Analysis of a Heat Sink in Microfluidic Applications

Lorenzo Vecchietti

December 2024

Contents

Problem Description

The study focuses on a silicon chip that integrates cooling channels, with water acting as the coolant to effectively remove the heat generated by a uniform heat source applied to its surface. The chip features an array of cooling channels, each with specific geometric dimensions, and a solid material layer that separates the heat source from the cooling channels. The primary objective of this analysis

is to determine the thermal resistance of the system and identify the optimal configuration of the cooling channels that will achieve the most efficient heat removal from the chip.

Data

Parameter	Value (SI Units)
Width of the chip (W)	0.01 m
Length of the chip (L)	0.01 m
Height of the chip	$5 \cdot 10^{-4}$ m
Heat source intensity	$8 \cdot 10^{3}$ W/m^2
Channel depth (h_{fin})	$4 \cdot 10^{-4}$ m
Solid material thickness	$1 \cdot 10^{-5}$ m
Maximum pressure drop	50 kPa
Water inlet temperature	293.15 K
Silicon thermal conductivity	150 W/m $\cdot$ K
Water thermal conductivity	0.6 W/m $\cdot$ K
Water viscosity	0.001 Pa $\cdot$ s
Water specific heat	4186 J/kg $\cdot$ K

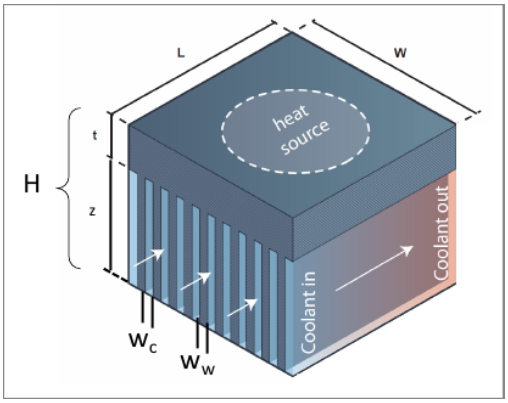

Figure 1: Schematic of the cooling setup

Assumptions

Geometry

We assume that the width of the cooling channels (w_c) is equal to the width of the fins (w_w), i.e., $w_c = w_w$. This assumption leads to the following relationship between the total width of the chip (W) and the number of fins (n):

$$W = (2n + 1) \cdot w_c$$

In this equation, n represents the number of fins.

Nusselt Number

The Nusselt number (Nu) is a dimensionless number that characterizes the convective heat transfer at the boundary between a fluid and a solid. In the laminar flow regime, where the Reynolds number is less than 10^3 ((Re $<$ 10ˆ3)), the Nusselt number remains constant and is typically given by:

$$Nu = 4.36$$

We assume that the flow within each of the microfluidic channels remains laminar, which is generally the case for microchannels with small dimensions.

Heat Transfer Modeling

Heat transfer from the chip to the coolant occurs through three primary mechanisms:

1. **Conduction through the solid material**:

$$R_{cond} = \frac{1}{t \cdot A_{chip}}$$

   where t is the thickness of the solid material and A_{chip} is the heat source area on the chip.

2. **Convection between the channel walls and water**:

$$R_{conv} = \frac{1}{h \cdot A_{wall}} = \frac{1}{\frac{k_f \cdot Nu}{w_c} \cdot \frac{w_c + 2z}{2w_c} \cdot A_{chip}}$$

   where h is the heat transfer coefficient, k_f is the thermal conductivity of the fluid, and z is the height of the channel.

3. **Sensible heat transfer as the water absorbs heat**:

$$R_{heat} = \frac{1}{\rho c_p f}$$

   where ρ is the coolant density, c_p is the specific heat capacity of water, and f is the volumetric flow rate.

The flow rate f is derived using the Hagen-Poiseuille equation for a rectangular channel, assuming that the width of the channel (w_c) is much smaller than the height of the fin (h_{fin}):

$$\Delta P = \frac{12\mu L}{(1 - 0.63 \cdot \frac{w_c}{z}) \cdot n_c \cdot w_c^3 \cdot z} \cdot f$$

where $n_c = n + 1$ represents the total number of channels.

