ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

DEPARTMENT OF
INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

<Specify semester and year>

LaTeX Report Template

<Specify report type>

Titlepage
Logo
Placeholder

Ritvik Ranjan
rranjan@ethz.ch

<Specify date>

Advisors: <Specify advisors>

Professor: <Specify professor>

Abstract

The abstract summarizes what this report is about. It focusses on the big picture and does not go into details. You should write concisely about the following points:

- Describe the **background** of your project: what is the motivation for your project and why is it important?
- Describe the **objectives** of your project.
- Describe the **problems** that must be addressed to achieve the objectives—why are these problems difficult?
- Describe your **approach** and **methods**.
- Summarize the most important **results**.
- State the main **conclusion** and its significance.

The abstract typically takes half a page and should not be longer than one full page. Try to write a draft of the abstract early on to have a good idea of your project, but revise the abstract as the project progresses. Write the final version of the abstract once the report is otherwise complete.

The remainder of this document contains an example on structure and content of the report. This template is meant to guide you and not to force you into a certain structure—just make sure you and your advisors agree on content and structure of the report *before* you start writing it. Appendix A gives more specific guidelines for some major project areas (e.g., hardware designs). If you are new to LaTeX or want to learn some best practices, you should also check the short LaTeX guide in Appendix B.

Acknowledgments

You rarely work in complete isolation, and this is the place to acknowledge contributions by others.

Declaration of Originality

Download the official declaration of originality[1], print it, and sign it. When printing your thesis, replace this sheet with the physically signed original paper. For the digital version, scan the filled declaration of originality (either before signing or remove your signature[2] from the image) and replace the text inside this file with `\includepdf[scale=0.9]{declaration_of_originality}`.

[1] https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf

[2] By removing your signature from the digital version of your report, you enable us to share your report with collaborators without having to deal with privacy concerns.

Contents

Chapter 1

Introduction

- Why set of linear equations arise in scientific computing - Why some of the matrices are sparse - Time and memory complexity of solving linear systems - How does permutation help either reducing fill-in, memory or improving parallelism - In this work, we focus solely on the permutation problem. We explore existing methods for sparse symmetric matrix reordering. We explore any possible optimizations and parallelization strategies. We look for single thread, multi-thread and distributed memory implementations. - We thoroughly evaluate the performance of the different reordering methods and compare them in various qualities for a lot of different matrices.

Sparse linear systems of equations form the computational backbone of modern scientific computing, arising naturally across virtually every domain of computational science and engineering. These systems arise in a huge range of practical applications, including in areas as diverse as quantum chemistry, computer graphics, computational fluid dynamics, power networks, machine learning, and optimization. The prevalence of sparse matrices stems from the fundamental mathematical property that many physical phenomena exhibit local interactions—adjacent elements in a discretized domain interact strongly, while distant elements have minimal or no direct coupling.

In computational fluid dynamics (CFD), sparse linear systems arising from implicit PDE discretizations dominate the computational workload. Finite element method (FEM) discretizations of partial differential equations naturally produce sparse matrices because only adjacent balls are coupled in the physical system, as opposed to dense systems where every element interacts with every other element. Similarly, in structural mechanics, quantum chemistry calculations, circuit simulation, and image processing, the underlying mathematical models preserve locality relationships that translate directly into sparse matrix structures.

The sparsity arises from the discretization process itself. When continuous problems described by partial differential equations are discretized using methods such as finite differences, finite elements, or finite volumes, the resulting algebraic equations connect only neighboring grid points or elements. Conceptually, sparsity corresponds to systems with few pairwise interactions. For instance, in a three-dimensional finite element mesh,

each node typically connects to only a small neighborhood of adjacent nodes, regardless of the total problem size, resulting in matrices where the number of non-zero elements is roughly equal to the number of rows or columns rather than being proportional to n².

1.1. Emergence of Sparse Linear Systems

Sparse linear systems of the form $Ax = b$, where $A \in \mathbb{R}^{n \times n}$ is sparse and symmetric positive definite, form the computational backbone of modern scientific computing. These systems arise naturally across virtually every domain of computational science and engineering, including quantum chemistry, computer graphics, computational fluid dynamics, power networks, machine learning, and optimization. The prevalence of sparse matrices stems from the fundamental mathematical property that many physical phenomena exhibit local interactions—adjacent elements in a discretized domain interact strongly, while distant elements have minimal or no direct coupling.

Mathematically, a matrix A is considered sparse when the number of nonzero entries $\text{nnz}(A)$ satisfies $\text{nnz}(A) = O(n)$ rather than $O(n^2)$ as in dense matrices. This sparsity typically arises from the discretization of partial differential equations (PDEs) using finite difference, finite element, or finite volume methods.

The sparsity arises from the discretization process itself. When continuous problems described by partial differential equations are discretized using methods such as finite differences, finite elements, or finite volumes, the resulting algebraic equations connect only neighboring grid points or elements. Conceptually, sparsity corresponds to systems with few pairwise interactions. For instance, in a three-dimensional finite element mesh, each node typically connects to only a small neighborhood of adjacent nodes, regardless of the total problem size, resulting in matrices where the number of non-zero elements is roughly equal to the number of rows or columns rather than being proportional to n^2.

[example for quantum transport; vincent's ppt; add photos]

1.2. Matrix Permutation

Matrix permutations serve as a fundamental preprocessing step that can dramatically improve both computational efficiency and memory requirements for sparse linear system solution. The basic principle involves reordering the rows and columns of the matrix A to obtain a permuted system $PAP^T\hat{x} = Pb$, where P is a permutation matrix, such that the reordered matrix exhibits superior properties for factorization.

1.2.1. Fill-in Reduction

The primary motivation for permutation is fill-in reduction during matrix factorization. Fill-in occurs when zeros in the original matrix become non-zero during factorization, effectively destroying the sparse structure.

Consider the solution of a sparse system $Mx = b$, where M is an $n \times n$ symmetric, positive definite matrix. Using Gaussian elimination, we can find the Cholesky factor-

ization $M = LL^T$, where L is a lower triangular matrix with positive diagonal entries. The solution is then obtained by solving two triangular systems: $Ly = b$ and $L^T x = y$.

The computational complexity of this procedure depends critically on the sparsity of both M and L. If column j of L contains d_j nonzeros, then the factorization and solution can be performed in space proportional to $\sum\_jd\_j$ (the number of nonzeros in L) and time proportional to $\sum\_jd\_j^2$. Ignoring numerical cancellation, L will have nonzeros below the diagonal everywhere that M does, plus additional locations. The *fill* is defined as the set of below-diagonal positions where L is nonzero but M is zero.

If P is a permutation matrix, then PMP^T represents a reordering of the rows and columns of M, and the fill in the triangular factor can vary drastically for different choices of P. We can think of P as determining the order in which variables are eliminated from the system.

Finding the elimination order that minimizes fill is an NP-complete problem. Moreover, most sparse matrices inherently suffer from large fill-in: for any positive ϵ, there exists a constant $c(\epsilon)$ such that almost all $n \times n$ symmetric matrices with $c(\epsilon)n$ nonzeros have at least $(1 - \epsilon)n^2/2 - O(n)$ fill for every elimination order. This theoretical limitation underscores why effective heuristic reordering methods are essential.

The impact of proper ordering can be dramatic in practice: while the Cholesky factorization of an original matrix might result in L having about 8% nonzero density, appropriate reordering methods can reduce this to less than 1%. This reduction translates directly to computational savings, potentially reducing factorization cost from $O(n^3)$ to $O(n^{1.5})$ for many practical problems.

==========

The introduction motivates your work and puts it into a bigger context. It should answer the following questions: What is the background of this work? What is the state of the art? Why is this project necessary to advance the state of the art? What are the problems that have to be solved and why are they difficult? What are your contributions to solve these problems? How do you evaluate your solution to show that it is adequate and applicable?

An introduction written along these questions naturally follows the *SPSE*[1] approach. In the *situation*, you set the scene for your work and catch the interest of the readers by showing the importance and generality of the scene. In the *problem*, you spot an issue in the scene and show why and how it significantly taints the scene. In the *solution*, you outline your solution to that issue. Finally, in the *evaluation*, you present the main arguments why the claimed solution actually does solve the problem.

In the following chapters, you will elaborate each of the four SPSE elements in detail: In *Background*, you lay the foundations for an in-depth understanding of the situation and the problem. In *Implementation and Optimizations*, you show how others have address this (or similar) problems and why their solutions are not sufficient or applicable. In *Implementation*, you specify your solution, which you then evaluate rigorously for strengths and weaknesses in *Results*.

[1]The SPSE approach was established in a book [1], but is also briefly summarized in a more recent article [2], which is available online.

At the end of the introduction, you should explicitly show this structure to the reader by briefly explaining how this report is organized. Instead of using the general SPSE terminology and the chapter names mentioned above, we urge you to use the domain-specific terminology established in the introduction and point to chapters using cross references (e.g., refering to Chapter 2 instead of "the Background chapter").

