \qquad (29)$$

2.3.1 Procedure in Matlab

The script for calculating the displacements due to thermal expansion can be found in chapter 5.6. As the degrees of freedom has increased from one to two, a new edof matrix was used for the last task.

First ΔT was calculated for each element by taking the mean value of all three nodes in an element and then subtracting T_0. By looping through the elements, $\boldsymbol{D\varepsilon_0}$ was calculated with material- and ΔT depending parameters, see equation 20 and 21. As the circuit was mounted on a PCB board, there could not be displacements in the x- and y-direction on that boundary and the corresponding degrees of freedom were given the value 0. The same procedure was used to fixate the displacements in the x-direction at the symmetry line. The rest of the displacements were left as unknown.

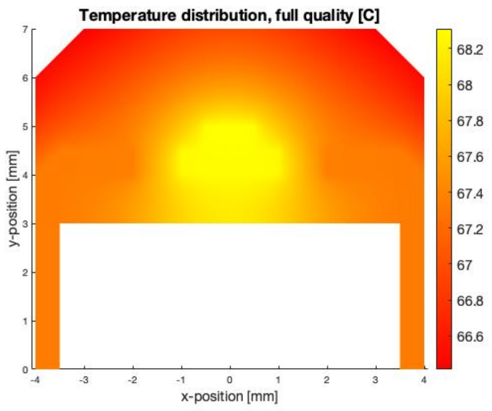
The element stiffness matrices $\boldsymbol{K}_{\mathrm{e}}$ and thermal force vectors $\boldsymbol{f}_{\Delta \mathrm{Te}}$ were calculated with the CALFEM functions plante.m and plantf.m respectively and then assembled to a stiffness matrix $\boldsymbol{K}$ and thermal force vector $\boldsymbol{f}_{\Delta \mathrm{T}}$. With solveq.m and extract.m the displacement field was calculated and plotted with a magnifying factor of 100.

The full Matlab script can be found in chapter 5.7. By subtracting $\boldsymbol{D\varepsilon_0}$ from the stress, calculated with the function plants.m, the resulting stress matrix was found. The effective von Mises stress, see equation 29, was calculated from the stress matrix. To calculate the maximum stress the Matlab function max() was used. The von Mises stress for each node was calculated with Matlab function extract.m and then plotted.

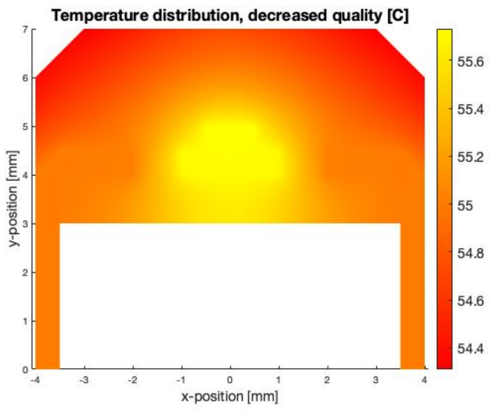
3 Results

3.1 Stationary temperature distribution

The stationary temperature distribution in the circuit for normal streaming quality is presented in figure 4a. The maximum temperature was found to be 68,31°C, located in the silicon die and in the copper piece closest to the silicon. The heat distribution in the chip with a decreased streaming quality of 25% is shown in figure 4b. It has a maximum temperature of 55,73°C.



(a) Stationary temperature distribution, full streaming quality.

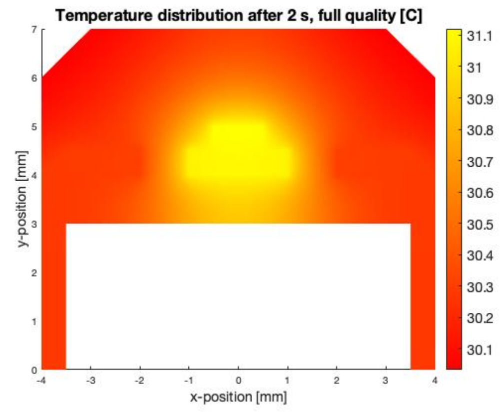


(b) Stationary temperature distribution, 25 reduced streaming quality.