The total thermal resistance is a combination of these three contributions, which allows for the calculation of the chip's surface temperature while ensuring that constraints on pressure drop and coolant flow are respected.

Minimum Channel Width (w_c)

In order to maintain the laminar flow assumption and to ensure that the Hagen-Poiseuille equation is applicable, we require that the channel width w_c remains sufficiently small relative to the fin height h_{fin}. Specifically, the condition:

$$w_c < h_{fin} \quad \Rightarrow \quad \frac{W}{2n+1} < h_{fin}$$

results in the following constraint:

$$n < 0.5 \left(\frac{W}{h_{fin}} - 1 \right) = 12$$

This ensures that the channel width remains small enough for laminar flow, allowing the use of the Hagen-Poiseuille equation to determine the flow rate.

Analysis

Sensible Heat Resistance

The resistance to heat transfer due to sensible heat is inversely proportional to the coolant's density ρ, specific heat capacity c_p, and the volumetric flow rate f. As the flow rate increases, the heat resistance decreases.

The flow rate is constrained by the maximum pressure drop ΔP that the coolant pump can generate. The volumetric flow rate also decreases as the number of fins increases, primarily due to the increased skin friction between the fluid and the channel walls. Furthermore, the viscosity of the coolant limits the flow rate, as illustrated in the figures below.

The resistance associated with sensible heat, denoted as R_{heat}, is a function of the flow rate and is plotted in Figure 4. As expected, the heat resistance

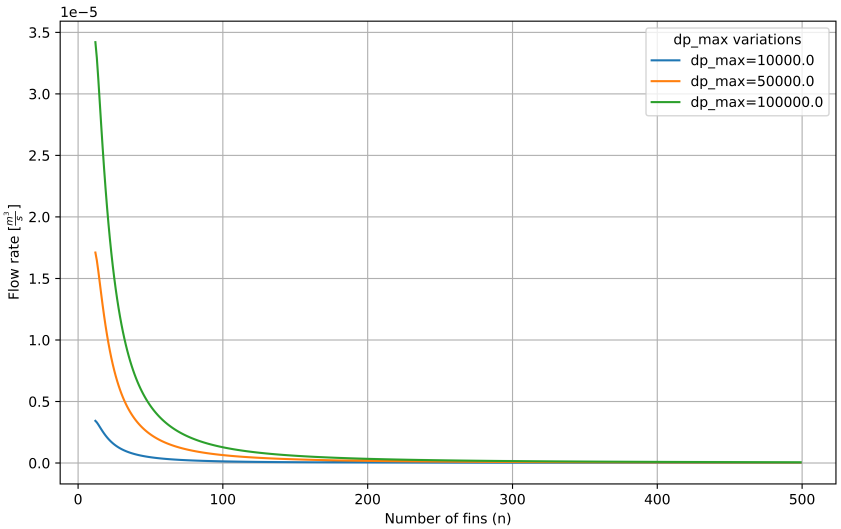

Figure 2: Volumetric flow rate vs. maximum pressure drop and number of fins

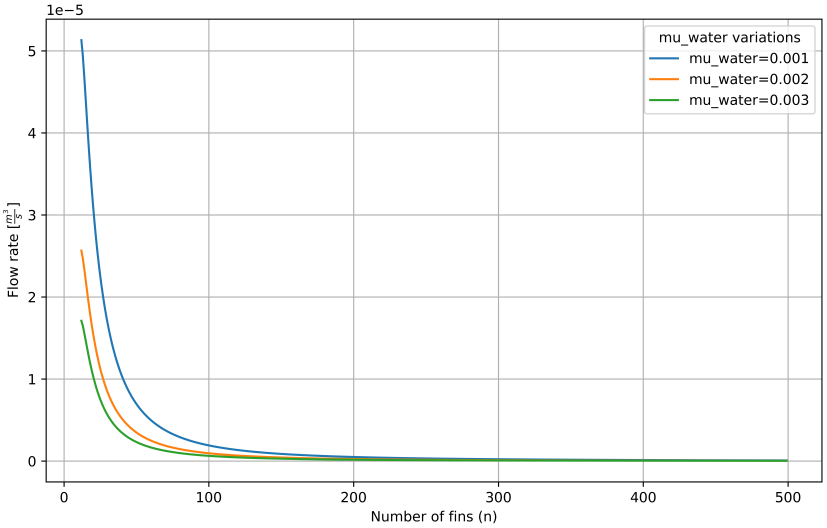

Figure 3: Volumetric flow rate vs. viscosity and number of fins

increases with the number of fins because the flow rate decreases with a higher number of fins.