Chapter 2

Background

Before we look into the algorithms and theory behind reordering techniques, we first take a dive into the factorization process which produces further non-zeroes, which in this and the entire thesis' context, is called as fill-ins. Fill-ins depends only on the structure of the matrix, ie. the positions where the initial non-zero entries are placed. It may be trivial but important to note, that if some numerical arithmetic results to a zero in the factored matrix, it is usually still considered as a fill-in.

Suppose the set of set of n equations that are to be solved are

$$Ax = b \tag{2.1}$$

where A is a sparse matrix, x is the vector of unknowns. The sparsity, or the number of non-zero entries in A determines the fill-in of it's Cholesky factor when it is employed to solve the aforementioned set of equations. Suppose the Cholesky factorization of A is given by LL^T, where L is a lower triangular matrix (with a positive diagonal) and L^T is the transpose of L. The efficiency of solving this set of equations is dependent on the number of non-zero entries in L. It has been shown (George and Liu) that we can facotr and solve the set of equations in space proportional to $\sum\_jd\_j$ and time complexity is $\sum\_jd\_j^2$, where d_j is the number of non-zeroes in the jth column of L.

2.1. Elimnination Trees and Symbolic Factorization

The symbolic factorization phase of sparse Cholesky decomposition determines the sparsity pattern of the factor LL L without computing numerical values. This analysis is crucial for allocating memory and understanding computational dependencies before performing the more expensive numerical factorization. The elimination tree, a fundamental data structure in sparse linear algebra, provides an elegant framework for characterizing these sparsity patterns and serves as the foundation for efficient symbolic factorization algorithms.

2.1.1. Definition and Construction

Consider a sparse symmetric positive definite matrix A of order n with its Cholesky factorization $A = LL^T$. The elimination tree $T(A)$ captures the structural relationships between columns of the Cholesky factor L.

Definition 2.1. For each column j of the Cholesky factor L, let

$$f(j) = \min\{i > j : \ell\_ij \neq 0\}$$

denote the row index of the first off-diagonal nonzero entry. If column j contains no off-diagonal nonzeros, we set $f(j) = j$.

The elimination tree $T(A)$ is constructed by defining $f(j)$ as the parent of node j for each $j = 1, 2, \ldots, n-1$. Node n serves as the root since $f(n) = n$.

Theorem 2.1. *The elimination tree $T(A)$ is a spanning tree of the filled graph $G^+(A)$ satisfying:*

$$\textit{parent}[j] = \min\{i > j : \ell\_ij \neq 0\}$$

2.1.2. Fundamental Properties

The elimination tree reveals the inherent structure of sparse Cholesky factorization through several key properties.

Theorem 2.2 (Path Characterization of Fill-in)**.** *An entry $\ell\_ij \neq 0$ exists in the Cholesky factor if and only if node i is an ancestor of node j in the elimination tree $T(A)$.*

This theorem provides a direct correspondence between nonzero entries in L and ancestor-descendant relationships in the tree, making the elimination tree a complete characterization of the factor's sparsity pattern.

Theorem 2.3 (Subtree Independence)**.** *Let $T[x\_i]$ and $T[x\_j]$ be two disjoint subtrees of $T(A)$. Then $\ell\_st = 0$ for any $x\_s \in T[x\_i]$ and $x\_t \in T[x\_j]$.*

This independence property reveals the potential parallelism available in sparse factorization, as computations involving disjoint subtrees can proceed without interference.

2.1.3. Sparsity Pattern Characterization

Row Structure Analysis The elimination tree enables precise characterization of row sparsity patterns through the concept of row subtrees.

Theorem 2.4 (Row Structure)**.** *Let $i > j > k$. The entry $\ell\_ij \neq 0$ if and only if node x_j is an ancestor of some node x_k in the elimination tree where $a\_ik \neq 0$.*

Definition 2.2 (Row Subtree)**.** For row i of L, the row subtree $T\_r[x\_i]$ is defined as:

$$T\_r[x\_i] = \{x\_j : \ell\_ij \neq 0, j \leq i\}$$

The row subtree can be constructed by identifying all nodes x_k where $a\_ik \neq 0$ and including all ancestors of these nodes up to (but not including) node i itself. Equivalently, $T\_r[x\_i]$ is obtained from the elimination tree $T[x\_i]$ by pruning at the leaves corresponding to nonzeros in the original matrix A.

Column Structure Analysis The dual perspective considers column sparsity patterns:

Theorem 2.5 (Column Structure)**.** *The structure of column j of L is given by:*

$$\{x\_i : \ell\_ij \neq 0, i \geq j\} = Adj\_G(A)(T[x\_j]) \cup \{x\_j\}$$

where

$$Adj\_G(A)(S) = \{x \notin S : x \in Adj\_G(A)(v) \text{ for some } v \in S\}$$

denotes the adjacency of set S in the original graph $G(A)$.

This characterization shows that the nonzeros in column j of L consist of node j itself plus all nodes adjacent (in the original graph) to any node in the subtree rooted at j.

2.2. Sequential Algorithms for Elimination Tree and Symbolic Factorization

2.2.1. Elimination Tree Construction

The elimination tree can be computed efficiently using a union-find data structure that tracks connected components during symbolic elimination. The following algorithm constructs the elimination tree parent array:

Using path compression and union-by-rank heuristics, this algorithm achieves complexity $O(m\alpha(m,n))$, where m is the number of nonzeros in A and α is the inverse Ackermann function.

2.2.2. Symbolic Factorization

Once the elimination tree is constructed, the sparsity pattern of L can be computed by traversing row subtrees:

The marker array prevents redundant traversals. This algorithm runs in $O(\text{nnz}(L))$ time, making it output-sensitive.

2.2.3. Efficient Row and Column Counting

For memory allocation, it is often sufficient to compute only the number of nonzeros in each row and column:

Gilbert, Ng, and Peyton developed a more sophisticated approach that computes these counts in $O(m\alpha(m,n))$ time by exploiting postorder traversals of the elimination tree and least common ancestor computations, avoiding explicit traversal of row subtrees.

2.3. Heuristics classification

There are several heuristics that have been proposed to reduce the fill-in developed over the years. These heuristics can be broadly classified in the following categories: Bandwidth minimization, minimum degree (and it's variants), nested dissection, banded structure methods using hypergraphs, and recently, machine learning approaches.

2.3.1. Bandwidth Minimization

Bandwidth minimization refers to the problem of permuting the rows and columns of a matrix such that the non-zero entries are as close to the diagonal as possible. Gaussian elimination can be performed in $O(nb^2)$ time on matrices of dimension n and bandwidth b, which is faster than the forward $O(n^3)$ algorithm when b is smaller than n (Lim A. et al., 2006a). Additionally, this problem is NP-complete, but several heuristics have been developed to approximate the optimal solution.

The Cuthill-McKee algorithm, introduced in 1969 by Cuthill and McKee (Cuthill E. and J. McKee, 1969), employs a breadth-first search to generate a level structure for graphs, aiming to reduce matrix bandwidth. While it became the most popular approach for bandwidth minimization throughout the 1970s, it is not without drawbacks. For instance, it can be computationally intensive, and the resulting bandwidth may not always match the width of the level structure (Chinn P.Z. et al., 1982). To address some of these issues, George (George J.A., 1971) proposed a reverse ordering, leading to the Reverse Cuthill-McKee algorithm. Subsequently, Gibbs et al. (Gibbs N.E. et al., 1976) introduced the GPS algorithm, which also relies on the concept of level structures.

We take a look at the reverse Cuthill-McKee algorithm, which is still widely used in practice.

2.3.2. Minimum Degree

The minimum degree algorithm determines the order in which to eliminate variables (or pivot) during Gaussian elimination to minimize fill-in (creation of new non-zero entries) in sparse matrices. At each step, it chooses the node with the minimum degree (fewest connections) for elimination.

The minimum degree algorithm operates on the adjacency graph of the sparse matrix. It begins by initializing the graph, where each vertex represents a variable. At each iteration, the algorithm selects the vertex with the smallest degree (i.e., the fewest neighbors), which corresponds to the variable whose elimination is expected to introduce the least fill-in. Upon elimination, the chosen vertex is removed from the graph, and all its neighbors are connected to each other, forming a clique to preserve the matrix structure. The degrees of the affected vertices are then updated to reflect the new connections. This process is repeated until all vertices have been eliminated. The resulting elimination order directly determines the pivot sequence used during matrix factorization.

(placeholder for general algorithm)

Variants of the minimum degree algorithm include:

1. Multiple Minimum Degree (MMD)

   When multiple nodes have the same minimum degree, uses additional tie-breaking rules. Often selects based on secondary criteria like external degree or node numbering.

2. Approximate Minimum Degree (AMD)

   Uses approximations to avoid expensive exact degree calculations. Employs concepts like "supervariables" and "element absorption". Much faster than exact minimum degree while maintaining similar fill-in quality.

3. Constrained Minimum Degree

   Incorporates additional constraints (like maintaining bandwidth). Balances minimum degree objective with other structural requirements.

