Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I
Effiziente Algorithmen und wissensbasierte Systeme

Praktikumsbericht

Zeichnen von Schaltplänen mittels KLay Layered aus KIELER

Johannes Zink

Eingereicht am 26.09.2016
Überarbeitet am 10.07.2017

Betreuer:
Prof. Dr. Alexander Wolff
Fabian Lipp, M. Sc.

Inhaltsverzeichnis

1 Einführung

Ein Problem aus der Graphentheorie ist das Zeichnen von Graphen. Dieses findet beispielsweise beim Zeichnen von Schaltplänen, die als Graph aufgefasst werden können, eine praktische Anwendung. Hierfür gibt es verschiedene Ansätze. Ein solcher ist das kräftebasierte Zeichnen. Ein anderer ist das hierarchische Zeichnen von Graphen.

1.1 Hierarchisches Zeichnen nach Sugiyama et al.

Das hierarchische Zeichnen von Graphen wurde von Sugiyama et al. 1981 vorgestellt [STT81]. Der Algorithmus aus dieser Arbeit nimmt einen gerichteten Graphen entgegen und gibt eine Zeichnung zurück, bei der die Knoten auf übereinanderliegenden horizontalen Linien (Lagen) angeordnet sind. Die gerichteten Kanten verlaufen dabei überwiegend nach oben (alternativ nach unten). Intern läuft der Algorithmus in fünf Phasen ab, das sind in dieser Reihenfolge:

1. Gerichtete Kreise aufbrechen
2. Zuweisung der Knoten auf Lagen
3. Anzahl der Kreuzungen reduzieren
4. Positionierung der Knoten innerhalb ihrer Lagen
5. Zeichnen der Kanten

Da einige der Phasen im Allgemeinen NP-schweren Optimierungsproblemen entsprechen, ist für die optimale Lösung kein effizienter Algorithmus bekannt. Stattdessen wird auf Polynomialzeit-Heuristiken zurückgegriffen, die in der Praxis ordentliche Ergebnisse liefern.

1.2 KIELER und KLay Layered

Das *Kiel Integrated Environment for Layout Eclipse Rich Client* (kurz: *KIELER*) ist ein Forschungsprojekt der Universität Kiel, um das graphische modellbasierte Design von komplexen Systemen zu verbessern, hauptsächlich für automatisches Zeichnen von graphischen Komponenten [KIE]. Als Algorithmen zum Graphzeichnen stehen im KIELER-Projekt neben externen Algorithmen wie denen von Graphviz auch eigene Algorithmen zur Verfügung. Diese bilden das Unterprojekt *KLay* und sind in Java implementiert [KLa]. Das sind:

- *KLay Planar*: Algorithmus zum orthogonalen Zeichnen von Graphen, ausgerichtet auf planare oder fast planare Graphen.
- *KLay Tree*: Algorithmus zum Zeichnen von Graphen, ausgerichtet auf Bäume.
- *KLay Force*: Kräftebasierter Ansatz, der derzeit keine Ports zu unterstützten scheint.

- *KLay Layered*: Hierarchischer Ansatz basierend auf dem Algorithmus von Sugiyama et al., der um *Ports* und rekursives Zeichnen erweitert wurde. Ports können an Knoten platziert werden und als Start- bzw. Endpunkte von Kanten gesetzt werden. Rekursives Zeichnen meint hier, dass innerhalb jedes Knoten ein weiterer Graph gezeichnet werden kann und dort auch wieder in jedem Knoten ein weiterer Graph usw. Labels (Beschriftungen) werden für Knoten, Kanten und Ports unterstützt. Die Ports können an ihren Knoten mit vier Freiheitsgraden gesetzt werden. Das sind die PortConstraints „FREE“ (der Algorithmus bringt den Port irgendwo am Knoten an), „FIXED_SIDE“ (der Algorithmus bringt den Port irgendwo an einer bestimmten Seite des Knotens an), „FIXED_ORDER“ (der Algorithmus bringt den Port gemäß einer angegebenen Reihenfolge an einer festgelegten Seite des Knotens an) und „FIXED_POSITION“ (die Positionen der Ports relativ zum Knoten werden vom Anwender von Anfang an festgelegt). In dieser Reihenfolge lassen die vier Konfigurationen dem Algorithmus KLay Layered mehr Freiheit zur Platzierung der Ports („FREE“ volle Freiheit; „FIXED_POSITION“ keine Freiheit). Für die Größe und Position der Knoten und Ports können eigene Werte gesetzt werden.

