МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерні науки,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційне та програмне забезпечення системи моделювання фізичних процесів в надпровідниках»
здобувача групи ІН - 01 Серебрякова Арсенія Євгеновича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Арсеній СЕРЕБРЯКОВ
(підпис)

Керівник, Сергій ШАПОВАЛОВ
кандидат фізико-математичних наук,
доцент
_____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Серебрякова Арсенія Євгеновича

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для прогнозування курсу валют. 3) Розробка інтелектуальної системи з прогнозування курсу валют. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.05.24-08.05.24	
2	*Огляд технологій, що використовуються для моделювання та симуляції схем електричних кіл з надпровідними пристроями*	08.05.24-10.05.24	
3	*Розробка інформаційного та програмного забезпечення системи моделювання фізичних процесів в надпровідниках*	10.05.24-20.05.24	
4	*Аналіз отриманих результатів*	20.05.24-26.05.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	26.05.24-31.05.24	

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 52 стр., 18 рис., 1 додаток, 26 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі моделювання та симуляції фізичних процесів в електричному колі з надпровідниками.

Об’єкт дослідження — інформаційне та програмне забезпечення системи моделювання фізичних процесів в надпровідниках.

Мета роботи — розробка інформаційного та програмного забезпечення системи моделювання фізичних процесів в надпровідниках.

Методи дослідження — алгоритми аналізу текстових даних та інструменти моделювання й симуляції схем електричного кола.

Результати — розроблено програмне забезпечення що дозволяє автоматизувати процес моделювання-симуляції схем електричного кола з надпровідними пристроями. Розроблено спеціальну бібліотеку для моделювання схем електричних кіл у додатку LibrePCB. Реалізовано графічний та командний інтерфейси програмного додатку та інтегровані в нього симулятори електричного кола. Створено інсталятор для програмного додатку, який дозволяє встановити його на будь-яку операційну систему.

ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЕЛЕКТРИЧНЕ КОЛО, НАДПРОВІДНІ ПРИСТРОЇ, АВТОМАТИЗАЦІЯ, АНАЛІЗ ТЕКСТУ

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі моделювання та симуляція складних електричних кіл, відіграють ключову роль у розробці нових технологій. Надпровідники, завдяки своїм унікальним властивостям, відкривають широкі можливості для розвитку електротехніки та електроніки. Останні декілька десятиліть Джозефсонські елементи активно досліджувались [1]. Їм приділяють особливу увагу, оскільки надпровідні пристрої використовуються як пам’ять в квантових комп’ютерах [2]. Точні та ефективні методи моделювання та симуляції є ключовими для розуміння та оптимізації цих систем. Таким чином, розробка додатку для оптимізації процесу моделювання та симуляції схем електричного кола з надпровідними пристроями стає актуальною задачею, що відповідає потребам сучасної науки та технологій.

Об’єкт дослідження. Моделювання та симуляція схем електричного кола з надпровідними пристроями.

Предмет дослідження. Теоретичні та практичні аспекти реалізація інформаційного та програмного забезпечення для оптимізація моделювання та симуляції схем електричного кола з надпровідними пристроями.

Гіпотеза. Застосування рекурсивного аналізу тексту для переведення схем електричного кола з одного формату в інший дозволяє автоматизувати процес моделювання-симуляції схем електричного кола.

Новизна. Розроблено програмний додаток що дозволяє автоматизувати процес моделювання-симуляції схем електричного кола, що не має інших існуючих альтернатив. Дослідження в роботі доповідалися на міжнародній конференції молодих вчених «Інформатика, математика, автоматика» (IMA-2024) [23].

Структура. Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз існуючих рішень

Фізичні явища та процеси не завжди залюбки допускають в сферу свого моделювання інформаційні технології. Це стосується досліджень і надпровідників. Проте деякі рекомендації у використанні пакетів прикладних програм SALOME, OpenFOAM, Paraview [3] та технології CUDA для моделювання складних фізичних процесів [4] є застосовними.

Нажаль системного інформаційного та програмного забезпечення поки ще не існує. І це стає предметом досліджень.

Надпровідність — це сукупність фізичних властивостей, які спостерігаються в певних матеріалах, коли електричний опір зникає, а магнітні поля витісняються з матеріалу. Будь-який матеріал, що демонструє ці властивості, є надпровідником. Монографія [5] відслідковує послідовність відкриттів в цих матеріалах і з фізичної точки зору описує різні моделі, що пояснюють та передбачають нові ефекти в надпровідниках. Надпровідним елементом електричного кола є Josephson Junction. Це електронний пристрій, який складається з двох надпровідників, розділених тонкою діелектричною або нормальною областю (див рис. 1.1) [6]. Він виявляє явище джозефсонівської тунельної керованої провідності, що полягає в тому, що електрони можуть проходити через тонку діелектричну перегородку між двома надпровідниками без будь-якого опору. Це приводить до таких явищ, як квантова тунельна провідність та квантові ефекти [7].

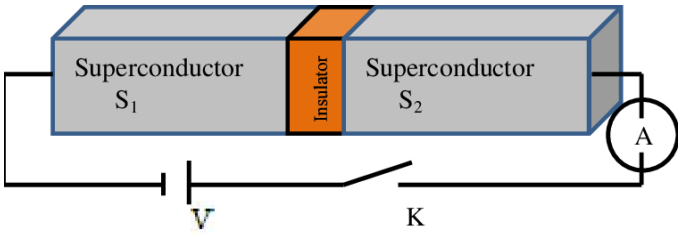

Рисунок 1.1 – Схематичний вигляд Josephson junction

1.2 Існуючі інструменти моделювання та симуляції

1.2.1 Системи моделювання електричного кола

Існує ряд програмних додатків для моделювання електричного кола, такі як CircuitLab, EveryCircuit, Circuit Diagram. Усі вони не дозволяють додати надпровідний елемент до електричного кола. Одним із програмних додатків що вирішує цю проблему є LibrePCB – крос-платформний пакет для автоматизації електронного проектування для малювання схем і розробки друкованих плат [8]. Він отримав свою популярність серед фахівців, оскільки він дозволяє створити користувацькі бібліотеку з потрібними елементами, розробити потрібні плати та одразу замовити їх. Він буде використаний для створення користувацької бібліотеки з елементами, яких не вистачає.

1.2.2 Системи симуляції електричного кола

Для симуляції ж електричного кола з надпровідниками існує невеликий набір програмного забезпечення. Більшість симуляторів використовують за вхідний формат електричного кола SPICE 3 нотацію. Ця нотація була розроблена у 1989 році, як оновлена версія для однойменного симулятора SPICE (Simulation Program for Integrated Circuits Emphasis) [9, 11]. Тому одним із основних завдань програмного додатку буде переведення електричного кола з заданого типу файлу у нотацію SPICE.

Існують наступні симуляторами які підтримують Josephson Junction:

- JSIM (Josephson SIMulator) [12].
- JoSIM (оновлена альтернатива JSIM) [13].

Вони є легкими, швидкими програмними додатками з консольним інтерфейсом, які приймають за вхідні дані файл електричного кола записаний у SPICE нотації та деякі додаткові дані щодо проведення самої симуляції.

В нотації SPICE компонент електричного кола ідентифікується за допомогою назви [10]. Префікс слугує для визначення типу, а назва після нього ідентифікує конкретний компонент (див. рис. 1.2, 1.3). Також після кожного елементу йдуть назви мереж, що з'єднують компоненти між собою. У кінці йде опис його атрибутів. Це може бути як просто значення опору чи індуктивності, так і більш складні атрибути, такі як вид струму чи напруги та значення що описують її рівняння.

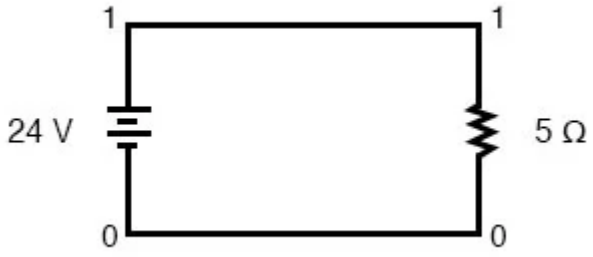

Рисунок 1.2 – Електричне коло

```
simple circuit
 v1 1 0 dc 24
 r1 1 0 5
 .end
```

Рисунок 1.3 – Електричне коло у нотації SPICE

1.3 Постановка задачі

Огляд існуючих джерел інформації та рішень програмного забезпечення в моделюванні фізичних процесів та явищ надав можливість сформулювати поставку задачі - створити інформаційне та програмне забезпечення системи моделювання фізичних процесів в надпровідниках. Для її вирішення потрібне виконання наступних завдань:

1. Для моделювання схем електричного кола потрібно створити спеціальну користувацьку бібліотеку LibrePCB з потрібними компонентами.

2. Розробити програмне забезпечення, що дозволить отримати схему електричного кола у записі SPICE із LibrePCB схеми, так щоб можна провести її симуляцію у JSIM/JoSIM додатку.
3. Створити графічний і консольний інтерфейси для програмного додатку із можливістю запуску симуляції одразу із нього. Також він має мати глибоку кастомізацію в залежності від різних потреб користувача.
4. Створити інсталятор для програмного забезпечення, що дозволить встановити його на будь-яку операційну систему разом із спеціальною бібліотекою LibrePCB.

У результаті виконання всіх завдань буде створено незалежний програмний додаток, який дозволить значно полегшити та пришвидшити роботу фізиків у сфері надпровідних пристроїв.

2 МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ

2.1 Менеджер бібліотек в LibrePCB

Пакет LibrePCB дозволяє легко будувати схеми електричного кола. Для цього він використовує різні бібліотеки. Після встановлення LibrePCB, щоб створити будь-яку схему, потрібно спочатку встановити базові бібліотеки:

1. LibrePCB Base
2. LibrePCB Connectors
3. LibrePCB Integrated Circuits

Library Manager дозволяє встановити бібліотеки з віддаленого серверу, локально імпортувати чи розробити свою [14]. Одним із завданням роботи є створення користувацької бібліотеки яка буде містити всі потрібні для симуляції компоненти. Ці компоненти можна знайти в посібнику користувача JoSIM симулятора [15]. У LibrePCB у базовій бібліотеці вже є деякі елементи, які не потрібно буде створювати:

1. Резистор (Resistor) – префікс R у нотації SPICE
2. Індуктор (Inductor) – префікс L у нотації SPICE
3. Конденсатор (Capacitor) – префікс C у нотації SPICE

Також є елементи які слугують для побудови кола, але вони відрізняються від попередніх і будуть представлені по іншому:

4. Заземлення (Ground Supply, GND Supply) – використовуються коли елемент заземлений. Тоді мережа в якій він заземлений дорівнює 0.
5. Живлення постійної напруги (Voltage at the common collector, VCC Supply) – використовується щоб показати постійну напругу. Є заземленим з однієї із сторін. Найчастіше використовують значення напруги 2.5 вольт.

Усі інші потрібні елементи відсутні. Тому бібліотека має містити наступні елементи:

1. Джозефсонівський контакт (Josephson junction) – префікс B у нотації SPICE

2. Джерело напруги (Voltage Source) – префікс V у нотації SPICE.
3. Джерело струму (Current Source) – префікс I у нотації SPICE.
4. Фазове джерело (Phase Source) – префікс P у нотації SPICE.
5. Лінія електропередачі (Transmission Line) – префікс T у нотації SPICE.
6. Модель (Model) – спеціальний елемент, що використовується для опису усіх атрибутів джозефсонікського контакту. Він може бути використаний декількома контактами.
7. Аналіз перехідних процесів (Transient Analysis) – спеціальний елемент, що відповідає за опис усіх параметрів симуляції.
8. Підсхема (Subcircuit) – спеціальний елемент, що відповідає іншій схемі, що використовується в середині даній. Вона є дуже важливою для оптимізації запису схем. Префікс X у нотації SPICE. Також як аргумент має мати файл в якому вона записана.
9. Вихідні дані (Output) – спеціальний елемент, який потрібен для запису результатів симуляції. Запис результатів може йти як у консоль, так і у файл, в залежності від параметрів користувача.
10. Текст (Text) – спеціальний елемент, який потрібен щоб зробити певні коментарі щодо зробленого електричного кола.

Також кожне із джерел має мати певний тип. В залежності від типу, джерело буде описано різним рівнянням з різними параметрами. Тому потрібно зробити схеми елементів для кожного його типу. Існують наступні типи джерел:

1. Piece Wise Linear (PWL). SPICE запис – $pwl(0\ 0\ T_1\ A_1 \dots T_n\ A_n)$. Це джерело лінійно інтерполює значення амплітуди для кожного моменту часу в симуляції між заданими амплітудами. Початкові два значення повинні бути нульовими на початку симуляції.
2. Pulse. SPICE запис – $pulse\ (A_1\ A_2\ [\ T_D[\ T_R\ \Big[\ T_F\Big[\ PW\ [\ PER\ ]\Big]\Big]]])$. Це джерело генерує імпульс між двома амплітудами (A_1 і A_2), починається через T_D і має час наростання та спаду (T_R & T_F), які за замовчуванням дорівнюють розміру кроку симуляції перехідного

процесу. PW та PER позначають ширину імпульсу та період відповідно. Ці значення за замовчуванням дорівнюють часу зупинки симуляції перехідного процесу, якщо їх не вказано. Це джерело дозволяє безперервно генерувати імпульс із заданою частотою.