2.3.3. Nested Dissection

Nested dissection is a divide and conquer heuristic for the solution of sparse symmetric systems of linear equations based on graph partitioning. Nested dissection can be viewed as a recursive divide-and-conquer algorithm on an undirected graph; it uses separators in the graph, which are small sets of vertices whose removal divides the graph approximately in half [3].

Nested dissection consists of the following steps: Form an undirected graph in which the vertices represent rows and columns of the system of linear equations, and an edge represents a nonzero entry in the sparse matrix representing the system. Recursively partition the graph into subgraphs using separators, small subsets of vertices the removal of which allows the graph to be partitioned into subgraphs with at most a constant fraction of the number of vertices. Perform Cholesky decomposition (a variant of Gaussian elimination for symmetric matrices), ordering the elimination of the variables by the recursive structure of the partition: each of the two subgraphs formed by removing the separator is eliminated first, and then the separator vertices are eliminated [4].

All the sequential algorithms for determining the elimination ordering of a graph G can be described by the following general algorithm: Generate the tree of separators for G and perform a tree traversal on the separator tree to order the vertices, where this traversal must visit a node before any of its parents.

Chapter 3

Implementation and Optimizations

Discuss the state-of-the art and other related work. Depending on how much related work exists and how central the comparison to it is in your thesis (discuss this with your advisors), the introduction may contain sufficient related work and this chapter can be omitted.

3.1. SuiteSparse implementation of RCM and Minimum Degree

The Reverse Cuthill-McKee (RCM) algorithm in SuiteSparse provides bandwidth reduction for symmetric sparse matrices through a breadth-first search strategy combined with degree-based ordering heuristics. The implementation processes disconnected components separately and employs pseudo-peripheral vertex selection to minimize profile and bandwidth.

The SuiteSparse RCM implementation incorporates several refinements over the basic algorithm. The pseudo-peripheral vertex selection uses multiple BFS traversals to identify vertices that are approximately diametrically opposite, which typically results in better bandwidth reduction than arbitrary starting points. The degree-based sorting of neighbors during BFS traversal helps create a more systematic ordering that tends to group low-degree vertices together, further improving the resulting bandwidth.

The algorithm's effectiveness stems from its ability to produce orderings where vertices with similar connectivity patterns are placed close together in the permutation. This locality property translates directly into reduced bandwidth and improved cache performance during matrix operations, making RCM particularly valuable for iterative solvers and direct factorization methods that benefit from band structure preservation.

3.1.1. AMD Algorithm Overview

The Approximate Minimum Degree (AMD) algorithm implemented in SuiteSparse follows a refined elimination-based approach that balances computational efficiency with fill-in minimization. The algorithm operates on a quotient graph representation and

Algorithm 1: SuiteSparse RCM Algorithm

input : Symmetric matrix A ($n \times n$)
output: Permutation P for bandwidth reduction

Initialize:;
Compute degree$[i] = |\text{adj}(i)|$ for all vertices i;
Find connected components $R\_1, R\_2, \ldots, R\_k$ using DFS;
Initialize visited$[i]$ = false for all i;
Set perm_index = 0;
Process each component:;
foreach *connected component R_j* **do**
    Find pseudo-peripheral start vertex:;
    start = `FindPseudoPeripheral`(R_j);
    `// Select vertex with minimum degree among maximum-distance vertices`
    Cuthill-McKee BFS traversal:;
    Initialize queue $Q = \{\text{start}\}$;
    Set visited[start] = true;
    Initialize CM_order = [start];
    while $Q \neq \emptyset$ **do**
        u = dequeue(Q);
        neighbors = sort(unvisited_adj(u),by_degree_ascending);
        foreach $v \in$ *neighbors* **do**
            if $\neg$*visited*$[v]$ **then**
                Set visited$[v]$ = true;
                enqueue(Q, v);
                Append v to CM_order;
            end
        end
    end
    Apply reverse ordering (RCM):;
    for $i = 0$ **to** $|CM\_order| - 1$ **do**
        P[perm_index $+ i$] = CM_order[$|$CM_order$| - 1 - i$];
    end
    perm_index $\leftarrow$ perm_index $+$ $|$CM_order$|$;
end
return P;

incorporates several key optimizations including aggressive absorption, approximate degree updates, and dense row detection.

The key innovation in SuiteSparse's AMD implementation lies in its aggressive absorption strategy and approximate degree computation. The aggressive absorption phase identifies elements that can be completely absorbed into the current pivot, reducing the size of the quotient graph and improving cache locality. The approximate degree updates provide a computationally efficient method to maintain ordering decisions without exact degree computation, which becomes prohibitively expensive as elimination progresses.

The dense row detection mechanism addresses a common pathology in minimum degree algorithms where elimination of high-degree vertices can lead to excessive fill-in. When the algorithm detects that a newly formed element exceeds a threshold based on the matrix size, it defers elimination of the associated variables, effectively implementing a hybrid strategy that combines minimum degree with nested dissection principles.

3.1.2. COLAMD Algorithm Overview

The Column Approximate Minimum Degree (COLAMD) algorithm in SuiteSparse extends the minimum degree concept to rectangular matrices by operating on a bipartite graph representation. Unlike AMD which works on the symmetric structure $A^T A$, COLAMD directly processes the rectangular matrix A to produce a column ordering that minimizes fill-in during factorization.

The COLAMD algorithm addresses the unique challenges of rectangular matrix ordering by maintaining both row and column degree information throughout the elimination process. The bipartite graph formulation allows the algorithm to track how column eliminations affect the sparsity structure without explicitly forming the potentially much denser $A^T A$ matrix. The dense row detection mechanism prevents pathological behavior when processing matrices with highly dense rows, which could otherwise lead to excessive fill-in during factorization.

The degree update strategy in COLAMD carefully accounts for the fill-in patterns specific to rectangular matrices, where eliminating a column affects all other columns that share nonzero entries in the same rows. The algorithm's ability to handle supernodes (groups of columns with identical sparsity patterns) further enhances its effectiveness for structured matrices commonly arising in finite element applications.

Algorithm 2: SuiteSparse AMD Algorithm

input : Symmetric matrix A ($n \times n$), Control parameters
output: Permutation P, Info statistics

Initialize:;
Convert A to $A + A^T$ pattern if unsymmetric;
Build quotient graph $G = (V, E)$ from A;
Initialize degree lists Head$[d]$ for $d = 0$ to n;
Set degree$[v] = |\text{adj}(v)|$ for all vertices v;
Place each vertex in appropriate degree list;
Main elimination loop:;
for $k = 1$ **to** n **do**
    Select pivot:;
    Find minimum degree d with non-empty Head$[d]$;
    Select pivot p from Head$[d]$;
    Remove p from degree list;
    Set $P[k] = p$ (add to elimination ordering);
    Element absorption:;
    foreach *element e adjacent to p* **do**
        if $|L\_e \cap L\_p| = |L\_e|$ **then**
            Absorb e into p (aggressive absorption);
        end
    end
    Form new element e_p:;
    $L\_e\_p = \text{adj}(p) \setminus \{\text{absorbed elements}\}$;
    Mark e_p as new element;
    Update degrees (approximate):;
    foreach *uneliminated vertex $v \in L\_e\_p$* **do**
        external_degree$[v] = |\text{adj}(v) \cap \text{uneliminated}|$;
        bound $= |L\_e\_p| - |L\_e\_p \cap \text{adj}(v)|$;
        degree$[v] \approx$ external_degree$[v]$ + bound;
        Move v to new degree list;
    end
    Dense row detection:;
    if $|L\_e\_p| > \max(\alpha\sqrt{n}, 16)$ **then**
        Mark e_p as dense element;
        Move dense variables to end of ordering;
    end
end
Post-processing:;
Apply elimination tree post-ordering;
Compute final permutation statistics;
return *P and Info*;

Algorithm 3: SuiteSparse COLAMD Algorithm

input : Matrix A ($m \times n$) in CSC format, Control parameters
output: Column permutation P, Info statistics

Initialize:;
Treat A as bipartite graph $G = (R \cup C, E)$;
Set col_degree$[j]$ = nonzeros in column j, place in degree list;
Detect dense rows: if row_degree$[i] >$ threshold, mark dense;
Main elimination loop:;
for $k = 1$ **to** n **do**
    Select minimum degree column c (tie-break by density);
    Set $P[k] = c$;
    Update degrees:;
    foreach *row i connected to c* **do**
        foreach *uneliminated column j in row i* **do**
            fill_count $= |R(c) \cap R(j)|$;
            new_degree = col_degree$[j] + |R(c)| -$ fill_count $- 1$;
            Apply dense penalty if row i is dense;
            Move j to Head[new_degree];
        end
    end
    Mark eliminated rows and handle supernode formation;
end
Place remaining dense columns at end of ordering;
return *P and Info*;

3.2. Nested Dissection using METIS and SCOTCH

Nested dissection represents a divide-and-conquer approach to matrix ordering that recursively partitions the graph using small vertex separators, ordering the separated components before the separator vertices. METIS and SCOTCH implement sophisticated multilevel nested dissection algorithms that combine graph coarsening, separator finding, and refinement techniques to produce high-quality orderings for large sparse matrices.