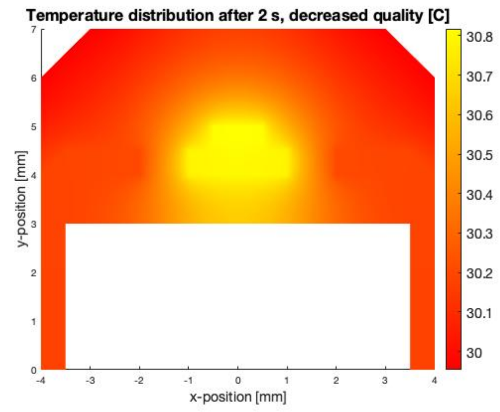
Figure 4: Stationary temperature distribution.

3.2 Transient temperature distribution

In figure 5, 6 and 7 the temperature distributions after 2 seconds, 100 seconds and 300 seconds are plotted, with full streaming quality next to a 25% reduced streaming quality. Maximum temperature after 20 minutes of full streaming quality is 68,26°C and maximum temperature for decreased streaming quality is 55,70°C.

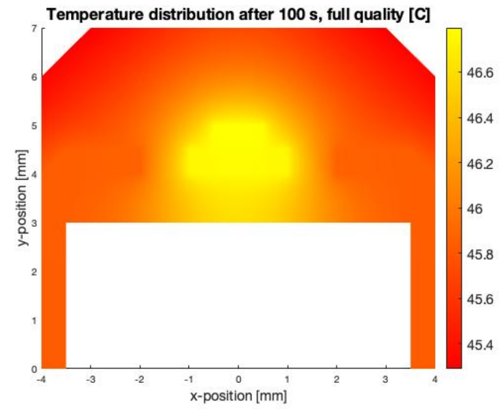


(a) Temperature distribution after 2 s, full streaming quality.

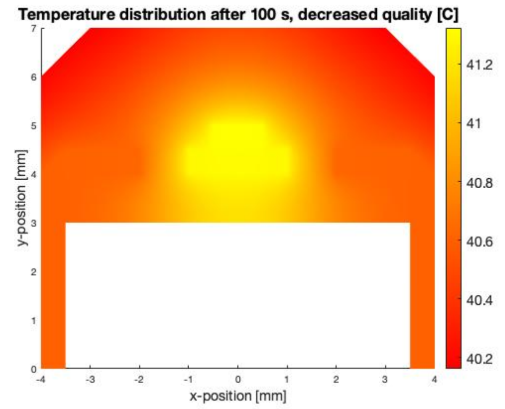


(b) Temperature distribution after 2 s, reduced streaming quality.

Figure 5: Temperature distribution after 2 seconds.

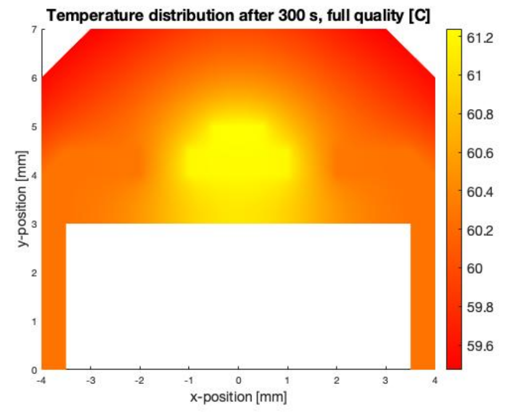


(a) Temperature distribution after 100 s, full streaming quality.

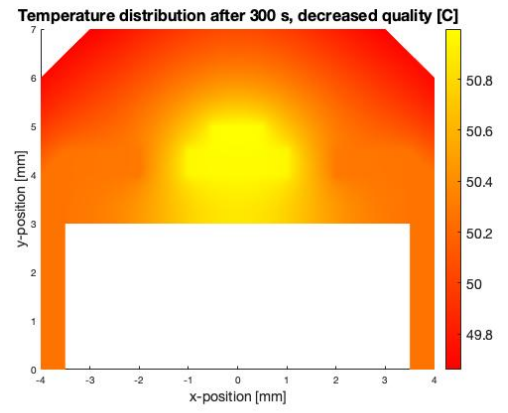


(b) Temperature distribution after 100 s, reduced streaming quality.