3. Sinusoidal. SPICE запис – $\sin(A_0\ A\ [f\ [\ T_D[\theta]]])$. Це джерело, яке генерує синусоїдальний сигнал зі зсувом A_0 зсувом і A амплітудою на частоті f яка за замовчуванням дорівнює $\frac{1}{T_{STOP}}$. T_D встановлює час зупинки та θ модулює амплітуду сигналу. Функція генерує точку даних для кожного кроку симуляції перехідного процесу на основі наступного рівняння:

$$f(t) = A_0 + A * \sin\big(2\pi f(t - T_D)\big) * e^{-\theta(t-T_D)}$$

4. Custom Waveform. SPICE запис – $cus(wavefile\ T_S\ SF\ IM\ [\ T_D\ PER])$. Це джерело дозволяє згенерувати функцію на основі точок всередині текстового хвильового файлу. Цей файл повинен містити один рядок чисел, розділених пробілами. Наприклад 0 2 3 6 2 1 0. Кожне число в цьому рядку представляє амплітуду, відокремлену часовим кроком T_S і масштабується за допомогою масштабного коефіцієнта SF. Значення між точками інтерполюються без інтерполяції (0), лінійно (1) або кубічно (2). Функція може стати періодичною, якщо PER встановлено на 1, тоді шаблон повторюється протягом усього моделювання. Форма сигналу починається лише з T_D
5. DC. SPICE запис – $dc\ A$. Джерело постійного струму, напруги чи фази, яке завжди знаходиться в A у будь-який момент часу під час симуляції.
6. Noise. SPICE запис – $noise(A\ T_D\ T_{STEP})$. Це джерело створює значення шуму для часового кроку за умови, що він знаходиться після T_D. Значення шуму, що повертається, обчислюється за формулою: $f(t) = A\frac{GRAND()}{\sqrt{T_{STEP}}}$, де $GRAND()$ є гауссовою функцією генерування випадкових чисел.
7. Exponential. SPICE запис – $exp(A_1\ \ A_2\ \ T_{D1}\ \ \tau_1\ \ T_{D2}\ \tau_2\ )$. Повертає різні значення для 3 різних часових відрізків:

- Для $t <\ T_{D1}:\ f(t) = A_1$

- Для $T_{D1} \leq t < T_{D2}: f(t) = A_1 + (A_2 - A_1)(1 - e^{\frac{t-T_{D1}}{\tau_1}})$
- Для $T_{D2} \leq t: f(t) = A_1 + (A_2 - A_1)\left(1 - e^{\frac{t-T_{D1}}{\tau_1}}\right) + (A_1 - A_1)\left(1 - e^{\frac{t-T_{D2}}{\tau_2}}\right)$

Як можна побачити то бібліотека має містити багато різних компонентів, які будуть використані для моделювання

Обраний інструментарій та програмне забезпечення дозволять виконати всі проектні завдання та створити додаток, що може стати застосовним в якості одного з блоків інформаційного та програмного забезпечення системи моделювання фізичних процесів в надпровідниках.

2.2 Мова програмування

В постановці завдання поставлено розробити програмне забезпечення що буде обробляти швидко великий набір даних та включати в собі графічний і консольний інтерфейси. Також програмний додаток має бути крос-платформним, тобто підтримуватися на різних операційних системах. Виходячи з попередніх потреб, можуть підійти наступні мови для вирішення завдання:

1. C++
2. Java
3. Python

При порівнянні швидкодії цих мов програмування, C++ є найшвидшою [16]. Завдяки своїй продуктивності, ефективності у використанні пам'яті, широким можливостям бібліотек, контролю над ресурсами та швидкодії, це робить її ідеальною для обробки великих обсягів даних у реальному часі та перенесення на різні платформи з легкістю [17]. Тому вона найкраще підійде для вирішення поставлених у роботі завдань.

2.3 Середовище розробки

Але окрім самої мови програмування потрібно обрати ще інтегроване середовище розробки (IDE), фреймворк для створення графічного інтерфейсу та утиліту для створення інсталятору . Qt Framework для C++ є ідеальним вибором для розробки такого додатку [18].

По-перше, Qt йде в комплекті зі своїм IDE, що легко дозволе розробляти програмний додаток. Він має багато різних параметрів налаштування, різних версій компілятора Qt та корисних функцій, що забезпечать швидку та зручну розробку додатку.

По-друге, він надає широкий спектр інструментів для розробки графічного інтерфейсу користувача (GUI), що дозволяє з легкістю створювати інтуїтивно зрозумілі та естетично привабливі інтерфейси для взаємодії з додатком. Графічний інтерфейс має підтримку CSS стилів, що дозволить ще більш детально налаштувати його [24].

По-третє, Qt є кросплатформим фреймворком, що дозволяє однаково легко розробляти програмне забезпечення для різних операційних систем, включаючи Windows, macOS та Linux. Це робить Qt ідеальним вибором для проектів, які вимагають широкої сумісності з різними платформами без необхідності повторної розробки програмного забезпечення.

На останок, Qt має свою утиліту Qt Installer Framework, яка дозволяє легко створити інсталятор для будь-якої C++/Qt програми [19]. Для цього потрібно лише мати директорію з додатком та вказати певні параметри для створення інсталятора.

Таким чином, використання Qt у поєднанні з C++ забезпечить не лише потужну функціональність додатку, але й його швидку розробку та переносимість на різні платформи.

3 Програмна реалізація

3.1 Аналіз вхідних даних

Основною функцією програмного додатку є створення електричного кола в нотації SPICE в залежності від того як LibrePCB зберігає його. По-перше, створюємо тестовий проект з базовим електричним колом. Другим кроком знаходимо в директорії збереження проекту відповідний файл «circuit.lp» в якому зберігається електричне коло (див. рис. 3.1).

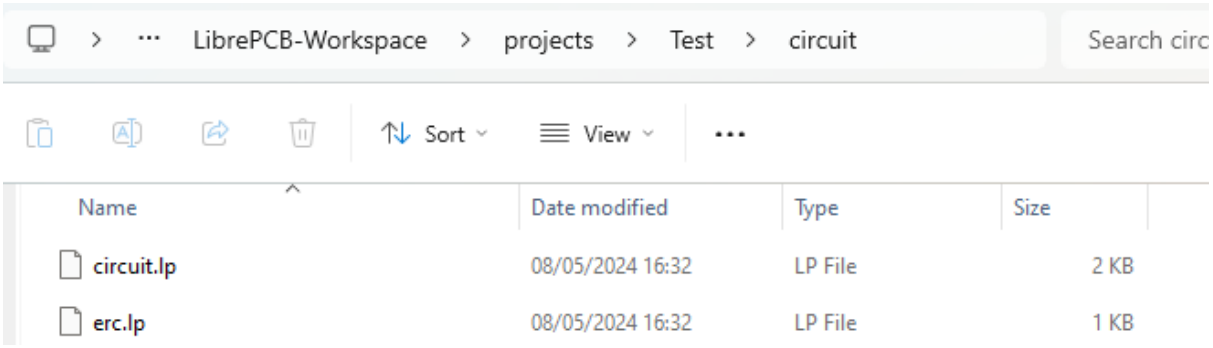

Рисунок 3.1 – Файл з електричним колом

В ньому представлена інформація, яка описує структуру електричного кола та компоненти, що його складають (рис. 3.2). Дані можна поділити на складені об’єкти та їх атрибути. Складеними об’єктами будуть:

1. librepcb_circuit: Це кореневий елемент файлу, що містить усі інші елементи опису електричного кола.
2. variant: Вказує на варіант (або конфігурацію) електричного кола та містить назву та опис цієї конфігурації.
3. netclass: Визначає класи мереж, які використовуються для з'єднання компонентів у схемі.
4. net: Представляє окрему мережу, яка з'єднує компоненти у схемі, з вказаною назвою.

5. component: Описує окремий компонент електричного кола, включаючи його ідентифікатор, назву, значення, атрибути (такі як індуктивність чи опір) та зв'язки з мережами.

```
(librepcb_circuit
 (variant a783d742-eadf-441c-905e-cf9173201d13 (name "Std")
  (description "Standard assembly")
 )
 (netclass 1579c873-83dd-4037-9834-cd7ff0baa38d (name "default"))
 (net 7b98e870-6df5-4526-a914-69668318df5a (auto true) (name "N3")
  (netclass 1579c873-83dd-4037-9834-cd7ff0baa38d)
 )
 (net a23729ed-014d-4974-a5ca-22b5bc17dcda (auto true) (name "N1")
  (netclass 1579c873-83dd-4037-9834-cd7ff0baa38d)
 )
 (net e8187002-d814-441a-af64-2ea4d5d68f2a (auto true) (name "N2")
  (netclass 1579c873-83dd-4037-9834-cd7ff0baa38d)
 )
 (component 14b7a76f-e797-4271-baa5-041c9e5a6cee
  (lib_component 506bd124-6062-400e-9078-b38bd7e1aaee)
  (lib_variant 62a7598c-17fe-41cf-8fa1-4ed274c3adc2)
  (name "L1") (value "{{INDUCTANCE}}")
  (lock_assembly false)
  (attribute "INDUCTANCE" (type inductance) (unit millihenry) (value ""))
  (signal 5b36d330-6f19-4391-8f95-1c2f6a658286 (net 7b98e870-6df5-4526-a914-69668318df5a))
  (signal 777f11cd-9d4e-4b2b-aafa-7e7a836ff56e (net a23729ed-014d-4974-a5ca-22b5bc17dcda))
 )
 (component 5cb378f0-3d5e-4780-ae34-1a5d43bdb0a7
  (lib_component ef80cd5e-2689-47ee-8888-31d04fc99174)
  (lib_variant a5995314-f535-45d4-8bd8-2d0b8a0dc42a)
  (name "R1") (value "{{RESISTANCE}}")
  (lock_assembly false)
  (attribute "RESISTANCE" (type resistance) (unit ohm) (value ""))
  (signal 3452d36e-1ce8-4b7c-8e5b-90c2e4929ed8 (net e8187002-d814-441a-af64-2ea4d5d68f2a))
  (signal ad623f98-9e73-49c3-9404-f7cfa99d17cd (net a23729ed-014d-4974-a5ca-22b5bc17dcda))
 )
)
```

Рисунок 3.2 – Електричне коло у circuit.lp

Усі складені об’єкти, окрім кореневого, мають свій UUID (Universally Unique Identifier) та ім’я [20]. Вони використовуються для ідентифікації та відображення елемента у електричному колі. Також до складених об’єктів можна віднести attribute та signal, оскільки вони мають вкладені об’єкти:

6. Attribute: Це властивість компонента, яка містить додаткову інформацію про його характеристики.
7. Signal: Це зв'язок між компонентами електричного кола через мережі.

Кожен об’єкт можна описати через UUID, ім’я та набір простих та складених атрибутів. Хоча attribute немає свого унікального ідентифікатора, але його можна створити під час створення об’єкта, щоб гарантувати унікальність атрибута.

Дане електричне коло потрібно зчитати та перевести у певні об’єкти. Для цього була створена наступна ієрархія класів:

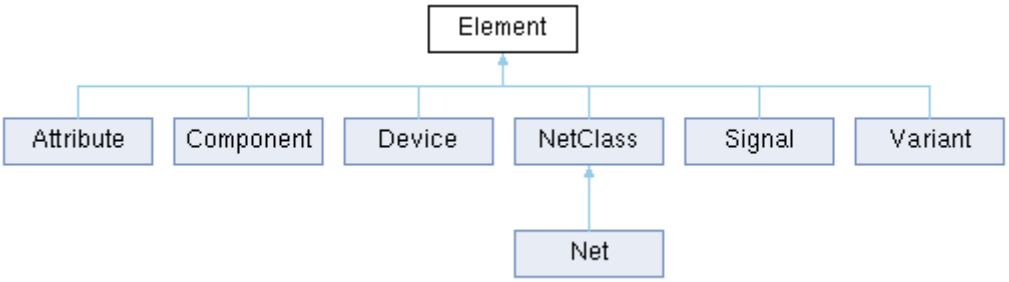

Рисунок 3.3 – Ієрархія C++ класів для представлення елементів електричного кола

Простим у реалізації методом зчитування електричного кола є використання regex [21]. Регулярний вираз (regular expression або скорочено regex) - це один із способів пошуку підрядків (відповідностей) в рядках. Здійснюється це за допомогою проглядання рядка у пошуках деякого шаблону. Хоча написання шаблону є досить складною задачею, але у результаті можна отримати досить простий код, де буде лише пошук певних елементів у цьому шаблоні.