2 Rahmenbedingungen

Das Praktikum wurde in Zusammenarbeit mit einem Uni-externen Kooperationspartner unter Begleitung von Professor Alexander Wolff und Fabian Lipp von der Universität Würzburg abgehalten. Das entstandene Programm kommt beim Kooperationspartner auch zur praktischen Verwendung.

2.1 Anforderungen

Gefordert war ein Programm, das aus einer Datei mit Daten eines Schaltplans im JSON-Format automatisch eine Zeichnung generiert. Das Programm soll auch in der Praxis zum Einsatz kommen. Die Schaltpläne enthalten:

- Knoten, die vom Typ *Device*, *DeviceConnector*, *Connector* und *Splice* sind und gemäß ihres Typs verschieden dargestellt werden sollen.
- Pins, die je einem Knoten zugeordnet sind.
- Wires (Kabel oder Drähte), bei denen jedes an zwei oder mehr Pins angeschlossen ist. Mehrere Wires können als Multiwires zusammengefasst sein. Das sind in der Schaltplanzeichnung weiterhin einzelne Wires, deren Zusammengehörigkeit gekennzeichnet werden kann. In der Praxis können das Kabel oder Kabelstränge sein.
- Pin2Pins, das sind Paare von Pins, die in der Praxis zusammengesteckt werden und so die zugehörigen Knoten, in dem Fall Devices und DeviceConnectors oder zwei Connectors verbinden. In der Schaltplanzeichnung sollen die so verbundenen Pins einschließlich ihrer Knoten beieinander liegend gezeichnet werden.

Mit der Menge von Pin2Pins werden implizit Knotenvereinigungen definiert. Es wird angenommen, dass in jedem gültigen Schaltplan jede Knotenvereinigung einer der folgenden entspricht:

- Ein Connector wird mit einem anderen Connector vereinigt ($1:1$ Verhältnis)
- Ein Device wird mit einem oder mehreren DeviceConnectors vereinigt ($1:n$ Verhältnis)

In solchen Knotenvereinigungen sollen einerseits die Einzelbestandteile erkennbar bleiben, aber andererseits soll auch erkennbar sein, dass es sich um eine Knotenvereinigung handelt.

Die zurückgegebene Zeichnung sollte möglichst übersichtlich sein. Welche Bauteile wie mit welchen anderen Bauteilen verdrahtet sind, sollte schnell und eindeutig erkennbar sein. Hierzu sollten Kantenkreuzungen, Kantenüberlagerungen und Kantenknicke vermieden werden. Ein orthogonaler Zeichenstil wurde zwar nicht ausdrücklich gefordert, erschien in Anlehnung an handgezeichnete Vorlagen aber als eindeutig sinnvoll.

2.2 Lösungsansatz

Der Schaltplan kann als ungerichteter Hypergraph aufgefasst werden. Dabei entspricht die Knotenmenge den implizit definierten Knotenvereinigungen vereinigt mit den Knoten ohne Pin2Pin-Verbindungen. Die Kantenmenge entspricht den Wires.
Zur Umsetzung der Aufgabe wurde KLay Layered aus KIELER verwendet. Gründe hierfür waren:

- Ports werden unterstützt und werden zur Darstellung der Pins genutzt. Da diese nicht in KLay Force unterstützt werden, schied dieser beispielsweise aus.
- Für Objekte der Zeichnung können Beschriftungen gesetzt werden, in die kein anderes Objekt hineinragt.
- Kanten werden im orthogonalen Zeichenstil gezeichnet.
- Ursprünglich sollten bestimmte Devices in der obersten Lage und bestimmte Devices in der untersten Lage gezeichnet werden. KLay Layered unterstützt das. Diese Anforderung wurde später teilweise zurückgenommen.
- Durch Heuristiken zur Anordnung der Knoten, Kanten und Ports wird die Anzahl der Kantenkreuzungen und Kantenknicke gering gehalten. So können übersichtliche Zeichnungen erzeugt werden.

Da KLay Layered einen gerichteten Graphen im KIELER-internen Java-Format entgegen nimmt, bestand die Hauptaufgabe darin, die erhaltenen Daten in das passende Format zu übersetzen und die Ausgabe gegebenenfalls noch so nachzubearbeiten, dass mit der resultierenden Zeichnung den Anforderungen entsprochen wird.