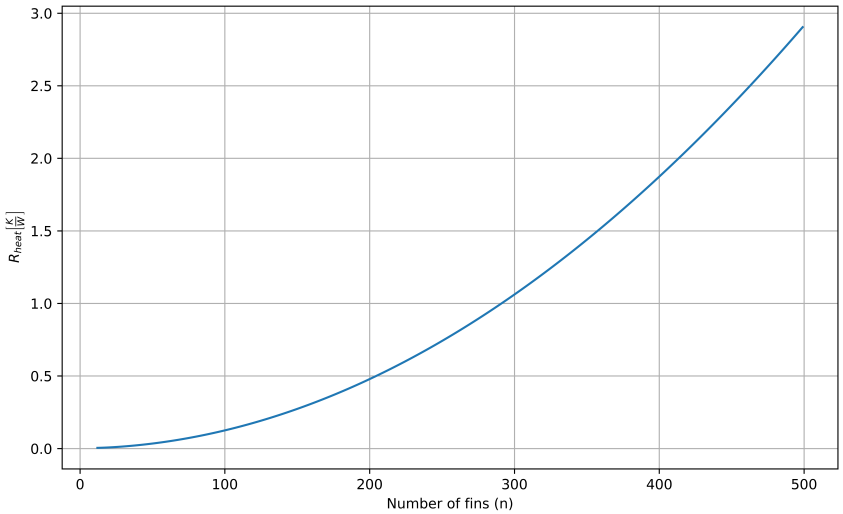

Figure 4: Sensible heat resistance vs. number of fins

Convective Heat Resistance

Convective heat resistance is influenced by two primary factors:

1. The increase in heat transfer area due to the addition of fins.
2. The increase in the heat transfer coefficient as the Reynolds number rises.

Both factors contribute to a reduction in convective resistance as the number of fins increases. The convective resistance R_{conv} is inversely proportional to these factors. As shown in Figure 5, the convective resistance decreases as the number of fins increases.

Conductive Heat Resistance

The conductive resistance, which is modeled based on the thermal conductivity of the solid material, remains constant regardless of the number of fins. Using the provided parameters, the conductive resistance is calculated as:

$$R_{cond} = 6.67 \times 10^3 \, \frac{\mathrm{K}}{\mathrm{W}}$$

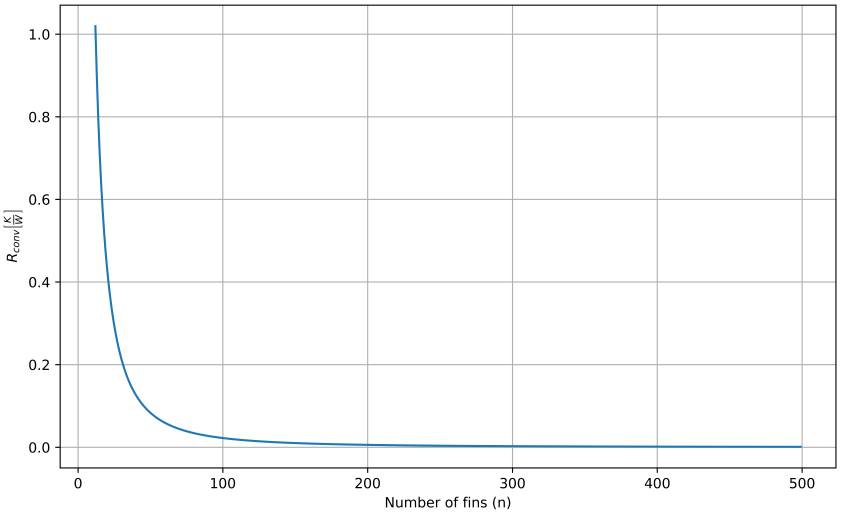

Figure 5: Convective resistance vs. number of fins

Total Resistance

The total thermal resistance is the sum of the conductive, convective, and sensible heat resistances. Initially, as the number of fins increases, the total resistance decreases. However, beyond around 64 fins, the resistance due to sensible heat becomes dominant, causing the total resistance to increase. This behavior is clearly visible in Figure 6, which plots the total resistance as a function of the number of fins.