Але як виявилось при реалізації цим методом, вбудовані regex в C++ не підтримують dot match everything mode (спеціальна модифікація регулярних виразів, що дозволяє пройти по декільком рядкам). Це не дозволяє написати регулярний вираз для об’єктів що записані у декілька рядків. Qt пропонує свої регулярні вирази. І хоча вони підтримують потрібну модифікацію, вони не є досить швидкими та в залежності від версії компілятора Qt їх швидкість може сильно відрізнятись. Потрібно знайти рішення що буде швидким та не залежним від компілятора.

В ідеальному варіанті потрібно зчитувати дані і одразу їх і переводити в потрібні об’єкти. Тому у даному дослідженні розроблено метод рекурсивного аналізу тексту для створення об'єктів, які представляють електричне коло у програмі [22]. Цей метод використовується для автоматичного розпізнавання та створення об'єктів на основі вхідного тексту, що містить відповідні дані про компоненти електричного кола. Аналіз тексту виконується з використанням

рекурсивних алгоритмів, які дозволяють ідентифікувати ключові слова та структури, необхідні для створення об'єктів. Кожне відповідне слово або дані з тексту інтерпретуються як атрибути чи компоненти електричного кола, що піддаються подальшій обробці для створення відповідних об'єктів. Цей підхід дозволяє автоматизувати процес створення об'єктів електричного кола з урахуванням усіх вказаних атрибутів.

Основною функцією в даній реалізації є parseElement:

```
void LibreNetlistParser::parseElement(QString parentUuid, QString::iterator last)
{
    currentCharacter++;
    QString name = nextWord(); // get the name of element/property/nested element.
    currentCharacter++;
    QString value = nextWord(); // get either uuid/property or attribute name.
    // if it is an element, then there is gonna be a whitespace character
afterwards.
    if (CharacterUtils::isWhitespaceCharacter(*currentCharacter)) {
        // create new element based on its name and uuid (name for attribute).
        Element *element = createNewElement(name, value);
        parseComponent(element->getUuid(), last); //parse the nested properties or
elements.
        // if the element has an parent element, we need to add it to the parent as
a property.
        // only the highest elements in hierarchy don't have parents.
        if (!parentUuid.isEmpty()) {
            Element *parent = elementMap[parentUuid].get();
            parent->setProperty(name, element);
        }
    }
    // if it as property, then there is gonna be ')' afterwards.
    else if (CharacterUtils::isCloseParanthesis(*currentCharacter)) {
        // get the parent element from the storage using uuid
        Element *parent = elementMap[parentUuid].get();
        //if an element with current uuid is already in the storage, just add it to
the parent.
        if (elementMap.contains(value)) {
            Element *element = elementMap[value].get();
            parent->setProperty(name, element);
        } else {
            parent->setProperty(name, value);
        }
    }
    currentCharacter++;
}
```

Вона зчитує 2 послідовні слова і в залежності від наступного символу, або створює новий елемент та додає його у сховище, або додає дані до вже існуючого елемента.

Також створено декілька додаткових функцій таких як nextWord() та nextDataInQuotes(), що дозволяють зчитати наступне слово чи наступні дані у

лапках відповідно. Вони використовують покажчик на поточний символ із заданого вхідного тексту.

Створення нового елемента відбувається за допомогою патерну проектування «Фабричний метод» [25, 26]. Фабричний метод — це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів. Він був реалізований через мапу з ім'ям елемента та функцією створення покажчика на нього.

```
   QMap<QString, std::function<QSharedPointer<Element>()>> elementFactory;

LibreNetlistParser::LibreNetlistParser()
{
    elementFactory["variant"] = []() { return QSharedPointer<Variant>::create(); };
    elementFactory["netclass"] = []() { return QSharedPointer<NetClass>::create();
};
    elementFactory["net"] = []() { return QSharedPointer<Net>::create(); };
    elementFactory["component"] = []() { return QSharedPointer<Component>::create();
};
    elementFactory["model"] = []() { return QSharedPointer<Component>::create(); };
    elementFactory["attribute"] = []() { return QSharedPointer<Attribute>::create();
};
    elementFactory["signal"] = []() { return QSharedPointer<Signal>::create(); };
    elementFactory["device"] = []() { return QSharedPointer<Device>::create(); };
   }
```

Це дозволяє легко створити новий елемент базуючись на його типу при зчитуванні даних та не мати прив'язку до конкретного його типу, а прив'язку до абстрактного класу Element, який є батьківським класом для усіх інших елементів. Таким чином можна занести усі елементи до відповідної мапи, де ключем буде їх унікальний ідентифікатор та отримати їх, якщо інший елемент посилається на них. Оскільки attribute не має свого унікального ідентифікатора, то створюємо для нього свій.

```
Element* LibreNetlistParser::createNewElement(QString name, QString uuid)
{
    // create pointer to Element's child from the factory.
    QSharedPointer<Element> element = elementFactory[name]();
    // if it is an attribute, we need to create uuid ourselves and set the name to
the attribute
    if (element->getElementType() == "attribute") {
        QString name = uuid;
        uuid = UUIDGenerator::generateUUID();
        element->setProperty("name", name);
    }
    element->setProperty("uuid", uuid);
    elementMap[uuid] = std::move(element);
    return elementMap[uuid].get();
```

}

Під кінець потрібно лише створити об’єкт класу Circuit що містить усі потрібні дані, такі як variant, netclass та списки net та component об’єктів.

3.2 Спеціальна бібліотека LibrePCB

Щоб побудувати електричне коло для симуляції, потрібно спочатку створити спеціальну користувацьку бібліотеку для LibrePCB, яка буде мати усі потрібні елементи.

Створимо потрібну бібліотеку. Для цього потрібно відкрити менеджер бібліотек в LibrePCB і перейти у вкладку «Create Local Library». Заповнюємо усі потрібні дані у вікні та натискаємо кнопку «Create Library» (див. рис. 3.4).

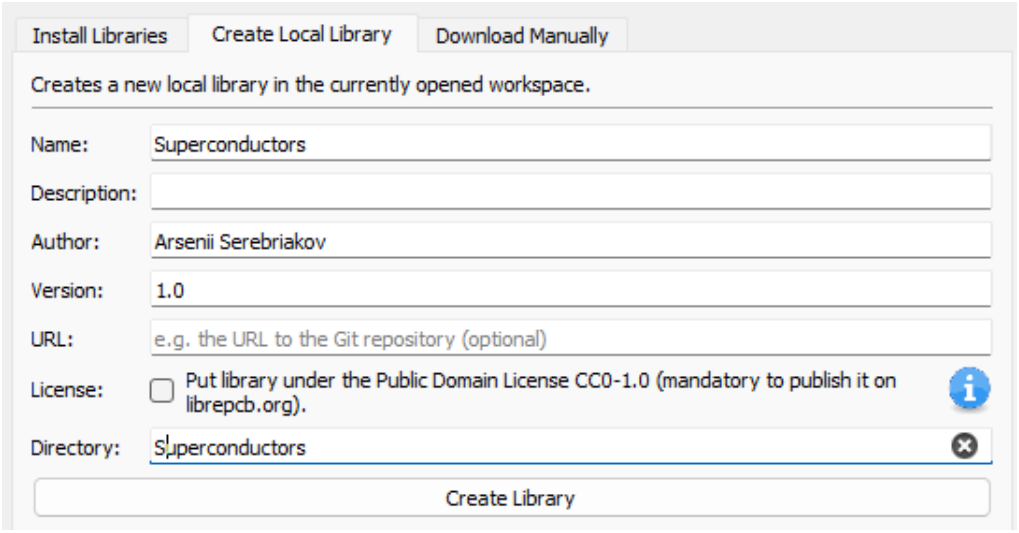

Рисунок 3.4 – Користувацька бібліотека LibrePCB

Наступним кроком потрібно створити потрібні категорії. Спочатку створимо базову категорію Superconductors в якій будемо розміщати усі інші потрібні підкатегорії. У результаті створено наступний список (див. рис. 3.5) категорій, який буде містити усі потрібні компоненти.

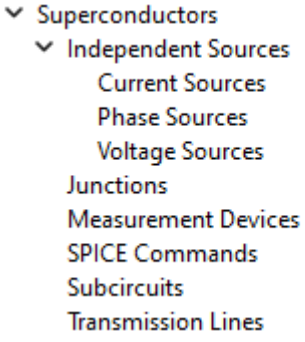

Рисунок 3.5 – Ієрархія категорій

В самому проекті вони будуть виглядати наступним чином (див. рис. 3.6). Тобто усі джерела відокремлені в окрему підкатегорію.

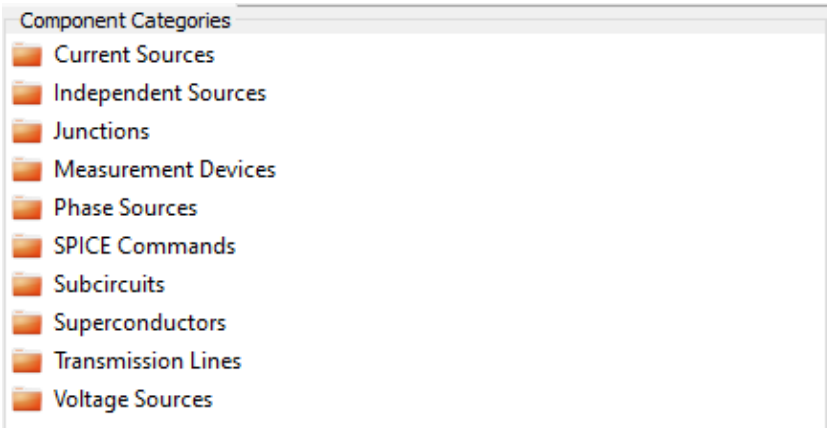

Рисунок 3.6 – Категорії компонентів

Далі потрібно створити символи (symbols) для усіх компонентів та самі компоненти. Символи будуть використовуватися для відображення їх у електричному колі, в той час як дані о компонентах будуть нести інформацію щодо їх атрибутів, префікси назв та додаткові значення, які будуть використані для ідентифікації певних особливих компонентів.

Для кожного із компоненту створено спеціальні позначки (див. рис. 3.7) та префікси, щоб відокремити їх один від одного.

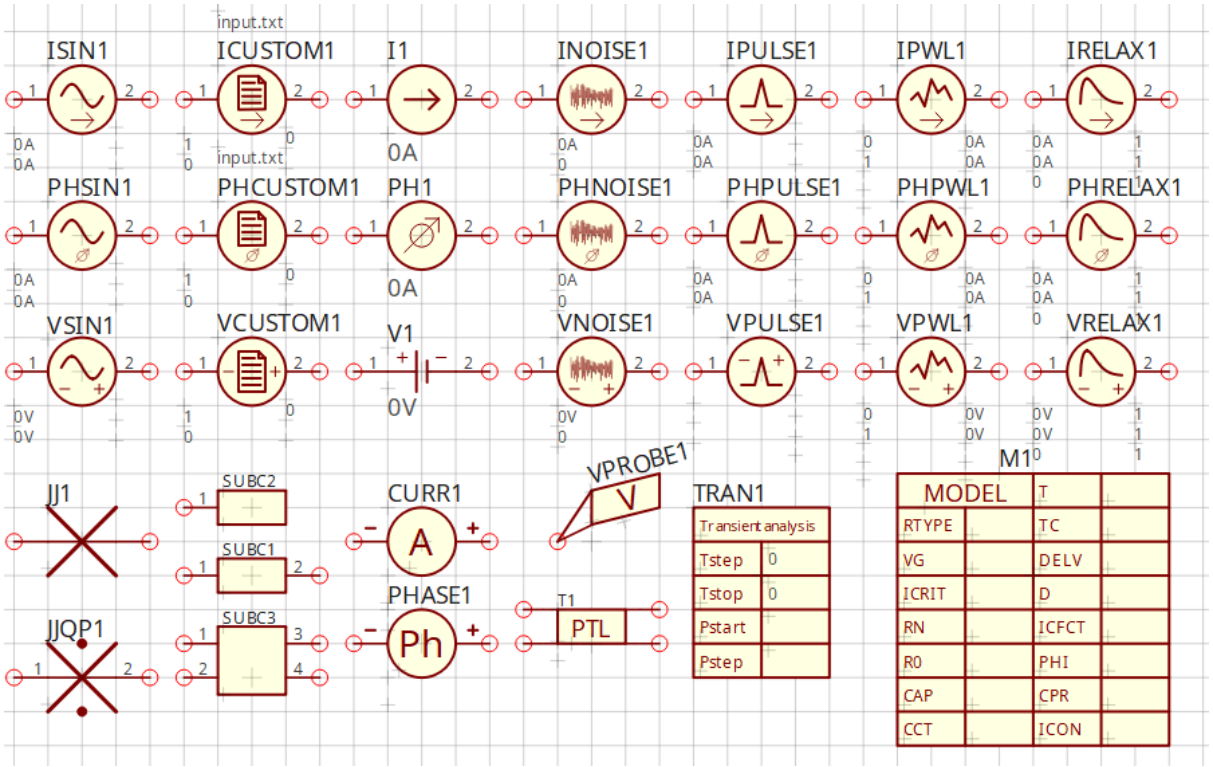

Рисунок 3.7 – Створені компоненти в LibrePCB

Також для кожного із компоненту додано список атрибутів (див. рис. 3.8) що відповідає параметрам їх рівнянь та потрібних даних.