3 Umsetzung

Die Umsetzung erfolgt über ein Java-Programm. Verwendet wird Java 1.7 und das KIELER Pragmatics Release 2015/06. Das Programm nimmt Schaltplan-Daten als Java-Objekt oder json-Datei entgegen und kann den gezeichneten Schaltplan als svg-Datei ausgeben. Das Programm läuft in 3 Schritten ab. Das sind:

1. *Übersetzungsphase (Preprocessing)*: Die Schaltplan-Daten im speziellen, vom Anwender vorgegebenen Format werden aufbereitet und ins KIELER-Format übersetzt. Dabei werden Informationen aus der Eingabe mit KIELER-Objekten verknüpft, um eine Rückübersetzung und die Nachbearbeitung möglich zu machen.

2. *Layoutingphase*: Dem Algorithmus KLay Layered werden die übersetzten Daten zum Zeichnen übergeben. Dieser gibt eine Zeichnung Z_0 zurück.

3. *Nachbearbeitungsphase (Postprocessing)*: Da KLay Layered nicht alle Anforderungen an die Zeichnung nativ unterstützt, werden verschiedene Ergänzungen und Korrekturen, wie das Aufteilen von zusammengesetzten Knoten, nachträglich an Z_0 durchgeführt. Die Informationen hierfür stammen aus der Übersetzungsphase. Am Ende dieser Phase liegt die fertige Zeichnung Z_1 vor.

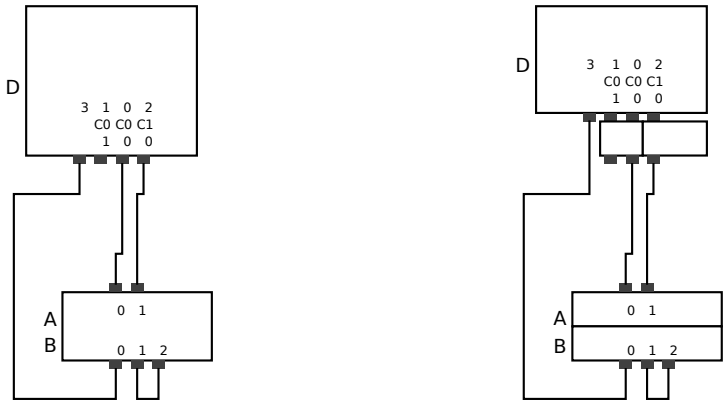
„Einfache“ Objekte wie Knoten und Pins (in KIELER als Ports), die im KIELER-Modell eine naheliegende Entsprechung haben, werden direkt übersetzt. Die Handhabung der „besonderen“ Objekte und Gegebenheiten wird im folgenden vorgestellt.

3.1 Zusammengesetzte Knoten

Über die Pin2Pins werden Pins von zwei verschiedenen Knoten eng miteinander verbunden und die zugehörigen Knoten damit vereinigt. So entstehen neue Vereinigungen von Knoten. Dabei ist nicht jede Art von Knotenvereinigung möglich. In einem gültigen Schaltplan gibt es zu jedem Pin höchstens ein Pin2Pin und die Pin2Pins sind so gesetzt, dass einer der folgenden Fälle zutrifft.

1. Ein Connector A ist mit einem Connector B vereinigt. Jedes Pin2Pin eines Pins von A muss mit einem Pin von B gebildet werden und umgekehrt. Nicht notwendigerweise alle Pins von A und B müssen in einem Pin2Pin enthalten sein. Gezeichnet werden die Knoten als sich berührend Rechtecke übereinander, wobei zusammengehörige Pins auf derselben senkrechten Geraden liegen.

2. Ein Device D wird mit n DeviceConnectors $C_1, \ldots, C_n$ vereinigt. Jedes Pin2Pin eines Pins von D muss mit einem DeviceConnector aus $C_1, \ldots, C_n$ gebildet sein und umgekehrt. Die DeviceConnectors $C_1, \ldots, C_n$ haben untereinander keine Pin2Pins. Nicht notwendigerweise alle Pins von D müssen in einem Pin2Pin enthalten sein. Wenn es zu einem Pin p_D von D ein Pin2Pin mit einem Pin p_C eines DeviceConnectors gibt, so dürfen Wires zu anderen Knoten nur an p_C angeschlossen sein. Gezeichnet wird der Device als großes Rechteck, an das entweder auf der