The temperature of the chip follows a similar trend. As the number of fins increases, the temperature of the chip decreases up to the optimal point, after which the temperature begins to rise again due to the increased resistance.

Discussion

Optimal Number of Fins

From the analysis, it is evident that there is an optimal number of fins at which the total thermal resistance is minimized, resulting in the lowest possible temperature on the chip. For the given configuration, 64 fins yield the best performance, providing the optimal balance between heat dissipation and system constraints.

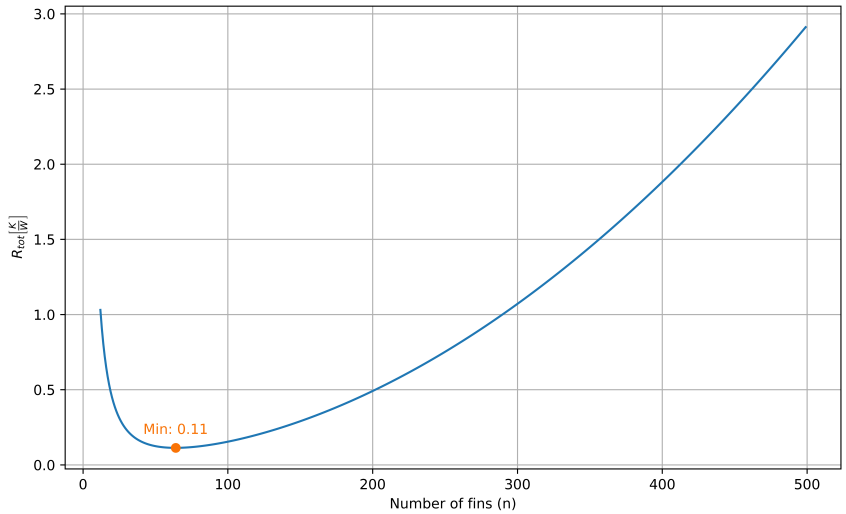

Figure 6: Total resistance vs. number of fins

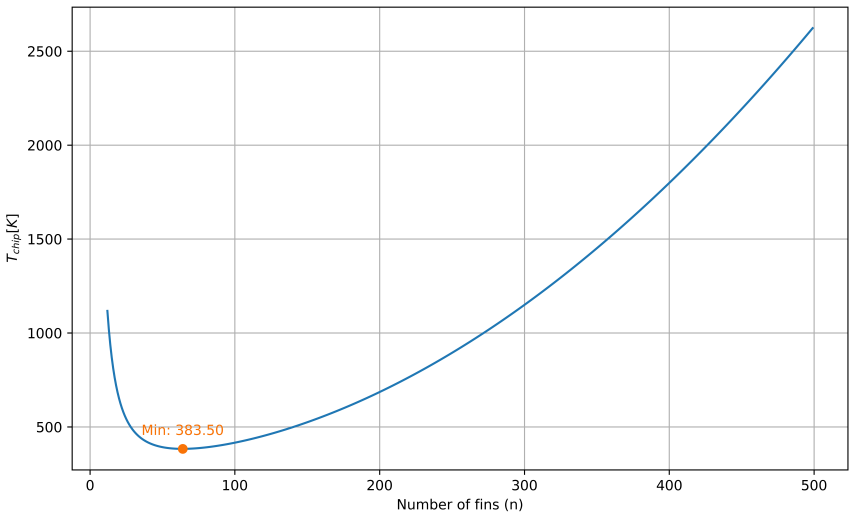

Figure 7: Chip temperature vs. number of fins

Number of fins	Minimum R_{tot}	Channel width	Temperature on the chip
64	$0.113\,\frac{\mathrm{K}}{\mathrm{W}}$	$7.75 \cdot 10^{-5}\,\mathrm{m}$	$383.50\,\mathrm{K}$

Table 2: Optimized heatsink

Influence of Resistance Components

The conductive resistance is relatively small compared to the other two resistance components. For low numbers of fins, the convective resistance dominates, but it is limited by the small heat transfer area. As the number of fins increases, the pump power limitation becomes more significant, and the resistance due to sensible heat increases substantially.