Attributes of Component

	Key	Type	Value	Unit
1	V0	Voltage	0	V
2	V	Voltage	0	V
3	FREQ	Frequency		Hz
4	TD	String		
5	THETA	String		

Рисунок 3.8 – Атрибути компонента

Окрім цього додано базове значення (див. рис. 3.9), що описує тип компонента, якщо потрібно. Усе це дозволить легко записати їх у SPICE нотації.

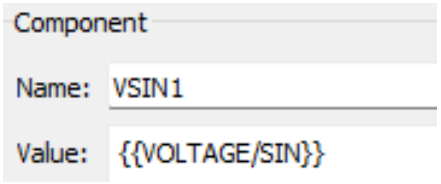

Рисунок 3.9 – Значення компонента

3.3 Електричне коло у SPICE нотації

Тепер коли є спеціальна бібліотека та механізм створення об’єкту електричного кола з усіма компонентами, можна перейти до створення запису в SPICE нотації. Спочатку створимо тестове електричне коло (див. рис. 3.10) з різними компонентами, щоб перевірити як вони будуть працювати один із одним.

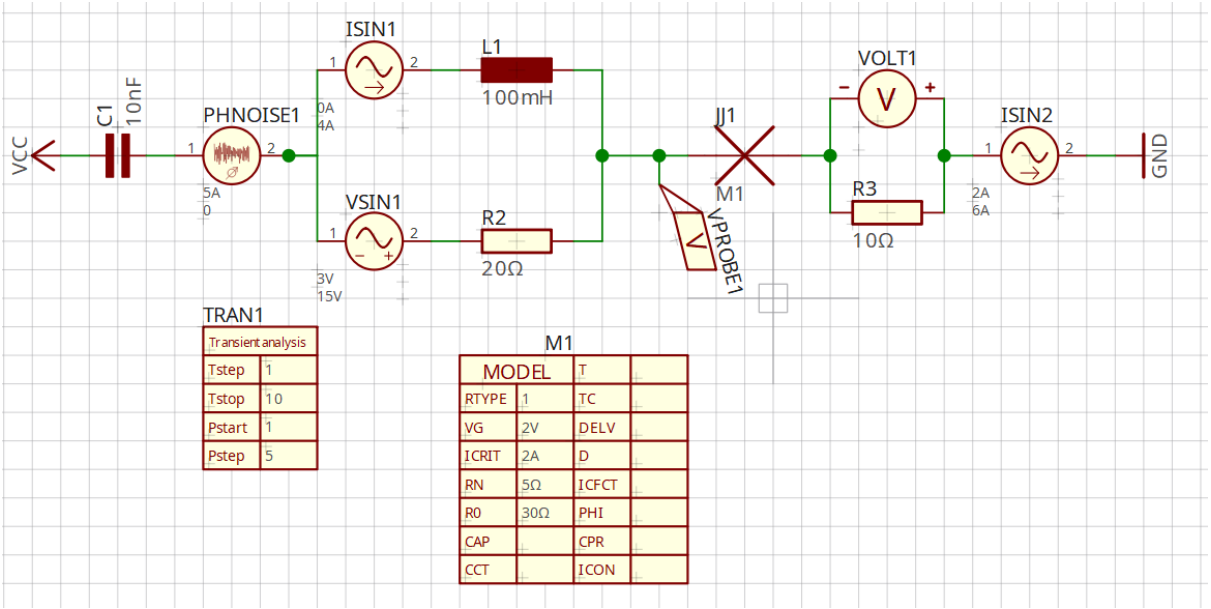

Рисунок 3.10 – Тестова схема електричного кола

Для конвертації створимо 2 класи. Один для переведення компонентів у строки, інший для організації цього запису. Основною функцією є

```
QString SpicePrinter::printComponent(Component component, QString parentUUID)
{
    QString result = getSpiceName(component.getName()) + WORD_SEPARATOR;
    QList<Attribute> list = component.getAttributeList();
    if (component.getValue() == "{{SUBCIRCUIT}}") {
        QString subcircuit = list.first().getValue().toUpper();
        result = SUBCIRCUIT.arg(component.getName(), subcircuit);
        component.removeAttribute(list.first());
    }
    if (!parentUUID.isEmpty()) {
        result += writeSignal(parentUUID, netLabelMap) + WORD_SEPARATOR;
    }
    for (Signal &signal : component.getSignalList()) {
        if (signal.getNet().getUuid() != parentUUID) {
            result += writeSignal(signal.getNet().getUuid(), netLabelMap) +
WORD_SEPARATOR;
        }
    }
    if (component.getValue() == "VCC") {
        result += "0 " + AppSettings::getVCCValue();
    }
    result += AttributeUtils::writeAttributes(component.getAttributeList(),
component.getValue());
    return result;
}
```

Вона проводить запис компонента у строковий формат. По-перше, вона отримує ім’я компонента у SPICE нотації. Потім вона додає послідовність мереж починаючи з батьківської, якщо така є. І під кінець вона використовує іншу функцію, яка додає тип компонента, якщо такий є, та атрибути.

```
QString AttributeUtils::writeAttributes(QList<Attribute> attributes, QString value,
bool includeName)
{
    QList<Attribute>::iterator begin = attributes.begin();
    QList<Attribute>::iterator end = attributes.end();
    QString sourceType = getSourceType(value);
    QString attributesLine = writeAttributes(begin, end, includeName);
    return sourceType == EMPTY_STRING ? attributesLine : sourceType.toLower() + "("
+ attributesLine + ")";
}
```

У результаті виклику даної функція на компоненті електричного кола вона повертає його запис у SPICE нотації (див. рис. 3.11).

PHNOISE1 60 20 noise(0 0 1)

Рисунок 3.11 – Результат роботи функції

Також були створені інші функції для відображення інших спеціальних об’єктів, таких як модель, аналіз перехідних процесів, вихідні дані, тощо. Їх можна побачити у додатку з кодом.

Основною функцією, що організовую процес створення запису електричного кола у SPICE нотації є (див. додаток):

```
QString SpiceNetlistProducer::produceSpiceNetlist(const Circuit &circuit, const
ConversionParams &params)
```

Вона отримує список компонентів та створює на основі них мапу, яка за ключ має унікальний ідентифікатор мережі, а за значення – сет із усіх компонентів, що підключені до цієї мережі. Це дозволяє рекурсивно пройти по всім компонентам, які підключені до мережі, та перевести їх у текстовий формат поки усі компоненти не будуть пройдені. Таким чином при записі зрозуміло порядок підключення елементів та не пропустити деякі із них.

```
QString SpiceNetlistProducer::writeComponents(QString parentSignalUuid,
                                               Component component,
                                               SpicePrinter printer,
                                               QMap<QString, QSet<Component>>
netComponentsMap,
                                               QSet<QString> *usedComponents)
{
    if (usedComponents->contains(component.getUuid())
        || component.getValue() == GROUND
        || netComponentsMap.empty()) {
        return EMPTY_STRING;
    }
    usedComponents->insert(component.getUuid());
    QString result = printer.print(component, parentSignalUuid).trimmed() +
LINE_SEPARATOR;
    for (Signal &signal : component.getSignalList()) {
        if (signal.getNet().getUuid() != parentSignalUuid) {
            QSet<Component> componentList =
netComponentsMap[signal.getNet().getUuid()];
            for (const Component &component : componentList) {
                result += writeComponents(signal.getNet().getUuid(),
                                          component,
                                          printer,
                                          netComponentsMap,
                                          usedComponents);
            }
        }
    }
    return result;
   }
```

Але окрім цього потрібно ще спочатку знайти хоча б один компонент, що підключений для мережі. Також створено мапу, яка зберігає як ключ унікальний ідентифікатор мережі, а як значення, її простий запис. Це дозволить мати зрозумілі значення при їх запису. Потім відбується запис моделей. Якщо модель була вказана в джозефсонікському контакті, але немає

наявної у схемі електричного кола, то створюємо базову модель на основі заданих налаштувань.

Також важливим моментом є створення підсхем. Якщо користувач хоче, щоб задана схема була використана, як підсхема у майбутньому, то потрібно провести відповідний аналіз. Використовуємо мапу із UUID мереж та компонентів для того, щоб знайти мережі які з'єднують компоненти лише з однієї сторони, тобто не мають компонента із іншої. Це буде означати, що ця мережа буде використана як вхідна/вихідна та дозволе під'єднати її в коло.

```
QString getSubcircuitIO(QMap<QString, QString> netNumberMap,
                        QMap<QString, QSet<Component>> netComponentsMap)
{
    QList<QString> inOutList;
    for (auto const &key : netComponentsMap.keys()) {
        QSet<Component> components = netComponentsMap[key];
        if (components.size() == ONE_CONNECTION) {
            inOutList.push_back(netNumberMap[key]);
        }
    }
    std::sort(inOutList.begin(), inOutList.end());
    QString result;
    for (QString &signalNumber : inOutList) {
        result += signalNumber + " ";
    }
    result.chop(1);
    return result;
   }
```

Якщо ж користувач хоче створити звичайну схему, то потрібно імпортувати усі підмережі, додати вивід результату та аналіз перехідних процесів, якщо вони є. У результаті отримаємо готову для симуляції схему електричного кола.

Оскільки JSIM та JoSIM майже в усьому схожі, але мають трохи деякі відмінності, то в залежності від обраного користувачем симулятору деякі елементи записуються по різному (наприклад модель джозефсонікського контакту).

Також був доданий заголовок, який можна відключити чи змінити шаблон у налаштуваннях. Він дозволяє зберегти час конвертації, ім'я проекту та під який симулятор вона була проведена.

3.4 Графічний інтерфейс

Для створення графічного інтерфейсу використано Qt Creator. Він має інструменти для створення макету та різні елементи, такі як:

- Widget – елемент який слугує контейнером для інших елементів.
- Label – відображає текст який не можна змінити.
- Button – кнопка.
- TextField – текстовий рядок, який користувач може редагувати.
- TextArea – багаторядкове поле, яке користувач може редагувати
- MenuBar – меню зверху у вікні програми.

Стиль усіх цих елементів можна змінити за допомогою CSS, а саме .qss файлу. Але він підтримує лише CSS2, тому глибокої сучасної кастомізації не вдасться зробити. Для цього розміщуємо відповідний файл з усіма стилями у ресурсах та загружаємо його при запуску програми.

Також можливо використати зображення замість тексту. Таким чином можна отримати більш зрозумілий та простіший інтерфейс. Їх також потрібно розмістити у ресурсах та додати потім при редагуванні макета.

У класі головного вікна при його ініціалізації можна створити усі потрібні об’єкти. Для того щоб прив’язати дії до відповідних кнопок – потрібно прописати усі функції які має виконувати вікно та використати функцію connect() з відповідними параметрами. Після цього графічний інтерфейс буде наділеним потрібним функціоналом.

У результаті маємо наступний графічний інтерфейс (див. рис. 3.12). Він має 2 основних текстових поля. Вони слугують для введення та виведення

запису електричного кола. Ліве вікно відповідає за запис вхідного файлу формату LibrePCB, а в той час як правий – SPICE.