(a) Vor dem Postprocessing **(b)** Nach dem Postprocessing

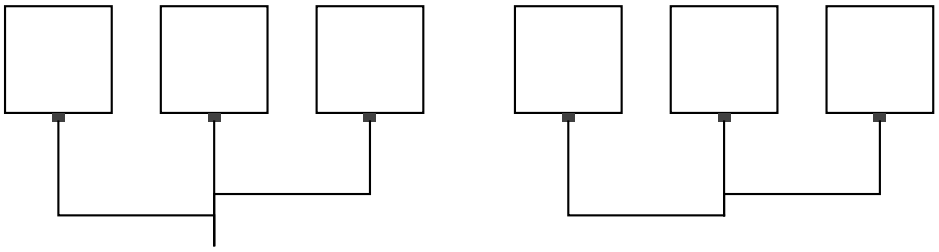
Abb. 1: Schaltplan mit zwei zusammengesetzen Knoten. Die obere Zusammensetzung besteht aus einem Device D mit 4 Pins. 3 von 4 Pins (die Pins mit Nummern 0 bis 2) von D sind via Pin2Pin mit einem Pin eines DeviceConnectors verbunden. Die zwei DeviceConnectors $C0$ und $C1$ sind an D angeschlossen.
Darunter befindet sich ein zusammengesetzter Knoten aus den Connectors A und B. Nur zwei Pins je Connector sind über Pin2Pin verbunden, B hat noch einen weiteren Pin. Für die Layoutingphase werden diese Vereingungen wie ein Knoten gehandhabt und erst im Postprocessing wieder in ihre Bestandteile zerlegt.

Ober- oder der Unterseite (das ist vorgegeben) die Pins und die DeviceConnectors gezeichnet werden. Zusammengehörige Pins liegen auch hier auf derselben senkrechten Gerade.

Da KLay Layered solche zusammengesetzten Knoten nicht unterstützt, erstelle ich stattdessen in der Übersetzungsphase einen Stellvertreterknoten, der von der Größe her die Fläche der Knoten, die er vertritt, abdeckt. Im Postprocessing wird er wieder durch seine Bestandteile ersetzt. Der Stellvertreterknoten enthält als Ports genau die Pins der Bestandteile, die Wires zu anderen Knoten enthalten können. Das sind alle Pins außer die Pins eines Devices, die mit einem Pin eines DeviceConnectors verbunden sind. Dadurch, dass zusammengehörige Pins in der resultierenden Zeichnung auf derselben senkrechten Geraden gezeichnet werden müssen, ist die Anordnung der Pins nicht frei. Insbesondere ist jeder relevante Pin fest entweder an der Ober- oder der Unterseite des Stellvertreterknotens. Ein Beispiel für zusammengesetzte Knoten ist in Abb. 1 zu sehen.

3.2 Wires mit mehr als zwei Endpunkten (Hyperkanten)

In den zu zeichnenden Schaltplänen kann es Wires geben, die an mehr als zwei Pins angeschlossen sind. Das sind Kabel- oder Drahtnetze, die in der Zeichnung wie andere Wires als orthogonal verlaufende Linien gezeichnet werden sollen, in diesem Fall mit Abzweigungen. Kanten im KIELER-Modell müssen genau zwei Endpunkte haben. Indem für



(a) Vor dem Postprocessing **(b)** Nach dem Postprocessing

Abb. 2: Wire mit drei Endungen. Dies wird mit einem Dummyknoten gelöst. Der Dummyknoten befindet sich im linken Bild unten an dem Linienende, das scheinbar ins nichts läuft. Rechts derselbe Schaltplan mit nachbearbeitetem Wire.

jedes Paar von Endpunkten einer Hyperkante eine Kante gesetzt wird, kann KLay Layered in gewisser Weise auch Hyperkanten darstellen. Um hierbei allerdings unerwartete und unschöne Umsetzungen zu vermeiden, löse ich solche Hyperkanten mittels Dummyknoten auf. Ich erstelle einen Dummyknoten von Höhe 0 und Breite 0 und eine einfache Kante zwischen jedem Hyperkantenendpunkt und diesem Dummyknoten. Da der Dummyknoten selbst nicht sichtbar ist, muss nur in dem Fall, dass alle zugehörigen Kanten aus derselben Richtung diesen Knoten erreichen, der Verlauf nachbearbeitet werden. In dem Fall werden die Kantenlinien bis zur nächsten davorliegenden Kantenabzweigung verkürzt wie in Abb. 2 zu sehen ist.