Figure 6: Temperature distribution after 100 seconds.



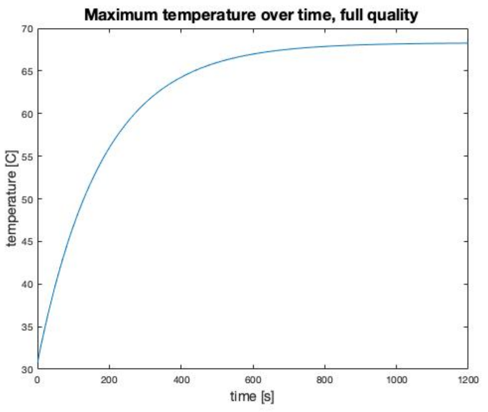
(a) Temperature distribution after 300 s, full streaming quality.



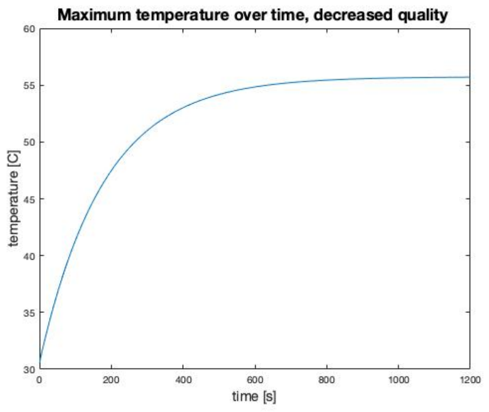
(b) Temperature distribution after 300 s, reduced streaming quality.

Figure 7: Temperature distribution after 300 seconds.

The maximum temperature is plotted as a function of time for full streaming quality in figure 8a and for decreased streaming quality in figure 8b.



(a) Maximum temperature plotted over time, full streaming quality.

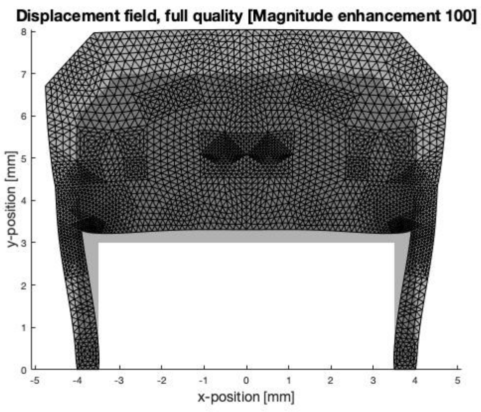


(b) Maximum temperature plotted over time, decreased streaming quality.

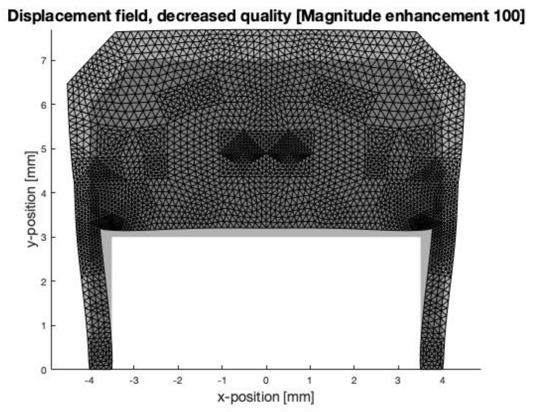
Figure 8: Maximum temperature plotted over 20 minutes.

3.3 Thermal expansion

Displacement fields due to thermal expansion are presented in figure 9 where figure 9a shows the displacement field for the circuit with full streaming quality and figure 9b shows the displacement field for the circuit with a 25% reduction in streaming quality.



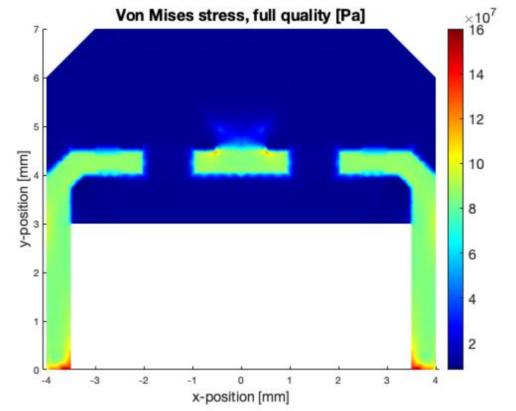
(a) Displacement field for full streaming quality.