The multilevel nested dissection approach in METIS provides superior ordering quality compared to single-level methods by operating at multiple scales. The coarsening phase creates a hierarchy of increasingly smaller graphs while preserving essential structural properties through heavy edge matching. This matching strategy prioritizes edges with large weights, which in the context of matrix ordering typically correspond to strong structural connections that should be preserved during coarsening.

The separator computation on the coarsest level benefits from reduced problem size while maintaining global structural awareness. The refinement phase during projection ensures that separators remain high-quality as they are mapped back to finer graph levels. The recursive application of this process creates a natural hierarchy where large components are isolated first, followed by progressively smaller substructures, resulting in elimination orderings with excellent fill-in characteristics for sparse direct solvers.

Algorithm 4: Multilevel Graph Coarsening

input : Graph $G = (V, E)$
output: Hierarchy of progressively coarser graphs

```
Hierarchy = [G] // Start with original graph
CurrentGraph = G;
while |V(CurrentGraph)| > COARSENING_THRESHOLD do
    Find heavy edge matching to preserve graph structure;
    Matching = HeavyEdgeMatching(CurrentGraph);
    Contract matched edges to create coarser graph;
    CoarserGraph = ContractEdges(CurrentGraph, Matching);
    Add to hierarchy;
    Hierarchy.append(CoarserGraph);
    CurrentGraph = CoarserGraph;
end
return Hierarchy;
```

Algorithm 5: Heavy Edge Matching Algorithm

input : Graph G with edge weights
output: Maximal matching favoring heavy edges

```
1  Matching = {};
2  Matched = {} // Track matched vertices
3  Sort edges by weight (descending) for better matching quality;
4  SortedEdges = SortByWeight(E(G), descending = true);
5  foreach edge (u,v) in SortedEdges do
6      if u ∉ Matched and v ∉ Matched then
7          Matching.add((u,v));
8          Matched.add(u);
9          Matched.add(v);
10     end
11 end
12 return Matching;
```

3.3. Parallel-Nested Dissection

3.4. Parallelizing minimum degree

Algorithm 6: Parallel AMD Algorithm

input : Matrix A ($n \times n$), parameters mult, lim
output: Elimination ordering

Preprocessing:;
$G \leftarrow$ `initialize_quotient_graph`$(A + A^T)$ `// Compute symmetric pattern`
Initialize degree lists for each thread:;
forall *tid* $= 0$ **to** *num_threads* $- 1$ ***in parallel*** **do** `initialize_degree_list`(tid) ;
Main elimination loop:;
while $|V| > 0$ **do**
    Find minimum approximate degree across all threads:;
    amd $\leftarrow$ `find_global_minimum_degree`();
    Select distance-2 independent set of pivots:;
    $D \leftarrow$ `distance_2_independent_set`(amd, mult, lim);
    if $|D| = 0$ **then**
        break `// No more valid pivots`
    end
    Eliminate pivots in parallel:;
    forall *pivot* $p \in D$ ***in parallel*** **do** tid $\leftarrow$ `get_thread_id`();
    `eliminate_pivot`(tid, p);
    ;
    Barrier synchronization;
    barrier();
end
return *elimination ordering*;

3.5. New Coarsening approaches in Nested Dissection

3.6. GPU Implementation of RCM

GPU Implementation of RCM is basically a parallel implementation of the breadth-first search (BFS) algorithm. Much research is available on parallel BFS, and the implementation in this thesis is based on the NVIDIA work on GPU-accelerated BFS in [5].

This GPU breadth-first search implementation uses a level-synchronous algorithm that processes the graph one depth level at a time, maintaining two key data structures: a vertex frontier (vertices to be explored in the current iteration) and an edge frontier (all neighbors of vertices in the current frontier). The algorithm begins with a single source vertex and alternates between two fundamental operations across BFS levels.

Each BFS iteration follows a two-phase process. In the expansion phase, the algorithm takes the current vertex frontier and performs parallel neighbor gathering to create the edge frontier. Multiple threads cooperatively read the adjacency lists of frontier vertices from the compressed sparse row (CSR) representation, collecting all outgoing edges. In the contraction phase, the algorithm filters this edge frontier by checking each neighbor's visitation status, removing already-visited vertices and duplicates to produce the vertex frontier for the next iteration. This process continues until the vertex frontier becomes empty, indicating the traversal is complete.

The expansion phase uses a multi-granularity approach to handle the irregular degree distributions common in real graphs. For vertices with small adjacency lists, the algorithm employs scan-based gathering where threads use prefix sum to compute scatter offsets, creating a perfectly packed array of neighbors that allows all threads to participate in memory reads without SIMD lane waste. For medium-sized adjacency lists, warp-based gathering assigns entire 32-thread warps to cooperatively process single vertices, with threads strip-mining through the adjacency list in parallel. For very large adjacency lists, CTA-based gathering enlists entire thread blocks (hundreds of threads) to process individual high-degree vertices. This hybrid strategy automatically adapts to the workload characteristics and ensures efficient GPU utilization regardless of degree distribution.

I took an already existing implementation of the parallel BFS algorithm from the NVIDIA CUDA SDK [6] and adapted it for reordering the graph using RCM.

Chapter 4

Implementation

This chapter explains your contributions, be it algorithms, architectures, libraries, hardware, or others. As usual, follow a top-down approach and break the chapter into meaningful sections. It may even be necessary to split the chapter into multiple chapters (e.g., to separate architecture from implementation in a hardware design).

Derive a structure for this chapter that is meaningful for your report and discuss it with your advisors.

Chapter 5

Results

In this chapter, you want to show that your implementation meets the objectives. Start by briefly describing the type of results you have collected and how these results relate to the objectives.

5.1. Evaluation setup

Precisely describe your measurement/evaluation setup in a top-down manner. For example:

- If you used a development platform, which one and how was it configured?
- Which tools did you use?
- What input data set / program / signals / ... did you use? If your data set is very large, you should fully define and describe it in an appendix chapter and refer to that definition here using simple labels.
- If your implementation is configurable, which configuration(s) did you evaluate?

The information given here (with references to external work) should be sufficient to reproduce the setup you used for your measurements.

5.2. Other sections

Structure the rest of this chapter into sections. For example, each section could discuss a different figure of merit or a different part of your implementation. As usual, discuss structure and content in advance with your advisors.

5.3. Comparison to related work

Compare your results to those achieved by others working on similar problems. This can be done in a separate section or directly in the sections above.

5.4. Current limitations

Demonstrating that your implementation meets the objectives is usually done by showing a lower bound on a given figure of merit. Conversely, you should also illuminate the limitations of your implementation by showing upper bounds on these (or other appropriate) figures of merit. Critically examining your ideas and their implementation trying to find their limits is also part of your work!

Chapter 6

Conclusion and Future Work

Draw your conclusions from the results and summarize your contributions. Point out aspects that need to be investigated further.

The conclusion can be structured inversely to the introduction: Summarize how the *evaluation* backs the *solution*, which solves the *problem*. Describe how your contributions improve the *situation* and what other (potentially newly discovered) problems have to be solved in the future.

Be concise: the conclusion normally fits on a single page and is rarely longer than two pages.

Appendix A

Topic-Specific Guidelines

A.1. IC design (ASIC or FPGA) projects

For IC design projects, the part of the report where you present your main contributions (the *Implementation* chapter in this template) usually consists of two chapters: *Hardware Architecture* and *Design Implementation*.

A.1.1. Hardware architecture

In the Hardware Architecture chapter, you should describe the architecture and decisions that led to it. Block diagrams and descriptions of control flow, data flow, and interfaces go there. The described architecture can be more general than what you actually implemented, e.g., through parameters.

A.1.2. Design implementation

The Design Implementation chapter is about the architecture variant you actually implemented. It can be meaningful to merge this chapter with the Results chapter to relate central figures of merit directly to implementation choices and tradeoffs; discuss this with your advisors.

Functional verification

Describe how you verified the design implementation functionally. For example, describe how your testbench interfaced the golden model and your circuit. Figure A.1 illustrates a sample setup.

Reference a ReadMe file that describes how the testbench can be launched.