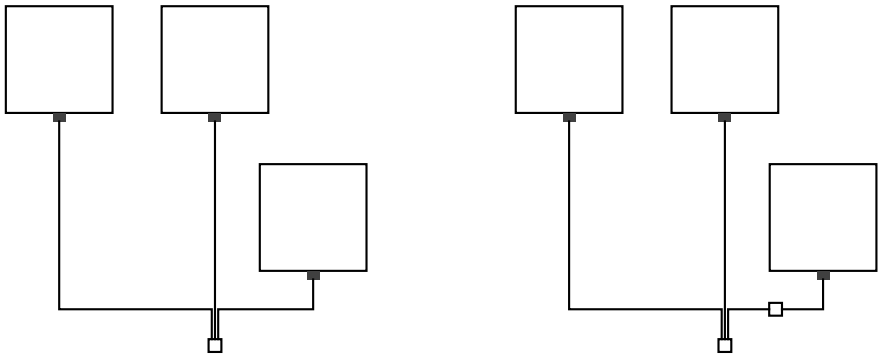
3.3 Einfügen der Wires

Fasst man den Schaltplan als Graphen auf, so entsprechen die Wires den Kanten. Nach Abschnitt 3.2 können alle Wires auf einfache Kanten mit zwei Endpunkten reduziert werden. Diese sollen in der Schaltplanzeichnung ohne Richtungsanzeige erscheinen. KLay Layered erwartet Kanten mit zwei Endpunkten, die jedoch eine Richtung haben müssen. Für jede ungerichtete Kante aus dem Schaltplan, kann KLay Layered eine Kante mit beliebiger Richtung übergeben werden, die nach der Layoutingphase wieder als ungerichtet angenommen werden kann.

Da KLay Layered über gerichtete Kanten definierte Hierarchien bzw. Abhängigkeiten deutlich macht, ist die Richtung für die zurückgegebene Zeichnung entscheidend. Statt die Richtung willkürlich oder zufällig zu setzen, füge ich die Kanten gemäß einer Breitensuche ein. Dabei werden vorgegebene Devices als Startknoten verwendet und die Kanten werden in der Richtung eingefügt, in der benachbarte Knoten in der Breitensuche erreicht werden.

3.4 Splices mit Knotengrad 2

Während Devices, DeviceConnectors und Connectors für komplexere Bauteile stehen, stehen Splices für räumlich kleinere und einfachere Verbindungspunkte von Wires, bei-



(a) Vor dem Postprocessing **(b)** Nach dem Postprocessing

Abb. 3: Schaltplan mit zwei Splices. Der Splice mit Knotengrad 3 wurde KLay Layered explizit als Knoten übergeben und liegt in einer der Ebenen von KLay Layered. Der Splice mit Knotengrad 2 wurde nicht übergeben, sondern im Nachhinein auf die längste Strecke seiner Kante gesetzt. Das linke Bild entspricht dem Schaltplan nach der Layoutingphase und vor der Nachbearbeitungsphase. Das rechte Bild dem Schaltplan nach der Nachbearbeitungsphase.

spielsweise Lötpunkte. Bei den Schaltplanzeichnungen wird dem Rechnung getragen, indem Splices als sehr kleine Rechtecke ohne Beschriftung gezeichnet werden. Sie können somit auch einfach im Nachhinein auf einem gezeichneten Wire positioniert werden, ohne dass benachbarte parallele Wires anders gezeichnet werden müssten. Bei Splices mit Knotengrad 2 setze ich das um, indem in der Übersetzungsphase ihre zwei inzidenten Kanten zu einer Kante verschmolzen werden und der Splice nicht KLay Layered mit übergeben wird. In der Nachbearbeitungsphase werden sie an einer geeigneten Stelle des Wireverlaufs wieder zugefügt. Bei Splices vom Grad größer als 2 ist das so nicht möglich, da ein Verschmelzen der inzidenten Kante eine Hyperkante entstehen lassen würde. Ein einfaches Beispiel ist in Abb. 3 zu finden. In einem Testdatensatz aus 57 Schaltplänen konnte die mittlere Anzahl der Kantenkreuzungen so gegenüber den Zeichnungen, für die auch für Splices mit Grad 2 ein expliziter Knoten in KLay Layered angelegt wurde, um 8,7 % und die mittlere Anzahl der Kantenknicke um 10,8 % gesenkt werden.