The point of minimal total resistance occurs around 64 fins, where both convective and sensible heat resistances balance each other. The conductive resistance remains negligible throughout the entire process and does not significantly influence the overall system performance.

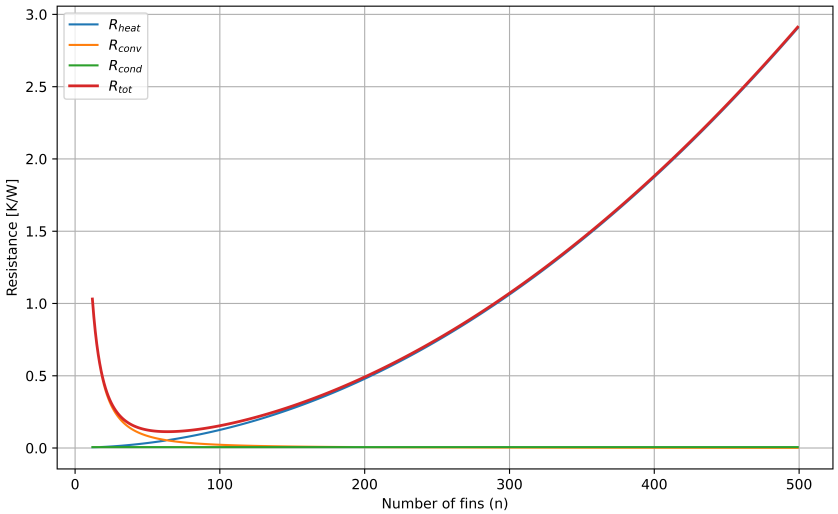

Figure 8: Comparison of resistance components

Effect of Silicon Thermal Conductivity

The influence of silicon's thermal conductivity is minimal compared to the other factors. Since the thermal conductivity of silicon is several orders of magnitude

lower than the thermal conductivity of the coolant and the convective resistance, changes in the silicon's thermal conductivity primarily affect the total resistance curve by shifting it up or down, without altering the optimal number of fins.

Strategies to Improve Heat Extraction

To further enhance heat extraction from the chip, one potential strategy is to explore higher flow rates, which could be achieved by using a more powerful pump. As shown in Figure 2, a more powerful pump would result in a higher flow rate, which, in turn, would lower the sensible heat resistance, especially at higher numbers of fins. However, this approach comes with the potential downside that the pump itself could generate additional heat, which would need to be dissipated.

Alternatively, one could consider using a different coolant with a lower viscosity (μ) while aiming to maintain the same level of thermal conductivity. This could help improve heat transfer while keeping the system's overall resistance manageable, as demonstrated in Figure 9.

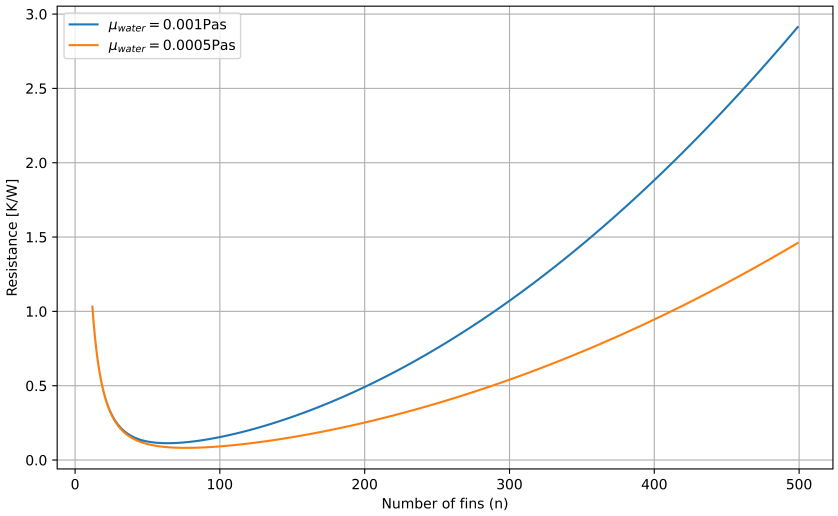

Figure 9: Resistance vs. number of fins for different viscosities

Another strategy could involve optimizing the channel geometry, striking a better balance between the number of fins and the flow resistance. This may require computational fluid dynamics (CFD) methods for more precise optimization, though such methods would significantly increase computational complexity.