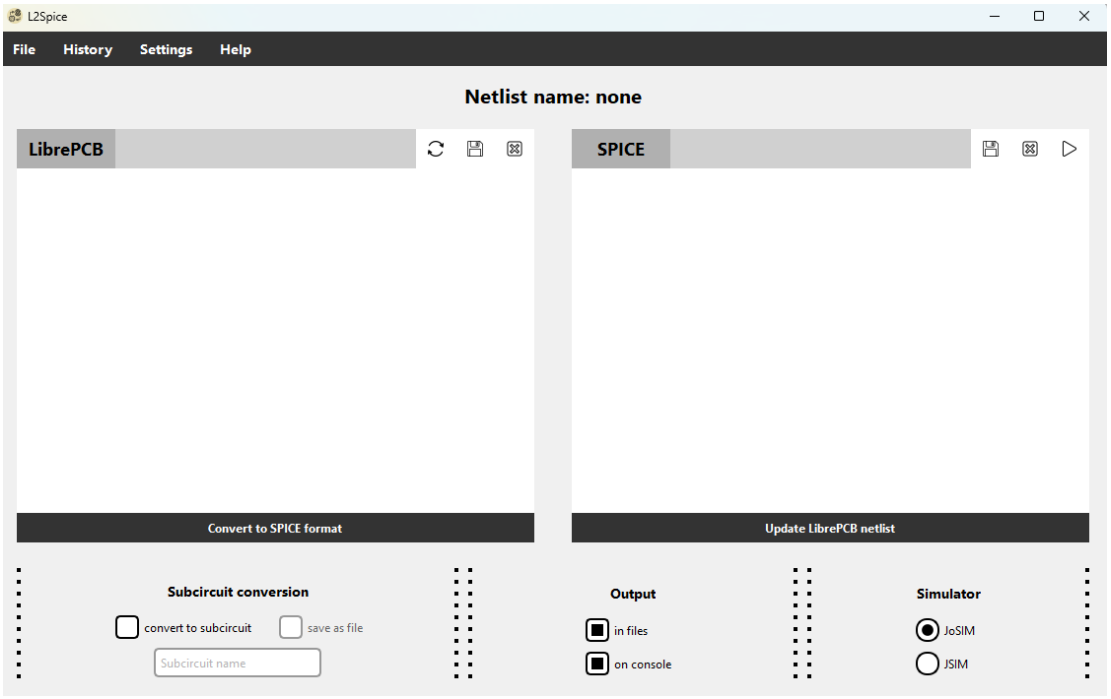

Рисунок 3.12 – Графічний інтерфейс програмного додатку

Для того щоб загрузити дані із файлу можна скористатися функцією у меню «File > Open LibrePCB Netlist» чи натиснути Ctrl + O. Також можна просто скопіювати дані із файлу та вставити у поле напряму. Після цього натискаємо кнопку «Convert to SPICE format» та отримуємо схему електричного кола записаного в SPICE нотації (див. рис. 3.13).

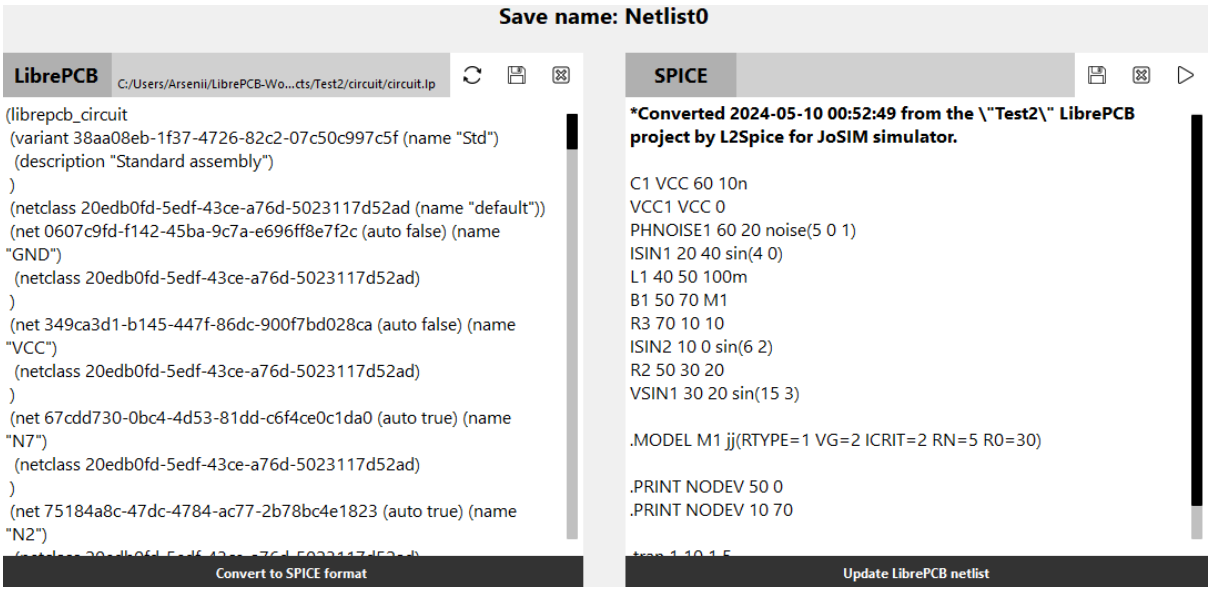

Рисунок 3.13 – Конвертація через програмний додаток

Програмний додаток містить наступний додатковий функціонал:

1. Можливість оновити атрибути у файлі LibrePCB на основі нових у запису в форматі SPICE
2. Збереження історії конвертацій та можливість переходу до передніх станів.
3. Збереження обох схем у файл. Назву файлу можна побачити над відповідним полем. Якщо вона є, то дані збережуться в нього, якщо ж ні, то відкриється меню роботи з файлами де можна буде обрати куди зберегти дані
4. Оновлення даних із файлу. Якщо файл був відкритий, але після цього в нього були внесені нові зміни, наприклад оновлено схему, то дані у полі можна оновити натиснувши відповідну кнопку.
5. Закриття файлу та очищення текстового поля. Це можна зробити натиснувши кнопку із хрестиком.
6. Запуск симуляції. Програма дозволяє одразу запустити симуляцію. Але для цього потрібно встановити симулятор окремо та вказати шлях до бінарного файлу у налаштуваннях.

Програмний додаток дозволяє змінити параметри конвертації (див. рис. 3.14). Можливо задати чи є схема підсхемою, та якщо вона є то їй можна дати назву та одразу зберегти після конвертації. Можливо також задати саме куди записано результат симуляції. Користувач може зустрітися із ситуаціями коли він хоче отримати результат лише у консоль чи лише у файл. Також програмний додаток дозволяє змінити симулятор, який можна запустити після конвертації.

Рисунок 3.14 – Параметри конвертації

Налаштування дозволяють змінити певні параметри, які будуть корисні користувачу. Це може бути шаблон заголовку, розмір історії, шлях до бінарних

файлів симулятора чи шлях до базової директорії відкриття файлу. Усі налаштування зберігаються у файлі і будуть використані при наступному відкритті програми. Також в окремому файлі зберігається базові значення моделі, які будуть використані якщо модель була вказана для джозефсонікського контакту, але не наявна в схемі. Ці значення можна змінити якщо користувач бажає цього.

3.5 Консольний інтерфейс

Програмний додаток має консольний інтерфейс. Він дозволяє швидко працювати із файлами без використання графічного інтерфейсу. Для виконання програми через консоль потрібно викликати файл L2Spice.exe з наступними аргументами командного рядка:

- -h або --help – параметр командного рядка, який використовується для відображення довідкового повідомлення для програми. При виклику, програма наддасть коротку інформацію про її використання, зокрема список доступних опцій командного рядка, їх опис та приклади їх використання.
- -v або --version – параметр командного рядка, який використовується для показу поточної версію програми.
- -i або --input – параметр командного рядка, який використовується для вказівки вхідний файл для програми. При використанні цього прапора потрібно вказати ім'я вхідного файлу одразу після прапора. Після цього програма використовуватиме вказаний файл як вхідні дані для процесу конвертації. Цей прапор є обов'язковим для використання при конвертації.
- -o або --output – параметр командного рядка, який використовується для вказівки файлу з результатом конвертації. При використанні цього прапора потрібно вказати ім'я вихідного файлу одразу після прапора. Програма запише результат перетворення у вказаний файл. Цей прапорець є обов'язковим для використання при конвертації.

- -s або --subcircuit – необов'язковий прапорець командного рядка, який використовується для того, щоб вказати програмі, що дана схема повинна бути перетворена у підсхему.
- -j або --jsim – необов'язковий прапорець командного рядка, який використовується для вказівки програмі, що дана схема повинна бути перетворена для JSIM-симулятора. За замовчуванням, програма конвертує схему для симулятора JoSIM.
- -wc або --without-console – необов'язковий прапорець командного рядка який використовується для перенаправлення виводу з файлу на консоль. Це може бути корисним для тестування, якщо користувач хоче побачити результати перетворення негайно. Зазвичай, вивід залежить від того що вказано у схемі, але за допомогою цього прапора користувач може вилучити з перетворення всі оператори «.FILE».
- -wf або --without-file – необов'язковий прапорець командного рядка, який використовується для вилучення консольного виводу з симуляції. Його можна використовувати, якщо користувач не хоче отримати консольний вивід під час симуляції.

3.6 Інсталятор

Qt Framework пропонує інструмент Qt Installer Framework. За допомогою нього можна створити будь-який інсталятор. Для цього потрібно створити директорію з мета файлами та розмістити там проект. Після цього в командній строчці потрібно запустити створення інсталятора. Він дозволяє легко встановити програмний додаток на будь-яку операційну систему.

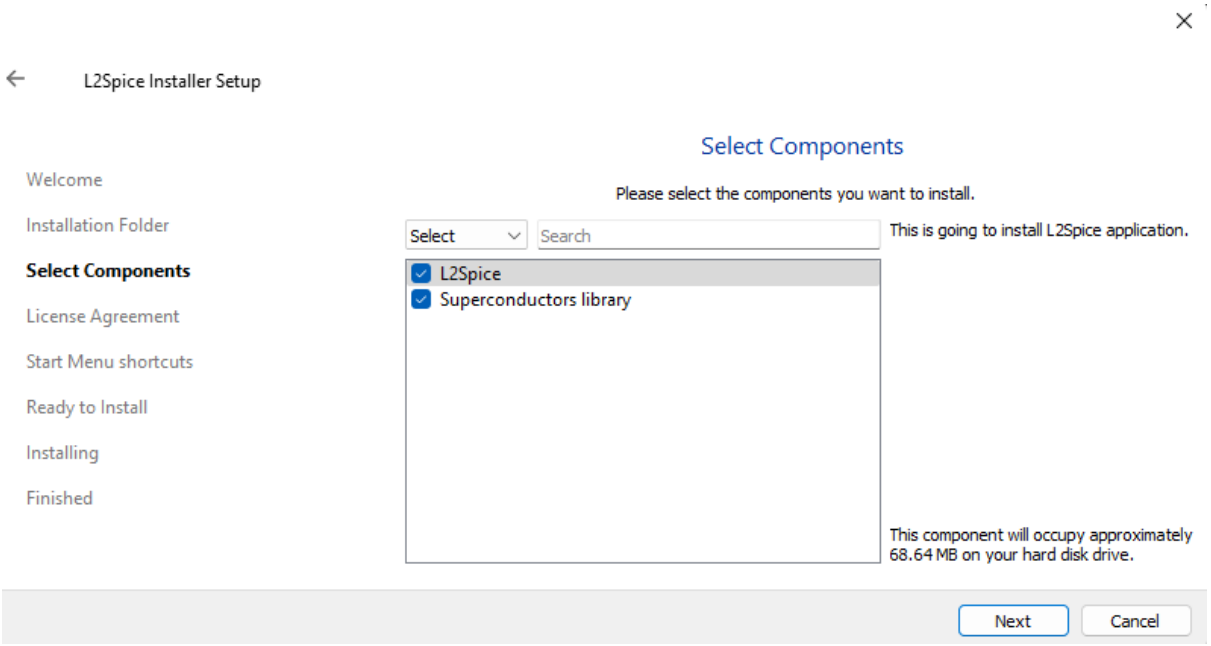

Рисунок 3.15 – Інсталятор програмного додатку

ВИСНОВКИ

В бакалаврській кваліфікаційній роботі досліджено інформаційне та програмне забезпечення системи моделювання фізичних процесів в надпровідниках. Всі поставлені завдання щодо проведення досліджень виконані. Розроблено програмний додаток оптимізації процесу моделювання-симуляції схем електричного кола з надпровідними пристроями. Результатами роботи є наступне:

1. Створено спеціальну користувацьку бібліотеку LibrePCB для моделювання схем електричного кола з потрібними компонентами.
2. Розроблено програмне забезпечення, що дозволяє отримати схему електричного кола у записі SPICE із LibrePCB схеми, так щоб можна провести її симуляцію у JSIM/JoSIM додатку.
3. Створено графічний і консольний інтерфейси для програмного додатку із можливістю запуску симуляції одразу із нього. Він має мати глибоку кастомізацію в залежності від різних потреб користувача.
4. Створено інсталятор для програмного забезпечення, що дозволяє встановити його на будь-яку операційну систему разом із спеціальною бібліотекою LibrePCB.

Застосування створеного програмного забезпечення дозволяє автоматизувати процес моделювання та симуляції схем електричних кіл з надпровідними пристроями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Barone, A., & Paternò, G. (1982). *Physics and applications of the Josephson effect*. Wiley.

2. Fedorov, A., Shnirman, A., Schön, G., & Kidiyarova-Shevchenko, A. (2007). *Reading out the state of a flux qubit by Josephson transmission line solitons*. American Physical Society.