3.5 Multiwires

Multiwires sind keine eigenen Wires, sondern eine Zusammenfassung aus mehreren Wires oder Multiwires, die in der Praxis zu einem Kabel oder Kabelstrang zusammengefasst werden. Für die Zeichnung wird dabei nur gefordert, dass Pins dieser Wires am selben Knoten nebeneinander liegen, nicht aber, dass die Wires nebeneinander verlaufend gezeichnet werden. Die Beschränkung der Pins wird erreicht, indem die Reihenfolge der Ports, die diesen Pins entsprechen, manuell gesetzt wird und so festgehalten wird. Das geschieht, indem die Wires der Reihe nach durchgegangen werden. Wird hierbei ein Wire entdeckt, das Teil eines Multiwires ist, werden die anderen Wires dieses Multiwires

gesucht und deren Ports als nächstes neben den Port dieses Wires platziert. Das ist eine vergleichsweise einfache Umsetzung, die bei verschachtelten Multiwires (Multiwires enthalten Multiwires oder einzelne Wires sind Teil von mehreren Multiwires oder Wires verschiedener Multiwires enden am selben Pin) zu Problemen führen kann. Da in den gegebenen Schaltplänen allerdings nur wenige Multiwires und kaum verschachtelte Multiwires vorhanden waren, wurde diese naive Umsetzung gewählt, die die Anforderungen in den gegebenen Plänen erfüllen konnte.

3.6 Mehrere Wires an einem Pin

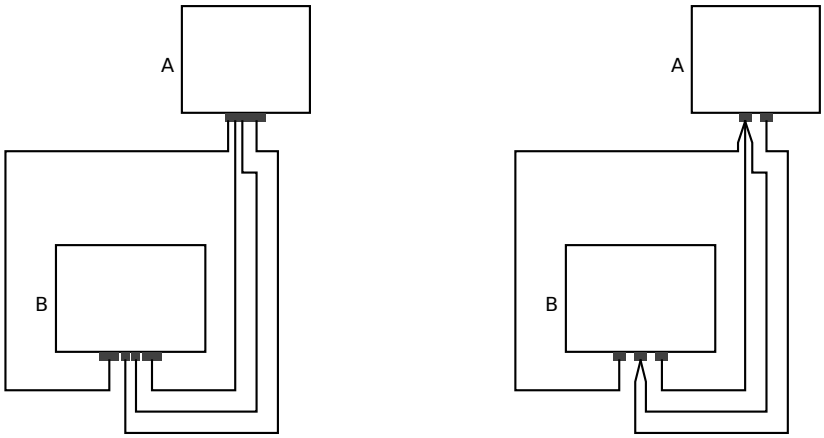
Wenn mehrere Wires an einem Pin eingehen, so sollen diese in gefächerter Form an den Pin-Eingängen eingehen. Um Überlagerungen von Kantenverläufen zu vermeiden, erstelle ich für den betroffenen Pin statt eines Ports mehrere schmalere Ports, die jeweils einen Endpunkt für genau ein Wire bilden. Wenn in meiner Umsetzung die reguläre Portbreite für einen Pin b ist und n Wires an einem Pin eingehen, hat jeder dieser schmaleren Ports eine Breite von b/n. Für diese mehreren Ports, die einem Pin entsprechen, muss für die Layoutingphase festgelegt werden, dass sie in jedem Fall nebeneinander gezeichnet werden. Daher wird für den betroffenen Knoten als PortConstraint „FIXED_ORDER“ gesetzt. Im Postprocessing werden diese schmaleren Ports zu einem Port von Breite b zusammengefasst und die parallel eingehenden Enden der Wires werden ggf. diagonal auf den Mittelpunkt dieses Ports umgelegt wie in Abb. 4 zu sehen ist.

3.7 Drehung von Connector-Connector-Vereinigungen

Zusammengesetzte Knoten, die aus zwei Connectors bestehen, sind so gezeichnet, dass sich der eine Connector oben mit nach oben zeigenden Pins und der andere unten mit nach unten zeigenden Pins befindet. Oft können Kreuzungen und Kantenknicke gespart werden, wenn die Zuordnung der Connectors als oberer oder unterer Teil sinnvoll gewählt ist. Da in KLay Layered die gerichteten Kanten vorzugsweise in einer festgelegten Richtung gezeichnet werden (hier: von oben nach unten), ist es sinnvoll einen Connector, der Teil einer ConnectorConnectorUnion ist, mit vielen eingehenden Kanten oben zu platzieren und einen mit vielen ausgehenden Kanten unten zu platzieren, um Kantenkreuzungen und Kantenknicke zu vermeiden. Ich addiere für jeden der zwei Connectors eingehende Kanten mit positivem Vorzeichen und ausgehende Kanten mit negativem Vorzeichen auf. Der Connector mit dem größeren Wert wird als oberer Teil der ConnectorConnectorUnion genommen. In einem Testdatensatz aus 57 Schaltplänen konnte die mittlere Anzahl der Kantenkreuzungen so gegenüber einer zufälligen Ausrichtung um 10,9 % und die mittlere Anzahl der Kantenknicke um 4,2 % gesenkt werden.