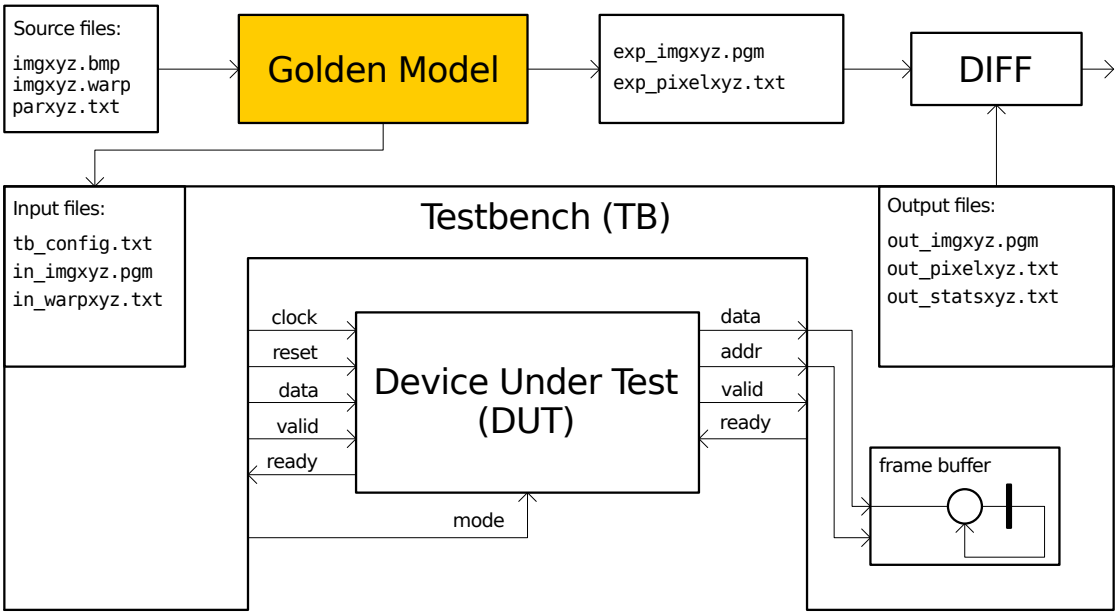

Figure A.1.: Testbench used for functional verification.

Back-end implementation

In ASIC projects, remember to discuss both front- and back-end implementation. That is, if you took special measures for floorplanning, DFT, or clock or power distribution, you will want to mention it here. In this case, make sure to add results that show the impact of your measures.

A.1.3. Results

Typical application-specific figures of merit of your hardware design are SNR, throughput, and memory/interface bandwidth. Moreover, you should also specify technology-specific figures such as area (for ASICs) or resource (for FPGAs) requirements, timing constraints, and power and energy consumption.

A.1.4. Data sheet

If your ASIC is getting fabricated, you need to write a data sheet for it. You should put this data sheet into the appendix of your report. You are free to write the data sheet in a standalone document and include a PDF file here or to write it in the source files of your report.

Sections of a typical IC data sheet are:

- Features
- Applications

- Packaging
- Bonding diagram like the one in Fig. A.2.
- Pinout diagram like the one in Fig. A.3.
- Interface description
- Register map
- Operation modes:
  - Functional modes
  - Test modes
- Electrical specifications
  - Recommended operating regions
  - Absolute maximum ratings

For more information, up-to-date bonding diagrams, and other technology-specific data ask the DZ.

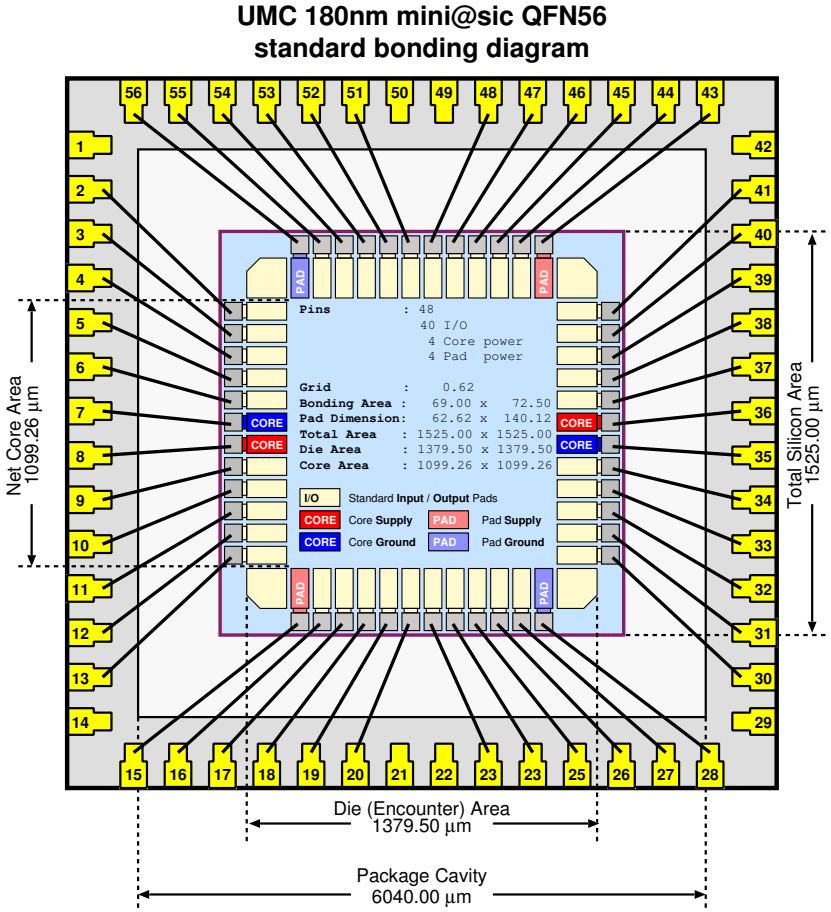

Figure A.2.: Standard bonding diagram for QFN56 UMC 180 nm mini@sics.

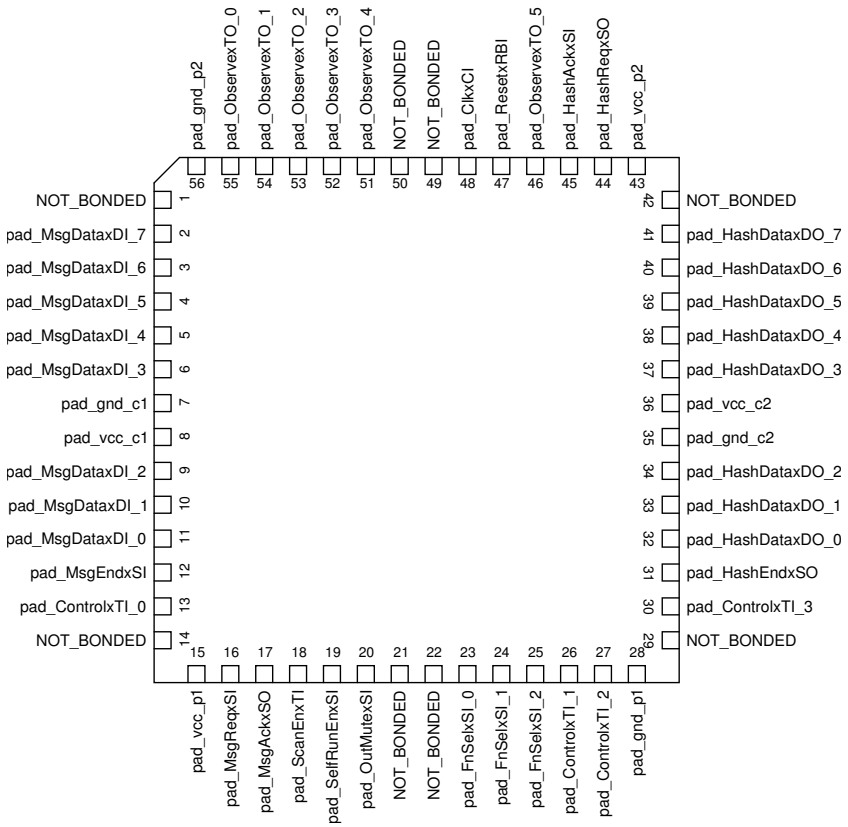

Figure A.3.: Pinout for MyFancyChip.

Appendix B

Compact Guide to LaTeX and the *iisreport* Class

Writing a report with LaTeX might at first not be as intuitive as with WYSIWYG editors. However, once you get used to the (rather simple) syntax, you will soon discover how powerful it is and how it helps you achieve tasks that are very difficult (if not impossible) to achieve with WYSIWYG editors.

This report template provides everything you need to get you started working with LaTeX. The rest of this chapter contains a short guide with examples for commonly used features.

B.1. Building the document

Generate a PDF file from this template by simply executing `make` in the directory where the top-level `.tex` file is in. Internally, this will invoke the `latexmk` program, which is the simplest and most consistent way to build a LaTeX document and is included in all recent LaTeX distributions. Additionally, `make` will check the `fig/` directory and generate PDF files for those figure raw files it knows how to compile (more on this in Appendix B.7).

B.2. Text editing and spacing

White spaces and line breaks are automatically inserted when the document is compiled. For this, the number of spaces between two words is irrelevant, but a different space length is automatically inserted after a period to make sentences better distinguishable. If you write a period that does not end a sentence, e.g., when mentioning Prof. Dr. S. Body, you need to escape the spaces between the abbreviated words with a backslash \. If you want to prevent LaTeX from breaking a line at a specific white space, you have to replace that space with a tilde ~. LaTeX also automatically hyphenates English words, so it can break a line within a word, although it does so only cautiously. Automatic

hyphenation can fail, e.g., for non-standard words, causing overly long lines. In such cases you have two options: First, if you have to break a standard word at an uncommon position, you can insert a `\-` at that position in the word. Second, you can define the hyphenation of a non-standard word by adding `\hyphenation{Jab-ber-woc-ky}` to the preamble[1] of your document.