3. Моделювання фізичних процесів: Комп'ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 105 «Прикладна фізика та наноматеріали», спеціалізації «Прикладна фізика» / КПІ ім. Ігоря Сікорського ; уклад.: Н. Ф. Димитрієва. – Електронні текстові дані (1 файл: 11,6 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 96 с.

4. Моделювання фізичних процесів із використанням технології CUDA : монографія / І. В. Гущин, В. М. Куклін, О. В. Мішин, О. В. Приймак. – Х. : ХНУ імені В. Н. Каразіна, 2017. – 116 с.

5. Combescot P. Superconductivity. An Introduction. – Cambridge University Press. 2022. – 400 p.

6. *What are Josephson junctions? How do they work?* (1997, November 24). Scientific American. https://www.scientificamerican.com/article/what-are-josephson-juncti/

7. Lerner, R. G., & Trigg, G. L. (Eds.). (2005). *Encyclopedia of physics* (3rd ed.). Wiley-VCH.

8. *LibrePCB*. (n.d.). Create electronics the easy way | LibrePCB. https://librepcb.org/

9. *A. general structure and conventions*. (n.d.). LTwiki-Wiki for LTspice. https://ltwiki.org/LTspiceHelp/LTspiceHelp/A_General_Structure_and_Conventions.htm

10. *Scientific notation in SPICE | scientific notation and metric prefixes | electronics textbook*. (n.d.). All About Circuits - Electrical Engineering & Electronics Community.

https://www.allaboutcircuits.com/textbook/direct-current/chpt-4/scientific-notation-in-spice/

11. *SPICE - a brief overview*. (n.d.). Penn Engineering | Inventing the Future. https://www.seas.upenn.edu/~jan/spice/spice.overview.html

12. *GitHub - superconductor-electronics/jsimn: The famous JSIM josephson simulator.* (n.d.). GitHub. https://github.com/Superconductor-Electronics/JSIMn

13. *JoSIM documentation*. (n.d.). https://joeydelp.github.io/JoSIM/

14. *Create a local library | librepcb documentation*. (n.d.). Create electronics the easy way | LibrePCB. https://librepcb.org/docs/quickstart/create-local-library/

15. *Syntax guide - josim documentation*. (n.d.). https://joeydelp.github.io/JoSIM/syntax/

16. Rajendran, M. (2023, October 7). *Unraveling the speed disparity: Why python lags behind java and C++*. Medium. https://medium.com/cognitivecraftsman/unraveling-the-speed-disparity-why-python-lags-behind-java-and-c-3328f8313598

17. *C++ language tutorials*. (n.d.). https://cplusplus.com/doc/tutorial/

18. *Qt | development framework for cross-platform applications*. (n.d.). Qt | Tools for Each Stage of Software Development Lifecycle. https://www.qt.io/product/framework

19. *Qt installer framework manual*. (n.d.). Qt Documentation | Home. https://doc.qt.io/qtinstallerframework/

20. *RFC 9562: Universally unique identifiers (uuids)*. (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc9562

21. *Regular-Expressions.info - regex tutorial, examples and reference - regexp patterns*. (n.d.). Regular-Expressions.info - Regex Tutorial, Examples and Reference - Regexp Patterns. https://www.regular-expressions.info/

22. Ghadage, O. P. (2021, October 26). *What is Recursive Algorithm? Types and Methods | Simplilearn*.

Simplilearn.com. https://www.simplilearn.com/tutorials/data-structure-tutorial/recursive-algorithm

23. Інформаційне та програмне забезпечення системи моделювання фізичних процесів в надпровідниках. Матер. міжн. наук. конф. молодих вчених «Інформатика, математика, автоматика, ІМА-2024», СумДУ, 2024, с. 95-96

24. CSS: Cascading style sheets | MDN. (n.d.). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/CSS

25. Factory method. (n.d.). Refactoring and Design Patterns. https://refactoring.guru/design-patterns/factory-method

26. Software design patterns tutorial - geeksforgeeks. (n.d.). GeeksforGeeks. https://www.geeksforgeeks.org/software-design-patterns/

ДОДАТОК

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSettings>

#include <src/app/app_controller.h>
#include <src/app/app_internal_storage.h>

#include <src/app/app_header.h>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:

    void convertToSpice();
    void updateLibrePCB();
    void subcircuitCheckBoxStateChanged(int state);
    void nextNetlist();
    void previousNetlist();
    void lastNetlist();
    void openDirectoryDialog();
    void saveSpice();
    void saveSpiceAs();
    void closeSpice();
    void openLibre();
    void refreshLibre();
    void saveLibre();
    void saveLibreAs();
    void closeLibre();
    void openUserManual();
    void openLibreDocumentation();
    void loadExample();
    void simulate();

private:
    AppController appController;
    AppInternalStorage* storage;
    AppHeader header;
    Ui::MainWindow *ui;
    ConversionParams lastParams;
    ConversionParams getConversionParams();
    void saveSubcircuitIfNeeded(QString subcircuit, ConversionParams params);
    void saveAndUpdateState();
    void changeState(AppState node);
    void saveSpiceNetlist(bool forcedFileDialog);
    void saveLibreNetlist(bool forcedFileDialog);
};
#endif // MAINWINDOW_H
```

```
#ifndef LIBRE_NETLIST_PARSER_H
#define LIBRE_NETLIST_PARSER_H

#include <QMap>
#include <QSharedPointer>
#include <src/circuit/circuit.h>
#include <src/circuit/element/element.h>

/**
 * LibreNetlistParser is used to parse the LibrePCB circuit.
 */
class LibreNetlistParser
{
private:
    QMap<QString, std::function<QSharedPointer<Element>()>> elementFactory;
    QString::iterator currentCharacter;
    QMap<QString, QSharedPointer<Element>> elementMap;

    QString nextWord();
    QString nextDataInQuotes();
    void parseComponents(QString::iterator last);
    void parseComponent(QString parentUuid,  QString::iterator last);
    Element* createNewElement(QString name, QString uuid);
    void parseElement(QString parentUuid, QString::iterator last);
public:
    LibreNetlistParser();
    /**
     * Parses the LibrePCB circuit provided as a QString and creates an instances of
Circuit class.
     * @param input - LibrePCB circuit.
     * @return an instances of Circuit.
     */
    Circuit parseLibreNotation(QString input);
};

#endif // LIBRE_NETLIST_PARSER_H


#include "libre_netlist_updater.h"
#include "src/utils/attribute_utils.h"
#include "src/utils/regex_utils.h"
#include "src/utils/global_variables.h"
#include <QList>
#include <QMap>
#include <QRegularExpression>
#include <functional>
#include <regex>

const QString EMPTY_STRING = "";
const QString DEFAULT_UNIT = "none";

void LibreNetlistUpdater::setSimulatorVersion(int version)
{
    simulatorVersion = version;
}

QList<QString> split(QString::iterator iterator,
                     QString::iterator end,
                     std::function<bool(QChar)> test)
{
    QList<QString> list;
    QString result;
    while (iterator != end) {
        if (test(*iterator) && !result.isEmpty()) {
            list.push_back(result);
            result = EMPTY_STRING;
        } else {
```

```
            result += *iterator;
        }
        iterator++;
    }
    if (!result.isEmpty()) {
        list.push_back(result);
    }
    return list;
}

QList<QString> splitRows(QString::iterator iterator, QString::iterator end)
{
    return split(iterator, end, [](QChar c) { return c == '\n'; });
}

QList<QString> splitParams(QString::iterator iterator, QString::iterator end){
    return split(iterator, end, [](QChar c) { return c == ' ' || c == '(' || c ==
')'; });
}

QList<QString> findAllMatches(QRegularExpression regex, QString text)
{
    QRegularExpressionMatchIterator it = regex.globalMatch(text);
    QList<QString> list;
    while (it.hasNext()) {
        QRegularExpressionMatch match = it.next();
        list.append(match.captured(0));
    }
    return list;
}

QString getAttribute(QList<QString> attributes, QString desiredName)
{
    for (QString &attribute : attributes) {
        QRegularExpressionMatch match = RegexUtils::attributeRegex.match(attribute);
        if (match.captured(1) == desiredName) {
            return match.captured(0);
        }
    }
    if (!attributes.isEmpty()) {
        return attributes.first();
    }
    return EMPTY_STRING;
}

std::string findTextByPattern(const std::string &text, const std::string &pattern)
{
    std::regex regexPattern(pattern);
    std::smatch match;
    if (std::regex_search(text, match, regexPattern)) {
        return match.str();
    }
    return EMPTY_STRING.toStdString();
}

QString getSubString(QRegularExpression regex, QString input, int captureGroup) {
    return regex.match(input).captured(captureGroup);
}

QString LibreNetlistUpdater::getNewUnit(QString param, QString attribute)
{
    QString unit = getSubString(RegexUtils::attributeRegex, attribute, 3);
    if (unit == DEFAULT_UNIT) {
        return unit;
    }
    QString unitPrefixSymbol = getSubString(RegexUtils::paramRegex, param, 2);
```

```
    QString unitFullPrefix = AttributeUtils::getFullUnitPrefix(unitPrefixSymbol);
    QString unitWithoutPrefix = AttributeUtils::getUnitWithoutPrefix(unit);
    return unitFullPrefix + unitWithoutPrefix;
}

QString LibreNetlistUpdater::getNewAttribute(QString attribute, QString value,
QString unit)
{
    QRegularExpressionMatch match = RegexUtils::attributeRegex.match(attribute);
    QStringList capturedText = match.capturedTexts();
    attribute.replace(capturedText.at(2), QString("(unit %1)").arg(unit));
    attribute.replace(capturedText.at(4), QString("(value \"%1\")").arg(value));
    return attribute;
}

QMap<QString, QString> LibreNetlistUpdater::getComponents(QString textToUpdate)
{
    QMap<QString, QString> map;
    QRegularExpressionMatchIterator it =
RegexUtils::componentRegex.globalMatch(textToUpdate);
    while (it.hasNext()) {
        QRegularExpressionMatch match = it.next();
        QString result = match.captured(0);
        QString name = getSubString(RegexUtils::nameRegex, result, 1);
        map[name] = result;
    }

    return map;
}

QString LibreNetlistUpdater::updateAttribute(QString textToUpdate,
                                             QString *component,
                                             QString param,
                                             QString attribute)
{
    QString unit = getNewUnit(param, attribute);
    int group = unit == DEFAULT_UNIT ? 0 : 1;
    QString value = getSubString(RegexUtils::paramRegex, param, group);
    QString newAttribute = getNewAttribute(attribute, value, unit);
    QString oldComponent = *component;
    component->replace(attribute, newAttribute);
    return textToUpdate.replace(oldComponent, *component);
}

QString getLibrePCBName(QString name)
{
    QRegularExpressionMatch match = RegexUtils::jjSpiceNameRegex.match(name);
    if (match.hasMatch()) {
        return "JJ" + match.captured(1);
    }
    return name;
}

bool isValidAttribute(int simulatorVersion, QString attribute)
{
    return (simulatorVersion == GlobalVariables::SIMULATOR_VERSION_JSIM
            && GlobalVariables::JSIM_MODEL_ATTRIBUTES.contains(attribute))
           || (simulatorVersion == GlobalVariables::SIMULATOR_VERSION_JOSIM
               && GlobalVariables::JOSIM_MODEL_ATTRIBUTES.contains(attribute));
}

QString LibreNetlistUpdater::update(QString textToUpdate,
                                    QString params,
                                    QMap<QString, QString> componentsMap)
{
    QList<QString> paramList = splitParams(params.begin(), params.end());
```

```
    //return base text if it starts with "." and not a model
    //remove the parameter if it is ".MODEL"
    QRegularExpressionMatch match =
RegexUtils::specialDeclaration.match(paramList.first());
    if (match.hasMatch()) {
        if (match.captured(1) != "MODEL") {
            return textToUpdate;
        }
        paramList.pop_front();
    }
    QString name = getLibrePCBName(paramList.first());
    //remove the name;
    paramList.pop_front();
    QString component = componentsMap[name];
    //erase signals;
    QList<QString> signalList = findAllMatches(RegexUtils::signalRegex, component);
    paramList.erase(paramList.begin(), paramList.begin() + signalList.size());
    //check if the first character in first attribute is a number or not
    //if it is not, that it is a source type and we don't need to modify it for now
    if (!paramList.isEmpty() &&
RegexUtils::sourceTypesRegex.match(paramList.first()).hasMatch()) {
        paramList.pop_front();
    }
    QList<QString> attributes = findAllMatches(RegexUtils::attributeRegex,
component);
    for (int i = 0; i < paramList.size(); i++) {
        QString param = paramList[i];
        QRegularExpressionMatch match = RegexUtils::paramWithName.match(param);
        QString value = match.hasMatch() ? match.captured(2) : param;
        QString attribute = getAttribute(attributes, match.captured(1));
        if(!attribute.isEmpty()) {
            textToUpdate = updateAttribute(textToUpdate, &component, value,
attribute);
            attributes.removeOne(attribute);
        }
    }
    for (QString &attribute : attributes) {
        if(isValidAttribute(simulatorVersion, attribute)) {
            textToUpdate = updateAttribute(textToUpdate, &component, EMPTY_STRING,
attribute);
        }
    }

    return textToUpdate;
}

QString LibreNetlistUpdater::removeSubcircuitImports(QString param)
{
    QRegularExpressionMatchIterator it =
RegexUtils::subcircuitRegex.globalMatch(param);
    if (it.hasNext() && it.next().captured(0) == param) {
        return param;
    }
    while (it.hasNext()) {
        param.replace(it.next().captured(0), EMPTY_STRING);
    }
    return param;
}

bool canUpdate(QString row) {
    return !RegexUtils::comment.match(row).hasMatch()
           && !row.startsWith("VCC")
           && !row.startsWith("X");
}
```