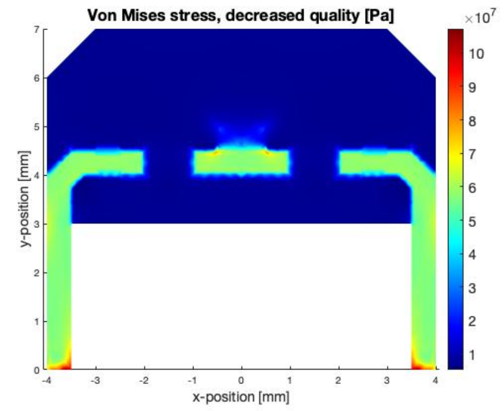
(b) Displacement field for decreased streaming quality.

Figure 9: Displacement fields due to thermal expansion.

The Von Mises stress distribution for full streaming quality and reduced streaming quality is presented in figure 10a and 10b. At (x = ± 3,5, y = 0) the maximum stress is found with a value of $1{,}60\cdot10^8$ Pa for full streaming quality and with a value of $1{,}07\cdot10^8$ Pa for reduced streaming quality.



(a) Von Mises stress distribution for full streaming quality.



(b) Von Mises stress distribution for decreased streaming quality.

Figure 10: Von Mises stress distribution.

4 Discussion

4.1 Stationary temperature distribution

The maximum temperature, as well as the temperature over all, is higher for the full streaming quality circuit than for the circuit with reduced streaming quality, as seen in the results in figure 4a and 4b. This was expected due to the proportional relationship between heat and streaming quality. The highest temperatures were found in the silicon die and the copper bordering the silicon. The high temperature in the silicon die simply comes from that the heat is generated there. Why the copper attached to the silicon has higher temperature than the surrounding silver epoxy can be explained with the copper's much higher thermal conductivity. The other copper parts don't show the same high temperatures because it's separated from the silicon by an insulating layer of silver epoxy. The temperature is the lowest in the silver epoxy in the upper corners, which shows the impact of the convection in the circuit, as the surrounding temperature lower the temperature in the circuit through convection. To improve the model, a finer mesh could be used.

4.2 Transient temperature distribution

The higher temperature in the circuit with full streaming quality, in comparison with the temperature in the circuit with reduced streaming quality, is confirmed again in the temperature distribution plots at the time 2 seconds, 100 seconds and 300 seconds in figure 5, 6 and 7. However, the difference in temperature between the a- and b-plots increases over time which can be explained with the figures 8a and 8b. In those figures it is shown that a higher heat Q results in a bigger incline as well as a equilibrium at a higher temperature.

Both in the temperature distribution plots 5, 6 and 7 it is shown that the biggest temperature change occurs in the beginning. After 300 seconds the temperature curve starts to flatten and the maximum temperature is closer to the stationary- than the initial temperature. 20 minutes after start the maximum temperature is only 0,05°C and 0,03°C below the stationary maximum temperature for the full streaming quality and the decreased streaming quality.

To more clearly illustrate the temperature change, a fixed temperature scale could be used. An improved model could as well have smaller time steps.

4.3 Thermal expansion

Figure 9a and 9b shows that the increased temperature results in a substantial expansion, both in the case of full streaming quality as well as when the quality was reduced with 25%. The expansion was expected in both cases, as the thermal increase results in a bigger ε_0 in equation 19. ε_0 also depends on the thermal expansion coefficient, which is 10 times bigger for silver epoxy in table 1 than the expansion coefficient for silicon and copper. As desired, the resulting displacements in figure 9a and 9b show a bigger displacement in the silver epoxy domains. The accuracy of the displacements is dictated by the fineness of the mesh, which means that with a very powerful computer the mesh could have been made much more detailed giving a more exact result.