A line of text is not broken at the same position as in the source code. For this reason, we suggest you put one sentence on one line of source code because this allows your VCS to track content changes much better than if you wrap lines within sentences. To start a new paragraph, insert an empty line. You can manually break a line by writing `\\`; however, this is rarely necessary and wide usage of it is a sign of fighting the typesetting system.

By default in this template, paragraphs start with a short indentation and are not separated by vertical white space, but this can be changed. If you prefer the latter, pass the `parskip` option to the `iisreport` document class, i.e., change the first line of your main document to `\documentclass[parskip]{iisreport}`.

B.2.1. Special characters

Many special characters are available, and they are all listed in *The Comprehensive LaTeX Symbol List* [7]. At the beginning you might not know what to search for, though, so it can be more helpful to use the *Detexify*[2] web app where you can draw the symbol you are looking for.

A frequent mistake is to mix up the three dashes (-, –, and —). The rules are simple [8]:

- The *hyphen*, `-`, is used between the elements of compound words; e.g., “run-time”.
- The *en-dash*, `--`, is used for ranges; e.g., “3–7”.
- The *em-dash*, `---`, is used for digressions within or at the end of a sentence—although you should use it sparingly.

Another frequent mistake are wrong quotation marks. Fortunately, this can also easily be avoided: Use `'text'` for ‘single quotation marks’ and `''text''` for “double quotation marks” (have a look at the source code to see the matching pairs). In American English, double quotes prevail and single quotes are typically only used inside double quotes.

B.2.2. Font faces and emphasis

The font face can be changed locally with the commands in Table B.1. The text to appear differently has to be put between the curly braces `{}`, i.e., the text is an *argument* to one of the commands. Some font faces can also be combined by nesting them. For example, `\textbf{\textit{some words}}` becomes ***some words***.

[1] The *preamble* of a document is formed by all code between the `\documentclass` command and the beginning of the document body after `\begin{document}`.

[2] http://detexify.kirelabs.org/classify.html

Command	Output
`\textrm{}`	Roman (the default in this document)
`\textsf{}`	Sans serif
`\texttt{}`	`Typewriter (i.e., all characters have the same width)`
`\textbf{}`	**Bold**
`\textit{}`	*Italic*
`\textsl{}`	*Slanted*
`\textsc{}`	SMALL CAPS

Table B.1.: Different font faces.

Command	Output sample
`\tiny`	quick brown foxes
`\scriptsize`	quick brown foxes
`\footnotesize`	quick brown foxes
`\small`	quick brown foxes
`\normalsize`	quick brown foxes
`\large`	quick brown foxes
`\Large`	quick brown foxes
`\LARGE`	quick brown foxes
`\huge`	quick brown foxes
`\Huge`	quick brown foxes

Table B.2.: Different font sizes.

When you want to *emphasize* text, use the `\emph{}` command instead of one of the commands in Table B.1. In this way, you separate a *property* of a piece of text (i.e., which text is emphasized) from its *formatting* (i.e., how emphasized text looks like). This is an important principle in typesetting with LaTeX. In this case, it allows you to define the formatting of all emphasized text independently of which text is meant to be emphasized.

B.2.3. Font sizes

The font size can be changed with the commands in Table B.2. These commands change the size within a given *scope*; for instance `{\Large some words}` only prints "some words" large.

Figure B.1.: Selected predefined colors, sorted by hue.

B.2.4. Coloring text

The color of text can be changed with the `\textcolor` and `\color` commands: The former takes two arguments, a declared color and the text to color. For example, `\textcolor{RoyalBlue}{I am royal}` becomes I am royal. The latter takes only a defined color and colors all text in its scope. For example, `{\color{RoyalPurple}So am I}` becomes So am I.

Figure B.1 shows a selection of predefined colors. The `xcolor` package manual [9] lists more colors and describes how to define custom colors.

B.3. Debugging

Occasionally, you will make syntax mistakes while writing a document, causing compilation to fail. In this case, the last lines of the console output will mention an error and point you to a `.log` file in the directory where you ran `make`. Open that log file. Even though that file can be very long and contains many technical details that are of no interest to you, finding errors is easy: simply search for lines starting with an exclamation mark!

If you use an undefined command, e.g., due to a typo, the error messages should be very helpful. If, however, you cause parentheses or environment delimiters to mismatch, the position of your mistake is hard to derive from the error messages.

A good technique to locate a mistake is to comment out recent changes until the document compiles neatly again. For recompilation, you should use `make clean all` to prevent errors that crept into temporary files from disturbing your bug hunt. To reduce compilation time for large documents, you can comment out chapters that are known to

be good. When you have a working version again, re-enable the code you commented out last piece-by-piece. This piecewise reduction should help you systematically find the mistake.

B.4. Math mode

LaTeX is probably the most powerful and elaborate tool to typeset mathematical content. Once you know a few core concepts, writing properly formatted mathematical content becomes quite simple.

To distinguish maths from regular text, maths is written in *math mode*. There are two categories that differ in their presentation: inline and displayed. Inline maths, e.g., $a^2 + b^2 = c^2$ is enclosed in `$` signs. It is meant for simple expressions. More complex content is displayed separately from the text. The most common way to display maths is inside the `equation` environment[3], for example:

$$\int_{-\infty}^{\infty} x \, \mathrm{d}x = 0. \tag{B.1}$$

Subscripts are written with `\sb{}`[4], superscripts with `\sp{}`, integrals with `\int`, and sums with `\sum`. Have a look at the source code of the last paragraph for usage examples.

Equations get a number by default, so you can label and refer to them (more on this in Appendix B.9). If you want to suppress an equation number, you can use the *starred* version of the equation environment, i.e., `equation*`.

Many mathematical symbols and functions are predefined, letting you express relations such as $\forall\, x \in \mathbb{R}\ \exists\, n \in \mathbb{N} : \ldots$ fluently. Wikipedia[5] has a list of all predefined mathematical symbols.

Variable names are by default one character long, causing `$ xy z $` to be typeset with identical spacing as `$ xyz $`. You should thus use single-letter variables whenever possible. If you have to use multi-letter variables, write them inside the `\var{}` command[6]. This causes x and y in the two-letter variable xy to be closer together. To make the variable clearly distinguishable from the next one, however, you may still have to insert a space[7] (as in $xy\,z$) or even an operator (as in $xy \cdot z$).

[3] *Environments* in LaTeX are similar to commands, but are usually used for larger chunks of code. For example, the entire document except the preamble is inside the `document` environment. Environments are formed with `\begin{environmentname} ... \end{environmentname}`.

[4] By default, LaTeX would allow to use the underscore _ for subscripts. In the `iisreport` document class, however, the underscore is a regular, printable character. The rationale is that the underscore is very common in technical designators and having to escape every single one is a common source of errors.

[5] https://en.wikibooks.org/wiki/LaTeX/Mathematics#List_of_Mathematical_Symbols

[6] The `var` command is not standard LaTeX but defined by the `iisreport` document class. If you want to use it elsewhere, it is very simple to implement: https://tex.stackexchange.com/a/129434/92384.

[7] The most common horizontal spacing macros in math mode are (in increasing order): `\,`, `\;`, `\enspace`, `\quad`, and `\qquad`. A complete list with examples is available here: https://tex.stackexchange.com/a/74354/92384. Keep in mind that frequent insertion of manual spacing may be a hack around a more fundamental problem.

When you want to use text in math mode (subscripts are a common use case for this), you must write that text inside the `\text{}` command to avoid the same problems as with multi-letter variables.

B.4.1. Delimiters: Parentheses, brackets, bars, and intervals

For simple parentheses and square brackets, you can readily use the `()` and `[]` characters, respectively. Curly braces are a bit more involved because `{}` are grouping characters. Thus, you would have to escape them and write `\{\}` instead. However, we recommend to use the `\cbr{}` command (short for "curly braces") instead. That command has the additional advantage of automatically sizing parentheses to the content. (The automatic sizing can be disabled by passing `[0]` as optional first argument to the command.) If you need automatically sized parentheses and square brackets, the commands are `\del{}` (for "delimiter") and `\sbr{}` (for "square bracket"), respectively. This allows you to effortlessly maintain readability even in deeply nested equations:

$$\left(E\left[\min\left\{X_1, X_2\right\}\right] - \left(\pi - \arccos\left(\frac{y}{r}\right)\right)\right)^n. \tag{B.2}$$

Absolute values are written with `\abs{}`, e.g.,

$$\left|\exp\left(x\pi i\right)\right| = 1 \quad \forall\, x \in \mathbb{R}, \tag{B.3}$$

while vector norms are written with `\norm{}`, e.g.,

$$\left\|\vec{x}\right\|_2 = \sqrt{\sum_{k=1}^{n} x_k^2} \quad \forall\, \vec{x} \in \mathbb{R}^n, \tag{B.4}$$

where the vector $\vec{x}$ was written with `\vec{x}` and the square root with `sqrt{..}`.