4 Ergebnis und Ausblick

Mit den vorgestellten Methoden können Schaltpläne eines speziellen Eingabeformats automatisch gezeichnet werden. Das eigentliche Layouting wird dabei von dem vorhandenen Programm KLay Layered aus KIELER übernommen, mit dem ich mich im Rah-



(a) Vor dem Postprocessing **(b)** Nach dem Postprocessing

Abb. 4: Schaltplan mit zwei Knoten A und B. Knoten A hat zwei Pins, bei denen an einem Pin drei, am anderen Pin ein Wire angeschlossen ist. Knoten B hat drei Pins, bei denen an einem Pin zwei an den anderen beiden Pins ein Wire angeschlossen ist. Wie im linken Bild zu erkennen ist, werden Pins mit mehreren Wires in kleinere Teile (ein Port je eingehendem Wire) aufgeteilt. Diese werden, wie im rechten Bild zu sehen ist, im Postprocessing wieder zusammengefügt und die Wireverläufe jeweils an der Mitte dieses Gesamtports, der einem Pin entspricht, zusammengeführt. Aus ästhetischen Gründen werden die Ports außerdem im Postprocessing etwas schmaler gezeichnet.

men des Praktikums befasst habe und bei dem ich einige der zur Verfügung gestellten Einstellungsmöglichkeiten genutzt habe. Vom eigenen Programm, das in Kapitel 3 vorgestellt wurde, werden Schaltpläne wie der in Abb. 5 ausgegeben. Die größten Pläne des Testdatensatzes mit einer niedrigen dreistelligen Zahl an Knoten werden an meinem handelsüblichen Notebook in der Größenordnung weniger Zehntelsekunden gezeichnet.

Ein offener Punkt ist eine vorteilhaftere Anordnung der Ports an den Knoten, an denen die Reihenfolge der Ports nicht völlig frei ist, um Kantenkreuzungen zu vermeiden. Wo es geht, wähle ich „FREE“ oder „FIXED_SIDE“, da dies die einzigen Einstellungen sind, in denen KLay Layered die Reihenfolge der Ports selbst so setzen kann, um gemäß der Heuristiken hierarchischer Algorithmen wenige Kantenkreuzungen zu erzeugen. Die verwendete Heuristik, die die vorhandene Heuristik um Ports erweitern, wird im Artikel von Schulze et al. [SSvH14] beschrieben. Bei diesem System von PortConstraints muss ich jedoch, sobald ich nur eine Einschränkung der Reihenfolge der Ports habe, die gesamte Reihenfolge festsetzen, also „FIXED_ORDER“ wählen. In mehreren Fällen wäre bei dieser Anwendung eine Gruppierung von Ports sinnvoll. Mit Gruppierung meine ich hier, dass die Ports innerhalb ihrer Gruppe frei vertauschbar sind und die Gruppen untereinander vertauscht werden können. Gruppierungen sollten rekursiv möglich sein (Innerhalb einer Gruppierung weitere Gruppierungen). Derzeit wird in folgenden Fällen auf „FIXED_ORDER“ gesetzt:

- Wenn mehr als ein DeviceConnector via Pin2Pin an einem Device angeschlossen wird oder neben einem DeviceConnector Pins des Devices ohne DeviceConnector bleiben.
- Wenn an einem Pin mehr als ein Wire angeschlossen ist, da dann schmalere Ports in Vertretung für einen Pin erstellt werden, also Ports, die nebeneinander verbleiben müssen.
- Wenn mehrere Wires ein Multiwire bilden.
- Wenn zwei Connectors via Pin2Pin verbunden sind. (Dieser Fall unterscheidet sich, da die gegenüberliegenden Pins gegenüber bleiben sollen, aber untereinander vertauscht werden können)

Eine denkbare Verbesserung wäre die Theorie zur Portanordnung und den zugehörigen Code von KLay Layered zu erweitern. Ein anderer einfacherer Ansatz wäre KLay Layered zwei mal aufzurufen. Zuerst mit freier Anordnung der Ports und dann mit fester Reihenfolge, die sich unter Einhaltung der gegebenen Constraints an der Reihenfolge orientiert, die bei der ersten Ausführung zurück gegeben wurde. Alternativ auch mehrfaches Aufrufen, das immer mit fester Reihenfolge geschieht. Anschließend wird lokal für jeden Knoten in einer Analyse die „bestmögliche“ Reihenfolge ermittelt und im nächsten Durchlauf als feste Reihenfolge verwendet.