Appendix

The code used for the investigation can be found in the following.

```python
import toml
import matplotlib.pyplot as plt
from math import ceil

DPI = 1000  # Define the resolution for saved plots


def plot_results(n_values, variable_name, values, plot_filename, mark_min=False):
    """
    Plots a graph of the given values against the number of fins and saves the plot.

    Parameters:
    n_values (iterable): The x-axis values (number of fins).
    variable_name (str): The label for the y-axis (variable name).
    values (iterable): The y-axis values (computed values for the variable).
    plot_filename (str): The file path to save the plot.
    mark_min (bool): Whether to mark the minimum value on the plot.
    """
    fig, ax1 = plt.subplots(figsize=(10, 6))
    ax1.plot(n_values, values)
    ax1.set_xlabel("Number of fins (n)")
    ax1.set_ylabel(variable_name)
    ax1.grid()

    if mark_min:
        # Mark the minimum value and annotate it
        min_value = min(values)
        min_index = values.index(min_value)
        ax1.plot(n_values[min_index], min_value, "o", color="#F97306")
        ax1.annotate(
            f"Min: {min_value:.2f}",
            (n_values[min_index], min_value),
            textcoords="offset points",
            xytext=(0, 10),
            ha="center",
            color="#F97306",
        )
        print(
            f"Minimum {variable_name} value: {min_value} at n = {n_values[min_index]}"
        )

    fig.savefig(plot_filename, dpi=DPI)
```

```
    plt.close(fig)


class HeatSink:
    """
    A class representing a heat sink model for calculating various heat-related parameters.
    """

    def __init__(self, config):
        """
        Initializes the HeatSink object with the parameters from the provided configuration.

        Parameters:
        config (dict): Configuration values from a TOML file.
        """
        # Load configuration values
        self.k_silicon = config["silicon"]["k"]
        self.k_water = config["water"]["k"]
        self.mu_water = config["water"]["mu"]
        self.c_p_water = config["water"]["cp"]
        self.rho_water = config["water"]["rho"]
        self.q = config["power"]
        self.w_chip = config["dimensions"]["w_chip"]
        self.l_chip = config["dimensions"]["l_chip"]
        self.h_fins = config["dimensions"]["h_fins"]
        self.t = config["dimensions"]["t"]
        self.dp_max = config["dp_max"]
        self.t_in = config["t_in"]
        self.nu = config["nu"]
        self.min_w = config["min_wc"]

    def compute_flow_rate(self, w_c, n):
        """
        Computes the flow rate based on the fin width and number of fins.

        Parameters:
        w_c (float): The width of the coolant channel.
        n (int): The number of fins.

        Returns:
        float: The computed flow rate.
        """
        return (
            self.dp_max
            * (1 - 0.63 * w_c / self.h_fins)
            * (n + 1)
```

```
            * (w_c**3)
            * self.h_fins
            / (12 * self.mu_water * self.l_chip)
        )

    def compute_chip_conductance(self, w_c=None, n=None):
        """
        Computes the conductance of the chip based on its dimensions and thermal properties.

        Returns:
        float: The chip conductance.
        """
        a_chip = self.w_chip * self.l_chip
        return self.t / (self.k_silicon * a_chip)

    def compute_convective_resistance(self, w_c, n=None):
        """
        Computes the convective resistance between the chip and coolant.

        Parameters:
        w_c (float): The width of the coolant channel.
        n (int): The number of fins (optional).

        Returns:
        float: The convective thermal resistance.
        """