Intervals are written with the `\intxy{}` commands, where each of `x` and `y` are either `o` for open or `c` for closed.

B.4.2. Differential and derivative operators

Differential and derivative operators are written with the following commands: `\dif x` is the simple differential operator, e.g., $\mathrm{d}x$. `\Dif x` is a derivative operator, e.g., $\mathrm{D}x$. `\od[n]{f}{x}` is the ordinary n-th derivative operator, e.g., $\frac{\mathrm{d}^n f}{\mathrm{d}x^n}$. `n` is optional and should be omitted for the first derivative. `\pd[n]{f}{x}` is the partial n-th derivative operator, e.g., $\frac{\partial^n f}{\partial x^n}$. Finally, `\md{f}{n}{x}{q}{y}{r}` is the mixed partial derivative operator, e.g., $\frac{\partial^n f}{\partial x^q \partial y^r}$, where `n` is the total order of differentiation and `q` and `r` are the orders of differentiation for `x` and `y`, respectively.

B.4.3. Vectors, matrices, and distinction of cases

Both vectors and matrices are written with the `Xmatrix` environments, where `X` defines the delimiter of the matrix and can be `p` for parentheses, `b` for brackets, `B` for curly braces, `v` for vertical bars, `V` for double vertical bars, or omitted for no delimiters. The matrix is written row-wise with the elements of a row separated by `&` and each row is terminated by `\\`. A column vector is just a matrix with one column, a row vector one with one row. For example:

$$x = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \quad A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \tag{B.5}$$

The `Xmatrix` environments center the columns by default. If you want a different alignment, use the starred variant[8] of the environments, which accepts a single character as optional argument[9]: `r` for right, `c` for center, and `l` for left.

To write case distinctions, use the `dcases` environment. For example:

$$a(v) = \begin{cases} 0 & \text{if } v \geq \mathrm{c}, \\ \epsilon > 0 & \text{else.} \end{cases} \tag{B.6}$$

If you want the curly brace to be on the right of the cases, use the `rcases` environment.

B.4.4. Multi-line equations

If you want to write a single equation that is longer than one line, use the `multline` (without 'i'!) environment. That environment switches to math mode by itself, so you *must not* use it inside `equation`. Use the line break command, `\\`, to define the two lines of the equation. For example:

$$\begin{aligned} \alpha + \beta + \gamma + \delta + \epsilon + \zeta + \eta + \theta + \iota + \kappa + \lambda + \mu + \nu + \xi + + \pi + \rho + \sigma + \tau \\ = \upsilon + \phi + \chi + \psi + \omega. \end{aligned} \tag{B.7}$$

To write multiple equations in series or an especially complicated multi-line equation, use the `IEEEeqnarray` environment. That environment takes a series of characters specifying the columns as argument. The most common argument is `rCl`, meaning one right-aligned column followed by a center-aligned separator followed by a left-aligned column. As with matrices, use `&` to separate columns and `\\` to separate equation lines. For example:

$$a = b + c \tag{B.8}$$

$$\begin{aligned} &= d + e + f + g + h + i + j + k \\ &\quad + l + m + n + o \end{aligned} \tag{B.9}$$

$$= p + q + r + s, \tag{B.10}$$

[8] The *starred variant* of an environment or a command simply has a star $*$ at the end of the environment or command name, respectively. Not all environments and commands have a starred variant.

[9] *Optional arguments* are always enclosed in square brackets `[]`.

where the first line of the second equation was ended by `\nonumber` to suppress numbering that part of the equation.

Browse through *How to Typeset Equations in LaTeX* [10] for further informations and solutions to more complex examples.

B.4.5. Definitions, theorems, lemmas, and proofs

Here are some examples on writing definitions, theorems, lemmas, and proofs.

Definition B.1 (Singularity)**.** Let U be an open subset of the complex numbers $\mathbb{C}$, $a \in U$, and f be a complex differentiable function defined on $U \setminus \{a\}$.

The point a is a *removable singularity* of f if there exists a holomorphic function g defined on all of U such that $f(z) = g(z)\ \forall\, z \in U \setminus \{a\}$.

The point a is a *pole* or *non-essential singularity* of f if there exists a holomorphic function g defined on U with $g(a) \neq 0$ and $n \in \mathbb{N}$ such that

$$f(z) = \frac{g(z)}{(z-a)^n} \quad \forall\, z \in U \setminus \{a\}\,. \tag{B.11}$$

The lowest such number n is called the *order of the pole*.

The point a is an *essential singularity* of f if it is neither a removable singularity nor a pole. The point a is an essential singularity iff the Laurent series has infinitely many powers of negative degree.

Theorem B.1 (Residue Theorem)**.** *Let f be analytic in the region G except for the isolated singularities $a_1, a_2, \ldots, a_m$. If γ is a closed rectifiable curve in G that does not pass through any of the points a_k and if $\gamma \approx 0$ in G, then*

$$\frac{1}{2\pi i}\int_{\gamma} f = \sum_{k=1}^{m} n(\gamma; a_k)\, Res(f; a_k). \tag{B.12}$$

Proof. Left as an exercise for the reader. □

Lemma B.2 (Schwarz)**.** *Let $D := \{z \in \mathbb{C} : |z| < 1\}$ be the open unit disk in the complex plane centered at the origin, and let $f : D \to \mathbb{C}$ be a holomorphic map such that $f(0) = 0$ and $|f(z)| \leq 1$ on D. Then, $|f(z)| \leq |z|\ \forall\, z \in D$ and $|f'(0)| \leq 1$. Moreover, if $|f(z)| = |z|$ for some $z \neq 0$ or $|f'(0)| = 1$, then $f(z) = az$ for some $a \in \mathbb{C}$ with $|a| = 1$.*

Proof. Beyond the scope of this document. □

B.5. Quantities with SI units

- quantity with a unit: 300 MHz
- unit alone: GV

- ranges of quantities with units: 2 Mibit/s to 256 Mibit/s
- number (especially for engineering notation or very large numbers): 10 000, 3.14×10^6, 5×10^{-12}
- ranges of numbers 5×10^{-12} to 3.14×10^6

Math mode not required, but can be used with it.

B.6. Enumerations and itemizations

Itemizations are ...

- unnumbered and
- written inside the `itemize` environment, where every item starts with `\item`.

Enumerations, on the other hand, are ...

1. numbered and
2. written inside the `enumerate` environment.

Both itemizations and enumerations can be nested. The indentation level and itemization items are then automatically adjusted:

1. This demonstrates that
   a) enumerations and
   b) itemizations
      - can be nested.

B.7. Floats: figures and tables

Both figures and tables normally form *floating* environments. This means that LaTeX will automatically place them near to where they were in the source code, but not at the exact same position. The placement algorithm is fairly sophisticated [11] and usually works reasonably well.

The base environment for figures is `figure`, the one for tables is `table`. Floats usually get a caption with the `\caption{}` command. If you want to refer to them (more on this in Appendix B.9), you additionally have to put a `\label{}` after the `\caption{}` but on the same line (to have correct page numbers even near page breaks).

To center-align the content of a float, use the `\centerfloat` command at the beginning of that float.

Figure B.2.: Example figure.

Decimal	Hexadecimal	Octal	Binary
10	A_{16}	12_8	1010_2
13	D_{16}	15_8	1101_2

Table B.3.: Simple example table with some values in different number systems.

B.7.1. Figures

Images can be included with the `\includegraphics[properties]{file_name}` command, where `properties` allows to, e.g., define the `width`, `height`, or `scale` of an image in the `key=value` syntax. The `file_name` is relative to the `fig/` directory and the default suffix, `.pdf`, can be omitted. An example figure is given in Fig. B.2.

Whenever possible, you should use SVGs for two reasons: First, they can be scaled losslessly to the target size and resolution. Second, they allow you to keep a small, modifiable source file of the graphic under version control and have the PDF file to be included built automatically.

We recommend using *Inkscape*[10]. Simply draw a figure in Inkscape, set the canvas to where you want the image border, save the original `.svg` file in the `fig/` directory, and use `\includegraphics` on the file name without suffix. When you run `make`, the corresponding PDF file will get built automatically and included in your document. If you use a VCS (we highly recommend to do so!), track the original `.svg` file but add the auto-built `.pdf` file to the ignore list (e.g., `fig/.gitignore`). If you include PDF files of which you have no source files, track that `.pdf` file in your VCS.

As Inkscape does not support embedding fonts in its SVG files, you should either only use standard, widely-available fonts[11] or track the auto-built PDF images in `fig/` with your VCS. If you choose the latter, though, be aware that other collaborators who do not have the font installed must not commit changes to the built PDF images (because text with missing fonts will be rendered incorrectly by their Inkscape).