In figure 10, it can be seen that the biggest stress was found in the copper parts of the chip. The maximum stresses were found on the inside of the copper "legs", where the copper is attached to the PCB board. This is because the copper is unable to expand or move as the "legs" bend because of the thermal expansion, resulting in much stress in the attachments of the chip. The maximum stress was 33% lower in the chip with the reduced quality, since the chip didn't expand as much. In figure

10 it can also be observed that the silver epoxy experiences the least stress, and that the silicon also experiences relatively little stress, but with a peak stress closest to where it is attached to one of the copper pieces. This is a reasonable result since the two have different values of E and α who together dictate the stress. Sources of error could arise from the mesh division, if crucial parts of the chip doesn't have enough resolution. This is substantially avoided by dividing the chip into subdomains and thus for example evade the situation where an element contains two different materials. Once again, the result would have been more accurate with a finer mesh.

5 Matlab code

5.1 Stationary temperature distribution

```
%% Stationary temperature distribution
%%%% Parameters %%%%
coord = p';                                     % coordinates from p
enod=t(1:3,:)';                                 % nodes of elements
nelm=size(enod,1);                              % number of elements
nnod=size(coord,1);                             % number of nodes
dof=(1:nnod)';                                  % degrees of freedom
ndof = length(dof);                             % number degrees of freedom
D = eye(2);                                     % empty constitutive matrix
K = zeros(ndof);                                % empty stiffness matrix
fl = zeros(ndof,1);                             % empty load vector
fb = zeros(ndof,1);                             % empty load boundary vector
eq = zeros(1,nelm);                             % empty heat supply per unit vector
k = zeros(1,nelm);                              % empty thermal conductivity vector
Tinf = 18;                                      % temperature far away
alphac = 40/(1000^2);                           % thermal transmittance
A = 10;                                         % thickness
Q = (5 * 10^7)*10^(-9);                          % heat supply (Si)
%Q = 0.75*Q;                                    % Netflix decreases video quality with 25%

%%%% Edof from mesh %%%%
dof_S=[(1:nnod)',(nnod+1:2*nnod)'];
for ie=1:nelm
    edof_S(ie,:)=[ie dof_S(enod(ie,1),:), dof_S(enod(ie,2),:),dof_S(enod(ie,3),:)];
    edof(ie,:)=[ie,enod(ie,:)];
end

%%%% Convection segments %%%%
er = e([1 2 5],:);                        % Reduced e
conv_segments = [6 25 26 27];             % Chosen boundary segments
edges_conv = [];
for i = 1:size(er,2)
    if ismember(er(3,i),conv_segments)
        edges_conv = [edges_conv er(1:2,i)];    % edges with convection
    end
end

%%%% Coordinates and structure plot %%%%
[ex,ey]=coordxtr(edof,coord,dof,3);
%eldraw2(ex,ey,[1 4 0],edof(:,1)) % Plot of structure

%%%% Heat supply per unit vector - eq %%%%
for i = 1:length(t)
    if t(4,i) == 4
        eq(1,i) = Q;
    end
end

%%%% Thermal conductivity vector - k %%%%
for i = 1:length(t)
    if t(4,i) == 3 || t(4,i) == 1
        k(1,i) = 385*0.001;
    elseif t(4,i) == 2
        k(1,i) = 5*0.001;
    else
        k(1,i) = 149*0.001;
    end
end

%%%% Stiffnes matrix K and load vector fl %%%%
for elnr=1:nelm
    [Ke, fle] = flw2te(ex(elnr,:), ey(elnr,:), A, k(1,elnr)*D, eq(1,elnr));
    indx = edof(elnr,2:end);
    K(indx,indx) = K(indx,indx)+Ke;
    fl(indx) = fl(indx) + fle;
end

%%%% Boundary conditions bc, only used before convection is added %%%%
bc = [edges_conv(1,:) edges_conv(2,:)];
bc = unique(bc)';
bc(:,2) = 18;

%%%% Total load vector f %%%%
fb = A*assemfb(zeros(ndof,1),edges_conv,coord,alphac,Tinf);     % boundary load vecto
f = fl + fb;

%%%% Total stiffness matrix %%%%
Kc = A*assemconv(zeros(ndof),edges_conv,coord,alphac);          % stiffness matrix fr
K = K + Kc;


%%%% Solver A
%%%% Solve and extract %%%%
a = solveq(K, f);
et = extract(edof, a);
maxa = max(a); % max temperature

%%%% Expand coordinates %%%%
exexp = [-flip(ex) ; ex];
eyexp = [flip(ey) ; ey];
etexp = [flip(et) ; et];

%%%% Plot result %%%%
patch(exexp',eyexp',etexp','EdgeColor','none')
title('Temperature distribution [C]')
colormap(autumn);
colorbar;
xlabel('x-position [mm]')
ylabel('y-position [mm]')
axis equal
```