```
QString LibreNetlistUpdater::updateNetlist(QString libreNetlist, QString
spiceNetlist)
{
    spiceNetlist = removeSubcircuitImports(spiceNetlist);
    QList<QString> rows = splitRows(spiceNetlist.begin(), spiceNetlist.end());
    QMap<QString, QString> componentsMap = getComponents(libreNetlist);
    for (QString row : rows) {
        row = row.trimmed();
        if (canUpdate(row)) {
            libreNetlist = update(libreNetlist, row, componentsMap);
        }
    }
    return libreNetlist;
}
```

```
#ifndef LIBRE_NETLIST_UPDATER_H
#define LIBRE_NETLIST_UPDATER_H

#include "src/utils/global_variables.h"

#include <QString>

/**
 * LibreNetlistUpdatar is used to update the LibrePCB circuit from the SPICE
netlist.
 */
class LibreNetlistUpdater
{
private:
    int simulatorVersion = GlobalVariables::SIMULATOR_VERSION_JSIM;
    QString getNewUnit(QString param, QString attribute);
    QString getNewAttribute(QString attribute, QString number, QString unit);
    QMap<QString, QString> getComponents(QString textToUpdate);
    QString updateAttribute(QString textToUpdate,
                            QString *component,
                            QString param,
                            QString attribute);
    QString update(QString textToUpdate,
                   QString params,
                   QMap<QString, QString> componentsMap);
    QString removeSubcircuitImports(QString param);
public:
    /**
     * Sets the current simulator version.
     * @param version - new simulator version.
     */
    void setSimulatorVersion(int version);

    /**
     * Updates the LibrePCB circuit using values from the SPICE netlits.
     *
     * @param libreNetlist - old LibrePCB circuit.
     * @param spiceNetlist - new SPICE netlist.
     * @return updated version of LibrePCB circuit.
     */
    QString updateNetlist(QString libreNetlist, QString spiceNetlist);
};

#endif // LIBRE_NETLIST_UPDATER_H
```

```
#include "libre_netlist_updater.h"
#include "src/utils/attribute_utils.h"
#include "src/utils/regex_utils.h"
#include "src/utils/global_variables.h"
#include <QList>
```

```
#include <QMap>
#include <QRegularExpression>
#include <functional>
#include <regex>

const QString EMPTY_STRING = "";
const QString DEFAULT_UNIT = "none";

void LibreNetlistUpdater::setSimulatorVersion(int version)
{
    simulatorVersion = version;
}

QList<QString> split(QString::iterator iterator,
                     QString::iterator end,
                     std::function<bool(QChar)> test)
{
    QList<QString> list;
    QString result;
    while (iterator != end) {
        if (test(*iterator) && !result.isEmpty()) {
            list.push_back(result);
            result = EMPTY_STRING;
        } else {
            result += *iterator;
        }
        iterator++;
    }
    if (!result.isEmpty()) {
        list.push_back(result);
    }
    return list;
}

QList<QString> splitRows(QString::iterator iterator, QString::iterator end)
{
    return split(iterator, end, [](QChar c) { return c == '\n'; });
}

QList<QString> splitParams(QString::iterator iterator, QString::iterator end){
    return split(iterator, end, [](QChar c) { return c == ' ' || c == '(' || c ==
')'; });
}

QList<QString> findAllMatches(QRegularExpression regex, QString text)
{
    QRegularExpressionMatchIterator it = regex.globalMatch(text);
    QList<QString> list;
    while (it.hasNext()) {
        QRegularExpressionMatch match = it.next();
        list.append(match.captured(0));
    }
    return list;
}

QString getAttribute(QList<QString> attributes, QString desiredName)
{
    for (QString &attribute : attributes) {
        QRegularExpressionMatch match = RegexUtils::attributeRegex.match(attribute);
        if (match.captured(1) == desiredName) {
            return match.captured(0);
        }
    }
    if (!attributes.isEmpty()) {
        return attributes.first();
    }
```

```
    return EMPTY_STRING;
}

std::string findTextByPattern(const std::string &text, const std::string &pattern)
{
    std::regex regexPattern(pattern);
    std::smatch match;
    if (std::regex_search(text, match, regexPattern)) {
        return match.str();
    }
    return EMPTY_STRING.toStdString();
}

QString getSubString(QRegularExpression regex, QString input, int captureGroup) {
    return regex.match(input).captured(captureGroup);
}

QString LibreNetlistUpdater::getNewUnit(QString param, QString attribute)
{
    QString unit = getSubString(RegexUtils::attributeRegex, attribute, 3);
    if (unit == DEFAULT_UNIT) {
        return unit;
    }
    QString unitPrefixSymbol = getSubString(RegexUtils::paramRegex, param, 2);
    QString unitFullPrefix = AttributeUtils::getFullUnitPrefix(unitPrefixSymbol);
    QString unitWithoutPrefix = AttributeUtils::getUnitWithoutPrefix(unit);
    return unitFullPrefix + unitWithoutPrefix;
}

QString LibreNetlistUpdater::getNewAttribute(QString attribute, QString value,
QString unit)
{
    QRegularExpressionMatch match = RegexUtils::attributeRegex.match(attribute);
    QStringList capturedText = match.capturedTexts();
    attribute.replace(capturedText.at(2), QString("(unit %1)").arg(unit));
    attribute.replace(capturedText.at(4), QString("(value \"%1\")").arg(value));
    return attribute;
}

QMap<QString, QString> LibreNetlistUpdater::getComponents(QString textToUpdate)
{
    QMap<QString, QString> map;
    QRegularExpressionMatchIterator it =
RegexUtils::componentRegex.globalMatch(textToUpdate);
    while (it.hasNext()) {
        QRegularExpressionMatch match = it.next();
        QString result = match.captured(0);
        QString name = getSubString(RegexUtils::nameRegex, result, 1);
        map[name] = result;
    }

    return map;
}

QString LibreNetlistUpdater::updateAttribute(QString textToUpdate,
                                             QString *component,
                                             QString param,
                                             QString attribute)
{
    QString unit = getNewUnit(param, attribute);
    int group = unit == DEFAULT_UNIT ? 0 : 1;
    QString value = getSubString(RegexUtils::paramRegex, param, group);
    QString newAttribute = getNewAttribute(attribute, value, unit);
    QString oldComponent = *component;
    component->replace(attribute, newAttribute);
    return textToUpdate.replace(oldComponent, *component);
```

```
}

QString getLibrePCBName(QString name)
{
    QRegularExpressionMatch match = RegexUtils::jjSpiceNameRegex.match(name);
    if (match.hasMatch()) {
        return "JJ" + match.captured(1);
    }
    return name;
}

bool isValidAttribute(int simulatorVersion, QString attribute)
{
    return (simulatorVersion == GlobalVariables::SIMULATOR_VERSION_JSIM
            && GlobalVariables::JSIM_MODEL_ATTRIBUTES.contains(attribute))
           || (simulatorVersion == GlobalVariables::SIMULATOR_VERSION_JOSIM
               && GlobalVariables::JOSIM_MODEL_ATTRIBUTES.contains(attribute));
}

QString LibreNetlistUpdater::update(QString textToUpdate,
                                    QString params,
                                    QMap<QString, QString> componentsMap)
{
    QList<QString> paramList = splitParams(params.begin(), params.end());
    //return base text if it starts with "." and not a model
    //remove the parameter if it is ".MODEL"
    QRegularExpressionMatch match =
RegexUtils::specialDeclaration.match(paramList.first());
    if (match.hasMatch()) {
        if (match.captured(1) != "MODEL") {
            return textToUpdate;
        }
        paramList.pop_front();
    }
    QString name = getLibrePCBName(paramList.first());
    //remove the name;
    paramList.pop_front();
    QString component = componentsMap[name];
    //erase signals;
    QList<QString> signalList = findAllMatches(RegexUtils::signalRegex, component);
    paramList.erase(paramList.begin(), paramList.begin() + signalList.size());
    //check if the first character in first attribute is a number or not
    //if it is not, that it is a source type and we don't need to modify it for now
    if (!paramList.isEmpty() &&
RegexUtils::sourceTypesRegex.match(paramList.first()).hasMatch()) {
        paramList.pop_front();
    }
    QList<QString> attributes = findAllMatches(RegexUtils::attributeRegex,
component);
    for (int i = 0; i < paramList.size(); i++) {
        QString param = paramList[i];
        QRegularExpressionMatch match = RegexUtils::paramWithName.match(param);
        QString value = match.hasMatch() ? match.captured(2) : param;
        QString attribute = getAttribute(attributes, match.captured(1));
        if(!attribute.isEmpty()) {
            textToUpdate = updateAttribute(textToUpdate, &component, value,
attribute);
            attributes.removeOne(attribute);
        }
    }
    for (QString &attribute : attributes) {
        if(isValidAttribute(simulatorVersion, attribute)) {
            textToUpdate = updateAttribute(textToUpdate, &component, EMPTY_STRING,
attribute);
        }
    }
```

```
    return textToUpdate;
}

QString LibreNetlistUpdater::removeSubcircuitImports(QString param)
{
    QRegularExpressionMatchIterator it =
RegexUtils::subcircuitRegex.globalMatch(param);
    if (it.hasNext() && it.next().captured(0) == param) {
        return param;
    }
    while (it.hasNext()) {
        param.replace(it.next().captured(0), EMPTY_STRING);
    }
    return param;
}

bool canUpdate(QString row) {
    return !RegexUtils::comment.match(row).hasMatch()
           && !row.startsWith("VCC")
           && !row.startsWith("X");
}

QString LibreNetlistUpdater::updateNetlist(QString libreNetlist, QString
spiceNetlist)
{
    spiceNetlist = removeSubcircuitImports(spiceNetlist);
    QList<QString> rows = splitRows(spiceNetlist.begin(), spiceNetlist.end());
    QMap<QString, QString> componentsMap = getComponents(libreNetlist);
    for (QString row : rows) {
        row = row.trimmed();
        if (canUpdate(row)) {
            libreNetlist = update(libreNetlist, row, componentsMap);
        }
    }
    return libreNetlist;
}


#ifndef SPICE_NETLIST_PRODUCER_H
#define SPICE_NETLIST_PRODUCER_H

#include "src/conversion/output/spice_printer.h"
#include <src/conversion/data/conversion_params.h>
#include <src/circuit/circuit.h>

/**
 * SpiceNetlistProducer is used to produce a SPICE netlist from the Circuit object
and with given params.
 */
class SpiceNetlistProducer
{
private:
    QString writeComponents(QString parentSignalUuid,
                            Component component,
                            SpicePrinter printer,
                            QMap<QString, QSet<Component>> netComponentsMap,
                            QSet<QString> *usedComponents);

public:
    /**
     * Creates a SPICE netlist.
     *
     * @param circuit - instance of the Circuit class.
     * @param params - instance of the ConversionParams class.
     * @return SPICE netlist.
     */
```

```
    QString produceSpiceNetlist(const Circuit &circuit, const ConversionParams
&params);
};

#endif // SPICE_NETLIST_PRODUCER_H


#include "spice_netlist_producer.h"

#include <QMap>
#include <QRegularExpression>
#include <QSet>

#include <src/app/app_settings.h>
#include <src/file/file_manager.h>
#include <src/utils/attribute_utils.h>
#include <src/utils/regex_utils.h>

const QString EMPTY_STRING = QString();
const QString LINE_SEPARATOR = QString("<br>");
const QString SUBCIRCUIT = QString(".SUBCKT %1 0
%2<br><br>*circuit<br>%3<br>.ENDS<br>");
const int ONE_CONNECTION = 1;
const int CONNECTION_ID_MULTIPLIER = 10;
const QString SUBCIRCUIT_VALUE = "{{SUBCIRCUIT}}";
const QString SUBCIRCUIT_NAME = "SUBCIRCUIT_NAME";
const QString GROUND = QString("GND");
const QString GROUND_ID = QString("0");
const QString END = QString(".end");
const QString DEFAULT_MODEL = QString(".MODEL %1 jj(%2)");
const QString MODEL_NAME = QString("name");