B.7.2. Tables

B.8. Algorithms and source code listings

Algorithm 7 shows an example algorithm. Have a look at the source code to discover how it works.

[10] Freely available at `https://inkscape.org`.
[11] Web safe fonts are good candidates for widely available fonts.

Algorithm 7: Disjoint decomposition.

input : A bitmap Im of size $w \times l$
output: A partition of the bitmap

```
1  special treatment of the first line;
2  for i ← 2 to l do
3      special treatment of the first element of line i;
4      for j ← 2 to w do
5          left ← FindCompress(Im[i, j − 1]);
6          up ← FindCompress(Im[i − 1, ]);
7          this ← FindCompress(Im[i, j]);
8          if left compatible with this then // O(left,this)==1
9              if left < this then Union(left,this);
10             else Union(this,left);
11         end
12         if up compatible with this then                    // O(up,this)==1
13             if up < this then Union(up,this);
               // this is put under up to keep tree as flat as possible
14             else Union(this,up);
               ;                                          // this linked to up
15         end
16     end
17     foreach element e of the line i do FindCompress(p);
18 end
```

Some	title	words
odd	odd	odd
even	even	even
odd	odd	odd

Table B.4.: Simple example table with different row and cell colors.

Day	Min. Temp.	Max. Temp.	Description
Monday	11 °C	22 °C	A clear day with lots of sunshine. However, a strong breeze will bring down the temperatures.
Tuesday	9 °C	19 °C	Cloudy with rain across many northern regions. Clear spells across most of Scotland and Northern Ireland, but rain reaching the far northwest.

Table B.5.: Example table with fixed column widths: 1.5 cm for columns two and three, 7 cm for column four. Content adopted from Wikibooks.

Source code listings are written inside the `lstlisting` environment, which takes optional arguments such as the `language` of the code, a `caption`, and a `label` in `key=value` syntax. Listing B.1 shows an example listing.

Listing B.1: Simple C code snippet.

```
int main()
{
  return 0;
}
```

External source code files can be included as listing with the `\lstinputlisting{}` command. Since you rarely want to include an entire file, you can specify the first and the last line with the `firstline` and the `lastline` key, respectively.

B.9. Citing and referencing

Use the `\cref{}` command to reference a document-internal label. Use the `\cite{}` command to cite an entry in the bibliography. You should always use a nonbreaking space, ~, before the `\cite{}` command to prevent the citation label from falling to the next line.

B.10. Printing and binding

When printing this document, pass the `print` option to the `iisreport` class. This will cause the layout to be optimized for printing.

B.11. Further reading

For a guide on writing research reports, we recommend the *Manual for Writers of Research Papers, Theses, and Dissertations* [12]. Furthermore, *The Elements of Style* [13] and *The Chicago Manual of Style* [14] are useful guides on concise writing in general. The latter is freely available online within the ETH network.[12]

If you are interested to learn more about LaTeX, you will find many resources online but the majority of them are written by casual amateurs and can teach you bad practices. Their approach is not strictly wrong but in the long term can cost you a lot of time compared to proper solutions. Thus, let us suggest to start all your TeX-related searches at the TeX StackExchange[13] community. Especially in the beginning, you will encounter common problems, for which there are usually several high-quality solutions on StackExchange, complete with an explanation of why the problem arose.

There are also some very good books on LaTeX. *The Not So Short Introduction to LaTeX2e* [15] is an excellent start, and *More Math into LaTeX* [16] teaches 99 % of what you need to know about typesetting maths. The first book[14] and the first section of the second book[15] are freely available online. For more advanced users, *The LaTeX Companion* [17] and the *The TeXbook* [8] are the definitive books.

B.12. *iisreport* options quick reference

The *iisreport* document class offers a few options that allow you to easily customize the layout of your report and configure certain features. Options can be specified inside the square brackets on the first line of the `report.tex` file, with individual options separated by commas. Here is an overview of all available options in alphabetic order:

- `oldfonts` brings back the fonts from the legacy IIS report template.
- `oldsubscript` disables making the underscore printable and makes it usable for subscripts instead.
- `parskip` causes paragraphs to start with a vertical space of half a line instead of indentation.
- `print` optimizes the page layout for printing and binding.

[12] http://www.chicagomanualofstyle.org
[13] https://tex.stackexchange.com/
[14] http://tobi.oetiker.ch/lshort/lshort.pdf
[15] http://www.ctan.org/tex-archive/info/Math_into_LaTeX-4/Short_Course.pdf

Appendix C

Task Description

Include the task description PDF file you got from your advisors with `\includepdf[pages=-, scale=0.9]{../path/to/task/description.pdf}`.

List of Acronyms

ASIC . . . . . application-specific integrated circuit

DFT . . . . . design for testability

DZ . . . . . Microelectronics Design Center at ETH Zurich

FPGA . . . . . field-programmable gate array

IC . . . . . integrated circuit

IIS . . . . . Integrated Systems Laboratory

PDF . . . . . Portable Document Format

SNR . . . . . signal-to-noise ratio

SPSE . . . . . Situation-Problem-Solution-Evaluation

SVG . . . . . scalable vector graphics

VCS . . . . . version control system

WYSIWYG . . . "what you see is what you get"

List of Figures

List of Tables

Bibliography

[1] M. Hoey, *On the Surface of Discourse*. Allen and Unwin, 1983.

[2] T. Miller and D. Parker, "Writing for the reader: A problem-solution approach," *English Teacher Forum*, no. 3, pp. 21–27, 2012. [Online]. Available: http://files.eric.ed.gov/fulltext/EJ997525.pdf

[3] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized Nested Dissection," *SIAM Journal on Numerical Analysis*, vol. 16, no. 2, pp. 346–358, Apr. 1979, publisher: Society for Industrial and Applied Mathematics. [Online]. Available: https://epubs.siam.org/doi/10.1137/0716027

[4] A. George, "Nested Dissection of a Regular Finite Element Mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, Apr. 1973, publisher: Society for Industrial and Applied Mathematics. [Online]. Available: https://epubs.siam.org/doi/10.1137/0710032

[5] D. Merrill, "Scalable GPU Graph Traversal."

[6] P. Kaleta, "kaletap/bfs-cuda-gpu," May 2025, original-date: 2019-11-14T12:55:50Z. [Online]. Available: https://github.com/kaletap/bfs-cuda-gpu

[7] S. Pakin, *The Comprehensive LaTeX Symbol List*, Jan. 2017. [Online]. Available: http://mirrors.ctan.org/info/symbols/comprehensive/symbols-a4.pdf

[8] D. E. Knuth, *The TEXbook*. Addison-Wesley, 1984.

[9] U. Kern, *xcolor: Driver-independent color extensions for LaTeX and pdfLaTeX*, May 2016. [Online]. Available: http://mirrors.ctan.org/macros/latex/contrib/xcolor/xcolor.pdf

[10] S. M. Moser, *How to Typeset Equations in LaTeX*, Sep. 2017. [Online]. Available: http://moser-isi.ethz.ch/docs/typeset_equations.pdf

[11] F. Mittelbach, "How to influence the position of float environments like figure and table in LaTeX," *TUGboat*, vol. 35, no. 3, pp. 248–254, 2014. [Online]. Available: https://www.latex-project.org/publications/tb111mitt-float.pdf

[12] K. L. Turabian, *A Manual for Writers of Research Papers, Theses, and Dissertations*. University of Chicago Press, 2013.

[13] W. Strunk, *Elements of Style*. Start Publishing LLC, 2012.

[14] U. of Chicago Press and U. of Chicago Press Editorial Staff, *The Chicago Manual of Style*. University of Chicago Press, 2017.

[15] T. Oetiker, *The Not So Short Introduction to LaTeX2e*, Jun. 2016. [Online]. Available: https://tobi.oetiker.ch/lshort/lshort.pdf

[16] G. Grätzer, *More Math Into LaTeX*. Springer International Publishing, 2016.

[17] F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, and C. Rowley, *The LaTeX Companion*, ser. Tools and Techniques for Computer Typesetting. Pearson Education, 2004.

[18] H. Kaeslin, *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, Apr. 2008.

[19] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to Elliptic Curve Cryptography*, ser. Springer Professional Computing. Springer, 2004.

[20] NIST, *Advanced Encryption Standard (AES) (FIPS PUB 197)*, National Institute of Standards and Technology, Nov. 2001.

[21] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, "OCB: A block-cipher mode of operation for efficient authenticated encryption," in *ACM Conference on Computer and Communications Security*, 2001, pp. 196–205.

[22] Xilinx. (2011, Nov.) Virtex-6 FPGA Configuration User Guide. UG360 (v3.4). [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf

[23] ——. (2011, Oct.) 7 Series FPGAs Configuration User Guide. UG470 (v1.2). [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[24] Wikipedia, "Isaac Newton," accessed October 1, 2012. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Isaac_Newton&oldid=514997436

[25] J. Wright, *siunitx: A comprehensive (SI) units package*, Aug. 2017. [Online]. Available: http://mirrors.ctan.org/macros/latex/contrib/siunitx/siunitx.pdf