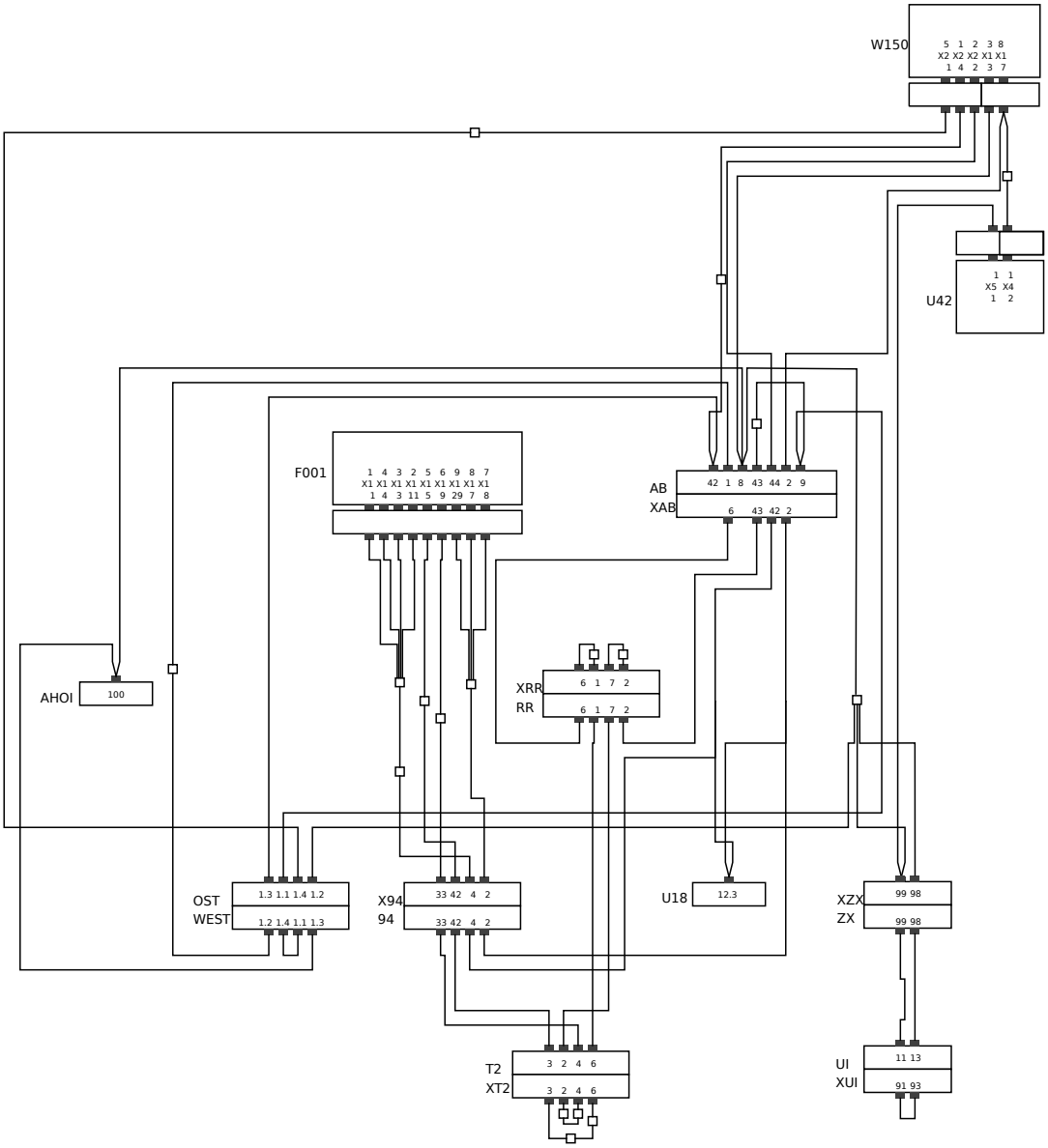


Abb. 5: Beispiel für einen ganzen Schaltplan. Der Schaltplan ist den realen Schaltplänen, von denen mir ein Teil als Testdatensatz zur Verfügung steht, nachempfunden. Benennungen von Knoten und Pins sind von mir frei und willkürlich gewählt worden.

Literatur

[KIE] Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER). https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Overview.

[KLa] KLay Layout Algorithmen (Teil von KIELER). https://rtsys.informatik.uni-kiel.de/confluence/pages/viewpage.action?pageId=328080.

[SSvH14] Christoph Daniel Schulze, Miro Spönemann und Reinhard von Hanxleden: Drawing layered graphs with port constraints. *J. Vis. Lang. Comput.*, 25(2):89–106, 2014.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.