5.2 assemfb.m

```
    function f = assemfb(f,edges_conv,coord,alphac,Tinf)
fbe = [0.5 ; 0.5].*alphac.*Tinf; % element boundary load vector

for ind = 1:length(edges_conv)
    dx = coord(edges_conv(1,ind),1)-coord(edges_conv(2,ind),1);
    dy = coord(edges_conv(1,ind),2)-coord(edges_conv(2,ind),2);
    L = sqrt(dx^2 + dy^2);  % calculating length
    f(edges_conv(:,ind)) =  f(edges_conv(:,ind)) + L.*fbe; % assembling element bound
end
```

5.3 assemconv.m

```
    function Kc = assemconv(Kc,edges_conv,coord,alphac)
Kce = [2 1;1 2].*alphac./6; % element stiffness matrix

for ind = 1:length(edges_conv)
    dx = coord(edges_conv(1,ind),1)-coord(edges_conv(2,ind),1);
    dy = coord(edges_conv(1,ind),2)-coord(edges_conv(2,ind),2);
    L = sqrt(dx^2 + dy^2); % calculating length
    Kc(edges_conv(:,ind),edges_conv(:,ind)) =  Kc(edges_conv(:,ind),edges_conv(:,ind)

end
```

5.4 Transient temperature distribution

```
%% Transient temperature distribution

%%%% Parameters %%%%
rho = zeros(1,nelm);                        % empty density vector
cp = zeros(1,nelm);                         % empty specific heat vector
C = zeros(ndof);                            % empty element matrix C
time = 20*60;                               % time from start in seconds
```

```
aold = 30.*ones(ndof,1);                      % start temperature at all nodes
deltat = 1;                                   % time steps
timevec = linspace(deltat, time, time)';     % time vector
anew = zeros(ndof,1);                         % temperature vector
max_a = zeros(time,1);                        % max temperature vector

%%%% Density rho and specific heat vector cp %%%%
for i = 1:length(t)
    if t(4,i) == 3 ||t(4,i)== 1
        rho(1,i) = 8930/10^9;
        cp(1,i) = 386;
    elseif t(4,i) == 2
        rho(1,i) = 2500/10^9;
        cp(1,i) = 1000;
    else
        rho(1,i) = 2530/10^9;
        cp(1,i) = 703;
    end
end

%%%% Element matrix - C %%%%
C = assemc(edof, ex, ey, C, cp, rho, A);

%%%% Implicit time stepping %%%%
for i = deltat:deltat:time
    anew = (C+deltat.*K)\(C*aold + deltat.*f);
    max_a(i,1) = max(anew);
    aold = anew;
end

a_max = max(max_a)      % max temperature


%%%% Extract %%%%
et = extract(edof, anew);

%%%% Expand coordinates %%%%
exexp = [-flip(ex) ; ex];
```

```
eyexp = [flip(ey) ; ey];
etexp = [flip(et) ; et];

%%%% Plot result %%%%
plot(timevec, max_a, '-');                          % plot max temp over time
%patch(exexp',eyexp',etexp','EdgeColor','none');    % plot tempdistribution
title('Maximum temperature, full quality ')
colormap(autumn);
xlabel('time [s]')
ylabel('temperature [C]')
```

5.5 assemc.m

```
    function C = assemc(edof, ex, ey, C, cp, rho, A)

[nie,n]=size(edof);
    t=edof(:,2:n);
    for i = 1:nie
      C(t(i,:),t(i,:)) = C(t(i,:),t(i,:))+A.*plantml(ex(i,:),ey(i,:), rho(1,i)*cp(1,i
    end
```