QString SpiceNetlistProducer::writeComponents(QString parentSignalUuid,
                                              Component component,
                                              SpicePrinter printer,
                                              QMap<QString, QSet<Component>>
netComponentsMap,
                                              QSet<QString> *usedComponents)
{
    if (usedComponents->contains(component.getUuid())
        || component.getValue() == GROUND
        || netComponentsMap.empty()) {
        return EMPTY_STRING;
    }
    usedComponents->insert(component.getUuid());
    QString result = printer.print(component, parentSignalUuid).trimmed() +
LINE_SEPARATOR;
    for (Signal &signal : component.getSignalList()) {
        if (signal.getNet().getUuid() != parentSignalUuid) {
            QSet<Component> componentList =
netComponentsMap[signal.getNet().getUuid()];
            for (const Component &component : componentList) {
                result += writeComponents(signal.getNet().getUuid(),
                                          component,
                                          printer,
                                          netComponentsMap,
                                          usedComponents);
            }
        }
    }
    return result;
}

QMap<QString, QSet<Component>> createNetComponentsMap(QList<Component> components)
{
    QMap<QString, QSet<Component>> map;
    for (Component &component : components) {
```

```
        QList<Signal> signalList = component.getSignalList();
        for (Signal &signal : signalList) {
            QSet<Component> components = map[signal.getNet().getUuid()];
            components.insert(component);
            map[signal.getNet().getUuid()] = components;
        }
    }
    return map;
}

Component findComponent(QString uuid, QList<Component> componentMap)
{
    for (Component &component : componentMap) {
        for (Signal &signal : component.getSignalList()) {
            if (signal.getNet().getUuid() == uuid) {
                return component;
            }
        }
    }
    return Component();
}

QString getSubcircuitIO(QMap<QString, QString> netNumberMap,
                        QMap<QString, QSet<Component>> netComponentsMap)
{
    QList<QString> inOutList;
    for (auto const &key : netComponentsMap.keys()) {
        QSet<Component> components = netComponentsMap[key];
        if (components.size() == ONE_CONNECTION) {
            inOutList.push_back(netNumberMap[key]);
        }
    }
    std::sort(inOutList.begin(), inOutList.end());
    QString result;
    for (QString &signalNumber : inOutList) {
        result += signalNumber + " ";
    }
    result.chop(1);
    return result;
}

void writeSubcircuit(QMap<QString, QString> *subcircuits, QString subcircuitName)
{
    if ((*subcircuits).contains(subcircuitName)) {
        return;
    }
    QString subcircuitFile = AppSettings::getSubcircuitDir() + "/" + subcircuitName
+ ".cir";
    QString subcircuit = FileManager::loadFile(subcircuitFile);
    subcircuit.replace("\n", LINE_SEPARATOR);
    (*subcircuits)[subcircuitName] = subcircuit;
    QRegularExpressionMatchIterator it =
RegexUtils::subcircuitIdentifierRegex.globalMatch(subcircuit);
    while (it.hasNext()) {
        QRegularExpressionMatch match = it.next();
        writeSubcircuit(subcircuits, match.captured(1));
    }
}

void writeSubcircuit(QMap<QString, QString> *subcircuits, Component component)
{
    for (Attribute &attribute : component.getAttributeList()) {
        if (attribute.getName() == SUBCIRCUIT_NAME) {
            writeSubcircuit(subcircuits, attribute.getValue());
            break;
        }
```

```
    }
}

QString getAllSubcircuits(QSet<QString> usedComponents, QList<Component> components)
{
    QMap<QString, QString> subcircuits;
    for (Component &component : components) {
        if (usedComponents.contains(component.getUuid())
            && component.getValue() == SUBCIRCUIT_VALUE) {
            writeSubcircuit(&subcircuits, component);
        }
    }
    QString result;
    for (const auto &subcircuit : subcircuits) {
        if (!subcircuit.isEmpty()) {
            result += subcircuit + LINE_SEPARATOR + LINE_SEPARATOR;
        }
    }
    return result;
}

QMap<QString, QString> createNetLabelMap(QList<Net> netMap)
{
    QMap<QString, QString> netNumberMap;
    QSet<QString> usersNetLabels;
    for (const auto &net : netMap) {
        QString name = net.getName();
        if (name == GROUND || name.isEmpty()) {
            netNumberMap.insert(net.getUuid(), GROUND_ID);
        } else if (net.getAutoMode() == false) {
            netNumberMap[net.getUuid()] = net.getName();
            usersNetLabels.insert(net.getName());
        }
    }
    int counter = 1;
    for (const auto &net : netMap) {
        if (netNumberMap.contains(net.getUuid())) {
            continue;
        }
        QString number;
        do {
            number = QString::number(CONNECTION_ID_MULTIPLIER * counter++);
        } while (usersNetLabels.contains(number));
        netNumberMap[net.getUuid()] = number;
    }
    return netNumberMap;
}

QString getFirstComponentUuid(QMap<QString, QString> netNumberMap)
{
    for (const auto &key : netNumberMap.keys()) {
        if (netNumberMap[key] != GROUND_ID) {
            return key;
        }
    }
    return EMPTY_STRING;
}

QSet<QString> findAllUsedModel(QString netlist)
{
    QSet<QString> usedModels;
    QRegularExpressionMatchIterator it =
RegexUtils::jjSpiceRow.globalMatch(netlist);
    while (it.hasNext()) {
        usedModels.insert(it.next().captured(1));
    }
```

```
    return usedModels;
}

Component createFakeModel()
{
    Component modelComponent;
    modelComponent.setProperty("value", "{{MODEL/JJ}}");
    QList<QPair<QString, QString>> model = AppSettings::getDefaultModel();
    for (QPair<QString, QString> &pair : model) {
        QSharedPointer<Element> attribute = QSharedPointer<Attribute>::create();
        attribute->setProperty("name", pair.first);
        attribute->setProperty("value", pair.second);
        modelComponent.setProperty("attribute", attribute.get());
    }
    return modelComponent;
}

QString SpiceNetlistProducer::produceSpiceNetlist(const Circuit &circuit, const
ConversionParams &params)
{
    QList<Component> components = circuit.getComponents();
    QMap<QString, QSet<Component>> netComponentsMap =
createNetComponentsMap(components);
    QMap<QString, QString> netLabelMap = createNetLabelMap(circuit.getNets());
    Component component = findComponent(getFirstComponentUuid(netLabelMap),
components);
    QSet<QString> usedComponents;
    SpicePrinter printer(netLabelMap, netComponentsMap, params);
    QString netlist = writeComponents(EMPTY_STRING,
                                      component,
                                      printer,
                                      netComponentsMap,
                                      &usedComponents);
    if (!circuit.getTexts().empty()) {
        netlist = printer.printTexts(circuit.getTexts()) + netlist;
    }

    QSet<QString> usedModels = findAllUsedModel(netlist);
    if (!usedModels.empty()) {
        netlist += LINE_SEPARATOR;
        for (Component &model : circuit.getModels()) {
            netlist += printer.print(model) + LINE_SEPARATOR;
            usedModels.remove(model.getName());
        }
        Component fakeModel = createFakeModel();
        for (const QString &modelName : usedModels) {
            fakeModel.setProperty(MODEL_NAME, modelName);
            netlist += printer.print(fakeModel) + LINE_SEPARATOR;
        }
    }
    if (params.getSubcircuitStatus()) {
        QString io = getSubcircuitIO(netLabelMap, netComponentsMap);
        netlist = SUBCIRCUIT.arg(params.getSubcircuitName(), io, netlist);
    } else {
        QString subcircuits = getAllSubcircuits(usedComponents, components);
        netlist = subcircuits + netlist;

        if (!circuit.getOutputs().empty() && (params.getConsoleOutput() ||
params.getFileOutput())) {
            netlist += LINE_SEPARATOR + printer.printOutputs(circuit.getOutputs());
        }
        if (!circuit.getTran().getName().isEmpty()) {
            netlist += LINE_SEPARATOR + printer.print(circuit.getTran());
        }
        if (!netlist.trimmed().isEmpty()) {
            netlist += LINE_SEPARATOR + END;
```

```
        }
    }
    return netlist.trimmed();
}


#ifndef CONSOLE_APP_H
#define CONSOLE_APP_H

#include <src/app/app_controller.h>
#include <src/app/app_header.h>
#include <src/file/file_manager.h>

class ConsoleApplication
{
    AppController appController;
    AppHeader appHeader;
    int argc;
    char **argv;

public:
    ConsoleApplication(int argc, char **argv);
    int exec();
private:
    QString convertHtmlToPlain(QString);
};

#endif // CONSOLE_APP_H


#include "console_app.h"
#include "src/console/flag.h"
#include "src/utils/global_variables.h"
#include <iostream>
#include <qfileinfo.h>
#include <src/utils/text_utils.h>
#include <vector>

const Flag HELP_FLAG = Flag("-h", "--help");
const Flag VERSION_FLAG = Flag("-v", "--version");
const Flag INPUT_FLAG = Flag("-i", "--input");
const Flag OUTPUT_FLAG = Flag("-o", "--output");
const Flag SUBCIRCUIT_FLAG = Flag("-s", "--subcircuit");
const Flag WITHOUT_FILE_FLAG = Flag("-wf", "--without-file");
const Flag WITHOUT_CONSOLE_FLAG = Flag("-wc", "--without-console");
const Flag JSIM_FLAG = Flag("-j", "--jsim");

ConsoleApplication::ConsoleApplication(int argc, char **argv)
    : argc(argc)
    , argv(argv)
{}

std::string get_option(const std::vector<std::string> &args, const Flag flag)
{
    for (auto it = args.begin(), end = args.end(); it != end; ++it) {
        if (*it == flag.getFlag() || *it == flag.getAlias()) {
            if (it + 1 != end) {
                return *(it + 1);
            }
        }
    }
    return "";
}

bool has_option(const std::vector<std::string> &args, const Flag flag)
{
    for (auto it = args.begin(), end = args.end(); it != end; ++it) {
```

```
        if (*it == flag.getFlag() || *it == flag.getAlias()) {
            return true;
        }
    }
    return false;
}

ConversionParams getConversionParams(std::vector<std::string> args) {
    bool subcircuitStatus = has_option(args, SUBCIRCUIT_FLAG);
    QString subcircuitName = QString::fromStdString(get_option(args,
SUBCIRCUIT_FLAG));
    bool fileOutput = !has_option(args, WITHOUT_FILE_FLAG);
    bool consoleOutput = !has_option(args, WITHOUT_CONSOLE_FLAG);
    int converterVersion = has_option(args, JSIM_FLAG) ?
GlobalVariables::SIMULATOR_VERSION_JSIM
                                                       :
GlobalVariables::SIMULATOR_VERSION_JOSIM;
    return ConversionParams(subcircuitStatus,
                            subcircuitName,
                            fileOutput,
                            consoleOutput,
                            converterVersion);
}

int ConsoleApplication::exec()
{
    const std::vector<std::string> args(argv + 1, argv + argc);
    if (argc > 16) {
        throw std::runtime_error("Too many input parameters!");
    }
    if (has_option(args, HELP_FLAG)) {
        QString help = FileManager::loadFile("settings/help.txt");
        std::cout << help.toStdString() << std::endl;
        return 0;
    }
    if (has_option(args, VERSION_FLAG)) {
        std::cout << GlobalVariables::VERSION << std::endl;
        return 0;
    }
    bool hasInput = has_option(args, INPUT_FLAG);
    bool hasOutput = has_option(args, OUTPUT_FLAG);
    if (!hasInput || !hasOutput) {
        throw std::runtime_error("Input and output files must be provided!");
    }

    QString inputFileName = QString::fromStdString(get_option(args, INPUT_FLAG));
    QString outputFileName = QString::fromStdString(get_option(args, OUTPUT_FLAG));
    QString result;
    if (QFileInfo(inputFileName).suffix() == "lp") {
        QString input = FileManager::loadFile(inputFileName);
        ConversionParams conversionParams = getConversionParams(args);
        result = appController.convertToSpice(input, conversionParams);
    }
    if (QFileInfo(inputFileName).suffix() == "cir" &&
QFileInfo(outputFileName).suffix() == "lp") {
        QString newSpice = FileManager::loadFile(inputFileName);
        QString oldLibre = FileManager::loadFile(outputFileName);
        result = appController.updateLibre(oldLibre, newSpice);
    }
    result = TextUtils::convertHtmlToPlain(result);
    bool fileIsSaved = FileManager::save(outputFileName, result);
    if (!fileIsSaved) {
        std::cout << "Cannot save the output file" << std::endl;
        return 1;
    }
    std::cout << "Converted successfully!" << std::endl;
```

```
    return 0;
}
```