        a_w = (w_c + 2 * self.h_fins) * self.w_chip * self.l_chip / (2 * w_c)
        htc = self.k_water * self.nu / w_c
        return 1 / (htc * a_w)

    def compute_sensible_heat_resistance(self, w_c, n):
        """
        Computes the sensible heat resistance based on the flow rate.

        Parameters:
        w_c (float): The width of the coolant channel.
        n (int): The number of fins.

        Returns:
        float: The sensible heat resistance.
        """
        f = self.compute_flow_rate(w_c, n)
        return 1 / (self.rho_water * self.c_p_water * f)

    def compute_total_resistance(self, w_c, n):
        """
```

```
        Computes the total thermal resistance considering conductive, convective, and heat t

        Parameters:
        w_c (float): The width of the coolant channel.
        n (int): The number of fins.

        Returns:
        float: The total thermal resistance.
        """
        r_heat = self.compute_sensible_heat_resistance(w_c, n)
        r_cond = self.compute_chip_conductance()
        r_conv = self.compute_convective_resistance(w_c, n)
        return r_cond + r_conv + r_heat

    def compute_surface_temperature(self, w_c, n):
        """
        Computes the surface temperature of the chip based on the total thermal resistance.

        Parameters:
        w_c (float): The width of the coolant channel.
        n (int): The number of fins.

        Returns:
        float: The surface temperature of the chip.
        """
        r_total = self.compute_total_resistance(w_c, n)
        return self.t_in + r_total * self.q * self.w_chip * self.l_chip

    def compute_wc(self, n):
        """
        Computes the coolant channel width based on the number of fins.

        Parameters:
        n (int): The number of fins.

        Returns:
        float: The coolant channel width.
        """
        return self.w_chip / (2 * n + 1)

    def compute_all_results(self, n_min, n_max, method):
        """
        Computes the results for a range of fins based on the provided method.

        Parameters:
        n_min (int): The minimum number of fins.
```

```
        n_max (int): The maximum number of fins.
        method (function): The method to compute the results (e.g., heat resistance).

        Returns:
        tuple: A tuple of n_values (x-values) and results (y-values).
        """
        n_values = range(n_min, n_max)
        results = []

        for n in n_values:
            w_c = self.compute_wc(n)
            result = method(w_c, n)
            results.append(result)

        return n_values, results


# Load configuration from TOML file
config = toml.load("config.toml")

# Create the HeatSink object with the loaded configuration
heat_sink = HeatSink(config)

# Exploration range for fins
n_max = ceil((heat_sink.w_chip / heat_sink.min_w - 1) / 2)
n_min = ceil((heat_sink.w_chip / heat_sink.h_fins - 1) * 0.5)

# Compute and plot various results
n_values, r_heat_values = heat_sink.compute_all_results(
    n_min, n_max, heat_sink.compute_sensible_heat_resistance
)
plot_results(
    n_values, r"$R_{heat} \left[\frac{K}{W}\right]$", r_heat_values, "images/r_heat.png"
)

n_values, r_conv_values = heat_sink.compute_all_results(
    n_min, n_max, heat_sink.compute_convective_resistance
)
plot_results(
    n_values, r"$R_{conv} \left[\frac{K}{W}\right]$", r_conv_values, "images/r_conv.png"
)

n_values, r_tot_values = heat_sink.compute_all_results(
    n_min, n_max, heat_sink.compute_total_resistance
)
plot_results(
```

```
    n_values,
    r"$R_{tot} \left[\frac{K}{W}\right]$",
    r_tot_values,
    "images/r_tot.png",
    True,
)

n_values, surf_temp_values = heat_sink.compute_all_results(
    n_min, n_max, heat_sink.compute_surface_temperature
)
plot_results(
    n_values,
    r"$T_{chip} \left[K\right]$",