5.6 Thermal expansion

```
    %% Thermal expansion
clc;

%%%% Parameters %%%%
ndof2 = 2*ndof;                   % number degrees of freedom
Depsilon0 = zeros(3,nelm);        % epsilon0*D
E = [7 165 128].*10^9;            % Youngs modulus [Ag epoxy, Si, Cu]
v = [0.3 0.22 0.36];              % Poissons ratio [Ag epoxy, Si, Cu] *SKA DENNA VA MIN
alpha = [40 2.6 17.6].*10^(-6); % Expansion coefficient [Ag epoxy, Si, Cu]
T0 = 30;                          % start temperature
deltaT = zeros(nelm, 1);          % vector for the elements difference in temperature
K2 = zeros(ndof2);                % stiffness matrix
D2 = zeros(3);                    % D matrix
```

```
ep = [2 A];                        % [p-type thickness]
f2 = zeros(ndof2, 1);              % thermal force vector
DAgEp = E(1)./((1+v(1))*(1-2*v(1))).*[1-v(1) v(1) 0; v(1) 1-v(1) 0 ; 0 0 0.5*(1-2*v(1
DSi = E(2)./((1+v(2))*(1-2*v(2))).*[1-v(2) v(2) 0; v(2) 1-v(2) 0 ; 0 0 0.5*(1-2*v(2))
DCu = E(3)./((1+v(3))*(1-2*v(3))).*[1-v(3) v(3) 0; v(3) 1-v(3) 0 ; 0 0 0.5*(1-2*v(3))
Seff_nod = zeros(nnod,1);          % Von Mises effective stress in the nodes
Seff_el = zeros(nelm,1);           % Von Mises effective stress in an element
sigma = zeros(nelm,3);             % stress

%%%% Delta T %%%%
for elnr = 1:nelm
    deltaT(elnr,1) = (a(t(1,elnr))+a(t(2,elnr))+a(t(3,elnr)))/3 - T0;
end

%%%% D2*epsilon0 %%%%
for elnr = 1:nelm
    if t(4,elnr) == 3 || t(4,elnr) == 1
        Depsilon0(1:2,elnr) =  alpha(3).*E(3).*deltaT(elnr)./(1-2.*v(3));    % Cu
    elseif t(4,elnr) == 2
        Depsilon0(1:2,elnr) =  alpha(1).*E(1).*deltaT(elnr)./(1-2.*v(1));   %Ag epoxy
    else
        Depsilon0(1:2,elnr) = alpha(2).*E(2).*deltaT(elnr)./(1-2.*v(2));    % Si
    end
end

%%%% Boundary conditions %%%%
% PCB board boundary
ed_segments = [12]; % Chosen boundary segments
edges_ed = [];
for i = 1:size(er,2)
    if ismember(er(3,i),ed_segments)
        edges_ed = [edges_ed er(1:2,i)];
    end
end

uniquefreedom1 = unique(edges_ed); % unique degrees of freedom
len = size(uniquefreedom1,1);
for i = 1:len
```

```
    for col = 2:7
        for row = 1:nelm
            if edof_S(row, col) == uniquefreedom1(i)
                if mod(col,2) == 0
                    uniquefreedom1 = [uniquefreedom1; edof_S(row,col+1)];   % fixing
                elseif mod(col,2) ~= 0
                    uniquefreedom1 = [uniquefreedom1; edof_S(row,col-1)];   % fixing

                end
            end
        end
    end
end

% Left boundary in figure
ed_segments = [19 20 21 22]; % Chosen boundary segments
edges_ed = [];
for i = 1:size(er,2)
    if ismember(er(3,i),ed_segments)
        edges_ed = [edges_ed er(1:2,i)];
    end
end

uniquefreedom2 = unique(edges_ed);
len = size(uniquefreedom2,1);

for i = 1:len
    for col = 2:7
        for row = 1:nelm
            if edof_S(row, col) == uniquefreedom2(i)
                if mod(col,2) ~= 0
                    uniquefreedom2(i) = [uniquefreedom2; edof_S(row,col-1)];     % fi
                end
            end
        end
    end
end
```

```
% bc
uniquefreedom1 = unique(uniquefreedom1);
uniquefreedom2 = unique(uniquefreedom2);
uniquefreedomtot = [uniquefreedom1; uniquefreedom2];
bc = zeros(size(uniquefreedomtot,1), 2);

for i = 1:size(uniquefreedomtot,1)
    bc(i,1) = uniquefreedomtot(i);  % all fixated degrees of freedom
end

%%%% Stiffness matrix K2 and thermal force f2 %%%%
for elnr = 1: nelm
    if t(4,elnr) == 3 || t(4, elnr) == 1
        D2 = DCu;
    elseif t(4,elnr) == 2
        D2 = DAgEp;
    else
        D2 = DSi;
    end
    K2e = plante(ex(elnr,:),ey(elnr,:),ep,D2);
    f2e = plantf(ex(elnr,:),ey(elnr,:),ep,Depsilon0(:, elnr)');
    f2 = insert(edof_S(elnr, :), f2, f2e);
    indx = edof_S(elnr,2:end);
    K2(indx,indx) = K2(indx,indx)+K2e;
end

%%%% Solver %%%%
disp = solveq(K2, f2, bc);
ed = extract(edof_S, disp);

%%%% Expand coordinates %%%%
exexp = [-flip(ex) ; ex];
eyexp = [flip(ey) ; ey];

%%%% Calculate displaced coordinates %%%%
mag = 100; % Magnification (due to small deformations)
exd = ex + mag.*ed(:,1:2:end);
exdm =[-flip(exd) ; exd];
```

```
eyd = ey + mag.*ed(:,2:2:end);
eydm = [flip(eyd) ; eyd];
figure()
patch(exexp',eyexp',[0 0 0],'EdgeColor','none','FaceAlpha',0.3)
hold on
patch(exdm',eydm',[0 0 0],'FaceAlpha',0.3)
axis equal
title('Displacement field, full quality [Magnitude enhancement 100]')
xlabel('x-position [mm]')
ylabel('y-position [mm]')
```