    surf_temp_values,
    "images/temp_chip.png",
    True,
)


# Additional plots
def vary_and_plot(
    n_min, n_max, param_values, heatsink, method, ylabel, file, param_name
):
    """
    Varies a parameter (like viscosity or pressure drop) and plots the results.

    Parameters:
    n_min (int): The minimum number of fins.
    n_max (int): The maximum number of fins.
    param_values (list): List of values for the parameter to vary.
    heatsink (HeatSink): The HeatSink object.
    method (function): The method to compute the results (e.g., flow rate).
    ylabel (str): The label for the y-axis.
    file (str): The filename to save the plot.
    param_name (str): The name of the parameter being varied.
    """
    fig, ax = plt.subplots(figsize=(10, 6))
    for param in param_values:
        setattr(heatsink, param_name, param)
        n_values, results = heatsink.compute_all_results(n_min, n_max, method)
        ax.plot(
            n_values, results, label=f"{param_name}={param}"
        )  # Add a label for legend

    ax.set_xlabel("Number of fins (n)")
    ax.set_ylabel(ylabel)
```

```
    ax.grid()
    ax.legend(title=f"{param_name} variations")
    plt.savefig(file, dpi=DPI)
    plt.close(fig)


# Plot different resistances
fig, ax1 = plt.subplots(figsize=(10, 6))
n_values, r_cond_values = heat_sink.compute_all_results(
    n_min, n_max, heat_sink.compute_chip_conductance
)
ax1.plot(n_values, r_heat_values, label="$R_{heat}$")
ax1.plot(n_values, r_conv_values, label="$R_{conv}$")
ax1.plot(n_values, r_cond_values, label="$R_{cond}$")
ax1.plot(n_values, r_tot_values, label="$R_{tot}$", linewidth=2)
ax1.set_xlabel("Number of fins (n)")
ax1.set_ylabel("Resistance [K/W]")
ax1.grid()
ax1.legend()
fig.savefig("images/resistance_comparison.png", dpi=DPI)

# Vary viscosity and plot the results
mu_values = [1e-3, 2e-3, 3e-3]
vary_and_plot(
    n_min,
    n_max,
    mu_values,
    heat_sink,
    heat_sink.compute_flow_rate,
    r"Flow rate [$\frac{m^3}{s}$]",
    "images/mu_flowrate.png",
    "mu_water",
)

# Vary dp_max and plot the results
dp_max_values = [1e4, 5e4, 1e5]
vary_and_plot(
    n_min,
    n_max,
    dp_max_values,
    heat_sink,
    heat_sink.compute_flow_rate,
    r"Flow rate [$\frac{m^3}{s}$]",
    "images/dp_flowrate.png",
    "dp_max",
)
```

```python
# Plot different resistances ad different dps
fig, ax1 = plt.subplots(figsize=(10, 6))
heatsink = HeatSink(config)
n_values, r_tot_values_1 = heat_sink.compute_all_results(
    n_min, n_max, heatsink.compute_total_resistance
)
ax1.plot(
    n_values, r_tot_values_1, label=f"$\Delta P = {heatsink.dp_max} \mathrm{{Pa}}$"
)
heatsink.dp_max = 1e5
n_values, r_tot_values_2 = heat_sink.compute_all_results(
    n_min, n_max, heatsink.compute_total_resistance
)
ax1.plot(
    n_values, r_tot_values_2, label=f"$\Delta P = {heatsink.dp_max} \mathrm{{Pa}}$"
)
ax1.set_xlabel("Number of fins (n)")
ax1.set_ylabel("Resistance [K/W]")
ax1.grid()
ax1.legend()
fig.savefig("images/resistance_comparison_dp.png", dpi=DPI)

# Plot different resistances ad different mus
fig, ax1 = plt.subplots(figsize=(10, 6))
heatsink = HeatSink(config)
n_values, r_tot_values_1 = heat_sink.compute_all_results(
    n_min, n_max, heatsink.compute_total_resistance
)
ax1.plot(
    n_values,
    r_tot_values_1,
    label=f"$\mu_{{water}} = {heatsink.mu_water} \mathrm{{Pa s}}$",
)
heatsink.mu_water = 5e-4
n_values, r_tot_values_1 = heat_sink.compute_all_results(
    n_min, n_max, heatsink.compute_total_resistance
)
ax1.plot(
    n_values,
    r_tot_values_1,
    label=f"$\mu_{{water}} = {heatsink.mu_water} \mathrm{{Pa s}}$",
)
ax1.set_xlabel("Number of fins (n)")
ax1.set_ylabel("Resistance [K/W]")
ax1.grid()
```

```
ax1.legend()
fig.savefig("images/resistance_comparison_mu.png", dpi=DPI)
```