5.7 Von Mises stress

```
    %%%% Von Mises %%%%
clc;

%%%% Stress sigma %%%%
for i = 1:nelm
    if t(4,i) == 3 || t(4, i) == 1
        D2 = DCu;
    elseif t(4,i) == 2
        D2 = DAgEp;
    else
        D2 = DSi;
    end
    es = plants(ex(i,:),ey(i,:),ep,D2,ed(i,:));
    sigma(i,:) = es;
end

sigma= sigma - Depsilon0';

%%%% Effective Von Mises stress %%%%
for i = 1:nelm
    if t(4,i) == 3 || t(4, i) == 1
        szz=v(3).*(sigma(i,1) + sigma(i,2)) - alpha(3)*E(3)*deltaT(i);
    elseif t(4,i) == 2
        szz=v(1).*(sigma(i,1) + sigma(i,2)) - alpha(1)*E(1)*deltaT(i);
    else
```

```
        szz=v(2).*(sigma(i,1) + sigma(i,2)) - alpha(2)*E(2)*deltaT(i);
    end
    Seff_el(i,1) = sqrt(sigma(i,1)^2 + sigma(i,2)^2 + szz^2 - sigma(i,1)*sigma(i,2) -
end

%%%% Maximum stress and coordinates %%%%
maxstress = max(Seff_el)
exmax = ex(elementnr,1)
eymax = ey(elementnr,1)

%%%% Node stress %%%%
for i=1:2*nelm
    [c0,c1]=find(edof_S(:,2:4)==i);
    Seff_nod(i,1)=sum(Seff_el(c0))./size(c0,1);
end

Seff_nod = extract(edof,Seff_nod);

%%%% Expand coordinates %%%%
exexp = [-flip(ex) ; ex];
eyexp = [flip(ey) ; ey];
Seff_nod = [flip(Seff_nod) ; Seff_nod];


%%%% Plot result %%%%
patch(exexp',eyexp',Seff_nod','EdgeColor','none');
title('Von Mises stress')
colormap(jet(5000));
colorbar;
xlabel('x-position [mm]')
ylabel('y-position [mm]')
axis equal
```